Laboratorium 1

- **Interpreter psql**
- **Operacje administracyjne**
- **Podstawy**

Wymagany materiał z wykładu:

- **Rozdział 1 – SQL – typy danych i podstawowe rozkazy**

Interpreter psql

Podstawowa składnia polecenia psql:

```
psql [-f nazwa_pliku] [nazwa_bazy] [nazwa_użytkownika]
```

Elementy w nawiasach [] są opcjonalne. Jeśli *nazwa_użytkownika* lub *nazwa_bazy* nie zostaną podane to psql użyje w tych miejscach nazwy użytkownika z UNIX'a.

Zatem jeśli nasze konto nazywa się *aaa* to polecenia:

```
psql
psql aaa
psql aaa aaa
```

są równoważne. Opcja `-f nazwa_pliku` pozwala na wykonanie komend SQL zapisanych w podanym pliku.

Po uruchomieniu interpreter pokaże znak zachęty:

```
aaa=>
```

Oznacza to, że psql jest gotowy do przyjęcia nowego polecenia psql albo komendy SQL.

Polecenia psql zaczynają się od znaku `\` . Przykładowe polecenia psql:
`\q` – wyjście z psql
`\r` – skasowanie bieżącego polecenia SQL
`\?` – wykaz poleceń psql z opisami
`\d` – wyświetlenie listy tabel

Komendy SQL mogą być wpisywane w wielu liniach i muszą się kończyć znakiem `;` (średnik)
Wpisując polecenia SQL warto zauważyć zmiany znaku zachęty:

`=>` oznacza, że psql jest gotowy na przyjęcie nowego polecenia SQL albo psql
`->` oznacza, że w tej linii kontynuujemy polecenie SQL rozpoczęte w poprzedniej linii (`\r` powróci do `=>`)
`'>` oznacza, że w poprzedniej linii otwarto i nie zamknięto apostrofu - trzeba go zamknąć
`">` oznacza, że w poprzedniej linii otwarto i nie zamknięto cudzysłowia - trzeba go zamknąć
`(>` oznacza, że w poprzedniej linii otwarto i nie zamknięto nawiasu - trzeba go zamknąć

Operacje administracyjne

Tworzenie bazy danych (z linii poleceń UNIX'a):
```
createdb [nazwa_bazy]
```
Nazwa bazy w nawiasach [] jest opcjonalna, domyślnie przyjmowana jest nazwa taka jak nazwa użytkownika.

Usuwanie bazy danych (z linii poleceń UNIX'a):
```
dropdb nazwa_bazy
```

UWAGA!!!

Użytkownik może posiadać wiele baz danych, domyślną bazą jest baza o nazwie odpowiadającej loginowi użytkownika. **Nieużywane bazy należy USUWAĆ!!!**

Zrzut kopii bazy danych do pliku tekstowego(z linii poleceń UNIX'a):
```
pg_dump nazwa_bazy > nazwa_pliku
```

Zmiana hasła w bazie danych (z linii poleceń psql):
```
alter user nazwa_uzytkownika with password 'nowy_password';
```

Podstawy

Przykład 1: Utworzenie tabeli (relacji), wyświetlenie listy tabel, wyświetlenie opisu tabeli.
```
create table studenci (imie varchar(20), nazwisko varchar(20), indeks int4);
\d
\d studenci
```

Przykład 2: Wpisanie wierszy do tablicy.
```
insert into studenci values ('Adam', 'Malysz', 1);
insert into studenci values ('James', 'Bond', 2);
insert into studenci values ('Adam', 'Mickiewicz', 3);
```

Przykład 3: Wyświetlanie wierszy.
```
select * from studenci;
select nazwisko, imie from studenci;
```

Zadanie 1: Wyświetlić nazwisko i nr. indeksu.

Przykład 4: Wyświetlanie wierszy – c.d.
```
select * from studenci where imie='Adam';
```

Zadanie 2: Wyświetlić nazwisko i imię tych studentów, których indeks=2

Przykład 5: Aktualizacja danych.
```
update studenci set indeks=7 where nazwisko='Bond';
update studenci set imie='Adas' where imie='Adam';
```

Przykład 6: Modyfikacja tablicy.
```
alter table studenci add column tel varchar(20);
alter table studenci drop column indeks;
```

Zadanie 3: Dopisać nr. tel. dla podanych osób.

Zadanie 4: Dopisać kolumnę „wiek” typ: int4 i uzupełnić tak. aby wiek różnych osób był powyżej i poniżej 25 lat.

Zadanie 5: Wyświetlić osoby starsze niż 25 lat.

Zadanie 6: Wyświetlić nazwiska i imiona posortowane wg. nazwiska.

Zadanie 7: Wyświetlić nazwiska i imiona osób posortowane wg. wieku.

Przykład 7: Kasowanie wierszy.
```
delete from studenci where imie='James';
```

Zadanie 8: Wykonaj kopię zapasową bazy danych. Następnie usuń bazę danych. Utwórz ją na nowo i załaduj zawartość bazy danych z kopii zapasowej. Sprawdź zawartość odzyskanej bazy danych.

Zadanie 9: Skasować jednym poleceniem wszystkie wiersze w tablicy studenci.

Przykład 10: Usuwanie tablicy.
```
drop table studenci;
```

UWAGA: Przed wykonaniem poniższych przykładów należy załadować bazę danych do ćwiczeń (wg. opisu na ostatniej stronie materiałów)

Przykład 9: Wyświetlić listę podkategorii i nazwy kategorii do których należą.
```
SELECT W1.name, W2.name FROM subcategories W1, categories W2
WHERE W1.cid = W2.cid;
```

Przykład 10: Wyświetlić listę podkategorii i nazwy kategorii do których należą. Kolumna z podkategoriami powinna nazywać się "podkategorie" a kolumna z kategoriami "kategorie".
```
SELECT W1.name AS podkategorie, W2.name AS kategorie
FROM subcategories W1, categories W2
WHERE W1.cid = W2.cid;
```

Laboratorium 2

- **Operacje mnogościowe - unia, przekrój, różnica**
- **Produkt (iloczyn kartezjański)**
- **Połączenie**

Wymagany materiał z wykładu:

- **Rozdział 2 – Podstawowe zapytania – SELECT**
- **Rozdział 3 – Zagadnienia zaawansowane**

Operacje mnogościowe

unia (suma zbiorów)

Przykład 1: Wyświetlić w jednej kolumnie nazwy wszystkich kategorii i podkategorii.

```
SELECT name FROM categories UNION SELECT name FROM subcategories;

        name
--------------------
 Advertisement
 Amusement
 BTI
 Biology
 Cartoons
 Commercial
 Education
 Geography
 KT
 Mathematics
 Movies
 Others
 Physics
 Technology
 Telecommunications
 budowa
 wykłady
(17 rows)
```

Proszę zwrócić uwagę, że nazwa `Others` została wyświetlona tylko raz mimo, że występuje zarówno w tabeli `categories` jak i w tabeli `subcategories`. Można się o tym przekonać wykonując kolejno polecenia:

```
SELECT name FROM categories;
SELECT name FROM subcategories;
```

Podobnie, nazwa `Telecommunications` została wyświetlona tylko raz, mimo, że występuje dwa razy jako podkategoria.

przekrój (część wspólna zbiorów)

Przykład 2: Wyświetlić nazwy które występują zarówno w tabeli kategorii jak i podkategorii.

```
SELECT name FROM categories INTERSECT SELECT name FROM subcategories;

  name
--------
 Others
(1 row)
```

różnica

Przykład 3: Wyświetlić nazwy kategorii z wyjątkiem takich nazw, które występują również w tabeli podkategorii.

```
SELECT name FROM categories EXCEPT SELECT name FROM subcategories;

    name
------------
 Amusement
 Education
 Commercial
 KT
(4 rows)
```

Produkt (iloczyn kartezjański)

Przykład 4: Wyświetlić produkt powstały ze skrzyżowania tabel: categories i subcategories

```
SELECT * FROM categories, subcategories;
```

Takie zapytanie spowoduje skrzyżowanie wierszy z obu tabel na zasadzie „każdy z każdym”.

Połączenie

Połączenie jest podzbiorem produktu. Otrzymuje się je przed podanie odpowiedniego warunku łączenia.

Przykład 5: Wyświetlić listę podkategorii i nazwy kategorii do których należą.

```
SELECT subcategories.name, categories.name FROM categories, subcategories
WHERE categories.cid = subcategories.cid;
```

```
        name        |    name
--------------------+------------
 Advertisement      | Amusement
 Cartoons           | Amusement
 Movies             | Amusement
 Biology            | Education
 Physics            | Education
 Technology         | Education
 Telecommunications | Education
 Mathematics        | Education
 Geography          | Education
 Others             | Others
 Telecommunications | Commercial
 BTI                | KT
 wykłady            | KT
 budowa             | KT
(14 rows)
```

Aliasy tabel

Aby uniknąć konieczności przepisywania w zapytaniu długich nazw tabel (`categories` i `subcategories`) można stosować aliasy. Poniższe zapytanie jest równoważne przedstawionemu wcześniej.

Przykład 6:

```
SELECT W1.name, W2.name FROM categories W2, subcategories W1 WHERE W2.cid = W1.cid;
```

Nadanie nazw kolumnom

W powyższym przykładzie obie kolumny zwrócone jako wynik zapytania mają takie same nazwy (`name`). Każdej kolumnie można jednak nadać nową nazwę. Ilustruje to poniższy przykład.

Przykład 7: Wyświetlić listę podkategorii i nazwy kategorii do których należą. Kolumna z podkategoriami powinna nazywać się "podkategorie", a kolumna z kategoriami "kategorie".

```
SELECT W1.name AS podkategorie, W2.name AS kategorie FROM subcategories W1, categories W2
WHERE W1.cid = W2.cid;
```

Zadania

Zadanie 1: Wyświetlić identyfikatory filmów w których w tytule filmu lub w nazwie autora występuje słowo Techno (`... WHERE xxxx ~ 'Techno' ...` gdzie `xxxx` jest nazwą kolumny w której szukamy tego słowa).

```
 mid
-----
 120
 140
 141
(3 rows)
```

Zadanie 2: Jak wyżej, ale wypisać także autora i tytuł każdego znalezionego filmu.

```
 mid |       author        |        title
-----+---------------------+---------------------
 120 |                     | Technology LSA-PLUS
 140 | Lucent Technologies | Lucent 1
 141 | Lucent Technologies | Lucent 2
(3 rows)
```

Zadanie 3: Załadować ze zrzutu `~rstankie/db/stud2.dump` tabelę episodes_list2. Tabela episodes_list2 zawiera dane z tabeli episodes_list z wprowadzonymi drobnymi zmianami. Ustalić, które wiersze zostały zmienione/usunięte/dodane.

Zadanie 4: Wyświetlić tytuły filmów (is_movie = 1) z tabeli episodes_list.
```
(44 rows)
```

Zadanie 5: Wyświetlić tytuł filmu, ID kategorii i ID podkategorii, do której należy dany film.
```
(44 rows)
```

Zadanie 6: Wyświetlić tytuł filmu, NAZWĘ kategorii i NAZWĘ podkategorii, do której należy dany film.
```
(43 rows)
```
Zastanów się dlaczego zapytanie zwraca o jeden wiersz mniej niż w zadaniu 5. Znajdź brakujący wiersz i wyjaśnij.

Zadanie 7: Wyświetlić numery i tytuły filmów należących do kategorii 63.

```
    mid   |            title
----------+-----------------------------
       70 | Intranet (advertisement)
       69 | OVS Sample 1
       63 | Fore Systems (advertisement)
      109 | OVS Introduction
       72 | OVS Sample 2
      140 | Lucent 1
      141 | Lucent 2
(7 rows)
```

Zadanie 8: Wyświetlić numery i tytuły filmów należących do kategorii "Education" posortowane po tytułach.

```
 mid |               title
-----+-----------------------------------
 106 | Course in a box
  98 | Image coding (part 1)
 100 | Image coding (part 4)
 103 | Microcosmos (high quality)
 121 | Microcosmos (low quality)
 122 | Microcosmos (medium quality)
 119 | Overhead telecommunications lines
  77 | Revolution to orbit
 111 | Saga of life (high quality)
 123 | Saga of life (low quality)
 124 | Saga of life (medium quality)
 110 | Southern Africa Safari
 120 | Technology LSA-PLUS
 112 | The World's Deadliest Volcanoes
 101 | Video coding
(15 rows)
```

Zadanie 9: Wyświetlić listę epizodów (jako episode) oraz tytuły filmów, których te epizody są fragmentami (jako movie). Posortować według tytułu filmu (rosnąco) oraz wg czasu początku epizodu (malejąco).

```
                   episode                    |                  movie
----------------------------------------------+---------------------------------------
 fragment                                     | Bolek & Lolek - Race to the North Pole
 Snails' sexual act                           | Microcosmos (high quality)
 The ladybird                                 | Microcosmos (high quality)
 The bee                                      | Microcosmos (high quality)
 The first morning                            | Microcosmos (high quality)
 Landing                                      | Microcosmos (high quality)
 prof. Jajszczyk                              | Prezentacja Katedry Telekomunikacji AGH
 Galunggung, Tambora, Toba (Indonesia)        | The World's Deadliest Volcanoes
 Krakatau (Indonesia)                         | The World's Deadliest Volcanoes
 Mount Rainier (Washington, USA)              | The World's Deadliest Volcanoes
 Mount St. Helens (Washington, USA)           | The World's Deadliest Volcanoes
 Volcanoes under glacier Vatnajökull (Iceland)| The World's Deadliest Volcanoes
 Eldfell (Heimaey, Iceland)                   | The World's Deadliest Volcanoes
 Soufriere Hills (Montserrat, West Indies)    | The World's Deadliest Volcanoes
 Popocatepetl (Mexico)                        | The World's Deadliest Volcanoes
 Nevado del Ruiz (Colombia)                   | The World's Deadliest Volcanoes
 Pacaya (Guatemala)                           | The World's Deadliest Volcanoes
 Kilauea (Hawaii, USA)                        | The World's Deadliest Volcanoes
(18 rows)
```

Zadanie 10: Jak wyżej tylko dodać jeszcze kolumnę 'start' z czasem początku epizodu w formacie: HH:MM:SS. W tabeli czas podany jest w milisekundach. Zalecane użycie funkcji reltime(int4).

```
  start   |                    episode                    |                  movie
----------+-----------------------------------------------+----------------------------------------
 00:02:21 | fragment                                      | Bolek & Lolek - Race to the North Pole
 00:14:56 | Snails' sexual act                            | Microcosmos (high quality)
 00:10:57 | The ladybird                                  | Microcosmos (high quality)
 00:09:10 | The bee                                       | Microcosmos (high quality)
 00:03:44 | The first morning                             | Microcosmos (high quality)
 00:01:04 | Landing                                       | Microcosmos (high quality)
 00:01:31 | prof. Jajszczyk                               | Prezentacja Katedry Telekomunikacji AGH
 00:40:58 | Galunggung, Tambora, Toba (Indonesia)         | The World's Deadliest Volcanoes
 00:40:18 | Krakatau (Indonesia)                          | The World's Deadliest Volcanoes
 00:35:05 | Mount Rainier (Washington, USA)               | The World's Deadliest Volcanoes
 00:30:54 | Mount St. Helens (Washington, USA)            | The World's Deadliest Volcanoes
 00:27:40 | Volcanoes under glacier Vatnajökull (Iceland) | The World's Deadliest Volcanoes
 00:22:46 | Eldfell (Heimaey, Iceland)                    | The World's Deadliest Volcanoes
 00:16:42 | Soufriere Hills (Montserrat, West Indies)     | The World's Deadliest Volcanoes
 00:15:30 | Popocatepetl (Mexico)                         | The World's Deadliest Volcanoes
 00:14:08 | Nevado del Ruiz (Colombia)                    | The World's Deadliest Volcanoes
 00:13:23 | Pacaya (Guatemala)                            | The World's Deadliest Volcanoes
 00:01:44 | Kilauea (Hawaii, USA)                         | The World's Deadliest Volcanoes
(18 rows)
```

Zadanie 11: Jak wyżej, ale wyświetlić dodatkowo nazwę kategorii i podkategorii.

```
  start   |                  episode                  |                  movie                  | category  | subcategory
----------+-------------------------------------------+-----------------------------------------+-----------+-------------
 00:02:21 | fragment                                  | Bolek & Lolek - Race to the North Pole  | Amusement | Cartoons
 00:14:56 | Snails' sexual act                        | Microcosmos (high quality)              | Education | Biology
 00:10:57 | The ladybird                              | Microcosmos (high quality)              | Education | Biology
 00:09:10 | The bee                                   | Microcosmos (high quality)              | Education | Biology
 00:03:44 | The first morning                         | Microcosmos (high quality)              | Education | Biology
 00:01:04 | Landing                                   | Microcosmos (high quality)              | Education | Biology
 00:01:31 | prof. Jajszczyk                           | Prezentacja Katedry Telekomunikacji AGH | KT        | wykłady
 00:40:58 | Galunggung, Tambora, Toba (Indonesia)     | The World's Deadliest Volcanoes         | Education | Geography
 00:40:18 | Krakatau (Indonesia)                      | The World's Deadliest Volcanoes         | Education | Geography
 00:35:05 | Mount Rainier (Washington, USA)           | The World's Deadliest Volcanoes         | Education | Geography
 00:30:54 | Mount St. Helens (Washington, USA)        | The World's Deadliest Volcanoes         | Education | Geography
 00:27:40 | Volcanoes under glacier Vatnajökull (Iceland)| The World's Deadliest Volcanoes      | Education | Geography
 00:22:46 | Eldfell (Heimaey, Iceland)                | The World's Deadliest Volcanoes         | Education | Geography
 00:16:42 | Soufriere Hills (Montserrat, West Indies) | The World's Deadliest Volcanoes         | Education | Geography
 00:15:30 | Popocatepetl (Mexico)                     | The World's Deadliest Volcanoes         | Education | Geography
 00:14:08 | Nevado del Ruiz (Colombia)                | The World's Deadliest Volcanoes         | Education | Geography
 00:13:23 | Pacaya (Guatemala)                        | The World's Deadliest Volcanoes         | Education | Geography
 00:01:44 | Kilauea (Hawaii, USA)                     | The World's Deadliest Volcanoes         | Education | Geography
(18 rows)
```

Zadanie 12: Wyświetlić tytuły epizodów, których czas trwania jest większy od 100000. Wyświetlić również czasy trwania epizodów i posortować wg czasu trwania.

```
                episode_title                | episode_length
---------------------------------------------+----------------
 Volcanoes under glacier Vatnajökull (Iceland) |         109100
 Snails' sexual act                          |         114490
 The ladybird                                |         139000
 Mount Rainier (Washington, USA)             |         172210
 Mount St. Helens (Washington, USA)          |         245790
 Eldfell (Heimaey, Iceland)                  |         279300
 The first morning                           |         318690
 Soufriere Hills (Montserrat, West Indies)   |         349090
 Kilauea (Hawaii, USA)                       |         660330
(9 rows)
```

Laboratorium 3

- **Klauzule GROUP BY, HAVING oraz funkcje agregujące**

Wymagany materiał z wykładu:

- **Rozdział 4 – Grupowanie**
- **Rozdział 5 – NULL**

Klauzule GROUP BY, HAVING oraz funkcje agregujące

Przykład 1: Wyświetlić sumaryczny czas trwania wszystkich filmów.

```
select sum(lenmsec) from movies_list;

   sum
----------
 77924437
(1 row)
```

Przykład 2: Wyświetlić identyfikator kategorii i minimalny czas trwania filmów należących do poszczególnych kategorii.

```
select cid, min(lenmsec) from movies_list group by cid;

 cid |   min
-----+---------
  60 |  155532
  61 |   49901
  63 |    9932
  64 |   30159
     | 5436129
(5 rows)
```

Przykład 3: Wyświetlić identyfikator kategorii i minimalny czas trwania filmów należących do poszczególnych kategorii. Wyświetlić tylko te kategorie, dla których minimalny czas trwania filmu jest większy od 35000 ms.

```
select cid, min(lenmsec) from movies_list group by cid having min(lenmsec)>35000;

 cid |  min
-----+--------
  60 | 155532
  61 |  49901
     | 5436129
(2 rows)
```

Przykład 4: Policzyć ile jest wierszy w tabeli movies_list.

```
select count(*) from movies_list;

 count
-------
    44
(1 row)
```

Przykład 5: Policzyć ile jest filmów w poszczególnych kategoriach (dwa sposoby: w pierwszym nie są zliczane te wiersze, w których pole `cid` zawiera wartość NULL; drugi sposób zlicza wszystkie wiersze).

```
select cid, count(cid) from movies_list group by cid;

 cid | count
-----+-------
  60 |    13
  61 |    15
  63 |     7
  64 |     8
     |     0
(5 rows)

select cid, count(*) from movies_list group by cid;

 cid | count
-----+-------
  60 |    13
  61 |    15
  63 |     7
  64 |     8
     |     1
(5 rows)
```

Przykład 6: Policzyć ile spośród wszystkich zdefiniowanych filmów jest przypisanych do jakiejś kategorii.

```
select count(mid) from movies_list where cid is not null;

 count
-------
    43
(1 row)
```

Przykład 7: Policzyć ile spośród wszystkich zdefiniowanych filmów jest przypisanych do jakiejś kategorii nie używając żadnych warunków (where).

```
select count(cid) from movies_list;

 count
-------
    43
(1 row)
```

Przykład 8: Policzyć do ilu różnych kategorii są przypisane filmy.

```
select count(DISTINCT cid) from movies_list;

 count
-------
     4
(1 row)
```

Zadania:

Zadanie 1: Wyświetlić identyfikator kategorii i sumę czasów trwania filmów należących do poszczególnych kategorii.

```
 cid |   sum
-----+----------
  60 | 11882469
  61 | 56491621
  63 |   804406
  64 |  3309812
     |  5436129
(5 rows)
```

Zadanie 2: Wyświetlić identyfikator kategorii i sumę czasów trwania filmów należących do poszczególnych kategorii. Do obliczania sumy brać tylko filmy, których wartość parametru *stream* > 1 500 000.

```
 cid |   sum
-----+---------
  60 |  155532
  61 | 5709429
  63 |  772848
  64 | 2283686
     | 5436129
(5 rows)
```

Zadanie 3: Wyświetlić identyfikator kategorii i sumę czasów trwania filmów należących do poszczególnych kategorii. Do obliczania sumy brać tylko filmy, których wartość parametru *stream* > 1 500 000. Wyświetlić tylko kategorie, dla których suma czasów trwania filmu jest większa od 900 000 ms.

```
 cid |   sum
-----+---------
  61 | 5709429
  64 | 2283686
     | 5436129
(3 rows)
```

Zadanie 4. Policz dla ilu epizodów (nie filmów) wprowadzono opis jako tekst (descrtype=1).

```
liczba opisow
--------------
            7
(1 row)
```

Zadanie 5. Wyświetl nazwy kategorii, które mają przynajmniej 2 subkategorie. Jako drugą kolumnę wyświetl ile subkategorii należy do danej kategorii.

```
 Nazwa kategorii | count
-----------------+-------
 KT              |     3
 Education       |     6
 Amusement       |     3
(3 rows)
```

Zadanie 6. Wypisz nazwy subkategorii, dla których maksymalny czas trwania epizodów należących do danej subkategorii jest dłuższy od 10 sekund. Dodatkowo wypisz czas trwania najdłuższego epizodu i liczbę epizodów spełniających warunek dla danej subkategorii.

```
 Subkategoria |  max   | count
--------------+--------+-------
 Biology      | 318690 |     5
 Geography    | 660330 |    11
 Cartoons     |  11480 |     1
(3 rows)
```

Zadanie 7. Wypisz nazwy subkategorii, dla których maksymalna długość epizodu należącego do danej subkategorii jest mniejsza od 350 sekund. Jako drugą kolumnę wypisz długość najdłuższego epizodu, a jako trzecią kolumnę liczbę epizodów należących do danej subkategorii. Posortuj wyniki według nazwy subkategorii.

```
 Nazwa podkategorii |  max   | count
--------------------+--------+-------
 Biology            | 318690 |     5
 Cartoons           |  11480 |     1
 wykŁady            |   6970 |     1
 (3 rows)
```

Zadanie 8. Policz ile filmów ma różne słowa kluczowe.

```
 count
-------
    13
(1 row)
```

Zadanie 9. Policz ile filmów ma zdefiniowany odnośnik URL (parametr descrhtml).

```
count
-------
    20
(1 row)
```

Zadanie 10. Policz ile filmów nie ma zdefiniowanego odnośnika URL (parametr descrhtml).

```
count
-------
    24
(1 row)
```

Zadanie 11. Policz ile jest zdefiniowanych różnych odnośników URL dla filmów.

```
count
-------
    10
(1 row)
```

Zadanie 12. Wypisz nazwy filmów i odnośniki URL posortowane według nazwy filmu. Jeśli odnośnik URL nie jest zdefiniowany powinien pojawić się napis "URL not defined" (użyj COALESCE).

```
                          title                          |                    url
---------------------------------------------------------+-----------------------------------
 Bolek & Lolek - Australian steppes                      | ../html_descriptions/bolek_lolek
 Bolek & Lolek - Gold town                               | ../html_descriptions/bolek_lolek
 Bolek & Lolek - On tiger's trails                       | ../html_descriptions/bolek_lolek
 Bolek & Lolek - Pharaoh's tomb                          | ../html_descriptions/bolek_lolek
 Bolek & Lolek - Race to the North Pole                  | ../html_descriptions/bolek_lolek
 Bolek & Lolek - Smuggler                                | ../html_descriptions/bolek_lolek
 Course in a box                                         | URL not defined
 Fore Systems (advertisement)                            | URL not defined
 Gustavus and the fly                                    | URL not defined
 ...
 ...
(44 rows)
```

Laboratorium 4

- **Podzapytania**
- **Widoki**

Wymagany materiał z wykładu:

- **Rozdział 7 – Widoki**
- **Rozdział 8 – Zagnieżdżanie zapytań**

Podzapytania

Wynik jednego zapytania może być użyty np. jako warunek innego zapytania

Podzapytania proste

Podzapytanie proste jest wykonywane przed wykonaniem zapytania głównego.

Przykład 1: Wyświetlić identyfikatory filmów i czasy trwania filmów, których czas trwania jest większy od średniego czasu trwania filmu (posortowane po mid).

```
select mid, lenmsec from movies_list where lenmsec > (select avg(lenmsec) from movies_list)
order by mid;
```

```
 mid | lenmsec
-----+---------
  80 | 5436129
  98 | 5501871
 100 | 6965989
 101 | 5385605
 103 | 4379470
 110 | 3571616
 111 | 6422197
 112 | 2741565
 113 | 6876310
 119 | 1897170
 121 | 4554451
 122 | 4534809
 123 | 3289977
 124 | 6363459
(14 rows)
```

Podzapytaniem jest tutaj: `select avg(lenmsec) from movies_list`. Podzapytanie to zwraca średni czas filmu (1771009.9318181818).

Przykład 2: Wyświetlić nazwy kategorii, do których przypisano jakieś filmy. Wymagane użycie podzapytania. Nie wolno natomiast krzyżować tabel.

```
select name from categories where cid = any (select cid from movies_list group by cid);
```

```
    name
------------
 Amusement
 Education
 Commercial
 KT
(4 rows)
```

Ten sam rezultat można by osiągnąć przez zapytania z iloczynem kartezjańskim. Zapytania z iloczynem kartezjańskim dające ten sam rezultat:

```
select W1.name from categories W1, movies_list W2 where W1.cid = W2.cid group by W1.name;
```

```
select distinct on (W1.name) W1.name from categories W1, movies_list W2 where W1.cid = W2.cid;
```

Należy jednak pamiętać, że nie w każdej sytuacji wyniki będą identyczne (zależnie od zapytania i zawartości bazy). Będzie to widać w przykładzie 6.

Przykład 3: Wyświetlić nazwy kategorii, do których nie przypisano żadnych filmów.

```
select name from categories where cid <> all (select cid from
movies_list where cid is not null group by cid);
```

```
  name
--------
 Others
(1 row)
```

Przykład 4: Wynik podzapytania może być także użyty w charakterze tabeli wejściowej dla zapytania. Użycie aliasu dla nazwy tabeli jest w tym przypadku obowiązkowe. Ilustruje to następujący przykład:

Wyświetlić statystykę długości nazw kategorii i podkategorii.

```
select char_length(name) as "name length", count(*) from (select name from categories union
select name from subcategories) as names group by "name length" order by "name length";
 name length | count
-------------+-------
           2 |     1
           3 |     1
           6 |     3
           7 |     3
           8 |     1
           9 |     3
          10 |     2
          11 |     1
          13 |     1
          18 |     1
(10 rows)
```

Interpretacja wyników jest następująca: w bazie jest jedna nazwa kategorii lub podkategorii o długości dwóch znaków, 3 nazwy o długości 6 znaków itd.

Podzapytania złożone

Podzapytanie złożone wykonywane jest dla każdej krotki zapytania głównego.

Przykład 5: Wyświetlić nazwy kategorii, dla których średni czas trwania filmów (należących do danej kategorii) jest większy niż 120000 ms.

```
select name from categories where (select avg(lenmsec) from movies_list where
movies_list.cid=categories.cid)>120000;
   name
-----------
 Amusement
 Education
 KT
(3 rows)
```

Przykład 6: Wyświetlić identyfikator filmu i nazwę kategorii, do której należy dany film

```
select m.mid, (select name from categories c where c.cid = m.cid) from movies_list m;

(44 rows)
```

zauważyć różnicę w stosunku do zapytania:

```
SELECT m.mid, c.name FROM movies_list m, categories c where c.cid = m.cid;

(43 rows)
```

W wynikach zapytania z podzapytaniem pojawia się film o identyfikatorze 80, który nie został przypisany do żadnej kategorii. Zapytanie z iloczynem kartezjańskim go nie wyświetla. Ponieważ w tabeli `movies_list` w polu `cid` dla tego filmu jest wartość NULL, warunek `c.cid=m.cid` usuwa z wyników iloczynu kartezjańskiego ten wiersz. Porównanie jakiejkolwiek wartości z wartością NULL zwraca fałsz.
Ponadto należy zauważyć, że pierwsze zapytanie wyświetla nieznaną nazwę kolumny `?column?`. Zalecane jest użycie aliasu w zapytaniu.

ZADANIA

Zadanie 1: Wyświetlić nazwy podkategorii należących do kategorii 'Education' i średnie czasy trwania filmów należących do poszczególnych podkategorii.

```
        name        |         avg
--------------------+----------------------
 Biology            |  4730854.142857142857
 Physics            |
 Technology         | 1027413.5000000000000
 Telecommunications |  3715850.000000000000
 Mathematics        |
 Geography          |  2741565.000000000000
(6 rows)
```

Zadanie 2: Wyświetlić tytuły wszystkich filmów nie należących do kategorii 'Education' (stosując podzapytania), posortowane rosnąco:

```
                          title
-------------------------------------------------------
 Bolek & Lolek - Australian steppes
 Bolek & Lolek - Gold town
 Bolek & Lolek - On tiger's trails
 Bolek & Lolek - Pharaoh's tomb
 Bolek & Lolek - Race to the North Pole
 Bolek & Lolek - Smuggler
 Fore Systems (advertisement)
 Gustavus and the fly
 Gustavus is a muff
 Gustavus is cheating
 Gustavus participates in traffic
 Gustavus wants to marry
 Infomercial
 Intranet (advertisement)
 Lucent 1
 Lucent 2
 Matematyczny model transmisji pakietów w kanale (M/M/1)
 Nowy budynek KT
 Nowy budynek KT
 OVS Introduction
 OVS MPEG1 1536k
 OVS MPEG1 2048k
 OVS MPEG1 2048k PAL
 OVS Sample 1
 OVS Sample 2
 Prezentacja Katedry Telekomunikacji AGH
 The Big Blue
 Wystąpienie rektora AGH
(28 rows)
```

Zadanie 3: Wyświetlić nazwy kategorii, w których minimalna długość filmu w bajtach jest większa niż 2000000.

```
   name
-----------
 Amusement
 Education
 KT
(3 rows)
```

Zadanie 4: Podać statystykę liczby epizodów w filmach. Poniższe wyniki oznaczają, że w bazie jest 40 filmów, dla których nie zdefiniowano epizodów, 2 filmy z jednym epizodem, jeden film z pięcioma epizodami itd.

```
 no of episodes | no of films
----------------+-------------
             11 |           1
              5 |           1
              1 |           2
              0 |          40
(4 rows)
```

Zadanie 5. Wypisać ID, nazwę i długość filmów, które są dłuższe od średniej z długości pozostałych filmów. Jako ostatnią kolumnę wyświetlić średnią z długości pozostałych filmów:

```
 mid |             title                | lenmsec | avg lenmsec excl. given mid
-----+----------------------------------+---------+-----------------------------
  80 | Image coding (part 2)            | 5436129 |        1685774.604651162791
 122 | Microcosmos (medium quality)     | 4534809 |        1706735.534883720930
 101 | Video coding                     | 5385605 |        1686949.581395348837
  98 | Image coding (part 1)            | 5501871 |        1684245.720930232558
 100 | Image coding (part 4)            | 6965989 |        1650196.465116279070
 119 | Overhead telecommunications lines | 1897170 |        1768075.976744186047
 124 | Saga of life (medium quality)    | 6363459 |        1664208.790697674419
 111 | Saga of life (high quality)      | 6422197 |        1662842.790697674419
 110 | Southern Africa Safari           | 3571616 |        1729135.372093023256
 103 | Microcosmos (high quality)       | 4379470 |        1710348.069767441860
 121 | Microcosmos (low quality)        | 4554451 |        1706278.744186046512
 123 | Saga of life (low quality)       | 3289977 |        1735685.116279069767
 112 | The World's Deadliest Volcanoes  | 2741565 |        1748438.883720930233
 113 | The Big Blue                     | 6876310 |        1652282.023255813953
(14 rows)
```

Zadanie 6. Wypisz nazwy subkategorii, dla których sumaryczny czas trwania filmów należących do danej subkategorii jest dłuższy od 1000 sekund. Dodatkowo wypisz czas trwania najdłuższego filmu i liczbę filmów dla danej subkategorii. Użyj podzapytań.

```
        name        |   max   | count
--------------------+---------+-------
 budowa             |  846550 |     2
 Cartoons           |  607362 |    11
 wykłady            |  757567 |     3
 Movies             | 6876310 |     1
 Biology            | 6422197 |     7
 Technology         | 1897170 |     2
 Geography          | 2741565 |     1
 Telecommunications | 6965989 |     5
(8 rows)
```

Zadanie 7. Wypisz nazwy subkategorii, dla których średni czas trwania epizodów jest dłuższy od 15 sekund. Dodatkowo wypisz średni czas trwania epizodów i liczbę epizodów należących do danej subkategorii. Użyj podzapytań. W zapytaniu głównym nie używaj iloczynu kartezjańskiego.

```
   name    |         avg          | count
-----------+----------------------+-------
 Biology   | 142582.000000000000 |     5
 Geography | 194136.363636363636 |    11
(2 rows)
```

Zadanie 8. Wykonaj zadanie 7 bez używania iloczynu kartezjańskiego w zapytaniu głównym i w którymkolwiek z podzapytań.

```
   name    |         avg          | count
-----------+----------------------+-------
 Biology   | 142582.000000000000 |     5
 Geography | 194136.363636363636 |    11
(2 rows)
```

Widoki

Zadanie 9: Utworzyć widok o nazwie `srednie_v` (polecenie CREATE VIEW) zawierający nazwy podkategorii należących do kategorii „Education" oraz średnie czasy trwania filmów w poszczególnych podkategoriach.
Wyświetlić dane z widoku `srednie_v`.

```
        name        |         avg
--------------------+----------------------
 Biology            |  4730854.142857142857
 Physics            |
 Technology         | 1027413.5000000000000
 Telecommunications |  3715850.000000000000
 Mathematics        |
 Geography          |  2741565.000000000000
(6 rows)
```

Zadanie 10: Utworzyć nową tabelę o nazwie `srednie_t` (polecenie CREATE TABLE) zawierającą nazwy podkategorii należących do kategorii „Education" oraz średnie czasy trwania filmów w poszczególnych podkategoriach.
Wyświetlić dane z tabeli `srednie_t`.

```
        name        |         avg
--------------------+----------------------
 Biology            |  4730854.142857142857
 Physics            |
 Technology         | 1027413.5000000000000
 Telecommunications |  3715850.000000000000
 Mathematics        |
 Geography          |  2741565.000000000000
(6 rows)
```

Zadanie 11: Jeden z filmów należących do podkategorii „Biology" ma czas trwania równy 6363459. Zmienić czas trwania tego filmu na 1 000 000.
Następnie wyświetlić zawartość widoku `srednie_v`:

```
        name        |         avg
--------------------+----------------------
 Biology            |  3964645.714285714286
 Physics            |
 Technology         | 1027413.5000000000000
 Telecommunications |  3715850.000000000000
 Mathematics        |
 Geography          |  2741565.000000000000
(6 rows)
```

oraz tabeli `srednie_t`:

```
        name        |         avg
--------------------+----------------------
 Biology            |  4730854.142857142857
 Physics            |
 Technology         | 1027413.5000000000000
 Telecommunications |  3715850.000000000000
 Mathematics        |
 Geography          |  2741565.000000000000
(6 rows)
```

Dlaczego średnia długość filmów należących do kategorii „Biology" zmieniła się w przypadku widoku `srednie_v` a nie zmieniła w tabeli `srednie_t` ?

Zastosowanie rozkazu select przy aktualizacji bazy

Zadanie 12: Ustawić pole „sysflags" w tabeli `movies_list` na wartość 7 dla tego filmu, dla którego czas trwania jest najdłuższy ze wszystkich filmów.

```
UPDATE 1
```

Zadanie 13: Ustawić pole „sysflags" w tabeli `movies_list` na wartość 8 dla tych filmów, które są najkrótsze w swoich podkategoriach.

```
UPDATE 11
```

Laboratorium 5

- **Złączenia**
- **„CROSS JOIN”**
- **„INNER JOIN”**
- **„LEFT/RIGHT JOIN”**
- **„FULL JOIN”**
- **Wielokrotne użycie „JOIN”**

Wymagany materiał z wykładu:

- **Rozdział 5 – NULL**
- **Rozdział 9 – Rodzaje złączeń**

Złączenia

Złaczenia, podobnie jak połączenie oraz podzapytania, umożliwiają pobieranie w zapytaniu danych z kilku tabel jednocześnie. Złączenie powstaje z dwóch tabel, przy czym sposób tworzenia wierszy zależy od typu złączenia. Istnieją dwa podstawowe typy złączeń – bezwarunkowe (CROSS JOIN) oraz warunkowe (pozostałe).

Dla złączeń warunkowych warunek może być podany na trzy sposoby:

- przy użyciu klauzuli „ON *warunek_logiczny*" – wybierane są wiersze spełniające wskazany warunek
- przy użyciu klauzuli „USING (*lista_kolumn)*" – wybierane są te wiersze, które we wskazanych kolumnach zawierają identyczne wartości
- przy użyciu klauzuli „NATURAL" – odpowiada klauzuli USING z listą kolumn zawierającą wszystkie kolumny o takich samych nazwach w obu tabelach.

W przypadku użycia klauzuli ON złączenie zawiera kolumny z obu tabel. W przypadku klauzul USING i NATURAL, kolumny użyte w tych warunkach występują w wyniku tylko raz.

CROSS JOIN

Złączenie bezwarunkowe (CROSS JOIN) odpowiada funkcjonalnością iloczynowi kartezjańskiemu.

Wyświetlić produkt powstały ze skrzyżowania tabel: categories i subcategories

```
SELECT * FROM categories CROSS JOIN subcategories;
```

Powyższe zapytanie jest równoważne zapytaniu:

```
SELECT * FROM categories, subcategories;
```

INNER JOIN

Złączenie warunkowe INNER JOIN odpowiada połaczeniu (iloczyn kartezjański z warunkiem).

Wyświetlić listę podkategorii i nazwy kategorii do których należą.

```
SELECT * FROM categories c INNER JOIN subcategories s ON c.cid = s.cid;
```

Zapytanie jest równoważne następującemu zpytaniu:

```
SELECT * FROM categories c, subcategories s WHERE c.cid = s.cid;
```

Porównać wyniki z zapytaniami:

```
SELECT * FROM categories c INNER JOIN subcategories s USING (cid);
SELECT * FROM categories c NATURAL INNER JOIN subcategories s;
```

Zwrócić uwagę na liczbę i nazwy kolumn.

LEFT/RIGHT JOIN

Zapytanie odpowiada zapytaniu INNER JOIN, z tym, że zawiera także wiersze z LEWEJ/PRAWEJ tabeli, które nie mają odpowiednika w drugiej tabeli.

```
select m.mid, s.name from movies_list m LEFT JOIN subcategories s USING(sid);
```

Zwrócić uwagę, że wyświetlony został film nie przypisany do żadnej podkategorii.

Zadanie: użyć "RIGHT JOIN", porównać wyniki.

FULL JOIN

Zapytanie odpowiada zapytaniu INNER JOIN, z tym, że zawiera także wiersze z LEWEJ tabeli, które nie mają odpowiednika w PRAWEJ tabeli, oraz wiersze z PRAWEJ tabeli, które nie mają odpowiednika w lewej tabeli.

```
select m.mid, s.name from movies_list m FULL JOIN subcategories s USING(sid);
```

Zwrócić uwagę, że wyświetlone zostały zarówno filmy nie przypisane do żadnej podkategorii, jak też podkategorie nie zawierające żadnych filmów.

Wielokrotne użycie JOIN

Klauzula JOIN może być użyta wielokrotnie, np:

```
SELECT e.title, c.name as category, s.name as subcategory FROM episodes_list e
INNER JOIN movies_list m USING (mid)
INNER JOIN categories c USING (cid)
INNER JOIN subcategories s USING (sid)
WHERE is_movie=1;
```

W takim przypadku najpierw wykonywane jest złączenie tabel `movies_list` i `episodes_list`, wynik jest łączony z `categories`, a następnie z `subcategories`.

Zadania

Zadanie 1: Wyświetlić nazwę kategorii i sumę czasów trwania filmów należących do poszczególnych kategorii. Użyj CROSS JOIN.

```
    name    |   sum
------------+----------
 Amusement  | 11882469
 Commercial |   804406
 Education  | 56491621
 KT         |  3309812
(4 rows)
```

Zadanie 2: Policzyć ile epizodów (jako episode nr) należy do danego filmu. Wypisać tytuły filmów, których te epizody są fragmentami (jako movie). Posortować według tytułu filmu (rosnąco). Dodatkowo wyświetlić nazwę kategorii i podkategorii. Użyj CROSS JOIN.

```
 episode nr |                  movie                  | category  | subcategory
------------+-----------------------------------------+-----------+-------------
          1 | Bolek & Lolek - Race to the North Pole  | Amusement | Cartoons
          5 | Microcosmos (high quality)              | Education | Biology
          1 | Prezentacja Katedry Telekomunikacji AGH | KT        | wykłady
         11 | The World's Deadliest Volcanoes         | Education | Geography
(4 rows)
```

Zadanie 3: Wykonaj **Zadanie 1** i **Zadanie 2** z użyciem INNER JOIN.

Zadanie 4: Wyświetlić tytuł filmu, NAZWĘ kategorii i NAZWĘ podkategorii, do której należy dany film. Mają być wyświetlone wszystkie filmy, nawet te nie przypisane do kategorii.

```
(44 rows)
```

Zadanie 5: Policzyć ile filmów należy do danej kategorii. W statystyce uwzględnić filmy, które nie należą do żadnej kategorii pod nazwą "--n/a--".

```
count |  coalesce
-------+------------
    13 | Amusement
     7 | Commercial
    15 | Education
     8 | KT
     1 | --n/a--
(5 rows)
```

Zadanie 6: Wyświetlić listę, zawierającą dwie kolumny: tytuł epizodu, tytuł filmu. Na wyniku mają sie pojawić wszystkie filmy (nawet te, które nie posiadają epizodów).

Uwaga: wyniki zostały skrócone!!!

```
 The bee                                  | Microcosmos (high quality)
 The ladybird                             | Microcosmos (high quality)
 Snails' sexual act                       | Microcosmos (high quality)
                                          | Wystąpienie rektora AGH
                                          | Nowy budynek KT
                                          | Nowy budynek KT
                                          | The Big Blue
                                          | Lucent 1
                                          | Lucent 2
(58 rows)
```

Zadanie 7: Dla każdej kategorii wyświetlić sumę długości filmów należących do niej. Dla kategorii, które nie zawierają żadnych filmów suma długości ma być równa 0 (nie NULL!!!). Filmy nie należące do żadnej kategorii należy uwzględnić jako osobną kategorię bez nazwy (NULL).

```
   case   |    name
----------+------------
 11882469 | Amusement
   804406 | Commercial
 51128162 | Education
  3309812 | KT
        0 | Others
  5436129 |
(6 rows)
```

Laboratorium 6

- **Wyszukiwanie ciągów znaków**
- **Tworzenie sekwencji**
- **Korzystanie z sekwencji**

Wymagany materiał z wykładu:

- **Rozdział 6 – Operatory porównania**
- **Rozdział 10 – Pojęcie klucza i inne warunki (constraints)**
- **Rozdział 11 – Generatory sekwencji**

Wyszukiwanie ciągów znaków

Porównania ciągów znaków można dokonać za pomocą:
- operatorów LIKE lub NOT LIKE
- operatorów porównania wyrażeń regularnych:
  - `~` równe wyrażeniu regularnemu, case sensitive
  - `!~` różne od wyrażenia regularnego, case sensitive
  - `~*` równe wyrażeniu regularnemu, case insensitive
  - `!~*` różne od wyrażenia regularnego, case insensitive

W pierwszym przypadku wzorzec może zawierać:
_ dowolny pojedynczy znak
% dowolny ciąg znaków (zero lub więcej)

W przypadku wyrażeń regularnych między innymi można wymienić następujące znaki specjalne:
`.` dowolny pojedynczy znak
[ace] jeden ze znaków a lub c lub e
[a-c] jeden ze znaków a, b, c
[a-cx-z] jeden ze znaków a, b, c, x, y, z
`*` zero lub więcej wystąpień znaku poprzedzającego (czyli `.*` będzie oznaczać dowolny ciąg znaków, zaś `t*` dowolną liczbę wystąpień znaku 't')
\{n\} dokładnie n wystąpień znaku poprzedzającego
`^` początek ciągu znaków
`$` koniec ciągu znaków

Przykład 1: Na dwa sposoby wyświetlić wszystkie tytuły filmów i epizodów, które zaczynają się od litery 'T'.

```
SELECT title FROM episodes_list WHERE title ~ '^T';
SELECT title FROM episodes_list WHERE title like 'T%';
              title
--------------------------------
 Technology LSA-PLUS
 The World's Deadliest Volcanoes
 The first morning
 The bee
 The ladybird
 The Big Blue
(6 rows)
```

Przykład 2: Na dwa sposoby wyświetlić wszystkie tytuły filmów i epizodów, które zawierają literę 'R'.

```
select title from episodes_list where title ~'R';
select title from episodes_list where title like '%R%';
                 title
---------------------------------------
 Revolution to orbit
 Nevado del Ruiz (Colombia)
 Mount Rainier (Washington, USA)
 Bolek & Lolek - Race to the North Pole
(4 rows)
```

Zadanie 1: Wyświetlić wszystkie tytuły filmów i epizodów, które zaczynają się od litery 'T' i nie kończą znakiem 'e'.

```
              title
--------------------------------
 Technology LSA-PLUS
 The World's Deadliest Volcanoes
 The first morning
 The ladybird
(4 rows)
```

Zadanie 2: Wyświetlić wszystkie tytuły filmów i epizodów, które zaczynają się od litery 'T' i kończą znakiem 'e'. Użyć tylko jeden warunek.

```
    title
--------------
 The bee
 The Big Blue
(2 rows)
```

Zadanie 3: Na dwa sposoby wyświetlić tytuły filmów i epizodów które zawierają słowo 'The' lub 'the'.

```
                title
--------------------------------------
 Gustavus and the fly
 The World's Deadliest Volcanoes
 Southern Africa Safari
 Bolek & Lolek - Race to the North Pole
 The first morning
 The bee
 The ladybird
 The Big Blue
(8 rows)
```

Zadanie 4: Wyświetlić tytuły filmów i epizodów w których występuje cyfra ze zbioru: 0, 1, 2, 3, 5, 6.

```
                          title
-------------------------------------------------------
 OVS Sample 1
 OVS MPEG1 1536k
 OVS MPEG1 2048k
 OVS MPEG1 2048k PAL
 OVS Sample 2
 Image coding (part 1)
 Image coding (part 2)
 Matematyczny model transmisji pakietów w kanale (M/M/1)
 Lucent 1
 Lucent 2
(10 rows)
```

Zadanie 5: Wyświetlić tytuły filmów i epizodów w których występują koło siebie cztery dowolne cyfry ze zbioru: 0, 1, 2, 3, 5, 6

```
      title
-----------------
 OVS MPEG1 1536k
(1 row)
```

Zadanie 6: Wyświetlić identyfikatory filmów dla których rozmiar pliku (lenbin) zawiera się w granicach 10 000 000 i 30 000 000. Zastosować operator between.

```
 mid
-----
  70
 108
(2 rows)
```

Zadanie 7: Wyświetlić tytuły filmów z zadania 6. Posłużyć się podzapytaniem (nie krzyżować tabel) i operatorem *in*.

```
          title
--------------------------
 Intranet (advertisement)
 Infomercial
(2 rows)
```

Tworzenie sekwencji

Przykład 3: Utworzenie najprostszej sekwencji. Domyślnie przyjmowane jest minimum=1, maksimum=2^{63}-1, start=1 oraz skok=1. *Domyślne wartości w innych przypadkach - patrz manual.*

```
CREATE SEQUENCE s_1;
select nextval('s_1');
 nextval
---------
       1
(1 row)
SELECT nextval('s_1');
 nextval
---------
       2
(1 row)
SELECT currval('s_1');
 currval
---------
       2
(1 row)
```

Przykład 4: Utworzenie sekwencji o zdefiniowanych wartościach minimum (-10) i maksimum (100), wartości początkowej (100) i skoku (-2).

```
CREATE SEQUENCE s_2 INCREMENT -2 MAXVALUE 100 MINVALUE -10 START 100;
SELECT nextval('s_2');
 nextval
---------
     100
(1 row)
SELECT nextval('s_2');
 nextval
---------
      98
(1 row)
```

Przykład 5: Utworzenie sekwencji cyklicznej.

```
CREATE SEQUENCE s_3 INCREMENT 1 MAXVALUE 5 MINVALUE 1 START 4 CYCLE;
SELECT nextval('s_3');
 nextval
---------
       4
(1 row)
SELECT nextval('s_3');
 nextval
---------
       5
(1 row)
SELECT nextval('s_3');
 nextval
---------
       1
(1 row)
SELECT nextval('s_3');
 nextval
---------
       2
(1 row)
```

Przykład 6: Osiągnięcie wartości granicznej sekwencji

```
CREATE SEQUENCE s_4 INCREMENT 1 MAXVALUE 5 MINVALUE 1 START 4;
SELECT nextval('s_4');
 nextval
---------
       4
(1 row)
SELECT nextval('s_4');
 nextval
---------
       5
(1 row)
SELECT nextval('s_4');
SELECT setval('s_4',3);   (ustawienie wartości bieżącej)
SELECT nextval('s_4');
```

Korzystanie z sekwencji

Sekwencji używa się najczęściej w celu uzyskania unikalnego klucza w tabeli. Można to zrobić na kilka sposobów.

Przykład 7: „Tradycyjna" definicja tabeli.

```
CREATE TABLE ksiazki (id int4, tytul varchar(15));
```

Jeżeli chcemy zapewnić, że w kolumnie `id` mają znajdować się unikalne wartości z sekwencji s_1 musimy przy wstawianiu każdego wiersza wprowadzać inkrementowaną wartość z tej sekwencji:

```
INSERT INTO ksiazki VALUES (nextval('s_1'),'Pan Tadeusz');
```

Jeżeli się o tym zapomni można spowodować niespójność bazy:

```
INSERT INTO ksiazki (tytul) VALUES ('Pinokio');
SELECT * FROM ksiazki;
 id |    tytul
----+-------------
  3 | Pan Tadeusz
    | Pinokio
(2 rows)

DROP TABLE ksiazki;
```

Przykład 8: Dla wygody można ustawić domyślną wartość kolumny `id`.

```
CREATE TABLE ksiazki (id int4 DEFAULT nextval('s_1'), tytul varchar(15));
INSERT INTO ksiazki (tytul) VALUES ('Pinokio');
INSERT INTO ksiazki (tytul) VALUES ('Pan Tadeusz');
SELECT * FROM ksiazki;
 id |    tytul
----+-------------
  4 | Pinokio
  5 | Pan Tadeusz
(2 rows)

DROP TABLE ksiazki;
```

Przykład 9: Można skorzystać z typu danych `serial`. Zmienna o typie serial przyjmuje wartości od 0 do +2147483647 ponieważ typ `serial` odpowiada typowi `integer`. Jeśli wymagane są większe wartości dla sekwencji należy użyć typu `bigserial`, który odpowiada typowi `bigint`. Wówczas możliwy jest zakres od do 2^{63}-1.

```
CREATE TABLE ksiazki (id serial,  tytul varchar(15));
NOTICE:  CREATE TABLE will create implicit sequence 'ksiazki_id_seq' for SERIAL column
'ksiazki.id'
```

W praktyce, utworzona została „zwykła" sekwencja o nazwie `'ksiazki_id_seq'`. Sekwencja ma domyślnie ustawioną wartość maksymalną na 2^{63}-1. Jednakże jeśli typ kolumny zdefiniowano jako serial, wówczas maksymalna możliwa do wstawienia wartość będzie 2^{31}-1.

```
\ds
              List of relations
Schema |     Name     |   Type   |  Owner
--------+--------------+----------+----------
 public | books_id_seq | sequence | rstankie
 public | s_1          | sequence | rstankie
 public | s_2          | sequence | rstankie
 public | s_3          | sequence | rstankie
 public | s_4          | sequence | rstankie
(5 rows)

INSERT INTO ksiazki (tytul) VALUES ('Pan Tadeusz');
INSERT INTO ksiazki (tytul) VALUES ('Pinokio');
INSERT INTO ksiazki (tytul) VALUES ('Puchatek');
```

```
SELECT * FROM ksiazki;
 id |    tytul
----+-------------
  1 | Pan Tadeusz
  2 | Pinokio
  3 | Puchatek
(3 rows)

DROP TABLE ksiazki;

\ds
         List of relations
 Schema | Name |   Type   |  Owner
--------+------+----------+----------
 public | s_1  | sequence | rstankie
 public | s_2  | sequence | rstankie
 public | s_3  | sequence | rstankie
 public | s_4  | sequence | rstankie
(4 rows)
```

Uwaga! Skasowanie tabeli `ksiazki` spowodowało automatyczne usunięcie sekwencji `ksiazki_id_seq`.

Laboratorium 7

- **Transakcje**
- **Blokady wierszy**
- **Izolacja transakcji**

Wymagany materiał z wykładu:

- **Rozdział 12 – Transakcje**

Transakcje

Najistotniejszą sprawą przy projektowaniu i korzystaniu z bazy danych jest zachowanie spójności informacji w KAŻDYCH warunkach, również w przypadku równoczesnego dokonywania operacji przez wielu użytkowników. Jednym z mechanizmów, umożliwiających zapewnienie spójności są transakcje.

Transakcja jest wykonywana na zasadzie „wszystko albo nic", tzn. albo wszystkie komendy w transakcji zostaną poprawnie wykonane, albo (w razie dowolnego błędu) wszystkie dokonane w transakcji zmiany zostaną odwołane.

Domyślnie w systemie Postgres transakcja obejmuje jedno polecenie wydane przez użytkownika (baza zawsze wykonuje polecenia użytkownika w transakcji). Zatem polecenie `select * from categories;` jest w rzeczywistości wykonywane jako:

```
begin;
select * from categories;
commit;
```

Można jednak wymusić poleceniami `BEGIN` i `COMMIT` wykonanie kilku poleceń w ramach jednej transakcji.

Do wykonania poniższych ćwiczeń należy otworzyć dwa okna, w obu uruchomić *psql*. Zapytania z lewej kolumny należy wykonywać w jednym oknie, zapytania z drugiej - w drugim. Polecenia należy wykonywać kolejno, w odpowiednich oknach. Jeżeli w jednej linijce są dwa polecenia dla różnych okien to można je wykonać w dowolnej kolejności.

Przykład transakcji z *rollback*:

Lp	Okno 1	Okno 2
1.		`\d`
2.	`begin;` *otwarcie transakcji*	
3.	`create table test as select * from categories;`	
4.	`\d` *tabela test jest widoczna tylko wewnątrz tej transakcji*	`\d` *tabela test nie jest widoczna bo transakcja w oknie 1 nie została jeszcze zatwierdzona*
5.	`rollback;` *odwołanie transakcji*	
6.	`\d` *tabela test nie istnieje, bo transakcja została odwołana*	`\d`

Przykład transakcji z *commit*:
Zwrócić uwagę na to, że tabela 'osoby' nie jest widoczna do czasu wykonania 'commit'.

Lp	Okno 1	Okno 2
7.	`begin;`	
8.	`create table osoby (imie varchar(40), nazw varchar(40), konto int4);`	
9.	`\d`	`\d`
10.	`insert into osoby (imie, nazw, konto) values ('Jan', 'Kos', 3000);`	
11.	`select * from osoby;` *wprowadzone wiersze są widoczne tylko wewnątrz transakcji*	`select * from osoby;` *zwraca błąd, bo na zewnątrz niezatwierdzonej transakcji tabela osoby nie istnieje*
12.	`insert into osoby (imie, nazw, konto) values ('Ala', 'Waga', 4000);`	
13.	`select * from osoby;`	
14.	`commit;` *zatwierdzenie transakcji*	
15.	`\d`	`\d`
16.		`select * from osoby;` *zarówno tabela jak i wprowadzone wiersze są widoczne dla wszystkich*

Błąd w transakcji powoduje przejście do trybu "Abort state". Przez błąd należy rozumieć zarówno błąd w składni polecenia, jak też polecenie, które narusza ograniczenia bazy danych. Naruszenie ograniczeń ma miejsce np. wtedy, gdy kolumna w tabeli ma zawierać wartości dodatnie a polecenie próbuje wstawić wartość ujemną. Błędem NIE jest polecenie update (delete) które aktualizuje (kasuje) 0 wierszy (nie znajdzie wiersza pasującego do klauzuli *where*).
W momencie przejścia do trybu *abort state wyk*onywany jest automatycznie rollback a reszta instrukcji w transakcji jest ignorowna. Takie zachowanie pozwala bez ryzyka wykonać zestaw komend (np. ze skryptu) bez sprawdzania, czy każda z nich się wykonała poprawnie.

Lp	Okno 1	Okno 2
17.	`begin;`	`select * from osoby;`
18.	`update osoby set konto=konto-1000 where nazw='Kos';`	
19.	`update osoby set konto=konto+1000 where nazw='Waga';`	
20.	`select` **`konta`** `from osoby;` ***UWAGA: w tej linijce jest celowy błąd - 'konta'***	
21.	`select konto from osoby;` *komenda zostanie zignorowana - Abort State*	`select * from osoby;`
22.	`commit;` *koniec transakcji, transakcja została ODWOŁANA ze względu na wcześniejszy błąd*	
23.		`select * from osoby;` *zawartość tabeli bez zmian*

Blokady wierszy

Modyfikowanie tych samych danych równocześnie przez kilka transakcji może prowadzić do błędów (takich jak błędne wyliczenie stanu konta, sprzedaż jednego towaru kilku osobom). Blokady wierszy są jednym z mechanizmów pozwalających na uniknięcie takiej sytuacji.

Wykonanie polecenia „update" powoduje automatyczną blokadę modyfikowanego wiersza. Inne transakcje próbujące zmodyfikować taki wiersz muszą poczekać na zakończenie pierwszej transakcji. Blokada obowiązuje do końca transakcji, w której została założona.

Lp	Okno 1	Okno 2
24.	`begin;`	`begin;`
25.	`update osoby set konto=1000 where nazw='Waga';`	
26.		`update osoby set konto=konto+1000 where nazw='Waga';` *polecenie oczekuje na zakończenie konkurencyjnej transakcji*
27.	`commit;`	*po commit w oknie 1 polecenie z poprzedniej linii zostanie wykonane*
28.		`commit;`
29.		`select * from osoby;`

Poniższy przykład prezentuje często popełniany błąd, polegający na niezablokowaniu wiersza przeznaczonego do modyfikacji już w momencie odczytu aktualnej wartości. Powoduje to, że w obu transakcjach zostanie odczytane dodatnie saldo konta pani Wagi i w konsekwencji zaakceptowanie obu przelewów; po ich wykonaniu pani Waga ma debet (a to nie powinno się zdarzyć - drugi przelew powinien zostać odrzucony).

Lp	Okno 1	Okno 2
30.		`update osoby set konto=1000 where nazw='Waga';` *przywrócenie danych w bazie do stanu pierwotnego*
31.	`begin;`	`begin;`
32.	`select konto from osoby where nazw='Waga';`	
33.		`select konto from osoby where nazw='Waga';`
34.	`update osoby set konto=konto-1000 where nazw='Waga';`	
35.	`update osoby set konto=konto+1000 where nazw='Kos';`	
36.	`commit;`	
37.		*użytkownik nie wie, że stan konta się zmienił od czasu zapytania z linii 33, dlatego kontynuuje wykonanie przelewu* `update osoby set konto=konto-1000 where nazw='Waga';`
38.		`update osoby set konto=konto+1000 where nazw='Kos';`
39.		`commit;`
40.		`select * from osoby;` *tutaj p. Waga ma już debet*

Rozwiązaniem jest użycie w obu transakcjach „select for update”, który zablokuje wiersz już w momencie odczytu.

Lp	Okno 1	Okno 2
41.		`update osoby set konto=1000 where nazw='Waga';` *przywrócenie danych w bazie do stanu pierwotnego*
42.	`begin;`	`begin;`
43.	`select konto from osoby where nazw='Waga' for update;` *zapewniamy sobie wyłączność na dostęp do wyświetlonych danych*	
44.		`select konto from osoby where nazw='Waga' for update;` *select jest zablokowany do czasu zakończenia innych transakcji operujących na tym wierszu*
45.	`update osoby set konto=konto-1000 where nazw='Waga';`	
46.	`update osoby set konto=konto+1000 where nazw='Kos';`	
47.	`commit;`	
48.		*select został wykonany; p. Waga ma stan konta = 0 więc użytkownik podejmuje decyzję o przerwaniu transakcji* `rollback;`

Blokowanie wierszy może prowadzić do powstania zakleszczeń (deadlocks). W takich sytuacjach postgres radzi sobie sam.

Lp	Okno 1	Okno 2
49.	`begin;`	`begin;`
50.		`update osoby set konto=1000 where nazw='Waga';`
51.	`update osoby set konto=2000 where nazw='Kos';`	
52.		`update osoby set konto=1000 where nazw='Kos';`
53.	`update osoby set konto=2000 where nazw='Waga';`	
54.	`commit;`	`commit;`

W przedstawionym poniżej przykładzie nie da się bezpośrednio zastosować blokowania rekordów za pomocą „select for update." Operujemy tutaj dwoma tabelami: kursy i miejsca. Pierwsza z nich zawiera informacje o kursach autobusów, druga o miejscach zarezerwowanych w poszczególnych kursach. Pojawienie się rekordu w tabeli miejsca oznacza, że dane miejsce jest zajęte. Konflikt pomiędzy transakcjami w poniższym przykładzie wynika stąd, że obie transakcje odczytają, że miejsce nr. 1 w kursie 1 jest wolne, a w konsekwencji wstawią dwie rezerwacje tego samego miejsca

Lp	Okno 1	Okno 2

55.	*Przygotowanie tabel:*
	`create table kursy (kurs int4, opis varchar(40));`
	`create table miejsca (kurs int4, miejsce int4);`
	`insert into kursy (kurs, opis) values (1, 'Kurs1');`	

56.	`begin;`	`begin;`
57.	*Sprawdzamy, czy miejsce jest wolne* `select * from miejsca where kurs=1;`	
58.	*Miejsce jest wolne – rezerwujemy je* `insert into miejsca (kurs, miejsce) values (1, 1);`	
59.		*Sprawdzamy, czy miejsce jest wolne* `select * from miejsca where kurs=1;`
60.		*Miejsce jest wolne – rezerwujemy je* `insert into miejsca (kurs, miejsce) values (1, 1);`
61.	`commit;`	`commit;`
62.	`select * from miejsca where kurs=1;`	

Opisany konflikt można rozwiązać na dwa sposoby. Pierwszym z nich jest użycie PRZED sprawdzeniem wolnych miejsc komendy `LOCK TABLE miejsca in ACCESS EXCLUSIVE MODE`, co spowoduje, że nasza transakcja uzyska wyłączność na dostęp do tej tabeli. Wyłączność dotyczy również odczytu danych, co oznacza, że inne transakcje nie są w stanie sprawdzać wolnych miejsc. Jest to istotna wada tego rozwiązania, gdyż przy dużych bazach radykalnie spowolni korzystanie z systemu.

Drugie rozwiązanie zakłada, że wszyscy korzystający z bazy działają według tej samej procedury. Jeżeli wszyscy zainteresowani rezerwacją miejsc wykonają przed sprawdzeniem wolnych miejsc „select for update" na wierszu opisującym konkretny kurs, to efektywnie tylko jedna transakcja operująca na danym kursie będzie działać, reszta będzie czekać:

Lp	Okno 1	Okno 2
63.	`delete from miejsca;`	
64.	`begin;`	`begin;`
65.	`select kurs from kursy where kurs=1 for update;`	
66.		`select kurs from kursy where kurs=1 for update;`
67.	*Sprawdzamy, czy miejsce jest wolne* `select * from miejsca where kurs=1;`	
68.	*Miejsce jest wolne – rezerwujemy je* `insert into miejsca (kurs, miejsce) values (1, 1);`	
69.	`commit;`	
70.		*Sprawdzamy, czy miejsce jest wolne* `select * from miejsca where kurs=1;`
71.		*Miejsce jest zajęte – wykonujemy rollback* `rollback;`
72.	`select * from miejsca where kurs=1;`	

Zysk w porównaniu do LOCK TABLE jest podwójny: transakcje operujące na różnych kursach nie blokują się wzajemnie, ponadto transakcje, które w celach czysto informacyjnych uzyskują informacje o wolnych miejscach nie są blokowane wcale.

Izolacja transakcji

W bazie danych mogą występować trzy niepożądane zjawiska:
- dirty reads - transakcja odczytuje dane wprowadzone ale niezatwierdzone przez inną transakcję.
- non-repeatable reads - zjawisko jest widoczne w sytuacji, gdy transakcja odczytuje dane, czeka chwilę, następnie ponownie odczytuje te same dane i stwierdza, że dane uległy zmianie (przez inną zatwierdzoną transakcję)
- phantom read - transakcja ponownie wykonuje zapytanie zwracające określone wiersze i stwierdza, że pojawiły się nowe wiersze spełniające zadane kryterium (wstawione przez inną, zatwierdzoną transakcję)

Zajwisko „dirty-reads" nie występuje w Postgresie, co można zauważyć w poprzednich przykładach. Występowanie zjawisk „non-repeatable reads" oraz „phantom read" zależy od bieżącego poziomu izolacji transakcji. Domyślnym poziomem izolacji jest poziom „Read Committed", w którym te zjawiska występują. Można korzystać także z poziomu „Serializable", w którym te zjawiska są eliminowane.

Tryb serializable emuluje wykonywanie transakcji tak, jakby były wykonywane jedna po drugiej a nie równolegle. Stąd w trybie „Serializable" transakcja „widzi" stan bazy z momentu rozpoczęcia transakcji. Zmiany wprowadzone do bazy przez inne transakcje nie są widoczne. Ponadto operacje na wierszu (UPDATE, DELETE, SELECT FOR UPDATE), który jest aktualizowany przez konkurencyjną transakcję są wstrzymywane do czasu zakończenia tej transakcji. Jeżeli konkurencyjna transakcja zostanie odwołana to bieżąca transakcja będzie działać nadal. Jeżeli konkurencyjna transakcja zostanie zatwierdzona, to bieżąca transakcja zostanie zerwana, ze względu na brak możliwości dalszego emulowania wykonywania transakcji jedna po drugiej.

Lp	Okno 1	Okno 2
73.	*przykład w domyślnym trybie pracy bazy danych* `begin;`	`begin;`
74.	`select * from osoby;` *zwrócić uwagę na stan konta p. Wagi*	
75.		`update osoby set konto=1500 where nazw='Waga';`
76.		`commit;`
77.	`select * from osoby;` *stan konta p. Wagi uległ zmianie (non-repeatable read)*	
78.	`update osoby set konto=2000 where nazw='Waga';` *update wykonuje się normalnie*	
79.	`commit;`	

Przykład zerwania transakcji (wygrała transakcja, która wcześniej zrobiła „update"). Zwrócić uwagę na fakt, że oba selecty w oknie 1 podają te same dane mimo commit'a w oknie 2. Jest to właściwe narzędzie do tworzenia raportów, zamraża bowiem spójny obraz bazy danych z chwili rozpoczęcia transakcji.

Lp	Okno 1	Okno 2
80.	`begin;`	`begin;`
81.	`set transaction isolation level serializable;` *włączamy tryb "serializable"*	
82.	`select * from osoby;`	
83.		`update osoby set konto=1000 where nazw='Waga';`
84.		`commit;`
85.	`select * from osoby;` *tym razem select podaje wartość taką samą jak poprzednio*	
86.	`update osoby set konto=2000 where nazw='Waga';` *próba aktualizacji nie powiedzie się, ze względu na modyfikację danych w konkurencyjnej transakcji*	
87.	`rollback;`	

Tryb "serializable" – transakcja nie jest zerwana ze względu na rollback tej drugiej transakcji

Lp	Okno 1	Okno 2
88.	`begin;`	`begin;`
89.	`set transaction isolation level serializable;`	
90.	`select * from osoby;`	
91.		`update osoby set konto=1000 where nazw='Waga';`
92.	`update osoby set konto=2000 where nazw='Waga';` *polecenie czeka na zakończenie konkurencyjnej transakcji*	
93.		`rollback;`
94.	`commit;`	

Przykład użycia trybu „serializable" do poprawnej realizacji przelewu

Lp	Okno 1	Okno 2
95.		`update osoby set konto=1000 where nazw='Waga';` *przywrócenie danych w bazie do stanu pierwotnego*
96.	`begin;`	`begin;`
97.	`set transaction isolation level serializable;`	`set transaction isolation level serializable;`
98.		`select konto from osoby where nazw='Waga';`
99.	`select konto from osoby where nazw='Waga';`	
100.	`update osoby set konto=konto-1000 where nazw='Waga';`	
101.	`update osoby set konto=konto+1000 where nazw='Kos';`	
102.	`commit;`	
103.		`update osoby set konto=konto-1000 where nazw='Waga';` *tutaj występuje błąd szeregowania (nasze zapytania przeplotły się z inną transakcją) – dalsze polecenia są ignorowane aż do końca transakcji*
104.		`update osoby set konto=konto+1000 where nazw='Kos';`
105.		`commit;` *efektywnie jest to rollback ze względu na błąd szeregowania*

Głównym powodem korzystania z trybu SERIALIZABLE jest zapewnienie spójnego obrazu danych przy tworzeniu raportów z dużych baz danych. Należy podkreślić, że do ochrony przed konkurencyjnymi transakcjami ten tryb NIE JEST konieczny, a czasem wręcz jest szkodliwy gdyż w razie kolizji powoduje zrywanie transakcji, co przy dużych i intensywnie użytkowanych bazach może wręcz uniemożliwiać pracę. Podstawowym mechanizmem ochrony przed konkurencyjnymi transakcjami są blokady wierszy (begin ... select for update ... update ... commit).

Laboratorium 8

- **Kursory**
- **Klauzule LIMIT oraz OFFSET (przy rozkazie SELECT)**
- **Prawa dostępu**
- **PL/pgSQL**

Wymagany materiał z wykładu:

- **Rozdział 13 – PL/SQL – Podstawy**
- **Rozdział 14 – PL/SQL – Triggery i funkcje**

Kursory

Przykład 1:

```
select sid from subcategories order by sid;

begin;
declare xyz cursor for select sid from subcategories order by sid;
fetch 5 from xyz;
fetch 1 from xyz;
fetch backward 1 from xyz;
move backward 3 from xyz;
fetch 1 from xyz;
fetch all from xyz;
move backward all from xyz;
fetch all from xyz;
close xyz;
commit;
```

Zadanie 1: Wyświetlić po 5 wierszy: tytuł filmu, metodę kompresji i nazwę kategorii, do której należy dany film. Wyniki sortowane po tytule. Można użyć iloczyn kartezjański.

```
                title                  | compress | category
---------------------------------------+----------+-----------
 Bolek & Lolek - Australian steppes    | MPEG 1   | Amusement
 Bolek & Lolek - Gold town             | MPEG 1   | Amusement
 Bolek & Lolek - On tiger's trails     | MPEG 1   | Amusement
 Bolek & Lolek - Pharaoh's tomb        | MPEG 1   | Amusement
 Bolek & Lolek - Race to the North Pole | MPEG 1   | Amusement
(5 rows)


           title             | compress |  category
-----------------------------+----------+------------
 Bolek & Lolek - Smuggler    | MPEG 1   | Amusement
 Course in a box             | MPEG 1   | Education
 Fore Systems (advertisement) | MPEG 1   | Commercial
 Gustavus and the fly        | MPEG 1   | Amusement
 Gustavus is a muff          | MPEG 1   | Amusement
(5 rows)

...
```

Klauzule limit oraz offset (przy rozkazie select):

Przykład 2:

```
select sid from subcategories order by sid limit 5;
select sid from subcategories order by sid limit 5 offset 5;
select sid from subcategories order by sid offset 10;
```

Uwagi:
1. do korzystania z tych klauzul nie jest konieczne otwarcie transakcji
2. przy wyświetlaniu partiami wyników zapytania należy koniecznie użyć klauzuli order by; jeżeli się jej nie użyje wyniki mogą być nieprzewidywalne (np. zobaczymy dwa razy ten sam wiersz)

Zadanie 2: Zrealizować zadanie 1 korzystając z klauzul limit/offset. Wyniki jak w zad 2.

Prawa dostępu

Uwagi:
1. Ćwiczenia wykonujemy w parach
2. Jedna osoba pełni rolę administratora bazy danych, na jej bazie danych wykonujemy ćwiczenia. (Administratorem bazy jest osoba, która ją założyła).
3. Druga osoba jest zwykłym użytkownikiem, zalogowanym do bazy.

Administrator bazy loguje się ze swojego terminala do bazy komendą:
`psql nazwa_bazy`

Użytkownik loguje się ze swojego terminala komendą:
`psql nazwa_bazy własny_login`

Lp	Administrator bazy	Użytkownik
1.	`select * from categories;`	`select * from categories;` *Zapytanie nie powiedzie się, ze względu na brak uprawnień do tej operacji.*
2.	`grant select on categories to public;` *Zezwala na wykonywanie komendy select wszystkim użytkownikom.*	
3.		`select * from categories;`
4.	`\z` *Wyświetla bieżące uprawnienia na obiektach (tabelach, widokach i sekwencjach)*	`\z`
5.	`revoke select on categories from public;` *Odwołanie prawa dostępu.*	
6.		`insert into categories (cid, name, cserid) values (50, 'test', 1);` *Komenda się nie wykona, brak uprawnień.*
7.	***Zadanie:*** *jednym poleceniem nadać prawo do wykonywania insert oraz delete na tabeli categories dla Użytkownika, nie dla wszystkich.*	
8.		*przetestować nadane uprawnienia*
9.	***Zadanie:*** *Nadać komplet uprawnień dla Użytkownika dla tabeli categories*	
10.		*przetestować* `select, insert, update, delete`
11.	***Zadanie:*** *Odwołać prawo do update, delete i select dla tabeli categories dla Użytkownika*	

Zadanie 3: Należy umożliwić Użytkownikowi dodawanie nowych wierszy do tabeli `categories`, oraz wyświetlanie tylko kolumn `cid` oraz `name`. Użytkownik nie może mieć prawa odczytu do kolumny `cserid`!

PL / pgSQL (Opis języka: PostgreSQL Programmer's Guide – PL/pgSQL - SQL Procedural Language)

Przykład 3: Prosta funkcja dodająca 1 do argumentu

```
create function add_one (int4) returns int4 as '
BEGIN
-- To jest komentarz
        return $1+1;
END;
' LANGUAGE 'plpgsql';

select add_one(9);
drop function add_one(int4);
```

Przykład 4: Funkcja, która do tablicy wpisuje wiersze w podanym zakresie `id`.

```
create table test (id int4, name text);

create function fill (int4, int4) returns bool as '
declare
        i int4;
begin
        for i in $1 .. $2 loop
          insert into test (id, name) values (i, NULL);
          raise notice ''INSERT  % '', i;
        end loop;
        return true;
end;
' language 'plpgsql';

select fill(1, 20);
select * from test;
drop function fill(int4, int4);
```

Przykład 5: Funkcja zwraca liczbę wierszy, dla których `name` jest różne od `NULL`, ponadto wyświetla ostrzeżenie dla każdego wiersza, gdzie `name = NULL`.

```
create function x_1 () returns int4 as '
declare
        r test%rowtype;
        c int4 default 0;
begin
        for r in select id, name from test loop
        if r.name isnull then
                raise notice ''name for id % is null'', r.id;
        else
                c=c+1;
        end if;
        end loop;
        return c;
end;
' language 'plpgsql';

select x_1();
update test set name='xyz' where id=7;
select x_1();
drop function x_1();
```

Przykład 6: Funkcja zwraca `id` pierwszego elementu z posortowanej listy, dla którego `name` jest różne od `NULL`.

```
create function x_3 () returns int4 as '
declare
        r test%rowtype;
begin
       for r in select id, name from test order by id loop
              EXIT  when r.name is not null;
       end loop;
       return r.id;
end;
' language 'plpgsql';

select x_3();

drop function x_3();
```

Przykład 7: Trigger, który po każdym `insert` wyświetla sumę stanów kont.

```
create table konta (id serial, konto int4 default 0);

create function x_2 () returns trigger as '
declare
        c int4;
begin
        select sum(konto) into c from konta;
        raise notice ''Suma kont wynosi %'', c;
        return NEW;
end;
' language 'plpgsql';

create trigger x_2t after insert on konta for each
row execute procedure x_2();

insert into konta (konto) values (10);
insert into konta (konto) values (14);
```

Przykład 8: Trigger, który decyduje o tym, czy `INSERT` będzie wykonany czy też nie:

```
create function x_4 () returns trigger as '
declare
         c int4;
begin
         c=(select max(konto)+2 from konta);
         IF NEW.konto >= c THEN
           return NEW;
         ELSE
          raise notice ''Wartosc pola konta musi byc wieksza o co najmniej 2 od maksymalnej
wartosci w kolumnie konta w tabeli, czyli musi miec wartosc przynajmniej %'',c;
        return NULL;
        END IF;
end;
' language 'plpgsql';

create trigger x_4t before insert on konta for each
row execute procedure x_4();

insert into konta (konto) values (11);
insert into konta (konto) values (16);

drop trigger x_4t on konta;
```

Przykład 9: Trigger, który modyfikuje wprowadzane dane przed aktualizacją wiersza (zamiast zmieniać starą wartość na nową dodaje do nową do starej).

```
create function x_5 () returns trigger as '
begin
       NEW.konto=NEW.konto+OLD.konto;
       return NEW;
end;
' language 'plpgsql';

create trigger x_5t before update on konta for each
row execute procedure x_5();

select * from konta;
update konta set konto=1000;
select * from konta;

drop trigger x_5t on konta;
```

Zadanie 4: Napisać funkcję silnia przyjmującą jako argument wartość `int4` i zwracająca `int4`.

Zadanie 5: Napisać funkcję, która liczy wartość średnią z kolumny `konta.konto`.

Zadanie 6: Zmodyfikować trigger `x_2t`, tak, aby działał również po każdym `insert`, `update` oraz `delete`.

Zadanie 7: Napisać trigger, który zapewni unikalność wpisów w kolumnie `konto` w tabeli `konta`.

WAŻNE:

- `return WARTOŚĆ` – powoduje wykonanie INSERT/UPDATE/DELETE ze wskazaną wartością
- `return NULL` – powoduje że INSERT/UPDATE/DELETE w triggerach "before" nie zostanie wykonany, ale transakcja będzie wykonywana dalej
- `raise exception '' komunikat''` – powoduje zerwanie transakcji ze wskazanym komunikatem o błędzie

Laboratorium 9

- **Ograniczenia dla tabeli**
- **Klucz główny**
- **Klucz obcy**

Wymagany materiał z wykładu:

- **Rozdział 10 – Pojęcie klucza i inne warunki (constraints)**

Ograniczenia dla tabeli

Przykład 1: Utworzenie tabeli z ograniczeniami. Ograniczenia można nakładać na poszczególne kolumny lub na całą tabelę. Poniższe polecenie tworzy tabelę złożoną z czterech kolumn. Wartość w kolumnie nr powinna być unikalna, domyślną wartością w kolumnie imie ma być '?', w kolumnie nazw nie może być pustego pola. Ponadto unikalną wartość musi mieć para imie, nazw. Wartość nr musi być mniejsza od 5.

```
CREATE TABLE osoby (nr int4 UNIQUE, wydzial varchar(10), imie varchar(10) DEFAULT '?', nazw
varchar(10) NOT NULL, UNIQUE (imie, nazw), CHECK (nr < 5));
NOTICE:  CREATE TABLE/UNIQUE will create implicit index 'osoby_nr_key' for table 'osoby'
NOTICE:  CREATE TABLE/UNIQUE will create implicit index 'osoby_imie_key' for table 'osoby'
```

Automatycznie utworzone zostały odpowiednie indeksy, które zapewnią spełnienie ograniczeń unikalności. Nazwy indeksów `osoby_nr_key` oraz `osoby_imie_key` są zarazem nazwami utworzonych ograniczeń. Ograniczenie odpowiadające za sprawdzanie czy `nr` ma wartość większą od 5 nazywa się `$1`.

```
\d osoby
 Column  |         Type          |  Modifiers
---------+-----------------------+-------------
 nr      | integer               |
 wydzial | character varying(10) |
 imie    | character varying(10) | default '?'
 nazw    | character varying(10) | not null
Indexes: osoby_imie_key unique btree (imie, nazw),
         osoby_nr_key unique btree (nr)
Check constraints: "$1" (nr < 5)
```

```
INSERT INTO osoby VALUES(1,'KT','Jan','Kowal');
INSERT INTO osoby VALUES(1,'KT','Anna','Jopek');
ERROR:  Cannot insert a duplicate key into unique index osoby_nr_key
```

Nie można wstawić drugiego wiersza zawierającego nr =1.

```
INSERT INTO osoby VALUES(2,'KT','Anna','Kowal');
INSERT INTO osoby VALUES(3,'KT','Jan','Kowal');
ERROR:  Cannot insert a duplicate key into unique index osoby_imie_key
```

Nie można wstawić drugiego wiersza zawierającego `imie='Jan'` i `nazw='Kowal'`.

```
INSERT INTO osoby VALUES(6,'KT','Adam','Malysz');
ERROR:  ExecInsert: rejected due to CHECK constraint "$1" on "osoby"
```

Nie można wstawić wartości nr większej od 5.

```
INSERT INTO osoby (nr, wydzial, imie) VALUES(4,'KT','Tomasz');
ERROR:  ExecAppend: Fail to add null value in not null attribute nazw
```

Wartość `nazw` nie może być NULL.

```
INSERT INTO osoby (nr, wydzial, nazw) VALUES(4,'KT','Pol');
```

Domyślnie zostanie ustawione `imie` na wartość '?'.

```
SELECT * FROM osoby;
 nr | wydzial | imie | nazw
----+---------+------+-------
  1 | KT      | Jan  | Kowal
  2 | KT      | Anna | Kowal
  4 | KT      | ?    | Pol
(3 rows)

DROP TABLE osoby;
```

Klucz główny

Przykład 2: Utworzenie tabeli z kluczem głównym.

```
CREATE TABLE wydz (nazwa varchar(10) PRIMARY KEY, adres varchar(10));
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index 'wydz_pkey' for table 'wydz'
```

Utworzenie klucza głównego powoduje automatyczne utworzenie indeksu na kolumnie, która jest kluczem głównym.

```
INSERT INTO wydz VALUES('EAiE','B-1');
INSERT INTO wydz VALUES('IMIR','B-3');
INSERT INTO wydz VALUES('KT','D-5');
INSERT INTO wydz VALUES('EAiE','B-16');
ERROR:  Cannot insert a duplicate key into unique index wydz_pkey
```

Ostatni `insert` zakończył się błędem, gdyż kolumna `nazwa` zawiera już wartość 'EAiE'.

Klucz obcy

Przykład 3: Utworzenie tabeli z kluczem obcym.

```
CREATE TABLE osoby (nr int4, wydzial varchar(10), imie varchar(10), nazw varchar(10), FOREIGN
KEY (wydzial) REFERENCES wydz(nazwa));
NOTICE:  CREATE TABLE will create implicit trigger(s) for FOREIGN KEY check(s)
```

Automatycznie utworzony został trigger odpowiedzialny za sprawdzenie wprowadzanej wartości z kluczem obcym.

Taki sam efekt można osiągnąć następującymi poleceniami:

```
CREATE TABLE osoby (nr int4, wydzial varchar(10) REFERENCES wydz(nazwa),imie
varchar(10), nazw varchar(10));
CREATE TABLE osoby (nr int4, wydzial varchar(10), imie varchar(10), nazw varchar(10),
CONSTRAINT ogran_1 FOREIGN KEY (wydzial) REFERENCES wydz(nazwa));
```

Polecenia te są równoważne. Jedyną różnicą jest to, że w ostatnim przypadku nadajemy ograniczeniu nazwę `ogran_1`, zaś w pozostałych nadawana jest nazwa domyślna `$1`.

```
INSERT INTO osoby VALUES(1,'KT','Tom','Hanks');
INSERT INTO osoby VALUES(2,'MECH','John','Brown');
ERROR:  $1 referential integrity violation - key referenced from osoby not found in wydz
```

Drugi insert spowodował błąd gdyż klucz obcy (kolumna `nazwa` w tabeli `wydz`) nie zawiera wartości 'MECH'.

```
SELECT * FROM osoby;
 nr | wydzial | imie | nazw
----+---------+------+-------
  1 | KT      | Tom  | Hanks
(1 row)

INSERT INTO osoby VALUES(2,'IMIR','Janko','Muzykant');

SELECT * FROM osoby;
 nr | wydzial | imie  |   nazw
----+---------+-------+----------
  1 | KT      | Tom   | Hanks
  2 | IMIR    | Janko | Muzykant
(2 rows)

DELETE FROM wydz WHERE nazwa='IMIR';
ERROR:  $1 referential integrity violation - key in wydz still referenced from osoby
```

Błąd, gdyż skasowanie wiersza spowodowałoby utratę spójności bazy (w tabeli osoby istnieją wiersze zawierające `wydzial='IMIR'`).

```
UPDATE wydz SET nazwa='IMiR' WHERE nazwa='IMIR';
ERROR:  $1 referential integrity violation - key in wydz still referenced from osoby
UPDATE osoby SET wydzial='IMiR' WHERE wydzial='IMIR';
ERROR:  $1 referential integrity violation - key referenced from osoby not found in wydz
```

Powyższe dwa polecenia również spowodowałyby utratę spójności bazy.

```
UPDATE wydz SET nazwa='EAIiE' WHERE nazwa='EAiE';
UPDATE wydz SET adres='B-2' WHERE nazwa='IMIR';
```

Te polecenia zakończyły się sukcesem, gdyż nie powodują utraty spójności bazy (w tabeli osoby nie ma wiersza zawierającego `wydzial='EAiE'` więc można dokonać zmiany nazwy w tabeli `wydz`).

```
DROP TABLE osoby;
```

Utworzone automatycznie triggery są usuwane również automatycznie.

Przykład 4: Utworzenie tabeli z kluczem obcym w sposób umożliwiający modyfikowanie kolumn powiązanych kluczem obcym. Podejście 1.

```
CREATE TABLE osoby (nr int4, wydzial varchar(10) REFERENCES wydz(nazwa) DEFERRABLE, imie
varchar(10), nazw varchar(10));
```

Ustawienie atrybutu `DEFERRABLE` sprawia, że możliwe będzie uruchomienie trybu odroczenia sprawdzania ograniczeń. Będą one sprawdzane dopiero w momencie zakończenia transakcji (wydanie polecenia commit).

```
INSERT INTO osoby VALUES(1,'KT','Tom','Hanks');
INSERT INTO osoby VALUES(2,'IMIR','Janko','Muzykant');
SELECT * FROM wydz;
 nazwa | adres
-------+-------
 EAIiE | B-1
 KT    | D-5
 IMIR  | B-2
(3 rows)

SELECT * FROM osoby;
 nr | wydzial | imie  |   nazw
----+---------+-------+----------
  1 | KT      | Tom   | Hanks
  2 | IMIR    | Janko | Muzykant
(2 rows)

BEGIN;
SET CONSTRAINTS ALL DEFERRED;
```

Tu nastąpiło uruchomienie trybu odroczenia sprawdzania ograniczeń. W przypadku nie włączenia tego trybu ograniczenia byłyby sprawdzane na bieżąco i oba poniższe polecenia spowodowałyby błąd.

```
UPDATE wydz SET nazwa='IMiR' WHERE nazwa='IMIR';
UPDATE osoby SET wydzial='IMiR' WHERE wydzial='IMIR';
```

Oba te polecenia wykonały się gdyż nie sprawdzano ograniczeń. Gdyby wykonano tylko jednego z powyższych update-ów nastąpiłoby naruszenie spójności bazy i commit zwróciłby błąd.

```
COMMIT;

SELECT * FROM wydz;
 nazwa | adres
-------+-------
 EAIiE | B-1
 KT    | D-5
 IMiR  | B-2
(3 rows)

SELECT * FROM osoby;
 nr | wydzial | imie  |   nazw
----+---------+-------+----------
  1 | KT      | Tom   | Hanks
  2 | IMiR    | Janko | Muzykant
(2 rows)

DROP TABLE osoby;
UPDATE wydz SET nazwa='IMIR' WHERE nazwa='IMiR';
```

Przykład 5: Utworzenie tabeli z kluczem obcym w sposób umożliwiający modyfikowanie kolumn powiązanych kluczem obcym. Podejście 2.

```
CREATE TABLE osoby (nr int4, wydzial varchar(10) REFERENCES wydz(nazwa) INITIALLY DEFERRED,
imie varchar(10), nazw varchar(10));
```

Jeżeli w momencie tworzenia tabeli ustawimy dla tworzonego klucza obcego `INITIALLY DEFERRED` wówczas trybu odroczenia sprawdzania ograniczeń zostanie włączony na stałe. Nie ma wówczas potrzeby włączania tego trybu na początku transakcji.

```
INSERT INTO osoby VALUES(1,'KT','Tom','Hanks');
INSERT INTO osoby VALUES(2,'IMIR','Janko','Muzykant');
SELECT * FROM wydz;
SELECT * FROM osoby;
```

```
BEGIN;
UPDATE wydz SET nazwa='IMiR' WHERE nazwa='IMIR';
UPDATE osoby SET wydzial='IMiR' WHERE wydzial='IMIR';
COMMIT;

SELECT * FROM wydz;
 nazwa | adres
-------+-------
 EAIiE | B-1
 KT    | D-5
 IMiR  | B-2
(3 rows)

SELECT * FROM osoby;
 nr | wydzial | imie  |   nazw
----+---------+-------+----------
  1 | KT      | Tom   | Hanks
  2 | IMiR    | Janko | Muzykant
(2 rows)

DROP TABLE osoby;
UPDATE wydz SET nazwa='IMIR' WHERE nazwa='IMiR';
```

Przykład 6: Utworzenie tabeli z kluczem obcym w sposób umożliwiający modyfikowanie kolumn powiązanych kluczem obcym. Podejście 3.

```
CREATE TABLE osoby (nr int4, wydzial varchar(10), imie varchar(10), nazw varchar(10), FOREIGN
KEY (wydzial) REFERENCES wydz(nazwa) ON UPDATE CASCADE);
```

Tworząc klucz obcy w tabeli `osoby` zdefiniowano akcję, jaka powinna być wykonana,jeżeli nastąpi uaktualnienie (update) tabeli `wydz`. Akcja `CASCADE` oznacza, że zmiana wartości w kolumnie `nazwa` w tabeli `wydz` ma spowodować automatyczną zmianę w kolumnie `wydzial` w tabeli `osoby`. Dzięki temu spójność bazy jest zapewniona automatycznie.

```
INSERT INTO osoby VALUES(1,'KT','Tom','Hanks');
INSERT INTO osoby VALUES(2,'IMIR','Janko','Muzykant');
SELECT * FROM wydz;
SELECT * FROM osoby;

UPDATE wydz SET nazwa='IMiR' WHERE nazwa='IMIR';
```

Jednocześnie nastąpi aktualizacja tabeli `osoby`. W kolumnie `wydzial` wszystkie wartości 'IMIR' zostaną automatycznie zamienione na 'IMiR'.

```
SELECT * from wydz;
 nazwa | adres
-------+-------
 EAIiE | B-1
 KT    | D-5
 IMiR  | B-2
(3 rows)

SELECT * FROM osoby;
 nr | wydzial | imie  | nazw
----+---------+-------+-------
  1 | KT      | Tom   | Hanks
  2 | IMiR    | Janko | Muzykant
(2 rows)

DROP TABLE osoby;
UPDATE wydz SET nazwa='IMIR' WHERE nazwa='IMiR';
```

Inne akcje, które mogą być ustawione jako `ON UPDATE`:

- NO ACTION lub RESTRICT - ustawienie domyślne, generuje błąd naruszenia spójności bazy (jak w przykładzie 3)
- SET NULL - ustawienie wartości NULL
- SET DEFAULT - ustawienie wartości domyślnej

Przykład 7: Przykład analogiczny do poprzedniego. Dodatkowo ustawiono domyślną akcję, która powinna być wykonana gdy z tabeli `wydz` zostanie usunięty wiersz, do którego jest odniesienie z tabeli `osoby`.

```
CREATE TABLE osoby (nr int4, wydzial varchar(10), imie varchar(10), nazw varchar(10), FOREIGN
KEY (wydzial) REFERENCES wydz(nazwa) ON UPDATE CASCADE ON DELETE SET NULL);

INSERT INTO osoby VALUES(1,'KT','Tom','Hanks');
INSERT INTO osoby VALUES(2,'IMIR','Janko','Muzykant');

DELETE FROM wydz WHERE nazwa='IMIR';
SELECT * FROM wydz;

 nazwa | adres
-------+-------
 EAIiE | B-1
 KT    | D-5
(2 rows)

SELECT * FROM osoby;
 nr | wydzial | imie  |   nazw
----+---------+-------+----------
  1 | KT      | Tom   | Hanks
  2 |         | Janko | Muzykant
(2 rows)

DROP TABLE osoby;
INSERT INTO wydz VALUES('IMIR','B-2');
```

Uwaga! `ON DELETE` może być również ustawiony na `CASCADE`. W takiej sytuacji skasowane zostaną również wszystkie wiersze z tabeli, w których jest odniesienie do kasowanego wiersza.

Uwaga do zadań 1–3: W zadaniach chodzi o poprawienie istniejącej konstrukcji bazy danych z filmami tak, aby wymuszała określone związki pomiędzy danymi oraz określone wartości. Dla celów ćwiczenia zostały przygotowane cztery pliki, znajdujące się w katalogu '~rstankie/db'. Są to: m.dump, c.dump oraz s.dump. Pliki te zawierają komendy SQL tworzące tabele movies_list, categories, subcategories (odpowiednio) oraz wpisujące do nich dane. Dla uproszczenia pracy każdy z tych plików zawiera na początku komendę powodującą usunięcie odpowiedniej tabeli jeżeli taka istnieje.

Aby wykonać zadania 1–3 należy:
1. Skopiować te pliki do swojego katalogu
2. Zmodyfikować odpowiednio w tych plikach komendę SQL tworzącą tabelę (dowolnym edytorem)
3. Załadować z poziomu psql komendy z plików poleceniem (przykładowo):
\i c.dump

Dla zweryfikowania zadań 1–3 należy wykonać testy znajdujące się w pliku ~rstankie/db/testy.txt. Podano wyniki jakie należy uzyskać.

Zadanie 1: Utworzyć tabelę categories tak, aby kolumna cid była kluczem głównym, zaś kolumna name miała unikalne i niepuste wartości.

Zadanie 2: Utworzyć tabelę subcategories tak, aby sid było kluczem głównym, kolumna cid była kluczem obcym związanym z categories.cid. Kolumna cid w tabeli subcategories ma być automatycznie aktualizaowana w przypadku zmiany cid w tabeli categories. Należy także zapewnić, by dana nazwa podkategorii nie występowała dwa razy w tej samej kategorii (w różnych kategoriach może wystąpić taka sama nazwa podkategorii). Skasowanie kategorii powinno spowodować automatyczne skasowanie podkategorii.

Zadanie 3: Utworzyć tabelę movies_list, w której mid jest kluczem głównym, a pola cid i sid zawierają poprawne wartości identyfikatorów kategorii i podkategorii, przy czym należy zapewnić, że wskazana podkategoria należy do wskazanej kategorii. Domyślną wartością dla tych pól ma być 1. W przypadku usunięcia kategorii bądź podkategorii w polach cid i sid mają być automatycznie wstawiane wartości domyślne. W przypadku aktualizacji wartości identyfikatorów kategorii lub podkategorii w tabelach categories oraz subcategories wartości pól cid oraz sid w movies_list mają być automatycznie aktualizowane.

Zadanie 4: Utworzyć tabelę osoby zawierającą nazwisko, imię i nr. pesel. Nazwisko i imię nie mogą być puste. PESEL musi być poprawny. Algorytm sprawdzania poprawności nr. PESEL znajduje się na stronie: http://wipos.p.lodz.pl/zylla/ut/pesel.html

Laboratorium 10

- **Przykładowa baza danych z zastosowaniem triggerów do automatycznej aktualizacji kilku tabel**

Wymagany materiał z wykładu:

- **Rozdział 15 – Implementacje dziedziczenia**
- **Rozdział 16 – Teoria relacyjnych baz danych**

Przykładowa baza danych z zastosowaniem triggerów do automatycznej aktualizacji kilku tabel (`~rstankie/db/sklep_n.sql`).

OPIS TABEL

```
Table "klienci"                 Table "produkty"
 Attribute |    Type             Attribute |    Type
-----------+-------------       -----------+-------------
 k_id      | integer             prod_id   | integer
 imie      | varchar(10)         nazwa     | varchar(40)
 nazwisko  | varchar(10)         cena      | integer
                                 stan      | integer

Table "zamow_szczegoly"         Table "zamow_lista"
 Attribute |  Type               Attribute |  Type
-----------+---------           -----------+---------
 zs_id     | integer             zam_id    | integer
 zam_id    | integer             k_id      | integer
 prod_id   | integer             wartosc   | integer
 liczba    | integer
```

W tabeli **`klienci`** przechowywana jest lista klientów sklepu: unikalny identyfikator klienta `k_id`, imie w kolumnie `imie` i nazwisko w kolumnie `nazwisko`.
W tabeli **`produkty`** przechowywana jest lista dostępnych produktów: unikalny identyfikator produktu `prod_id`, nazwa produktu `nazwa`, cena jednostkowa `cena` oraz liczba sztuk danego produktu będąca na stanie sklepu `stan`.
W tabeli **`zamow_lista`** przechowywana jest lista zamówień. Zamówienie jest opisywane przez unikalny identyfikator `zam_id`, identyfikator klienta, który dokonał zamówienie `k_id` oraz wartość zamówienia `wartosc`. Klient o podanym identyfikatorze musi istnieć (klucz obcy: `REFERENCES klienci(k_id)`).
W tabeli **`zamow_szczegoly`** przechowywane są szczegóły dotyczące poszczególnych zamówień. Każdy „szczegół" ma swój unikalny identyfikator `zs_id`, składa się z dokładnie jednego produktu `prod_id` wybranego z listy produktów (klucz obcy: `REFERENCES produkty(prod_id)`). Każdy „szczegół" należy do konkretnego zamówienia `zam_id`. Zamówienie musi istnieć (klucz obcy: `REFERENCES zamow_lista(zam_id)`). Dla każdego produktu można liczbę zamawianych sztuk - kolumna `liczba`.

OPIS TRIGGERÓW I FUNKCJI

Trigger **`t_spr_stan_przed_insert`** jest wykonywany przed wykonaniem każdego polecenia `insert` do tabeli `zamow_szczegoly` w celu sprawdzenia czy żądana liczba sztuk danego produktu jest na stanie.

```
CREATE TRIGGER t_spr_stan_przed_insert BEFORE INSERT ON zamow_szczegoly FOR EACH ROW EXECUTE
PROCEDURE spr_stan_przed_insert();
```

Wywoływana jest funkcja sprawdzająca **`spr_stan_przed_insert()`**. `NEW` jest zmienną przechowującą nowy wiersz. W przypadku, gdy warunek nie jest spełniony na ekranie pojawia się komunikat (`RAISE NOTICE`) i zwracana jest wartość `NULL`. Zwracana wartość `NULL` zapewnia, że `insert` nie zostanie wykonany.

```
CREATE FUNCTION spr_stan_przed_insert() RETURNS TRIGGER AS '
BEGIN
       IF NEW.liczba > (SELECT stan FROM produkty WHERE prod_id=NEW.prod_id) THEN
              RAISE NOTICE ''Operacja nie zostala wykonana. Nie ma tyle sztuk na stanie'';
              RETURN NULL;
       ELSE
              RETURN NEW;
       END IF;
END;
' LANGUAGE 'plpgsql';
```

Trigger **`t_spr_stan_przed_update`** jest wykonywany przed wykonaniem każdego polecenia `update` na tabeli `zamow_szczegoly` w celu sprawdzenia czy żądana liczba sztuk danego produktu jest na stanie.

```
CREATE TRIGGER t_spr_stan_przed_update BEFORE UPDATE ON zamow_szczegoly FOR EACH ROW EXECUTE
PROCEDURE spr_stan_przed_update();
```

Wywoływana jest funkcja sprawdzająca **`spr_stan_przed_update()`**.

```
CREATE FUNCTION spr_stan_przed_update() RETURNS TRIGGER AS '
DECLARE
	przywroc int4;
BEGIN
	IF NEW.prod_id = OLD.prod_id THEN
		przywroc = OLD.liczba;
	ELSE
		przywroc = 0;
	END IF;
	IF NEW.liczba > (SELECT stan FROM produkty WHERE prod_id=NEW.prod_id) + przywroc THEN
		RAISE NOTICE ''Operacja nie zostala wykonana. Nie ma tyle sztuk na stanie'';
		RETURN NULL;
	ELSE
		RETURN NEW;
	END IF;
END;
' LANGUAGE 'plpgsql';
```

Po każdym `insert` do tabeli `zamow_szczegoly` następuje automatyczne uaktualnienie wartości zamówienia `wartosc` w tabeli `zamow_lista` (dodanie ceny produktu pomnożonej przez liczbę zamówionych sztuk) oraz zmniejszenie liczby sztuk na stanie (`stan` w tabeli `produkty`). Zapewnia to trigger **`aktual_po_insert`**.

```
CREATE TRIGGER aktual_po_insert AFTER INSERT ON zamow_szczegoly FOR EACH ROW EXECUTE PROCEDURE
aktualizuj_po_insert();
```

Wywołuje on funkcję **`aktualizuj_po_insert()`**.

```
CREATE FUNCTION aktualizuj_po_insert() RETURNS TRIGGER AS '
DECLARE
	cena int4;
	ident int4;
	produkt produkty.nazwa%TYPE;
BEGIN
	SELECT currval(''zamow_szczegoly_zs_id_seq'') INTO ident;
	SELECT produkty.cena INTO cena FROM produkty WHERE prod_id=NEW.prod_id;
	SELECT produkty.nazwa INTO produkt FROM produkty WHERE prod_id=NEW.prod_id;
	UPDATE zamow_lista SET wartosc = wartosc + cena * NEW.liczba WHERE
zamow_lista.zam_id=NEW.zam_id;
	UPDATE produkty SET stan = stan - NEW.liczba WHERE prod_id=NEW.prod_id;
	RAISE NOTICE ''Wybrano produkt: %, sztuk: %, cena: %. Wpisano na pozycji: %.'',
produkt, NEW.liczba, cena, ident;
	RETURN NULL;
END;
' LANGUAGE 'plpgsql';
```

Podobnie po każdej operacji `update` na tabeli `zamow_szczegoly` należy odpowiednio zaktualizować tabele `zamow_lista` i `produkty`. Zapewnia to trigger **`aktual_po_update`** wywołujący funkcję **`aktualizuj_po_update()`**.

```
CREATE TRIGGER aktual_po_update AFTER UPDATE ON zamow_szczegoly FOR EACH ROW EXECUTE PROCEDURE
aktualizuj_po_update();

CREATE FUNCTION aktualizuj_po_update() RETURNS TRIGGER AS '
DECLARE
        cena int4;
        produkt_old produkty.nazwa%TYPE;
        produkt_new produkty.nazwa%TYPE;

BEGIN
        SELECT produkty.cena INTO cena FROM produkty WHERE prod_id=OLD.prod_id;
        UPDATE zamow_lista SET wartosc = wartosc - cena * OLD.liczba WHERE
zamow_lista.zam_id=OLD.zam_id;
        SELECT produkty.cena INTO cena FROM produkty WHERE prod_id=NEW.prod_id;
        UPDATE zamow_lista SET wartosc = wartosc + cena * NEW.liczba WHERE
zamow_lista.zam_id=NEW.zam_id;

        UPDATE produkty SET stan = stan + OLD.liczba WHERE prod_id=OLD.prod_id;
        UPDATE produkty SET stan = stan - NEW.liczba WHERE prod_id=NEW.prod_id;



        IF NEW.prod_id <> OLD.prod_id THEN
                SELECT produkty.nazwa INTO produkt_new FROM produkty WHERE prod_id=NEW.prod_id;
                SELECT produkty.nazwa INTO produkt_old FROM produkty WHERE prod_id=OLD.prod_id;
                RAISE NOTICE  ''Zrezygnowano z produktu: %. Wybrano produkt: %, sztuk: %, cena:
%.'', produkt_old, produkt_new, NEW.liczba, cena;
        END IF;

        RETURN NEW;
END;
' LANGUAGE 'plpgsql';
```

Konieczne jest również zapewnienie spójności bazy w przypadku skasowania wierszy z tabeli zamow_szczegoly. Jeżeli jakiś wiersz jest kasowany to produkt musi zostać przywrócony na stan, zaś wartość zamówienia musi być pomniejszona. Zapewnia to trigger **`aktual_przed_delete`** wywołujący funkcję **`aktualizuj_przed_delete()`**.

```
CREATE TRIGGER aktual_przed_delete BEFORE DELETE ON zamow_szczegoly FOR EACH ROW EXECUTE
PROCEDURE aktualizuj_przed_delete();

CREATE FUNCTION aktualizuj_przed_delete() RETURNS TRIGGER AS '
DECLARE
        cena int4;
BEGIN
        SELECT produkty.cena INTO cena FROM produkty WHERE prod_id=OLD.prod_id;
        UPDATE zamow_lista SET wartosc = wartosc - cena * OLD.liczba WHERE
zamow_lista.zam_id=OLD.zam_id;
        UPDATE produkty SET stan = stan + OLD.liczba WHERE prod_id=OLD.prod_id;
        RETURN OLD;
END;
' LANGUAGE 'plpgsql';
```

Należy zauważyć, że w przypadku skasowania całego zamówienia (skasowanie w tabeli `zamow_lista`) spowoduje skasowanie wszystkich wierszy związanych z tym zamówieniem z tabeli `zamow_szczegoly` (zapewnia to klauzula `ON DELETE CASCADE` -patrz polecenie tworzące tę tabelę). W momencie automatycznego kasowania tych wierszy również będzie wykonywany trigger `aktual_przed_delete` i wszystkie produkty z tego zamówienia będą przywrócone na stan sklepu.

ĆWICZENIA

1. Utworzyć nową bazę danych.
```
createdb username_sklep
```

2. Wejść do bazy i załadować zawartość bazy z pliku
```
psql username_sklep
\i /export/home/staff/rstankie/db/sklep_n.sql
```

3. Obejrzeć bazę danych.
```
\d
select * from klienci;
 k_id |  imie  | nazwisko
------+--------+----------
    1 | Adam   | Malysz
    2 | Tomasz | Klos
    3 | Dziad  | Borowy
    4 | Mirek  | Kowal
(4 rows)

select * from produkty;
 prod_id |          nazwa           | cena | stan
---------+--------------------------+------+------
       1 | Pentium III 600MHz       | 1200 |    2
       2 | Dysk twardy IBM 10GB     |  400 |    5
       3 | Dysk twardy Seagate 15GB |  550 |    5
       4 | Pentium II 450MHz        |  300 |    1
       5 | Stacja dyskow 1,44       |   30 |    2
       6 | obudowa                  |  125 |    2
       7 | mysz logitech scroll     |   70 |    2
       8 | Karta muzyczna RIVA TNT  |  200 |    2
       9 | klawiatura               |   50 |    2
(9 rows)

select * from zamow_lista;
 zam_id | k_id | wartosc
--------+------+---------
(0 rows)

select * from zamow_szczegoly;
 zs_id | zam_id | prod_id | liczba
-------+--------+---------+--------
(0 rows)
```

4. Utworzyć dwa zamówienia: dla klienta o k_id=2 i k_id=3
```
insert into zamow_lista (k_id) values (2);
insert into zamow_lista (k_id) values (3);
select * from zamow_lista;
 zam_id | k_id | wartosc
--------+------+---------
      1 |    2 |       0
      2 |    3 |       0
```

Wartości zamówień są równe 0.

5. Dla zamówienia o zam_id=1 wybrać procesor (prod_id=1) - 1 szt. oraz dysk twardy (prod_id=2) 2 szt.
```
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (1, 1, 1);
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (1, 2, 2);
```

6. Sprawdzić wartość zamówienia oraz liczbę sztuk na stanie.

```
select * from zamow_lista;
 zam_id | k_id | wartosc
--------+------+---------
      2 |    3 |       0
      1 |    2 |    2000
```

Na kwotę 2000 składa się 1*1200 za procesor + 2*400 za dysk twardy.

```
select * from produkty;
 prod_id |          nazwa           | cena | stan
---------+--------------------------+------+------
       3 | Dysk twardy Seagate 15GB |  550 |    5
       4 | Pentium II 450MHz        |  300 |    1
       5 | Stacja dyskow 1,44       |   30 |    2
       6 | obudowa                  |  125 |    2
       7 | mysz logitech scroll     |   70 |    2
       8 | Karta muzyczna RIVA TNT  |  200 |    2
       9 | klawiatura               |   50 |    2
       1 | Pentium III 600MHz       | 1200 |    1
       2 | Dysk twardy IBM 10GB     |  400 |    3
```

Przedtem liczba procesorów (prod_id=1) była 2, zaś liczba dysków twardych (prod_id=2) była 5.

7. Zmienić liczbę dysków twardych na 1.

```
update zamow_szczegoly set liczba=1 where zs_id=2;
select * from zamow_lista;
 zam_id | k_id | wartosc
--------+------+---------
      2 |    3 |       0
      1 |    2 |    1600

select * from zamow_szczegoly;
 zs_id | zam_id | prod_id | liczba
-------+--------+---------+--------
     1 |      1 |       1 |      1
     2 |      1 |       2 |      1

select * from produkty;
 prod_id |          nazwa           | cena | stan
---------+--------------------------+------+------
       3 | Dysk twardy Seagate 15GB |  550 |    5
       4 | Pentium II 450MHz        |  300 |    1
       5 | Stacja dyskow 1,44       |   30 |    2
       6 | obudowa                  |  125 |    2
       7 | mysz logitech scroll     |   70 |    2
       8 | Karta muzyczna RIVA TNT  |  200 |    2
       9 | klawiatura               |   50 |    2
       1 | Pentium III 600MHz       | 1200 |    1
       2 | Dysk twardy IBM 10GB     |  400 |    4
```

8. Wykonać następujące polecenia i po każdym z nich obserwować zmiany w tabelach zamow_szczegoly, zamow_lista i produkty.

```
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (1, 7, 3);
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (1, 7, 1);
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (1, 3, 2);
update zamow_szczegoly set liczba=6 where zs_id=5;
update zamow_szczegoly set liczba=5 where zs_id=5;
update zamow_szczegoly set prod_id=9, liczba=1 where zs_id=4;
delete from zamow_szczegoly where zs_id=1;
insert into zamow_szczegoly (zam_id, prod_id, liczba) values (2, 6, 1);
delete from zamow_lista where zam_id=1;
delete from zamow_szczegoly where zs_id=6;
```

Po tej ostatniej operacji stan sklepu (w tabeli produkty) powinien być dokładnie taki sam jak na początku (patrz p.3).

Przetestuj inne, dowolne operacje na sklepie. Spróbuj doprowadzić do niespójności bazy.

Laboratorium 11
i następne

- **Zadanie zaliczeniowe - indywidualne projekty baz danych**
- **Materiały dodatkowe - przestrzenie nazw (schema)**

Wymagany materiał z wykładu:

- **Rozdział 15 – Implementacje dziedziczenia**
- **Rozdział 16 – Teoria relacyjnych baz danych**

Schema

W bazie danych funkcjonuje pojęcie „przestrzeni nazw" – namespace, określane jako SCHEMA. Domyślnie istnieje przestrzeń nazzw „public" w której są tworzone wszystkie obiekty bazy (tabele, widoki, etc.). Możliwe jest utworzenie własnych przestrzeni nazw w bazie danych - każda przestrzeń musi mieć unikalną nazwę w obrębie bazy danych. Poszczególne obiekty bazy (tabele, widoki) muszą mieć nazwy unikalne tylko w obrębie jednej przestrzeni nazw, co ułatwia pisanie baz danych przez grupy programistów.

Dokumentacja:
`Postgres User's Guide -> Data Definition -> Schemas`

Dostęp do obiektów w bazie korzystającej z przestrzeni nazw jest możliwy na dwa sposoby:
- przez podanie pełnej nazwy obiektu postaci nazwa_schema.nazwa_obiektu, np. `select public.categories.nazwa from public.categories;`
- podając samą nazwę obiektu (`select nazwa from categories;`) i polegając na ścieżce przeszukiwania przestrzeni nazw

Wyświetlenie ścieżki przeszukiwania przestrzeni nazw:
```
SHOW search_path;

 search_path
--------------
 $user,public
(1 row)
```

Nazwa `$user` oznacza nazwę bieżącego użytkownika. Jeżeli przestrzeń o takiej nazwie nie istnieje, wpis jest ignorowany.

Utworzenie przestrzeni nazw:
```
create schema xxx;
```

Ustawienie nowej ścieżki przeszukiwania przestrzeni nazw:
```
SET search_path TO xxx,public;
```
Przeszukiwane będą (w kolejności) schema xxx i schema public.

Wyświetlenie dostępnych przestrzeni nazw (tylko Postgres):
```
select * from pg_namespace;
select nspname, usename, nspacl from pg_namespace, pg_user where nspowner=usesysid;
```

Usunięcie przestrzeni nazw:
```
drop schema xxx;
```

Przykład 1:
Utwórz przestrzeń nazw „video"
```
create schema video;
```

Ustaw ścieżkę przeszukiwania przestrzeni nazw na: video,public
```
SET search_path TO video,public;
```

Utwórz tabelę `categories` w przestrzeni nazw `video` (na podstawie tabeli `categories` w schema `public`), wyświetl listę tabel przed i po utworzeniu tabeli:
```
\d
create table categories as select * from categories;
\d
```

W momencie wywołania tabela `video.categories` nie istnieje, więc zostanie użyta `public.categories`. Utworzona zostanie tabela `categories` w pierwszej napotkanej przestrzeni nazw, czyli `video.categories`. Tabela ta zasłania tabelę `public.categories`.

Usuń wszystkie rekordy w tabeli `categories` w przestrzeni nazw `public`:
```
delete from public.categories;
```

Wyświetl rekordy w tabelach `categories`, `public.categories`, `video.categories`.
```
select * from categories;
select * from public.categories;
select * from video.categories;
```

Struktura bazy danych używanej w laboratorium

Załadowanie danych potrzebnych do ćwiczeń (z linii poleceń UNIX'a):

```
dropdb nazwa_bazy
createdb
psql -f ~rstankie/db/stud.dump
```

Table:	***movies_list***		
Field name	**Type (SQL)**	**Length**	**Description**
mid	int4	4	Movie id
sysflags	int4	4	Status of the film (Available, Unavailable, etc.)
lenbin	int4	4	Movie length in bytes
stream	int4	4	Bit rate of the video strem [bps]
compress	varchar	16	Encoding (MPEG-I, MPEG-II, etc.)
dimensions	varchar	16	Screen size in pixels
fps	int4	4	Number of frames per second
cid	int4	4	Category Identifier (from table: ***categories***)
sid	int4	4	Subcategory Identifier (from table: ***subcategories***)
author	varchar	256	Name of the author
lenmsec	int4	4	Movie length in milliseconds
filename	varchar	256	Name of the disk file containing the movie (OVS filename)
tokenrate	int4	4	RSVP parameter
bucketsize	int4	4	RSVP parameter
peakbitrate	int4	4	RSVP parameter
license	varchar	256	Licensing terms
expiry	date		License expiry date
mserid	int4	4	Record serial number, increased each time a record is modified

Table:	***episodes_list***		
Field name	**Type (SQL)**	**Length**	**Decsription**
eid	int4	4	Episode id
mid	int4	4	Id of the movie (from: ***movies_list***)
episode_start	int4	4	Start time (miliseconds from the beginning of the movie)
episode_end	int4	4	End time (miliseconds from the beginning of the movie)
descrhtml	varchar	256	URL pointer to page containing episode description
title	varchar	256	Episode title
keywords	varchar	1024	Keywords assiciated with this episode (comma separated list)
is_movie	int4	4	Field is equal to 1 if the episode is a whole movie, 0 in other cases
eserid	int4	4	Record serial number, increased each time a record is modified
descrtype	int4	4	Description type (hypertext=0 - pointer is in *descrhtml*, text=1)

Table:	***categories***		
Field name	**Type (SQL)**	**Length**	**Decsription**
cid	int4	4	Category ID
name	varchar	30	Category name
cserid	int4	4	Record serial number, increased each time a record is modified

Table:	***subcategories***		
Field name	**Type (SQL)**	**Length**	**Decsription**
sid	int4	4	Subcategory ID
name	varchar	30	Subcategory name
cid	int4	4	Category ID (a given subcategory belongs to only one category)
sserid	int4	4	Record serial number, increased each time a record is modified

Każdemu filmowi odpowiada dokładnie jeden wiersz w tabeli *movies_list*. Wiersz ten zawiera m.in. informację o id filmu (*mid*) oraz numerze kategorii i numerze podkategorii (*cid*, *sid*) do której należy dany film.

W tabeli *episodes_list* są przechowywane dwa rodzaje wierszy:

- wiersze, w których kolumna *is_movi*e ma wartość 1 opisują całe filmy (jeden wiersz dla każdego filmu), w szczególnośći kolumna *title* zawiera tytuł filmu odpowiadający danemu filmowi (*mid*).
- wiersze, w których kolumna *is_movie* ma wartość 0 opisują fragmenty filmów (epizody). Dla danego filmu (*mid*) może ich być dowolnie dużo, może również ich nie być wcale.

Tabele *categories* i *subcategories* zawierają powiązania między identyfikatorami kategorii i podkategorii (*cid*, *sid*) a ich nazwami (*name*).