ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

ΑΝΑΦΟΡΑ ΕΞΑΜΗΝΙΑΙΑΣ ΕΡΓΑΣΙΑΣ

Σύστημα αποθήκευσης και διαχείρισης πληροφοριών διαγωνισμού μαγειρικής

(https://github.com/georginio2000/databases2024_team122)

Κουσερής Γεώργιος – 03121004

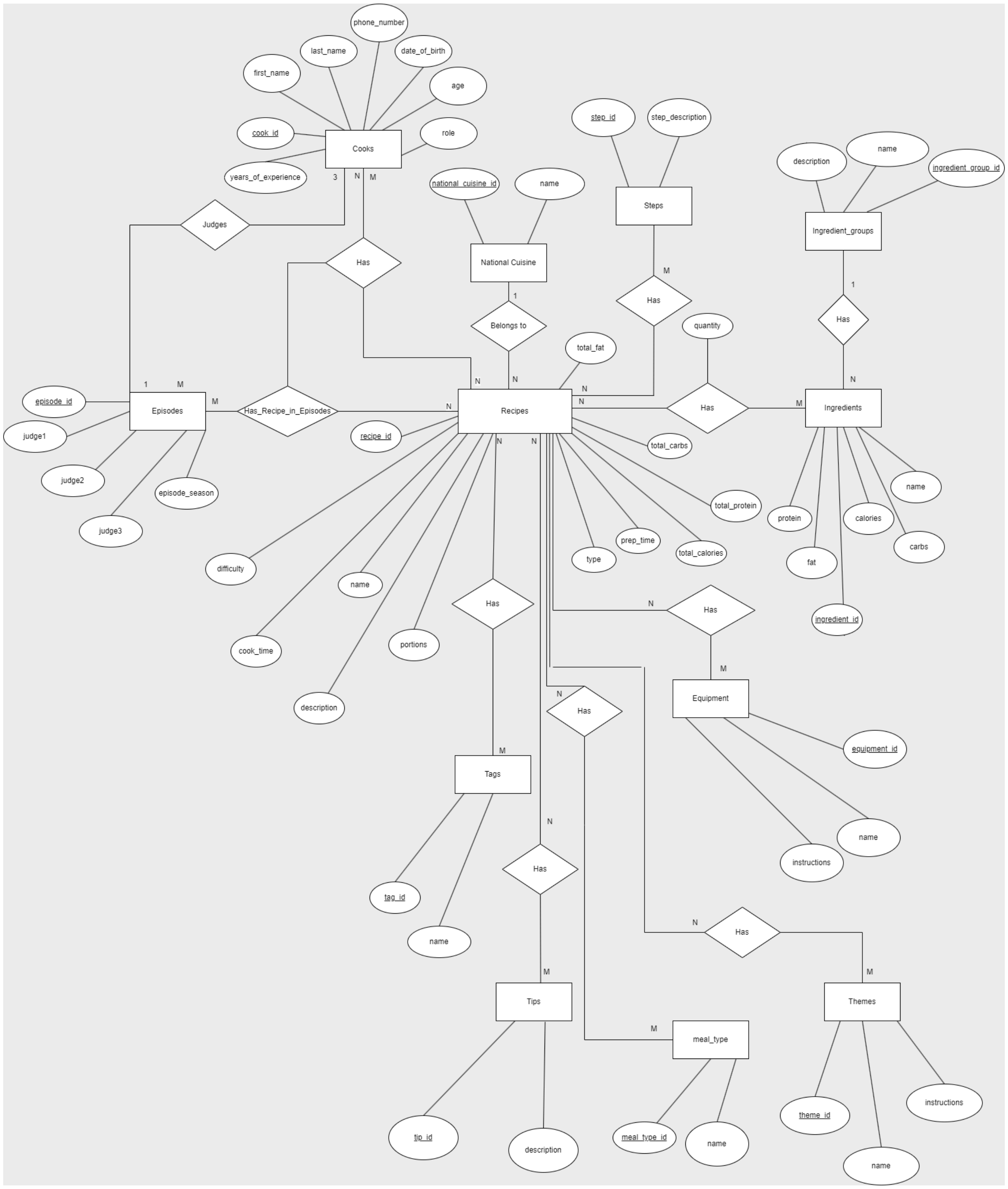
Μπεληγιάννης Νικόλαος– 03121878

Γκιώκας Νικόλαος– 03121156

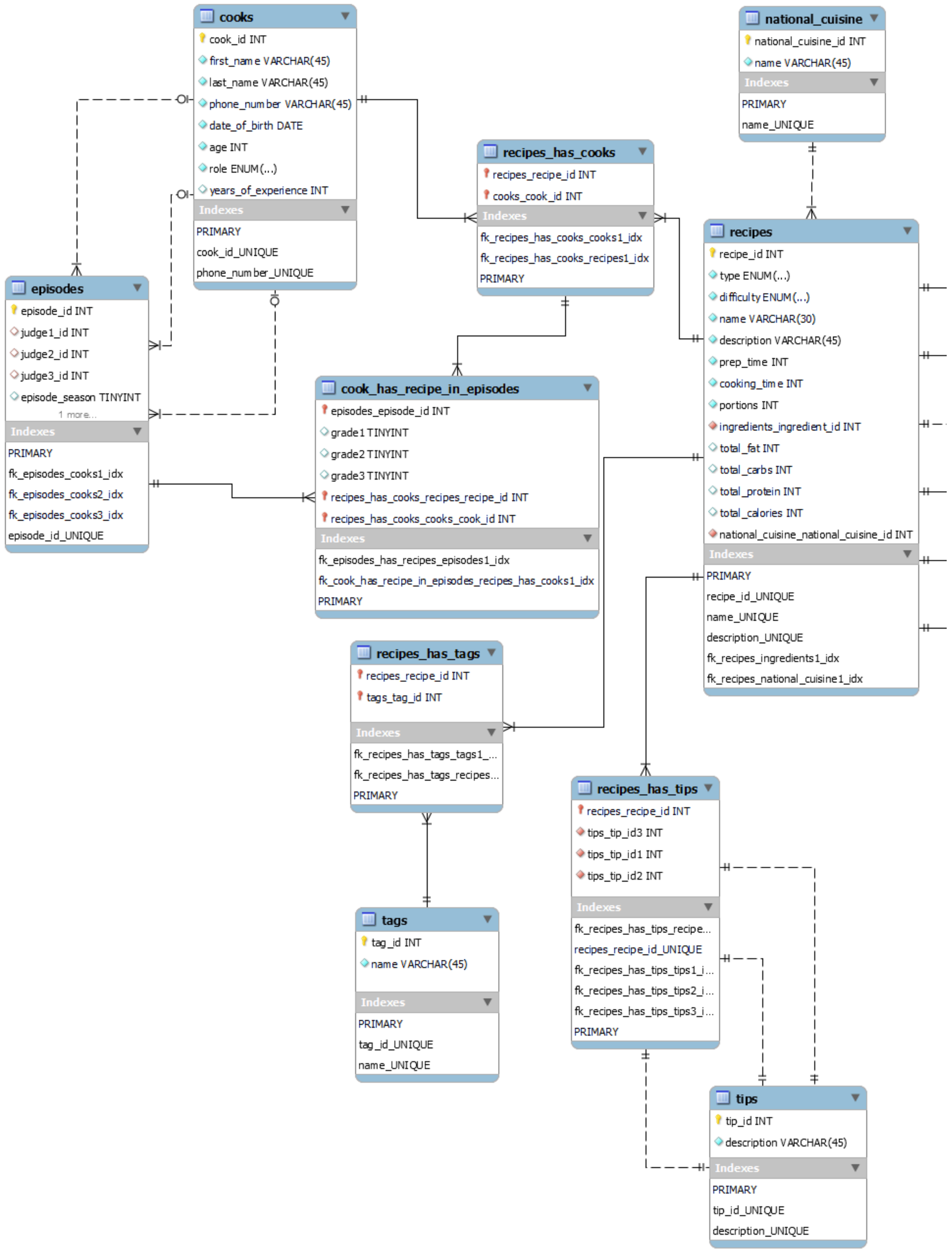
<u>ΠΕΡΙΕΧΟΜΕΝΑ</u>

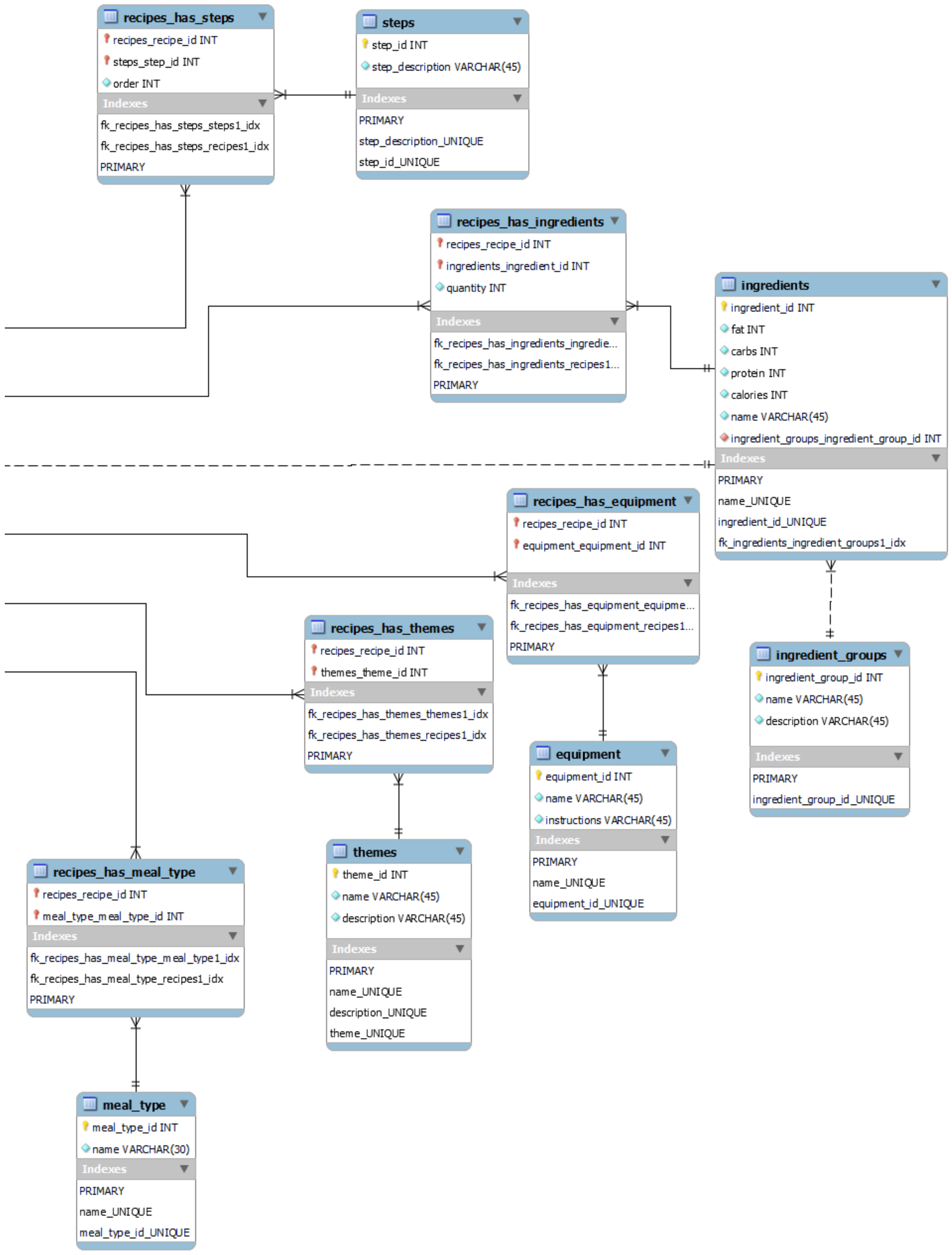
Σχεδίαση

ER

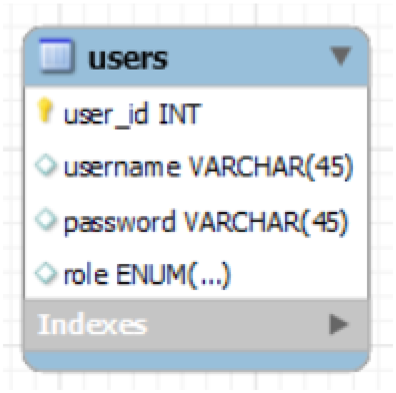


RELATIONAL SCHEMA


cooks
cook_id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
phone_number VARCHAR(45)
date_of_birth DATE
age INT
role ENUM(...)
years_of_experience INT
Indexes
PRIMARY
cook_id_UNIQUE
phone_number_UNIQUE
national_cuisine
national_cuisine_id INT
name VARCHAR(45)
Indexes
PRIMARY
name_UNIQUE
recipes_has_cooks
recipes_recipe_id INT
cooks_cook_id INT
Indexes
fk_recipes_has_cooks_cooks1_idx
fk_recipes_has_cooks_recipes1_idx
PRIMARY
recipes
recipe_id INT
type ENUM(...)
difficulty ENUM(...)
name VARCHAR(30)
description VARCHAR(45)
prep_time INT
cooking_time INT
portions INT
ingredients_ingredient_id INT
total_fat INT
total_carbs INT
total_protein INT
total_calories INT
national_cuisine_national_cuisine_id INT
Indexes
PRIMARY
recipe_id_UNIQUE
name_UNIQUE
description_UNIQUE
fk_recipes_ingredients1_idx
fk_recipes_national_cuisine1_idx
episodes
episode_id INT
judge1_id INT
judge2_id INT
judge3_id INT
episode_season TINYINT
1 more...
Indexes
PRIMARY
fk_episodes_cooks1_idx
fk_episodes_cooks2_idx
fk_episodes_cooks3_idx
episode_id_UNIQUE
cook_has_recipe_in_episodes
episodes_episode_id INT
grade1 TINYINT
grade2 TINYINT
grade3 TINYINT
recipes_has_cooks_recipes_recipe_id INT
recipes_has_cooks_cooks_cook_id INT
Indexes
fk_episodes_has_recipes_episodes1_idx
fk_cook_has_recipe_in_episodes_recipes_has_cooks1_idx
PRIMARY
recipes_has_tags
recipes_recipe_id INT
tags_tag_id INT
Indexes
fk_recipes_has_tags_tags1_...
fk_recipes_has_tags_recipes...
PRIMARY
tags
tag_id INT
name VARCHAR(45)
Indexes
PRIMARY
tag_id_UNIQUE
name_UNIQUE
recipes_has_tips
recipes_recipe_id INT
tips_tip_id3 INT
tips_tip_id1 INT
tips_tip_id2 INT
Indexes
fk_recipes_has_tips_recipe...
recipes_recipe_id_UNIQUE
fk_recipes_has_tips_tips1_i...
fk_recipes_has_tips_tips2_i...
fk_recipes_has_tips_tips3_i...
PRIMARY
tips
tip_id INT
description VARCHAR(45)
Indexes
PRIMARY
tip_id_UNIQUE
description_UNIQUE

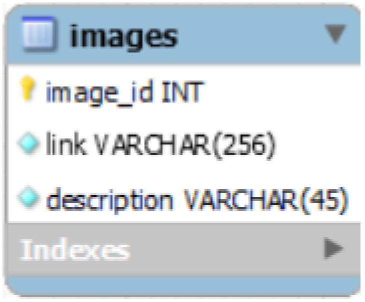

recipes_has_steps
recipes_recipe_id INT
steps_step_id INT
order INT
Indexes
fk_recipes_has_steps_steps1_idx
fk_recipes_has_steps_recipes1_idx
PRIMARY
steps
step_id INT
step_description VARCHAR(45)
Indexes
PRIMARY
step_description_UNIQUE
step_id_UNIQUE
recipes_has_ingredients
recipes_recipe_id INT
ingredients_ingredient_id INT
quantity INT
Indexes
fk_recipes_has_ingredients_ingredie...
fk_recipes_has_ingredients_recipes1...
PRIMARY
ingredients
ingredient_id INT
fat INT
carbs INT
protein INT
calories INT
name VARCHAR(45)
ingredient_groups_ingredient_group_id INT
Indexes
PRIMARY
name_UNIQUE
ingredient_id_UNIQUE
fk_ingredients_ingredient_groups1_idx
recipes_has_equipment
recipes_recipe_id INT
equipment_equipment_id INT
Indexes
fk_recipes_has_equipment_equipme...
fk_recipes_has_equipment_recipes1...
PRIMARY
recipes_has_themes
recipes_recipe_id INT
themes_theme_id INT
Indexes
fk_recipes_has_themes_themes1_idx
fk_recipes_has_themes_recipes1_idx
PRIMARY
ingredient_groups
ingredient_group_id INT
name VARCHAR(45)
description VARCHAR(45)
Indexes
PRIMARY
ingredient_group_id_UNIQUE
equipment
equipment_id INT
name VARCHAR(45)
instructions VARCHAR(45)
Indexes
PRIMARY
name_UNIQUE
equipment_id_UNIQUE
recipes_has_meal_type
recipes_recipe_id INT
meal_type_meal_type_id INT
Indexes
fk_recipes_has_meal_type_meal_type1_idx
fk_recipes_has_meal_type_recipes1_idx
PRIMARY
themes
theme_id INT
name VARCHAR(45)
description VARCHAR(45)
Indexes
PRIMARY
name_UNIQUE
description_UNIQUE
theme_UNIQUE
meal_type
meal_type_id INT
name VARCHAR(30)
Indexes
PRIMARY
name_UNIQUE
meal_type_id_UNIQUE

Για χρήση στο authentication προστέθηκε επίσης το εξής table



το οποίο βλέπουν οι cooks μέσω ενός FK.

Για πιθανή εισαγωγή εικόνων αρχικοποιούμε επίσης το table:



το οποίο οποιοδήποτε entity μπορεί να δεί επίσης μέσω ενός FK.

Το ER προέκυψε έπειτα από προσεκτική μελέτη των απαιτήσεων και έχοντας δώσει περισσότερη προτεραιότητα στην ορθότητα των δεδομένων και των σχέσεών τους και λιγότερη στην απόδοση. Έπειτα, το relational σχήμα σχεδιάστηκε σύμφωνα με την κανονική μορφή Boyce-Codd.

Υλοποιηση

DDL

```
DROP TABLE IF EXISTS cook_has_recipe_in_episodes;
DROP TABLE IF EXISTS recipes_has_cooks;
DROP TABLE IF EXISTS recipes_has_themes;
DROP TABLE IF EXISTS recipes_has_ingredients;
DROP TABLE IF EXISTS recipes_has_equipment;
DROP TABLE IF EXISTS recipes_has_tips;
DROP TABLE IF EXISTS recipes_has_tags;
DROP TABLE IF EXISTS recipes_has_meal_type;
DROP TABLE IF EXISTS recipes_has_steps;
DROP TABLE IF EXISTS tags;
DROP TABLE IF EXISTS episodes;
DROP TABLE IF EXISTS cooks;
DROP TABLE IF EXISTS themes;
DROP TABLE IF EXISTS steps;
DROP TABLE IF EXISTS equipment;
DROP TABLE IF EXISTS meal_type;
DROP TABLE IF EXISTS tips;
DROP TABLE IF EXISTS recipes;
DROP TABLE IF EXISTS national_cuisine;
DROP TABLE IF EXISTS ingredients;
DROP TABLE IF EXISTS ingredient_groups;
DROP TABLE IF EXISTS users;

-- Table ingredient_groups

CREATE TABLE IF NOT EXISTS ingredient_groups (
  ingredient_group_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  description VARCHAR(45) NOT NULL,
  PRIMARY KEY (ingredient_group_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX ingredient_group_id_UNIQUE ON ingredient_groups (ingredient_group_id);

-- Table ingredients

CREATE TABLE IF NOT EXISTS ingredients (
  ingredient_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  fat INT UNSIGNED NOT NULL,
  carbs INT UNSIGNED NOT NULL,
  protein INT UNSIGNED NOT NULL,
  calories INT UNSIGNED NOT NULL,
  name VARCHAR(45) NOT NULL,
  ingredient_groups_ingredient_group_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (ingredient_id),
  CONSTRAINT fk_ingredients_ingredient_groups1
    FOREIGN KEY (ingredient_groups_ingredient_group_id)
    REFERENCES ingredient_groups (ingredient_group_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;



CREATE UNIQUE INDEX name_UNIQUE ON ingredients (name);
CREATE UNIQUE INDEX ingredient_id_UNIQUE ON ingredients (ingredient_id);
CREATE INDEX fk_ingredients_ingredient_groups1_idx ON ingredients
(ingredient_groups_ingredient_group_id);

-- Table national_cuisine
CREATE TABLE IF NOT EXISTS national_cuisine (
  national_cuisine_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  PRIMARY KEY (national_cuisine_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX name_UNIQUE ON national_cuisine (name);

-- Table recipes
CREATE TABLE IF NOT EXISTS recipes (
  recipe_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  type ENUM("COOKING", "BAKING") NOT NULL,
  difficulty ENUM("VERY_EASY", "EASY", "NORMAL", "DIFFICULT", "VERY_DIFFICULT") NOT NULL,
  name VARCHAR(30) NOT NULL,
  description VARCHAR(45) NOT NULL,
  prep_time INT UNSIGNED NOT NULL,
  cooking_time INT UNSIGNED NOT NULL,
  portions INT NOT NULL,
  ingredients_ingredient_id INT UNSIGNED NOT NULL,
  national_cuisine_national_cuisine_id INT UNSIGNED NOT NULL,
  total_fat INT NULL,
  total_carbs INT NULL,
  total_protein INT NULL,
  total_calories INT NULL,
  PRIMARY KEY (recipe_id),
  CONSTRAINT fk_recipes_ingredients1
    FOREIGN KEY (ingredients_ingredient_id)
    REFERENCES ingredients (ingredient_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_national_cuisine1
    FOREIGN KEY (national_cuisine_national_cuisine_id)
    REFERENCES national_cuisine (national_cuisine_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE UNIQUE INDEX recipe_id_UNIQUE ON recipes (recipe_id);
CREATE UNIQUE INDEX name_UNIQUE ON recipes (name);
CREATE UNIQUE INDEX description_UNIQUE ON recipes (description);
CREATE INDEX fk_recipes_ingredients1_idx ON recipes (ingredients_ingredient_id);
CREATE INDEX fk_recipes_national_cuisine1_idx ON recipes (national_cuisine_national_cuisine_id);

-- Table tips
CREATE TABLE IF NOT EXISTS tips (
  tip_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  description VARCHAR(45) NOT NULL,
  PRIMARY KEY (tip_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX tip_id_UNIQUE ON tips (tip_id);
CREATE UNIQUE INDEX description_UNIQUE ON tips (description);

-- Table meal_type
CREATE TABLE IF NOT EXISTS meal_type (
  meal_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(30) NOT NULL,
  PRIMARY KEY (meal_type_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX name_UNIQUE ON meal_type (name);
CREATE UNIQUE INDEX meal_type_id_UNIQUE ON meal_type (meal_type_id);

-- Table equipment
CREATE TABLE IF NOT EXISTS equipment (
  equipment_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  instructions VARCHAR(45) NOT NULL,
  PRIMARY KEY (equipment_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX name_UNIQUE ON equipment (name);
CREATE UNIQUE INDEX equipment_id_UNIQUE ON equipment (equipment_id);

-- Table steps
CREATE TABLE IF NOT EXISTS steps (
  step_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  step_description VARCHAR(45) NOT NULL,
  PRIMARY KEY (step_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX step_description_UNIQUE ON steps (step_description);
CREATE UNIQUE INDEX step_id_UNIQUE ON steps (step_id);

-- Table themes
CREATE TABLE IF NOT EXISTS themes (
  theme_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  description VARCHAR(45) NOT NULL,
  PRIMARY KEY (theme_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX name_UNIQUE ON themes (name);
CREATE UNIQUE INDEX description_UNIQUE ON themes (description);
CREATE UNIQUE INDEX theme_UNIQUE ON themes (theme_id);

-- Table cooks
CREATE TABLE IF NOT EXISTS cooks (
  cook_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(45) NOT NULL,
  last_name VARCHAR(45) NOT NULL,
  phone_number VARCHAR(45) NOT NULL,
  date_of_birth DATE NOT NULL,
  age INT UNSIGNED NOT NULL,
  role ENUM("A", "B", "C", "SOUS_CHEF", "CHEF") NOT NULL,
  years_of_experience INT UNSIGNED NULL,
  PRIMARY KEY (cook_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX cook_id_UNIQUE ON cooks (cook_id);
CREATE UNIQUE INDEX phone_number_UNIQUE ON cooks (phone_number);
```

```sql
-- Table episodes
CREATE TABLE IF NOT EXISTS episodes (
  episode_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  judge1_id INT UNSIGNED NULL,
  judge2_id INT UNSIGNED NULL,
  judge3_id INT UNSIGNED NULL,
  episode_season TINYINT NULL,
  episode TINYINT NULL,
  PRIMARY KEY (episode_id),
  CONSTRAINT fk_episodes_cooks1
    FOREIGN KEY (judge1_id)
    REFERENCES cooks (cook_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_episodes_cooks2
    FOREIGN KEY (judge2_id)
    REFERENCES cooks (cook_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_episodes_cooks3
    FOREIGN KEY (judge3_id)
    REFERENCES cooks (cook_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE UNIQUE INDEX episode_id ON episodes (episode_id);
CREATE INDEX fk_episodes_cooks1_idx ON episodes (judge1_id);
CREATE INDEX fk_episodes_cooks2_idx ON episodes (judge2_id);
CREATE INDEX fk_episodes_cooks3_idx ON episodes (judge3_id);

-- Table tags
CREATE TABLE IF NOT EXISTS tags (
  tag_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  PRIMARY KEY (tag_id))
ENGINE = InnoDB;

CREATE UNIQUE INDEX tag_id_UNIQUE ON tags (tag_id);
CREATE UNIQUE INDEX name_UNIQUE ON tags (name);

-- Table recipes_has_steps
CREATE TABLE IF NOT EXISTS recipes_has_steps(
  recipes_recipe_id INT UNSIGNED NOT NULL,
  steps_step_id INT UNSIGNED NOT NULL,
  `order` INT UNSIGNED NOT NULL,
  PRIMARY KEY (recipes_recipe_id, steps_step_id),
  CONSTRAINT fk_recipes_has_steps_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_steps_steps1
    FOREIGN KEY (steps_step_id)
    REFERENCES steps (step_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_steps_steps1_idx ON recipes_has_steps (steps_step_id);
CREATE INDEX fk_recipes_has_steps_recipes1_idx ON recipes_has_steps (recipes_recipe_id);
```

```sql
-- Table recipes_has_meal_type
CREATE TABLE IF NOT EXISTS recipes_has_meal_type (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  meal_type_meal_type_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (meal_type_meal_type_id, recipes_recipe_id),
  CONSTRAINT fk_recipes_has_meal_type_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_meal_type_meal_type1
    FOREIGN KEY (meal_type_meal_type_id)
    REFERENCES meal_type (meal_type_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_meal_type_meal_type1_idx ON recipes_has_meal_type
(meal_type_meal_type_id);
CREATE INDEX fk_recipes_has_meal_type_recipes1_idx ON recipes_has_meal_type (recipes_recipe_id);

-- Table recipes_has_tags
CREATE TABLE IF NOT EXISTS recipes_has_tags (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  tags_tag_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (tags_tag_id, recipes_recipe_id),
  CONSTRAINT fk_recipes_has_tags_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_tags_tags1
    FOREIGN KEY (tags_tag_id)
    REFERENCES tags (tag_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_tags_tags1_idx ON recipes_has_tags (tags_tag_id);
CREATE INDEX fk_recipes_has_tags_recipes1_idx ON recipes_has_tags (recipes_recipe_id);


-- Table recipes_has_tips
CREATE TABLE IF NOT EXISTS recipes_has_tips (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  tips_tip_id3 INT UNSIGNED NOT NULL,
  tips_tip_id1 INT UNSIGNED NOT NULL,
  tips_tip_id2 INT UNSIGNED NOT NULL,
  PRIMARY KEY (recipes_recipe_id),
  CONSTRAINT fk_recipes_has_tips_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_tips_tips1
    FOREIGN KEY (tips_tip_id3)
    REFERENCES tips (tip_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_tips_tips2
```

```sql
    FOREIGN KEY (tips_tip_id1)
    REFERENCES tips (tip_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_tips_tips3
    FOREIGN KEY (tips_tip_id2)
    REFERENCES tips (tip_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_tips_recipes1_idx ON recipes_has_tips (recipes_recipe_id);
CREATE UNIQUE INDEX recipes_recipe_id_UNIQUE ON recipes_has_tips (recipes_recipe_id);
CREATE INDEX fk_recipes_has_tips_tips1_idx ON recipes_has_tips (tips_tip_id3);
CREATE INDEX fk_recipes_has_tips_tips2_idx ON recipes_has_tips (tips_tip_id1);
CREATE INDEX fk_recipes_has_tips_tips3_idx ON recipes_has_tips (tips_tip_id2);

-- Table recipes_has_equipment
CREATE TABLE IF NOT EXISTS recipes_has_equipment (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  equipment_equipment_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (equipment_equipment_id, recipes_recipe_id),
  CONSTRAINT fk_recipes_has_equipment_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_equipment_equipment1
    FOREIGN KEY (equipment_equipment_id)
    REFERENCES equipment (equipment_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_equipment_equipment1_idx ON recipes_has_equipment
(equipment_equipment_id);
CREATE INDEX fk_recipes_has_equipment_recipes1_idx ON recipes_has_equipment (recipes_recipe_id);

-- Table recipes_has_ingredients
CREATE TABLE IF NOT EXISTS recipes_has_ingredients (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  ingredients_ingredient_id INT UNSIGNED NOT NULL,
  quantity INT NOT NULL,
  PRIMARY KEY (ingredients_ingredient_id, recipes_recipe_id),
  CONSTRAINT fk_recipes_has_ingredients_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_ingredients_ingredients1
    FOREIGN KEY (ingredients_ingredient_id)
    REFERENCES ingredients (ingredient_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_ingredients_ingredients1_idx ON recipes_has_ingredients
(ingredients_ingredient_id);
CREATE INDEX fk_recipes_has_ingredients_recipes1_idx ON recipes_has_ingredients
(recipes_recipe_id);
```

```sql
-- Table recipes_has_themes
CREATE TABLE IF NOT EXISTS recipes_has_themes (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  themes_theme_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (recipes_recipe_id, themes_theme_id),
  CONSTRAINT fk_recipes_has_themes_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_themes_themes1
    FOREIGN KEY (themes_theme_id)
    REFERENCES themes (theme_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX fk_recipes_has_themes_themes1_idx ON recipes_has_themes (themes_theme_id);
CREATE INDEX fk_recipes_has_themes_recipes1_idx ON recipes_has_themes (recipes_recipe_id);

-- Table recipes_has_cooks
CREATE TABLE IF NOT EXISTS recipes_has_cooks (
  recipes_recipe_id INT UNSIGNED NOT NULL,
  cooks_cook_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (recipes_recipe_id, cooks_cook_id),
  CONSTRAINT fk_recipes_has_cooks_recipes1
    FOREIGN KEY (recipes_recipe_id)
    REFERENCES recipes (recipe_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_recipes_has_cooks_cooks1
    FOREIGN KEY (cooks_cook_id)
    REFERENCES cooks (cook_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
KEY_BLOCK_SIZE = 2;

CREATE INDEX fk_recipes_has_cooks_cooks1_idx ON recipes_has_cooks (cooks_cook_id);
CREATE INDEX fk_recipes_has_cooks_recipes1_idx ON recipes_has_cooks (recipes_recipe_id);

-- Table cook_has_recipe_in_episodes
CREATE TABLE IF NOT EXISTS cook_has_recipe_in_episodes (
  episodes_episode_id INT UNSIGNED NOT NULL,
  recipes_has_cooks_recipes_recipe_id INT UNSIGNED NOT NULL,
  recipes_has_cooks_cooks_cook_id INT UNSIGNED NOT NULL,
  grade1 TINYINT NULL,
  grade2 TINYINT NULL,
  grade3 TINYINT NULL,
  PRIMARY KEY (episodes_episode_id, recipes_has_cooks_cooks_cook_id,
recipes_has_cooks_recipes_recipe_id),
  CONSTRAINT fk_episodes_has_recipes_episodes1
    FOREIGN KEY (episodes_episode_id)
    REFERENCES episodes (episode_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_cook_has_recipe_in_episodes_recipes_has_cooks1
    FOREIGN KEY (recipes_has_cooks_recipes_recipe_id , recipes_has_cooks_cooks_cook_id)
    REFERENCES recipes_has_cooks (recipes_recipe_id , cooks_cook_id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;

CREATE INDEX fk_episodes_has_recipes_episodes1_idx ON cook_has_recipe_in_episodes
(episodes_episode_id);
CREATE INDEX fk_cook_has_recipe_in_episodes_recipes_has_cooks1_idx ON cook_has_recipe_in_episodes
(recipes_has_cooks_recipes_recipe_id, recipes_has_cooks_cooks_cook_id);


DROP TABLE IF EXISTS users ;
CREATE TABLE IF NOT EXISTS users (
  user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  username VARCHAR(45) NOT NULL,
  password VARCHAR(60) NOT NULL,  -- Assuming using bcrypt which generates 60-character hashes
  role ENUM('admin', 'user') NOT NULL,
  PRIMARY KEY (user_id),
  UNIQUE INDEX username_UNIQUE (username))
ENGINE = InnoDB;
```

PROCEDURES AND TRIGGERS

Ορίζουμε procedures για ανανέωση διατροφικών στοιχείων(θεωρητικά οποιαδήποτε αλλαγή στο table recipes_has_ingredients αρκεί ως event για ανανέωση των δεδομένων ωστόσο προσθέσαμε ένα procedure που ανανεώνει όλες τις συνταγές ανεξαρτήτως event για , δημιουργία τυχαίων επεισοδίων, και δημιουργία τυχαίων βαθμών ως εξής:

```
-- procedure to be used by triggers

DROP PROCEDURE IF EXISTS UpdateRecipeNutritionalValues;


CREATE PROCEDURE UpdateRecipeNutritionalValues(IN recipe_id INT UNSIGNED)

BEGIN

    -- Update the total nutritional values for the given recipe

    UPDATE recipes r

    JOIN (

        SELECT

            rhi.recipes_recipe_id,

            SUM(rhi.quantity * i.carbs) AS total_carbs,

            SUM(rhi.quantity * i.fat) AS total_fat,

            SUM(rhi.quantity * i.protein) AS total_protein,

            SUM(rhi.quantity * i.calories) AS total_calories

        FROM

            recipes_has_ingredients rhi

            JOIN ingredients i ON rhi.ingredients_ingredient_id =
i.ingredient_id

        WHERE

            rhi.recipes_recipe_id = recipe_id

        GROUP BY

            rhi.recipes_recipe_id
    ) AS nutritional_sums ON r.recipe_id =
nutritional_sums.recipes_recipe_id

    SET

        r.total_carbs = nutritional_sums.total_carbs,

        r.total_fat = nutritional_sums.total_fat,

        r.total_protein = nutritional_sums.total_protein,

        r.total_calories = nutritional_sums.total_calories;

END;


-- setting up triggers


DROP TRIGGER IF EXISTS after_insert_recipes_has_ingredients;


CREATE TRIGGER after_insert_recipes_has_ingredients

    AFTER INSERT ON recipes_has_ingredients

    FOR EACH ROW

    BEGIN

        CALL UpdateRecipeNutritionalValues(NEW.recipes_recipe_id);

    END;



DROP TRIGGER IF EXISTS after_update_recipes_has_ingredients;


CREATE TRIGGER after_update_recipes_has_ingredients

    AFTER UPDATE ON recipes_has_ingredients

    FOR EACH ROW

    BEGIN

        CALL UpdateRecipeNutritionalValues(NEW.recipes_recipe_id);

    END;


DROP TRIGGER IF EXISTS after_delete_recipes_has_ingredients;
```

```
CREATE TRIGGER after_delete_recipes_has_ingredients
    AFTER DELETE ON recipes_has_ingredients
    FOR EACH ROW
    BEGIN
        CALL OverallUpdateRecipeNutritionalValues(OLD.recipes_recipe_id);
    END;



-- procedure in case of need of overall computation
DROP PROCEDURE IF EXISTS OverallUpdateRecipeNutritionalValues;

CREATE PROCEDURE OverallUpdateRecipeNutritionalValues()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE recipe_id_var INT UNSIGNED;

    -- Declare a cursor to iterate over all recipes
    DECLARE recipe_cursor CURSOR FOR
        SELECT recipe_id FROM recipes;

    -- Declare a NOT FOUND handler for the cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Open the cursor
    OPEN recipe_cursor;

    -- Loop through all recipes
    read_loop: LOOP
        FETCH recipe_cursor INTO recipe_id_var;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Update the total nutritional values for each recipe
        UPDATE recipes r
        JOIN (
            SELECT
                rhi.recipes_recipe_id,
                SUM(rhi.quantity * i.carbs) AS total_carbs,
                SUM(rhi.quantity * i.fat) AS total_fat,
                SUM(rhi.quantity * i.protein) AS total_protein,
                SUM(rhi.quantity * i.calories) AS total_calories
            FROM
                recipes_has_ingredients rhi
                JOIN ingredients i ON rhi.ingredients_ingredient_id =
i.ingredient_id
            WHERE
                rhi.recipes_recipe_id = recipe_id_var
            GROUP BY
                rhi.recipes_recipe_id
        ) AS nutritional_sums ON r.recipe_id =
nutritional_sums.recipes_recipe_id
        SET
            r.total_carbs = nutritional_sums.total_carbs,
            r.total_fat = nutritional_sums.total_fat,
            r.total_protein = nutritional_sums.total_protein,
            r.total_calories = nutritional_sums.total_calories;
    END LOOP;

    -- Close the cursor
    CLOSE recipe_cursor;
END;




DROP PROCEDURE IF EXISTS GenerateAnnualCompetition;


CREATE PROCEDURE GenerateAnnualCompetition()
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE j INT DEFAULT 1;
    DECLARE rand_cuisine INT;
    DECLARE rand_cook INT;
    DECLARE rand_recipe INT;
    DECLARE rand_judge1 INT;
    DECLARE rand_judge2 INT;
    DECLARE rand_judge3 INT;
    DECLARE rejected BOOLEAN;

    DECLARE curr_season INT DEFAULT 0;

    -- Get the current season and increment it
    SELECT current_season INTO curr_season FROM current_season ORDER BY
season_id DESC LIMIT 1;
    SET curr_season = curr_season + 1;
    INSERT INTO current_season (current_season) VALUES (curr_season);

    CREATE TEMPORARY TABLE IF NOT EXISTS selected_cuisines (cuisine_id
INT);

    -- Loop through 10 episodes
    WHILE i <= 10 DO
```

```
        -- Select 3 unique judges for the episode
        REPEAT
            SET rand_judge1 = (SELECT cook_id FROM cooks ORDER BY RAND()
LIMIT 1);
            SET rand_judge2 = (SELECT cook_id FROM cooks WHERE cook_id NOT
IN (rand_judge1) ORDER BY RAND() LIMIT 1);
            SET rand_judge3 = (SELECT cook_id FROM cooks WHERE cook_id NOT
IN (rand_judge1, rand_judge2) ORDER BY RAND() LIMIT 1);
        UNTIL NOT EXISTS (
            SELECT 1 FROM episodes e
            WHERE e.episode = i - 1
            AND (e.judge1_id IN (rand_judge1, rand_judge2, rand_judge3)
                OR e.judge2_id IN (rand_judge1, rand_judge2, rand_judge3)
                OR e.judge3_id IN (rand_judge1, rand_judge2, rand_judge3))
        )
        END REPEAT;


        -- Insert episode details
        INSERT INTO episodes (episode_season, episode, judge1_id,
judge2_id, judge3_id) VALUES (curr_season, i, rand_judge1, rand_judge2,
rand_judge3);
        SET @episode_id = LAST_INSERT_ID();
        TRUNCATE TABLE selected_cuisines;
        SET j = 1;
        WHILE j <= 10 DO
            REPEAT
                SET rejected = FALSE;
                -- Select random recipe
                SET rand_recipe = (SELECT recipe_id FROM recipes ORDER BY
RAND() LIMIT 1);
                SET rand_cuisine = (SELECT
national_cuisine_national_cuisine_id FROM recipes WHERE recipe_id =
rand_recipe);
```

```
                -- Ensure cuisine is not in current episode
                IF EXISTS (SELECT 1 FROM selected_cuisines WHERE
cuisine_id = rand_cuisine) THEN
                    SET rejected = TRUE;
                END IF;
            UNTIL rejected = FALSE
            END REPEAT;
            -- Insert the selected cuisine into the temporary table
            INSERT INTO selected_cuisines (cuisine_id) VALUES
(rand_cuisine);


            -- Select 1 random recipe from the selected national cuisine
and associated cook
            REPEAT
                SET rejected = FALSE;


                -- Select random recipe
                SET rand_recipe = (SELECT recipe_id FROM recipes WHERE
national_cuisine_national_cuisine_id = rand_cuisine ORDER BY RAND() LIMIT
1);
                -- Select random cook associated with the selected recipe
                SET rand_cook = (SELECT cook_id FROM cooks WHERE cook_id
IN (SELECT cooks_cook_id FROM recipes_has_cooks WHERE recipes_recipe_id =
rand_recipe) ORDER BY RAND() LIMIT 1);


            UNTIL rejected = FALSE
            END REPEAT;
            -- Insert cook, recipe, and episode relationship
            INSERT INTO cook_has_recipe_in_episodes (episodes_episode_id,
recipes_has_cooks_recipes_recipe_id, recipes_has_cooks_cooks_cook_id)
            VALUES (@episode_id, rand_recipe, rand_cook);
            SET j = j + 1;
        END WHILE;
        SET i = i + 1;
    END WHILE;
END;



DROP PROCEDURE IF EXISTS UpdateGrades;
CREATE PROCEDURE UpdateGrades()
BEGIN
    -- Update grades for all entries in cook_has_recipe_in_episodes
    UPDATE test.cook_has_recipe_in_episodes
    SET
        grade1 = FLOOR(1 + RAND() * 5),
        grade2 = FLOOR(1 + RAND() * 5),
        grade3 = FLOOR(1 + RAND() * 5);
END
```

DML

Φορτώνουμε δεδομένα από csv αρχεία με τον εξής τρόπο:

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/table_name.csv'
INTO TABLE database_name.ingredients
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(column1,column2…);
```

```
-- DML LOADING DATA FROM CUSTOM MADE CSV FILES

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/ingredient_groups.csv'

INTO TABLE test.ingredient_groups

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(name, description);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/ingredients.csv'

INTO TABLE test.ingredients

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS
```

```
(fat, carbs, protein, calories,
name,ingredient_groups_ingredient_group_id);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/meal_type.csv'

INTO TABLE test.meal_type

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(name);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/national_cuisine.csv'

INTO TABLE test.national_cuisine

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

( name);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/equipment.csv'

INTO TABLE test.equipment

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(name, instructions);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/steps.csv'

INTO TABLE test.steps

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(step_description);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/tags.csv'

INTO TABLE test.tags

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(name);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/themes.csv'

INTO TABLE test.themes

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

( name, description);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/tips.csv'

INTO TABLE test.tips
```

```
FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(description);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes.csv'

INTO TABLE `test`.`recipes`

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 LINES

(type, difficulty, name, description, prep_time, cooking_time, portions,
ingredients_ingredient_id, national_cuisine_national_cuisine_id)

SET

  total_fat = NULL,

  total_carbs = NULL,

  total_protein = NULL,

  total_calories = NULL;



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/cooks.csv'

INTO TABLE test.cooks

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(first_name, last_name, phone_number, date_of_birth, age, role,
years_of_experience);
```

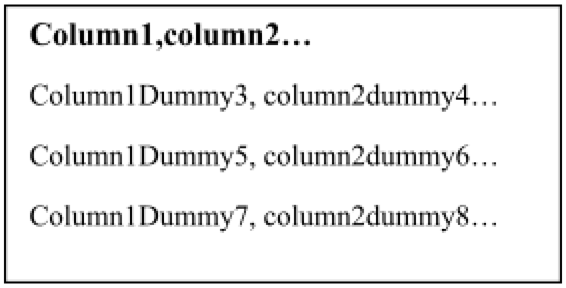
```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_steps.csv'

INTO TABLE test.recipes_has_steps

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, steps_step_id, `order`);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_meal_type.csv'

INTO TABLE test.recipes_has_meal_type

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, meal_type_meal_type_id);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_tags.csv'

INTO TABLE test.recipes_has_tags

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, tags_tag_id);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_tips.csv'
```

```
INTO TABLE test.recipes_has_tips

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, tips_tip_id1, tips_tip_id2, tips_tip_id3);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_ingredients.csv'

INTO TABLE test.recipes_has_ingredients

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, ingredients_ingredient_id, quantity);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_equipment.csv'

INTO TABLE test.recipes_has_equipment

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id,equipment_equipment_id);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_themes.csv'

INTO TABLE test.recipes_has_themes

FIELDS TERMINATED BY ','
```

```
ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, themes_theme_id);



INSERT INTO current_season (current_season) VALUES (1);



LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/recipes_has_cooks.csv'

INTO TABLE test.recipes_has_cooks

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS

(recipes_recipe_id, cooks_cook_id);
```

FAKE DATA

Table_name.csv:



```
Column1,column2…
Column1Dummy3, column2dummy4…
Column1Dummy5, column2dummy6…
Column1Dummy7, column2dummy8…
```

Overall:

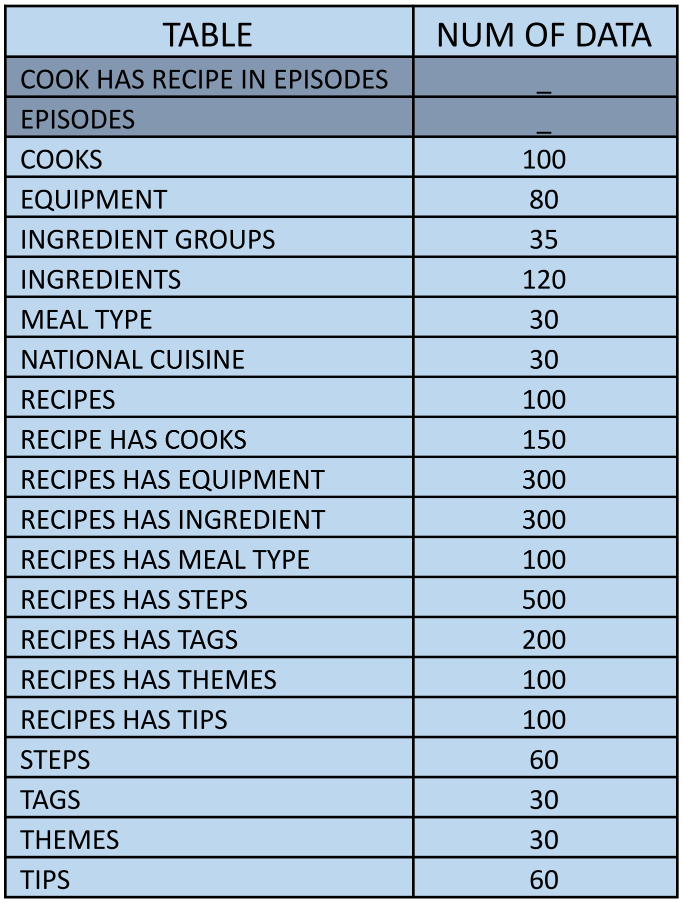


TABLE	NUM OF DATA
COOK HAS RECIPE IN EPISODES	_
EPISODES	_
COOKS	100
EQUIPMENT	80
INGREDIENT GROUPS	35
INGREDIENTS	120
MEAL TYPE	30
NATIONAL CUISINE	30
RECIPES	100
RECIPE HAS COOKS	150
RECIPES HAS EQUIPMENT	300
RECIPES HAS INGREDIENT	300
RECIPES HAS MEAL TYPE	100
RECIPES HAS STEPS	500
RECIPES HAS TAGS	200
RECIPES HAS THEMES	100
RECIPES HAS TIPS	100
STEPS	60
TAGS	30
THEMES	30
TIPS	60

Queries

```
--WINNER--


SELECT

    cooks.cook_id,

    cooks.first_name,

    cooks.last_name,

    cooks.role,

    SUM(coalesce(grade1, 0) + coalesce(grade2, 0) + coalesce(grade3, 0))
AS total_grades

FROM

    cook_has_recipe_in_episodes cre

    JOIN episodes e ON cre.episodes_episode_id = e.episode_id

    JOIN cooks ON cre.recipes_has_cooks_cooks_cook_id = cooks.cook_id

WHERE

    e.episode_season = 1

GROUP BY

    cooks.cook_id

ORDER BY

    total_grades DESC,

    FIELD(cooks.role, 'CHEF', 'SOUS_CHEF', 'A', 'B', 'C'),

    cooks.cook_id

LIMIT 1;



--1--
```

```
-- Calculate the average score per cook and national cuisine
SELECT
    c.cook_id,
    CONCAT(c.first_name, ' ', c.last_name) AS cook_name,
    nc.name AS national_cuisine_name,
    AVG(COALESCE(cre.grade1, 0) + COALESCE(cre.grade2, 0) +
COALESCE(cre.grade3, 0)) / 3 AS average_score
FROM
    cooks c
JOIN
    recipes_has_cooks rc ON c.cook_id = rc.cooks_cook_id
JOIN
    recipes r ON r.recipe_id = rc.recipes_recipe_id
JOIN
    national_cuisine nc ON r.national_cuisine_national_cuisine_id =
nc.national_cuisine_id
JOIN
    cook_has_recipe_in_episodes cre ON
cre.recipes_has_cooks_recipes_recipe_id = r.recipe_id
    AND cre.recipes_has_cooks_cooks_cook_id = rc.cooks_cook_id
GROUP BY
    c.cook_id, nc.name
ORDER BY
    cook_name, national_cuisine_name;



--2--


-- Given National Cuisine ID
SET @national_cuisine_id = 1; -- Replace with the actual national cuisine
ID
```

```
-- Given Season

SET @season = 1; -- Replace with the actual season (assuming season
represents the year or can be used as a proxy)


-- Find cooks belonging to the given national cuisine and participated in
episodes in the given season

SELECT DISTINCT c.cook_id, c.first_name, c.last_name, nc.name AS
national_cuisine, e.episode_season, e.episode

FROM cooks c

INNER JOIN recipes_has_cooks rhc ON c.cook_id = rhc.cooks_cook_id

INNER JOIN recipes r ON rhc.recipes_recipe_id = r.recipe_id

INNER JOIN national_cuisine nc ON r.national_cuisine_national_cuisine_id =
nc.national_cuisine_id

LEFT JOIN cook_has_recipe_in_episodes cre ON rhc.recipes_recipe_id =
cre.recipes_has_cooks_recipes_recipe_id

LEFT JOIN episodes e ON cre.episodes_episode_id = e.episode_id

WHERE nc.national_cuisine_id = @national_cuisine_id

  AND e.episode_season = @season;


--3--


SELECT

    CONCAT(c.first_name, ' ', c.last_name) AS cook_name,

    c.age,

    COUNT(rhc.recipes_recipe_id) AS recipe_count

FROM

    cooks c

JOIN

    recipes_has_cooks rhc ON c.cook_id = rhc.cooks_cook_id

WHERE

    c.age < 30

GROUP BY
```

```
    c.cook_id
ORDER BY
    recipe_count DESC;

--4--



SELECT
    c.cook_id,
    c.first_name,
    c.last_name
FROM
    cooks c
LEFT JOIN
    episodes e1 ON c.cook_id = e1.judge1_id
LEFT JOIN
    episodes e2 ON c.cook_id = e2.judge2_id
LEFT JOIN
    episodes e3 ON c.cook_id = e3.judge3_id
WHERE
    e1.judge1_id IS NULL
    AND e2.judge2_id IS NULL
    AND e3.judge3_id IS NULL;

--5--


-- Given Year
SET @year = 1; -- Replace with the actual year


-- Find judges with the same number of episodes in a given year with more
than 3 appearances
```

```
WITH judge_appearances AS (
  SELECT
    judge_id,
    COUNT(*) AS num_episodes
  FROM (
    SELECT judge1_id AS judge_id
    FROM episodes
    WHERE episode_season = @year
    UNION ALL
    SELECT judge2_id AS judge_id
    FROM episodes
    WHERE episode_season = @year
    UNION ALL
    SELECT judge3_id AS judge_id
    FROM episodes
    WHERE episode_season = @year
  ) AS judges_in_episodes
  WHERE judge_id IS NOT NULL
  GROUP BY judge_id
  HAVING num_episodes > 3
),
judge_counts AS (
  SELECT
    num_episodes,
    GROUP_CONCAT(judge_id) AS judges
  FROM judge_appearances
  GROUP BY num_episodes
  HAVING COUNT(*) > 1
)
SELECT *
FROM judge_counts;
```

```
--6--

      --6.1--
WITH RecipeTagPairs AS (
    SELECT
        r1.recipes_recipe_id AS recipe1,
        r2.recipes_recipe_id AS recipe2,
        LEAST(r1.tags_tag_id, r2.tags_tag_id) AS tag1,
        GREATEST(r1.tags_tag_id, r2.tags_tag_id) AS tag2
    FROM
        recipes_has_tags r1
    JOIN
        recipes_has_tags r2 ON r1.recipes_recipe_id = r2.recipes_recipe_id
    WHERE
        r1.tags_tag_id < r2.tags_tag_id
),
CommonTagPairs AS (
    SELECT DISTINCT
        t1.recipe1,
        t2.recipe2,
        t1.tag1,
        t1.tag2
    FROM
        RecipeTagPairs t1
    JOIN
        RecipeTagPairs t2 ON t1.tag1 = t2.tag1 AND t1.tag2 = t2.tag2
    WHERE
        t1.recipe1 <> t2.recipe2
),
TagPairCounts AS (
    SELECT
        tag1,
        tag2,
        COUNT(*) AS pair_count
    FROM
        CommonTagPairs
    GROUP BY
        tag1, tag2
)
SELECT
    tag1,
    tag2,
    pair_count
FROM
    TagPairCounts
ORDER BY
    pair_count DESC
LIMIT 3;



      --6.2--


USE test_script;


-- Find the three most common pairs of tags that appear in at least two
different recipes
SELECT
    t1.tag_id AS tag1,
    t2.tag_id AS tag2,
    COUNT(DISTINCT rt1.recipes_recipe_id) AS recipe_count
```

```
FROM

    recipes_has_tags rt1

FORCE INDEX (PRIMARY)

JOIN

    recipes_has_tags rt2

FORCE INDEX (PRIMARY)

ON rt1.recipes_recipe_id = rt2.recipes_recipe_id AND rt1.tags_tag_id <
rt2.tags_tag_id

JOIN

    tags t1

FORCE INDEX (PRIMARY)

ON rt1.tags_tag_id = t1.tag_id

JOIN

    tags t2

FORCE INDEX (PRIMARY)

ON rt2.tags_tag_id = t2.tag_id

GROUP BY

    tag1, tag2

HAVING

    COUNT(DISTINCT rt1.recipes_recipe_id) > 1

ORDER BY

    recipe_count DESC

LIMIT 3;




--7--



-- Step 1: Determine the maximum number of episode participations by any
cook

SELECT
```

```
    COUNT(chre.recipes_has_cooks_cooks_cook_id) AS max_participations
FROM
    cook_has_recipe_in_episodes chre
GROUP BY
    chre.recipes_has_cooks_cooks_cook_id
ORDER BY
    max_participations DESC
LIMIT 1;


-- Step 2: Find all cooks who participated at least 5 times fewer than the
cook with the most participations
SELECT
    c.cook_id,
    CONCAT(c.first_name, ' ', c.last_name) AS cook_name,
    COUNT(chre.recipes_has_cooks_cooks_cook_id) AS participations
FROM
    cooks c
JOIN
    cook_has_recipe_in_episodes chre ON c.cook_id =
chre.recipes_has_cooks_cooks_cook_id
GROUP BY
    c.cook_id
HAVING
    participations <= (
        SELECT
            MAX(participations) - 5
        FROM (
            SELECT
                COUNT(chre.recipes_has_cooks_cooks_cook_id) AS
participations
            FROM
                cook_has_recipe_in_episodes chre
            GROUP BY

                chre.recipes_has_cooks_cooks_cook_id

        ) AS subquery

    );




--8--


-- Find the episode with the most equipment used

SELECT e.episode_id, COUNT(re.equipment_equipment_id) AS equipment_count

FROM episodes e

JOIN cook_has_recipe_in_episodes cre ON e.episode_id =
cre.episodes_episode_id

JOIN recipes_has_cooks rc ON rc.recipes_recipe_id =
cre.recipes_has_cooks_recipes_recipe_id AND rc.cooks_cook_id =
cre.recipes_has_cooks_cooks_cook_id

JOIN recipes_has_equipment re ON re.recipes_recipe_id =
rc.recipes_recipe_id

GROUP BY e.episode_id

ORDER BY equipment_count DESC

LIMIT 1;



--AND WITH FORCE INDEX---



-- Active: 1716624751642@@127.0.0.1@3307@test

-- Alternative query using FORCE INDEX

SELECT e.episode_id, COUNT(re.equipment_equipment_id) AS equipment_count

FROM episodes e

FORCE INDEX (PRIMARY)
```

```
JOIN cook_has_recipe_in_episodes cre FORCE INDEX (PRIMARY) ON e.episode_id
= cre.episodes_episode_id

JOIN recipes_has_cooks rc FORCE INDEX (PRIMARY) ON rc.recipes_recipe_id =
cre.recipes_has_cooks_recipes_recipe_id AND rc.cooks_cook_id =
cre.recipes_has_cooks_cooks_cook_id

JOIN recipes_has_equipment re FORCE INDEX (PRIMARY) ON
re.recipes_recipe_id = rc.recipes_recipe_id

GROUP BY e.episode_id

ORDER BY equipment_count DESC

LIMIT 1;




--9--


SELECT

    e.episode_season AS season,

    AVG(r.total_carbs) AS average_total_carbs

FROM

    episodes e

JOIN

    cook_has_recipe_in_episodes chre ON e.episode_id =
chre.episodes_episode_id

JOIN

    recipes_has_cooks rhc ON chre.recipes_has_cooks_recipes_recipe_id =
rhc.recipes_recipe_id

JOIN

    recipes r ON rhc.recipes_recipe_id = r.recipe_id

GROUP BY

    e.episode_season

ORDER BY

    e.episode_season;
```

```
--10--

WITH ParticipationCount AS (
    SELECT
        nc.name AS cuisine_name,
        e.episode_season AS season,
        COUNT(*) AS participations
    FROM
        cook_has_recipe_in_episodes cre
    JOIN
        recipes r ON cre.recipes_has_cooks_recipes_recipe_id = r.recipe_id
    JOIN
        national_cuisine nc ON r.national_cuisine_national_cuisine_id =
nc.national_cuisine_id
    JOIN
        episodes e ON cre.episodes_episode_id = e.episode_id
    GROUP BY
        nc.name, e.episode_season
    HAVING
        participations >= 3
),
ConsecutiveSeasonCounts AS (
    SELECT
        pc1.cuisine_name,
        pc1.season AS season1,
        pc1.participations AS participations1,
        pc2.season AS season2,
        pc2.participations AS participations2
    FROM
        ParticipationCount pc1
    JOIN
```

```
        ParticipationCount pc2 ON pc1.cuisine_name = pc2.cuisine_name
    WHERE
        pc1.season = pc2.season - 1
        AND pc1.participations = pc2.participations
)
SELECT
    cuisine_name,
    season1,
    participations1 AS participations,
    season2
FROM
    ConsecutiveSeasonCounts;




--11--

SELECT
    CONCAT(j1.first_name, ' ', j1.last_name) AS judge_name,
    CONCAT(cook.first_name, ' ', cook.last_name) AS cook_name,
    SUM(
        CASE
            WHEN e.judge1_id = j1.cook_id THEN chre.grade1
            WHEN e.judge2_id = j1.cook_id THEN chre.grade2
            WHEN e.judge3_id = j1.cook_id THEN chre.grade3
            ELSE 0
        END
    ) AS total_score
FROM
    cook_has_recipe_in_episodes chre
```

```
JOIN

    episodes e ON chre.episodes_episode_id = e.episode_id

JOIN

    cooks cook ON chre.recipes_has_cooks_cooks_cook_id = cook.cook_id

JOIN

    cooks j1 ON e.judge1_id = j1.cook_id OR e.judge2_id = j1.cook_id OR
e.judge3_id = j1.cook_id

GROUP BY

    j1.cook_id, cook.cook_id

ORDER BY

    total_score DESC

LIMIT 5;



--12--


SELECT

    e.episode_season AS season,

    e.episode_id,

    e.episode,

    AVG(

        CASE r.difficulty

            WHEN 'VERY_EASY' THEN 1

            WHEN 'EASY' THEN 2

            WHEN 'NORMAL' THEN 3

            WHEN 'DIFFICULT' THEN 4

            WHEN 'VERY_DIFFICULT' THEN 5

        END

    ) AS average_difficulty

FROM

    episodes e
```

```
JOIN

    cook_has_recipe_in_episodes chre ON e.episode_id =
chre.episodes_episode_id

JOIN

    recipes_has_cooks rhc ON chre.recipes_has_cooks_recipes_recipe_id =
rhc.recipes_recipe_id AND chre.recipes_has_cooks_cooks_cook_id =
rhc.cooks_cook_id

JOIN

    recipes r ON rhc.recipes_recipe_id = r.recipe_id

GROUP BY

    e.episode_season, e.episode_id

ORDER BY

    e.episode_season, average_difficulty DESC;


--13--



SET @A = 1;

SET @B = 2;

SET @C = 3;

SET @SOUS_CHEF = 4;

SET @CHEF = 5;


-- Create a subquery to calculate the professional training level for each
judge and cook

WITH training_levels AS (

  SELECT

    e.episode_id,

    COALESCE(NULLIF(c1.role, ''), 'C') AS judge1_role,

    COALESCE(NULLIF(c2.role, ''), 'C') AS judge2_role,

    COALESCE(NULLIF(c3.role, ''), 'C') AS judge3_role,

    COALESCE(NULLIF(c.role, ''), 'C') AS cook_role

  FROM
```

```
    episodes e
  LEFT JOIN cooks c1 ON e.judge1_id = c1.cook_id
  LEFT JOIN cooks c2 ON e.judge2_id = c2.cook_id
  LEFT JOIN cooks c3 ON e.judge3_id = c3.cook_id
  JOIN cook_has_recipe_in_episodes cri ON e.episode_id =
cri.episodes_episode_id
  JOIN cooks c ON cri.recipes_has_cooks_cooks_cook_id = c.cook_id
)
-- Calculate the average training level for each episode
SELECT
  episode_id,
  (
    IF(judge1_role = 'A', @A, IF(judge1_role = 'B', @B, IF(judge1_role =
'C', @C, IF(judge1_role = 'SOUS_CHEF', @SOUS_CHEF, @CHEF)))) +
    IF(judge2_role = 'A', @A, IF(judge2_role = 'B', @B, IF(judge2_role =
'C', @C, IF(judge2_role = 'SOUS_CHEF', @SOUS_CHEF, @CHEF)))) +
    IF(judge3_role = 'A', @A, IF(judge3_role = 'B', @B, IF(judge3_role =
'C', @C, IF(judge3_role = 'SOUS_CHEF', @SOUS_CHEF, @CHEF)))) +
    IF(cook_role = 'A', @A, IF(cook_role = 'B', @B, IF(cook_role = 'C',
@C, IF(cook_role = 'SOUS_CHEF', @SOUS_CHEF, @CHEF))))
  ) / 4 AS avg_training_level
FROM
  training_levels
ORDER BY
  avg_training_level ASC
LIMIT 1;



--14--


SELECT th.name AS theme_name, COUNT(*) AS appearance_count
FROM themes th
```

```
JOIN recipes_has_themes rht ON th.theme_id = rht.themes_theme_id

GROUP BY th.theme_id

ORDER BY appearance_count DESC

LIMIT 1;



--15--


SELECT ig.name AS ingredient_group_name

FROM ingredient_groups ig

LEFT JOIN ingredients i ON ig.ingredient_group_id =
i.ingredient_groups_ingredient_group_id

LEFT JOIN recipes_has_ingredients rhi ON i.ingredient_id =
rhi.ingredients_ingredient_id

WHERE rhi.ingredients_ingredient_id IS NULL;
```

Οδηγίες εγκατάστασης

STEPS

1)Download MySQL Installer 8.0.37 for Microsoft Windows.

2)Download and Install Python 3.11.2, ensure you check the option to add PYTHON to your PATH.

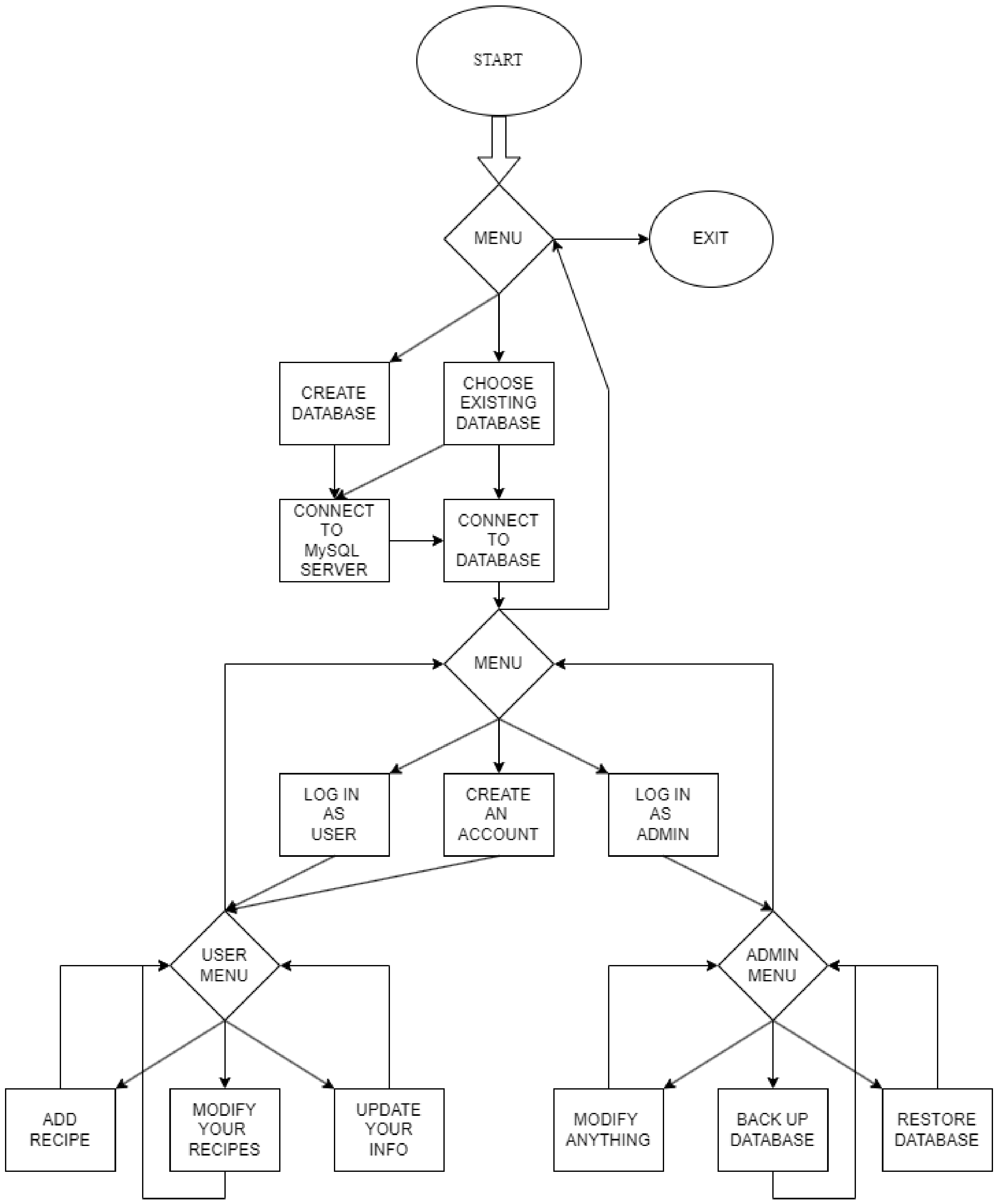
3)Download and install Visual Studio Code. Also install the Python extension by Microsoft.

4)In your VS Code, select the Python Interpreter.

5)Install with pip, necessary libraries: pip install -r requirements.txt.

```
requirements.txt{
      mysql-connector-python
      bcrypt
}
```

Οδηγίες χρήσης


START
MENU
EXIT
CREATE DATABASE
CHOOSE EXISTING DATABASE
CONNECT TO MySQL SERVER
CONNECT TO DATABASE
MENU
LOG IN AS USER
CREATE AN ACCOUNT
LOG IN AS ADMIN
USER MENU
ADMIN MENU
ADD RECIPE
MODIFY YOUR RECIPES
UPDATE YOUR INFO
MODIFY ANYTHING
BACK UP DATABASE
RESTORE DATABASE