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`	*integer*	unique identifier of the user	**Primary Key**
`name`	*text*	name of the user	**Unique**
`email`	*text*	email of the user	**Unique**
`created_date`	*text*	date of instance creation in the UTC	**N\A**
`updated_date`	*text*	date of the last update in the UTC	

Indexes for user entity:

Name	Type
`name`	*B-tree*
`email`	*B-tree*

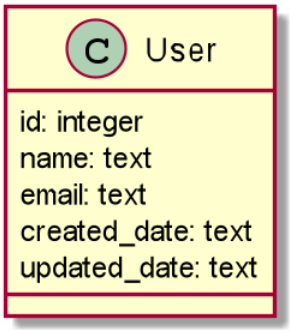


Figure 1: User representation in the database

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`	*integer*	unique identifier of the event	**Primary Key**
`title`	*text*	title of the event	**Unique**
`date`	*text*	start date of the event in the UTC	
`created_date`	*text*	date of instance creation in the UTC	**N\A**
`updated_date`	*text*	date of the last update in the UTC	

Indexes for event entity:

Name	Type
`title`	*B-tree*
`date`	*B-tree*

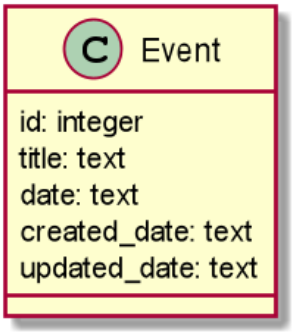


Figure 2: Event representation in the database

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`	*B-tree*
`user_id`	*B-tree*
`category`	*B-tree*

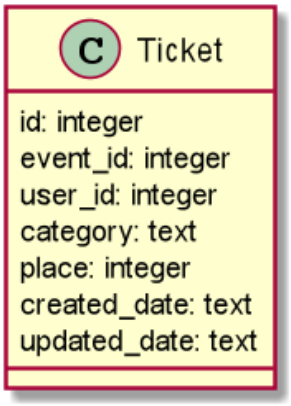


Figure 3: Ticket representation in the database

3 Implementation of the database design in the PostgresSQL

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

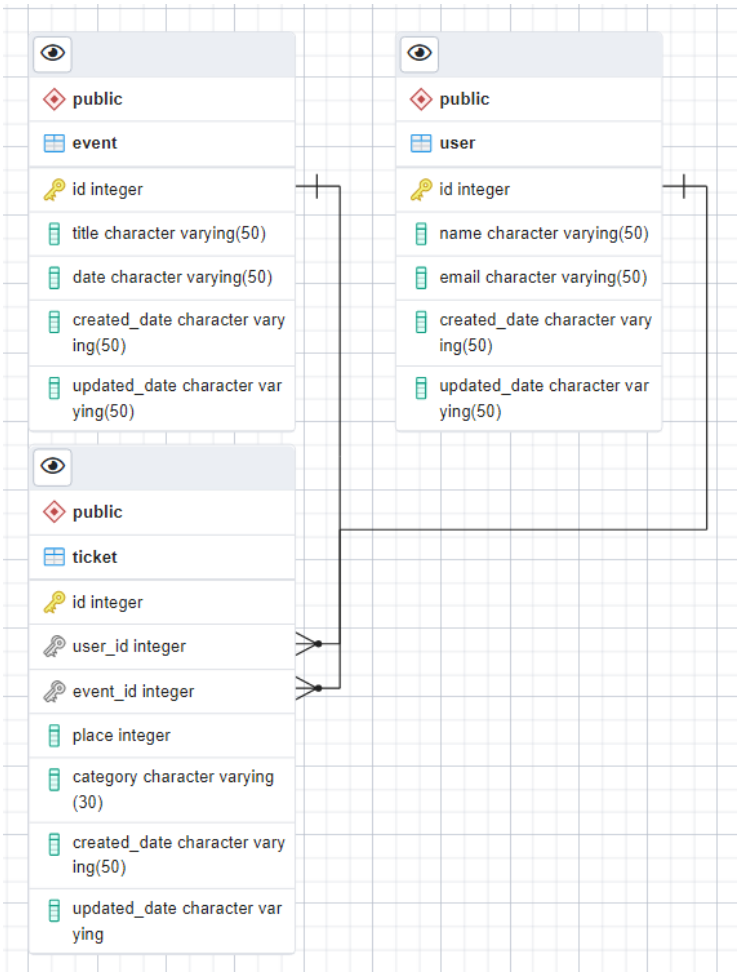


Figure 4: Entities relation in the database

4 PostgresSQL index comparation

4.1 Index comparation

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 actully 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
```

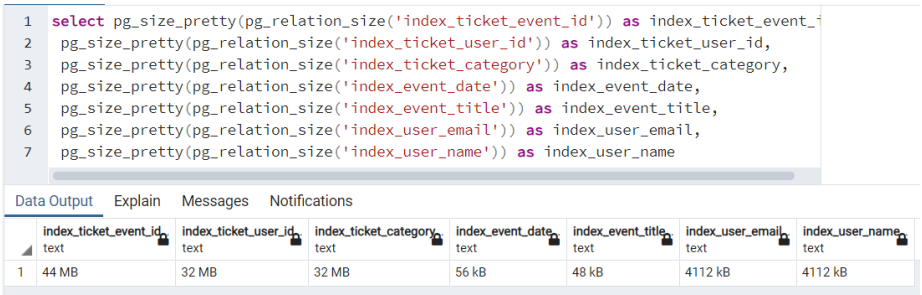


Figure 5: Represents how much size each index takes using hash indexes

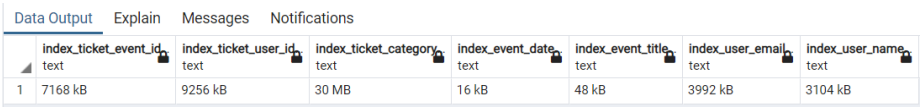
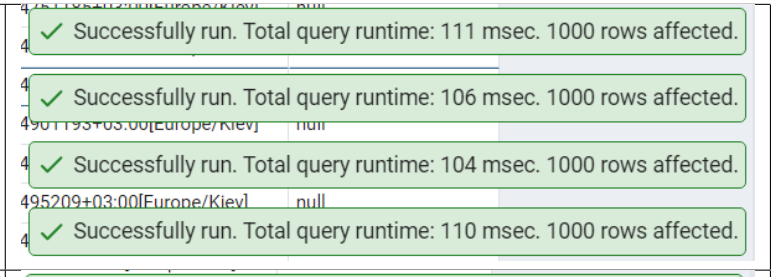
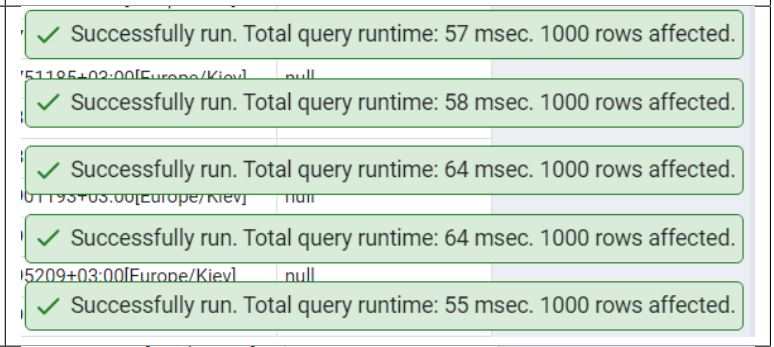
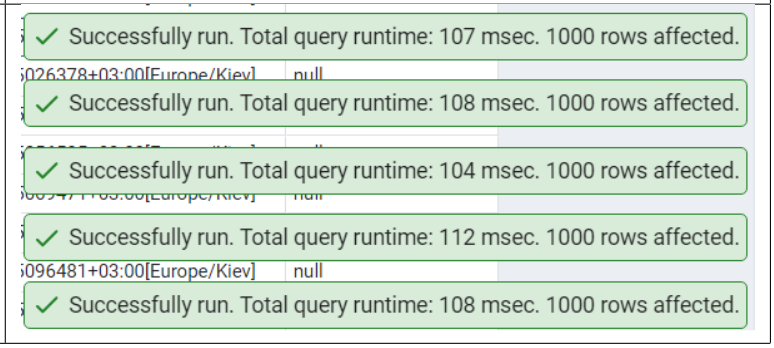


Figure 6: Represents how much size each index takes using B-tree indexes

4.1.2 Speed of queries

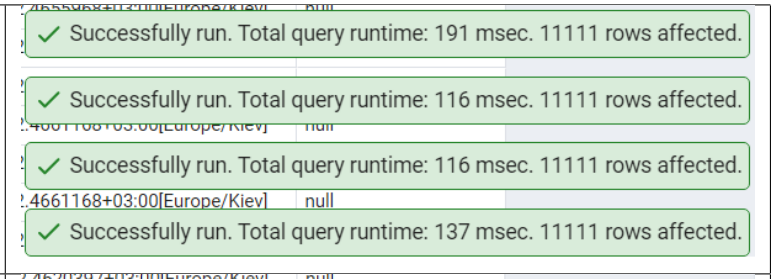
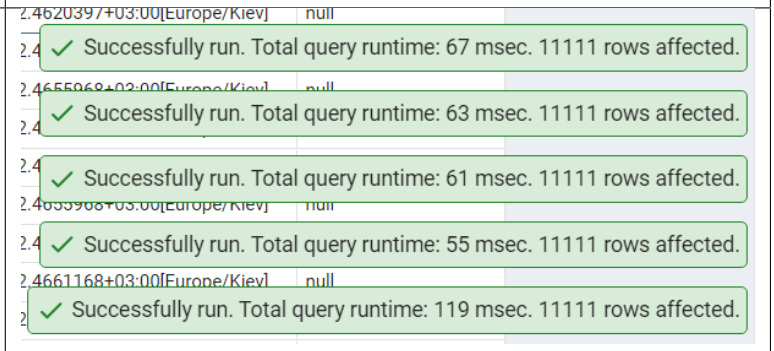
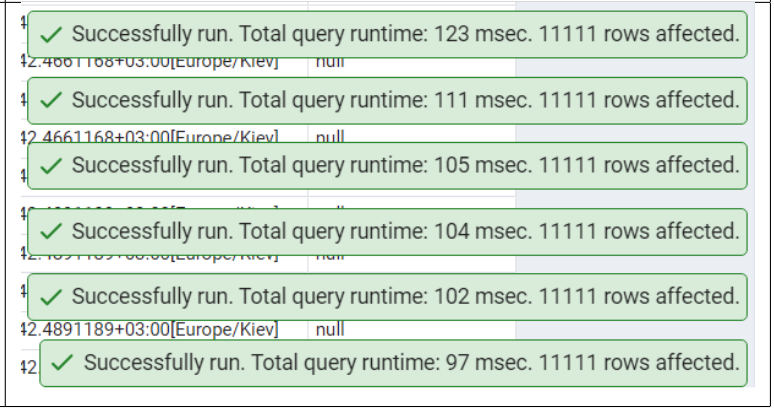
Query for selecting user by name (exact match):

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

B-tree	
hash	
no index	

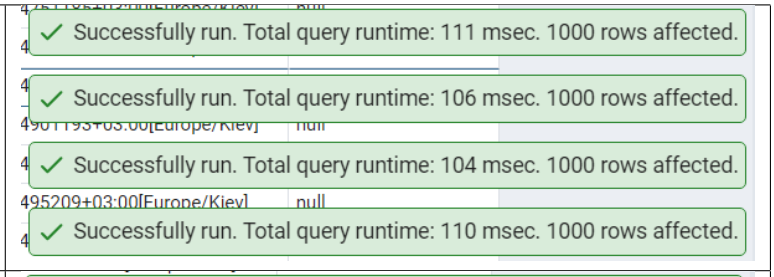
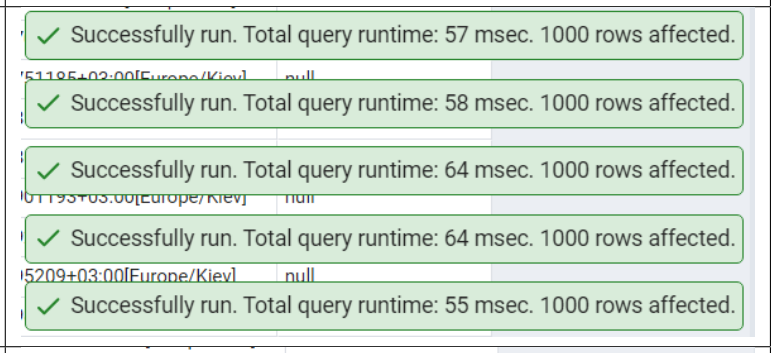
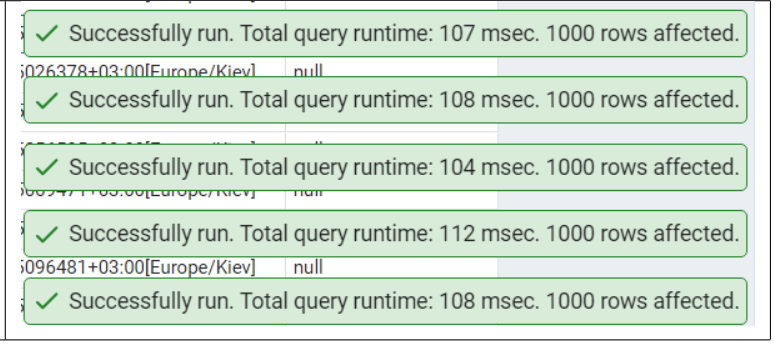
Query for selecting user by name (contains):

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

B-tree	
hash	
no index	

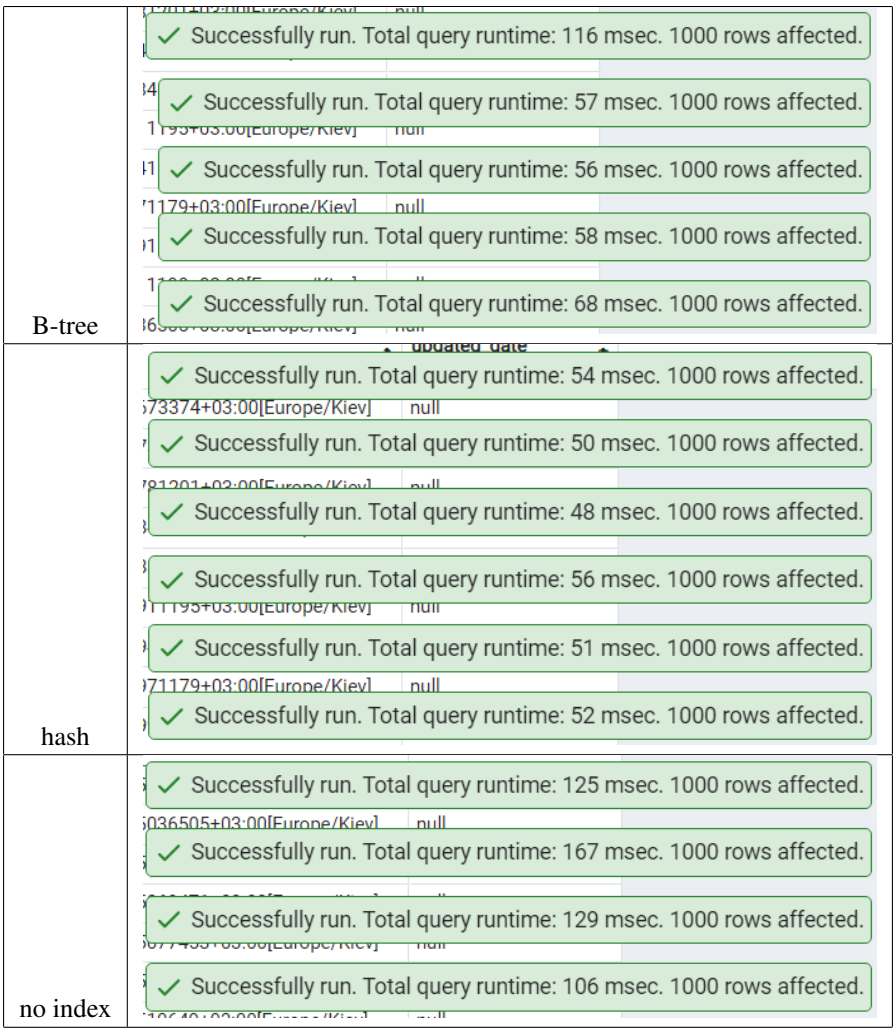
Query for selecting user by email (contains):

```
SELECT * FROM public."user" WHERE email LIKE "%17@%"
```

B-tree	
hash	
no index	

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@%"
```



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.