DOCUMENTATION - DB/Backend
“PROJECTIFY”

Backend: http://localhost:5000/docs
Database:
Entities:
1.Users
```
CREATE TABLE public.users (
	user_id int4 DEFAULT nextval('user_id_seq'::regclass) NOT NULL,
	"name" varchar(100) NOT NULL,
	email varchar(100) NOT NULL,
	role_id int4 NULL,
	"password" varchar(100) NOT NULL,
	age int4 NULL,
	gender varchar(10) NULL,
	blood_group varchar(5) NULL,
	joined_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	modified_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	department_id int4 NULL,
	CONSTRAINT users_email_key UNIQUE (email),
	CONSTRAINT users_pkey PRIMARY KEY (user_id),
	CONSTRAINT users_department_id_fkey FOREIGN KEY (department_id)
REFERENCES public.departments(department_id) ON DELETE SET NULL,
	CONSTRAINT users_role_id_fkey FOREIGN KEY (role_id) REFERENCES
public.roles(role_id)
);
```

2.User Story History
```
CREATE TABLE public.user_story_history (
	history_id serial4 NOT NULL,
	story_id int4 NULL,
	project_id int4 NULL,
	title varchar(255) NULL,
	description text NULL,
	status_id int4 NULL,
	created_by int4 NULL,
	created_at timestamp NULL,
	estimated_time interval NULL,
	archived_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	modified_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT user_story_history_pkey PRIMARY KEY (history_id)
);
```

3.User Story

```
CREATE TABLE public.user_story (
	story_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	project_id int4 NULL,
	title varchar(255) NOT NULL,
	description text NULL,
	status_id int4 NULL,
	created_by int4 NULL,
	created_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	estimated_time interval NULL,
	modified_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT user_story_pkey PRIMARY KEY (story_id),
	CONSTRAINT user_story_created_by_fkey FOREIGN KEY (created_by) REFERENCES
public.users(user_id),
	CONSTRAINT user_story_project_id_fkey FOREIGN KEY (project_id) REFERENCES
public.projects(project_id) ON DELETE CASCADE,
	CONSTRAINT user_story_status_id_fkey FOREIGN KEY (status_id) REFERENCES
public.status(status_id)
);
```

4.User Projects

```
CREATE TABLE public.user_projects (
	user_id int4 NOT NULL,
	project_id int4 NOT NULL,
	CONSTRAINT user_projects_pkey PRIMARY KEY (user_id, project_id),
	CONSTRAINT user_projects_project_id_fkey FOREIGN KEY (project_id)
REFERENCES public.projects(project_id) ON DELETE CASCADE,
	CONSTRAINT user_projects_user_id_fkey FOREIGN KEY (user_id) REFERENCES
public.users(user_id) ON DELETE CASCADE
);
```

5.User History

```
CREATE TABLE public.user_history (
	history_id serial4 NOT NULL,
	user_id int4 NOT NULL,
	"name" varchar(100) NULL,
	email varchar(100) NULL,
	role_id int4 NULL,
	"password" varchar(100) NULL,
	age int4 NULL,
	gender varchar(10) NULL,
	blood_group varchar(5) NULL,
	joined_at timestamp NULL,
	modified_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	department_id int4 NULL,
	changed_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT user_history_pkey PRIMARY KEY (history_id),
	CONSTRAINT user_history_user_id_fk FOREIGN KEY (user_id) REFERENCES
public.users(user_id)
);
```

6.Time Tracking

```
CREATE TABLE public.time_tracking (
	time_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	story_id int4 NULL,
	user_id int4 NULL,
	log_date date DEFAULT CURRENT_DATE NULL,
	hours_logged interval NULL,
	CONSTRAINT time_tracking_pkey PRIMARY KEY (time_id),
	CONSTRAINT time_tracking_story_id_fkey FOREIGN KEY (story_id) REFERENCES
public.user_story(story_id) ON DELETE CASCADE,
	CONSTRAINT time_tracking_user_id_fkey FOREIGN KEY (user_id) REFERENCES
public.users(user_id) ON DELETE CASCADE
);
```

7.Status

```
CREATE TABLE public.status (
	status_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	"type" varchar(100) NOT NULL,
	CONSTRAINT status_pkey PRIMARY KEY (status_id)
);
```

8.Roles

```
CREATE TABLE public.roles (
	role_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	"name" varchar(100) NOT NULL,
	CONSTRAINT roles_pkey PRIMARY KEY (role_id)
);
```

9.Role Permissions

```
CREATE TABLE public.role_permissions (
	role_id int4 NOT NULL,
	permission_id int4 NOT NULL,
	CONSTRAINT role_permissions_pkey PRIMARY KEY (role_id, permission_id),
	CONSTRAINT role_permissions_permission_id_fkey FOREIGN KEY
(permission_id) REFERENCES public.permissions(permission_id) ON DELETE CASCADE,
	CONSTRAINT role_permissions_role_id_fkey FOREIGN KEY (role_id) REFERENCES
public.roles(role_id) ON DELETE CASCADE
);
```

10.Projects

```
CREATE TABLE public.projects (
	project_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	title varchar(100) NOT NULL,
	status_id int4 NULL,
	created_by int4 NULL,
	created_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	deadline date NULL,
	modified_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	active bool NULL,
	CONSTRAINT projects_pkey PRIMARY KEY (project_id),
	CONSTRAINT projects_created_by_fkey FOREIGN KEY (created_by) REFERENCES
public.users(user_id),
	CONSTRAINT projects_status_id_fkey FOREIGN KEY (status_id) REFERENCES
public.status(status_id)
);
-- Table Triggers
create trigger trigger_update_active before
insert
   or
update
   on
   public.projects for each row execute function update_active_flag();
```

11.Project History

```
CREATE TABLE public.projects_history (
	projects_history_id serial4 NOT NULL,
	project_id int4 NULL,
	title varchar(100) NULL,
	status_id int4 NULL,
	created_by int4 NULL,
	created_at timestamp NULL,
	deadline date NULL,
	modified_at timestamp NULL,
	CONSTRAINT projects_history_pkey PRIMARY KEY (projects_history_id)
);
```

12.Project Attachments

```
CREATE TABLE public.project_attachments (
	id serial4 NOT NULL,
	project_id int4 NOT NULL,
	attachment_id int4 NOT NULL,
	user_story_id int4 NULL,
	CONSTRAINT project_attachments_pkey PRIMARY KEY (id),
	CONSTRAINT fk_project_attachments_user_story FOREIGN KEY (user_story_id)
REFERENCES public.user_story(story_id) ON DELETE SET NULL,
	CONSTRAINT project_attachments_attachment_id_fkey FOREIGN KEY
(attachment_id) REFERENCES public.attachments(attachment_id) ON DELETE CASCADE,
	CONSTRAINT project_attachments_project_id_fkey FOREIGN KEY (project_id)
REFERENCES public.projects(project_id) ON DELETE CASCADE
);
```

13.Permissions

```
CREATE TABLE public.permissions (
	permission_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE
1 MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	"name" varchar(100) NOT NULL,
	CONSTRAINT permissions_pkey PRIMARY KEY (permission_id)
);
```

14.Events

```
CREATE TABLE public.events (
	event_id serial4 NOT NULL,
	user_id int4 NOT NULL,
	title varchar(255) NOT NULL,
	deadline date NOT NULL,
	active bool DEFAULT true NULL,
	CONSTRAINT events_pkey PRIMARY KEY (event_id),
	CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES public.users(user_id)
ON DELETE CASCADE
);
-- Table Triggers
create trigger trigger_update_event_active before
insert
   or
update
   on
   public.events for each row execute function update_event_active_status();
```

15.Departments

```
CREATE TABLE public.departments (
	department_id serial4 NOT NULL,
	"name" varchar(100) NOT NULL,
	CONSTRAINT departments_pkey PRIMARY KEY (department_id)
);
```

16.Comments

```
CREATE TABLE public."comment" (
	comment_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1 
MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	story_id int4 NULL,
	user_id int4 NULL,
	comment_text text NOT NULL,
	comment_time timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT comment_pkey PRIMARY KEY (comment_id),
	CONSTRAINT comment_story_id_fkey FOREIGN KEY (story_id) REFERENCES 
public.user_story(story_id) ON DELETE CASCADE,
	CONSTRAINT comment_user_id_fkey FOREIGN KEY (user_id) REFERENCES 
public.users(user_id) ON DELETE CASCADE
);
```

17.Attachments

```
CREATE TABLE public.attachments (
	attachment_id serial4 NOT NULL,
	filename text NOT NULL,
	file_type text NOT NULL,
	created_by int4 NOT NULL,
	created_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT attachments_pkey PRIMARY KEY (attachment_id),
	CONSTRAINT unique_attachment_per_user UNIQUE (filename, created_by)
);
```

18.Assignment

```
CREATE TABLE public."assignment" (
	assignment_id int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 
1 MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,
	story_id int4 NULL,
	user_id int4 NULL,
	assigned_at timestamp DEFAULT CURRENT_TIMESTAMP NULL,
	CONSTRAINT assignment_pkey PRIMARY KEY (assignment_id),
	CONSTRAINT assignment_story_id_user_id_key UNIQUE (story_id, user_id),
	CONSTRAINT assignment_story_id_fkey FOREIGN KEY (story_id) REFERENCES 
public.user_story(story_id) ON DELETE CASCADE,
	CONSTRAINT assignment_user_id_fkey FOREIGN KEY (user_id) REFERENCES 
public.users(user_id) ON DELETE CASCADE
);
```

Stored Procedures:

1. To verify user credentials:

```sql
-- DROP FUNCTION public.verify_user_credentials(text, text);
CREATE OR REPLACE FUNCTION public.verify_user_credentials(p_email text,
p_password text)
RETURNS TABLE(user_id integer, name character varying, email character
varying, role_id integer)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT u.user_id, u.name, u.email, u.role_id
   FROM users u
   WHERE u.email = p_email
   AND u.password = crypt(p_password, u.password);
END;
$function$
;
```

2. Upload attachment:

```sql
-- DROP FUNCTION public.upload_attachment_record(text, text, int4);
CREATE OR REPLACE FUNCTION public.upload_attachment_record(p_name text,
p_file_type text, p_created_by integer)
RETURNS integer
LANGUAGE plpgsql
AS $function$
DECLARE
   existing_id INTEGER;
BEGIN
   -- Check for existing attachment
   SELECT attachment_id
   INTO existing_id
   FROM attachments
   WHERE filename = p_name AND created_by = p_created_by
   LIMIT 1;
   -- If it exists, return it
   IF existing_id IS NOT NULL THEN
       RETURN existing_id;
   END IF;
   -- Otherwise, insert new and return new id
   INSERT INTO attachments(filename, file_type, created_by)
   VALUES (p_name, p_file_type, p_created_by)
   RETURNING attachment_id INTO existing_id;
   RETURN existing_id;
END;
```

```
$function$
;
```

3. Update user:

```
-- DROP FUNCTION public.update_user_with_history(int4, varchar, varchar, int4,
varchar, varchar, varchar);
CREATE OR REPLACE FUNCTION public.update_user_with_history(p_user_id integer,
p_name character varying, p_email character varying, p_age integer, p_gender
character varying, p_blood_group character varying, p_department_name
character varying)
RETURNS text
LANGUAGE plpgsql
AS $function$
DECLARE
 v_department_id INT;
BEGIN
 -- Step 1: Get department_id from department_name
 SELECT department_id INTO v_department_id
 FROM departments
 WHERE name = p_department_name;
 -- If department not found, raise error
 IF NOT FOUND THEN
   RAISE EXCEPTION 'Department "%" not found.', p_department_name;
 END IF;
 -- Step 2: Insert or update history
 IF EXISTS (SELECT 1 FROM user_history WHERE user_id = p_user_id) THEN
   -- Update history
   UPDATE user_history
   SET
     name = u.name,
     email = u.email,
     role_id = u.role_id,
     password = u.password,
     age = u.age,
     gender = u.gender,
     blood_group = u.blood_group,
     joined_at = u.joined_at,
     modified_at = CURRENT_TIMESTAMP,
     department_id = u.department_id
   FROM users u
   WHERE user_history.user_id = p_user_id AND u.user_id = p_user_id;
 ELSE
   -- Insert history
   INSERT INTO user_history (
     user_id, name, email, role_id, password, age, gender,
     blood_group, joined_at, modified_at, department_id
   )
   SELECT
     user_id, name, email, role_id, password, age, gender,
     blood_group, joined_at, modified_at, department_id
   FROM users
   WHERE user_id = p_user_id;
 END IF;
 -- Step 3: Update users table
 UPDATE users
 SET
   name = p_name,

   age = p_age,
   gender = p_gender,
   blood_group = p_blood_group,
   department_id = v_department_id,
   modified_at = CURRENT_TIMESTAMP
 WHERE user_id = p_user_id;
 RETURN 'User updated successfully.';
END;
$function$
;
```

4. Update user story:

```
-- DROP FUNCTION public.update_user_story_with_history(int4, varchar, text,
int4, interval);
CREATE OR REPLACE FUNCTION public.update_user_story_with_history(p_story_id
integer, p_title character varying, p_description text, p_status_id integer,
p_estimated_time interval)
RETURNS text
LANGUAGE plpgsql
AS $function$
DECLARE
   v_story user_story%ROWTYPE;
BEGIN
   -- 1. Store current record in variable
   SELECT * INTO v_story FROM user_story WHERE story_id = p_story_id;
   -- 2. Archive old record to history
   INSERT INTO user_story_history (
       story_id, project_id, title, description, status_id,
       created_by, created_at, estimated_time, modified_at
   )
   VALUES (
       v_story.story_id, v_story.project_id, v_story.title,
v_story.description,
       v_story.status_id, v_story.created_by, v_story.created_at,
       v_story.estimated_time, CURRENT_TIMESTAMP
   );
   -- 3. Update existing record
   UPDATE user_story
   SET
       title = p_title,
       description = p_description,
       status_id = p_status_id,
       estimated_time = p_estimated_time,
       modified_at = CURRENT_TIMESTAMP
   WHERE story_id = p_story_id;
   RETURN 'User story updated with history recorded';
END;
$function$
;
```

5. Update project:

```
-- DROP FUNCTION public.update_project_with_versioning(int4, varchar, date);
CREATE OR REPLACE FUNCTION public.update_project_with_versioning(p_project_id
integer, p_new_title character varying, p_new_deadline date)
RETURNS text
LANGUAGE plpgsql
AS $function$
DECLARE
    old_project RECORD;
BEGIN
    -- Step 1: Fetch the current project details
    SELECT * INTO old_project
    FROM projects
    WHERE project_id = p_project_id;
    IF NOT FOUND THEN
        RETURN 'Project not found';
    END IF;
    -- Step 2: Archive the old version into the history table
    INSERT INTO projects_history (
        project_id,
        title,
        status_id,
        created_by,
        created_at,
        deadline,
        modified_at
    )
    VALUES (
        old_project.project_id,
        old_project.title,
        old_project.status_id,
        old_project.created_by,
        old_project.created_at,
        old_project.deadline,
        old_project.modified_at
    );
    -- Step 3: Update the original project record
    UPDATE projects
    SET title = p_new_title,
        deadline = p_new_deadline,
        modified_at = CURRENT_TIMESTAMP
    WHERE project_id = p_project_id;
    RETURN 'Project updated and version archived successfully';
 END;
 $function$
 ;

 -- DROP PROCEDURE public.update_event_deadline(int4, date);
 CREATE OR REPLACE PROCEDURE public.update_event_deadline(IN p_event_id integer,
 IN p_new_deadline date)
 LANGUAGE plpgsql
 AS $procedure$
 BEGIN
  UPDATE events
  SET deadline = p_new_deadline
  WHERE event_id = p_event_id;
  IF NOT FOUND THEN
    RAISE NOTICE 'No event found with ID % for update.', p_event_id;
  ELSE
    RAISE NOTICE 'Event % deadline updated to %.', p_event_id, p_new_deadline;
  END IF;
 END;
 $procedure$
 ;
```

6. Update event active status:

```
-- DROP FUNCTION public.update_event_active_status();
CREATE OR REPLACE FUNCTION public.update_event_active_status()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
BEGIN
 -- Set active to TRUE if deadline is today or later
 IF NEW.deadline >= CURRENT_DATE THEN
   NEW.active := TRUE;
 ELSE
   NEW.active := FALSE;
 END IF;
 RETURN NEW;
END;
$function$
;
```

7. Update active flag in projects:

```
-- DROP FUNCTION public.update_active_flag();
CREATE OR REPLACE FUNCTION public.update_active_flag()
RETURNS trigger
LANGUAGE plpgsql
```

```
AS $function$
BEGIN
 NEW.active := NEW.deadline > CURRENT_DATE;
 RETURN NEW;
END;
$function$
;
```

8. Update user story status:

```
-- DROP FUNCTION public.sp_update_user_story_status(int4, int4);
CREATE OR REPLACE FUNCTION public.sp_update_user_story_status(p_story_id
integer, p_status_id integer)
RETURNS void
LANGUAGE plpgsql
AS $function$
BEGIN
 UPDATE user_story
 SET status_id = p_status_id,
     modified_at = CURRENT_TIMESTAMP
 WHERE story_id = p_story_id;
END;
$function$
;
```

9. Get project's user stories:

```
-- DROP FUNCTION public.sp_get_user_stories_by_project(int4);
CREATE OR REPLACE FUNCTION public.sp_get_user_stories_by_project(p_project_id
integer)
RETURNS TABLE(story_id integer, title character varying, description text,
estimated_time interval, created_at timestamp without time zone, modified_at
timestamp without time zone, status_id integer, type character varying,
created_by integer, project_id integer, project_title character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
 RETURN QUERY
 SELECT
   us.story_id,
   us.title,
   us.description,
   us.estimated_time,
   us.created_at,
   us.modified_at,
   us.status_id,
   s.type,
   us.created_by,
   us.project_id,
   p.title AS project_title
 FROM
   user_story us
   LEFT JOIN status s ON us.status_id = s.status_id
   LEFT JOIN projects p ON us.project_id = p.project_id
 WHERE
   us.project_id = p_project_id;
END;
$function$
;
```

10. Get project's attachments:

```
-- DROP FUNCTION public.sp_get_attachments_by_project(int4);
CREATE OR REPLACE FUNCTION public.sp_get_attachments_by_project(p_project_id
integer)
RETURNS TABLE(attachment_id integer, file_name character varying, path text,
uploaded_by integer, uploaded_at timestamp without time zone, story_id integer,
project_id integer)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       a.attachment_id,
       a.file_name,
       a.path,
       a.uploaded_by,
       a.uploaded_at,
       a.story_id,
       a.project_id
   FROM
       attachment a
   WHERE
       a.project_id = p_project_id;
END;
$function$
;
```

11. Search projects by title:

```
-- DROP FUNCTION public.search_projects_by_title(text);
CREATE OR REPLACE FUNCTION public.search_projects_by_title(p_query text)
RETURNS TABLE(project_id integer, title character varying, created_by
character varying, created_at timestamp without time zone, status character
varying, deadline date)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
```

```
        p.project_id,
        p.title,
        u.name AS created_by,
        p.created_at,
        s.type AS status,
        p.deadline
    FROM projects p
    JOIN users u ON p.created_by = u.user_id
    JOIN status s ON p.status_id = s.status_id
    WHERE LOWER(p.title) LIKE '%' || LOWER(p_query) || '%';
END;
$function$
;
```

12. Search members:

```
-- DROP FUNCTION public.search_members(text);
CREATE OR REPLACE FUNCTION public.search_members(query text)
RETURNS json
LANGUAGE plpgsql
AS $function$
BEGIN
 RETURN (
    SELECT json_agg(u)
    FROM (
      SELECT
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        d.name AS department,
        u.joined_at,
        u.modified_at
      FROM users u
      JOIN departments d ON u.department_id = d.department_id
      WHERE LOWER(u.name) LIKE '%' || LOWER(query) || '%'
    ) u
 );
END;
$function$
;
```

13. Insert Comments:

```
-- DROP PROCEDURE public.insert_comment(int4, int4, text);
```

```
CREATE OR REPLACE PROCEDURE public.insert_comment(IN p_story_id integer, IN
p_user_id integer, IN p_comment_text text)
LANGUAGE plpgsql
AS $procedure$
BEGIN
   INSERT INTO public."comment" (story_id, user_id, comment_text)
   VALUES (p_story_id, p_user_id, p_comment_text);
END;
$procedure$
;
```

14. Add event in calendar:

```
-- DROP FUNCTION public.add_event(int4, varchar, date);
CREATE OR REPLACE FUNCTION public.add_event(p_user_id integer, p_title
character varying, p_deadline date)
RETURNS void
LANGUAGE plpgsql
AS $function$
BEGIN
   INSERT INTO events (user_id, title, deadline)
   VALUES (p_user_id, p_title, p_deadline);
END;
$function$
;
```

15. Add project

```
-- DROP FUNCTION public.add_project(text, int4, int4, timestamp);
CREATE OR REPLACE FUNCTION public.add_project(p_title text, p_status_id
integer, p_created_by integer, p_deadline timestamp without time zone)
RETURNS integer
LANGUAGE plpgsql
AS $function$
DECLARE
   new_project_id INTEGER;
BEGIN
   INSERT INTO projects (title, status_id, created_by, created_at, deadline)
   VALUES (p_title, p_status_id, p_created_by, NOW(), p_deadline)
   RETURNING project_id INTO new_project_id;
   RETURN new_project_id;
END;
$function$
;
```

16. Add user story:

```
-- DROP FUNCTION public.add_user_story(int4, varchar, text, int4, int4,
interval);
```

```
CREATE OR REPLACE FUNCTION public.add_user_story(p_project_id integer, p_title
character varying, p_description text, p_status_id integer, p_created_by
integer, p_estimated_time interval)
RETURNS integer
LANGUAGE plpgsql
AS $function$
DECLARE
   new_story_id INT;
BEGIN
   INSERT INTO user_story (
       project_id,
       title,
       description,
       status_id,
       created_by,
       estimated_time
   )
   VALUES (
       p_project_id,
       p_title,
       p_description,
       p_status_id,
       p_created_by,
       p_estimated_time
   )
   RETURNING story_id INTO new_story_id;
   RETURN new_story_id;
END;
$function$
;
```

17. Assign attachment(s) to story:

```
-- DROP FUNCTION public.assign_attachment_to_story(int4, int4, int4);
CREATE OR REPLACE FUNCTION public.assign_attachment_to_story(p_attachment_id
integer, p_project_id integer, p_user_story_id integer)
RETURNS text
LANGUAGE plpgsql
AS $function$
BEGIN
   -- First try to update an existing record
   UPDATE project_attachments
   SET user_story_id = p_user_story_id
   WHERE project_id = p_project_id
     AND attachment_id = p_attachment_id;
   IF FOUND THEN
       RETURN 'Attachment successfully assigned to user story.';
   ELSE
       -- If no record was updated, insert a new one
```

```
        INSERT INTO project_attachments (attachment_id, project_id,
user_story_id)
        VALUES (p_attachment_id, p_project_id, p_user_story_id);
        RETURN 'No existing record found. New assignment created successfully.';
    END IF;
EXCEPTION
    WHEN unique_violation THEN
        RETURN 'Error: Duplicate assignment attempted. Record already exists.';
    WHEN foreign_key_violation THEN
        RETURN 'Error: The specified attachment_id or user_story_id does not
exist. Assignment failed.';
    WHEN OTHERS THEN
        RETURN 'An unexpected error occurred: ' || SQLERRM;
END;
$function$
;
```

18. Assign attachment(s) to project:

```
-- DROP PROCEDURE public.assign_attachments_to_project(int4, _int4);
CREATE OR REPLACE PROCEDURE public.assign_attachments_to_project(IN
p_project_id integer, IN p_attachment_ids integer[])
LANGUAGE plpgsql
AS $procedure$
DECLARE
    aid INTEGER;
BEGIN
    FOREACH aid IN ARRAY p_attachment_ids
    LOOP
        -- check if record already exists (with NULL user_story_id allowed)
        IF NOT EXISTS (
            SELECT 1
            FROM project_attachments
            WHERE project_id = p_project_id
              AND attachment_id = aid
              AND user_story_id IS NULL
        ) THEN
            INSERT INTO project_attachments (project_id, attachment_id,
user_story_id)
            VALUES (p_project_id, aid, NULL);
        END IF;
    END LOOP;
END;
$procedure$
;
```

19. Assign member to project:

```
-- DROP FUNCTION public.assign_member_to_project(int4, int4);
```

```
CREATE OR REPLACE FUNCTION public.assign_member_to_project(p_project_id
integer, p_user_id integer)
RETURNS text
LANGUAGE plpgsql
AS $function$
BEGIN
   -- Insert assignment
   INSERT INTO user_projects (project_id, user_id)
   VALUES (p_project_id, p_user_id);
   RETURN 'Member assigned successfully';
END;
$function$
;
```

20. Assign user to story:

```
-- DROP FUNCTION public.assign_user_to_story(int4, int4);
CREATE OR REPLACE FUNCTION public.assign_user_to_story(p_story_id integer,
p_user_id integer)
RETURNS text
LANGUAGE plpgsql
AS $function$
BEGIN
   INSERT INTO assignment (story_id, user_id)
   VALUES (p_story_id, p_user_id)
   ON CONFLICT (story_id, user_id) DO NOTHING;
   RETURN 'User assigned successfully';
END;
$function$
;
```

21. Change user password:

```
-- DROP PROCEDURE public.change_user_password(text, text, text);
CREATE OR REPLACE PROCEDURE public.change_user_password(IN p_email text, IN
p_current_password text, IN p_new_password text)
LANGUAGE plpgsql
AS $procedure$
BEGIN
 -- Step 1: Check if current password is correct
 IF EXISTS (
   SELECT 1 FROM users
   WHERE email = p_email
   AND password = crypt(p_current_password, password)
 ) THEN
   -- Step 2: Update to new hashed password
   UPDATE users
   SET password = crypt(p_new_password, gen_salt('bf')),
       modified_at = CURRENT_TIMESTAMP
   WHERE email = p_email;
 ELSE
   -- Step 3: Raise error if current password is invalid
   RAISE EXCEPTION 'Current password is incorrect';
 END IF;
END;
$procedure$
;
```

22. Create User:

```
-- DROP FUNCTION public.create_user(varchar, varchar, varchar, int4, varchar,
varchar, varchar);
CREATE OR REPLACE FUNCTION public.create_user(p_name character varying, p_email
character varying, p_password character varying, p_age integer, p_gender
character varying, p_blood_group character varying, p_department_name
character varying)
RETURNS void
LANGUAGE plpgsql
AS $function$
DECLARE
   dept_id INT;
BEGIN
   SELECT department_id INTO dept_id
   FROM departments
   WHERE name = p_department_name;
   IF dept_id IS NULL THEN
       RAISE EXCEPTION 'Department "%" not found', p_department_name;
   END IF;
   INSERT INTO users (
       name, email, password, age, gender, blood_group, department_id
   ) VALUES (
       p_name,
       p_email,
       crypt(p_password, gen_salt('bf')),  -- hashed password
       p_age,
       p_gender,
       p_blood_group,
       dept_id
   );
END;
$function$
;
```

23. Delete event from calendar by id:

```
-- DROP PROCEDURE public.delete_event_by_id(int4);
CREATE OR REPLACE PROCEDURE public.delete_event_by_id(IN p_event_id integer)
```

```
LANGUAGE plpgsql
AS $procedure$
BEGIN
 DELETE FROM events
 WHERE event_id = p_event_id;
 IF NOT FOUND THEN
   RAISE NOTICE 'No event found with ID % for deletion.', p_event_id;
 ELSE
   RAISE NOTICE 'Event % deleted.', p_event_id;
 END IF;
END;
$procedure$
;
```

24. Get all departments:

```
-- DROP FUNCTION public.get_all_departments();
CREATE OR REPLACE FUNCTION public.get_all_departments()
RETURNS TABLE(department_id integer, name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       d.department_id,
       d.name
   FROM departments d
   ORDER BY d.name;
END;
$function$
;
```

25. Get all projects:

```
-- DROP FUNCTION public.get_all_project_summary();
CREATE OR REPLACE FUNCTION public.get_all_project_summary()
RETURNS TABLE(project_id integer, title character varying, created_by character
varying, created_at timestamp without time zone, status character varying,
deadline date)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       p.project_id,
       p.title,
       u.name AS created_by,
       p.created_at,
       s.type AS status,
       p.deadline
   FROM projects p
   JOIN users u ON p.created_by = u.user_id
   JOIN status s ON p.status_id = s.status_id
   ORDER BY p.created_at DESC;
END;
$function$
;
```

26. Get all projects in alphabetical order:

```
-- DROP FUNCTION public.get_all_project_summary_alphabetical();
CREATE OR REPLACE FUNCTION public.get_all_project_summary_alphabetical()
RETURNS TABLE(project_id integer, title character varying, created_by character
varying, created_at timestamp without time zone, status character varying,
deadline date)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        p.project_id,
        p.title,
        u.name AS created_by,
        p.created_at,
        s.type AS status,
        p.deadline
    FROM projects p
    JOIN users u ON p.created_by = u.user_id
    JOIN status s ON p.status_id = s.status_id
    ORDER BY p.title ASC;  -- 🔤 Alphabetically
END;
$function$
;
```

27. Get all users:

```
-- DROP FUNCTION public.get_all_users();
CREATE OR REPLACE FUNCTION public.get_all_users()
RETURNS TABLE(user_id integer, name character varying, email character
varying, role_id integer, age integer, gender character varying, blood_group
character varying, joined_at timestamp without time zone, modified_at
timestamp without time zone, department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        u.user_id,
        u.name,
        u.email,
        u.role_id,
        u.age,
        u.gender,
        u.blood_group,
        u.joined_at,
        u.modified_at,
        d.name AS department_name
    FROM users u
    LEFT JOIN departments d ON u.department_id = d.department_id;
END;
$function$
;
```

28. Get assigned members to story:

```
-- DROP FUNCTION public.get_assigned_members(int4);
CREATE OR REPLACE FUNCTION public.get_assigned_members(p_story_id integer)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
department_name character varying, joined_at timestamp without time zone,
modified_at timestamp without time zone, role_id integer)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        d.name AS department_name,
        u.joined_at,
        u.modified_at,
        u.role_id
    FROM assignment a
    JOIN users u ON a.user_id = u.user_id
    LEFT JOIN departments d ON u.department_id = d.department_id
    WHERE a.story_id = p_story_id;
END;
$function$
;
```

29. Get projects assigned to user:

```
-- DROP FUNCTION public.get_assigned_projects_by_user_id(int4);
```

```
CREATE OR REPLACE FUNCTION public.get_assigned_projects_by_user_id(p_user_id
integer)
RETURNS TABLE(title character varying, deadline date, status_id integer,
created_by_name character varying, created_at timestamp without time zone)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       p.title,
       p.deadline,
       p.status_id,
       u.name AS created_by_name,
       p.created_at
   FROM projects p
   JOIN user_projects up ON p.project_id = up.project_id
   JOIN users u ON u.user_id = p.created_by
   WHERE up.user_id = p_user_id
     AND p.active = true;
END;
$function$
;
```

30. Get project's attachments:

```
-- DROP FUNCTION public.get_attachments_by_project(int4);
CREATE OR REPLACE FUNCTION public.get_attachments_by_project(p_project_id
integer)
RETURNS TABLE(project_id integer, attachment_id integer, name character
varying, file_type character varying, created_by_name character varying,
created_by integer)
LANGUAGE plpgsql
AS $function$
BEGIN
 RETURN QUERY
 SELECT
   pa.project_id,
   a.attachment_id,
   a.filename::varchar AS name,
   a.file_type::varchar AS file_type,
   u.name::varchar AS created_by_name,
   u.user_id AS created_by
 FROM attachments a
 JOIN project_attachments pa ON pa.attachment_id = a.attachment_id
 JOIN users u ON a.created_by = u.user_id
 WHERE pa.project_id = p_project_id;
END;
$function$
;
```

31. Get events by user:

```
-- DROP FUNCTION public.get_events_by_user(int4);
CREATE OR REPLACE FUNCTION public.get_events_by_user(p_user_id integer)
RETURNS TABLE(event_id integer, title character varying, deadline date)
LANGUAGE plpgsql
AS $function$
BEGIN
 RETURN QUERY
 SELECT e.event_id, e.title, e.deadline
 FROM events e
 WHERE e.user_id = p_user_id AND e.active = TRUE;
END;
$function$
;
```

32. Get my projects by email:

```
-- DROP FUNCTION public.get_my_projects_by_email(varchar);
CREATE OR REPLACE FUNCTION public.get_my_projects_by_email(p_email character
varying)
RETURNS TABLE(project_id integer, title character varying, status_id integer,
created_at timestamp without time zone, deadline date, modified_at timestamp
without time zone)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       p.project_id,
       p.title,
       p.status_id,
       p.created_at,
       p.deadline,
       p.modified_at
   FROM projects p
   JOIN user_projects up ON p.project_id = up.project_id
   JOIN users u ON up.user_id = u.user_id
   WHERE u.email = p_email
   UNION
   SELECT
       p.project_id,
       p.title,
       p.status_id,
       p.created_at,
       p.deadline,
       p.modified_at
   FROM projects p
```

```
    JOIN users u ON p.created_by = u.user_id
    WHERE u.email = p_email;
END;
$function$
;
```

33. Get project members:

```
-- DROP FUNCTION public.get_project_members(int4);
CREATE OR REPLACE FUNCTION public.get_project_members(p_project_id integer)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        d.name AS department_name
    FROM users u
    JOIN user_projects up ON u.user_id = up.user_id
    LEFT JOIN departments d ON u.department_id = d.department_id
    WHERE up.project_id = p_project_id;
END;
$function$
;
```

34. Get projects created by user:(email)

```
-- DROP FUNCTION public.get_projects_created_by_email(varchar);
CREATE OR REPLACE FUNCTION public.get_projects_created_by_email(p_email
character varying)
RETURNS TABLE(project_id integer, title character varying, status_id integer,
created_at timestamp without time zone, deadline date, modified_at timestamp
without time zone)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        p.project_id,
        p.title,
        p.status_id,
        p.created_at,
        p.deadline,
        p.modified_at
    FROM projects p
    JOIN users u ON p.created_by = u.user_id
    WHERE u.email = p_email;
END;
$function$
;
```

35. Get user profile (by email):

```
-- DROP FUNCTION public.get_user_profile_by_email(varchar);
CREATE OR REPLACE FUNCTION public.get_user_profile_by_email(p_email character
varying)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
joined_at timestamp without time zone, modified_at timestamp without time
zone, department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        u.joined_at,
        u.modified_at,
        d.name AS department_name
    FROM users u
    LEFT JOIN departments d ON u.department_id = d.department_id
    WHERE u.email = p_email;
END;
$function$
;
```

36. Get user profile (by id):

```
-- DROP FUNCTION public.get_user_profile_by_id(int4);
CREATE OR REPLACE FUNCTION public.get_user_profile_by_id(p_user_id integer)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
joined_at timestamp without time zone, modified_at timestamp without time
zone, department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
```

```
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        u.joined_at,
        u.modified_at,
        d.name AS department_name
    FROM users u
    LEFT JOIN departments d ON u.department_id = d.department_id
    WHERE u.user_id = p_user_id;
END;
$function$
;
```

37. Get user story details:

```
-- DROP FUNCTION public.get_user_story_details(int4);
CREATE OR REPLACE FUNCTION public.get_user_story_details(p_story_id integer)
RETURNS json
LANGUAGE plpgsql
AS $function$
DECLARE
   result JSON;
BEGIN
   SELECT json_build_object(
       'story', (
           SELECT row_to_json(s)
           FROM (
               SELECT
                   us.story_id,
                   us.title,
                   us.description,
                   st.status_id,
                   st.type AS status_type,
                   us.created_at,
                   us.modified_at,
                   us.estimated_time,
                   u.user_id,
                   u.name AS created_by,
                   u.email AS creator_email
               FROM user_story us
               JOIN users u ON us.created_by = u.user_id
               JOIN status st ON st.status_id = us.status_id
               WHERE us.story_id = p_story_id
           ) s
       ),
       'comments', (
           SELECT json_agg(row_to_json(c))
           FROM (
               SELECT
                   c.comment_id,
                   c.comment_text,
                   c.comment_time,
                   u.name AS commented_by,
                   u.email AS commenter_email
               FROM comment c
               JOIN users u ON c.user_id = u.user_id
               WHERE c.story_id = p_story_id
           ) c
       ),
       'attachments', (
           -- MODIFIED SECTION: This subquery now correctly joins through the
           -- project_attachments table to find attachments linked to the user
story.
           SELECT json_agg(row_to_json(att))
           FROM (
               SELECT
                   a.attachment_id,
                   a.filename,
                   a.file_type,
                   a.created_at AS uploaded_at, -- Assuming 'created_at' is the
upload timestamp in the attachments table
                   u.name AS uploaded_by,
                   u.user_id AS uploaded_by_id
               FROM project_attachments pa
               -- Join to get the attachment details (filename, type, etc.)
               JOIN attachments a ON pa.attachment_id = a.attachment_id
               -- Join to get the name of the user who created the attachment
               JOIN users u ON a.created_by = u.user_id
               -- Filter for the specific user story ID
               WHERE pa.user_story_id = p_story_id
           ) att
       ),
       'time_tracking', (
           SELECT json_agg(row_to_json(t))
           FROM (
               SELECT
                   t.time_id,
                   t.log_date,
                   t.hours_logged,
                   u.name AS logged_by,
                   u.email AS logger_email
               FROM time_tracking t
               JOIN users u ON t.user_id = u.user_id
               WHERE t.story_id = p_story_id
           ) t
       )
   ) INTO result;
   RETURN result;
END;
$function$
;
```

38. Get users not assigned to a project:

```
-- DROP FUNCTION public.get_users_not_assigned_to_project(int4);
CREATE OR REPLACE FUNCTION
public.get_users_not_assigned_to_project(p_project_id integer)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
joined_at timestamp without time zone, modified_at timestamp without time
zone, department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
    RETURN QUERY
    SELECT
        u.user_id,
        u.name,
        u.email,
        u.age,
        u.gender,
        u.blood_group,
        u.joined_at,
        u.modified_at,
        d.name AS department_name
    FROM users u
    LEFT JOIN departments d ON u.department_id = d.department_id
    WHERE u.user_id NOT IN (
        SELECT up.user_id
        FROM user_projects up
        WHERE up.project_id = p_project_id
    );
END;
$function$
;
```

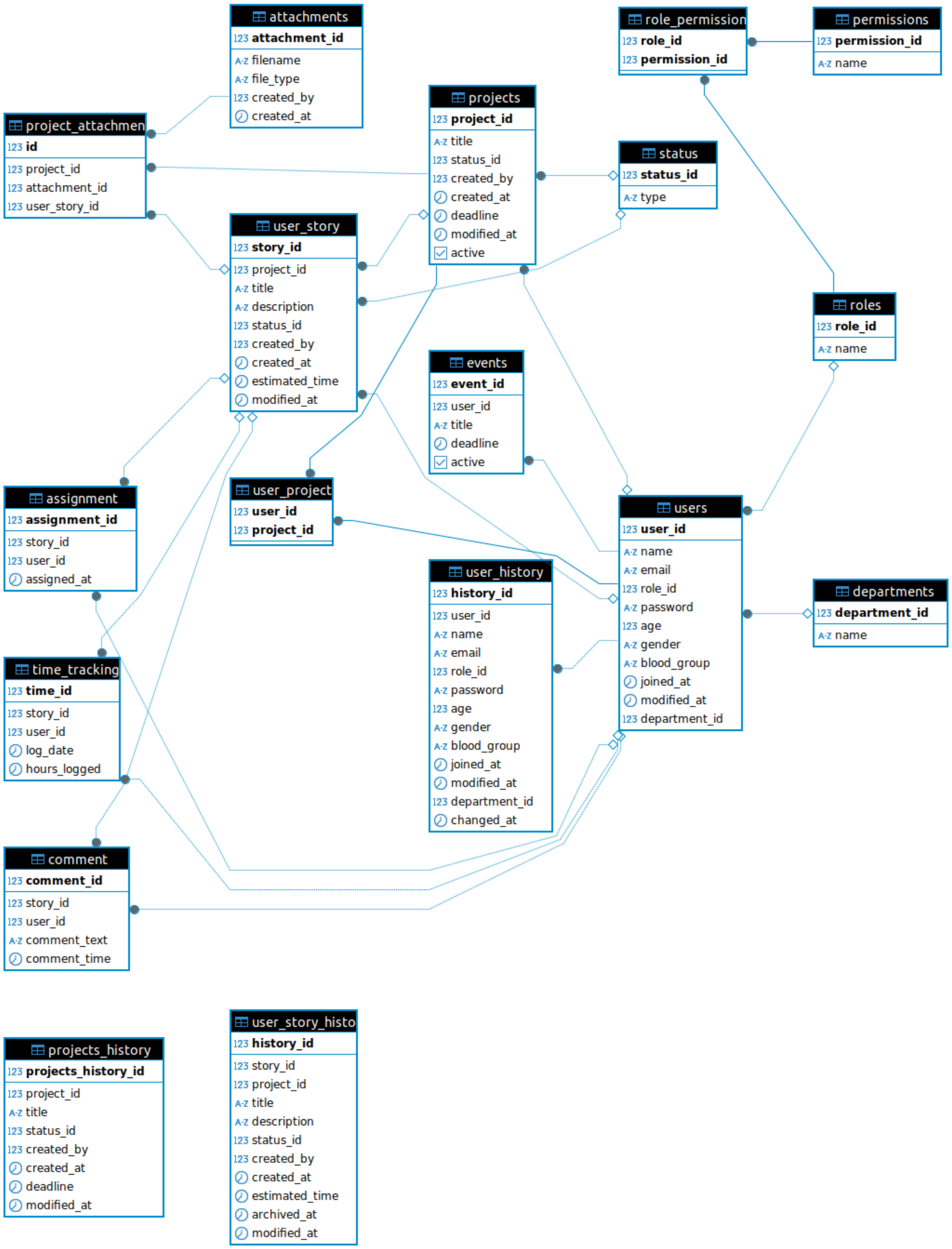
39. Get users not assigned to story:

```
-- DROP FUNCTION public.get_users_not_assigned_to_story(int4);
CREATE OR REPLACE FUNCTION public.get_users_not_assigned_to_story(p_story_id
integer)
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
joined_at timestamp without time zone, modified_at timestamp without time
zone, department_name character varying)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       u.user_id,
       u.name,
       u.email,
       u.age,
       u.gender,
       u.blood_group,
       u.joined_at,
       u.modified_at,
       d.name AS department_name
   FROM users u
   LEFT JOIN departments d ON u.department_id = d.department_id
   WHERE u.user_id NOT IN (
       SELECT a.user_id
       FROM assignment a
       WHERE a.story_id = p_story_id
   );
END;
$function$
;
```

40. Get users not in projects:

```
-- DROP FUNCTION public.get_users_not_in_user_projects();
CREATE OR REPLACE FUNCTION public.get_users_not_in_user_projects()
RETURNS TABLE(user_id integer, name character varying, email character
varying, age integer, gender character varying, blood_group character varying,
joined_at timestamp without time zone, modified_at timestamp without time
zone)
LANGUAGE plpgsql
AS $function$
BEGIN
   RETURN QUERY
   SELECT
       u.user_id,
       u.name,
       u.email,
       u.age,
       u.gender,
       u.blood_group,
       u.joined_at,
       u.modified_at
   FROM users u
   LEFT JOIN user_projects up ON u.user_id = up.user_id
   WHERE up.project_id IS NULL;
END;
$function$
;
```

ERD:


attachments
attachment_id
filename
file_type
created_by
created_at
role_permission
role_id
permission_id
permissions
permission_id
name
project_attachmen
id
project_id
attachment_id
user_story_id
projects
project_id
title
status_id
created_by
created_at
deadline
modified_at
active
status
status_id
type
user_story
story_id
project_id
title
description
status_id
created_by
created_at
estimated_time
modified_at
roles
role_id
name
events
event_id
user_id
title
deadline
active
user_project
user_id
project_id
assignment
assignment_id
story_id
user_id
assigned_at
users
user_id
name
email
role_id
password
age
gender
blood_group
joined_at
modified_at
department_id
departments
department_id
name
user_history
history_id
user_id
name
email
role_id
password
age
gender
blood_group
joined_at
modified_at
department_id
changed_at
time_tracking
time_id
story_id
user_id
log_date
hours_logged
comment
comment_id
story_id
user_id
comment_text
comment_time
projects_history
projects_history_id
project_id
title
status_id
created_by
created_at
deadline
modified_at
user_story_histo
history_id
story_id
project_id
title
description
status_id
created_by
created_at
estimated_time
archived_at
modified_at