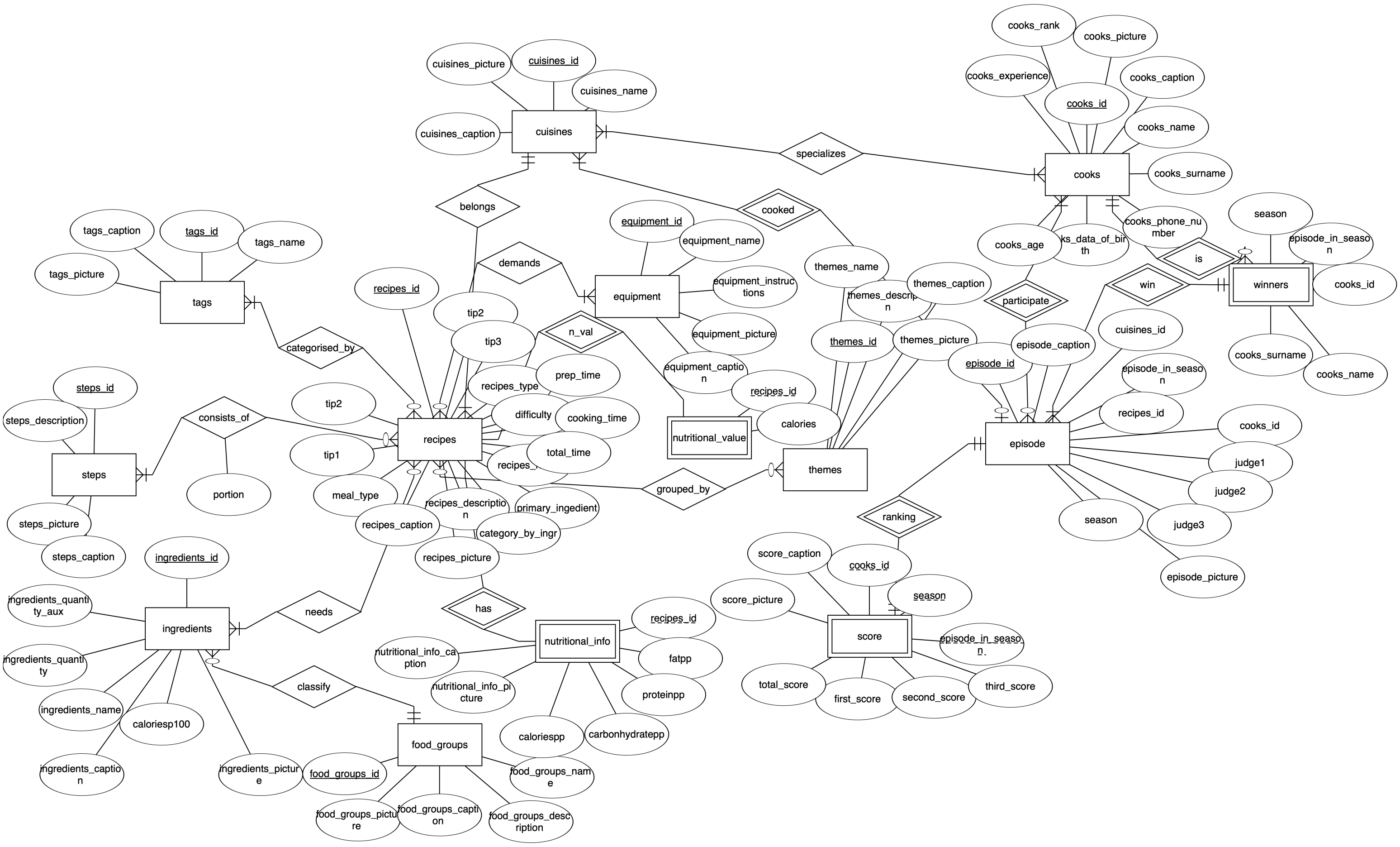
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

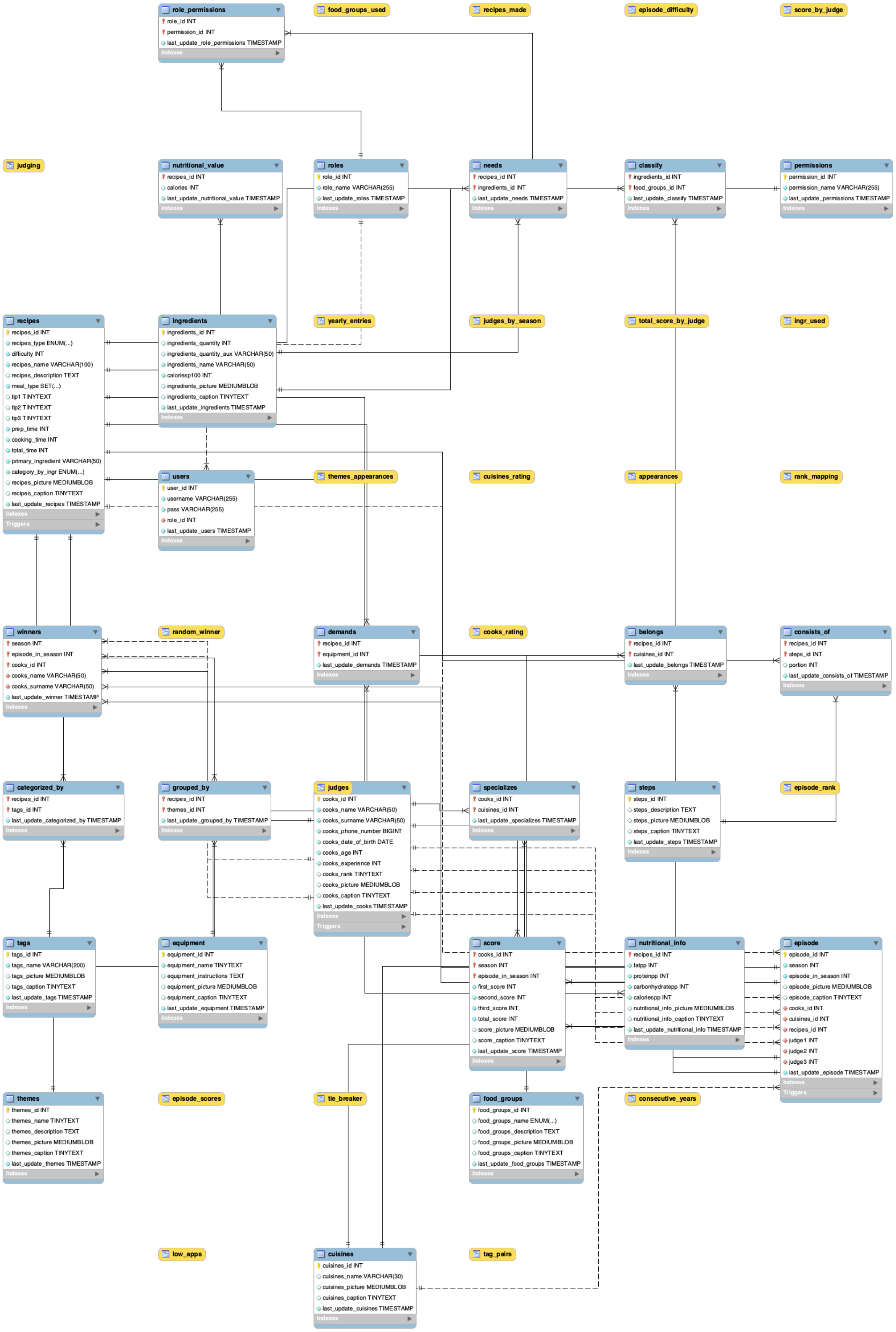
ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ
ΑΝΑΦΟΡΑ ΕΞΑΜΗΝΙΑΙΑΣ ΕΡΓΑΣΙΑΣ
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2023-2024

ΟΜΑΔΑ 83
ΒΑΒΟΥΡΑΚΗΣ ΗΛΙΑΣ: 03121043
ΦΙΛΙΠΠΟΥ ΕΥΑΓΓΕΛΙΑ: 03121008

Διάγραμμα ER



Σχεσιακό Διάγραμμα



Όπως φαίνεται κι από τα διαγράμματα παραπάνω, οι κύριες οντότητες της βάσης μας είναι οι συνταγές (recipes), οι μάγειρες (cooks) και τα επεισόδια (episode). Οι συνταγές σχετίζονται με αρκετές από τις υπόλοιπες όπως τα tags, τα cuisines και το equipment κ.α., προκειμένου να πραγματοποιηθούν οι απαιτούμενες κατηγοριοποιήσεις των συνταγών στις διάφορες ομάδες, καθώς και να δοθούν τα απαραίτητα στοιχεία για την υλοποίησή τους. Στη συνέχεια, οι μάγειρες οφείλουν να δηλώσουν τις εθνικές κουζίνες στις οποίες ειδικεύονται, ώστε σε κάθε επεισόδιο να εκτελούν συνταγές μόνο από τις κουζίνες αυτές. Σε κάποια επεισόδια επίσης συμμετέχουν και ως κριτές, ενώ η βαθμολογία για τους εκάστοτε διαγωνιζόμενους μάγειρες κάθε επεισόδιο της κάθε σεζόν καταγράφεται σε ξεχωριστό table.
Για τις εθνικές κουζίνες δημιουργήθηκε ξεχωριστό table, διότι συνδέεται με τις βασικές οντότητες της βάσης μας και είναι κρίσιμες για την εξέλιξη του διαγωνισμού.

INDICES

Τα διάφορα indices προσφέρουν γρηγορότερη πρόσβαση σε columns των tables και συνεπώς οδηγούν στην αποδοτικότερη αναζήτησή κατά την εκτέλεση των queries.
Καταρχάς, γνωρίζουμε ότι τα ευρετήρια δημιουργούνται αυτόματα για τα columns που είναι primary keys για όλα τα tables, που είναι απαραίτητο γιατί η χρήση των primary keys είναι πολύ συχνή τόσο στα where, order by, group by και join clauses, όσο και στα διάφορα triggers που κατασκευάζονται. Επιπλέον, ευρετήρια δημιουργήσαμε και για ορισμένα foreign keys που αποτελούν primary keys ή μέρος τους σε άλλες οντότητες, όπως για παράδειγμα το index για το season στο table score.Εκτός αυτών, τα υπόλοιπα indices επιλέχθηκαν με στόχο την απόδοση των queries. Στην περίπτωση διαφορετικών queries θα πρέπει να επέλθει αλλαγή στην επιλογή των indices. Όλα τα ευρετήρια βρίσκονται στο DDL script schema.sql.

ΠΡΟΔΙΑΓΡΑΦΕΣ ΚΑΙ ΥΠΟΘΕΣΕΙΣ

Το σύστημα υλοποιεί ένα σχεσιακό μοντέλο για αποθήκευση πληροφοριών για τον διαγωνισμό μαγειρικής και παράλληλα επιτρέπει την πρόσβαση σε δύο ειδών χρηστές: διαχειριστές και μάγειρες. Εξασφαλίζεται επίσης τόσο η συνέπεια των δεδομένων που αποθηκεύονται όσο και και οι περιορισμοί που καθορίζονται από τους κανονισμούς του διαγωνισμού, όπως για παράδειγμα ότι μία εθνική κουζίνα δεν επιτρέπεται να αντιπροσωπεύεται σε τρία συνεχόμενα επεισόδια του διαγωνισμού σε μία δεδομένη σεζόν. Θεωρούμε ότι το σύστημα παράγει διάφορες αναφορές σχετικά με την επίδοση των μαγείρων, περιλήψεις επεισοδίων και άλλα.

DDL και DML scripts

Παρατίθεται το DDL script όπως ακριβώς δίνεται και στο git repo:

```
drop table if exists winners;
drop table if exists categorized_by;
drop table if exists belongs;
drop table if exists nutritional_value;
drop table if exists specializes;
drop table if exists cuisines;
drop table if exists demands;
drop table if exists equipment;
drop table if exists consists_of;
drop table if exists steps;
drop table if exists needs;
drop table if exists classify;
drop table if exists food_groups;
drop table if exists nutritional_info;
drop table if exists tags;
drop table if exists grouped_by;
drop table if exists themes;
drop table if exists ingredients;
drop table if exists score;
drop table if exists episode;
drop table if exists cooks;
drop table if exists recipes;
drop table if exists roles;
drop table if exists users;
drop table if exists permissions;
drop table if exists role_permissions;
drop view if exists cooks_rating;
drop view if exists cuisine_rating;
drop view if exists judges;
drop view if exists judges_by_season;
drop view if exists tag_pairs;
drop view if exists appearances;
drop view if exists low_apps;
drop view if exists yearly_entries;
drop view if exists consecutive_years;
drop view if exists judging;
drop view if exists score_by_judge;
drop view if exists total_score_by_judge;
drop view if exists episode_difficulty;
drop view if exists rank_mapping;
drop view if exists episode_rank;
drop view if exists recipes_made;
drop view if exists themes_appearances;
drop view if exists ingr_used;
drop view if exists food_groups_used;

create table recipes(
recipes_id int unsigned not null auto_increment,
    recipes_type enum('Pastry', 'Bakery') not null,
    difficulty int unsigned not null check (difficulty>=1 and difficulty<=5),
    recipes_name varchar(100) not null unique,
    recipes_description text,
    meal_type set('Breakfast','Brunch','Lunch','Snack','Dinner','Dessert','Appetizer') not null,
    tip1 tinytext default null,
    tip2 tinytext default null,
    tip3 tinytext default null,
    prep_time int unsigned not null,
    cooking_time int unsigned not null,
    total_time int unsigned not null,
    primary_ingredient varchar(50) not null,
        category_by_ingr enum('Vegetarian', 'Seafood', 'Eggs', 'Cereals and Potatoes', 'Dairy
Products', 'Meat','Poultry','Nuts') not null,
    recipes_picture mediumblob default null,
recipes_caption tinytext default null,
            last_update_recipes timestamp not null default current_timestamp on update
current_timestamp,
    primary key (recipes_id)
);

create index idx_difficulty on recipes(difficulty);

create table cuisines(
cuisines_id int unsigned not null auto_increment,
    cuisines_name varchar(30),
    cuisines_picture mediumblob default null,
cuisines_caption tinytext default null,
            last_update_cuisines timestamp not null default current_timestamp on update
current_timestamp,
    primary key (cuisines_id)
);

create index idx_cuisines_name on cuisines (cuisines_name);

create table belongs(
recipes_id int unsigned not null,
    cuisines_id int unsigned not null,
            last_update_belongs timestamp not null default current_timestamp on update
current_timestamp,
    primary key (recipes_id,cuisines_id),
      constraint fk_recipes_id_belongs foreign key (recipes_id) references recipes (recipes_id) on
delete restrict on update cascade,
     constraint fk_cuisines_id_belongs foreign key (cuisines_id) references cuisines (cuisines_id) on
delete restrict on update cascade
);

create table tags(
tags_id int unsigned not null auto_increment,
    tags_name varchar(200) not null unique,
    tags_picture mediumblob default null,
tags_caption tinytext default null,
    last_update_tags timestamp not null default current_timestamp on update current_timestamp,
    primary key (tags_id)
);

create table categorized_by(
recipes_id int unsigned not null,
    tags_id int unsigned not null,
         last_update_categorized_by timestamp not null default current_timestamp on update
current_timestamp,
primary key (recipes_id,tags_id),
      constraint fk_recipes_id_ca foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade,
    constraint fk_tags_id foreign key (tags_id) references tags (tags_id) on delete restrict on update
cascade
);

create table equipment(
equipment_id int unsigned not null auto_increment,
   equipment_name tinytext not null,
   equipment_instructions text,
   equipment_picture mediumblob default null,
equipment_caption tinytext default null,
            last_update_equipment timestamp not null default current_timestamp on update
current_timestamp,
   primary key (equipment_id)
);

create table demands(
recipes_id int unsigned not null,
   equipment_id int unsigned not null,
            last_update_demands timestamp not null default current_timestamp on update
current_timestamp,
   primary key(recipes_id,equipment_id),
     constraint fk_recipes_id_de foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade,
    constraint fk_equiment_id foreign key (equipment_id) references equipment (equipment_id) on
delete restrict on update cascade
);

create table steps(
steps_id int unsigned not null auto_increment,
   steps_description text,
   steps_picture mediumblob default null,
steps_caption tinytext default null,
   last_update_steps timestamp not null default current_timestamp on update current_timestamp,
   primary key (steps_id)
);

create table consists_of(
recipes_id int unsigned not null,
   steps_id int unsigned not null,
   portion int unsigned,
            last_update_consists_of timestamp not null default current_timestamp on update
current_timestamp,
primary key (recipes_id,steps_id),
     constraint fk_recipes_id_co foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade,
     constraint fk_steps_id foreign key (steps_id) references steps (steps_id) on delete restrict on
update cascade
);

create index idx_portion on consists_of (portion);

create table ingredients(
ingredients_id int unsigned not null auto_increment,
   ingredients_quantity int unsigned,
   ingredients_quantity_aux varchar(50),
   ingredients_name varchar(50) not null,
   caloriesp100 int unsigned not null,
   ingredients_picture mediumblob default null,
ingredients_caption tinytext default null,
            last_update_ingredients timestamp not null default current_timestamp on update
current_timestamp,
   primary key (ingredients_id)
);

create table needs(
recipes_id int unsigned not null,
   ingredients_id int unsigned not null,
            last_update_needs timestamp not null default current_timestamp on update
current_timestamp,
   primary key (recipes_id,ingredients_id),
     constraint fk_recipes_id_ne foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade,
            constraint fk_ingredients_id_ne foreign key (ingredients_id) references ingredients
(ingredients_id) on delete restrict on update cascade
);

create table food_groups(
food_groups_id int unsigned not null auto_increment,
            food_groups_name enum('Vegetables','Fruits','Cereals and Potatoes','Dairy
Products','Legumes','Red Meat','White Meat','Eggs','Fish and Seafood','Added Oils, Fats and
Nuts'),
   food_groups_description text,
   food_groups_picture mediumblob default null,
food_groups_caption tinytext default null,
           last_update_food_groups timestamp not null default current_timestamp on update
current_timestamp,
   primary key (food_groups_id)
);

create table classify(
ingredients_id int unsigned not null,
   food_groups_id int unsigned not null,
            last_update_classify timestamp not null default current_timestamp on update
current_timestamp,
   primary key (ingredients_id,food_groups_id),
            constraint fk_ingredients_id_cl foreign key (ingredients_id) references ingredients
(ingredients_id) on delete restrict on update cascade,
           constraint fk_food_groups_id foreign key (food_groups_id) references food_groups
(food_groups_id) on delete restrict on update cascade
);

create table nutritional_info(
recipes_id int unsigned not null,
   fatpp int unsigned not null,
   proteinpp int unsigned not null,
   carbonhydratepp int unsigned not null,
   caloriespp int unsigned not null,
   nutritional_info_picture mediumblob default null,
nutritional_info_caption tinytext default null,
          last_update_nutritional_info timestamp not null default current_timestamp on update
current_timestamp,
   primary key (recipes_id),
     constraint fk_recipes_id_ni foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade
);

create index idx_carbonhydartes on nutritional_info (carbonhydratepp);
create index idx_calories on nutritional_info (caloriespp);
create table nutritional_value(
recipes_id int unsigned not null,
    calories int unsigned,
    primary key (recipes_id),
        last_update_nutritional_value timestamp not null default current_timestamp on update current_timestamp,
      constraint fk_recipes_id_nut foreign key (recipes_id) references recipes (recipes_id) on delete restrict on update cascade
);

create table themes(
themes_id int unsigned not null auto_increment,
    themes_name tinytext,
    themes_description text,
    themes_picture mediumblob default null,
themes_caption tinytext default null,
            last_update_themes timestamp not null default current_timestamp on update current_timestamp,
    primary key (themes_id)
);



create table grouped_by(
recipes_id int unsigned not null,
    themes_id int unsigned not null,
            last_update_grouped_by timestamp not null default current_timestamp on update current_timestamp,
primary key (recipes_id,themes_id),
       constraint fk_recipes_id_gr foreign key (recipes_id) references recipes (recipes_id) on delete restrict on update cascade,
constraint fk_themes_id foreign key (themes_id) references themes (themes_id) on delete restrict on update cascade
);

create table cooks(
cooks_id int unsigned not null auto_increment,
    cooks_name varchar(50) not null,
    cooks_surname varchar(50) not null,
    cooks_phone_number bigint unsigned not null unique,
    cooks_date_of_birth date not null,
    cooks_age int unsigned not null,
    cooks_experience int unsigned not null,
    cooks_rank tinytext,
    cooks_picture mediumblob default null,
cooks_caption tinytext default null,
            last_update_cooks timestamp not null default current_timestamp on update current_timestamp,
    primary key (cooks_id)
);

create index idx_cooks_name on cooks (cooks_name);
create index idx_cooks_surname on cooks (cooks_surname);
create index idx_cooks_age on cooks (cooks_age);


create table specializes (
cooks_id int unsigned not null,
    cuisines_id int unsigned not null,
          last_update_specializes timestamp not null default current_timestamp on update
current_timestamp,
   primary key (cooks_id,cuisines_id),
   constraint fk_cooks_id_specializes foreign key (cooks_id) references cooks (cooks_id) on delete
restrict on update cascade,
constraint fk_cuisines_id_specializes foreign key (cuisines_id) references cuisines (cuisines_id) on
delete restrict on update cascade
);

create table episode(
episode_id int unsigned not null auto_increment,
      season int unsigned not null,
                    episode_in_season int unsigned not null check (episode_in_season>=1 and
episode_in_season<=10),
      episode_picture mediumblob default null,
episode_caption tinytext default null,
      cooks_id int unsigned not null,
      cuisines_id int unsigned not null,
recipes_id int unsigned not null,
      judge1 int unsigned not null,
      judge2 int unsigned not null,
      judge3 int unsigned not null,
                  last_update_episode timestamp not null default current_timestamp on update
current_timestamp,
primary key (episode_id),
      constraint fk_cooks_id_episode foreign key (cooks_id) references cooks (cooks_id) on delete
restrict on update cascade,
constraint fk_recipes_id_episode foreign key (recipes_id) references recipes (recipes_id) on delete
restrict on update cascade,
constraint fk_judge1 foreign key (judge1) references cooks (cooks_id) on delete restrict on update
cascade,
constraint fk_judge2 foreign key (judge2) references cooks (cooks_id) on delete restrict on update
cascade,
constraint fk_judge3 foreign key (judge3) references cooks (cooks_id) on delete restrict on update
cascade,
        constraint fk_cuisines_id_episode foreign key (cuisines_id) references cuisines (cuisines_id)
on delete restrict on update cascade
);

create index idx_season on episode (season);
create index idx_episode_in_season on episode (episode_in_season);
create index idx_cooks_id on episode (cooks_id);
create index idx_cuisines_id on episode (cuisines_id);
create index idx_recipes_id on episode (recipes_id);
create index idx_judge1 on episode (judge1);
create index idx_judge2 on episode (judge2);
create index idx_judge3 on episode (judge3);

create table score(
cooks_id int unsigned not null,
   season int unsigned not null,
episode_in_season int unsigned not null,
   first_score int unsigned not null check (first_score>=1 and first_score<=5),
   second_score int unsigned not null check (second_score>=1 and second_score<=5),
   third_score int unsigned not null check (third_score>=1 and third_score<=5),
   total_score int unsigned not null check (total_score>=3 and total_score<=15),
   score_picture mediumblob default null,
score_caption tinytext default null,
   last_update_score timestamp not null default current_timestamp on update current_timestamp,
   primary key (cooks_id,season,episode_in_season),
       constraint fk_cooks_id_score foreign key (cooks_id) references cooks (cooks_id) on delete
restrict on update cascade,
      constraint fk_season foreign key (season) references episode (season) on delete restrict on
update cascade,
constraint fk_episode_in_season foreign key (episode_in_season) references episode
(episode_in_season) on delete restrict on update cascade
);

create index idx_cooks_id on score (cooks_id);
create index idx_season on score (season);
create index idx_episode_in_season on score (episode_in_season);
create index idx_first_score on score (first_score);
create index idx_second_score on score (second_score);
create index idx_third_score on score (third_score);
create index idx_total_score on score (total_score);

create table winners(
    season int unsigned not null,
episode_in_season int unsigned not null,
    cooks_id int unsigned not null,
    cooks_name varchar(50) not null,
    cooks_surname varchar(50) not null,
            last_update_winner timestamp not null default current_timestamp on update
current_timestamp,
constraint fk_cooks_id_winner foreign key (cooks_id) references cooks (cooks_id) on delete
restrict on update cascade,
    constraint fk_season_winner foreign key (season) references episode (season) on delete restrict
on update cascade,
constraint fk_episode_in_season_winner foreign key (episode_in_season) references episode
(episode_in_season) on delete restrict on update cascade,
constraint fk_cooks_name_winner foreign key (cooks_name) references cooks (cooks_name) on
delete restrict on update cascade,
constraint fk_cooks_surname_winner foreign key (cooks_surname) references cooks
(cooks_surname) on delete restrict on update cascade
);


-- Now for the app
create table roles (
    role_id int unsigned not null auto_increment,
    role_name varchar(255) not null unique,
primary key (role_id),
last_update_roles timestamp not null default current_timestamp on update current_timestamp
);

create table users (
    user_id int unsigned not null auto_increment,
    username varchar(255) not null unique,
    pass varchar(255) not null,
    role_id int unsigned not null,
    primary key (user_id),
    last_update_users timestamp not null default current_timestamp on update current_timestamp,
constraint fk_roles_users foreign key (role_id) references roles (role_id) on delete restrict on update
cascade
);

create table permissions (
    permission_id int unsigned not null auto_increment,
    permission_name varchar(255) not null unique,
    primary key (permission_id),
last_update_permissions timestamp not null default current_timestamp on update
current_timestamp
);

create table role_permissions (
   role_id int unsigned not null,
   permission_id int unsigned not null,
   primary key (role_id, permission_id),
last_update_role_permissions timestamp not null default current_timestamp on update
current_timestamp,
     constraint fk_roles_role_permissions foreign key (role_id) references roles (role_id) on delete
restrict on update cascade,
constraint fk_permissions foreign key (permission_id) references permissions (permission_id) on
delete restrict on update cascade
);

create view cooks_rating as select cooks_id, AVG(total_score) as average_score from score group
by cooks_id;
create view cuisines_rating as select cuisines_id,AVG(total_score) as average_score from score sc
join specializes sp on (sp.cooks_id=sc.cooks_id) group by cuisines_id;
create view judges as
select distinct
cooks.cooks_id,cooks.cooks_name,cooks.cooks_surname
from
cooks
join
episode
where
((cooks.cooks_id=episode.judge1) OR (cooks.cooks_id=episode.judge2) OR
(cooks.cooks_id=episode.judge3))
order by
cooks.cooks_id;
create view judges_by_season as select j.cooks_id,ep.season,count(j.cooks_id) as times from
judges j join episode ep on  (ep.cooks_id=j.cooks_id)  group by j.cooks_id,ep.season order by
j.cooks_id,ep.season;
create view tag_pairs as select c1.tags_id as tag1,c2.tags_id as tag2,count(*) as pair_count
from categorized_by c1
join categorized_by c2 on c1.recipes_id=c2.recipes_id and c1.tags_id<c2.tags_id
join episode e on c1.recipes_id=e.episode_id
group by c1.tags_id, c2.tags_id;

create view appearances as select cooks_id, count(*) as apps from episode group by cooks_id
order by apps desc;
create view low_apps as select cooks_id from appearances where apps < ((select max(apps) from
appearances) - 5) group by cooks_id order by cooks_id;

create view yearly_entries as select e.cuisines_id,e.season as season, count(*) as entries_count
from episode e group by  e.cuisines_id, e.season having count(*)>=3;
create view consecutive_years as select   y1.cuisines_id,y1.season as year1,y2.season as
year2,y1.entries_count as entries_count1,y2.entries_count as entries_count2 from yearly_entries
y1 join yearly_entries y2 on  y1.cuisines_id = y2.cuisines_id and y2.season = y1.season + 1;

create view judging as
select j.cooks_id as judges_id, e.cooks_id as cooks_id, e.episode_id, e.season,
episode_in_season as eis, if(j.cooks_id=judge1,1,0) as fsc,if(j.cooks_id=judge2,1,0) as
ssc,if(j.cooks_id=judge3,1,0) as tsc
from judges j
join episode e on(j.cooks_id=judge1 or j.cooks_id=judge2 or j.cooks_id=judge3);

create view score_by_judge as
```

```
select j.judges_id,j.cooks_id,j.season,j.eis,if(fsc=1,s.first_score,0) as fs,if(ssc=1,s.second_score,0)
as ss,if(tsc=1,s.third_score,0) as ts
from judging j
join score s on(j.season=s.season and j.eis=s.episode_in_season and j.cooks_id=s.cooks_id);

create view total_score_by_judge as
select judges_id,cooks_id,sum(fs+ss+ts) as total_score
from score_by_judge
group by judges_id,cooks_id
order by total_score desc limit 5;

create view episode_difficulty as
select  e.season,e.episode_in_season as eis,sum(r.difficulty) as difficulty
   from episode e
   join recipes r on(e.recipes_id=r.recipes_id)
   group by  e.season,e.episode_in_season
   order by e.season,e.episode_in_season;

create view rank_mapping as select 'C' as cook_rank, 1 as rank_value union all select 'B' as
cook_rank, 2 as rank_value union all select 'A' as cook_rank, 3 as rank_value union all select
'Sous Chef' as cook_rank, 4 as rank_value union all select 'Chef' as cook_rank, 5 as rank_value;

create view episode_rank as
select e.episode_id,
e.season,
e.episode_in_season,
e.cooks_id,
rm_cook.rank_value as cook_rank,
rm_judge1.rank_value as judge1_rank,
rm_judge2.rank_value as judge2_rank,
rm_judge3.rank_value as judge3_rank,
rm_cook.rank_value + rm_judge1.rank_value + rm_judge2.rank_value + rm_judge3.rank_value as
total_rank
from episode e join cooks c on e.cooks_id = c.cooks_id
join rank_mapping rm_cook on c.cooks_rank = rm_cook.cook_rank
join  cooks j1 on e.judge1 = j1.cooks_id
join  rank_mapping rm_judge1 on j1.cooks_rank = rm_judge1.cook_rank
join  cooks j2 on e.judge2 = j2.cooks_id
join rank_mapping rm_judge2 on j2.cooks_rank = rm_judge2.cook_rank
join cooks j3 on e.judge3 = j3.cooks_id
join rank_mapping rm_judge3 on j3.cooks_rank = rm_judge3.cook_rank
order by total_rank;

create view recipes_made as select distinct r.recipes_id,count(*) as appearance_counter
from episode e
join recipes r on r.recipes_id=e.recipes_id
group by r.recipes_id
order by r.recipes_id;

create view themes_appearances as select gb.themes_id, sum(rm.appearance_counter) as
total_appearances
   from grouped_by gb
   join recipes_made rm on gb.recipes_id = rm.recipes_id
   group by gb.themes_id
   order by gb.themes_id;

create view ingr_used as select distinct n.ingredients_id from needs n join (select distinct
recipes_id from episode) as allrec on (n.recipes_id = allrec.recipes_id) order by n.ingredients_id ;
create view food_groups_used as select distinct food_groups_id from classify cl join ingr_used iu
on (cl.ingredients_id=iu.ingredients_id) order by food_groups_id;
```

```
-- ---------------------------------------------------------------------------------
-- Triggers

DELIMITER //
create trigger enforce_cook_specialization
before insert on episode
for each row
begin
   declare specialization_count int;
   select count(*)
   into specialization_count
   from specializes
   where cooks_id = new.cooks_id
     and cuisines_id = new.cuisines_id;

   if specialization_count = 0 then
      signal sqlstate '45000'
      set message_text = 'Cook is not specialized in the specified cuisine';
   end if;
end;
//
DELIMITER ;
--

DELIMITER $$
create trigger check_total_time
before insert on recipes
for each row
begin
   if new.total_time != new.prep_time + new.cooking_time then
         signal sqlstate '45000' set message_text = 'Total time must be the sum of preparation time
and cooking time.';
   end if;
end $$
DELIMITER ;

--
DELIMITER $$
create trigger check_age_experience
before insert on cooks
for each row
begin
   if new.cooks_age <= new.cooks_experience then
            signal sqlstate '45000' set message_text = 'Age must be greater than years of cooking
experience.';
   end if;
end $$
DELIMITER ;
--

DELIMITER $$
create trigger check_recipe_consecutive_episodes
before insert on episode
for each row
begin
   declare ep1 int;
   declare ep2 int;

   set ep1=0;
```

```
    set ep2=0;

    if new.episode_in_season>2 then
select count(*) into ep1
from episode
where season = new.season
 and recipes_id = new.recipes_id
 and episode_in_season = new.episode_in_season - 1;

select count(*) into ep2
from episode
where season = new.season
 and recipes_id = new.recipes_id
 and episode_in_season = new.episode_in_season - 2;
end if;

    if ep1 > 0 and ep2 > 0 then
            signal sqlstate '45000' set message_text = 'A recipe cannot be in three consecutive
episodes of a season';
    end if;
end $$
DELIMITER ;

DELIMITER $$
create trigger check_cuisine_consecutive_episodes
before insert on episode
for each row
begin
    declare ep1 int;
    declare ep2 int;

    set ep1=0;
    set ep2=0;

    if new.episode_in_season>2 then
select count(*) into ep1
from episode
where season = new.season
 and cuisines_id = new.cuisines_id
 and episode_in_season = new.episode_in_season - 1;

select count(*) into ep2
from episode
where season = new.season
 and cuisines_id = new.cuisines_id
 and episode_in_season = new.episode_in_season - 2;
end if;

    if ep1 > 0 and ep2 > 0 then
            signal sqlstate '45000' set message_text = 'A cuisine cannot be in three consecutive
episodes of a season';
    end if;
end $$
DELIMITER ;

DELIMITER $$
create trigger check_cook_consecutive_episodes
before insert on episode
for each row
begin
```

```
    declare ep1 int;
    declare ep2 int;

    set ep1=0;
    set ep2=0;

if new.episode_in_season>2 then
select count(*) into ep1
from episode
where season = new.season
 and cooks_id = new.cooks_id
 and episode_in_season = new.episode_in_season - 1;

select count(*) into ep2
from episode
where season = new.season
 and cooks_id = new.cooks_id
 and episode_in_season = new.episode_in_season - 2;
end if;

    if ep1 > 0 and ep2 > 0 then
            signal sqlstate '45000' set message_text = 'A cook cannot appear in three consecutive
episodes of a season';
    end if;
end $$
DELIMITER ;

DELIMITER $$
create trigger check_judge_consecutive_episodes
before insert on episode
for each row
begin
    declare ep1 int;
    declare ep2 int;

    set ep1=0;
    set ep2=0;

if new.episode_in_season>2 then
select count(*) into ep1
from episode
where season = new.season
 and (judge1 = new.judge1 or judge2 = new.judge1 or judge3 = new.judge1)
 and episode_in_season = new.episode_in_season - 1;

select count(*) into ep2
from episode
where season = new.season
 and (judge1 = new.judge1 or judge2 = new.judge1 or judge3 = new.judge1)
 and episode_in_season = new.episode_in_season - 2;
end if;

    if ep1 > 0 and ep2 > 0 then
            signal sqlstate '45000' set message_text = 'A judge cannot appear in three consecutive
episodes of a season';
    end if;
end $$
DELIMITER ;
```

Συγκεκριμένα, διαγράφονται από τη βάση όλοι οι υπάρχοντες πίνακες και στη συνέχεια δημιουργούνται οι πίνακες και τα views που έχουμε ορίσει. Επίσης, στο script αυτό περιέχονται όλα τα constraints και τα triggers της βάσης.

Το DML script περιλαμβάνουν τα insertions με τα οποία γεμίζουμε τα tables της βάσης μας με δεδομένα και βρίσκονται στο path /sql/insertions.sql. Να σημειωθεί τέλος ότι η κλήρωση για τα επεισόδια γίνεται μέσω του συνημμένου αρχείου στο αποθετήριο σε γλώσσα python.

ΟΔΗΓΙΕΣ ΕΓΚΑΤΑΣΤΑΣΗΣ ΚΑΙ GIT REPO

Το repository της βάσης στο GitHub:
https://github.com/evaggeliafil/databases_project.git

Η εργασία αυτή δεν απαιτούσε τη δημιουργία ολοκληρωμένης εφαρμογής.
Για την εγκατάσταση της βάσης στον υπολογιστή μας χρησιμοποιήσαμε mysql μέσω xampp και ένα DBMS το Myql Workbench. Συνεπώς, το μόνο που απαιτήθηκε ήταν η σύνδεση με mysql server και η δημιουργία των scripts.