

# Design of the database

Andrii Yanechko

March 2022

# 1 Program Description

This is a database for ordering a ticket on the events.

## 2 Design database for CDP program

Main entities of the database are:

- User;
- Event;
- Ticket.

### 2.1 User entity

User entity, illustrated on a figure 2.1, represents user in the database and have several fields:

Name	Type	Description	Constraints
id	<i>integer</i>	unique identifier of the user	<b>Primary Key</b>
name	<i>text</i>	name of the user	<b>Unique</b>
email	<i>text</i>	email of the user	<b>Unique</b>
created_date	<i>text</i>	date of instance creation in the UTC	N/A
updated_date	<i>text</i>	date of the last update in the UTC	

Indexes for user entity:

Name	Type
name	<i>B-tree</i>
email	<i>B-tree</i>

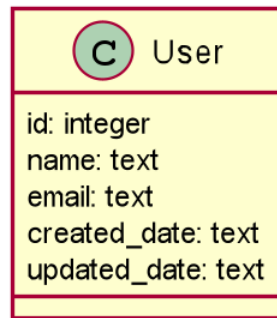


Figure 1: User representation in the database

## 2.2 Event entity

Event entity, illustrated on a figure 2 , represents event in the database and have several fields:

Name	Type	Description	Constraints
id	<i>integer</i>	unique identifier of the event	<b>Primary Key</b>
title	<i>text</i>	title of the event	<b>Unique</b>
date	<i>text</i>	start date of the event in the UTC	
created_date	<i>text</i>	date of instance creation in the UTC	<b>N/A</b>
updated_date	<i>text</i>	date of the last update in the UTC	

Indexes for event entity:

Name	Type
title	<i>B-tree</i>
date	<i>B-tree</i>

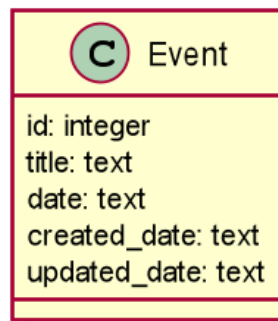


Figure 2: Event representation in the database

## 2.3 Ticket entity

Ticket entity, illustrated on a figure 3 , represents ticket in the database and have several fields:

Name	Type	Description	Constraints	
id	integer	unique identifier of the ticket	Primary Key	
user_id	integer	id of the user which has ordered this ticket	Secondary Key	
event_id	integer	id of the event on which ticket is booked	Secondary Key	Unique
place	integer	number of place of the ticket	N/A	
category	string	category of the ticket	N/A	
created_date	string	date of instance creation in the UTC		
updated_date	string	date of the last update in the UTC		

Indexes for ticket entity:

Name	Type
event_id	<i>B-tree</i>
user_id	<i>B-tree</i>
category	<i>B-tree</i>

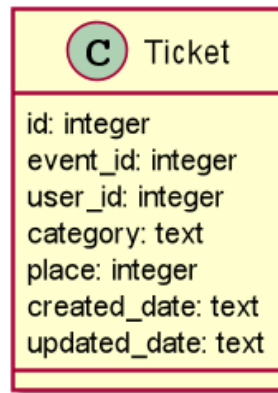


Figure 3: Ticket representation in the database

## 3 Implementation of the database design in the PostgreSQL

### 3.1 User table

```
CREATE TABLE public."user"
(
    id integer NOT NULL,
    name character varying(50) NOT NULL,
    email character varying(50) NOT NULL,
    created_date character varying(50),
    updated_date character varying(50),
    PRIMARY KEY (id),
    CONSTRAINT name_unique UNIQUE (name),
    CONSTRAINT email_unique UNIQUE (email)
);
```

```
ALTER TABLE IF EXISTS public."user"
OWNER to postgres;
```

### 3.2 Event table

```
CREATE TABLE public.event
(
  id integer NOT NULL,
  title character varying(50) NOT NULL,
  date character varying(50) NOT NULL,
  created_date character varying(50),
  updated_date character varying(50),
  PRIMARY KEY (id),
  CONSTRAINT title_date UNIQUE (title, date)
);

ALTER TABLE IF EXISTS public.event
OWNER to postgres;
```

### 3.3 Ticket table

```
CREATE TABLE public.ticket
(
  id integer NOT NULL,
  user_id integer NOT NULL,
  event_id integer NOT NULL,
  place integer NOT NULL,
  category character varying(30) NOT NULL,
  created_date character varying(50),
  updated_date character varying,
  PRIMARY KEY (id),
  CONSTRAINT unique_event_id_place UNIQUE (event_id, place),
  CONSTRAINT foreign_key_user_id FOREIGN KEY (user_id)
REFERENCES public."user" (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID,
  CONSTRAINT foreign_key_event_id FOREIGN KEY (event_id)
REFERENCES public.event (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
NOT VALID
);

ALTER TABLE IF EXISTS public.ticket
OWNER to postgres;
```

### 3.4 Database entity relations

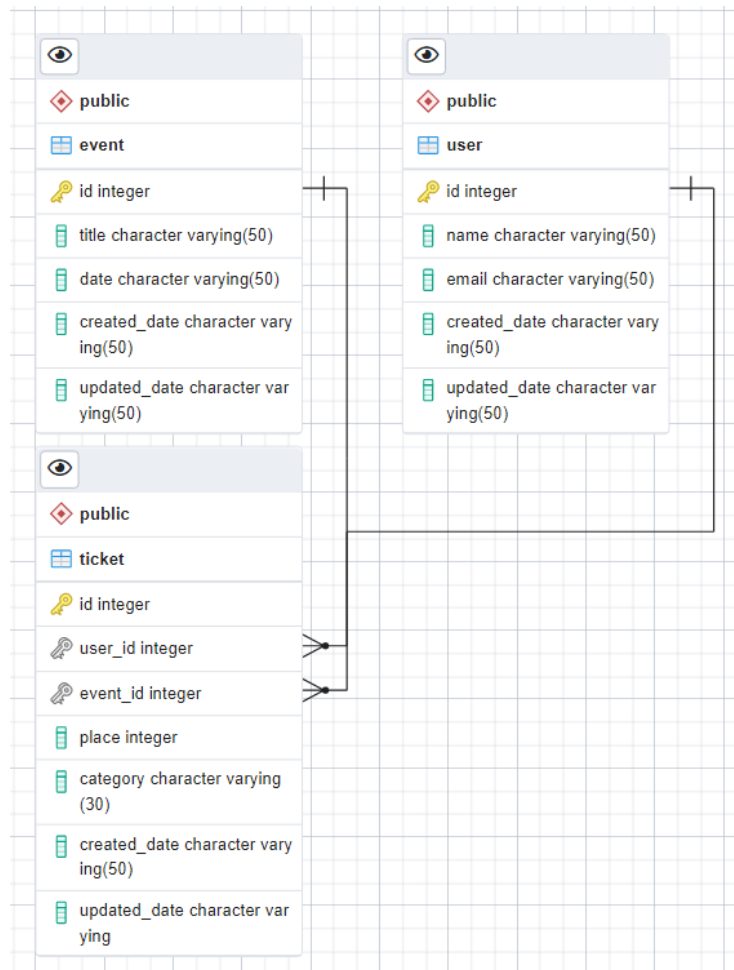


Figure 4: Entities relation in the database

### 3.5 Trigger for update time on UPDATE

Trigger function looks like:

```
CREATE OR REPLACE FUNCTION public.set_update_time()  
RETURNS trigger  
LANGUAGE 'plpgsql'  
VOLATILE  
COST 100  
AS $BODY$
```

```

BEGIN
new.updated_time = CURRENT_TIME(2);
RETURN new;
END;
$BODY$;

```

Trigger looks like:

```

CREATE TRIGGER set_update_time_on_update
BEFORE UPDATE OF id, name, email, created_date, updated_date
ON public."user"
FOR EACH ROW
EXECUTE FUNCTION public.set_update_time();

```

## 4 PostgreSQL index comparison

### 4.1 Index comparison

So, I've investigated 2 types of indexes **B-tree** and **hash**. I don't investigate 2 other indexes **GIN** and **GIST**, because they are not supporting a character varying, that is actually 95% of my indexes, so data for the investigation won't be accurate in that case.

#### 4.1.1 Size of indexes

Query that I used to measure indexes size:

```

select pg_size_pretty(pg_relation_size('index_ticket_event_id'))
as index_ticket_event_id,
pg_size_pretty(pg_relation_size('index_ticket_user_id')) as index_ticket_user_id,
pg_size_pretty(pg_relation_size('index_ticket_category')) as index_ticket_category,
pg_size_pretty(pg_relation_size('index_event_date')) as index_event_date,
pg_size_pretty(pg_relation_size('index_event_title')) as index_event_title,
pg_size_pretty(pg_relation_size('index_user_email')) as index_user_email,
pg_size_pretty(pg_relation_size('index_user_name')) as index_user_name

```

#### 4.1.2 Speed of queries

Query for selecting user by name (exact match):

```

SELECT * FROM public."user" WHERE "name" LIKE 'name 1050'

```

```

1 select pg_size_pretty(pg_relation_size('index_ticket_event_id')) as index_ticket_event_id,
2 pg_size_pretty(pg_relation_size('index_ticket_user_id')) as index_ticket_user_id,
3 pg_size_pretty(pg_relation_size('index_ticket_category')) as index_ticket_category,
4 pg_size_pretty(pg_relation_size('index_event_date')) as index_event_date,
5 pg_size_pretty(pg_relation_size('index_event_title')) as index_event_title,
6 pg_size_pretty(pg_relation_size('index_user_email')) as index_user_email,
7 pg_size_pretty(pg_relation_size('index_user_name')) as index_user_name

```

Data Output	Explain	Messages	Notifications			
<div> <div>index_ticket_event_id</div> <div>text</div> </div>	<div> <div>index_ticket_user_id</div> <div>text</div> </div>	<div> <div>index_ticket_category</div> <div>text</div> </div>	<div> <div>index_event_date</div> <div>text</div> </div>	<div> <div>index_event_title</div> <div>text</div> </div>	<div> <div>index_user_email</div> <div>text</div> </div>	<div> <div>index_user_name</div> <div>text</div> </div>
1 44 MB	32 MB	32 MB	56 kB	48 kB	4112 kB	4112 kB

Figure 5: Represents how much size each index takes using hash indexes

Data Output		Explain	Messages	Notifications			
	index_ticket_event_id text	index_ticket_user_id text	index_ticket_category text	index_event_date text	index_event_title text	index_user_email text	index_user_name text
1	7168 kB	9256 kB	30 MB	16 kB	48 kB	3992 kB	3104 kB

Figure 6: Represents how much size each index takes using B-tree indexes

B-tree	4	✓ Successfully run. Total query runtime: 111 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 106 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 104 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 110 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 111 msec. 1000 rows affected.				
hash	4	✓ Successfully run. Total query runtime: 57 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 58 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 64 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 64 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 55 msec. 1000 rows affected.				
no index	4	✓ Successfully run. Total query runtime: 107 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 104 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 112 msec. 1000 rows affected.				
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.				



Query for selecting user by name (contains):

SELECT \* FROM public."user" WHERE name LIKE "%name 5%"

B-tree	✓ Successfully run. Total query runtime: 191 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 116 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 116 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 137 msec. 11111 rows affected.
hash	✓ Successfully run. Total query runtime: 67 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 63 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 61 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 55 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 119 msec. 11111 rows affected.
no index	✓ Successfully run. Total query runtime: 123 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 111 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 105 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 104 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 102 msec. 11111 rows affected.
	✓ Successfully run. Total query runtime: 97 msec. 11111 rows affected.

Query for selecting user by email (contains):

SELECT \* FROM public."user" WHERE email LIKE "%170%"

B-tree	4	✓ Successfully run. Total query runtime: 111 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 106 msec. 1000 rows affected.
	4901193+03:00[Europe/Kiev]	null
	4	✓ Successfully run. Total query runtime: 104 msec. 1000 rows affected.
	495209+03:00[Europe/Kiev]	null
hash	4	✓ Successfully run. Total query runtime: 110 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 57 msec. 1000 rows affected.
	51185+03:00[Europe/Kiev]	null
	4	✓ Successfully run. Total query runtime: 58 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 64 msec. 1000 rows affected.
no index	4	✓ Successfully run. Total query runtime: 64 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 55 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 107 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 104 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 112 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 107 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.
	4	✓ Successfully run. Total query runtime: 108 msec. 1000 rows affected.

Query by email from users that have at least one ticket (contains):

```

SELECT public."user".* FROM public."user"
WHERE (SELECT COUNT(*)FROM public.ticket WHERE public.ticket.id = public."user".id)
> 0
AND public."user".email LIKE "%77%"

```

B-tree	<div>✓ Successfully run. Total query runtime: 116 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 57 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 56 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 58 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 68 msec. 1000 rows affected.</div>
hash	<div>✓ Successfully run. Total query runtime: 54 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 50 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 48 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 56 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 51 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 52 msec. 1000 rows affected.</div>
no index	<div>✓ Successfully run. Total query runtime: 125 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 167 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 129 msec. 1000 rows affected.</div> <div>✓ Successfully run. Total query runtime: 106 msec. 1000 rows affected.</div>

## 4.2 My conclusion

So as I can see **B-tree** index is not very useful in my case, I guess it's because my indexes are almost refers character varying types and that's why **B-tree** is not very efficient.

Taking look on the size which indexes are taking, **hash** is on the figure and on

the **b-tree** figure, **hash** index takes more size than **b-tree**, but taking a look at the efficient comparing tables **hash** is more-more efficient in case of character varying indexes.