У мові *SQL* підтримуються такі типи даних:

***CHAR**[ACTER](n)* — текстовий рядок фіксованої довжини в *n* символів. Максимальне значення n — 254 символи. Константа такого типу задається в апострофах (‘). Причому значення ‘‘ дозволяє включити в структуру і сам апостроф (‘).

VARCHAR — текст структури змінної довжини. Її максимальний розмір залежить від СУБД — 256 - 8000 символів.

Аналогічний йому тип ***LONG**[VARCHAR]* — розміром до 16K або 2G, залежно від реалізації. Різні СУБД організують збереження даних цих двох типів або в самій таблиці, або, найчастіше, в *окремому файлі*. А в таблиці зберігаються тільки *посилання на зсув* конкретного значення.

***BIT**(n)* — бітовий рядок *n*-ї довжини. Призначена для збереження двійкових значень як *єдиного цілого* (коди програм, зображення і т.д.).

Сьогодні більшість СУБД замість цього типу даних використовують тип ***BLOB*** — *Binary Large OBject*, який зберігається аналогічно типу *VARCHAR*.

***DEC**[IMAL](n,m)* — десяткове число в *дійсному виді*, тобто з явною десятковою крапкою. *n* — загальна кількість розрядів числа, *m* — кількість розрядів після крапки. Причому $m \le n$. Максимальна кількість розрядів *p* залежить від СУБД, але *p* повинно бути не менше *n*.

***NUMERIC**(n,m)* — аналогічно *DEC*, крім того, що *p* може бути не більше *n*.

***INT**[EGER]* — десяткове ціле. Його максимальний розмір і, відповідно, значення залежать від СУБД.

SMALLINT — підтримується не всіма СУБД. Аналогічний *INT*, але в два рази менше.

***FLOAT**(n)* — десяткове число з плаваючою крапкою, представлене *в експонентній* формі $x \times 10^y$, *де* n — кількість розрядів *числа* x, максимальне значення якого, як і максимальне значення *числа* y, залежить від конкретної реалізації.

REAL — збігається з *FLOAT*, за винятком того, що значення *n* дорівнює *максимальному* і встановлюється залежно від реалізації.

DOUBLE — у 2 рази більше *REAL*.

DATE
TIME } — назви говорять самі за себе. Характерним є те, що практично всі СУБД дозволяють установлювати формат введення і відображення даних цих типів, хоча кожна СУБД зберігає такі дані у своєму *внутрішньому специфічному форматі*, незалежному від формату відображення.

Домени

Для створення домену використовуються такі команди:

CREATE DOMAIN *ім'я_домену* *тип_поля*
[DEFAULT *значення*]
[*список обмежень*];

Приклад

```
CREATE DOMAIN Color CHAR(8)
    DEFAULT '???'
    [CONSTRAINT Valid_Colors]
    CHECK(VALUE IN('червоний', 'жовтий', 'зелений', 'синій', '???'));
```

Змінити або видалити значення за замовчуванням і список обмежень даного домену можна командою, аналогічною наведеній, ALTER DOMAIN.

Знищити існуючий домен можна командою

DROP DOMAIN *домен* *опція*;

де *опція* може приймати значення:

RESTRICT — домен *не буде* знищений, якщо на нього *є хоча б одне посилання* в таблицях;

CASCADE — команда буде виконана *не тільки для самого домена*, але й для таблиць з атрибутами, визначеними на його основі: тип таких полів буде перепризначений на *базовий тип* домену.

Підтримка користувальницьких типів даних в СУБД PostgreSQL

В СУБД ***PostgreSQL***, окрім доменів, реалізовано можливість створення *істинно користувальницьких типів* даних (*User Defined Type — UDT*) з визначенням їхніх справжніх реляційних доменів шляхом перелічення всіх можливих значень типу даних, що створюється:

CREATE TYPE *Користувальницький_тип* AS ENUM (*значення*_1, *значення*_2, …);

де ENUM — означає перелічення якихось констант без вказівки їхнього будь-якого базового типу. Наприклад:

CREATE TYPE color AS ENUM ('червоний', 'помаранчевий', 'жовтий', 'зелений', 'синій', 'фіолетовий');

Перелічуваний тип даних — це множина впорядкованих елементів, тому для роботи з ним призначені оператори порядку або порівняння (>, <, >=, <=) та спеціальні функції:

Таблиця 1 – Функції роботи з перелічуваним типом

Функція	**Опис**	**Приклад**
enum_first (anyenum)	Повертає перший елемент множини	enum_first(null::rainbow) = red
enum_last (anyenum)	Повертає останній елемент множини	enum_last(null::rainbow) = purple
enum_range (anyenum, anyenum)	Повертає діапазон елементів, які розташована між вказаними двома елементами множини	enum_range('orange'::rainbow, 'green'::rainbow) = {orange,yellow,green}
		enum_range(NULL, 'green'::rainbow) = {red,orange,yellow,green}
		enum_range('orange'::rainbow, NULL) = {orange,yellow,green,blue,purple}

Складені типи даних

Складні типи визначаються командою:

CREATE TYPE *Назва_типу* AS (*Назва_атрибуту Тип_атрибуту* [, ...])

Наприклад, для створення типу *Address*, який включає дані про місто, вулицю, будинок та квартиру, можна виконати команду:

```
CREATE TYPE Address AS
    ( City VARCHAR, Street VARCHAR, House SMALLINT, Flat SMALLINT);
```

Тепер до таблиці *Student* можна додати новий атрибут *Address* відповідного типу:

```
ALTER TABLE Student ADD COLUMN Address ADDRESS;
```

Для внесення даних до атрибуту складного типу використовується два варіанти:

'(*значення*_1 , *значення*_2 , …)'

або

ROW(*значення*_1 ,*значення*_2 , …),

де *значення*_1, *значення*_2, … — значення *атрибуту*_1, *атрибуту*_2 і т.д. опису складеного типу.

В запитах на отримання даних для доступу до атрибутів складного типу використовуються дужки, наприклад:

```
SELECT (Address).City FROM Student;
```

В СУБД *PostgreSQL* створено багато складних типів:

— типи, які автоматично встановлюються в процесі інсталяції, наприклад, геометричні, мережеві та бітові;

— типи, підтримку яких необхідно встановлювати окремо, наприклад, багатомірні куби та дерева.

Геометричні типи даних

Тип	**Розмірність**	**Опис**	**Форма представлення**
point	16 байт	Крапка на площині	(x,y)
line	32 байти	Нескінчена лінія (не повністю реалізована)	((x1,y1),(x2,y2))
lseg	32 байти	Обмежений лінійний сегмент (відрізок)	((x1,y1),(x2,y2))
box	32 байти	Прямокутник	((x1,y1),(x2,y2))
path	16+16n байт	Замкнутий шлях (теж саме, що й багатокутник)	((x1,y1),...)
		Відкритий шлях (ламана лінія)	[(x1,y1),...]
polygon	40+16n байт	Багатокутник (теж саме, що й замкнутий шлях)	((x1,y1),...)
circle	24 байти	Коло	<(x,y), r> (*центр* і *радіус*)

Функції обробки геометричних типів даних

Функція	Опис	Приклад
area(object)	Регіон	area(box '((0,0),(1,1))')
center(object)	Центр	center(box '((0,0),(1,2))')
diameter(circle), radius(circle)	Діаметр та радіус кола	diameter(circle '((0,0),2.0)')
height(box), width(box)	Вертикальний, горизонтальний розмір прямокутнику	height(box '((0,0),(1,1))')
isclosed(path)	Шлях замкнутий?	isclosed(path '((0,0),(1,1),(2,0))')
length(object)	Довжина	length(path '((-1,0),(1,0))')
npoints(path)	Кількість точок шляху	npoints(path '[(0,0),(1,1),(2,0)]')
npoints(polygon)	Кількість точок полігону	npoints(polygon '((1,1),(0,0))')
pclose(path), popen(path)	Замкнути шлях / Розімкнути шлях	pclose(path '[(0,0),(1,1),(2,0)]')

Для роботи з даними, які описують елементи обчислювальних мереж, зокрема мережевих адрес, в діалекті *PostgreSQL* було створено відповідні складені типи:

Типи даних мережевих адрес

Тип	Розмірність	Опис	Приклад
cidr	7 або 19 байт	IPv4 та IPv6 мережі	192.168.100.128/25 10.1.2.3/32
inet	7 або 19 байт	IPv4 та IPv6 host-вузли та мережі	
macaddr	6 байт	MAC адреси	'08002b:010203' '08:00:2b:01:02:03'

Додаткові структурні елементи

До таких елементів відносяться ***багатовимірні масиви*** (*array*), які є і додатковим елементом, і додатковим типом даних.

Загальний синтаксис опису масиву визначається двома варіантами:

'{ *значення*_1 *символ значення*_2 *символ* ... }'

або

ARRAY[*значення*_1 *символ значення*_2 *символ* ...],

де *символ* — символ-роздільник.

Для розділення стандартних типів використовується кома ‘,’, виключаючи тип BOX, для якого використовується символ ‘;’. Кожний елемент *значення* — це константа або підмасив. Наприклад, представлення масиву в операціях внесення даних може бути таким:

'{10000, 10000, 10000, 10000}';

'{{"Meeting", "Lunch"}, {"Training", "Presentation"}}');

або таким:

ARRAY[10000, 10000, 10000, 10000],

ARRAY[['Meeting', 'Lunch'], ['Training', 'Presentation']]);

Існують три типи модифікації елементів масиву:

повна модифікація — весь зміст масиву замінюється новими даними, які задано масивом-константою;

модифікація зрізу — модифікується лише інтервальна підмножина елементів;

модифікація елементу — модифікується окремий елемент за індексом.

Приклади зміни елементів масиву:

```
UPDATE    Sal_emp  SET   Pay_by_Quarter = '{25000,25000,27000,27000}'
    WHERE     Name = 'Carol';

UPDATE    Sal_emp  SET   Pay_by_Quarter = ARRAY[25000,25000,27000,27000]
    WHERE     Name = 'Carol';

UPDATE    Sal_emp  SET   Pay_by_Quarter[1:2] = '{27000,27000}'
    WHERE     Name = 'Carol';

UPDATE    Sal_emp  SET   Pay_by_Quarter[4] = 15000
    WHERE     Name = 'Bill';
```

Для обробки масивів СУБД *PostgreSQL* пропонує оператори та функції, представлені в наступних таблицях.

Оператори обробки масивів

Оператор	Опис	Приклад використання
=	дорівнює	ARRAY[1,2,3] = ARRAY[1,2,3] = true
<>	не дорівнює	ARRAY[1,2,3] <> ARRAY[1,2,4] = true
<	менше ніж	ARRAY[1,2,3] < ARRAY[1,2,4] = true
>	більше ніж	ARRAY[1,4,3] > ARRAY[1,2,4] = true
<=	менше ніж або дорівнює	ARRAY[1,2,3] <= ARRAY[1,2,3] = true

Оператор	Опис	Приклад використання
>=	більше ніж або дорівнює	ARRAY[1,4,3] >= ARRAY[1,4,3] = true
@> <@	елементи одного масиву містяться в другому	ARRAY[1,4,3] @> ARRAY[3,1] = true ARRAY[2,7] <@ ARRAY[1,7,4,2,6] = true
&&	перекриття	ARRAY[1,4,3] && ARRAY[2,1] = true
||	конкатенація масивів	ARRAY[1,2,3] || ARRAY[4,5,6] = {1,2,3,4,5,6} ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] = {{1,2,3},{4,5,6},{7,8,9}}
||	Включення елементу до масиву	3 || ARRAY[4,5,6] = {3,4,5,6}

Функції обробки масивів

Функція	Опис	Приклад використання
array_append (anyarray, anyelement)	Додати елемент до масиву	array_append(ARRAY[1,2], 3) = {1,2,3}
array_cat (anyarray, anyarray)	Об`єднати два масиви	array_cat(ARRAY[1,2, 3], ARRAY[4,5])= {1,2,3,4,5}
array_ndims (anyarray)	Отримати розмірність масиву	array_ndims(ARRAY[[1,2,3], [4,5,6]])=2
array_dims (anyarray)	Отримати текстове представлення розмірності масиву	array_dims(ARRAY[[1,2,3], [4,5,6]]) = [1:2][1:3]

Функція	**Опис**	**Приклад використання**
array_fill (anyelement, int[], [, int[]])	Отримати масив з ініціалізацією елементів	array_fill(7, ARRAY[3], ARRAY[2]) = [2:4]={7,7,7}
array_length (anyarray, int)	Отримати розмірність масиву вказаного рівня	array_length(array[1,2, 3], 1) = 3
array_lower (anyarray, int)	Отримати найменшу розмірність вказаного рівня масиву	array_lower('[0:2]= {1,2,3}'::int[], 1) = 0
array_prepend (anyelement, anyarray)	Додати елемент в початок масиву	array_prepend(1, ARRAY[2,3]) = {1,2,3}
array_to_string (anyarray, text)	Отримати текстову строку з елементів масиву, використовуючи вказаний символ-роздільник	array_to_string(ARRAY[1, 2, 3], '~^~') = 1~^~2~^~3
array_upper (anyarray, int)	Отримати найбільшу розмірність вказаного рівня масиву	array_upper(ARRAY[1, 2,3,4], 1) = 4
string_to_array (text, text)	Отримати масив з текстової строки, використовуючи вказаний символ-роздільник	String_to_array('xx~^~yy~^~zz', '~^~') = {xx,yy,zz}
unnest(anyarray)	Перетворити масив в рядки таблиці	Unnest(ARRAY[1,2])

Послідовність

CREATE SEQUENCE *назва*;

В цьому найпростішому варіанті буде створено таблицю з іменем назва, до єдиного кортежу якої за замовчуванням будуть записані такі значення: в поле *Ім'я* — назва, поля *Крок*, *Мінімальне* і *Початкове_значення* отримають значення 1, а поле *Максимальне_значення* — значення $\left(2^{63}-1\right)$ або 9 223 372 036 854 775 807, що можна перевірити командою

SELECT * FROM *назва*;

Якщо необхідно задати інші характеристики послідовності, то для цього використовується більш розширений синтаксис команди CREATE SEQUENCE:

CREATE [TEMP[ORARY]] SEQUENCE *назва*
[INCREMENT [BY] *приріщення*]
[MINVALUE *мінімальне_значення* | NO MINVALUE]
[MAXVALUE *максимальне_значення* | NO MAXVALUE]
[START [WITH] *початкове_значення*];

Необхідно мати на увазі, що при негативному значенні прирощення поля *Максимальне* і *Початкове_значення* за замовчуванням отримають значення (-1), а поле *Мінімальне_значення* — $\left(-2^{63}+1\right)$.

Функція **NEXTVAL(*назва*)** змінює поточне значення послідовності „*назва*“ на значення прирощення та повертає нове значення у вигляді величини типу integer. *Саме ця функція використовується при автоматичній генерації ідентифікаторів сутностей.*

Функція **CURRVAL(*назва*)** повертає значення, отримане функцією NEXTVAL останнім для послідовності „*назва*“ у поточній сесії. Якщо функція NEXTVAL ніколи не викликалась для цієї послідовності в цій сесії, то буде згенеровано повідомлення про помилку.

Функція **LASTVAL** повертає значення, яке найбільш часто поверталось функцією NEXTVAL у поточній сесії. Ця функція ідентична CURRVAL, за винятком того, що замість використання імені послідовності як аргументу вона витягує значення останньої послідовності, для якої була використана функція NEXTVAL у поточній сесії. Якщо функція NEXTVAL ніколи не викликалася в цій сесії, то буде згенеровано повідомлення про помилку.

SETVAL(*назва, нове_значення*) виконує „скидання“ лічильника послідовності „*назва*“. Це означає, що при наступному виклику функція NEXTVAL поверне та/або згенерує і поверне значення (*нове_значення* + 1), а функція CURRVAL — значення *нове_значення*.

Для видалення послідовності або декількох послідовностей одночасно використовується команда:

DROP SEQUENCE *назва1* [,*назва2, ...*];