

# Triggers, Lock and Privileges

# Reja:

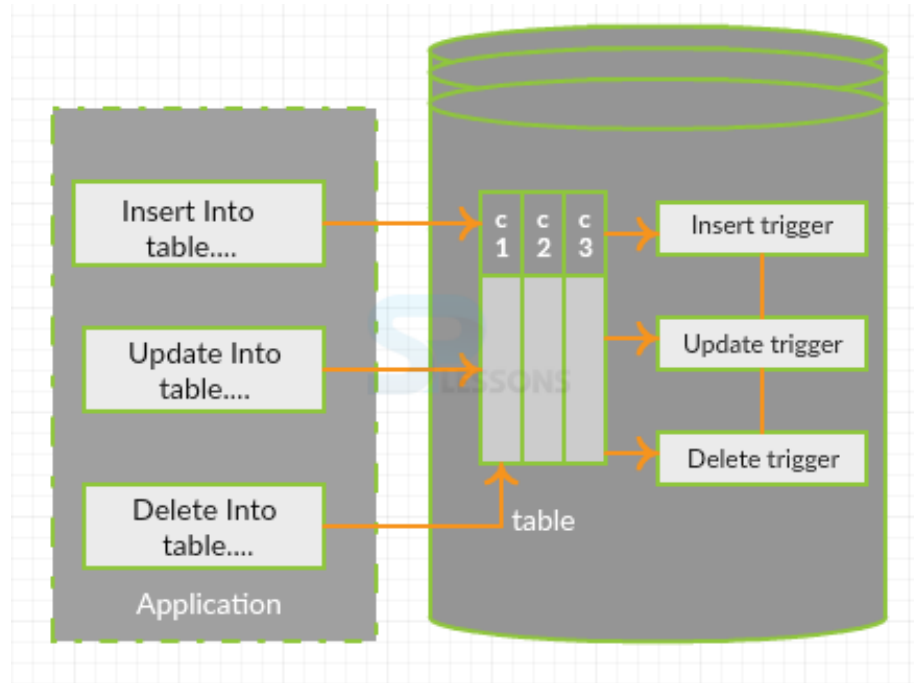
1. Triggers
2. Crypto
3. Lock
4. Privileges

# Triggers

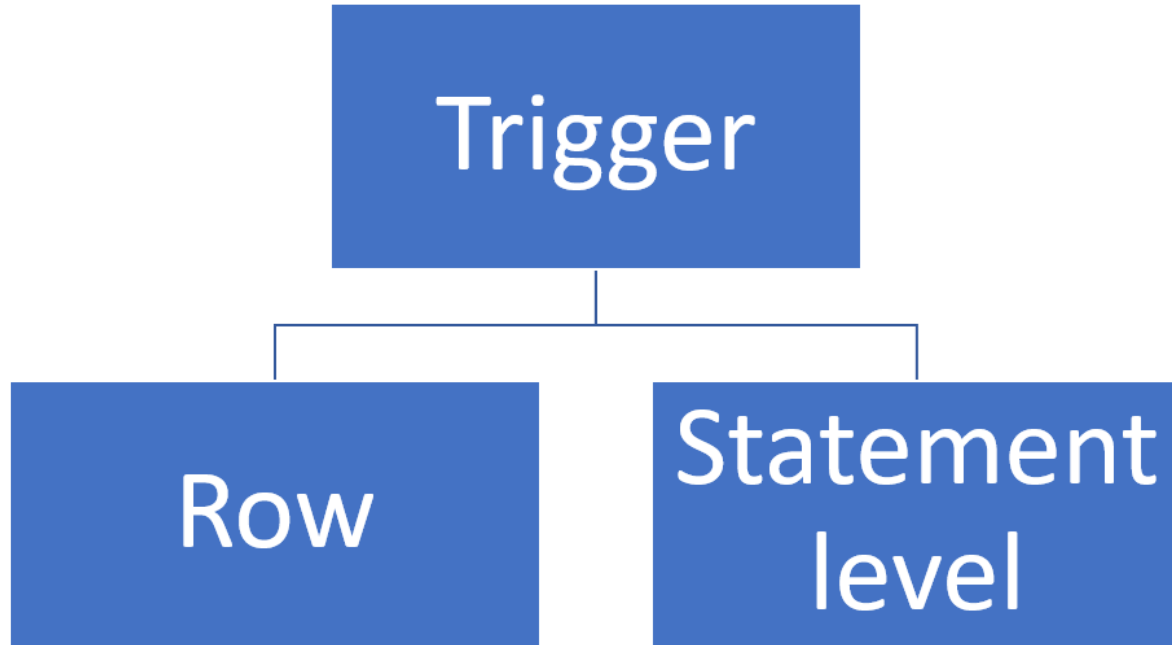
PostgreSQL da **trigger** – bu [INSERT](#), [UPDATE](#), [DELETE](#) yoki [TRUNCATE](#) amallari (trigger hodisalari) bajarilganda avtomatik tarzda ishga tushuvchi funksiyadir.

**Trigger** bu – foydalanuvchi funksiyasi ([user-defined function](#)) va jadval bilan bogʻlangan maxsus funksiya.

Yangi **Trigger** yaratish uchun dastlab trigger funksiya aniqlanadi (yaratiladi yoki mavjud boʻlsa belgilanadi) va keyinchalik trigger funksiya jadval bilan bogʻlanadi.



# Trigger



- [Creating Triggers](#)
- [Removing Triggers](#)
- [Altering Triggers](#)
- [Disabling Triggers](#)
- [Enabling Triggers](#)

# Creating triggers

PostgreSQL-da yangi trigger yaratish uchun quyidagi amallarni bajarish kerak:

- Birinchidan, [CREATE FUNCTION](#) buyrug`i yordamida trigger funksiyasini yaratish kerak .
- Ikkinchidan, trigger funksiyasini [CREATE TRIGGER](#) buyrug`i yordamida jadvalga bog`lash kerak.

# Create Function

Quyida trigger funktsiyasini yaratish sintaksisi tasvirlangan:

```
CREATE FUNCTION trigger_function()  
    RETURNS TRIGGER  
    LANGUAGE PLPGSQL  
AS $$  
BEGIN  
    -- trigger logic  
END;  
$$
```



# Create Trigger

**CREATE TRIGGER** buyrug'i yangi triggerni yaratadi. Quyida **CREATE TRIGGER** buyrug'ining asosiy sintaksisi tasvirlangan :

```
CREATE TRIGGER trigger_name  
  
    {BEFORE | AFTER} { event }  
  
ON table_name  
  
[FOR [EACH] { ROW | STATEMENT }]  
  
EXECUTE PROCEDURE trigger_function
```

Ushbu sintaksisda:

- 1) **CREATE TRIGGER** kalit so'zlardan keyin **trigger nomini** belgilang .
  - 2) Triggerni ishga tushish vaqtini belgilang. Bu **BEFORE** yoki **AFTER** bo'lishi mumkin, ya'ni biror event sodir bo'lishidan oldin yoki keyin ishga tushishi.
  - 3) Triggerni chaqiradigan event (hodisa) ni belgilang. Event **INSERT**, **DELETE**, **UPDATE** yoki **TRUNCATE** bo'lishi mumkin.
  - 4) **ON** kalit so'zdan keyin trigger bilan bog'langan jadval nomini belgilang .
  - 5) quyidagilar bo'lishi mumkin bo'lgan triggerlar turini belgilang:
    - **FOR EACH ROW** band tomonidan belgilangan qator darajasidagi trigger .
    - **FOR EACH STATEMENT** band tomonidan belgilangan bayonot darajasidagi trigger .Har bir satr uchun satr darajasidagi trigger ishga tushiriladi, har bir tranzaksiya uchun bayonot darajasidagi trigger ishga tushiriladi.
- Aytaylik, jadvalda 100 ta qator bo'lsa va **DELETE** hodisa sodir bo'lganda ishga tushiriladigan ikkita trigger mavjud .
- Agar **DELETE** hodisasi 100 qatorni o'chirib tashlasa, satr darajasidagi trigger har bir o'chirilgan qator uchun bir martadan 100 marta ishga tushadi. Boshqa tomondan, qancha satr o'chirilganidan qat'iy nazar, bayonot darajasidagi trigger bir marta ishga tushiriladi.
- Nihoyat**, **EXECUTE PROCEDURE** kalit so'zlardan keyin trigger funktsiyasi nomini belgilang .

# Create Trigger

```
create trigger  
my_first_trigger  
before  
insert  
on my_triggered_table  
for each row  
execute procedure  
my_first_trigger_function();
```

Name of your trigger.

Whether trigger should execute before or after triggering event.

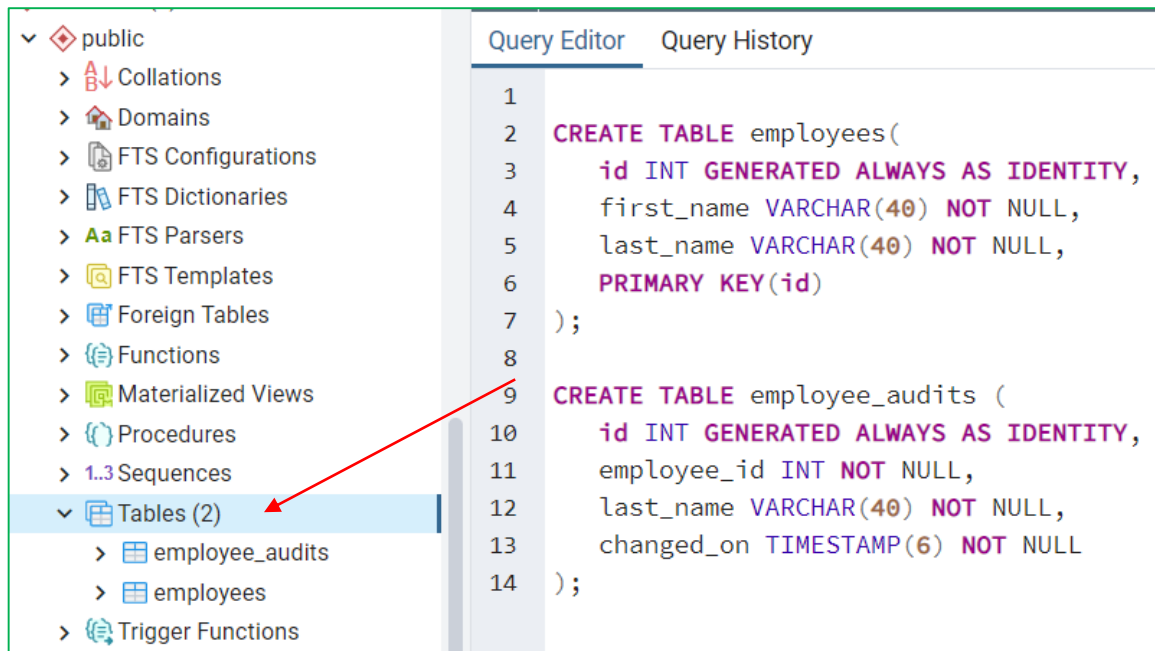
Event type(s) for this trigger.

Name of table to monitor.

Behavior if multiple rows are affected.

Function to run.

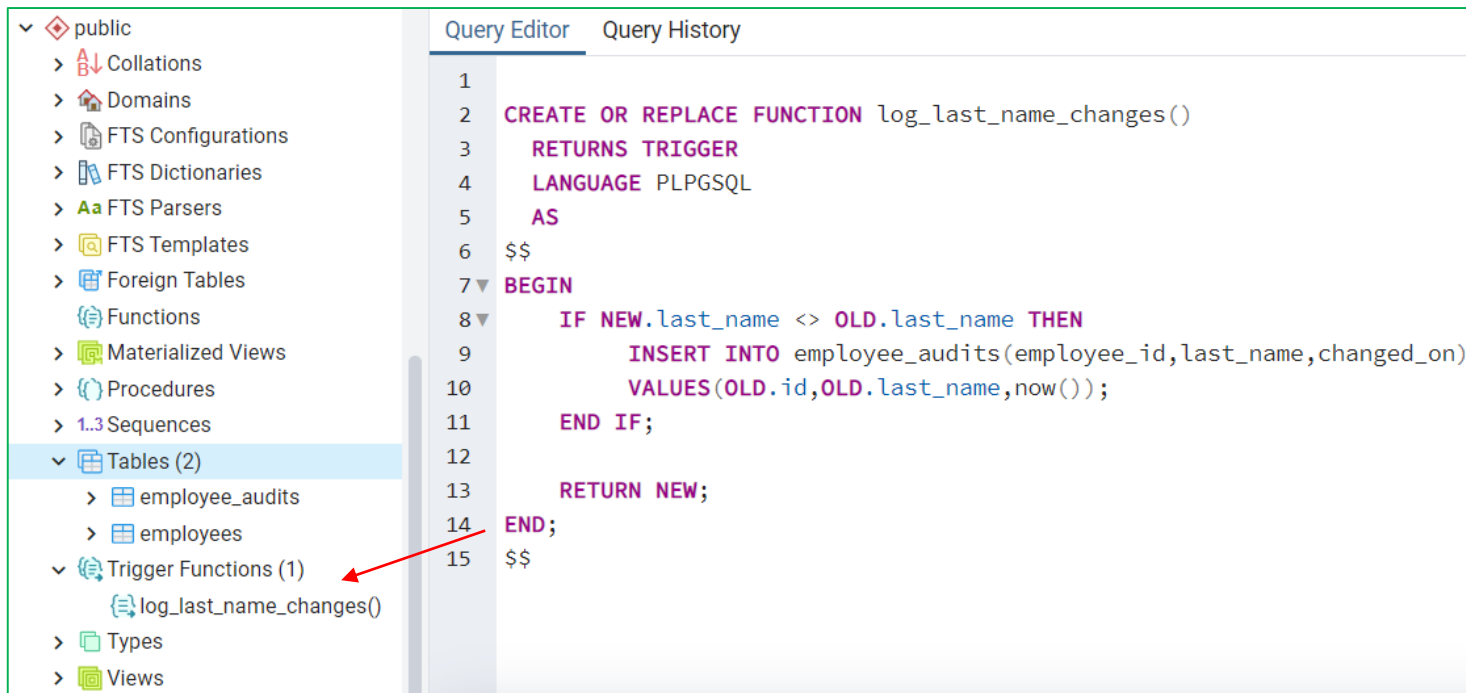
Triggerni yaxshiroq tushunish uchun **Query editor**da quyidagi bayonotni yozib **employees** va **employees** jadvalining **last\_name** ustunida o'zgarish bo'lganda o'zgarishlarni saqlab ketish uchun **employee\_audits** degan jadvallarni yaratib olamiz.



The screenshot displays a database management tool interface. On the left, a tree view shows the database structure under the 'public' schema. The 'Tables (2)' folder is expanded, showing 'employee\_audits' and 'employees'. A red arrow points from the 'employee\_audits' table to the corresponding table definition in the query editor. The query editor has two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing the following SQL code:

```
1  
2 CREATE TABLE employees(  
3     id INT GENERATED ALWAYS AS IDENTITY,  
4     first_name VARCHAR(40) NOT NULL,  
5     last_name VARCHAR(40) NOT NULL,  
6     PRIMARY KEY(id)  
7 );  
8  
9 CREATE TABLE employee_audits (  
10    id INT GENERATED ALWAYS AS IDENTITY,  
11    employee_id INT NOT NULL,  
12    last_name VARCHAR(40) NOT NULL,  
13    changed_on TIMESTAMP(6) NOT NULL  
14 );
```

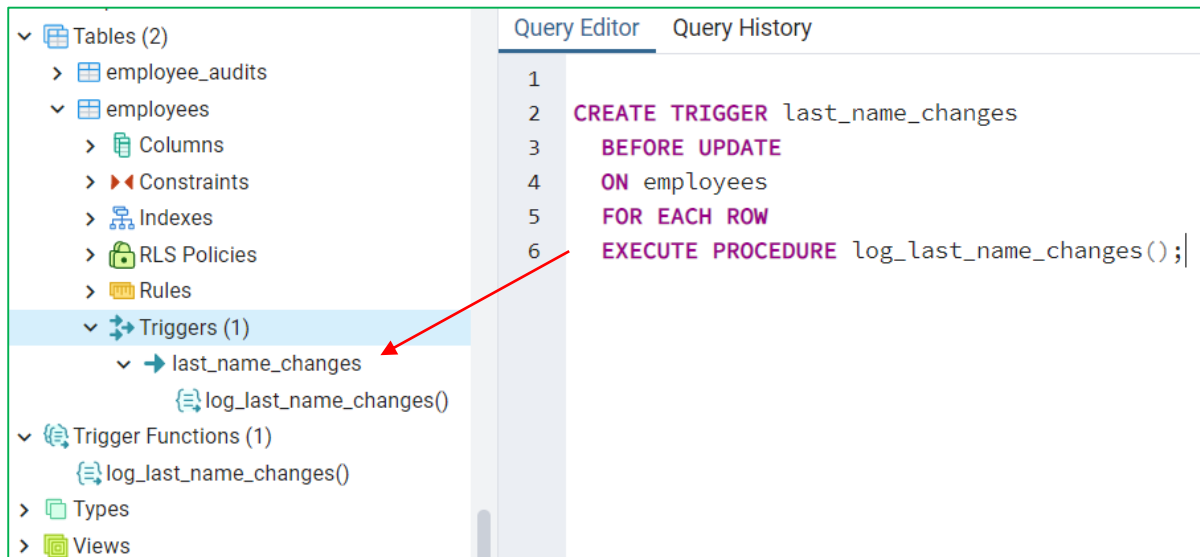
Employee jadvalida o'zgarish bo'lganda **employee\_audits** nomli jadvalga employee\_id,last\_name va o'zgarish bo'lgan vaqtni yozuvchi **trigger function**ni yaratib olamiz.



The screenshot displays a database management tool interface. On the left, a tree view shows the database structure under the 'public' schema. The 'Tables (2)' folder is expanded, showing 'employee\_audits' and 'employees'. The 'Trigger Functions (1)' folder is also expanded, showing a function named 'log\_last\_name\_changes()'. A red arrow points to this function. On the right, the 'Query Editor' tab is active, showing the SQL code for creating the function. The code is as follows:

```
1  
2 CREATE OR REPLACE FUNCTION log_last_name_changes()  
3 RETURNS TRIGGER  
4 LANGUAGE PLPGSQL  
5 AS  
6 $$  
7 BEGIN  
8     IF NEW.last_name <> OLD.last_name THEN  
9         INSERT INTO employee_audits(employee_id,last_name,changed_on)  
10        VALUES(OLD.id,OLD.last_name,now());  
11     END IF;  
12  
13     RETURN NEW;  
14 END;  
15 $$
```

**log\_last\_name\_changes()** trigger funksiyamizni **employees** jadvaliga bog`laymiz .Triggerimizning nomini **last\_name\_changes** deb nomlaymiz. employees jadvalining **last\_name** ustunining qiymati yangilanishidan oldin, o`zgarishlarni qayd qilish uchun trigger funksiyasi avtomatik ravishda chaqirilishi uchun quyidagi bayonot orqali triggerni yaratamiz.



The screenshot displays a database management tool interface. On the left, a tree view shows the database structure. The 'Triggers (1)' folder is expanded, and the 'last\_name\_changes' trigger is selected, indicated by a red arrow. The trigger's definition is shown as '{ log\_last\_name\_changes() }'. The main area on the right is the 'Query Editor', which contains the following SQL code:

```
1  
2 CREATE TRIGGER last_name_changes  
3 BEFORE UPDATE  
4 ON employees  
5 FOR EACH ROW  
6 EXECUTE PROCEDURE log_last_name_changes();
```

Quyidagi bayonot orqali **employees** jadvaliga bir nechta ma'lumot kiritib olamiz.

Query Editor	Query History
1	
2	<b>INSERT INTO</b> employees (first_name, last_name)
3	<b>VALUES</b> ('John', 'Doe');
4	
5	<b>INSERT INTO</b> employees (first_name, last_name)
6	<b>VALUES</b> ('Lily', 'Bush');



Data Output

Explain

Messages

Notifications

	id [PK] integer		first_name character varying (40)		last_name character varying (40)	
1		1	John		Doe	
2		2	Lily		Bush	

Yaratgan **last\_name\_changes** nomli triggerimizning ishlashini ko`rish uchun quyidagi bayonot orqali **employees** jadvalidagi id si 2, first\_name **Lily**, last\_name **Bush** bo`lgan employeening last\_name ni **Brown** ga o`zgartiramiz.





	Query Editor	Query History
1		
2	UPDATE	employees
3	SET	last_name = 'Brown'
4	WHERE	ID = 2;







	Data Output	Explain	Messages	Notifications
	<div><div>id</div><div>[PK] integer</div></div>		<div><div>first_name</div><div>character varying (40)</div></div>	<div><div>last_name</div><div>character varying (40)</div></div>
1		1	John	Doe
2		2	Lily	Brown








employees jadvali o'zgarish bo'lishidan oldin

Data Output	Explain	Messages	Notifications
 id [PK] integer 		first_name character varying (40) 	last_name character varying (40) 
1	1	John	Doe
2	2	Lily	Bush

employees jadvali o'zgarish bo'lgandan keyin

Data Output	Explain	Messages	Notifications
 id [PK] integer 		first_name character varying (40) 	last_name character varying (40) 
1	1	John	Doe
2	2	Lily	Brown

employees jadvalida o'zgarish bo'lishidan oldin biz yaratgan **last\_name\_changes** nomli trigger avtomatik ravishda ishladi va **log\_last\_name\_changes()** nomli **trigger function** **employee\_audits** jadvaliga o'zgarishdan oldingi ma'lumotni yozib qo'ydi.

Data Output		Explain	Messages	Notifications
	id integer 	employee_id integer 	last_name character varying (40) 	changed_on timestamp without time zone 
1	1	2	Bush	2022-01-09 05:15:40.901852



# Removing triggers

Triggerni o`chirish uchun quyidagi sintaksislardan foydalaniladi. Ushbu sintaksislardan foydalanib trigger o`chirilganda Data Basedan trigger jismoniy jihatdan o`chirib yuboriladi.

```
DROP TRIGGER [IF EXISTS] trigger_name  
ON table_name [ CASCADE | RESTRICT ];
```

```
DROP TRIGGER trigger_name;
```

# Altering triggers

**ALTER TRIGGER** bayonoti, triggerni qayta nomlash imkonini beradi. Quyida **ALTER TRIGGER** bayonotning sintaksisi ko'rsatilgan:

```
ALTER TRIGGER trigger_name  
ON table_name  
RENAME TO new_trigger_name;
```

Ushbu sintaksisda:

- Birinchidan, **ALTER TRIGGER** kalit soʻzdan keyin nomini oʻzgartirmoqchi boʻlgan trigger nomi belgilangan.
- Ikkinchidan, **ON** kalit soʻzdan keyin trigger bilan bogʻlangan jadval nomi belgilangan.
- Uchinchidan, **RENAME TO** kalit soʻzdan keyin triggerning yangi nomi belgilangan .

# Disabling triggers

Triggerni o`chirish uchun quyidagi **ALTER TABLE DISABLE TRIGGER** bayonotidan foydalaniladi:

```
ALTER TABLE table_name  
DISABLE TRIGGER trigger_name | ALL
```

Ushbu sintaksisda

- Birinchidan, **ALTER TABLE** kalit so`zlardan keyin trigger bog`langan jadval nomi belgilanadi .
- Ikkinchidan, **DISABLE TRIGGER** kalit so`zlardan keyin o`chirmoqchi bo`lgan trigger nomi belgilanadi yoki jadval bilan bog`liq barcha triggerlarni o`chirish uchun **ALL** kalit so`zdan foydalaniladi .

Triggerni o`chirib qo`yilganda, trigger hali ham Data Baseda mavjud. Biroq, trigger bilan bog`liq hodisa sodir bo`lganda, o`chirilgan trigger ishlamaydi.

# Enabling triggers

Triggerni yoki jadval bilan bog`langan barcha triggerlarni yoqish uchun quyidagi **ALTER TABLE ENABLE TRIGGER** bayonotdan foydalaniladi:

```
ALTER TABLE table_name  
ENABLE TRIGGER trigger_name | ALL;
```

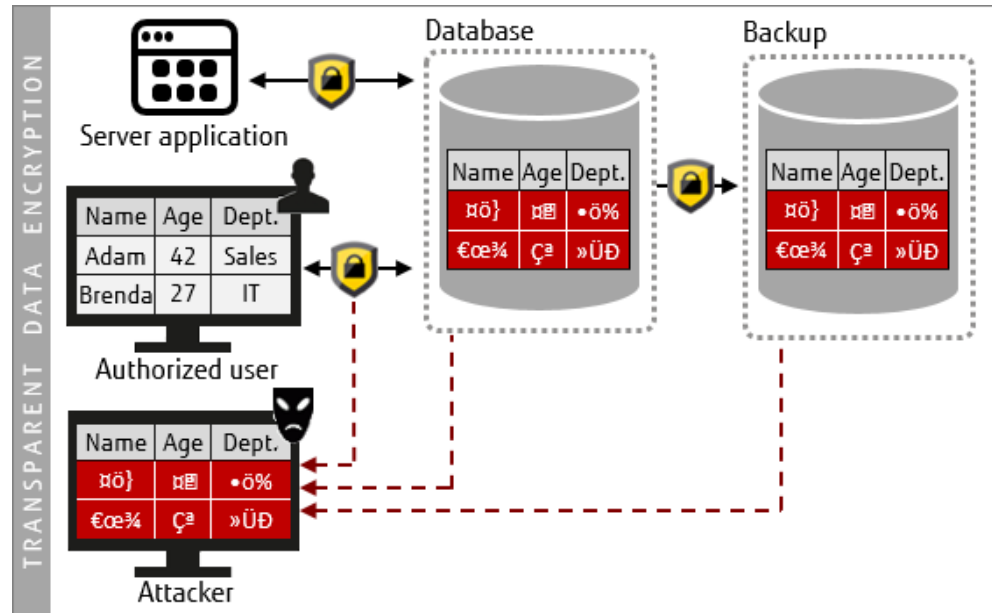
Ushbu sintaksisda:

- Birinchidan, yoqmoqchi bo`lgan trigger jadvalining nomi belgilanadi.
- Ikkinchidan, yoqmoqchi bo`lgan trigger nomi belgilanadi yoki jadval bilan bog`langan barcha triggerlarni yoqmoqchi bo`lsa, **ALL** kalit so`zidan foydalaniladi.

Trigger hodisasi sodir bo`lganda o`chirilgan trigger ishlamaydi. Uni ishlatish uchun uni yoqish kerak.

# Crypto

PostgreSQLda **pgcrypto** bizga foydalanuvchi hisob ma'lumotlarini xavfsiz saqlashga va tashqi hujumlardan himoya qilishga yordam beradi. Ya'ni bunda ma'lumotlar MO ga qandaydir algoritm asosida shifrlab saqlanadi.







# Crypto

Pgcryptoni yaxshiroq tushunish uchun **users** nomli jadvalga user ma'lumotlarini saqlashda pgcryptodan foydalanishni ko'rib chiqamiz. Buning uchun MO da users nomli jadval yaratamiz.

```
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  email TEXT NOT NULL UNIQUE,  
  password TEXT NOT NULL  
);
```



Data Output		Explain	Messages	Notifications
	<b>id</b> [PK] integer 	<b>email</b> character varying 	<b>password</b> text 	

# Crypto

**crypt** funksiya ikkita argument qabul qiladi:

1. Shifrlash uchun parol

2. Shifrlashda ishlatiladigan algoritm turi

Quyida shifrlash uchun ishlatiladigan **algoritm** turlari keltirilgan:

**Table F-17. Supported Algorithms for `crypt()`**

Algorithm	Max Password Length	Adaptive?	Salt Bits	Output Length	Description
bf	72	yes	128	60	Blowfish-based, variant 2a
md5	unlimited	no	48	34	MD5-based crypt
xdes	8	yes	24	20	Extended DES
des	8	no	12	13	Original UNIX crypt

# Crypto

Users jadvaliga id,email va password ma'lumotlari mavjud bo'lgan ob'ektlarni saqlashda ushning passwordini pgcrypto yordamida shifrlab saqlaymiz. Buning uchun avval pgcryptoni ishga tushirish kerak. Pgcryptoni ishga tushirish sintaksisi quyida keltirilgan:

Query Editor	Query History
1	
2	<code>create extension pgcrypto;</code>

Shundan so'ng **users** jadvaliga quyidagi so'rov orqali yangi ma'lumot qo'shamiz:

Query Editor	Query History
1	<code>INSERT INTO users (email, password) VALUES (</code>
2	<code>'johndoe@mail.com',</code>
3	<code>crypt('johnsrightpassword', gen_salt('bf'))</code>
4	<code>);</code>

Data Output	Explain	Messages	Notifications												
<table><tr><th></th><th>id</th><th>email</th><th>password</th></tr><tr><td></td><td>[PK] integer</td><td>character varying</td><td>text</td></tr><tr><td>1</td><td>1</td><td>johndoe@mail.com</td><td>\$2a\$06\$IE2JaDnXilyXQ1TAhuBzgOuA8QZgUD12NHuHkK8pAUKD6YSLuiJAY</td></tr></table>		id	email	password		[PK] integer	character varying	text	1	1	johndoe@mail.com	\$2a\$06\$IE2JaDnXilyXQ1TAhuBzgOuA8QZgUD12NHuHkK8pAUKD6YSLuiJAY			
	id	email	password												
	[PK] integer	character varying	text												
1	1	johndoe@mail.com	\$2a\$06\$IE2JaDnXilyXQ1TAhuBzgOuA8QZgUD12NHuHkK8pAUKD6YSLuiJAY												



# Crypto

Foydalanuvchini autentifikatsiya qilish uchun biz **crypt** dan yana foydalanamiz.

Query Editor Query History

```
1
2 SELECT id
3   FROM users
4  WHERE email = 'johndoe@mail.com'
5     AND password = crypt('johnsrightpassword', password);
```



Data Output	Explain	Messages	Notifications
<div><div></div><div>id</div><div>[PK] integer</div></div>			
1	1		

Query Editor Query History

```
1
2 SELECT id
3   FROM users
4  WHERE email = 'johndoe@mail.com'
5     AND password = crypt('johnswrongpassword', password);
```



Data Output	Explain	Messages	Notifications
<div><div></div><div>id</div><div>[PK] integer</div></div>			

# Locks in PostgreSQL

*Locks, Exclusive Locks* yoki *Write Locks* foydalanuvchilar tomonidan qatorni yoki butun jadvalni o'zgartirishni oldini oladi. UPDATE va DELETE orqali o'zgartirilgan qatorlar tranzaksiya davomida avtomatik tarzda bloklanadi. Bu tranzaksiya amalga oshirilmaguncha yoki orqaga qaytarilmaguncha boshqa foydalanuvchilarning qatorni o'zgartirishiga yo'l qo'ymaydi.

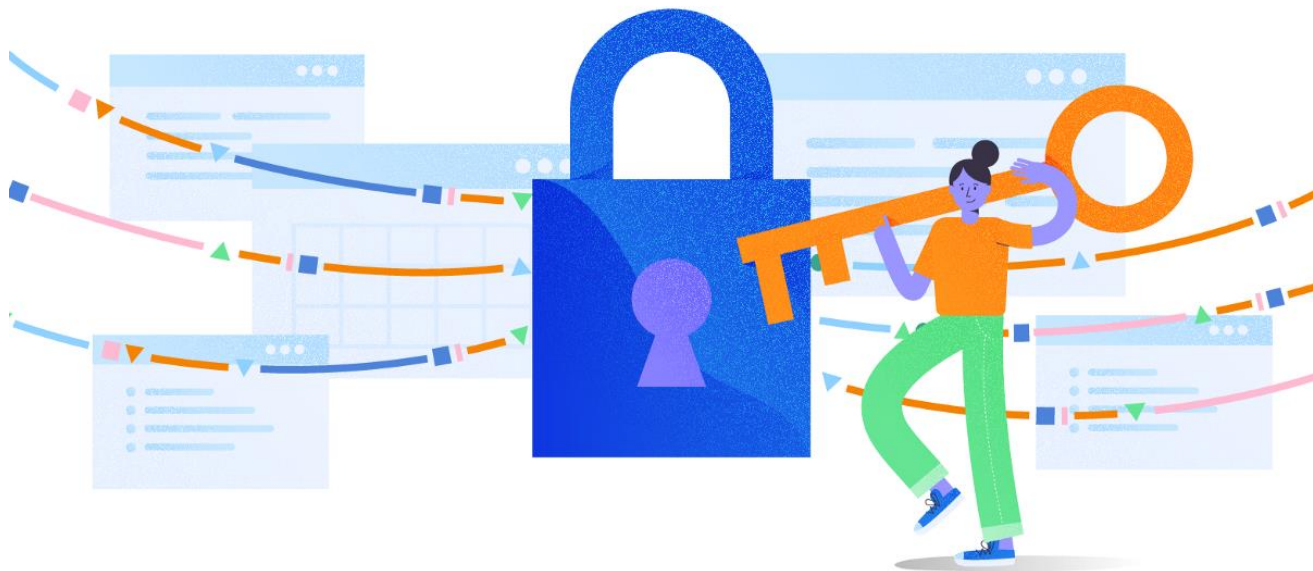
Foydalanuvchilar boshqa foydalanuvchilarni kutishlari kerak bo'lgan yagona vaqt - ular bir xil qatorni o'zgartirishga harakat qilishlaridir. Agar ular turli qatorlarni o'zgartirsa, kutish shart emas. SELECT so'rovlarini hech qachon kutish shart emas. Ma'lumotlar bazasi blokirovkalashni avtomatik ravishda amalga oshiradi. Biroq, ba'zi hollarda, qulflash qo'lda boshqarilishi kerak. Qo'lda qulflash LOCK buyrug'i yordamida amalga oshirilishi mumkin. Bu tranzaksiyaning bloklash turi va ko'lamini aniqlash imkonini beradi.

# Locks in PostgreSQL

LOCK buyrug`ining asosiy sintaksisi quyidagicha -

```
LOCK [ TABLE ] name IN lock_mode
```

- name** - Qulflash uchun mavjud jadvalning nomi (ixtiyoriy ravishda sxemaga muvofiq). Agar ONLY jadval nomidan oldin ko`rsatilgan bo`lsa, faqat o`sha jadval bloklanadi. Agar ONLY ko`rsatilmagan bo`lsa, jadval va uning barcha avlod jadvallari (agar mavjud bo`lsa) qulflanadi.
- lock\_mode** - Qulflash rejimi bu qulf qaysi qulf bilan zid **kelishini** belgilaydi. Agar qulflash rejimi belgilanmagan bo`lsa, u holda eng cheklovchi rejim bo`lgan ACCESS EXCLUSIVE ishlatiladi. Yana bo`lishi mumkin bo`lgan qiymatlar quyidagilardir: ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE.



## Locks in PostgreSQL

Table-Level  
Locking

**d\_test (Table)**

id	name
1	Heikki
2	Robert
3	Tom
4	Bruce
5	Dave
6	Hope

Row-Level Locking

### Lock Types

1. AccessShareLock
2. RowShareLock
3. RowExclusiveLock
4. ShareLock
5. ShareRowExclusiveLock
6. ExclusiveLock
7. AccessExclusiveLock

Share Lock or  
Exclusive Lock

**Note on Table-Locks:** Remember that all of these lock modes are table-level locks, even if the name contains the word "row"; the names of the lock modes are historical.

# Locks in PostgreSQL

```
LOCK TABLE company IN ACCESS EXCLUSIVE MODE;
```

Yuqoridagi buyruq **company** jadvalining tranzaksiya tugaguniga qadar qulflanganligini va tranzaksiyani tugatish uchun orqaga qaytarish yoki tranzaksiyani amalga oshirish kerakligini bildiradi.

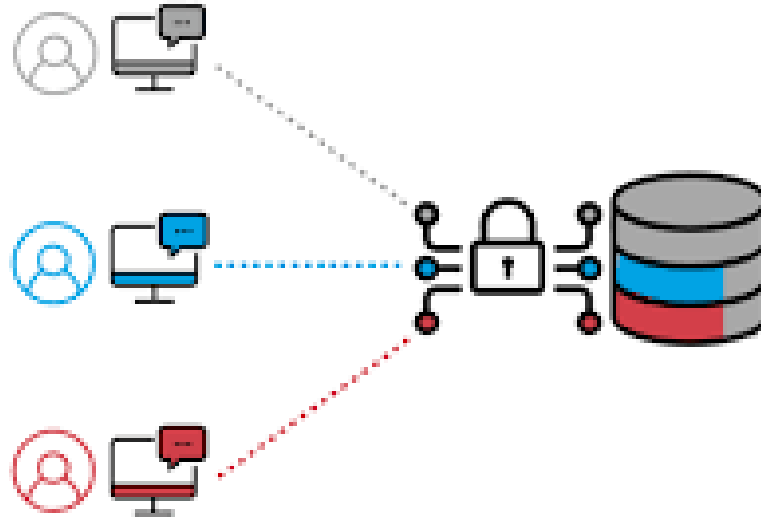
## Privileges in PostgreSQL

Har doim ma'lumotlar bazasida ob'ekt yaratilganda, unga **owner** ya'ni egasi tayinlanadi. Egasi odatda yaratish bayonotini bajargan shaxsdir. Ko'pgina turdagi ob'ektlar uchun boshlang'ich holatda faqat egasi (yoki superfoydalanuvchi) ob'ektni o'zgartirishi yoki o'chirishi mumkin. Boshqa rollar yoki foydalanuvchilarga undan foydalanishga ruxsat berish uchun **privileges** ya'ni *imtiyozlar* yoki *ruxsatnomalar* berilishi kerak.

PostgreSQLda har xil turdagi imtiyozlar –

SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY, EXECUTE va USAGE ka'bi imtiyozlar mavjud.

# Privileges in PostgreSQL





## Privileges in PostgreSQL



# Grant Privileges (*imtiyoz berish*)

PostgreSQL da yangi yaratilgan foydalanuvchi **role** o`ziga biriktirilgan parol bilan PostgreSQL ma`lumotlar bazasi serveriga kirishi mumkin. Biroq, u **jadvallar, ko`rinishlar, funktsiyalar** va boshqalar kabi ma`lumotlar bazasi ob`ektlariga hech narsa qila olmaydi .

Misol uchun, foydalanuvchi roli jadvaldan ma`lumotlarni tanlay olmaydi yoki muayyan funktsiyani bajara olmaydi.

Foydalanuvchi roliga ma`lumotlar bazasi ob`ektlariga o`zaro ta`sir o`tkazishga ruxsat berish uchun **GRANT** bayonoti yordamida foydalanuvchi roliga ma`lumotlar bazasi ob`ektlarida imtiyozlar berish kerak .

Quyida rolga jadvalda bir yoki bir nechta imtiyozlar beradigan **GRANT** bayonotining oddiy shakli ko'rsatilgan :

```
GRANT privilege_list | ALL
ON table_name
TO role_name;
```

Ushbu sintaksisda:

- Birinchidan, belgilash **privilege\_list** bo`lishi mumkin [SELECT](#), [INSERT](#), [UPDATE](#), [DELETE](#), [TRUNCATE](#) va boshqalar. Rolga jadvaldagi barcha imtiyozlarni berish uchun **ALL** variantidan foydalaniladi .
- Ikkinchidan, **ON** kalit so`zidan keyin jadval nomi belgilanadi.
- Uchinchidan, imtiyozlar bermoqchi bo`lgan rolning nomi belgilanadi.

## Revoke Privileges (*imtiyozni bekor qilish*)

PostgreSQLda foydalanuvchi roliga berilgan privilege (*imtiyozlar*) ni olib tashlash uchun REVOKE dan foydalaniladi. Quyida roldan bir yoki bir nechta jadvaldagi imtiyozlarni bekor qiluvchi **REVOKE** bayonoti sintaksisi ko'rsatilgan :

```
REVOKE privilege | ALL  
ON TABLE table_name | ALL TABLES IN SCHEMA schema_name  
FROM role_name;
```

Ushbu sintaksisda:

- Birinchidan, bekor qilmoqchi bo'lgan bir yoki bir nechta imtiyozlar belgilanadi. Barcha imtiyozlarni bekor qilish uchun **ALL** variantidan foydalaniladi.
- Ikkinchidan, **ON** kalit so'zdan keyin jadval nomini belgilanadi . Sxemadagi barcha jadvallardan belgilangan imtiyozlarni bekor qilish uchun **ALL TABLES** dan foydalaniladi.
- Uchinchidan, imtiyozlarni bekor qilmoqchi bo'lgan rolning nomi belgilanadi.

E`TIBORINGIZ UCHUN RAXMAT!