

Postgresql Advanced Features

Reja:

1. JSON, JSONB
2. Indexing
3. Postgres Cron
4. CREATING TABLE PARTITIONS
5. Vertical table

JSON

JSON JavaScript Object Notation degan ma`noni anglatadi. JSON **key-value** juftliklaridan tashkil topgan ochiq standart formatdir. JSON-dan asosiy foydalanish server va veb-ilova o`rtasida ma`lumotlarni uzatishdir. Boshqa formatlardan farqli o`laroq, JSON inson o`qiy oladigan matndir. PostgreSQL 9.2 versiyasidan boshlab mahalliy JSON ma`lumotlar turini qo`llab-quvvatlaydi. U JSON ma`lumotlarini manipulyatsiya qilish uchun ko`plab fuksiyalar va operatorlarni taqdim etadi.

JSON and JSONB Operators

Operator	Right Operand Type	Description	Example
->	int	JSON massiv elementini olish	'[1,2,3]':::json->2
->	text	JSON obyekt maydonini olish	'{"a":1,"b":2}':::json->'b'
->>	int	JSON massiv elementini matn sifatida olish	'[1,2,3]':::json->>2
->>	text	JSON obyekt maydonini matn sifatida olish	'{"a":1,"b":2}':::json->>'b'
#>	array of text	Belgilangan yo'lda JSON obyektini olish	'{"a":[1,2,3],"b":[4,5,6]}':::json#>'{a,2}'
#>>	array of text	Belgilangan yo'lda JSON obyektini matn sifatida olish	'{"a":[1,2,3],"b":[4,5,6]}':::json#>>'{a,2}'

JSON Functions

Function	Return Type	Description	Example	Example Result
<code>array_to_json</code> (anyarray [,pretty_bool])	json	Returns the array as JSON. A PostgreSQL multidimensional array becomes a JSON array of arrays. Line feeds will be added between dimension 1 elements if pretty_bool is true.	<code>array_to_json('{{1,5},{99,100}}':int[])</code>	<code>[[1,5],[99,100]]</code>
<code>row_to_json</code> (record [, pretty_bool])	json	Returns the row as JSON. Line feeds will be added between level 1 elements if pretty_bool is true.	<code>row_to_json</code> (row(1,'foo'))	<code>{"f1":1,"f2":"foo"}</code>
<code>to_json</code> (anyelement)	json	Returns the value as JSON. If the data type is not built in, and there is a cast from the type to json, the cast function will be used to perform the conversion. Otherwise, for any value other than a number, a Boolean, or a null value, the text representation will be used, escaped and quoted so that it is legal JSON.	<code>to_json</code> (<code>'Fred said "Hi."'::text</code>)	<code>"Fred said \"Hi.\""</code>

JSON Functions

<code>json_array_length(json)</code>	int	Eng tashqi JSON massivdagi elementlar sonini qaytaradi.	<code>json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]')</code>	5
<code>json_each(json)</code>	SETOF key text, value json	Eng tashqi JSON obyektini kalit/qiymat juftliklari to'plamiga kengaytiradi.	<code>select * from json_each('{"a":"foo", "b":"bar"}')</code>	key value -----+----- a "foo" b "bar"
<code>json_each_text(from_json json)</code>	SETOF key text, value text	Eng tashqi JSON obyektini kalit/qiymat juftliklari to'plamiga kengaytiradi. Qaytarilgan qiymat matn turi bo'ladi.	<code>select * from json_each_text('{"a":"foo", "b":"bar"}')</code>	key value -----+----- a foo b bar
<code>json_extract_path(from_json json, VARIADIC path_elems text[])</code>	json	path_elems tomonidan ko'rsatilgan JSON obyektini qaytaradi .	<code>json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4')</code>	<code>{"f5":99,"f6":"foo"}</code>
<code>json_extract_path_text(from_json json, VARIADIC path_elems text[])</code>	text	path_elems tomonidan ko'rsatilgan JSON obyektini qaytaradi .	<code>json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4','f6')</code>	foo

JSON Functions

<code>json_object_keys(json)</code>	SETOF text	JSON obyektidagi kalitlar to'plamini qaytaradi. Faqat "tashqi" ob'ekt ko'rsatiladi.	<code>json_object_keys('{ "f1": "abc", "f2": { "f3": "a", "f4": "b" } })</code>	<code>json_object_keys ----- ----- f1 f2</code>
<code>json_populate_record(base anyelement, from_json json, [, use_json_as_text bool=false])</code>	anyelement	from_json dagi ob'ektni ustunlari asos tomonidan belgilangan yozuv turiga mos keladigan qatorga kengaytiradi. Konvertatsiya eng yaxshi harakat bo'ladi; from_json da tegishli kaliti bo'lmagan bazadagi ustunlar null bo'lib qoladi. Agar ustun bir necha marta ko'rsatilgan bo'lsa, oxirgi qiymat ishlatiladi.	<code>select * from json_populate_record(null: :x, '{ "a": 1, "b": 2 }')</code>	<code>a b ---- 1 2</code>
<code>json_populate_recordset(base anyelement, from_json json, [, use_json_as_text bool=false])</code>	SETOF anyelement	from_json ichidagi ob'ektlarning eng tashqi to'plamini ustunlari baza tomonidan belgilangan yozuv turiga mos keladigan to'plamga kengaytiradi. Konvertatsiya eng yaxshi harakat bo'ladi; from_json da tegishli kaliti bo'lmagan bazadagi ustunlar null bo'lib qoladi. Agar ustun bir necha marta ko'rsatilgan bo'lsa, oxirgi qiymat ishlatiladi.	<code>select * from json_populate_recordset(n ull::x, '[{ "a": 1, "b": 2 }, { "a": 3, "b": 4 }]')</code>	<code>a b ---- 1 2 3 4</code>
<code>json_array_elements(json)</code>	SETOF json	JSON massivini JSON elementlari to'plamiga kengaytiradi.	<code>json_array_elements('[1, true, [2, false]]')</code>	<code>value ----- 1 true [2, false]</code>

JSONB

Table 9-41. Additional jsonb Operators

Operator	Right Operand Type	Description	Example
@>	jsonb	Does the left JSON value contain the right JSON path/value entries at the top level?	'{"a":1, "b":2}':::jsonb @> '{"b":2}':::jsonb
<@	jsonb	Are the left JSON path/value entries contained at the top level within the right JSON value?	'{"b":2}':::jsonb <@ '{"a":1, "b":2}':::jsonb
?	text	Does the string exist as a top-level key within the JSON value?	'{"a":1, "b":2}':::jsonb ? 'b'
?	text[]	Do any of these array strings exist as top-level keys?	'{"a":1, "b":2, "c":3}':::jsonb ? array['b', 'c']
?&	text[]	Do all of these array strings exist as top-level keys?	'["a", "b"]':::jsonb ?& array['a', 'b']
	jsonb	Concatenate two jsonb values into a new jsonb value	'["a", "b"]':::jsonb '["c", "d"]':::jsonb
-	text	Delete key/value pair or string element from left operand. Key/value pairs are matched based on their key value.	'{"a": "b"}':::jsonb - 'a'
-	integer	Delete the array element with specified index (Negative integers count from the end). Throws an error if top level container is not an array.	'["a", "b"]':::jsonb - 1
#-	text[]	Delete the field or element with specified path (for JSON arrays, negative integers count from the end)	'["a", {"b":1}]':::jsonb #- '{1,b}'

JSON

JSON ma'lumotlar turi bilan mashq qilish uchun MO da id va info deb nomlangan ustunlari bor bo'lgan **orders** jadvalini yaratib olamiz .

```
Query Editor  Query History
1 CREATE TABLE orders (
2   id serial NOT NULL PRIMARY KEY,
3   info json NOT NULL
4 );
```



Data Output		Explain	Messages	Notifications
	id		info	
▲	[PK] integer	✎	json	✎

orders jadvaliga bir nechta yangi ma'lumotlarni qo'shamiz.

Query Editor Query History

```
1 INSERT INTO orders (info)
2 VALUES ('{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'),
3         ('{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}'),
4         ('{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}');
```

Data Output Explain Messages Notifications

	id [PK] integer		info json	
1		1	{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}	
2		2	{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}	
3		3	{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}	

Quyidagi so`rov barcha customerlarni JSON shaklida olish uchun -> operatoridan foydalanadi :

```
SELECT info -> 'customer'  
AS customer FROM orders;
```

Query Editor		Query History	
1	SELECT info -> 'customer' AS customer		
2	FROM orders;		
Data Output		Explain	Messages
customer		Notifications	
	json		
1	"Lily Bush"		
2	"Josh William"		
3	"Mary Clark"		

Quyidagi so`rov barcha customerlarni JSON shaklida olish uchun ->> operatoridan foydalanadi :

```
SELECT info ->> 'customer'  
AS customer FROM orders;
```

Query Editor		Query History	
1	SELECT info ->> 'customer' AS customer		
2	FROM orders;		
Data Output		Explain	Messages
customer text			
1	Lily Bush		
2	Josh William		
3	Mary Clark		

Indexing

Indekslar - bu ma'lumotlar bazasi qidiruvi ma'lumotlarni qidirishni tezlashtirish uchun foydalanishi mumkin bo'lgan maxsus qidirish jadvallari. Oddiy qilib aytganda, indeks jadvaldagi ma'lumotlarga ko'rsatgichdir. Ma'lumotlar bazasidagi indeks kitobning orqa qismidagi mundarijaga juda o'xshaydi.

Misol uchun, agar siz maʼlum bir mavzuni muhokama qiladigan kitobning barcha sahifalariga havola qilmoqchi boʻlsangiz, avval barcha mavzularni alifbo tartibida koʻrsatadigan indeksga murojaat qilishingiz va keyin bir yoki bir nechta maxsus sahifa raqamlariga murojaat qilishingiz kerak.

Indeks SELECT soʻrovlarini va WHERE bandlarini tezlashtirishga yordam beradi; ammo, u UPDATE va INSERT iboralarini bilan maʼlumotlarni kiritishni sekinlashtiradi. Indekslar maʼlumotlarga taʼsir qilmasdan yaratilishi yoki oʻchirilishi mumkin.

Indeks yaratish indeksni nomlash, jadval va qaysi ustun yoki ustunlar indekslanishini belgilash hamda indeksning o'sish yoki kamayish tartibida ekanligini ko'rsatish imkonini beruvchi CREATE INDEX operatorini o'z ichiga oladi.

Indekslar, shuningdek, UNIQUE chekloviga o'xshash noyob bo'lishi mumkin, chunki indeks indeks joylashgan ustunlar yoki ustunlar birikmasidagi takroriy yozuvlarni oldini oladi.

Indexing

CREATE INDEX ning asosiy sintaksisi quyidagicha -

```
CREATE INDEX index_name ON table_name;
```

Index turlari

PostgreSQL bir nechta indeks turlarini taqdim etadi: **B-tree**, **Hash**, **GiST**, **SP-GiST** va **GIN**. Har bir Indeks turi so`rovlarning har xil turlariga eng mos keladigan boshqa algoritmdan foydalanadi. **CREATE INDEX** buyrug`i eng keng tarqalgan vaziyatlarga mos keladigan **B-tree** indekslarini yaratadi.

Bir ustunli indekslar

Bir ustunli indeks faqat bitta jadval ustuni asosida yaratilgan indeksdir. Asosiy sintaksis quyidagicha

```
CREATE INDEX index_name ON table_name (column_name);
```

Index turlari

Ko`p ustunli indekslar

Ko`p ustunli indeks jadvalning bir nechta ustunlarida aniqlanadi. Asosiy sintaksis quyidagicha -

```
CREATE INDEX index_name ON table_name (column1_name, column2_name);
```


Index turlari

Noyob indekslar

Noyob indekslar nafaqat ishlash, balki ma`lumotlar yaxlitligi uchun ham qo`llaniladi. Noyob indeks jadvalga takroriy qiymatlarni kiritishga ruxsat bermaydi. Asosiy sintaksis quyidagicha -

```
CREATE UNIQUE INDEX index_name on table_name (column_name);
```

Index turlari

Qisman indekslar

Qisman indeks - jadvalning kichik to'plami ustiga qurilgan indeks; kichik to'plam shartli ifoda bilan aniqlanadi (qisman indeksning predikati deb ataladi). Indeks faqat predikatni qondiradigan jadval qatorlari uchun yozuvlarni o'z ichiga oladi. Asosiy sintaksis quyidagicha -

```
CREATE INDEX index_name on table_name (conditional_expression);
```

Yashirin indekslar

Yashirin indekslar – ob'ekt yaratilganda ma'lumotlar bazasi serveri tomonidan avtomatik ravishda yaratiladigan indekslar. Indekslar avtomatik ravishda asosiy kalit cheklovlari va noyob cheklovlar uchun yaratiladi.

Indexing

DROP INDEX buyrug`i

Indeksni PostgreSQL **DROP** buyrug`i yordamida o`chirish mumkin .

Asosiy sintaksis quyidagicha -

```
DROP INDEX index_name;
```

Indekslerden qachon qochish kerak?

Indeksler ma'lumotlar bazasining ish faoliyatini yaxshilash uchun mo'ljallangan bo'lsa-da, ulardan qochish kerak bo'lgan holatlar mavjud. Quyidagi ko'rsatmalar indeksdan foydalanish qachon qayta ko'rib chiqilishi kerakligini ko'rsatadi –

- Indeksleri kichik jadvallarda ishlatmaslik kerak.
- Tez-tez, katta partiyalarni yangilash yoki kiritish operatsiyalariga ega jadvallarda ishlatmaslik kerak.
- Ko'p sonli NULL qiymatlarni o'z ichiga olgan ustunlarda indekslerden foydalanmaslik kerak.
- Tez-tez manipulyatsiya qilinadigan ustunlar indekslanmasligi kerak.

Postgres Cron (*pg_cron*)

pg_cron - bu PostgreSQL (9.5 yoki undan yuqori) uchun oddiy cronga asoslangan ish rejalashtiruvchisi, kengaytma sifatida ma'lumotlar bazasida ishlaydi. U oddiy cron bilan bir xil sintaksisdan foydalanadi, ammo u PostgreSQL buyruqlarini to'g'ridan-to'g'ri ma'lumotlar bazasidan rejalashtirish imkonini beradi. **pg_cron** bir nechta ishni parallel ravishda bajarishi mumkin, lekin u bir vaqtning o'zida ishning ko'pi bilan bitta nusxasini bajaradi. Jadvalda standart cron sintaksisi qo'llaniladi, unda * "har vaqt oralig'ida ishga tushirish" degan ma'noni anglatadi va ma'lum raqam "lekin faqat shu vaqtda" degan ma'noni anglatadi:

Postgres Cron (*pg_cron*)

```
-- Delete old data on Saturday at 3:30am (GMT)
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$);
schedule ----- 42

-- Vacuum every day at 10:00am (GMT) SELECT cron.schedule('0 10 * * *', 'VACUUM');
schedule ----- 43

-- Stop scheduling a job SELECT cron.unschedule(43);
unsubscribe ----- t
```

https://access.crunchydata.com/documentation/pg_cron/1.2.0/

PostgreSQL-da jadval bo'limi nima?

Jadvalni bo'lish - bu katta jadvalni kichikroq kichik jadvallarga bo'lish amaliyotidir va har bir kichik jadval alohida CREATE TABLE buyruqlari yordamida yaratiladi. Shunday qilib, har safar ma'lumotlarni so'raganingizda, PostgreSQL katta jadvalga kirish o'rniga kichikroq ma'lumotlar to'plamini skanerlaydi va qayta ishlaydi. Shunday qilib, so'rovlar samaradorligi sezilarli darajada yaxshilanadi.

PostgreSQL o'rnatilgan qismlarga ajratishning uchta turini qo'llab-quvvatlaydi:

- Diapazonni bo'lish: ma'lumotlar qatorlari ma'lum diapazonga to'g'ri keladigan ustun qiymatlari asosida bo'limlarga taqsimlanadi.
- Ro'yxat bo'limi: Jadval har bir bo'limda qaysi asosiy qiymatlar paydo bo'lishini aniq ro'yxatlash orqali bo'linadi.
- Xeshni bo'lish: Qatorlar bo'lim kalitining xesh qiymatidan foydalangan holda barcha bo'limlar bo'ylab teng taqsimlanadi.

CREATING TABLE PARTITIONS

PostgreSQL-da bo`limlar bilan jadval yaratish uchun **PARTITION BY** bandidan foydalanish sintaksisi quyidagicha:

1. Diapazonga bo'lingan jadval yaratish uchun:

```
CREATE TABLE table_name  
table_definition  
PARTITION BY RANGE (expression);
```

Misol

```
CREATE TABLE city (  
  id int4 NOT NULL PRIMARY KEY,  
  name varchar(30) NOT NULL,  
  state varchar(20),  
  population int4,  
)  
PARTITION BY RANGE (id);
```

Keyin bo'limlarni alohida yaratishingiz kerak

```
CREATE TABLE city_id1 PARTITION OF city  
FOR VALUES FROM (MINVALUE) TO (10);  
  
CREATE TABLE city_id2 PARTITION OF city  
FOR VALUES FROM (10) TO (20);  
  
CREATE TABLE city_id3 PARTITION OF city  
FOR VALUES FROM (20) TO (30);  
  
CREATE TABLE city_id4 PARTITION OF city  
FOR VALUES FROM (30) TO (MAXVALUE);
```


CREATING TABLE PARTITIONS

2. Ro'yxat bo'lingan jadval yaratish uchun:

```
CREATE TABLE table_name  
table_definition  
PARTITION BY LIST (expression);
```

Misol

```
CREATE TABLE cities (  
    city_id bigserial NOT NULL,  
    name text NOT NULL,  
    population bigint  
)  
PARTITION BY LIST (left(lower(name), 1));
```

Keyin bo'lim yarating

```
CREATE TABLE cities_ab PARTITION OF cities  
FOR VALUES IN ('a', 'b');  
  
CREATE TABLE cities_cd PARTITION OF cities  
FOR VALUES IN ('c', 'd');  
  
CREATE TABLE cities_ef PARTITION OF cities  
FOR VALUES IN ('e', 'f');  
  
CREATE TABLE cities_gh PARTITION OF cities  
FOR VALUES IN ('g', 'h');
```

CREATING TABLE PARTITIONS

3. Xesh bo'lingan jadval yaratish uchun:

```
CREATE TABLE table_name  
table_definition  
PARTITION BY HASH (expression);
```

Misol

```
CREATE TABLE orders (  
  order_id bigint NOT NULL,  
  cust_id bigint NOT NULL,  
  status text  
)  
PARTITION BY HASH (order_id);
```

Bo'limlarni yarating:

```
CREATE TABLE orders_p1 PARTITION OF orders  
FOR VALUES WITH (MODULUS 4, REMAINDER 0);  
  
CREATE TABLE orders_p2 PARTITION OF orders  
FOR VALUES WITH (MODULUS 4, REMAINDER 1);  
  
CREATE TABLE orders_p3 PARTITION OF orders  
FOR VALUES WITH (MODULUS 4, REMAINDER 2);  
  
CREATE TABLE orders_p4 PARTITION OF orders  
FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```

Vertical table

PostgreSQL ma'lumotlar omborida ob'ekt ma'lumotlari garizontal tartibda ustunlarda saqlanadi. Ularni o'qiganimizda ham garizontal jadval shaklida ko'rinadi. Ammo, jadvaldagi ma'lumotlarni garizontal emas, vertical tarzda ko'rsatish kerak bo'lib qolgan vaziyatlar ham uchrab turadi.

Vertical table

id	maker	model	type
1	E	2113	PC
2	E	2112	PC
3	A	1752	Laptop

Horizontal table

id	1
maker	E
model	2113
type	PC
id	2
maker	E
model	2112
type	PC
id	3
maker	A
model	1752
type	Laptop

Vertical table

Jadvaldagi ma'lumotlarni garizontal emas, vertical tarzda ko'rsatish kerak bo'lib qolgan vaziyatlarda quyidagi amallardan foydalanasiz. \x buyrug'i yozilgandan so'ng **Expanded display is on.** Jadvallar ko'rinishini verticalga o'zgartiradi, uni bekor qilish uchun yana \x buyrug'i yoziladi va **Expanded display is off.** Jadvallarning vertical ko'rinishini o'chiradi.

```
computer_db=# select * from product limit 3;
 maker | model | type
-----+-----+-----
 B      | 1121  | PC
 A      | 1232  | PC
 A      | 1233  | PC
(3 rows)

computer_db=#
```

```
computer_db=# \x
Expanded display is off.
computer_db=#
```

```
computer_db=# \x
Expanded display is on.
computer_db=# select * from product limit 3;
-[ RECORD 1 ]
 maker | B
 model | 1121
 type  | PC
-[ RECORD 2 ]
 maker | A
 model | 1232
 type  | PC
-[ RECORD 3 ]
 maker | A
 model | 1233
 type  | PC

computer_db=#
```

E`TIBORINGIZ UCHUN RAHMAT!