yazı tipi, logo, metin, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

PostgreSQL ile Çevrimiçi Eğitim Platformu Veritabanı Tasarımı Projesi

create table members (

id bigint primary key generated always as identity,

username varchar(50) unique not null,

email varchar(100) unique not null check(email ~\* '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),

passwords varchar(255) not null,

register\_date timestamp default current\_timestamp,

first\_name varchar(50),

last\_name varchar(50),

is\_active boolean default TRUE

);

create table categories (

id integer primary key generated always as identity,

category\_name varchar(150) unique not null,

description text

);

create table courses (

id bigint primary key generated always as identity,

title varchar(200) not null,

description text,

start\_date timestamp with time zone not null,

-- Farklı zaman dilimlerinde eğitimler varsa,

-- TIMESTAMP WITH TIME ZONE kullanarak zaman dilimi bilgisini de saklayabilirsin.

end\_date timestamp with time zone not null,

check(start\_date < end\_date),

instruction\_name varchar(100),

category\_id integer,

constraint fk\_category foreign key (category\_id) references categories(id)

);

create table enrollments (

id bigint primary key generated always as identity,

member\_id bigint not null,

course\_id bigint not null,

enrollment\_date timestamp default current\_timestamp,

constraint fk\_member foreign key (member\_id) references members(id),

constraint fk\_course foreign key (course\_id) references courses(id),

constraint uq\_member\_course unique (member\_id, course\_id) -- Aynı kullanıcı aynı eğitime bir kez katılabilir

);

create table certificates (

id bigint primary key generated always as identity,

certificate\_code varchar(100) unique not null,

issue\_date date not null default current\_date,

description text

);

create table CertAssignments (

id bigint primary key generated always as identity,

enrollment\_id bigint not null,

certificated\_id bigint not null,

assigned\_date date not null default current\_date,

constraint fk\_cert\_assignment\_enrollment foreign key (enrollment\_id) references enrollments(id),

constraint fk\_cert\_assignment\_certificate foreign key (certificated\_id) references certificates(id),

-- Aynı katılım (enrollment) için aynı sertifika bir daha atanamaz

constraint uq\_enrollment\_certificate unique (enrollment\_id, certificated\_id)

);

create table blogposts (

id bigint primary key generated always as identity,

title varchar(255) not null,

contents text not null,

published\_at timestamp default current\_timestamp,

author\_id bigint not null,

constraint fk\_blogpost\_author foreign key (author\_id) references members(id)

);

create table comment\_sys(

id bigint primary key generated always as identity,

content text not null,

created\_at timestamp default current\_timestamp,

post\_id bigint not null,

user\_id bigint not null,

constraint fk\_comment\_post foreign key (post\_id) references blogposts(id),

constraint fk\_comment\_user foreign key (user\_id) references members(id)

);

create table course\_reviews (

id bigint primary key generated always as identity,

course\_id bigint not null,

enrollment\_id bigint not null,

rating int not null check(rating between 1 and 5),

comment text,

created\_at timestamp default current\_timestamp,

constraint fk\_course\_review foreign key (course\_id) references courses(id),

constraint fk\_course\_enroll foreign key (enrollment\_id) references enrollments(id),

constraint uq\_enrollmet\_review unique(enrollment\_id)

);

create table notifications (

id bigint primary key generated always as identity,

user\_id bigint not null,

message text not null,

is\_read boolean default FALSE,

created\_at timestamp default current\_timestamp,

constraint fk\_notification\_user foreign key (user\_id) references members(id)

);

create table course\_statistics (

id bigint primary key generated always as identity,

course\_id bigint not null,

total\_enrolled int not null, -- Eğitime kayıtlı toplam kullanıcı

--total\_completed int not null, -- Eğitimi tamamlayan kullanıcı sayısı

success\_rate decimal(5,2), -- Başarı oranı (yüzde)

last\_updated timestamp default current\_timestamp,

constraint fk\_course\_stat foreign key (course\_id) references courses(id)

);

create table roles(

id bigint primary key generated always as identity,

role\_name varchar(15) unique not null

);

create table user\_roles (

id bigint primary key generated always as identity,

user\_id bigint not null,

role\_id bigint not null,

assigned\_at timestamp default current\_timestamp, -- Rol atama tarihi

constraint fk\_roles\_user foreign key (user\_id) references members(id),

constraint fk\_role foreign key (role\_id) references roles(id)

);

--Eklenen veriler

-- Örnek üyeler

INSERT INTO members (username, email, passwords, first\_name, last\_name, is\_active)

VALUES

('john\_doe', 'john.doe@example.com', 'hashed\_password\_1', 'John', 'Doe', TRUE),

('jane\_smith', 'jane.smith@example.com', 'hashed\_password\_2', 'Jane', 'Smith', TRUE),

('alice\_jones', 'alice.jones@example.com', 'hashed\_password\_3', 'Alice', 'Jones', FALSE);

select \* from members;

insert into members (username, email, passwords, first\_name, last\_name, is\_active)

values

('michael\_brown', 'michael.brown@example.com', 'hashed\_password\_4', 'Michael', 'Brown', TRUE),

('emily\_white', 'emily.white@example.com', 'hashed\_password\_5', 'Emily', 'White', TRUE),

('david\_clark', 'david.clark@example.com', 'hashed\_password\_6', 'David', 'Clark', FALSE),

('sophia\_miller', 'sophia.miller@example.com', 'hashed\_password\_7', 'Sophia', 'Miller', TRUE),

('liam\_wilson', 'liam.wilson@example.com', 'hashed\_password\_8', 'Liam', 'Wilson', FALSE);

-- Örnek kategoriler

INSERT INTO categories (category\_name, description)

VALUES

('Mathematics', 'Mathematics courses including algebra, calculus, and more.'),

('Computer Science', 'Courses related to programming, software development, and algorithms.'),

('Psychology', 'Courses focused on the study of the human mind and behavior.');

select \* from categories;

-- Örnek kurslar

INSERT INTO courses (title, description, start\_date, end\_date, instruction\_name, category\_id)

VALUES

('Introduction to Algebra', 'A beginner course in algebra.', '2025-05-01 10:00:00+00', '2025-05-30 16:00:00+00', 'Dr. John Doe', 1),

('Advanced Programming', 'Learn advanced programming concepts and techniques.', '2025-06-01 09:00:00+00', '2025-06-30 18:00:00+00', 'Dr. Jane Smith', 2),

('Introduction to Psychology', 'An overview of psychology principles and theories.', '2025-07-01 10:00:00+00', '2025-07-31 14:00:00+00', 'Dr. Alice Johnson', 3);

select \* from courses;

-- Örnek kayıtlar

INSERT INTO enrollments (member\_id, course\_id, enrollment\_date)

VALUES

(1, 1, '2025-04-25 10:00:00'),

(2, 2, '2025-04-26 11:00:00'),

(1, 3, '2025-04-25 12:00:00');

select \* from enrollments;

-- Örnek sertifikalar

INSERT INTO certificates (certificate\_code, issue\_date, description)

VALUES

('CERT001', '2025-05-15', 'Certificate for completing Introduction to Algebra.'),

('CERT002', '2025-06-15', 'Certificate for completing Advanced Programming.');

select \* from certificates;

-- Örnek sertifika atamaları

INSERT INTO CertAssignments (enrollment\_id, certificated\_id, assigned\_date)

VALUES

(1, 1, '2025-05-16'),

(2, 2, '2025-06-16');

select \* from CertAssignments;

-- Örnek blog yazıları

INSERT INTO blogposts (title, contents, author\_id)

VALUES

('Understanding Algebra', 'This post discusses key concepts of Algebra for beginners.', 1),

('Top Programming Tips', 'This blog highlights some important programming tips and tricks.', 2);

select \* from blogposts;

-- Örnek yorumlar

INSERT INTO comment\_sys (content, post\_id, user\_id)

VALUES

('Great post! Very helpful.', 1, 2),

('I learned a lot from this article.', 2, 3);

select \* from comment\_sys;

-- Örnek kurs değerlendirmeleri

INSERT INTO course\_reviews (course\_id, enrollment\_id, rating, comment)

VALUES

(1, 1, 5, 'Excellent course! Learned a lot about algebra.'),

(2, 2, 4, 'Good course but could use more examples.');

select \* from course\_reviews;

-- Örnek bildirimler

INSERT INTO notifications (user\_id, message)

VALUES

(1, 'You successfully enrolled in Introduction to Algebra.'),

(2, 'You successfully enrolled in Advanced Programming.');

select \* from notifications;

-- Örnek kurs istatistikleri

INSERT INTO course\_statistics (course\_id, total\_enrolled, success\_rate)

VALUES

(1, 1, 100.00),

(2, 1, 80.00);

select \* from course\_statistics;

-- Örnek roller

INSERT INTO roles (role\_name)

VALUES

('Admin'),

('Instructor'),

('Student');

select \* from roles;

-- Örnek kullanıcı rolleri

INSERT INTO user\_roles (user\_id, role\_id)

VALUES

(1, 1), -- John Doe için Admin rolü

(2, 2); -- Jane Smith için Instructor rolü

select \* from user\_roles;

-- Örnek loglar

INSERT INTO logs (user\_id, log\_message)

VALUES

(1, 'Admin role assigned.'),

(2, 'Instructor role assigned.');

select \* from logs;

--Trigger functions

--course\_statistics güncellemesi

/\*

create or replace function update\_total\_enrolled()

returns trigger as $$

begin

update course\_statistics

set total\_enrolled = coalesce(total\_enrolled, 0) + 1,--Böylece NULL + 1 gibi bir saçmalık çıkmıyor.

last\_update= now()

where course\_id = NEW.course\_id;

/\* set total\_enrolled = total\_enrolled +1,

last\_update = now()

where course\_id = NEW.course\_id;

\*/

-- Eğer hiçbir satır güncellenmediyse (böyle bir course\_id yoksa), hata verelim

if not found then --Bazı SQL komutları (örneğin SELECT, UPDATE, DELETE) çalıştıktan sonra son işlem bir satırı etkiledi mi? sorusuna cevap verir.

raise exception 'Bu % course\_id bulunamadı.', NEW.course\_id;

end if;

return new;

end;

$$ language plpgsql;

create trigger enroll\_update\_statistics

after insert on enrollments

for each row

execute function update\_total\_enrolled();

\*/

create or replace function update\_total\_enrolled()

returns trigger as $$

begin

if TG\_OP = 'INSERT' then

--Kayıt eklendiğinde

update course\_statistics

set total\_enrolled = coalesce(total\_enrolled, 0) + 1,

last\_updated = now()

where course\_id = new.course\_id;

-- Eğer hiç satır güncellenmediyse, bu course\_id mevcut değil demektir

/\* if not found then

raise exception 'Bu course\_id (%) bulunamadı.', NEW.course\_id;

end if;

\*/

return new;

elsif TG\_OP = 'DELETE' then

--Kayıt silindiğinde

update course\_statistics

set total\_enrolled = GREATEST(coalesce(total\_enrolled, 0) - 1, 0),

last\_updated = now()

where course\_id = old.course\_id;

if not found then

raise exception 'Bu course\_id (%), kurs bulunamadı.', OLD.course\_id;

end if;

return old;

end if;

return null; -- başka bir işlem olursa NULL döneriz

end;

$$ language plpgsql;

create trigger enroll\_update\_statistics\_insert

after insert on enrollments

for each row

execute function update\_total\_enrolled();

create trigger enroll\_update\_statistics\_delete

after delete on enrollments

for each row

execute function update\_total\_enrolled();

--course\_reviews sonrası başarı oranı (success\_rate) güncellemesi

create or replace function update\_success\_rate()

returns trigger as $$

declare

/\*değişken oluşturmak

DECLARE

değişken\_ismi veri\_tipi [DEFAULT başlangıç\_değeri];

\*/

completed\_count int default 0; --belli puan üstü veren kullanıcıları

total\_count int default 0; --tüm kullanıcıları

begin

select count(\*) into completed\_count from course\_reviews

where course\_id = new.course\_id and rating >= 3;

--3 ve üstü basarılı olarak kabul edilmiştir

select count(\*) into total\_count from enrollments

where course\_id = new.course\_id;

update course\_statistics

set success\_rate = case

--"Eğer şöyleyse, şunu yap; değilse, bunu yap" mantığı kuruyor.

when total\_count > 0 then

round((completed\_count::decimal / total\_count)\*100, 2)

else 0

end,

last\_updated = now()

where course\_id = new.course\_id;

return new;

end;

$$ language plpgsql;

create trigger review\_update\_statistics

after insert or update on course\_reviews

for each row

execute function update\_success\_rate();

--notifications sisteminde otomatik bildirim

/\*

($$) PostgreSQL'de bir şeyin

başlangıcını ve bitişini

belirtmek için kullanılıyor.

\*/

create or replace function enrollment\_notification()

returns trigger as $$

begin

if TG\_OP = 'INSERT' then

insert into notifications(user\_id, message, created\_at)

values (new.member\_id, 'Basariyla kursa kayit oldunuz!',now());

elsif TG\_OP = 'DELETE' THEN

delete from notifications

where user\_id = old.user\_id and message = 'Basariyla kursa kayit oldunuz!';

end if;

return null;

/\*

Burada trigger AFTER INSERT veya DELETE için çalışıyor.

Yani aslında zaten kayıt eklenmiş veya silinmiş.

Bu yüzden NEW veya OLD döndürmene gerek yok.

\*/

end;

$$ language plpgsql;

create trigger notify\_on\_enroll

after insert or delete on enrollments

for each row

execute function enrollment\_notification();

--certificates → Sertifika atanınca bildirim gönder

create or replace function certificate\_assigned\_notification()

returns trigger as $$

begin

insert into notifications (user\_id, message, created\_at)

values((select member\_id from enrollments where id = NEW.enrollment\_id), 'Tebrikler, sertifikanız başarıyla atandı!', now());

return new;

end;

$$ language plpgsql;

create trigger certificate\_assigned\_notify

after insert on CertAssignments

for each row

execute function certificate\_assigned\_notification();

--user\_roles → Admin atanınca özel işlem

create table logs (

id bigint primary key generated always as identity,

user\_id bigint not null,

log\_message text,

created\_at timestamp default current\_timestamp

);

create or replace function admin\_role\_assigned\_notification()

returns trigger as $$

declare

message\_text text;

begin

-- Kullanıcıya Admin rolü atandığında bildirim ekle

if TG\_OP = 'INSERT' THEN

if new.role\_id = (select id from roles where role\_name = 'Admin') then

message\_text := 'Tebrikler, Admin rolü başarıyla atandı!';

insert into notifications (user\_id, message, created\_at)

values(new.user\_id, message\_text, now());

-- Admin rolü atanan kullanıcı için bir log kaydı oluştur

insert into logs(user\_id, log\_message, created\_at)

values(new.user\_id, 'Admin rolü atandı.', now());

end if;

elsif TG\_OP = 'UPDATE' THEN

-- Admin rolü kaldırıldıysa

if old.role\_id is distinct from new.role\_id then

-- Eğer eski rol Admin değilken yeni rol Admin olmuşsa

if new.role\_id = (select id from roles where role\_name = 'Admin') then

message\_text := 'Tebrikler, Admin rolü başarıyla atandı!';

insert into notifications (user\_id, message, created\_at)

VALUES (OLD.user\_id, message\_text, now());

-- Admin rolü kaldırılan kullanıcı için log kaydı oluştur

insert into logs(user\_id, log\_message, created\_at)

values (new.user\_id, 'Admin rolü atandı.', now());

-- Eğer eski rol Admin iken yeni rol başka bir şey olmuşsa

if old.role\_id = (select id from roles where role\_name = 'Admin') then

message\_text := 'Admin rolü kaldırıldı.';

insert into notifications(user\_id, message, created\_at)

values(old.user\_id, message\_text, now());

insert into logs(user\_id, log\_message, created\_at)

values (old.user\_id, 'Admin rolü kaldırıldı.', now());

end if;

end if;

end if;

end if;

return new;

end;

$$ language plpgsql;

create trigger admin\_role\_notify

after insert or update on user\_roles

for each row

execute function admin\_role\_assigned\_notification();

/\*

ALTER TABLE tablo\_adi

ADD CONSTRAINT fk\_adi

FOREIGN KEY (alan\_adi) REFERENCES diger\_tablo(diger\_alan);

\*/

alter table logs

add constraint fk\_log\_user foreign key (user\_id) references members(id);

/\*

DELETE FROM categories;

TRUNCATE categories RESTART IDENTITY CASCADE;

TRUNCATE categories RESTART IDENTITY;

TRUNCATE CertAssignments RESTART IDENTITY;

TRUNCATE enrollments RESTART IDENTITY CASCADE;

\*/