1. Dodatkowe informacje

1.1. Każde zdanie SQL musi być zakończone średnikiem (**;**).

1.2. Odwołanie do argumentu funkcji w ciele funkcji jest postaci **$*n***, gdzie ***n*** jest numerem argumentu.

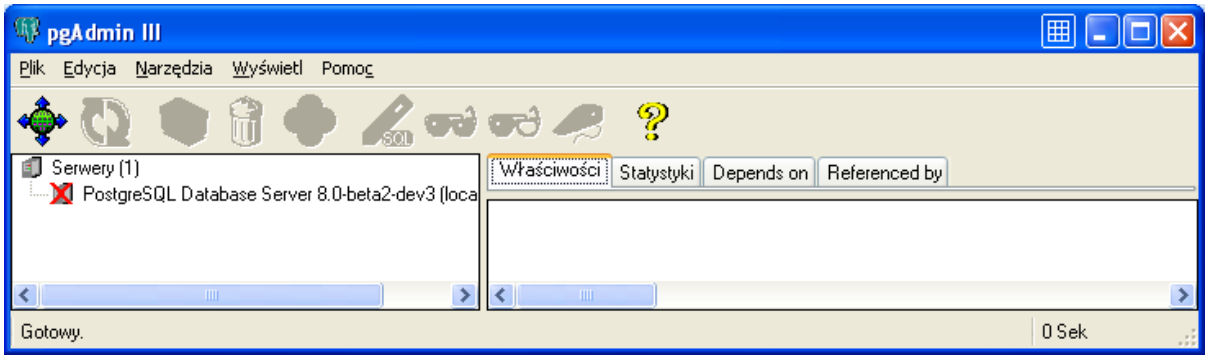
2. Czynności wstępne

2.1. Zalogować się do systemu Windows.

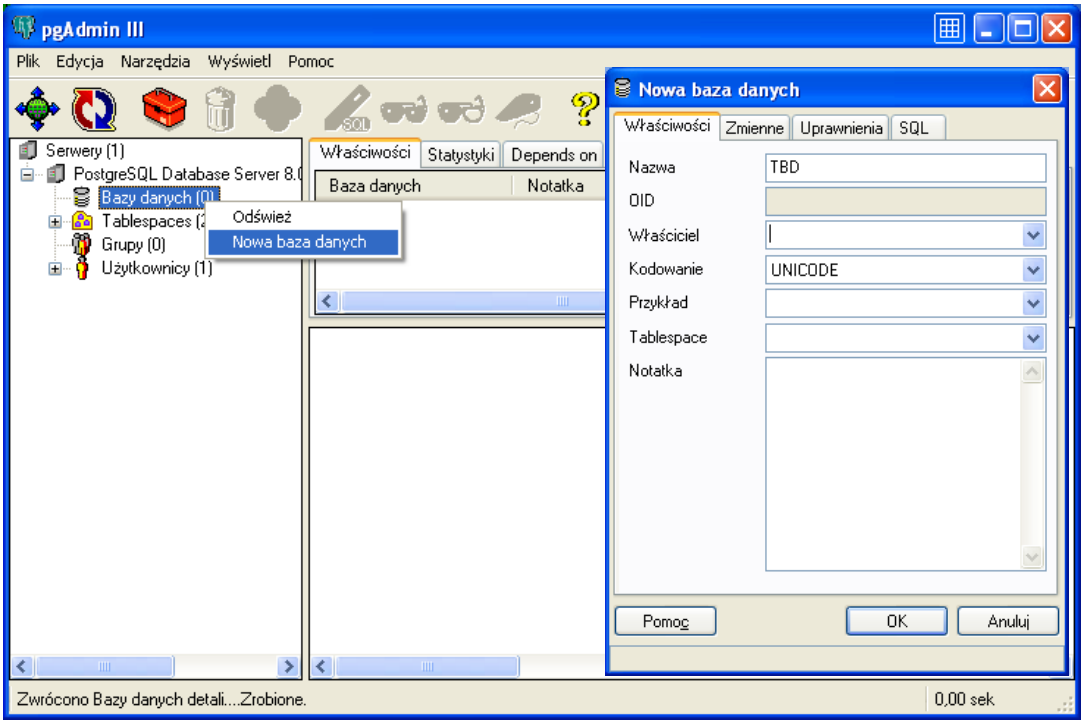
2.2. Jeżeli jest taka potrzeba, to uruchomić serwis Postgresa klikając: START → PROGRAMY → PostgreSQL *wersja* → Start Service

2.3. Uruchomić narzędzie do administracji systemem:
START → PROGRAMY → PostgreSQL versja → pgAdminIII

2.4. Dwukrotnie kliknąć na serwer Postgresa widoczny w lewym panelu okna:



2.5. Utworzyć przy pomocy menu kontekstowego nową bazę danych o dowolnej nazwie (tu: TBD) i dowolnym kodowaniu (UNICODE, UTF-8, itp.):



2.6. Otworzyć okno zapytań SQL, poprzez naciśnięcie przycisku 

3. Zadania

3.1. Poleceniem create table utworzyć tabelę o następującej strukturze:

Osoby (imie varchar(15), nazwisko varchar(15), PESEL varchar(11), data_ur timestamp)

3.2. Utworzyć dodatkową tabelę Pracownicy, wykorzystując polecenie:

```
create table Pracownicy (nr_prac integer, nr_zesp integer, pensja real) INHERITS (Osoby);
```

3.3. Wpisać 3 rekordy do tabeli Osoby:

Jan	Nowak	11111111111	1988-01-01
Adam	Kowalski	22222222222	1989-10-01
Anna	Krol	33333333333	1990-10-15

3.4. Wpisać 2 rekordy do tabeli Pracownicy:

Tomasz	Wicek	44444444444	1978-12-12	1	10	2500
Maria	Bialek	55555555555	1980-12-12	2	10	2000

3.5. Wyświetlić (poleceniem select) dane o tabelach Osoby i Pracownicy wpisane do perspektywy pg_tables, dodając frazę:

```
… where tablename = 'osoby' or tablename = 'pracownicy'
```

3.6. Wyświetlić nazwy i typy atrybutów tabeli Osoby:

```
select pa.attname, pt.typname
from pg_class pc, pg_attribute pa, pg_type pt
where pc.relname='osoby' and pc.oid =pa.attrelid and pt.oid = pa.atttypid;
```

3.7. Wyświetlić wartości niejawnej kolumny tableoid tabeli Pracownicy.

3.8. Wyświetlić wartości niejawnej kolumny tableoid tabeli Osoby. Co daje się zauważyć?

3.9. Potwierdzić swoje wcześniejsze obserwacje wyświetlając wszystkie dane wpisane do tabeli Osoby:

```
select tableoid, * from Osoby;
```

3.10. Do poprzednio zadanego zapytania dodać frazę only:

```
select tableoid, * from only Osoby;
```

3.11. Spróbować usunąć rekord dotyczący Marii Bialek z tabeli **Pracownicy**.

```
delete from Pracownicy where imie = 'Maria';
```

3.12. Sprawdzić czy rekord został usunięty **zarówno** z tabeli Pracownicy, **jak i** z tabeli Osoby.

3.13. Wpisać 2 rekordy do tabeli Pracownicy:

Witold	Wrembel	88888888888	02-02-1977	2	10	1950
Kamila	Bialek	99999999999	12-12-1983	3	20	2000

3.14. Ponownie wykonać polecenie 3.7. Czy daje się zauważyć jakąś zmianę?

3.15. Stworzyć nową tabelę, w której będą pamiętane informacje o premiach poszczególnych pracowników, przy czym atrybut premia_kwartalna będzie reprezentowany jako czteroelementowa tablica, a kolejne elementy tej tablicy będą liczbami całkowitymi; wskaźnikiem będzie numer kwartału:

```
create table premie (nr_prac integer, premia_kwartalna integer[]);
```

3.16. Wpisać następujące dane do nowoutworzonej tabeli:

```
insert into premie values (1, '{100,150,200,250}');
```

3.17. Wyświetlić wpisane do tablicy dane, wykonując zapytania typu:

```
Select * from premie;
select premia_kwartalna[1] from premie;
```

3.18. Stworzyć tabelę zawierającą informacje o książkach pożyczanych przez pracowników w zakładowej bibliotece – ich autorach, tytułach, wydawnictwie i roku wydania:

```
CREATE TABLE wypozyczenia (nr_prac integer, autor_tytul text[][]);
```

3.19. Do utworzonej tabeli wpisać 2 rekordy (dotyczące pracowników o numerach 1, i 2):

```
INSERT INTO wypozyczenia VALUES
(1, '{{"Tolkien", "Hobbit", "Iskry", 1980}, {"Dickens", "Klub Pickwicka", "MG", 1989}, {"Stone",
"Pasja zycia", "ZYSK I S-KA", 1999}}');
```

```
INSERT INTO wypozyczenia VALUES (2, '{{"Pascal", "Przewodnik", "lonely planet", 2010},
{"Archer", "Co do grosza", "REBIS Sp. z o.o.", 1999}}');
```

3.20. Wyświetlić wartości wpisane w tablicach; zaobserwować różnice i podobieństwa w otrzymywanych wynikach:

```
SELECT * FROM wypożyczenia;
SELECT nr_prac, autor_tytul[1][1] FROM wypozyczenia;
SELECT nr_prac, autor_tytul[1:3][1] FROM wypozyczenia;
SELECT nr_prac, autor_tytul[1:3][1:3] FROM wypozyczenia;
SELECT nr_prac, autor_tytul[1:3][2] FROM wypozyczenia;
SELECT nr_prac, autor_tytul[2][2] FROM wypozyczenia;
SELECT nr_prac, autor_tytul[2][1] FROM wypozyczenia;
```

3.21. Napisać funkcję w języku SQL, wyświetlającą informacje o nazwisku pracownika, którego numer podany jest parametrem. Ogólna postać funkcji jest następująca:

```
CREATE FUNCTION nazwafunkcji (typparametru1, typparametru2,…) RETURNS typwynikowy
AS 'ciałofunkcji'
LANGUAGE 'sql';

CREATE FUNCTION dane (integer) RETURNS text
AS 'select nazwisko from Pracownicy where nr_prac = $1'
LANGUAGE 'sql';
```

3.22. Przetestować działanie funkcji wpisując polecenie:

```
select dane(1) as nazwisko;
```

3.23. Napisać funkcję wyświetlającą wszystkie dane osobowe pracownika (imię, nazwisko, PESEL), którego numer podany jest parametrem. W tym celu zdefiniować najpierw typ, a dopiero w drugiej kolejności stosowną procedurę:

```
CREATE TYPE complex AS (i text, n text, p text);

CREATE FUNCTION dane2 (integer) RETURNS complex
AS 'select imie, nazwisko, PESEL from Pracownicy where nr_prac = $1'
LANGUAGE 'sql';

select dane2(2);
```

3.24. Napisać funkcję wyświetlającą wszystkie dane osobowe (imię, nazwisko, PESEL) wszystkich pracowników:

```
CREATE FUNCTION dane3 () RETURNS setof complex
AS 'select imie, nazwisko, PESEL from Pracownicy'
LANGUAGE 'sql';

select dane3();
```

3.25. Napisać funkcję wyświetlającą (tylko) **tytuły** książek pożyczonych przez pracownika o podanym parametrem funkcji numerze. Podjąć próbę takiego wskazania „współrzędnych" atrybutu tablicowego, aby w wyniku wykonania polecenia SELECT faktycznie pojawiły się tylko tytuły książek (a nie np. autorzy-tytuł).

3.26. Napisać funkcję w proceduralnym języku Postgresa – plpgsql, łączącą w jedno słowo dwa ciągi tekstowe podane parametrem:

```
CREATE OR REPLACE FUNCTION concat (text, text) RETURNS text AS
$$      /*to jest delimiter początkowy – może być dowolnym znakiem lub ciągiem znaków*/
    /*tu mogą się pojawić deklaracje poprzedzone słowem DECLARE */
    BEGIN
    RETURN $1||$2;
```

```
    END;
$$   /*to jest delimiter końcowy*/
LANGUAGE 'plpgsql';
```

3.27. Przetestować działanie funkcji – np.:

```
select concat('po','danie');
```

3.28. Napisać funkcję w proceduralnym języku Postgresa – plpgsql, zwracającą wartość pensji pracowników podwyższoną o 25% i przetestować jej działanie.

```
CREATE OR REPLACE FUNCTION extra_money (integer) RETURNS real AS
$$
    DECLARE zm real;
    BEGIN
    SELECT 1.25 * pensja INTO zm FROM pracownicy WHERE nr_prac = $1;
    RETURN zm;
    END;
$$
LANGUAGE 'plpgsql';
```

3.29. W celu zapamiętania numerów telefonów poszczególnych osób, do tabeli Osoby dodać 2 kolumny i dla 2wybranych osób wpisać do nich przykładowe dane:

```
ALTER TABLE Osoby ADD COLUMN prefix_tel TEXT;
ALTER TABLE Osoby ADD COLUMN tel TEXT;
UPDATE Osoby SET prefix_tel = '0-16' WHERE imie = 'Witold';
UPDATE Osoby SET tel = '7654321' WHERE imie = 'Witold';
UPDATE Osoby SET prefix_tel = '0' WHERE imie = 'Kamila';
UPDATE Osoby SET tel = '500010203' WHERE imie = 'Kamila';
```

3.30. Napisać funkcję łączącą dla pracowników wartości wpisane w kolumnie prefix_tel z wartościami kolumny tel:

```
CREATE OR REPLACE FUNCTION merge_fields(t_row pracownicy) RETURNS text AS
$$
    BEGIN
    RETURN t_row.imie || ' ' || t_row.nazwisko || ' ' || t_row.prefix_tel || t_row.tel;
    END;
$$
LANGUAGE plpgsql;

SELECT merge_fields(t.*) FROM pracownicy t
```

3.31. Napisać funkcję łączącą wartości wpisane w kolumnie prefix_tel z wartościami kolumny tel:

```
CREATE OR REPLACE FUNCTION merge_fields(t_row osoby) RETURNS text AS
$$
    BEGIN
    RETURN t_row.prefix_tel || t_row.tel;
    END;
$$
LANGUAGE plpgsql;

SELECT merge_fields(t.*) FROM osoby t ;
```

3.32. Napisać regułę uniemożliwiającą zmianę wartości atrybutu pensja dla aktualizowanego pracownika. Reguła ma ogólną postać:

```
CREATE RULE nazwareguły AS ON zdarzenie TO obiekt [WHERE warunek]
DO [ALSO|INSTEAD] [akcja | (akcje) | NOTHING];
```

```
CREATE RULE regula1
AS ON UPDATE TO Pracownicy
WHERE NEW.pensja <> OLD.pensja
DO INSTEAD NOTHING;
```

3.33. Sprawdzić poprawność działania reguły, a następnie usunąć regułę:

```
SELECT * FROM pracownicy;
UPDATE pracownicy SET nr_zesp = 30 WHERE nr_zesp = 20;
SELECT * FROM pracownicy;
UPDATE pracownicy SET pensja = 2000 WHERE imie = 'Witold';
SELECT * FROM pracownicy;
```

3.34. Napisać regułę, która nie dopuści na dopisanie nowego pracownika o numerze mniejszym bądź równym zeru.

3.35. Za pomocą reguł utworzyć modyfikowalne widoki (perspektywy), które normalnie nie są obsługiwane przez PostgreSQL. W tym celu utworzyć perspektywę tabeli Osoby:

```
CREATE VIEW osob_view AS SELECT imie, nazwisko, PESEL FROM osoby WHERE
imie='Witold ';

CREATE RULE reg2 AS ON INSERT TO osob_view DO INSTEAD INSERT INTO osoby
(imie, nazwisko, PESEL) VALUES (NEW.imie,NEW.nazwisko, NEW.PESEL);
```

3.36. Trigger definiuje się następującą składnią:

```
CREATE TRIGGER nazwa
BEFORE | AFTER /*czy trigger ma być wykonany przed czy po zdarzeniu*/
INSERT | UPDATE | DELETE /*których zdarzeń trigger dotyczy, można łączyć kilka przez OR)*/
ON tabela
FOR EACH
ROW | STATEMENT /*czy trigger ma być wywołany raz na rekord, czy raz na instrukcję*/
EXECUTE PROCEDURE procedura (parametry); /*co ma być wywołane jako obsługa triggera*/
```

Trigger usuwa się następującą składnią:

```
DROP TRIGGER nazwa ON tabela;
```

3.37. W celu pamiętania czasu modyfikacji danych w tabeli Premie, dodać do niej 1 kolumnę:

```
ALTER TABLE Premie ADD COLUMN last_updated timestamptz;
```

3.38. Napisać funkcję

```
CREATE OR REPLACE FUNCTION upd() RETURNS trigger AS
$$
    BEGIN
    NEW.last_updated = now();
    RETURN NEW;
    END;
$$
LANGUAGE plpgsql;
```

3.39. Utworzyć wyzwalacz, który dla każdego następnego wstawienia nowego wiersza (lub modyfikacji istniejącego) w tabeli Premie spowoduje umieszczenie aktualnego znacznika czasu w polu last_updated bieżącego rekordu tabeli:

```
CREATE TRIGGER last_upd
BEFORE insert OR update ON Premie
FOR EACH ROW
```

```
EXECUTE PROCEDURE upd();
```

3.40. Przetestować działanie napisanego wyzwalacza:

```
SELECT * FROM Premie;
INSERT INTO Premie VALUES (2, '{300,150,100,150}');
SELECT * FROM Premie;
```

3.41. Utworzyć tabelę TOWARY(id,nazwa,cena_netto) i wpisać następujące dane:

1	kabel	50
2	laptop	940
3	monitor	600

3.42. Napisać funkcję podatek_vat() oraz wyświetlić towary – tzn. (id, nazwa, cena_netto, podatek_vat(cena_netto), cena_netto + podatek_vat(cena) as cena_brutto).

3.43. Założyć tabelę TOWARY2(id,nazwa,cena,cena_vat,cena_brutto). Napisać wyzwalacz, który przy wprowadzaniu oraz uaktualnianiu krotek (id,nazwa,cena_netto), obliczy odpowiednio cenę_vat oraz cenę_brutto.

3.44. Po zatwierdzeniu wykonania zadań przez prowadzącego, usunąć bazę, utworzoną na początku laboratorium.