Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Εξάμηνο 6ο: Βάσεις Δεδομένων

Διδάσκοντες: Δ. Τσουμάκος, Μ. Κόνιαρης

Εξαμηνιαία Εργασία στις Βάσεις Δεδομένων

Ημερομηνία Παράδοσης: 26/05/2024

ΟΜΑΔΑ 79

Κατερίνα Μίχου, el21079

Αγγελική Νταλαπέρα, el21001

Γιάννης Πολυχρονόπουλος, el21089

Link: https://github.com/JohnnyPol/Database_Project_ECE_NTUA

ER-Diagram & Relational Diagram

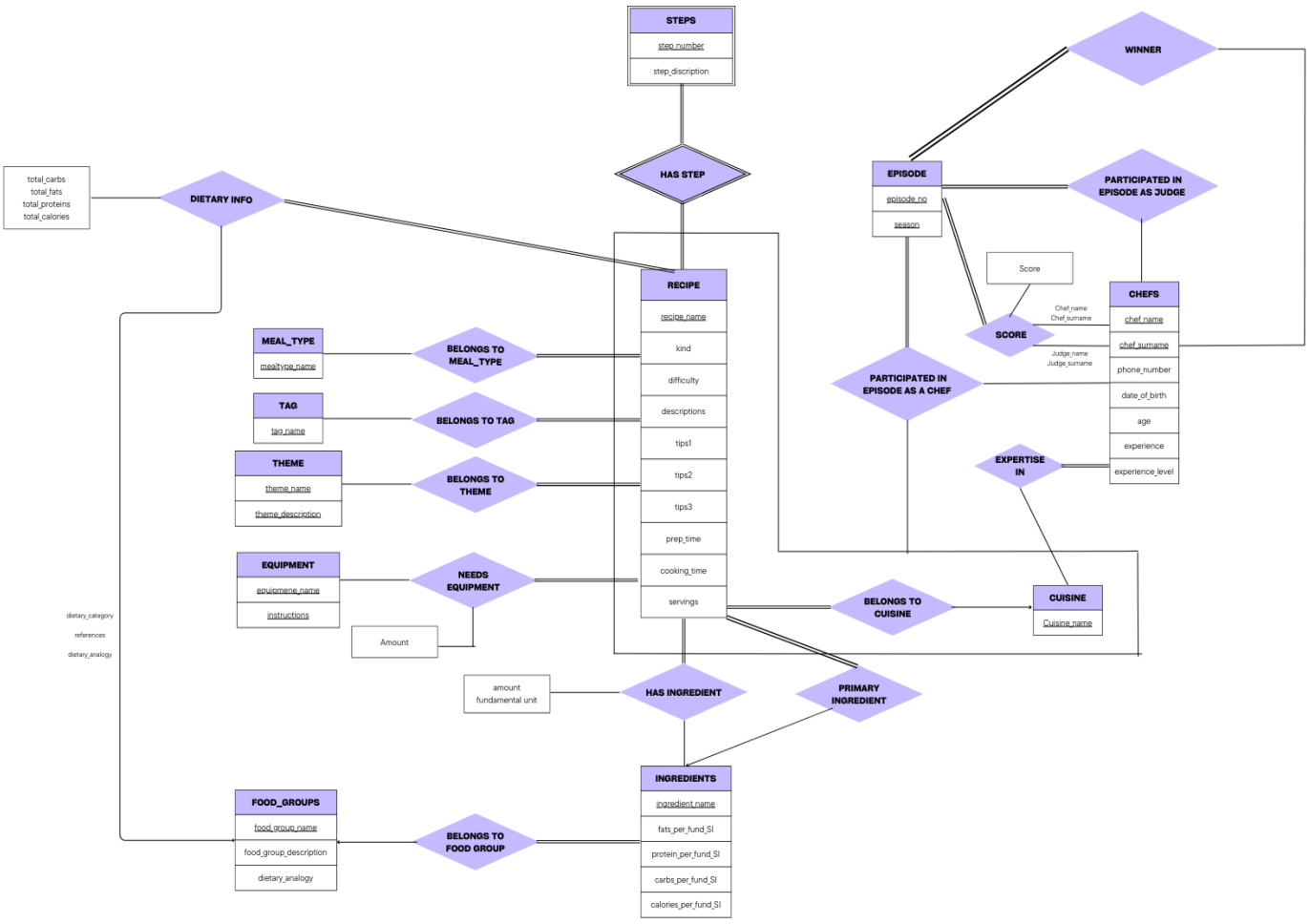
ER-Diagram:

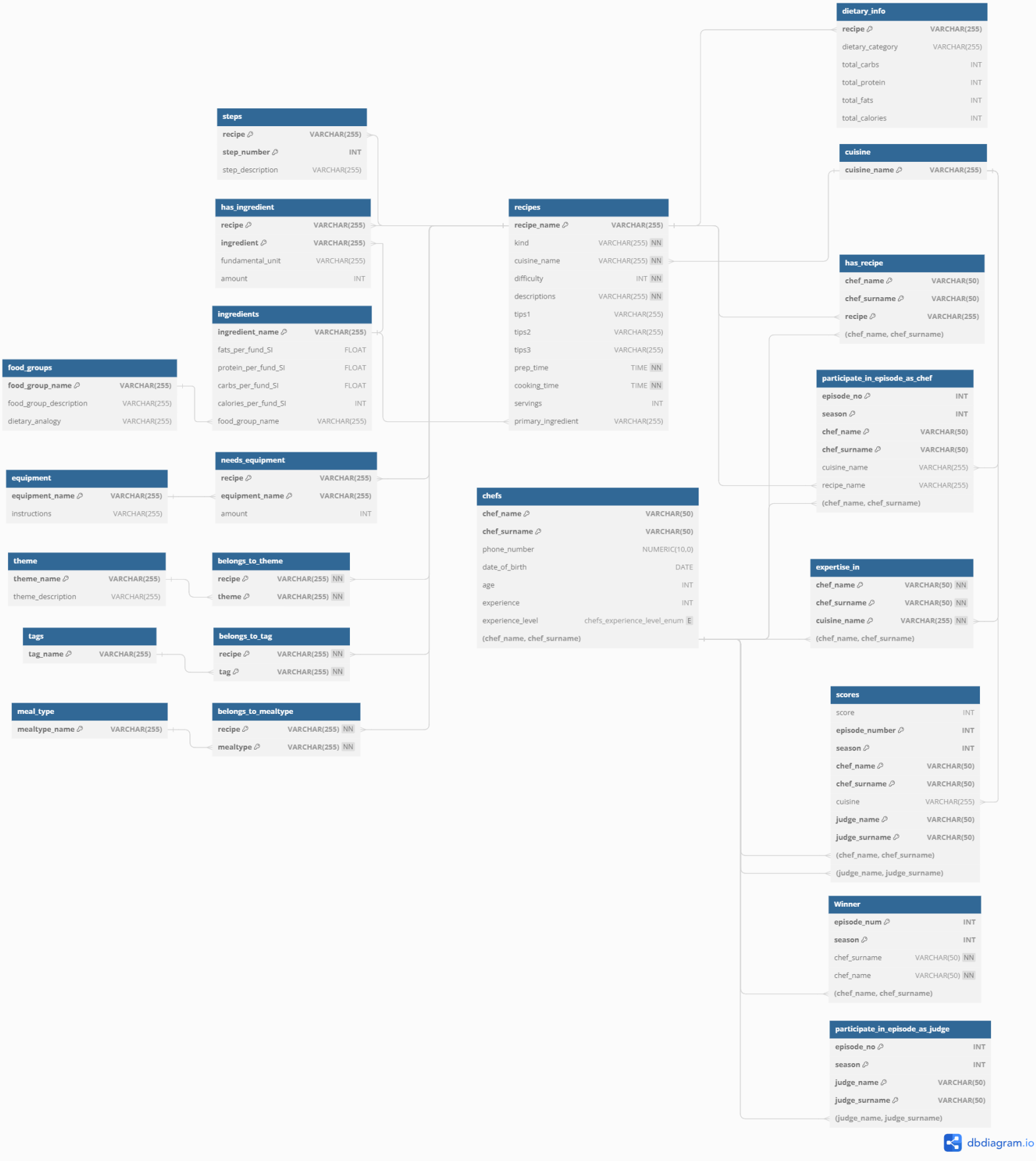
Αρχικά, σχεδιάσαμε το διάγραμμα οντοτήτων-συσχετίσεων (Entity-Relationship Diagram), όπως φαίνεται και στο παρακάτω σχήμα.

Σημείωση: Στο ER εμφανίζεται και η οντότητα επισόδειο προκειμένου να μπορούν να αποδοθούν ιδιότητες στους μάγειρες ως μάγειρες ή ως κριτές. Όμως δεν απαιτείται τελικά ως οντότητα για την λειτουργικότητα της βάσης και για αυτό δεν εμφανίζεται στο Relational Diagram ή τη βάση.

Relational Diagram:

Στη συνέχεια, σχεδιάσαμε το σχεσιακό σχήμα της βάσης μας, το οποίο χρησιμοποιήσαμε και ως έναυσμα για τον σχεδιασμό και την κατασκευή της μέσω του DDL Script (δίδεται παρακάτω). Το διάγραμμα αυτό παραδίδεται κάτω από το ER-Diagram.




dietary_info
recipe VARCHAR(255)
dietary_category VARCHAR(255)
total_carbs INT
total_protein INT
total_fats INT
total_calories INT
steps
recipe VARCHAR(255)
step_number INT
step_description VARCHAR(255)
cuisine
cuisine_name VARCHAR(255)
has_ingredient
recipe VARCHAR(255)
ingredient VARCHAR(255)
fundamental_unit VARCHAR(255)
amount INT
recipes
recipe_name VARCHAR(255)
kind VARCHAR(255) NN
cuisine_name VARCHAR(255) NN
difficulty INT NN
descriptions VARCHAR(255) NN
tips1 VARCHAR(255)
tips2 VARCHAR(255)
tips3 VARCHAR(255)
prep_time TIME NN
cooking_time TIME NN
servings INT
primary_ingredient VARCHAR(255)
has_recipe
chef_name VARCHAR(50)
chef_surname VARCHAR(50)
recipe VARCHAR(255)
(chef_name, chef_surname)
ingredients
ingredient_name VARCHAR(255)
fats_per_fund_SI FLOAT
protein_per_fund_SI FLOAT
carbs_per_fund_SI FLOAT
calories_per_fund_SI INT
food_group_name VARCHAR(255)
food_groups
food_group_name VARCHAR(255)
food_group_description VARCHAR(255)
dietary_analogy VARCHAR(255)
participate_in_episode_as_chef
episode_no INT
season INT
chef_name VARCHAR(50)
chef_surname VARCHAR(50)
cuisine_name VARCHAR(255)
recipe_name VARCHAR(255)
(chef_name, chef_surname)
needs_equipment
recipe VARCHAR(255)
equipment_name VARCHAR(255)
amount INT
equipment
equipment_name VARCHAR(255)
instructions VARCHAR(255)
chefs
chef_name VARCHAR(50)
chef_surname VARCHAR(50)
phone_number NUMERIC(10,0)
date_of_birth DATE
age INT
experience INT
experience_level chefs_experience_level_enum E
(chef_name, chef_surname)
theme
theme_name VARCHAR(255)
theme_description VARCHAR(255)
belongs_to_theme
recipe VARCHAR(255) NN
theme VARCHAR(255) NN
expertise_in
chef_name VARCHAR(50) NN
chef_surname VARCHAR(50) NN
cuisine_name VARCHAR(255) NN
(chef_name, chef_surname)
tags
tag_name VARCHAR(255)
belongs_to_tag
recipe VARCHAR(255) NN
tag VARCHAR(255) NN
scores
score INT
episode_number INT
season INT
chef_name VARCHAR(50)
chef_surname VARCHAR(50)
cuisine VARCHAR(255)
judge_name VARCHAR(50)
judge_surname VARCHAR(50)
(chef_name, chef_surname)
(judge_name, judge_surname)
meal_type
mealtype_name VARCHAR(255)
belongs_to_mealtype
recipe VARCHAR(255) NN
mealtype VARCHAR(255) NN
Winner
episode_num INT
season INT
chef_surname VARCHAR(50) NN
chef_name VARCHAR(50) NN
(chef_name, chef_surname)
participate_in_episode_as_judge
episode_no INT
season INT
judge_name VARCHAR(50)
judge_surname VARCHAR(50)
(judge_name, judge_surname)
dbdiagram.io

DDL & DML Scripts

- *DDL*

```
CREATE DATABASE IF NOT EXISTS Masterchef_NTUA_Edition;
USE Masterchef_NTUA_Edition;
DROP TABLE IF EXISTS has_recipe;
DROP TABLE IF EXISTS participate_in_episode_as_chef;
DROP TABLE IF EXISTS participate_in_episode_as_judge;
DROP TABLE IF EXISTS Winner;
DROP TABLE IF EXISTS belongs_to_mealtype;
DROP TABLE IF EXISTS belongs_to_tag;
DROP TABLE IF EXISTS needs_equipment;
DROP TABLE IF EXISTS has_ingredient;
DROP TABLE IF EXISTS dietary_info;
DROP TABLE IF EXISTS expertise_in;
DROP TABLE IF EXISTS steps;
DROP TABLE IF EXISTS meal_type;
DROP TABLE IF EXISTS tags;
DROP TABLE IF EXISTS equipment;
DROP TABLE IF EXISTS scores;
DROP TABLE IF EXISTS belongs_to_theme;
DROP TABLE IF EXISTS theme;
DROP TABLE IF EXISTS chefs;
DROP TABLE IF EXISTS recipes;
DROP TABLE IF EXISTS ingredients;
DROP TABLE IF EXISTS food_groups;
DROP TABLE IF EXISTS cuisine;
-- ----------------------------------------
-- Tables --
-- ----------------------------------------
CREATE TABLE cuisine ( cuisine_name VARCHAR(255) PRIMARY KEY,
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE food_groups (
    food_group_name VARCHAR(255) PRIMARY KEY,
    food_group_description VARCHAR(255),
    dietary_analogy VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE ingredients (
    ingredient_name VARCHAR(255) PRIMARY KEY,
    fats_per_fund_SI FLOAT,
    protein_per_fund_SI FLOAT,
    carbs_per_fund_SI FLOAT,
    calories_per_fund_SI INT,
    food_group_name VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description",
    FOREIGN KEY (food_group_name) REFERENCES food_groups(food_group_name)
);
CREATE TABLE recipes (
    recipe_name VARCHAR(255) PRIMARY KEY,
    kind VARCHAR(255) CHECK (kind in ("cooking", "baking")) NOT NULL,
    cuisine_name VARCHAR(255) NOT NULL,
    difficulty INT CHECK (difficulty BETWEEN 1 AND 5) NOT NULL,
    descriptions VARCHAR(255) NOT NULL,
    tips1 VARCHAR(255),
    tips2 VARCHAR(255),
    tips3 VARCHAR(255),
    prep_time TIME NOT NULL,
    cooking_time TIME NOT NULL,
    servings INT,
    primary_ingredient VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
```

```
    image_description VARCHAR(255) DEFAULT "Description",
    FOREIGN KEY (cuisine_name) REFERENCES cuisine (cuisine_name),
    FOREIGN KEY (primary_ingredient) REFERENCES ingredients (ingredient_name)
);
CREATE TABLE steps (
    recipe VARCHAR(255),
    step_number INT,
    step_description VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description",
    PRIMARY KEY (recipe, step_number),
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name)
);
CREATE TABLE theme (
    theme_name VARCHAR(255) PRIMARY KEY,
    theme_description VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE meal_type (
    mealtype_name VARCHAR(255) PRIMARY KEY,
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE tags (
    tag_name VARCHAR(255) PRIMARY KEY ,
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE equipment (
    equipment_name VARCHAR(255) PRIMARY KEY,
    instructions VARCHAR(255),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description"
);
CREATE TABLE chefs (
    chef_name VARCHAR(50),
    chef_surname VARCHAR(50),
    phone_number NUMERIC(10, 0),
    date_of_birth DATE,
    age INT,
    experience INT,
    experience_level ENUM(
        "3rd cook",
        "2nd cook",
        "1st cook",
        "sous chef",
        "chef"
    ),
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description",
    PRIMARY KEY (chef_name, chef_surname)
);
CREATE TABLE participate_in_episode_as_chef (
    episode_no INT,
    season INT,
    chef_name VARCHAR(50),
    chef_surname VARCHAR(50),
    cuisine_name VARCHAR(255),
    recipe_name VARCHAR(255),
    FOREIGN KEY (chef_name, chef_surname) REFERENCES chefs (chef_name, chef_surname),
    FOREIGN KEY (recipe_name) REFERENCES recipes (recipe_name),
    FOREIGN KEY (cuisine_name) REFERENCES cuisine (cuisine_name),
    PRIMARY KEY (
        episode_no,
        season,
        chef_name,
        chef_surname)
);
```

```sql
CREATE TABLE participate_in_episode_as_judge (
    episode_no INT,
    season INT,
    judge_name VARCHAR(50),
    judge_surname VARCHAR(50),
    FOREIGN KEY (judge_name, judge_surname) REFERENCES chefs (chef_name, chef_surname),
    PRIMARY KEY (
        episode_no,
        season,
        judge_name,
        judge_surname
    )
);
CREATE TABLE scores (
    score INT,
    episode_number INT,
    season INT,
    chef_name VARCHAR(50),
    chef_surname VARCHAR(50),
    cuisine VARCHAR(255),
    judge_name VARCHAR(50),
    judge_surname VARCHAR(50),
    PRIMARY KEY (
        episode_number,
        season,
        chef_name,
        chef_surname,
        judge_name,
        judge_surname
    ),
    FOREIGN KEY (chef_name, chef_surname) REFERENCES chefs (chef_name, chef_surname),
    FOREIGN KEY (judge_name, judge_surname) REFERENCES chefs (chef_name, chef_surname),
    FOREIGN KEY (cuisine) references cuisine(cuisine_name),
    CHECK (score BETWEEN 1 AND 5)
);
CREATE TABLE has_recipe(
    chef_name VARCHAR(50),
    chef_surname VARCHAR(50),
    recipe VARCHAR(255),
    FOREIGN KEY (chef_name, chef_surname) REFERENCES chefs (chef_name, chef_surname),
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    PRIMARY KEY (chef_name, chef_surname, recipe)
);
CREATE TABLE belongs_to_mealtype (
    recipe VARCHAR(255) NOT NULL,
    mealtype VARCHAR(255) NOT NULL,
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    FOREIGN KEY (mealtype) REFERENCES meal_type (mealtype_name),
    PRIMARY KEY (recipe, mealtype)
);
CREATE TABLE belongs_to_tag (
    recipe VARCHAR(255) NOT NULL,
    tag VARCHAR(255) NOT NULL,
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    FOREIGN KEY (tag) REFERENCES tags (tag_name),
    PRIMARY KEY (recipe, tag)
);
CREATE TABLE belongs_to_theme (
    recipe VARCHAR(255) NOT NULL,
    theme VARCHAR(255) NOT NULL,
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    FOREIGN KEY (theme) REFERENCES theme (theme_name),
    PRIMARY KEY (recipe, theme)
);
CREATE TABLE needs_equipment (
    recipe VARCHAR(255),
    equipment_name VARCHAR(255),
    amount INT DEFAULT 0,
    PRIMARY KEY (recipe, equipment_name),
```

```
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    FOREIGN KEY (equipment_name) REFERENCES equipment (equipment_name)
);
CREATE TABLE has_ingredient (
    recipe VARCHAR(255),
    ingredient VARCHAR(255),
    fundamental_unit VARCHAR(255),
    amount INT DEFAULT 0,
    PRIMARY KEY (recipe, ingredient),
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name),
    FOREIGN KEY (ingredient) REFERENCES ingredients (ingredient_name)
);
CREATE TABLE dietary_info (
    recipe VARCHAR(255) PRIMARY KEY,
    dietary_category VARCHAR(255),
    total_carbs INT DEFAULT 0,
    total_protein INT DEFAULT 0,
    total_fats INT DEFAULT 0,
    total_calories INT DEFAULT 0,
    the_image VARCHAR(255) DEFAULT "../Images/the-food-pyramid-explained_1250.jpg",
    image_description VARCHAR(255) DEFAULT "Description",
    FOREIGN KEY (recipe) REFERENCES recipes (recipe_name)
);
CREATE TABLE expertise_in (
    chef_name VARCHAR(50) NOT NULL,
    chef_surname VARCHAR(50) NOT NULL,
    cuisine_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (cuisine_name) REFERENCES cuisine (cuisine_name),
    FOREIGN KEY (chef_name, chef_surname) REFERENCES chefs (chef_name, chef_surname),
    PRIMARY KEY (
        chef_name,
        chef_surname,
        cuisine_name
    )
);
CREATE TABLE Winner (
    episode_num INT,
    season INT,
    chef_surname VARCHAR(50) NOT NULL,
    chef_name VARCHAR(50) NOT NULL,
    PRIMARY KEY (episode_num, season),
    FOREIGN KEY (chef_name, chef_surname) REFERENCES chefs (chef_name, chef_surname)
);
-- ----------------------------------------
-- Triggers --
-- ----------------------------------------
DELIMITER $$
-- Create a trigger where when we create the recipe we add the cuisine to the cuisine table
CREATE TRIGGER BeforeInsertRecipe
BEFORE INSERT ON recipes
FOR EACH ROW
BEGIN
    DECLARE cuisineExists INT;
    SELECT COUNT(*) INTO cuisineExists FROM cuisine WHERE cuisine_name = NEW.cuisine_name;
    IF cuisineExists = 0 THEN
        INSERT INTO cuisine (cuisine_name) VALUES (NEW.cuisine_name);
    END IF;
END$$
-- Create a trigger when we insert the recipe to insert a new row on dietary_info
CREATE TRIGGER GetDietaryCat AFTER INSERT ON recipes FOR EACH ROW
BEGIN
    DECLARE recipe VARCHAR(255);
    DECLARE ing VARCHAR(255);
    DECLARE cat VARCHAR(255);
    DECLARE Diet VARCHAR(255);
    SET recipe = NEW.recipe_name;
    SET ing = NEW.primary_ingredient;
    SELECT food_group_name
    INTO cat
```

```
    FROM ingredients
    WHERE ingredient_name = ing;
    SELECT dietary_analogy
    INTO Diet
    FROM food_groups
    WHERE food_group_name = cat;
   INSERT INTO dietary_info(recipe, dietary_category, total_carbs, total_protein, total_fats,
total_calories)
   VALUES (recipe, Diet, 0, 0, 0, 0);
END$$
-- Create a trigger when we insert an new row to recipe has ingredient
CREATE TRIGGER GetDietaryInfo AFTER INSERT ON has_ingredient FOR EACH ROW
BEGIN
    DECLARE fat FLOAT;
    DECLARE carbs FLOAT;
    DECLARE protein FLOAT;
    DECLARE cal INT;
    DECLARE amount INT;
    DECLARE fat_i FLOAT;
    DECLARE carbs_i FLOAT;
    DECLARE protein_i FLOAT;
    DECLARE cal_i INT;
    SET amount = new.amount;
    IF new.fundamental_unit in ("Kilo", "Litre") THEN
        SET amount = 1000 * amount;
    END IF;
    IF new.fundamental_unit in ("Kilo", "Litre", "Gram", "Ml", "Piece") THEN
        SELECT total_carbs, total_protein, total_fats, total_calories
        INTO carbs, protein, fat, cal
        FROM dietary_info
        WHERE recipe = new.recipe;
        SELECT fats_per_fund_SI, protein_per_fund_SI, carbs_per_fund_SI, calories_per_fund_SI
        INTO fat_i, protein_i, carbs_i, cal_i
        FROM ingredients
        WHERE ingredient_name = new.ingredient;
        SET fat = fat + amount * fat_i;
        SET carbs = carbs + amount * carbs_i;
        SET protein = protein + amount * protein_i;
        SET cal = cal + amount * cal_i;

        UPDATE dietary_info
        SET total_carbs = carbs, total_protein = protein, total_fats = fat, total_calories = cal
        WHERE recipe = new.recipe;
    END IF;
END$$
DELIMITER ;
-- ----------------------------------------
-- INDICES --
-- ----------------------------------------
CREATE INDEX idx_recipe_name ON recipes(recipe_name);
CREATE INDEX idx_recipe_tag ON belongs_to_tag(recipe, tag);
CREATE INDEX idx_participate_recipe ON participate_in_episode_as_chef(recipe_name);
CREATE INDEX idx_needs_equipment_recipe ON needs_equipment(recipe);
CREATE INDEX three_one_a ON scores(chef_name,chef_surname);
CREATE INDEX three_one_b ON scores(cuisine);
CREATE INDEX three_three ON chefs(age);
CREATE INDEX chef_participation_a ON participate_in_episode_as_chef(season);
CREATE INDEX chef_participation_b ON participate_in_episode_as_chef(chef_name,chef_surname);
CREATE INDEX judge_participation_a ON participate_in_episode_as_judge(season);
CREATE INDEX judge_participation_b ON participate_in_episode_as_judge(judge_name,judge_surname);
```

- DML (δεν παραδίδουμε εδώ τον κώδικα διότι περιέχει 2000 γραμμές περίπου)

https://github.com/JohnnyPol/Database_Project_ECE_NTUA/blob/main/SQL%20Scripts/DML.sql

Οδηγίες Εγκατάστασης

Η βάση δεδομένων μας σχεδιάστηκε με τη χρήση του MySQL ως Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS) και η εφαρμογή αναπτύχθηκε με τη χρήση του module Flask της Python. Συνεπώς, για την εγκατάσταση της βάσης δεδομένων, θα πρέπει να εξασφαλίσουμε ότι ο υπολογιστής μας διαθέτει αυτά τα εργαλεία.

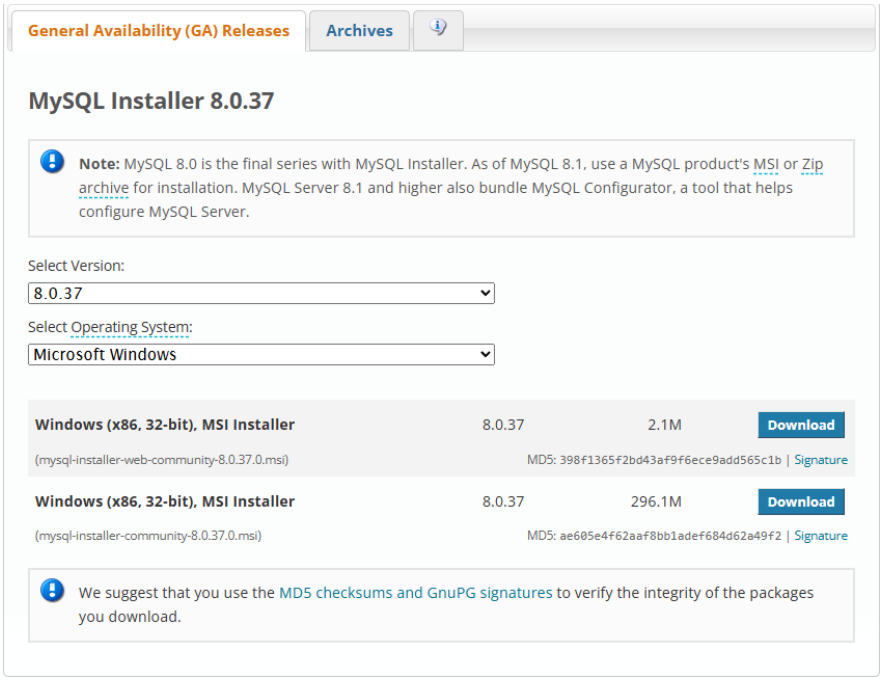
Εγκατάσταση MySQL

Windows

Σε λειτουργικό Windows, για την εγκατάσταση του MySQL θα εγκαταστήσουμε τον παρακάτω installer (έχουμε δυο επιλογές).

MySQL :: Download MySQL Installer

Πρώτος installer είναι ο mysql-installer-web-community-8.0.37.0.msi και ο δεύτερος είναι ο mysql-installer-community-8.0.37.0.msi. Εάν δεν υπάρχει πρόσβαση στο Internet κατά τη διάρκεια της εγκατάστασης κατεβάστε το δεύτερο αλλιώς δεν υπάρχει διαφορά.



Linux (Debian Based Distro)

Σε ένα τερματικό εκτελέστε τις ακόλουθες εντολές για να κατεβάσετε σε Debian Based Linux συστήματα το MySQL:

```
sudo apt update
sudo apt install mysql-server
```

Εγκατάσταση Python

Windows

Μπορούμε εγκαταστήσουμε την python είτε μέσω του διαδικτύου είτε μέσω του Microsoft Store (δίνονται τα αντίστοιχα link).

- Download Python | Python.org
- https://www.microsoft.com/store/productId/9NCVDN91XZQP?ocid=pdpshare

Linux (Debian Based Distro)

Σε ένα τερματικό εκτελέστε τις ακόλουθες εντολές:

```
sudo apt update
sudo apt install python3
```

Αφού εγκαταστήσουμε τα παραπάνω εργαλεία ήρθε η ώρα να δημιουργήσουμε την βάση δεδομένων μας. Αυτό μπορεί να υλοποιηθεί με δύο τρόπους:

1. Χρήση RDBMS

Προϋποθέτοντας πως μέσω του MySQL Installer έχουμε κατεβάσει και το MySQL Workbench (αν δεν το έχετε κατεβάσει, για δική σας διευκόλυνση κατεβάστε το) θα συνδεθούμε στον SQL server μας και θα ανοίξουμε τα ακόλουθα αρχεία στο φάκελο **SQL Scripts/App Scripts***(create_tables_and_indexes.sql,create_triggers_for_rdbms.sql, insert.sql)*. Αφού, ολοκληρώσουμε αυτό το βήμα τρέχουμε τα SQL Scripts (με την παραπάνω σειρά) για να κατασκευάσουμε την βάση δεδομένων μας (μαζί με τα tables, triggers και indexes) και να εισάγουμε τα επιθυμητά δεδομένα μας. Για το κομμάτι της κλήρωσης θα αναφερθούμε στη συνέχεια.

2. Χρήση Εφαρμογής

Η χρήση της εφαρμογής προυποθέτει πως η βάση δεδομένων έχει ήδη δημιουργηθεί, ώστε να μπορούμε να συνδεθούμε σε αυτή. Δημιουργούμε την βάση μας με την εντολή (την εντολή μπορούμε να την εκτελέσουμε είτε στο MySQL Workbench είτε σε MySQL Command Line Interface (CLI):

```
create database masterchef_ntua_edition
```

Για να μπορεί η εφαρμογή να συνδεθεί στη Βάση Δεδομένων μας πρέπει να παραθέσουμε τα σωστά credentials στο αρχείο Masterchef_DB/config.json. Αναλυτικότερα, πρέπει να αλλάξουμε τις τιμές των παρακάτω μεταβλητών με τα κατάλληλα credentials που έχετε θέσει είτε κατά την διάρκεια της εγκατάστασης του MySQL (credentials του χρήστη root) είτε αργότερα εάν δημιουργήσατε έναν καινούργιο χρήστη και θέλετε να συνδεθείτε με αυτόν (συνιστούμε να βάλετε τα credential του χρήστη root για άμεση πρόσβαση και διότι διαθέτει όλα τα privileges).

```
{
    "MYSQL_USER": "your user name",
    "MYSQL_PASSWORD": "your password",
    "MYSQL_DB": "masterchef_ntua_edition",
    "MYSQL_HOST": "localhost"
}
```

Έχοντας κατεβάσει την Python στον υπολογιστή μας, μας απομένει ένα βήμα για να τρέξουμε την εφαρμογή μας, κι αυτό είναι να κατεβάζουμε τα προ απαιτούμενα modules. Συγκεκριμένα, θα κατεβάζουμε τα modules που έχουμε διευθετήσει στο αρχείο requirements.txt με την εξής εντολή στο τερματικό μας.

```
pip install -r requirements.txt
```

Έχοντας ολοκληρώσει και αυτό το βήμα, είμαστε έτοιμοι να τρέξουμε την εφαρμογή μας (η εφαρμογή θα τρέξει τοπικά στη θύρα 3000).

Τρέχουμε το αρχείο ***run.py*** με την εντολή

```
python3 run.py
```

Έπειτα πηγαίνουμε στο browser μας και πατάμε το URL:

localhost:3000

Εάν τα credentials στο config.json αρχείο ήταν σωστά θα πρέπει να βλέπουμε πως υπήρξε μια επιτυχής σύνδεση.

Εφόσον δεν μας ζητήθηκε GUI, πήραμε την πρωτοβουλία να υλοποιήσουμε όποια διαδικασία επιθυμούσαμε με την χρήση παραμέτρων στο URL. Για την δημιουργία της βάσης δεδομένων θα θέσουμε την παράμετρο action ίση με “create_database” δηλαδή θα επισκεφθούμε το ακόλουθο URL:

localhost:3000/?action=create_database

Με την εντολή αυτή δημιουργούμε το schema της βάσης μας και εισάγουμε τα επιθυμητά δεδομένα μας.

Κλήρωση

Για την υλοποίηση της κλήρωσης, και στις δύο περιπτώσεις, θα πραγματοποιηθεί με την χρήση της εφαρμογής μας θέτοντας τις παραμέτρους action και season ίσες με “competition” και “season number” αντίστοιχα. Δηλαδή, θα επισκεφτούμε το URL:

http://localhost:3000/?action=competition&season=1

Συνάρτηση Κλήρωσης Επεισοδίων

Η συνάρτηση που κληρώνει τα επεισόδια μίας σεζόν (δηλαδή που διαλέγεο τυχαία ποιές κουζίνες, ποιοι μάγειρες και ποιες συνταγές θα συμπεριληφθούν σε κάθε επεισόδιο) παρατείθεται παρακάτω:

```
def competition(season):
    exp = {"chef": 5, "sous chef": 4, "1st cook": 3, "2nd cook": 2, "3rd cook": 1}
    cursor = db.connection.cursor()
    all_cuisines = []
    all_chefs = []
    all_recipes = []
    all_judges = []
    number_of_cuisines = 5
    number_of_judges = 3
    for i in range(10):
        first = max(0, (i - 3))
        for k in range(number_of_cuisines):
            while True:
                query_cuisine = """
                    SELECT cuisine_name
                    FROM cuisine
                    ORDER BY RAND()
                    LIMIT 1;"""
                cursor.execute(query_cuisine)
                cuisines = cursor.fetchone()
                if (k != 0) and cuisines in all_cuisines[number_of_cuisines * i :]:
                    continue
                elif (
                    all_cuisines[
                        number_of_cuisines * first : number_of_cuisines * i
                    ].count(cuisines)
                    >= 3
                ):
                    continue
                else:
                    all_cuisines.append(cuisines)
                    break
        counter = 0
        for cuisine in all_cuisines[number_of_cuisines * i :]:
            while True:

                query_chef = """
                    SELECT chef_name, chef_surname
                    FROM expertise_in
                    WHERE cuisine_name = %s
                    ORDER BY RAND()
                    LIMIT 1;"""
                cursor.execute(query_chef, (cuisine,))
                chef = cursor.fetchone()
                if (counter != 0) and (chef in all_chefs[number_of_cuisines * i :]):
                    continue
                elif (
                    (all_chefs[
                        number_of_cuisines * first : number_of_cuisines * i
                    ].count(chef) + all_judges[number_of_judges * first : number_of_judges *
i].count(chef)) >=3
                ):
                    continue
                else:
                    all_chefs.append(chef)
                    break
            counter += 1
        for cuisine in all_cuisines[number_of_cuisines * i :]:
            while True:
```

```
                query_recipe = """
                    SELECT recipe_name
                    FROM recipes
                    WHERE cuisine_name = %s
                    ORDER BY RAND()
                    LIMIT 1;"""
                cursor.execute(query_recipe, (cuisine,))
                recipe = cursor.fetchone()
                if (all_recipes[number_of_cuisines * first : number_of_cuisines *
i].count(recipe)>= 3):
                    continue
                else:
                    all_recipes.append(recipe)
                    break
        for k in range(number_of_judges):
            while True:
                query_judges = """
                    SELECT chef_name, chef_surname
                    FROM chefs
                    ORDER BY RAND()
                    LIMIT 1;"""
                cursor.execute(query_judges)
                judge = cursor.fetchone()
                if (k != 0 and (judge in all_judges[number_of_judges * i :])) or (
                    judge in all_chefs[number_of_cuisines * i :]
                ):
                    continue
                elif (
                    (all_chefs[
                        number_of_cuisines * first : number_of_cuisines * i
                    ].count(judge) + all_judges[number_of_judges * first : number_of_judges *
i].count(judge)) >=3
                ):
                    continue
                else:
                    all_judges.append(judge)
                    break
        insert_judge_query = f"""
        INSERT INTO participate_in_episode_as_judge
        VALUES (%s, %s, %s, %s) """
        for k in range(number_of_judges):
            cursor.execute(
                insert_judge_query,(
                    int(i + 1),
                    int(season),
                    all_judges[number_of_judges * i + k][0],
                    all_judges[number_of_judges * i + k][1],
                ),
            )
            db.connection.commit()
        insert_chef_query = f"""
        INSERT INTO participate_in_episode_as_chef
        VALUES (%s, %s, %s, %s, %s, %s); """
        for k in range(number_of_cuisines):
            cursor.execute(
                insert_chef_query,(
                    int(i + 1),
                    int(season),
                    all_chefs[number_of_cuisines * i + k][0],
                    all_chefs[number_of_cuisines * i + k][1],
                    all_cuisines[number_of_cuisines * i + k][0],
                    all_recipes[number_of_cuisines * i + k][0],
                ),
            )
            db.connection.commit()
        insert_recipe_query = f"""
        INSERT IGNORE INTO has_recipe
        VALUES (%s, %s, %s); """
        for k in range(number_of_cuisines):
```

```
            cursor.execute(
                insert_recipe_query,(
                    all_chefs[number_of_cuisines * i + k][0],
                    all_chefs[number_of_cuisines * i + k][1],
                    all_recipes[number_of_cuisines * i + k][0],
                ),
            )
            db.connection.commit()
        insert_score_query = f"""
        INSERT INTO scores
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"""
        winner = 0
        max_score = 0
        for k in range(number_of_cuisines):
            score = 0
            for j in range(number_of_judges):
                temp = randint(1, 5)
                cursor.execute(
                    insert_score_query,(
                        int(temp),
                        int(i + 1),
                        int(season),
                        all_chefs[number_of_cuisines * i + k][0],
                        all_chefs[number_of_cuisines * i + k][1],
                        all_cuisines[number_of_cuisines * i +k][0],
                        all_judges[number_of_judges * i + j][0],
                        all_judges[number_of_judges * i + j][1],
                    ),
                )
                db.connection.commit()
                score += temp
            if score > max_score:
                max_score = score
                winner = k
            elif score == max_score:
                find_winner_query = f"""
                SELECT experience_level
                FROM chefs
                WHERE chef_name = %s and chef_surname = %s"""
                cursor.execute(
                    find_winner_query,(
                        all_chefs[number_of_cuisines * i + winner][0],
                        all_chefs[number_of_cuisines * i + winner][1],
                    ),
                )
                fc = cursor.fetchone()
                cursor.execute(
                    find_winner_query,(
                        all_chefs[number_of_cuisines * i + k][0],
                        all_chefs[number_of_cuisines * i + k][1],
                    ),
                )
                sc = cursor.fetchone()

                if exp[str(sc[0])] > exp[str(fc[0])]:
                    winner = k
        insert_winner_query = f"""
        INSERT INTO Winner
        VALUES (%s, %s, %s, %s)"""
        cursor.execute(
            insert_winner_query,(
                int(i + 1),
                int(season),
                all_chefs[number_of_cuisines * i + winner][1],
                all_chefs[number_of_cuisines * i + winner][0],
            ),
        )
        db.connection.commit()
    return all_chefs
```

Εξήγηση λειτουργίας:

Αρχικά, συνάρτηση *competition* δέχεται ένα όρισμα ***season*** που καθορίζει τον αριθμό της σεζόν για την οποία τρέχουμε την κλήρωση.

Στη συνέχεια δημιουργούμε ένα λεξικό ***exp*** (experience) που αντιστοιχεί τη βαθμίδα του κάθε μάγειρα (*π.χ. 1st cook, sous chef κτλ.*) σε νούμερα ανάλογα με τον βαθμο εμπειρίας (θα χρειαστούμε αυτές τις αντιστοιχίσεις όταν θα αποφασίζουμε τον νικητή σε περίπτωση ισοπαλίας).

Έπειτα αρχικοποιούνται κάποιες λίστες για να αποθηκευτούν οι κουζίνες (***all_cuisines***), οι σεφ (***all_chefs***), οι συνταγές (***all_recipes***) και οι κριτές (***all_judges***) και ορίζεται ο αριθμός των κουζινών (***number_of_cuisines***) και των κριτών (***number_of_judges***).

Κατόπιν εισερχόμαστε στον βρόχο που επιλέγει τυχαία (σύμφωνα με τους περιορισμούς πάντα) τα δεδομένα για τα 10 επεισόδια της σεζόν (*range(10)*). Επίσης υπολογίζουμε τη μεταβλητή ***first***, που χρησιμοποιείται για να περιορίσει τα αποτελέσματα κάποιων αναζήτησεων για διπλότυπα στα τρία προηγούμενα επεισόδια (προκειμένου να τηρηθεί ο περιορισμός ότι δεν μπορεί κάποιος μάγειρας/κριτής/ εθνική κουζίνα/ συνταγή να συμμετέχει συνεχόμενα σε περισσότερα από 3 επεισόδια). `{first = max(0, (i - 3))}`

Στη συνέχεια επιλέγονται τυχαία κουζίνες από τη βάση δεδομένων και εξασφαλίζεται ότι η ίδια κουζίνα δεν θα εμφανιστεί περισσότερες από 3 φορές στα προηγούμενα 3 επεισόδια και δεν θα επαναληφθεί μέσα στο ίδιο επεισόδιο. Πιο συγκεκριμένα:

Έχουμε τον βρόχο `for k in range(number_of_judges)` που επαναλαμβάνεται για τον αριθμό κουζινών που έχουμε καθορίσει στο ***number_of_cuisines***. Σε κάθε επεισόδιο, ορίζονται *10* κουζίνες με βάση τις προδιαγραφές της εκφώνησης.

Ο άπειρος βρόχος `while True` χρησιμοποιείται για να συνεχίζει να ψάχνει τυχαία κουζίνα μέχρι να βρεθεί μία που να πληροί τα κριτήρια μας.

Εντός αυτού το SQL query `SELECT cuisine_name FROM cuisine ORDER BY RAND() LIMIT 1` επιλέγει τυχαία μία κουζίνα από τον πίνακα cuisine. Το `ORDER BY RAND()` εξασφαλίζει την τυχαιότητα, ενώ το `LIMIT 1` περιορίζει τα αποτελέσματα σε μία μόνο κουζίνα και το αποτέλεσμα αποθηκεύεται στη μεταβλητή ***cuisines***. Εάν δεν είναι η πρώτη επανάληψη του βρόχου (δηλαδή, δεν είναι η πρώτη κουζίνα του επεισοδίου) και η κουζίνα υπάρχει ήδη στη λίστα ***all_cuisines*** για το τρέχον επεισόδιο `all_cuisines[number_of_cuisines*i:]`, τότε παραλείπεται αυτή η κουζίνα και επανέρχεται στην αρχή του βρόχου `while True` για να επιλέξει μια νέα κουζίνα. Η χρήση του `number_of_cuisines*i` ως δείκτης εξασφαλίζει ότι ελέγχουμε μόνο τις κουζίνες του τρέχοντος επεισοδίου.

Εάν η κουζίνα εμφανίζεται τουλάχιστον 3 φορές στις κουζίνες των προηγούμενων 3 επεισοδίων `all_cuisines[number_of_cuisines*first:number_of_cuisines*i]`, τότε

παραλείπεται αυτή η κουζίνα και επανέρχεται στην αρχή του βρόχου `while True` για να επιλέξει μια νέα κουζίνα.

Το `number_of_cuisines*first` και το `number_of_cuisines*i` ορίζουν το εύρος των κουζινών που εξετάζονται, με το `first` να είναι η αρχή του εύρους (μέχρι 3 επεισόδια πίσω) και το `i` να είναι το τέλος του εύρους (τρέχον επεισόδιο). Εάν η κουζίνα περάσει όλους τους παραπάνω ελέγχους, προστίθεται στη λίστα ***all_cuisines*** (`all_cuisines.append(cuisines)`). Μετά την προσθήκη, γίνεται έξοδος από τον βρόχο `while True` μέσω της εντολής `break`, οπότε προχωράμε στην επιλογή της επόμενης κουζίνας. Με αυτούς τους ελέγχους, ο κώδικας διασφαλίζει ότι οι κουζίνες που επιλέγονται για κάθε επεισόδιο είναι διαφορετικές από αυτές των τριών προηγούμενων επεισοδίων και ότι δεν επαναλαμβάνονται μέσα στο ίδιο επεισόδιο.

Στη συνέχεια επιλέγονται τυχαία σεφ με βάση την κουζίνα τους, συνταγές με βάση την κουζίνα τους και κριτές με διαδικασία παρόμοια με την παραπάνω (όχι εμφάνιση σε πάνω από 3 συνεχόμενα επεισόδια και όχι διπλότυπα εντός επεισοδίου).

Έπειτα καταχωρούνται οι κριτές, οι σεφ και οι συνταγές για το συγκεκριμένο επεισόδιο στη βάση δεδομένων στα αντίστοιχα tables. Επίσης καταχωρούνται οι συνταγές που ανήκουν στους σεφ (θα μας βοηθήσει σε μετέπειτα query).

Ακολουθεί η διαδικασία βαθμολόγησης και επιλογής νικητή επεισοδίου:

Αρχικά ορίζεται το SQL query `INSERT` που θα χρησιμοποιηθεί για να καταχωρηθούν οι βαθμολογίες στη βάση δεδομένων. Περιλαμβάνει οκτώ πεδία: βαθμός, αριθμός επεισοδίου, σεζόν, όνομα σεφ, επώνυμο σεφ, όνομα κουζίνας, όνομα κριτή και επώνυμο κριτή. Έπειτα αρχικοποιούνται οι μεταβλητές ***winner*** (για τον δείκτη του νικητή σεφ) και ***max_score*** (για το μέγιστο σκορ).

Για κάθε κουζίνα `for k in range(number_of_cuisines)`, αρχικοποιείται η μεταβλητή ***score*** στο *0*, για κάθε κριτή `for j in range(number_of_judges)`, παράγεται ένας τυχαίος βαθμός από *1* έως *5* με τη χρήση της `randint(1, 5)` που αντιστοιχεί στη βαθμολογία που δίνει στον εκάστοτε παίκτη. Έπειτα καταχωρείται η βαθμολογία στη βάση δεδομένων με τη χρήση της ***insert_score_query***.

Για να επιλέξουμε νικητή. προστίθεται η τυχαία βαθμολογία στη μεταβλητή ***score*** και ελέγχεται αν το ***score*** είναι μεγαλύτερο από το ***max_score***. Αν ναι, ενημερώνονται οι μεταβλητές ***max_score*** και ***winner***.

Επιπλέον, αν το ***score*** είναι ίσο με το ***max_score***, ελέγχεται η εμπειρία των μαγείρων για να αποφασιστεί ο νικητής σε περίπτωση ισοπαλίας, και αν έχουμε πάλι ισοψηφία τότε θέτουμε ως νικητή τον πρώτο που κληρώθηκε (εφόσον η κλήρωση γίνεται τυχαία θεωρούμε ότι αυτό πληροί την προδιαγραφή της εκφώνησης).

Διαφορετικοί Ρόλοι Χρήστών/User Permissions

Καθορισμός Αδειών και Δικαιωμάτων (*θεωρητικά*):

Μπορούμε να χρησιμοποιήσεις τις εντολές **GRANT** και **REVOKE** της *MySQL* για να καθορίσουμε τις άδειες που έχουν οι διαχειριστές και οι μάγειρες.

Οι διαχειριστές θα πρέπει να μπορούν να έχουν πλήρη πρόσβαση στη βάση δεδομένων, οπότε θα χρησιμοποιούσαμε την παρακάτω εντολή:

```
GRANT ALL PRIVILEGES ON database_name.* TO username@localhost;
```

(με αντικατάσταση προφανώς του κατάλληλου username)

Με βάση την εκφώνηση, οι μάγειρες μπορούν να έχουν περιορισμένα δικαιώματα, όπως να επεξεργάζονται συνταγές που τους έχουν ανατεθεί και να προσθέτουν νέες συνταγές οπότε θα χρησιμοποιούσαμε τις παρακάτω εντολές:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON database_name.recipes TO
'chefuser'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON database_name.steps TO
'chefuser'@'localhost';
```

Επίσης θα μπορούσαμε να φτιάξουμε *views* για τους διαφορετικούς χρήστες, π.χ.:

```
CREATE VIEW chef_view AS
SELECT r.recipe_name, r.cuisine_name, r.difficulty, r.descriptions,
r.prep_time, r.cooking_time, r.servings
FROM recipes r
JOIN has_recipe hr ON r.recipe_name = hr.recipe
WHERE hr.chef_name = CURRENT_USER();
```

Αυτό το *view* επιτρέπει στον μάγειρα να βλέπει μόνο τις συνταγές που του έχουν ανατεθεί.

Τέλος επιπλέον περιορισμοί μπορούν να υλοποιηθούν μέσω ενός UI.

Indices

Επεξήγηση των Indices:

Τα indices (ευρετήρια) χρησιμοποιούνται για να βελτιώσουν την απόδοση των ερωτημάτων (queries) στη βάση δεδομένων. Κάθε index δημιουργείται για να επιταχύνει την αναζήτηση και την ανάκτηση δεδομένων από έναν πίνακα, ελαχιστοποιώντας τον αριθμό των γραμμών που πρέπει να εξετάσει το σύστημα βάσεων δεδομένων. Το αν θα χρησιμοποιηθούν τελικά από τον optimizer εξαρτάται από την προσέγγιση που θα επιλέξει για τη διαχείριση των queries. Παρακάτω παραθέτουμε τα indices που έχουμε δημιουργήσει.

1. **Index στο recipe_name του πίνακα recipes**

   ```
   CREATE INDEX idx_recipe_name ON recipes(recipe_name);
   ```

   Αυτό το index βοηθά στην ταχύτερη αναζήτηση και φιλτράρισμα συνταγών με βάση το όνομα της συνταγής.

2. **Σύνθετο Index στους πίνακες belongs_to_tag για (recipe, tag)**

   ```
   CREATE INDEX idx_recipe_tag ON belongs_to_tag(recipe, tag);
   ```

   Αυτό το index επιταχύνει την αναζήτηση συνταγών με βάση τα ζεύγη ετικετών (tags).

3. **Index στο recipe_name του πίνακα participate_in_episode_as_chef**

   ```
   CREATE INDEX idx_participate_recipe ON participate_in_episode_as_chef
   (recipe_name);
   ```

   Το index βοηθά στην ταχύτερη αναζήτηση επεισοδίων στα οποία συμμετείχαν συγκεκριμένες συνταγές.

4. **Index στο recipe του πίνακα needs_equipment**

   ```
   CREATE INDEX idx_needs_equipment_recipe ON needs_equipment(recipe);
   ```

   Αυτό το index βοηθά στην ταχύτερη αναζήτηση του εξοπλισμού που χρειάζεται μια συνταγή.

Επιπλέον indices:

5. `CREATE INDEX three_one_a ON scores(chef_name,chef_surname);`

6. `CREATE INDEX three_one_b ON scores(cuisine);`

7. `CREATE INDEX three_three ON chefs(age);`

8. `CREATE INDEX chef_participation_a ON participate_in_episode_as_chef (season);`

9. `CREATE INDEX chef_participation_b ON participate_in_episode_as_chef (chef_name,chef_surname);`

10. `CREATE INDEX judge_participation_a ON participate_in_episode_as_judge (season);`

11. `CREATE INDEX judge_participation_b ON participate_in_episode_as_judge (judge_name,judge_surname);`

Τα indices επιλέχθηκαν ώστε να σχετίζονται με τα queries που ζητούνται από την εκφώνηση ώστε να εξυπηρετούν αυτά. Για περισσότερα queries ενδέχεται να χρειαζόμασταν επιπλέον indices.

Παραδοχές

- Χρησιμοποιήθηκε default εικόνα και περιγραφή εικόνας για αρκετά από τα πεδία
- Θεωρήσαμε ότι για τις ποσότητες συστατικών χρησιμοποιούνται μόνο οι μονάδες μέτρησης που ορίζονται στο αντίστοιχο enum και συγκεκριμένα πολλαπλάσιά τους (όπως ml→Liter)
- θεωρήθηκε ότι οι μάγειρες έχουν μόνο (και όλες τις) συνταγές της κουζίνας στην οποία έχουν εξειδίκευση
- Τα attributes των εικόνων και των περιγραφών των εικόνων δεν φαίνονται στο Relational Diagram καθώς λόγω πίεσης χρόνου προστέθηκαν τελευταία. Θεωρούμε ότι δεν αποτελεί μεγάλο πρόβλημα εφόσον δεν αποτελούν foreign keys
- Στο dietary_info έχουμε foreign key το dietary_category στο dietary_analogy του food_group το οποίο αν και δεν αναφέρεται ρητά ως foreign key κατά την κατασκευή του table, έχουμε φροντίσει να εξασφαλίζεται η ίδια λειτουργικότητα μέσω trigger και το αναφέρουμε ως FK στο relational diagram.
- Οι περιορισμοί της κλήρωσης (όχι διπλότυπα ανα επεισόδιο και όχι ίδιες συμμετοχές σε τρία συνεχόμενα επεισόδια) εξασφαλίζονται μέσω της συνάρτησης κλήρωσης, όπως περιγράφηκε παραπάνω. Δεν έχουμε ορίσει trigger εντός της βάσης που να το εξασφαλίζει, καθώς θεωρήσαμε ότι θα πρέπει ο administrator του διαγωνισμού να μπορεί να εισάγει και μόνος του entries σε περίπτωση που για κάποιο λόγο (έκτακτη αποχώρηση) σε κάποια σεζόν δεν υπάρχουν αρκετοί διαγωνιζόμενοι ώστε να τηρηθεί π.χ. ο κανόνας του μεγίστου των τριών συνεχόμενων συμμετοχών.

Queries

1ο Query

Εκτελούμε 2 επερωτήματα, ένα για τον μέσο όρο ανά μάγειρα και έναν για τον μέσο όρα ανά κουζίνα

```
-- Question 3.1 --

SELECT chef_name, chef_surname, AVG (score) AS average_score

FROM scores

GROUP BY chef_name, chef_surname;

SELECT cuisine, AVG (score) AS average_score

FROM scores

GROUP BY cuisine;
```

2ο Query

Για δοσμένες κουζίνες και season, κάνουμε left join τον πίνακα expertise_in με τον πίνακα participate_in_episode_as_a_chef και κρατάμε μόνο τις γραμμές που στον πρώτο πίνακα είχαν τη δοσμένη κουζίνα και στον 2ο πίνακα τη δοσμένη σεζόν. Όπου έχουμε null, δηλαδή για τους μάγειρες που δεν υπήρχαν στον 2ο πίνακα και άρα δεν έχουν συμμετάσχει σε επισόδειο για τη δεδομένη σεζόν, ορίζουμε ότι δε συμμετείχαν. Ταξινομούμε τον πίνακα με βάση το αν συμμετείχαν ή όχι. Έτσι έχουμε στην αρχή τους μάγειρες της κουζίνας που εμφανίστηκαν και μετά αυτούς που δεν εμφανίστηκαν.
Σημείωση: Επειδή ζητάει κουζίνα, θεωρήσαμε ότι η εμφάνιση αφορά εμφάνιση ως μάγειρα.
Στο συγκεκριμένο query έχουμε επιλέξει Ιαπωνική κουζίνα και την 1η σεζόν.

```
-- Question 3.2 --

SELECT DISTINCT ec.chef_name,  ec.chef_surname,

    CASE

        WHEN pe.chef_name IS NOT NULL THEN 'Participated in Episode'

        ELSE 'Not Participated in Episode'

    END AS participation_status

FROM expertise_in ec

LEFT JOIN

    participate_in_episode_as_chef pe

    ON ec.chef_name = pe.chef_name

    AND ec.chef_surname = pe.chef_surname

    AND pe.season = 1

WHERE ec.cuisine_name = 'Japanese'

ORDER BY participation_status, ec.chef_name, ec.chef_surname;
```

3ο Query

Χρησιμοποιούμε 2 βοηθητικά επερωτήματα. Το πρώτο υπολογίζει τις συνταγές που έχουν εκτελέσει οι μάγειρες κάτω των 30 ετών. Το δεύτερο υπολογίζει ποιος είναι ο μέγιστος αριθμός. Έτσι το βασικό επερώτημα επιλέγει από τις σειρές τους πίνακα που δημιούργησε το πρώτο επερώτημα τις γραμμές με αριθμό συνταγών ίσο με τον μέγιστο ώστε να πάρουμε όλους τους μάγειρες με το μέγιστο.

```sql
-- Question 3.3
WITH recipe_counts AS (
    SELECT
        hr.chef_name,
        hr.chef_surname,
        COUNT(hr.recipe) AS recipe_count
    FROM
        has_recipe hr
        JOIN chefs c ON hr.chef_name = c.chef_name AND hr.chef_surname = c.chef_surname
    WHERE
        c.age < 30
    GROUP BY
        hr.chef_name,
        hr.chef_surname
),
max_recipe_count AS (
    SELECT MAX(recipe_count) AS max_count
    FROM recipe_counts
)
SELECT
    rc.chef_name,
    rc.chef_surname,
    rc.recipe_count
FROM
    recipe_counts rc
    JOIN max_recipe_count mrc ON rc.recipe_count = mrc.max_count;
```

4ο Query

Χρησιμοποιούμε ένα εμφολευμένο ερώτημα για να επιλέξουμε το όνομα των μαγείρων που είναι κριτές και έπειτα από τους μάγειρες να επιλέξουμε αυτούς που δεν ανήκουν στον πίνακα του επερωτήματος, δηλαδή δεν ήταν ποτέ κριτές.

```
-- Question 3.4 --


SELECT chef_name, chef_surname

FROM chefs

WHERE (chef_name, chef_surname) NOT IN (

    SELECT DISTINCT judge_name, judge_surname

    FROM participate_in_episode_as_judge

);
```

5ο Query

Για δεδομένη season μετράμε πόσες εμφανίσεις έχει κάθε κριτής σε αυτή μέσω του COUNT σε συνδυασμό με το GROUP BY και επιλέγουμε μόνο τις γραμμές όπου έχουμε πάνω από 3 εμφανίσεις. (Ενδεικτικά με βάση τα δεδομένα μας χρησιμοποιούμε την 1η σαιζόν γιατί εκεί έχουμε κάποιον κριτή που ικανοποιεί τη συνθήκη).

```
-- Question 3.5 –


SELECT judge_name, judge_surname, COUNT(*) AS appearance_count

FROM participate_in_episode_as_judge

WHERE season = 1

GROUP BY judge_name, judge_surname

HAVING COUNT(*) > 3;
```

6ο Query

Για την εκτέλεση του query κάνουμε 2 βοηθητικά επερωτήματα. Το πρώτο μας βρίσκει πόσα ζευγάρια tag υπάρχουν. Το δεύτερο υπολογίζει σε πόσα επεισόδια εμφανίζεται το καθένα. Και τελικά το query μας επιλέγει τα πρώτα 3. Σε περίπτωση ισοβαθμίας, θα επιλεγούν και πάλι 3 με βάση το πώς έχουνε τοποθετηθεί στον πίνακα.
Χρησιμοποιώντας την εντολή Explain παίρνουμε το ακόλουθο trace για το query plan:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>		ALL					106	100.00	Using filesort
2	DERIVED	t1		index	PRIMARY,tag,idx_recipe_tag	tag	1022		65	100.00	Using index; Using temporary; Using filesort
2	DERIVED	t2		ref	PRIMARY,tag,idx_recipe_tag	PRIMARY	1022	masterchef_ntua_edition.t1.recipe	1	33.33	Using where; Using index
2	DERIVED	r		eq_ref	PRIMARY,idx_recipe_name	PRIMARY	1022	masterchef_ntua_edition.t1.recipe	1	100.00	Using index
2	DERIVED	p		ref	idx_participate_recipe	idx_participate_recipe	1023	masterchef_ntua_edition.t1.recipe	3	100.00	Using index

Σε αυτό μπορούμε να παρατηρήσουμε ότι ο optimizer εξετάζει διαφορετικά ευρετήρια για να κάνει τα απαραίτητα join
Εκτελούμε ξανά το query, αυτή τη φορά με forced index, ώστε να υποδείξουμε ότι θέλουμε να χρησιμοποιήσει αυτόν. Παίρνοντας και πάλι το trace έχουμε:

1	PRIMARY	<derived2>		ALL					110	100.00	Using filesort
2	DERIVED	r		index	PRIMARY,idx_recipe_name	cuisine_name	1022		54	100.00	Using index; Using temporary; Using filesort
2	DERIVED	t1		ref	idx_recipe_tag	idx_recipe_tag	1022	masterchef_ntua_edition.r.recipe_name	1	100.00	Using index
2	DERIVED	t2		ref	idx_recipe_tag	idx_recipe_tag	1022	masterchef_ntua_edition.r.recipe_name	1	33.33	Using where; Using index
2	DERIVED	p		ref	idx_participate_recipe	idx_participate_recipe	1023	masterchef_ntua_edition.r.recipe_name	3	100.00	Using index

Εδώ παρατηρούμε ότι αναγκάσαμε τον optimizer να χρησιμοποιήσει συγκεκριμένα ευρετήρια. Αυτό άλλαξε και τον τρόπο που γίνεται το join, αφού παρατηρούμε ότι π.χ. αρχικά είχαμε index join στον t1, ενώ τώρα έχουμε index join στον r. Η επιλογή του r ως του πίνακα που κάνουμε index join οδήγησε σε λιγότερες γραμμές, αφού ο πίνακας recipes έχει λιγότερες γραμμές από τον πίνακα belongs_to_tag.

```sql
-- Question 3.6 --
-- Find pairs of tags in recipes then count in how many episodes they appear
WITH RecipeTagPairs AS (
    SELECT
        r.recipe_name,
        t1.tag AS tag1,
        t2.tag AS tag2
    FROM recipes r
    JOIN belongs_to_tag t1 ON r.recipe_name = t1.recipe
    JOIN belongs_to_tag t2 ON r.recipe_name = t2.recipe
    WHERE t1.tag < t2.tag
),
EpisodeTagPairs AS (
    SELECT
        rtp.tag1,
        rtp.tag2,
        COUNT(DISTINCT p.episode_no, p.season) AS episode_count
    FROM RecipeTagPairs rtp
    JOIN participate_in_episode_as_chef p ON rtp.recipe_name = p.recipe_name
    GROUP BY rtp.tag1, rtp.tag2
)
SELECT
    etp.tag1,
    etp.tag2,
    etp.episode_count
FROM EpisodeTagPairs etp
ORDER BY etp.episode_count DESC
LIMIT 3;


-- Query Plan with force index
WITH RecipeTagPairs AS (
    SELECT
        r.recipe_name,
        t1.tag AS tag1,
        t2.tag AS tag2
    FROM recipes r
    JOIN belongs_to_tag t1 USE INDEX (idx_recipe_tag) ON r.recipe_name = t1.recipe
```

```
    JOIN belongs_to_tag t2 USE INDEX (idx_recipe_tag) ON r.recipe_name = t2.recipe
    WHERE t1.tag < t2.tag
),
EpisodeTagPairs AS (
    SELECT
        rtp.tag1,
        rtp.tag2,
        COUNT(DISTINCT p.episode_no, p.season) AS episode_count
    FROM RecipeTagPairs rtp
    JOIN participate_in_episode_as_chef p USE INDEX (idx_participate_recipe) ON rtp.recipe_name =
p.recipe_name
    GROUP BY rtp.tag1, rtp.tag2
)
SELECT
    etp.tag1,
    etp.tag2,
    etp.episode_count
FROM EpisodeTagPairs etp
ORDER BY etp.episode_count DESC
LIMIT 3;
-- View the query execution plan --
EXPLAIN
WITH RecipeTagPairs AS (
    SELECT
        r.recipe_name,
        t1.tag AS tag1,
        t2.tag AS tag2
    FROM recipes r
    JOIN belongs_to_tag t1 ON r.recipe_name = t1.recipe
    JOIN belongs_to_tag t2 ON r.recipe_name = t2.recipe
    WHERE t1.tag < t2.tag
),
EpisodeTagPairs AS (
    SELECT
        rtp.tag1,
        rtp.tag2,
        COUNT(DISTINCT p.episode_no, p.season) AS episode_count
    FROM RecipeTagPairs rtp
```

```
    JOIN participate_in_episode_as_chef p ON rtp.recipe_name = p.recipe_name
    GROUP BY rtp.tag1, rtp.tag2
)
SELECT
    etp.tag1,
    etp.tag2,
    etp.episode_count
FROM EpisodeTagPairs etp
ORDER BY etp.episode_count DESC
LIMIT 3;
EXPLAIN
WITH RecipeTagPairs AS (
    SELECT
        r.recipe_name,
        t1.tag AS tag1,
        t2.tag AS tag2
    FROM recipes r
    JOIN belongs_to_tag t1 FORCE INDEX (idx_recipe_tag) ON r.recipe_name = t1.recipe
    JOIN belongs_to_tag t2 FORCE INDEX (idx_recipe_tag) ON r.recipe_name = t2.recipe
    WHERE t1.tag < t2.tag
),
EpisodeTagPairs AS (
    SELECT
        rtp.tag1,
        rtp.tag2,
        COUNT(DISTINCT p.episode_no, p.season) AS episode_count
    FROM RecipeTagPairs rtp
    JOIN participate_in_episode_as_chef p FORCE INDEX (idx_participate_recipe) ON rtp.recipe_name
= p.recipe_name
    GROUP BY rtp.tag1, rtp.tag2
)
SELECT
    etp.tag1,
    etp.tag2,
    etp.episode_count
FROM EpisodeTagPairs etp
ORDER BY etp.episode_count DESC
LIMIT 3;
```

7ο Query

Για άλλη μια φορά χρησιμοποιούμε 2 βοηθητικά επερωτήματα. Στο πρώτο υπολογίζουμε τις εμφανίσεις κάθε μάγειρα και στο δεύτερο τον μέγιστο αριθμό εμφανίσεων. Με βάση αυτόν επιλέγουμε αυτούς που έχουν λιγότερες από 5 εμφανίσεις από αυτόν.
Για τον υπολογισμό των εμφανίσεων χρησιμοποιούμε 2 τρόπους. Στον πρώτο θεωρούμε ότι μας ενδιαφέρουν μόνο οι εμφανίσεις ως μάγειρες, ενώ στον δεύτερο ότι μας ενδιαφέρουν και ως κριτές, οπότε υπολογίζουμε τον συνολικό αριθμό με χρήση union all και sum.

```sql
-- Question 3.7 --

WITH ChefEpisodeCounts AS (

    SELECT chef_name, chef_surname, COUNT(*) AS episode_count

    FROM participate_in_episode_as_chef

    GROUP BY chef_name, chef_surname

),

MaxEpisodeCount AS (

    SELECT MAX(episode_count) AS max_count FROM ChefEpisodeCounts

)

SELECT c.chef_name, c.chef_surname, episode_count FROM ChefEpisodeCounts c, MaxEpisodeCount m

WHERE c.episode_count <= m.max_count - 5;

-- Alternative that counts judges

WITH ChefEpisodeCounts AS (

    SELECT chef_name, chef_surname, SUM(appearances) AS episode_count

    FROM (

        SELECT chef_name, chef_surname, COUNT(*) AS appearances

        FROM participate_in_episode_as_chef  GROUP BY chef_name, chef_surname

        UNION ALL

        SELECT judge_name, judge_surname, COUNT(*) AS appearances

        FROM participate_in_episode_as_judge

        GROUP BY judge_name, judge_surname

    ) AS combined_results

    GROUP BY chef_name, chef_surname

),

MaxEpisodeCount AS (

    SELECT MAX(episode_count) AS max_count FROM ChefEpisodeCounts

)

SELECT c.chef_name, c.chef_surname, episode_count

FROM ChefEpisodeCounts c, MaxEpisodeCount m

WHERE c.episode_count <= m.max_count - 5;
```

8ο Query

Χρησιμοποιούμε ένα βοηθητικό επερώτημα για να υπολογίσουμε τον συνολικό εξοπλισμό που θέλει κάθε επεισόδιο, τα ταξινομούμε με φθίνουσα σειρά και παίρνουμε το 1ο. Σε περίπτωση ισοβαθμίας παίρνουμε το πρώτο με βάση τη διάταξη του πίνακα, ώστε να εξασφαλίσουμε ότι πάντα παίρνουμε μόνο 1 (Αν θέλαμε όλα με το μέγιστο θα έπρεπε να εφαρμόσουμε την τεχνική του ερωτήματος 3).
Όπως και στο 6ο επερώτημα που μας ζητείται, εφαρμόζουμε και εναλλακτικό query plan με force index και έχουμε τα ακόλουθα query execution plans για τις δύο περιπτώσεις:

1	PRIMARY	<derived2>		ALL					254	100.00	Using filesort
2	DERIVED	ne		index	PRIMARY,idx_needs_equipment_recipe	equipment_name	1022		65	100.00	Using index; Using temporary
2	DERIVED	p		ref	PRIMARY,cuisine_name,idx_participate_recipe,chef_participation_a,chef_participation_b	idx_participate_recipe	1023	masterchef_ntua_edition.ne.recipe	3	100.00	Using index

1	PRIMARY	<derived2>		ALL					254	100.00	Using filesort
2	DERIVED	ne		index	idx_needs_equipment_recipe	idx_needs_equipment_recipe	1022		65	100.00	Using index; Using temporary
2	DERIVED	p		ref	idx_participate_recipe	idx_participate_recipe	1023	masterchef_ntua_edition.ne.recipe	3	100.00	Using index

Όπως και πριν παρατηρούμε ότι ενώ στην πρώτη περίπτωση ο optimizer ελέγχει διαφορετικά ευρετήρια, στη δεύτερη χρησιμοποιεί αποκλειστικά τα δοσμένα. Στη συγκεκριμένη περίπτωση, παρόλο που χρησιμοποιούνται διαφορετικά attributes για τον πίνακα needs_equipement (στην πρώτη περίπτωση το equipment_name, στην δεύτερη το index idx_needs_equipment_recipe στο attribute recipe) έχουμε ίδια σειρά στα joins, το οποίο είναι λογικό να συμβαίνει αφού έχουμε μόνο 2 joins.

```sql
-- Question 3.8 --

-- Find the episode that used the most accessories (equipment)

WITH EpisodeEquipmentCount AS (

    SELECT p.episode_no, p.season, COUNT(ne.equipment_name) AS equipment_count

    FROM participate_in_episode_as_chef p

    JOIN needs_equipment ne ON p.recipe_name = ne.recipe

    GROUP BY p.episode_no, p.season)

SELECT * FROM EpisodeEquipmentCount eec ORDER BY eec.equipment_count DESC LIMIT 1;
```

```sql
-- Find the episode that used the most accessories (equipment) with index hints
WITH EpisodeEquipmentCount AS (
    SELECT p.episode_no, p.season, COUNT(ne.equipment_name) AS equipment_count
    FROM participate_in_episode_as_chef p FORCE INDEX (idx_participate_recipe)
    JOIN needs_equipment ne FORCE INDEX (idx_needs_equipment_recipe) ON p.recipe_name = ne.recipe
    GROUP BY p.episode_no, p.season)
SELECT eec.episode_no, eec.season, eec.equipment_count
FROM EpisodeEquipmentCount eec
ORDER BY eec.equipment_count DESC
LIMIT 1;
-- View the query execution plan --
EXPLAIN
WITH EpisodeEquipmentCount AS (
    SELECT  p.episode_no, p.season, COUNT(ne.equipment_name) AS equipment_count
    FROM participate_in_episode_as_chef p
    JOIN needs_equipment ne ON p.recipe_name = ne.recipe
    GROUP BY p.episode_no, p.season
)
SELECT * FROM EpisodeEquipmentCount eec ORDER BY eec.equipment_count DESC
LIMIT 1;
EXPLAIN
WITH EpisodeEquipmentCount AS (
    SELECT p.episode_no, p.season, COUNT(ne.equipment_name) AS equipment_count
    FROM participate_in_episode_as_chef p FORCE INDEX (idx_participate_recipe)
    JOIN needs_equipment ne FORCE INDEX (idx_needs_equipment_recipe) ON p.recipe_name = ne.recipe
    GROUP BY p.episode_no, p.season
)
SELECT
    eec.episode_no,
    eec.season,
    eec.equipment_count
FROM EpisodeEquipmentCount eec
ORDER BY eec.equipment_count DESC
LIMIT 1;
```

9ο Query

Χρησιμοποιούμε join για να βρούμε όλες τις διατροφικές πληροφορίες που αντιστοιχούν σε συνταγές οι οποίες έχουν εμφανιστεί σε επεισόδια και χρησιμοποιούμε και πάλι AVG μαζί με GROUP BY season για να υπολογίσουμε τον μέσο αριθμό υδατανθράκων ανά σεζόν

```
-- Question 3.9 --

SELECT p.season,

       AVG(d.total_carbs) AS avg_carbs_per_season

FROM participate_in_episode_as_chef p

JOIN dietary_info d ON p.recipe_name = d.recipe

GROUP BY p.season;
```

10ο Query

Αρχικά για κάθε σεζόν και κουζίνα υπολογίζουμε τις εμφανίσεις της και κρατάμε μόνο αυτές που εμφανίζονται τουλάχιστον 3 φορές. Έπειτα κοιτάμε στη σειρά τα ζεύγη διαδοχικών σεζόν και για κάθε ζεύγος κρατάμε τις συνταγές που εμφανίζονται τον ίδιο αριθμό φορών στις 2 σεζόν (π.χ. 3 φορές στην 1η και 3 φορές στη 2η σεζόν). Έπειτα διατάσουμε τα αποτελέσματα ανά ζευγάρι σεζόν και ανά κοινό αριθμό εμφανίσεων.

```
-- Question 3.10 --

WITH CuisineSeasonEntries AS (

    SELECT

        p.cuisine_name,

        p.season,

        COUNT(*) AS entries

    FROM

        participate_in_episode_as_chef p

    GROUP BY

        p.cuisine_name,

        p.season

    HAVING
```

```
        COUNT(*) >= 3
),
ConsecutiveSeasonEntries AS (
    SELECT
        cse1.cuisine_name,
        cse1.season AS season1,
        cse2.season AS season2,
        cse1.entries AS entries1,
        cse2.entries AS entries2
    FROM
        CuisineSeasonEntries cse1
    JOIN
        CuisineSeasonEntries cse2 ON cse1.cuisine_name = cse2.cuisine_name
                                  AND cse1.season = cse2.season - 1
)
SELECT
    cuisine_name,
    season1,
    season2,
    entries1 AS entries
FROM
    ConsecutiveSeasonEntries
WHERE
    entries1 = entries2
ORDER BY season1, entries;
```

11ο Query

Για κάθε ζευγάρι μάγειρα και κριτή που εμφανίζεται στον πίνακα scores υπολογίζουμε μέσο sum τους συνολικούς βαθμούς που έχει λάβει ο μάγειρας από τον κριτή. Για να μπορέσουμε να επιλέξουμε τους 5 υψηλότερους ανά μάγειρα κάνουμε partition με βάση το όνομα του μάγειρα και φθίνουσα διάταξη στα partitions. Έπειτα χρησιμοποιούμε την αριθμό σειράς σε κάθε partition ως rank για να επιλέξουμε τους 5 πρώτους. Όπως και σε πολλά από τα προηγούμενα queries επιλέγουμε ό, τι και να γίνει τους πρώτους 5, δηλαδή αν υπάρχουν ισοβαθμίες, αυτές λύνονται με βάση τη διάταξη που επιλέγει η βάση.

```
-- Question 3.11 --
WITH JudgeCumulativeScores AS (
    SELECT p.chef_name,
           p.chef_surname,
           p.judge_name,
           p.judge_surname,
           SUM(p.score) AS cumulative_score
    FROM scores p
    GROUP BY p.chef_name, p.chef_surname, p.judge_name, p.judge_surname
),
RankedJudges AS (
    SELECT chef_name,
           chef_surname,
           judge_name,
           judge_surname,
           cumulative_score,
           ROW_NUMBER() OVER(PARTITION BY chef_name, chef_surname ORDER BY
cumulative_score DESC) AS my_rank
    FROM JudgeCumulativeScores
)
SELECT chef_name,
       chef_surname,
       judge_name,
       judge_surname,
       cumulative_score
FROM RankedJudges
WHERE my_rank <= 5
ORDER BY chef_name, chef_surname;
```

12ο Query

Για κάθε επεισόδιο, υπολογίζουμε τη συνολική δυσκολία με βάση τη δυσκολία των συνταγών που εμφανίζονται. Χρησιμοποιούμε partitions με βάση τη σεζόν τα οποία διατάσσονται με φθίνουσα σειρά δυσκολία και συνεπώς επιλέγοντας την πρώτη σειρά έχουμε το πιο δύσκολο επεισόδιο ανά σεζόν.

```
-- Question 3.12 --

WITH EpisodeCumulativeDifficulty AS (

    SELECT season,

           episode_no,

           SUM(difficulty) AS cumulative_difficulty

    FROM participate_in_episode_as_chef

    JOIN recipes ON participate_in_episode_as_chef.recipe_name =
recipes.recipe_name

    GROUP BY season, episode_no

),

RankedEpisodes AS (

    SELECT season,

           episode_no,

           cumulative_difficulty,

           ROW_NUMBER() OVER(PARTITION BY season ORDER BY
cumulative_difficulty DESC) AS my_rank

    FROM EpisodeCumulativeDifficulty

)

SELECT season,

       episode_no,

       cumulative_difficulty

FROM RankedEpisodes

WHERE my_rank = 1;
```

13ο Query

Αντίστοιχα με πριν θέλουμε να υπολογίσουμε το επεισόδιο με την ελάχιστη κατάρτιση. Η βασική διαφορά είναι ότι θα χρειαστούμε την πληροφορία με βάση τόσο τους κριτές όσο και τους μάγειρες. Οπότε βρίσκουμε ξεχωριστά την κατάρτιση των μαγείρων και των κριτών, κάνουμε natural join ώστε να πάρουμε έναν πίνακα με το επισόδειο και τις δύο καταρτίσεις τις οποίες προσθέτουμε για να πάρουμε τη συνολική κατάρτιση ανά επεισόδιο. Έπειτα εφαρμόζουμε αντίστοιχη διαδικασία με το προηγούμενο query για την επιλογή του επεισοδίου με τη μικρότερη κατάρτιση ανά σεζόν.

```sql
-- Question 3.13 --
WITH EpisodeChefCumulativeExperience AS (
    SELECT
        p.season,
        p.episode_no,
        SUM(
            CASE
                WHEN c.experience_level = 'chef' THEN 5
                WHEN c.experience_level = 'sous chef' THEN 4
                WHEN c.experience_level = '1st cook' THEN 3
                WHEN c.experience_level = '2nd cook' THEN 2
                WHEN c.experience_level = '3rd cook' THEN 1
                ELSE 0
            END
        ) AS cumulative_chef_experience
    FROM participate_in_episode_as_chef p
    JOIN chefs c ON p.chef_name = c.chef_name AND p.chef_surname = c.chef_surname
    GROUP BY p.season, p.episode_no
),
EpisodeJudgeCumulativeExperience AS (
    SELECT
        pj.season,
        pj.episode_no,
        SUM(
            CASE
                WHEN j.experience_level = 'chef' THEN 5
                WHEN j.experience_level = 'sous chef' THEN 4
```

```
                WHEN j.experience_level = '1st cook' THEN 3
                WHEN j.experience_level = '2nd cook' THEN 2
                WHEN j.experience_level = '3rd cook' THEN 1
                ELSE 0
            END
        ) AS cumulative_judge_experience
    FROM participate_in_episode_as_judge pj
    JOIN chefs j ON pj.judge_name = j.chef_name AND pj.judge_surname = j.chef_surname
    GROUP BY pj.season, pj.episode_no
),
CumulativeEpisodeExperience AS (
    SELECT season,
           episode_no,
           (c.cumulative_chef_experience + j.cumulative_judge_experience) AS
cumulative_experience
    FROM EpisodeChefCumulativeExperience c
    NATURAL JOIN EpisodeJudgeCumulativeExperience j
),
RankedEpisodes AS (
    SELECT season,
           episode_no,
           cumulative_experience,
           ROW_NUMBER() OVER(PARTITION BY season ORDER BY cumulative_experience) AS
my_rank
    FROM CumulativeEpisodeExperience
)
SELECT season,
       episode_no,
       cumulative_experience
FROM RankedEpisodes
WHERE my_rank = 1;
```

14ο Query

Υπολογίζουμε και πάλι με χρήση join πόσες φορές εμφανίζεται κάθε θεματική ενότητα, διατάσουμε τον πίνακα και παίρνουμε το πρώτο στοιχείο. Σε περίπτωση ισοψηφίας διαλέγουμε αυτό που έχει ορίσει η δευτερεύουσα διάταξη πρώτο. Αν θέλαμε όλα με την υψηλότερη θα ακολουθούσαμε την τακτική του ερωτήματος 3.

```
-- Question 3.14 --

SELECT b.theme, COUNT(*) AS appearance_count FROM belongs_to_theme b

JOIN participate_in_episode_as_chef p ON b.recipe = p.recipe_name

GROUP BY b.theme

ORDER BY appearance_count DESC

LIMIT 1;
```

15ο Query

Θέλουμε να βρούμε τις διατροφικές κατηγορίες που δεν εμφανίζονται στον διαγωνισμό. Οπότε κάνουμε LEFT JOIN τα food_groups με τον πίνακα που μας δείχνει τα food_groups που έχουν εμφανιστεί. Αυτός ο πίνακες έχει προκύψει με joins που για κάθε συνταγή του διαγωνισμού δίνουν τα υλικά που έχει αυτή και για αυτά την κατηγορία τους. Όσα στοιχεία του τελικού πίνακα έχουν null τις στήλες που αντιστοιχούν στα attributes του 2ου πίνακα, δηλαδή όσα αντιστοιχούν στις διατροφικές κατηγορίες που δεν εμφανίστηκαν, επιλέγονται και εμφανίζονται τα αντίστοιχα food_groups.

```
-- Question 3.15 –

SELECT fg.food_group_name

FROM food_groups fg

LEFT JOIN ( SELECT DISTINCT i.food_group_name FROM ingredients i

    JOIN has_ingredient hi ON i.ingredient_name = hi.ingredient

    JOIN participate_in_episode_as_chef p ON hi.recipe = p.recipe_name

) AS appeared_food_groups

ON fg.food_group_name = appeared_food_groups.food_group_name

-- Step 3: Find food groups that have never appeared

WHERE appeared_food_groups.food_group_name IS NULL;
```