Jezyk SQL (strukturalny jezyk zapytan)

Na bazie danych wykonywane sa nastepujace operacje:

1. dopisywanie, usuwanie i modyfikacja danych,
2. wyszukiwanie danych spelniajacych okreslone kryteria wyboru,
3. przetwarzanie danych, np. wykonywanie obliczen z wykorzystaniem operacji logicznych, arytmetycznych i funkcji wbudowanych w system zarzadzania bazami danych,
4. zapisywanie sekwencji operacji w postaci odpowiednich konstrukcji algorytmicznych - makra i makrodefinicje,
5. zapamietanie danych bedacych wynikiem wyszukiwania w celu dalszego wykorzystania, np. wydrukowania lub utworzenia nowych relacji w bazie.

Uzytkownik okreslajac, jakie operacje na danych maja byc wykonane, posluguje sie pewnym abstrakcyjnym jezykiem, tzw. jezykiem manipulacji danymi. Jezyki te we wspólczesnych systemach zarzadzania bazami danych maja bardzo duze mozliwosci. Pozwalaja nie tylko wybrac jedna z dostepnych w systemie operacji, ale umozliwiaja definiowanie operacji zlozonych, okreslania zlozonych kryteriów wyboru, w których korzysta sie z wbudowanych w system funkcji lub funkcji definiowanych przez uzytkownika (np. w systemie Access mozna definiowac wlasne funkcje w jezyku Visual Basic). Pozwalaja takze na tworzenie, zapamietanie, a nastepnie uruchamianie pewnych powtarzajacych sie sekwencji w postaci makr. Najbardziej popularnym obecnie jezykiem tego typu jest jezyk **SQL** (Structured Query Language). Pierwszy standard organizacji ISO tego jezyka powstal w 1987 roku, drugi w 1989, a trzeci w 1993 roku. W ksiazce uzywamy implementacji PostgreSQL 7.1 dostepnej w sieci pod adresem: `http://www.postgresql.org`. Wersja ta niewiele rózni sie od Standardu. Drobne róznice dotycza jedynie regul syntaktycznych poszczególnych konstrukcji.

Podstawowymi pojeciami w jezyku SQL sa:

- tabela (odpowiednik relacji),
- kolumna tabeli (atrybut relacji),
- wiersz tabeli (pojedyncza krotka),
- wartosc wpisana w komórke na przecieciu wiersza i kolumny.

Poslugujac sie jezykiem SQL mozemy wykonac wszystkie podstawowe operacje na danych:

- wyszukiwanie danych w bazie (oznaczone slowem kluczowym SELECT),
- wstawianie, modyfikowanie i usuwanie danych z bazy (INSERT INTO, UPDATE, DELETE FROM),
- dodawanie do bazy danych nowych relacji, indeksów i perspektyw (CREATE TABLE, CREATE INDEX, CREATE VIEW),
- ochrona danych w bazie (GRANT, REVOKE).

1 SQL jako jezyk definicji danych

1.1 Typy danych, tworzenie tabel

W zakresie typów danych nie ma zgodnosci wsród firm dostarczajacych systemy baz danych. W Standardzie jezyka SQL wystepuja typy napisowe, liczbowe, typy dla oznaczania dat i czasu. W PostgreSQL 7.1 zaimplementowano nastepujace typy danych:

typ	**opis**
BOOL	typ logiczny,
CHAR(n)	napis stalej dlugosci n,
VARCHAR(n)	napis zmiennej dlugosci.
TIME	czas,
DATE	data,
TIMESTAMP	czas i data,
FLOAT	liczba zmiennoprzecinkowa,
INTEGER	liczba calkowita.

Omawianie instrukcji SQL rozpoczynamy od instrukcji sluzacej do utworzenia nowej tabeli CREATE TABLE. Pokazemy jej zastosowanie na przykladzie tworzenia tabeli o nazwie *Pracownicy* z nastepujacymi kolumnami:

identyfikator pracownika,
nazwisko pracownika,
imie pracownika,
kod dzialu, w którym pracuje,
informacja czy pracownik jest ubezpieczony.

Instrukcja sluzaca do utworzenia takiej tabeli jest nastepujaca:

```
CREATE TABLE Pracownicy
( ID_prac INTEGER,
nazwisko VARCHAR(25),
imie VARCHAR(10),
kod_dz VARCHAR(10),
ubezpiecz BOOL).
```

Mozna okreslac domyslne wartosci, na przyklad jezeli zastapimy czesc dotyczaca kolumny *ubezpiecz* przez *ubezpiecz BOOL DEFAULT 't'*, to w ostatniej kolumnie bedzie wpisywana domyslna wartosc badz podana explicite w instrukcji INSERT.

W PostgreSQL mozna takze definiowac tabele, które dziedzicza atrybuty po tabelach zdefiniowanych wczesniej. Na przyklad tabela *Dyrektorzy* zdefiniowana jako:

```
CREATE TABLE Dyrektorzy
(wynagrodzenie  FLOAT,
samochod VARCHAR(10))
INHERITS (Pracownicy);
```

zawiera 7 atrybutów, z czego 5 jest identycznych jak w tabeli *Pracownicy* oraz dodatkowo zawiera atrybuty *wynagrodzenie*, *samochod*.

Tworzenie tabeli w PostgreSQL moze byc polaczone z ustalaniem wiezów spójnosci. Wiezy spójnosci sa to warunki zaimplementowane w bazie danych, które okreslaja kiedy dane sa zgodne z rzeczywistoscia. Wiezy ograniczaja mozliwe wartosci jakie moga pojawiac sie w wierszach tabeli. W PostgreSQL tworzeniu tabeli moze towarzyszyc ustalenie nastepujacych wiezów (CONSTRAINT):

- wiezy NOT NULL - w kolumnie nie jest dozwolona wartosc NULL,
- wiezy CHECK - warunek, który ma byc prawdziwy dla wszystkich wierszy tabeli,
- wiezy UNIQUE – wartosc atrybutu nie moze sie powtarzac w tabeli,
- wiezy PRIMARY KEY – okreslaja klucz podstawowy, czyli sa polaczeniem wiezów UNIQUE i NOT NULL,
- wiezy FOREIGN KEY – okreslaja klucz obcy, czyli atrybut zwiazany z kluczem z innej tabeli, tworzy sie przez zastosowanie REFERENCES w jednej tabeli oraz PRIMARY KEY w drugiej.

Utworzymy nowa tabele *Praca:*

```
CREATE TABLE Praca
( ID_trans INTEGER PRIMARY KEY,
ID_prac INTEGER REFERENCES Pracownicy,
data DATE NOT NULL,
czas FLOAT CHECK (czas > 0),
stawka FLOAT CHECK (stawka> 0 AND stawka< 50.00)
)
```

oraz zmodyfikujemy instrukcje tworzenia tabeli *Pracownicy*.

```
CREATE TABLE Pracownicy
( ID_prac INTEGER PRIMARY KEY,
nazwisko VARCHAR(25) NOT NULL,
imie VARCHAR(10),
kod_dz VARCHAR(10),
ubezpiecz BOOL).
```

Zauwazmy, ze po ustaleniu powyzszych wiezów kazdej operacji wpisywania lub modyfikacji danych bedzie towarzyszyc sprawdzanie pewnych warunków. Dla tabeli *Praca*, w której ID_trans jest kluczem glównym, sprawdza sie, czy wartosci atrybutu ID_trans w poszczególnych wierszach sa okreslone i unikatowe. Z kolei kazda wartosc atrybutu ID_prac z tabeli *Praca* powinna wystepowac w tabeli *Pracownicy*, wartosc atrybutu *data* musi byc okreslona, zas wartosci atrybutów *czas* i *stawka* musza spelniac warunki okreslone we wiezach, tj. *czas* musi byc dodatni, *stawka* jest dodatnia i mniejsza niz 50. Wiersze, które nie spelniaja warunków nie beda wpisywane do tabeli.

1.2 Zmiany w definicji tabeli

Instrukcja ALTER TABLE umozliwia wykonanie nastepujacych operacji:

- zmiana nazwy tabeli,
- zmiana nazwy kolumny w tabeli,
- dolaczenie nowej kolumny,
- dolaczenie wartosci domyslnej w kolumnie,
- usuniecie definicji wartosci domyslnej.

Instrukcje:

- ALTER TABLE Praca RENAME TO Godz_dod,
- ALTER TABLE Pracownicy RENAME COLUMN ID_prac to ID,
- ALTER TABLE Pracownicy ADD COLUMN Pesel CHAR(11),
- ALTER TABLE Pracownicy ALTER COLUMN ubezpiecz SET DEFAULT ‘t’,
- ALTER TABLE Pracownicy ALTER COLUMN ubezpiecz DROP DEFAULT,

sluza odpowiednio do zmiany nazwy tabeli *Praca,* zmiany nazwy kolumny ID_prac w tabeli *Pracownicy,* dolaczenie nowej kolumny w tabeli *Pracownicy,* ustalenie wartosci domyslnej w kolumnie *ubezpiecz* oraz usuniecie definicji wartosci domyslnej.

1.3 Definiowanie indeksów, klastrów i perspektyw

Obok instrukcji tworzenia tabeli w SQL dysponujemy instrukcja sluzaca do tworzenia indeksów. Indeksy stanowia podstawowa strukture w bazach danych umozliwiajaca szybkie wyszukiwanie informacji. Zalozenie nowego indeksu wykonuje sie przy pomocy instrukcji:

CREATE [UNIQUE] INDEX nazwa_indeksu ON nazwa_tabeli (nazwa_atrybutu, ..)

Slowo kluczowe UNIQUE gwarantuje stworzenie indeksu jednoznacznego. Kazda próba wpisania duplikatów atrybutów okreslajacych indeks spowoduje blad. Na przyklad w tabeli *Pracownicy* mozemy zwiazac indeks z dzialem pracownika:

CREATE INDEX ind_dz ON *Pracownicy* (kod_dz);

Zauwazmy, ze potrzebne jest przypisanie nazwy kazdemu indeksowi.

Z indeksem laczy sie pojecie klastra pamieci. Wykonanie instrukcji:
CLUSTER nazwa_indeksu ON nazwa_tabeli
powoduje przemieszczenie elementów tabeli zgodnie z uporzadkowaniem przez wskazany indeks, co ulatwia wyszukiwanie duplikatów. Wykonanie
CLUSTER ind_dz ON Pracownicy
spowoduje, ze wiersze dotyczace pracowników z tych samych dzialów beda pogrupowane i w zwiazku z tym instrukcja typu
SELECT * FROM Pracownicy WHERE kod_dz = 10
wykona sie szybciej.

Perspektywa jest tabela wirtualna. Mozna jej uzywac tak jakby byla tabela. Wiersze perspektywy nie sa przechowywane w bazie danych, sa jedynie wyliczane na zadanie. Perspektywy sluza do dostosowania bazy danych do potrzeb róznych grup uzytkowników. Stanowia 'perspektywe' z jakiej dana grupa widzi baze danych. Perspektywy umozliwiaja wykonywanie operacji podobnych do operacji wykonywanych na tabelach, to jest operacji wyszukiwania i modyfikowania danych. Ponizsza instrukcja definiuje perspektywe *Pracownicy-sprzedaz* zlozona z tych wierszy z tabeli *Pracownicy*, które dotycza osób zatrudnionych w dziale sprzedazy.
CREATE VIEW Pracownicy-sprzedaz AS
SELECT nazwisko, imie FROM Pracownicy WHERE kod_dz='sprzedaz';

1.4 Liczniki i wyzwalacze

W PostgreSQL oprócz tabel, indeksów i perspektyw mozna takze definiowac liczniki, wyzwalacze i funkcje. Licznik jest obiektem, który automatycznie zwieksza swoja wartosc po jego kazdorazowym odczycie. Liczniki sa uzywane do generowania unikalnych identyfikatorów wierszy. Licznik definiuje sie przy pomocy polecenia:

CREATE SEQUENCE nazwa_licznika

Trzy funkcje standardowe operuja na licznikach:

- nextval(nazwa) – zwraca kolejna wartosc licznika i jednoczesnie modyfikuje licznik,
- currval(nazwa) – zwraca wartosc z poprzedniego odwolania nextval, bez zmiany wartosci licznika,
- setval(nazwa, N) – ustala wartosc licznika na N.

Jezeli utworzymy licznik
CREATE SEQUENCE numer_prac;
to mozemy uzyc go przy tworzeniu tabeli:

```
CREATE TABLE Pracownicy
( ID_prac INTEGER DEFAULT nextval('numer_prac'),
nazwisko VARCHAR(25) NOT NULL,
imie VARCHAR(10),
kod_dz VARCHAR(10),
ubezpiecz BOOL).
```

Uwaga: jezeli licznik nie zostal zaincjalizowany przy pomocy *setval*, to domyslnie przyjmuje sie wartosc 1 jako poczatkowa.

Wyzwalacze sa procedurami uruchamianymi automatycznie na skutek zdarzen zachodzacych w bazie danych. Sluza do oprogramowania wiezów spójnosci i do oprogramowania stalych czynnosci, które powinny byc wykonywane w kazdej aplikacji korzystajacej z bazy danych. Wyzwalacze obu typów moga byc uruchamiane w wyniku nastepujacych polecen: INSERT (wstawianie), DELETE (usuwanie), UPDATE (modyfikacja). Wyzwalacz definiuje sie za pomoca polecenia:

```
CREATE TRIGGER nazwa_wyzwalacza {BEFORE|AFTER}
{zdarzenie_1 [OR zdarzenie_2 [OR zdarzenie_3]]}
ON nazwa_tabeli FOR EACH ROW
EXECUTE  PROCEDURE nazwa_procedury (argumenty,...);
```

gdzie zdarzenie_1, zdarzenie_2, zdarzenie_3 jest jednym z polecen INSERT, DELETE, UPDATE, zas nazwa_procedury jest nazwa procedury zdefiniowanej w jezyku C.
Na przyklad instrukcja:
CREATE TRIGGER usuwanie_prac AFTER DELETE
ON *Pracownicy* FOR EACH ROW EXECUTE PROCEDURE komunikat();
powoduje utworzenie wyzwalacza, który bedzie wywolywac procedure (która musi byc wczesniej zdefiniowana*) komunikat* po kazdej operacji usuwania danych z tabeli *Pracownicy*.

Uzytkownik PostgreSQL ma mozliwosc definiowania w SQL swoich wlasnych funkcji, których moze nastepnie uzywac w wyrazeniach i zapytaniach . Definicja funkcji jest postaci:

```
CREATE FUNCTION nazwa_funkcji ([typ_arg {,typ_arg}])
RETURNS typ_wyniku
AS ciag_zapytan_sql
LANGUAGE 'sql';
```

Zauwazmy, ze kazda funkcja zwraca pewien wynik - jest to wartosc zwracana przez ostatnie zapytanie z ciagu zapytan SQL (ciala funkcji). Odwolanie do argumentu funkcji w ciele funkcji jest postaci $n, gdzie n jest numerem argumentu.

Przyklad. Instrukcja

```
CREATE FUNCTION premia(FLOAT) RETURNS FLOAT AS
'SELECT 0.25 * $1;'
LANGUAGE 'sql'
```

definiuje funkcje jedno-argumentowa premia; cialo tej funkcji sklada sie tylko z jednego zapytania. Mozna jej uzyc w innych zapytaniach, np.
SELECT premia(Praca.stawka) AS odpowiedz.

1.5 Usuwanie tabel, indeksów, perspektyw, liczników, wyzwalaczy, funkcji

Instrukcje:
DROP TABLE nazwa_tabeli;
DROP INDEX nazwa_indeksu;
DROP VIEW nazwa_perspektywy;
DROP SEQUENCE nazwa_licznika;
DROP TRIGGER nazwa_wyzwalacza;
DROP FUNCTION nazwa_funkcji;
sluza odpowiednio, do usuwania tabeli, indeksu, perspektywy, licznika, wyzwalacza i funkcji.
Przyklad. Instrukcja:
DROP TABLE *Pracownicy;*
powoduje, ze dalsze odwolania do tabeli *Pracownicy* (na przyklad wstawianie danych) nie beda mogly byc wykonane.

2 SQL jako jezyk manipulowania danymi

2.1 Wstawianie danych

Dane do tabeli sa wstawiane za pomoca instrukcji:

INSERT INTO nazwa_tabeli VALUES (wartosc {, wartosc})

Załózmy, ze do tabeli *Pracownicy* chcemy wstawic dane dotyczace Jana Kowalskiego, zatrudnionego w dziale SPR, z identyfikatorem 12, ubezpieczonemu. Wystarczy wykonac instrukcje:
INSERT INTO Pracownicy VALUES (12,'Kowalski', 'Jan', 'SPR','t').
Warto zwrócic uwage na kolejnosc wartosci odpowiadajaca kolejnosci atrybutów podanych w instrukcji tworzacej tabele.
Mozliwa jest takze bardziej zlozona postac operacji wstawiania wykorzystujaca instrukcje SELECT.
Instrukcja
INSERT INTO Dyrektorzy(ID_prac, nazwisko, imie, kod_dz, ubezpiecz)
SELECT ID_prac, nazwisko, imie, kod_dz, ubezpiecz
FROM Pracownicy WHERE kod_dz='SPR';
wstawia dane dotyczace wszystkich pracowników z dzialu 'SPR' do tabeli Dyrektorzy, pozostawiajac atrybuty *wynagrodzenie, samochod* jako NULL

2.2 Usuwanie danych

Dane mozna usuwac z tabel bazy danych kierujac do systemu nastepujaca instrukcje:
DELETE FROM nazwa_tabeli
[WHERE warunek].
Jezeli uzyje sie opcjonalnego warunku, to zostana usuniete jedynie wiersze spelniajace warunek. Jezeli warunek nie wystepuje, to usuwa sie wszystkie wiersze z tabeli. Instrukcja
DELETE FROM *Pracownicy*
WHERE kod_dz='SPR';
usuwa wszystkich pracowników z dzialu SPR.

2.3 Modyfikacja danych

System zarzadzania baza danych dokonuje modyfikacji wartosci zapisanych w bazie danych, jezeli otrzyma do wykonania instrukcje:
UPDATE nazwa_tabeli
SET nazwa_atrybutu = wyrazenie {, nazwa_atrybutu = wyrazenie}
[WHERE warunek]
Instrukcja
UPDATE Praca
SET stawka = stawka * 1.15
WHERE data = '12. 24.98';
zwieksza stawke o 15% dla wszystkich pracowników pracujacych 24 grudnia 1998.

3 Wyszukiwanie danych

3.1 Wyswietlanie calej tabeli i wybranych kolumn

Centralna instrukcja SQL jest instrukcja sluzaca do wyszukiwania informacji z bazy danych. Jest to instrukcja SELECT, okreslajaca z jakich tabel w bazie danych maja byc wyswietlane dane, jakie warunki maja spelniac i w jakiej dokladnie postaci maja sie pojawiac przed uzytkownikiem.
Instrukcja
SELECT * FROM nazwa_tabeli;
sluzy do wyswietlania wszystkich wierszy z tabeli, zas
SELECT nazwa_atrybutu {, nazwa_atrybutu} FROM nazwa_tabeli;
wyswietla jedynie wartosci z wybranych kolumn.
Na przyklad, jezeli do tabeli *Pracownicy* zostala wpisana tylko jedna krotka, to wynikiem zapytania
SELECT ID_prac, nazwisko, imie, kod_dz FROM Pracownicy;
bedzie:

ID_prac	nazwisko	imie	kod_dz
12	Kowalski	Jan	SPR

Mozliwe jest wyswietlanie wartosci z wykonywaniem obliczen. Instrukcja
SELECT data, stawka * 1.15 FROM Praca;
wyswietli wartosci atrybutu *stawka* zwiekszone o 15 %, choc wartosci przechowywane w tabeli *Praca* nie zostana zmienione.

3.2 Wybieranie wierszy przy pomocy WHERE

Klauzula zapisana po slowie kluczowym WHERE sluzy do okreslania warunku jaki musza spelniac wyswietlane dane. W klauzulach mozna uzywac operatorów logicznych koniunkcji (AND), alternatywy (OR), negacji (NOT). Instrukcja
SELECT nazwisko, imie FROM Pracownicy WHERE kod_dz = 'SPR' OR kod_dz = 'INFO';
wyswietla nazwiska i imiona pracowników z dzialu SPR i INFO.

Umieszczenie slowa kluczowego DISTINCT po slowie kluczowym SELECT oznacza eliminacje powtarzajacych sie wierszy.

Ponadto, wyniki zapytania moga zostac uporzadkowane wedlug wybranych atrybutów, w porzadku rosnacym (domyslnie) lub malejacym (co zapisuje sie DESC).
Na przyklad
SELECT DISTINCT data, stawka FROM Praca ORDER BY data DESC;
wyswietli wszystkie wartosci rózne, odpowiadajace atrybutom *data* i *stawka* z tabeli *Praca,* uporzadkowane wedlug dat w porzadku malejacym.
Czesto w instrukcji wyszukiwania sprawdza sie czy napis jest zgodny z pewnym ustalonym wzorcem, np. czy zaczyna sie od ustalonej litery lub zawiera ustalone podslowo. W opisach wzorca symbol % zastepuje dowolny napis.
Na przyklad zapytanie:
SELECT ID_prac, nazwisko FROM Pracownicy WHERE nazwisko LIKE 'Kowalsk%';
sluzy do wyszukania wszystkich o nazwisku Kowalski lub Kowalska.

Rózne mozliwe wzorce sa nastepujace:

wzorzec	**porównanie**
LIKE 'A%'	zaczyna sie od A
LIKE '%A%'	zawiera A
LIKE ' A%'	zawiera A jako drugi znak
LIKE ' A%B%'	zaczyna sie od A i zawiera B
NOT LIKE 'A%'	nie zaczyna sie od A

W PostgreSQL 7.1 dopuszczalne jest takze uzywanie wyrazen regularnych we wzorcach.

3.3 Zlaczenia tabel

Informacja wypisywana w wyniku wykonania zapytania moze dotyczyc kilku tabel. Aby powiazac informacje zawarte w róznych tabelach definiuje sie predykat zlaczeniowy. Ponadto, w zwiazku z tym, ze istnieje potrzeba odrózniania kolumn z róznych tabel wykorzystuje sie konstrukcje poprzedzania nazwy kolumny nazwa tabeli. Czasami dla oznaczenia tabel uzywa sie aliasów, czyli nazw zastepczych.
Przypuscmy, ze oprócz tabeli *Pracownicy* w bazie danych jest takze zdefiniowana tabela *Dzial* z nastepujaca zawartoscia:

kod_dz	**nazwa_dzialu**	**siedziba**	**kierownik**
SPR	sprzedaz	Gdynia	2234
INFO	informatyczny	Gdansk	1212

Zapytanie
SELECT P.imie, P.nazwisko, D.nazwa_dzialu, D.siedziba
FROM Pracownicy P, Dzial D
WHERE P.kod_dz = D.kod_dz;
sluzy do wypisania imion i nazwisk wszystkich pracowników, wraz z podaniem nazwy dzialu, w którym pracuja oraz siedziby dzialu. Predykat zlaczeniowy w powyzszym przykladzie jest postaci:
Pracownicy. kod_dz = Dzial. kod_dz
(nazwy P i D sa odpowiednio aliasami dla nazw Pracownicy i Dzial).
Wiersz kazdego pracownika zostaje zlaczony z dokladnie jednym, odpowiadajacym przez wartosc atrybutu kod_dz, wierszem z tabeli *Dzial.*

Zapytanie
SELECT Pracownicy.nazwisko, Praca.stawka FROM Pracownicy, Praca
WHERE Pracownicy. ID_prac = Praca.ID_prac AND Praca.data = '12.24.98';
zwraca nazwiska i stawki tych pracowników, którzy pracowali 24 grudnia 98 r.

Dopuszczalne jest laczenie wiecej niz dwóch tabel, a takze laczenie ze soba tej samej tabeli.
Zapytanie
SELECT Pracownicy.nazwisko FROM Pracownicy, Praca, Dzial WHERE
Praca.data = ’12.24.98’ AND Dzial.nazwa_dz=’marketing’ AND
Pracownicy.kod_dz = Dzial.kod_dz AND Pracownicy.ID_prac = Praca.ID_prac;
zwraca nazwiska pracowników z dzialu marketingu, którzy pracowali 24 grudnia 98 r.

Zapytanie
SELECT P.ID_prac FROM Pracownicy P, Pracownicy K WHERE
K.nazwisko = ‘Kowalski’ AND K.imie = ‘Jan’ Dzial.kierownik = K.ID_prac
AND P.kod_dz = Dzial.kod_dz;
zwraca identyfikatory pracowników, którzy sa podwladnymi Jana Kowalskiego.

3.4 Zlaczenia przy pomocy UNION, INTERSECT, EXCEPT

W PostrgeSQL 7.1 dopuszczalne jest laczenie wyników zapytan przy pomocy UNION, INTERSECT, EXCEPT co odpowiada, odpowiednio, sumie, przekrojowi i róznicy w algebrze relacji.

Zapytanie

- SELECT nazwisko FROM Pracownicy UNION SELECT nazwisko FROM Dyrektorzy;

zwraca nazwiska osób wystepujacych w tabeli pracownicy lub Dyrektorzy,

- SELECT ID_prac FROM Pracownicy EXCEPT SELECT ID_prac FROM Praca WHERE data = ’12.24.00’

zwraca identyfikatory pracowników, którzy nie pracowali dnia 24 grudnia 2000.

Zauwazmy, ze zapytaniom wyszukujacym dane odpowiadaja operacje w algebrze relacji.

Na przyklad:

1. SELECT nazwisko, imie FROM *Pracownicy*; $\pi_{\text{nazwisko, imie}}(\text{Pracownicy})$

2. SELECT *P.nazwisko, P.kod_dz, S.stawka* FROM *Pracownicy P, Praca S*;

$\pi_{\text{nazwisko, kod_dz}}(P) \times \pi_{\text{stawka}}(S)$

3. SELECT * FROM *Praca* WHERE *stawka*>=30; $\sigma_{\text{stawka} > = 30}(\text{Praca})$

4. SELECT ID_prac FROM *Praca* WHERE stawka>=20 AND stawka=<30;

$\pi_{\text{ID_prac}} (\sigma_{\text{stawka} >= 20 \wedge \text{stawka} <= 30}(\text{Praca}))$

5. SELECT * FROM *R,S*; $R \times S$

6. SELECT nazwisko FROM Pracownicy UNION SELECT nazwisko FROM Dyrektorzy;

$\pi_{nazwisko}(Pracownicy) \cup \pi_{nazwisko}(Dyrektorzy)$

3.5 Zastosowanie funkcji sumarycznych

Dane w tabeli (lub kilku tabelach) moga zostac podsumowane przy uzyciu funkcji sumarycznych:
AVG - oblicza wartosc srednia wybranego atrybutu,
COUNT - liczba wierszy,
MIN - wartosc minimalna,
MAX - wartosc maksymalna,
SUM - suma wartosci.
Wyliczanie wartosci funkcji COUNT odbywa sie nastepujaco. Zapytanie
SELECT COUNT(*) FROM Pracownicy;
zwraca liczbe wierszy znajdujacych sie aktualnie w tabeli Pracownicy, zas
SELECT COUNT(nazwisko) FROM Pracownicy;
zwraca liczbe wierszy w tabeli Pracownicy, w których atrybut nazwisko ma wartosc rózna od NULL.
Zapytanie
SELECT COUNT(*), AVG(stawka) FROM Praca WHERE data = '01.01.99';
zwraca liczbe osób pracujacych 1 stycznia 99 r. oraz srednia stawke wyplacana za prace w tym dniu.

Przyklad.

Dane sa dwie tabele:

UCZEN

numer	nazwisko	imie	rok urodzenia
01	Kowalski	Jan	1988
02	Lipski	Andrzej	1989
03	Zalewska	Ewa	1989

OCENY

numer	ocena	przedmiot
01	5	matematyka
02	4	fizyka
03	4.5	matematyka
01	3	matematyka
01	2	historia

Zastosowanie wybranych funkcji przedstawiamy w ponizszej tabeli:

funkcja	przyklad operacji	wynik
COUNT	SELECT COUNT(*) FROM uczen	3

COUNT	SELECT COUNT(*) FROM uczen WHERE rok urodzenia = 1989	2
AVG	SELECT AVG(ocena) FROM oceny	3.7
SUM	SELECT SUM(ocena) FROM oceny WHERE przedmiot = fizyka	4
AVG	SELECT numer FROM oceny WHERE przedmiot = matematyka AND ocena > (SELECT AVG(ocena) FROM oceny)	01 03

□

3.6 Definiowanie grup wierszy

W jezyku SQL jest mozliwosc podzialu wynikowych wierszy zapytania na grupy i obliczenia wartosci funkcji sumarycznych na wartosciach nalezacych do poszczególnych grup. Grupa jest zbiorem wierszy z jedna lub kilkoma wspólnymi wartosciami.
Jezeli uzywamy klauzuli GROUP BY, to zapytanie moze zawierac nastepujace wyrazenia:

- nazwy atrybutów wystepujace na liscie GROUP BY,
- funkcje sumaryczne dotyczace innych atrybutów,
- stale wyrazenia.

Na przyklad instrukcja:
SELECT kod_dz, COUNT(*) AS liczba_pracownikow FROM *Pracownicy*
GROUP BY kod_dz
zwraca tabele postaci:

kod_dz	**liczba_pracownikow**
INFO	20
SPR	10

W pierwszej kolumnie zapisany jest kod dzialu, a w drugiej liczba pracowników pracujacych w danym dziale. Zauwazmy, ze suma wartosci z drugiej kolumny jest równa liczbie wierszy zapisanych w tabeli Pracownicy.
Dolaczajac klauzule:
HAVING warunek
mozna narzucic warunek, który musi spelniac kazda wyswietlana grupa.
Jezeli powyzsza instrukcje zastapimy przez:
SELECT kod_dz, COUNT(*) AS liczba_pracownikow FROM Pracownicy
GROUP BY kod_dz HAVING COUNT(*) > 10;
to otrzymamy tabele analogiczna, ale wypelniona informacjami tylko o tych dzialach, w których liczba pracowników przekracza 10 osób.

3.7 Zapytania zagniezdzone

Rozpatrzmy zapytanie:
SELECT numer_prac FROM Praca
WHERE stawka > (SELECT AVG(stawka) FROM Praca);
Jest to zapytanie zagniezdzone – zapytanie wewnetrzne, ujete w nawiasy, zwraca srednia stawke z tabeli Praca, zas zapytanie zewnetrzne zwraca identyfikatory pracowników, którzy maja stawke wieksza niz srednia.
Jak powinno wygladac zapytanie zwracajace takze nazwiska tych pracowników?

Zagniezdzanie zapytan odbywa sie wedlug nastepujacych schematów:

- SELECT atrybuty FROM tabele WHERE atrybut operator relacji (zapytanie)
- SELECT atrybuty FROM tabele WHERE atrybut IN (zapytanie)
- SELECT atrybuty FROM tabele WHERE atrybut operator relacji ALL (zapytanie)
- SELECT atrybuty FROM tabele WHERE atrybut operator relacji ANY (zapytanie)

W przypadku schematu ze slowem kluczowym ALL warunek powinien byc spelniony dla kazdego wiersza zwracanego przez zapytanie, w przypadku ANY – dla przynajmniej jednego wiersza.

Zapytanie:

- SELECT ID_prac FROM Praca WHERE stawka < ALL (SELECT Praca.stawka FROM Praca, Dzial WHERE Dzial.nazwa_dz= 'marketing' AND Praca.kod_dz=Dzial.kod_dz);

zwraca identyfikatory pracowników, którzy maja stawke nizsza od pracowników z dzialu marketingu,

- SELECT * FROM Pracownicy WHERE kod_dz IN (SELECT kod_dz From Dzial WHERE siedziba = 'Warszawa');

zwraca informacje o pracownikach pracujacych w dzialach z siedziba w Warszawie.

Przypuscmy, ze kolumna *data_zatr* zostala dodana do tabeli *Pracownicy* przy uzyciu:
ALTER TABLE Pracownicy ADD COLUMN data_zatr DATE.
Wtedy ponizsze zapytanie wylicza nazwy dzialów, które maja pracowników zatrudnionych wczesniej niz ich kierownicy.

```
SELECT nazwa_dz FROM Dzial WHERE EXISTS
(SELECT P.ID_prac FROM Pracownicy P, Pracownicy K, Dzial
WHERE P.data_zatr < K.data_zatr AND Dzial.kierownik = K.ID_prac AND Dzial.kod_dz =
P.kod_dz).
```

ZADANIA:

1. Wypisac nazwy dzialów, w których aktualnie jest zatrudniony co najmniej jeden pracownik.
2. Sporzadzic zestawienie zawierajace nazwe dzialu, nazwisko osoby z najwyzszym wynagrodzeniem w tym dziale oraz wartosc wynagrodzenia.
3. Obliczyc procentowy udzial poszczególnych dzialów w liczbie wszystkich pracowników i wynagrodzeniach w firmie.

3.8 Zastosowanie OUTER JOIN

Zapytanie

```
SELECT nazwisko FROM Pracownicy . Praca
WHERE Praca.ID_prac = Pracownicy.ID_prac;
```

zwraca jedynie nazwiska tych pracowników, którzy wystepuja w tabeli Praca. W niektórych sytuacjach wygodniejsza jest operacja OUTER JOIN, która dokonuje zlaczenia i dodatkowo wypisuje takze te wiersze, które nie sa zlaczone:

```
SELECT Pracownicy.nazwisko FROM Pracownicy  OUTER JOIN  Praca
ON Praca.ID_prac = Pracownicy.ID_prac.
```

3.9 Zastosowanie komendy SELECT do wstawiania danych, usuwania i modyfikacji

Polecenie
- INSERT INTO Dzial(kod_dz) SELECT DISTINCT kod_dz FROM Pracownicy;

sluzy do wstawienia kodów działów wystepujacych w tabeli Pracownicy do tabeli Dzial, przy czym pozostale atrybuty maja wartosc NULL,
- DELETE FROM Pracownicy WHERE ID_prac NOT IN (SELECT ID_prac FROM Praca);

usuwa informacje o tych pracownikach, którzy nie wystepuja w tabeli Praca,
- UPDATE Praca SET stawka = stawka * 1.5 WHERE ID_prac = (SELECT ID_prac FROM Pracownicy WHERE nazwisko = 'Kowalski' AND imie='Jan');

podwyzsza o 50% stawke Jana Kowalskiego,
- UPDATE Pracownicy SET wynagrodzenie = 20 * Praca.stawka FROM Praca WHERE Pracownicy.ID_prac = Praca.ID_prac AND data = '12.10.00';

ustala wynagrodzenie tych pracowników, którzy pracowali po 10 grudnia 2000;
nalezy zwrócic uwage na konstrukcje FROM, która jest nieodzowna jezeli wystepuje zlaczenie,
- SELECT ID_prac, nazwisko INTO Dane_prac FROM Pracownicy;

powyzsze zapytanie jest polaczeniem CREATE i SELECT – powoduje utworzenie nowej tabeli Dane_prac z dwiema kolumnami.

4 SQL jako jezyk nadzoru

Instrukcje SQL umozliwiaja administratorowi (a takze uprawnionym uzytkownikom) przyznawanie uprawnien do korzystania z zasobów bazy danych. Syntaktyka instrukcji przyznajacej uprawnienia jest nastepujaca:
GRANT uprawnienia {, uprawnienia}
ON nazwa_obiektu
TO nazwa_uzytk | nazwa_grupy_uzytk |PUBLIC
przy czym uprawnienia sa nastepujace:
SELECT - uprawnienie do czytania, czyli wyszukiwania,
INSERT - wstawianie,
UPDATE - modyfikacja,
DELETE - usuwanie,
ALL - wszystkie uprawnienia jednoczesnie.
Nazwa_obiektu moze byc nazwa tabeli lub perspektywy. Lista uzytkowników moze zawierac nazwy indywidualnych uzytkowników PostgreSQL, nazwy grup uzytkowników i PUBLIC na oznaczenie wszystkich uzytkowników.
Przyklad. Instrukcja
GRANT SELECT ON Pracownicy TO PUBLIC;
umozliwia wszystkim uzytkownikom odczytywanie danych z tabeli Pracownicy.
Uprawnienia odwoluje sie za pomoca instrukcji odwrotnej:
REVOKE uprawnienia {, uprawnienia}
ON nazwa_obiektu
FROM nazwa_uzytk | nazwa_grupy_uzytk |PUBLIC

Na przyklad instrukcja

REVOKE INSERT, DELETE, UPDATE
ON Pracownicy
FROM mars;
odbiera uzytkownikowi mars prawo do wstawiania, usuwania i modyfikacji danych w tabeli Pracownicy.

5 Transakcje

Z charakteru aplikacji korzystajacych z baz danych wynika, ze czesto operacja, która powinna byc wykonana nie jest pojedyncza instrukcja SQL, ale ciag instrukcji stanowiacy calosc. W takich sytuacjach korzysta sie z transakcji, czyli ciagu operacji które albo wszystkie zostana wykonane (jezeli to jest mozliwe), albo zadna operacja nie zostanie wykonana. W pierwszym przypadku mówimy o zatwierdzeniu transakcji, a w drugim - o wycofaniu z transakcji. Na przyklad operacja przelania pieniedzy z jednego konta na drugie jest pojedyncza operacja z punktu widzenia aplikacji bankowej mimo, ze sklada sie z dwóch operacji modyfikacji SQL-owych (powiekszenie salda na jednym koncie i zmniejszenie salda na drugim koncie). Aby zapewnic spójnosc bazy danych powinny zostac wykonane obie operacje modyfikacji lub zadna z nich. Zatem obie te operacje powinny stanowic pojedyncza transakcje.
W PostgreSQL instrukcja rozpoczynajaca transakcje jest BEGIN, zas konczaca jest ROLLBACK oznaczajace wycofanie albo COMMIT oznaczajace zatwierdzenie. Kazda transakcja jest wiec postaci:
BEGIN WORK
ciag_instrukcji_SQL
ROLLBACK WORK | COMMIT WORK
Uzytkownik korzystajacy z PostgreSQL otrzymuje informacje o efekcie wykonania kazdej operacji i sam musi zdecydowac która z operacji zakonczyc transakcje.

6 Tabele slownikowe

W PostgeSQL specjalne znaczenie maja tzw. tabele slownikowe, które zawieraja informacje o uzytkownikach, bazach danych, tabelach. Nazwy tabel slownikowych w PostgreSQL zaczynaja sie od przedrostka pg_. Tabela pg_user zawiera informacje o uzytkownikach, wsród jej atrybutów wystepuja:
usename - nazwa uzytkownika,
usesysid - jednoznaczny numer uzytkownika (przypisany w momencie dolaczenia do grupy),
usecreatedb - atrybut typu logicznego, okresla czy uzytkownik ma prawo tworzyc bazy,
usesuper - atrybut typu logicznego, okresla czy uzytkownik ma prawa administratora.
Tabela pg_database zawiera informacje o bazach i zawiera atrybuty:
datname - nazwa bazy,
datdba - numer uzytkownika, wlasciciela bazy.
Tabela pg_type sluzy do przechowywania informacji o typach danych (takze definiowanych przez uzytkowników). Zawiera atrybuty:
typname - nazwa typu danych,
typowner - numer uzytkownika, który zdefiniowal typ danych (numer administratora dla typów standardowych),
typlen - dlugosc typu.
Tabela pg_proc przechowuje informacje o procedurach takze funkcjach definiowanych przez uzytkownika. Zawiera miedzy innymi atrybuty:

proname - nazwa,
proowner - nazwa uzytkownika, wlasciciela procedury.
Tabel slownikowych mozna uzywac podobnie jak innych tabel, czyli kierowac do nich zapytania. Na przyklad zapytanie:

```
SELECT usename FROM pg_user
WHERE usesuper = 't';
```

wybiera wszystkich uzytkowników, którzy maja prawa administratora.
Zapytanie

```
SELECT usename, datname  FROM pg_user, pg_database
WHERE usesysid = datdba
ORDER BY usename;
```

wyswietla informacje o wszystkich uzytkownikach i zalozonych przez nich bazach.

Uzywane tabele:

CREATE TABLE *Pracownicy*
(ID_prac INTEGER,
nazwisko VARCHAR(25),
imie VARCHAR(10),
kod_dz VARCHAR(10),
ubezpiecz BOOL).

CREATE TABLE *Dyrektorzy*
(wynagrodzenie FLOAT,
samochod VARCHAR(10))
INHERITS (Pracownicy);

CREATE TABLE *Praca*
(ID_trans INTEGER PRIMARY KEY,
ID_prac INTEGER REFERENCES Pracownicy,
data DATE NOT NULL,
czas FLOAT CHECK (czas > 0),
stawka FLOAT CHECK (stawka> 0 AND stawka< 50.00)
)

CREATE TABLE *Dzial*
(kod_dz VARCHAR(10),
nazwa_dz VARCHAR(20),
siedziba VARCHAR(10),
kierownik INTEGER)