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

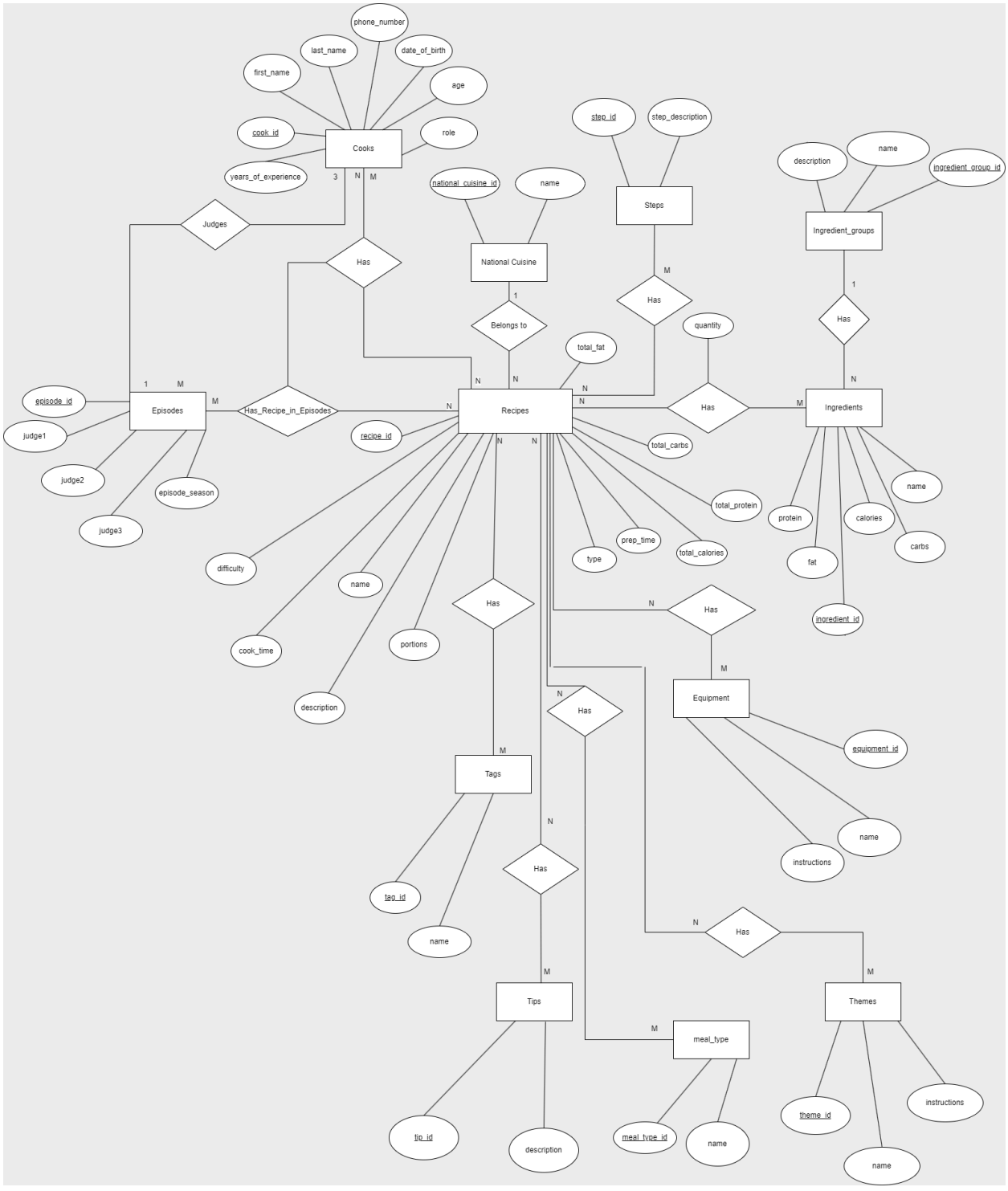
Μπεληγιάννης Νικόλαος– 03121878

Γκιώκας Νικόλαος– 03121156

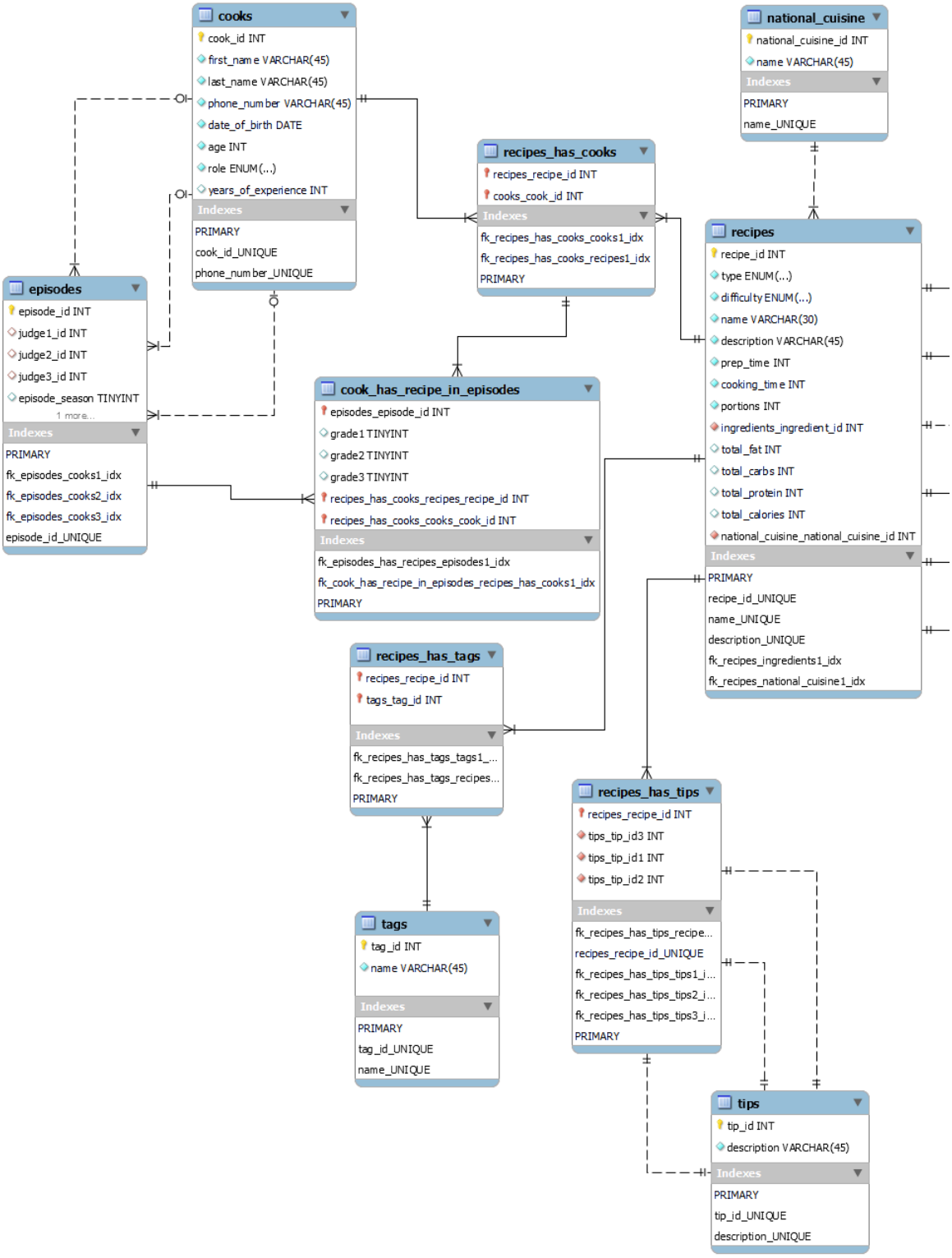
ΠΕΡΙΕΧΟΜΕΝΑ

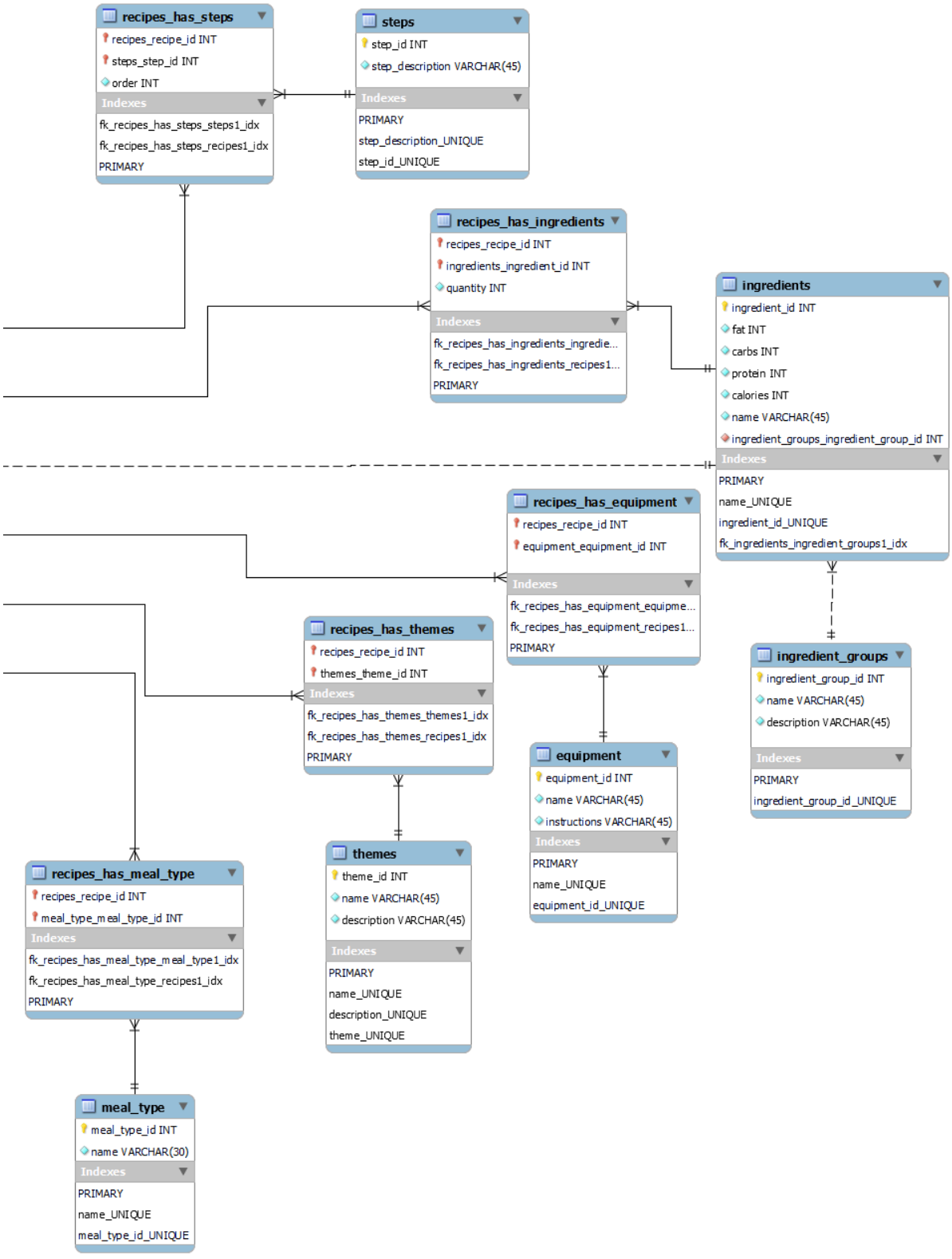
Σχεδίαση

ER

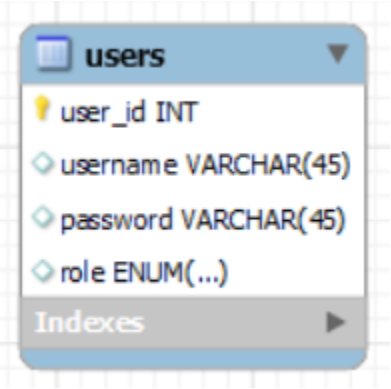


RELATIONAL SCHEMA


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
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
tags
tag_id INT
name VARCHAR(45)
Indexes
PRIMARY
tag_id_UNIQUE
name_UNIQUE
tips
tip_id INT
description VARCHAR(45)
Indexes
PRIMARY
tip_id_UNIQUE
description_UNIQUE

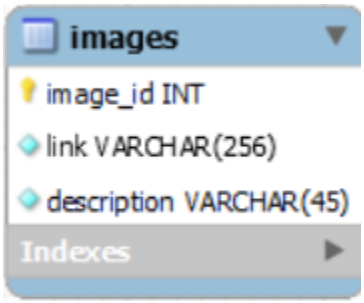

recipes_has_steps
recipes_recipe_id INT
steps_step_id INT
order INT
Indexes
fk_recipes_has_steps_steps1_idx
fk_recipes_has_steps_recipes1_idx
PRIMARY
steps
step_id INT
step_description VARCHAR(45)
Indexes
PRIMARY
step_description_UNIQUE
step_id_UNIQUE
recipes_has_ingredients
recipes_recipe_id INT
ingredients_ingredient_id 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
themes
theme_id INT
name VARCHAR(45)
description VARCHAR(45)
Indexes
PRIMARY
name_UNIQUE
description_UNIQUE
theme_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
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 recipes_has_steps;
DROP TABLE IF EXISTS recipes_has_meal_type;
DROP TABLE IF EXISTS recipes_has_tags;
DROP TABLE IF EXISTS recipes_has_tips;
DROP TABLE IF EXISTS recipes_has_equipment;
DROP TABLE IF EXISTS recipes_has_ingredients;
DROP TABLE IF EXISTS recipes_has_themes;
DROP TABLE IF EXISTS cook_has_recipe_in_episodes;
DROP TABLE IF EXISTS recipes_has_cooks;
DROP TABLE IF EXISTS recipes;
DROP TABLE IF EXISTS ingredients;
DROP TABLE IF EXISTS ingredient_groups;
DROP TABLE IF EXISTS national_cuisine;
DROP TABLE IF EXISTS tips;
DROP TABLE IF EXISTS meal_type;
DROP TABLE IF EXISTS equipment;
DROP TABLE IF EXISTS episodes;
DROP TABLE IF EXISTS cooks ;
DROP TABLE IF EXISTS tags;
DROP TABLE IF EXISTS steps;
DROP TABLE IF EXISTS themes;
DROP TABLE IF EXISTS images;

DROP TABLE IF EXISTS users ;
```

```
CREATE TABLE IF NOT EXISTS images (
  image_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  link VARCHAR(256) NOT NULL,
  description VARCHAR(46) NOT NULL,
  PRIMARY KEY (image_id) )
ENGINE = InnoDB;


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 ASC) VISIBLE)
ENGINE = InnoDB;




-- 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,
  user_id INT UNSIGNED NULL,
  PRIMARY KEY (cook_id),
  CONSTRAINT fk_cooks_users1
    FOREIGN KEY (user_id)
    REFERENCES users (user_id)
    ON DELETE SET NULL
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;

CREATE UNIQUE INDEX cook_id_UNIQUE ON cooks (cook_id);

CREATE UNIQUE INDEX phone_number_UNIQUE ON cooks (phone_number);



-- 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);




-- 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
    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);




-- 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);




CREATE TABLE IF NOT EXISTS current_season (

    season_id INT UNSIGNED NOT NULL AUTO_INCREMENT,

    current_season INT UNSIGNED NOT NULL,

    PRIMARY KEY (season_id)

);
```

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);


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);


INSERT INTO current_season (current_season) VALUES (0);
```

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

UI/AUTHENTICATION

```
import mysql.connector
import bcrypt
import getpass
import os

# Utility functions for hashing and verifying passwords
def hash_password(password):
    salt = bcrypt.gensalt()
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), salt)
    return hashed_password.decode('utf-8')

def verify_password(stored_hash, provided_password):
    return bcrypt.checkpw(provided_password.encode('utf-8'),
stored_hash.encode('utf-8'))

# Connect to MySQL server (not a specific database)
def connect_server():
    host = input("Enter MySQL server host (e.g., '127.0.0.1'): ")
    port = input("Enter MySQL server port (e.g., '3307'): ")
    user = input("Enter MySQL server user (e.g., 'root'): ")
    password = getpass.getpass("Enter MySQL server password: ")

    try:
        conn = mysql.connector.connect(
            host=host,
            port=port,
            user=user,
```

```
            password=password
        )
        return conn, host, port, user, password
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        exit(1)

# Connect to a specific database
def connect_db(host, port, user, password, database):
    try:
        return mysql.connector.connect(
            host=host,
            port=port,
            user=user,
            password=password,
            database=database
        )
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        exit(1)

# Function to execute DDL and DML from files
def execute_sql_file(cursor, sql_file):
    with open(sql_file, 'r') as file:
        sql_script = file.read()

    # Split the script into individual statements
    statements = sql_script.split(';')
    for statement in statements:
        if statement.strip():
            try:
```

```
                cursor.execute(statement.strip() + ';')
            except mysql.connector.Error as err:
                print(f"Error executing statement: {statement.strip()};")
                print(f"MySQL Error: {err}")
                break

# Create a new database and execute DDL and DML files
def create_database(conn, host, port, user, password):
    cursor = conn.cursor()
    database_name = input("Enter the name of the new database: ")

    try:
        cursor.execute(f"CREATE DATABASE {database_name}")
        print(f"Database '{database_name}' created successfully!")
    except mysql.connector.Error as err:
        print(f"Failed creating database: {err}")
        exit(1)

    conn.close()

    # Connect to the new database and execute DDL and DML files
    conn = connect_db(host, port, user, password, database_name)
    cursor = conn.cursor()

    ddl_file = 'ddl.sql'  # Assuming the DDL file is named ddl.sql and is
in the same directory
    execute_sql_file(cursor, ddl_file)



    dml_file = 'dml.sql'  # Assuming the DML file is named dml.sql and is
in the same directory
    execute_sql_file(cursor, dml_file)
```

```
    conn.commit()
    print("Tables created and initial data inserted successfully!")

    # Prompt for admin account details
    admin_username = input("Enter admin username: ")
    admin_password = getpass.getpass("Enter admin password: ")
    hashed_password = hash_password(admin_password)

    cursor.execute("INSERT INTO users (username, password, role) VALUES
(%s, %s, 'admin')",
                   (admin_username, hashed_password))
    conn.commit()

    print("Admin account created successfully!")

    cursor.close()
    conn.close()
    return database_name

# User authentication
def authenticate_user(host, port, user, password, database, username,
user_password):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor(dictionary=True)

    cursor.execute("SELECT user_id, password, role FROM users WHERE
username = %s", (username,))
    user = cursor.fetchone()
    cursor.close()
    conn.close()

    if user and verify_password(user['password'], user_password):
```

```
        return user['user_id'], user['role']
    return None, None

# Check if username exists
def check_username_exists(host, port, user, password, database, username):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor()

    cursor.execute("SELECT username FROM users WHERE username = %s",
(username,))
    user = cursor.fetchone()
    cursor.close()
    conn.close()

    return user is not None

# Create a new cook account
def create_cook_account(host, port, user, password, database,
phone_number, username, user_password):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor(dictionary=True)

    cursor.execute("SELECT cook_id FROM cooks WHERE phone_number = %s",
(phone_number,))
    cook = cursor.fetchone()

    if not cook:
        cursor.close()
        conn.close()
        return False, "Phone number not found."

    cook_id = cook['cook_id']
```

```
    hashed_password = hash_password(user_password)


    try:
        cursor.execute("INSERT INTO users (username, password, role)
VALUES (%s, %s, 'user')", (username, hashed_password))
        conn.commit()
        user_id = cursor.lastrowid
        cursor.execute("UPDATE cooks SET user_id = %s WHERE cook_id = %s",
(user_id, cook_id))
        conn.commit()
        cursor.close()
        conn.close()
        return True, "Account created successfully"
    except mysql.connector.IntegrityError as err:
        cursor.close()
        conn.close()
        return False, f"An error occurred: {err}"


# Add a new recipe
def add_recipe(host, port, user, password, database, user_id, recipe_name,
recipe_description, recipe_type, difficulty, prep_time, cooking_time,
portions, ingredients_ingredient_id,
national_cuisine_national_cuisine_id):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor()


    cursor.execute("INSERT INTO recipes (name, description, type,
difficulty, prep_time, cooking_time, portions, ingredients_ingredient_id,
national_cuisine_national_cuisine_id) VALUES (%s, %s, %s, %s, %s, %s, %s,
%s, %s)",
                   (recipe_name, recipe_description, recipe_type,
difficulty, prep_time, cooking_time, portions, ingredients_ingredient_id,
national_cuisine_national_cuisine_id))
    recipe_id = cursor.lastrowid
```

```
    cursor.execute("INSERT INTO recipes_has_cooks (recipes_recipe_id,
cooks_cook_id) VALUES (%s, (SELECT cook_id FROM cooks WHERE user_id =
%s))", (recipe_id, user_id))

    conn.commit()

    cursor.close()

    conn.close()


# Modify an existing recipe

def modify_recipe(host, port, user, password, database, user_id, role,
recipe_id, recipe_name, recipe_description):

    conn = connect_db(host, port, user, password, database)

    cursor = conn.cursor(dictionary=True)


    try:

        # Check if the recipe belongs to the user or if the user is an
admin

        if role != 'admin':

            cursor.execute("SELECT cook_id FROM cooks WHERE user_id = %s",
(user_id,))

            cook = cursor.fetchone()


            if not cook:

                print("Cook not found.")

                cursor.close()

                conn.close()

                return


            cook_id = cook['cook_id']


            cursor.execute("SELECT cooks_cook_id FROM recipes_has_cooks
WHERE recipes_recipe_id = %s AND cooks_cook_id = %s", (recipe_id,
cook_id))

            cook_association = cursor.fetchone()
```

```
            if not cook_association:
                print("You can't do that. This recipe does not belong to
you.")
                cursor.close()
                conn.close()
                return

        cursor.execute("SELECT name, description FROM recipes WHERE
recipe_id = %s", (recipe_id,))
        recipe = cursor.fetchone()

        if not recipe:
            print("Recipe not found.")
            cursor.close()
            conn.close()
            return

        # Maintain current values if "no change" is input
        if recipe_name.lower() == "no change":
            recipe_name = recipe['name']
        if recipe_description.lower() == "no change":
            recipe_description = recipe['description']

        cursor.execute("UPDATE recipes SET name = %s, description = %s
WHERE recipe_id = %s",
                       (recipe_name, recipe_description, recipe_id))
        conn.commit()
        print("Recipe modified successfully!")

    except mysql.connector.Error as err:
        print(f"Error: {err}")
    finally:
```

```
        cursor.close()
        conn.close()


# Update cook information
def update_cook_info(host, port, user, password, database, user_id,
first_name, last_name, phone_number, date_of_birth, age, role,
years_of_experience):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor(dictionary=True)


    cursor.execute("SELECT cook_id FROM cooks WHERE user_id = %s",
(user_id,))
    cook = cursor.fetchone()
    if not cook:
        print("Cook not found.")
        cursor.close()
        conn.close()
        return


    cook_id = cook['cook_id']


    if first_name.lower() == "no change":
        cursor.execute("SELECT first_name FROM cooks WHERE cook_id = %s",
(cook_id,))
        first_name = cursor.fetchone()['first_name']
    if last_name.lower() == "no change":
        cursor.execute("SELECT last_name FROM cooks WHERE cook_id = %s",
(cook_id,))
        last_name = cursor.fetchone()['last_name']
    if phone_number.lower() == "no change":
        cursor.execute("SELECT phone_number FROM cooks WHERE cook_id =
%s", (cook_id,))
        phone_number = cursor.fetchone()['phone_number']
```

```
    if date_of_birth.lower() == "no change":
        cursor.execute("SELECT date_of_birth FROM cooks WHERE cook_id =
%s", (cook_id,))
        date_of_birth = cursor.fetchone()['date_of_birth']
    if age.lower() == "no change":
        cursor.execute("SELECT age FROM cooks WHERE cook_id = %s",
(cook_id,))
        age = cursor.fetchone()['age']
    if role.lower() == "no change":
        cursor.execute("SELECT role FROM cooks WHERE cook_id = %s",
(cook_id,))
        role = cursor.fetchone()['role']
    if years_of_experience.lower() == "no change":
        cursor.execute("SELECT years_of_experience FROM cooks WHERE
cook_id = %s", (cook_id,))
        years_of_experience = cursor.fetchone()['years_of_experience']


    cursor.execute("UPDATE cooks SET first_name = %s, last_name = %s,
phone_number = %s, date_of_birth = %s, age = %s, role = %s,
years_of_experience = %s WHERE cook_id = %s",
                   (first_name, last_name, phone_number, date_of_birth,
age, role, years_of_experience, cook_id))
    conn.commit()
    print("Cook information updated successfully!")


    cursor.close()
    conn.close()


# User Menu for Custom Queries
def execute_user_query(host, port, user, password, database, user_id,
role):
    while True:
        print("\nUser Query Menu:")
        print("1. Add a new recipe")
```

```
        print("2. Modify an existing recipe")
        print("3. Update personal information")
        print("4. Exit")

        user_query_choice = input("Enter your choice: ").strip()

        if user_query_choice == '1':
            recipe_name = input("Enter recipe name: ")
            recipe_description = input("Enter recipe description: ")
            recipe_type = input("Enter recipe type (COOKING/BAKING): ")
            difficulty = input("Enter difficulty
(VERY_EASY/EASY/NORMAL/DIFFICULT/VERY_DIFFICULT): ")
            prep_time = input("Enter preparation time (minutes): ")
            cooking_time = input("Enter cooking time (minutes): ")
            portions = input("Enter number of portions: ")
            ingredients_ingredient_id = input("Enter ingredient ID: ")
            national_cuisine_national_cuisine_id = input("Enter national
cuisine ID: ")

            try:
                prep_time = int(prep_time)
                cooking_time = int(cooking_time)
                portions = int(portions)
                ingredients_ingredient_id = int(ingredients_ingredient_id)
                national_cuisine_national_cuisine_id =
int(national_cuisine_national_cuisine_id)
            except ValueError:
                print("Error: Ensure that prep_time, cooking_time,
portions, ingredients_ingredient_id, and
national_cuisine_national_cuisine_id are integers.")
                continue
```

```
            add_recipe(host, port, user, password, database, user_id,
recipe_name, recipe_description, recipe_type, difficulty, prep_time,
cooking_time, portions, ingredients_ingredient_id,
national_cuisine_national_cuisine_id)


        elif user_query_choice == '2':

            recipe_id = input("Enter recipe ID to modify: ")

            recipe_name = input("Enter new recipe name (or 'no change' to
keep current): ")

            recipe_description = input("Enter new recipe description (or
'no change' to keep current): ")


            try:

                recipe_id = int(recipe_id)

            except ValueError:

                print("Error: Recipe ID should be an integer.")

                continue


            modify_recipe(host, port, user, password, database, user_id,
role, recipe_id, recipe_name, recipe_description)


        elif user_query_choice == '3':

            first_name = input("Enter new first name (or 'no change' to
keep current): ")

            last_name = input("Enter new last name (or 'no change' to keep
current): ")

            phone_number = input("Enter new phone number (or 'no change'
to keep current): ")

            date_of_birth = input("Enter new date of birth (YYYY-MM-DD or
'no change' to keep current): ")

            age = input("Enter new age (or 'no change' to keep current):
")

            role = input("Enter new role (A/B/C/SOUS_CHEF/CHEF or 'no
change' to keep current): ")

            years_of_experience = input("Enter new years of experience (or
'no change' to keep current): ")
```

```
            update_cook_info(host, port, user, password, database,
user_id, first_name, last_name, phone_number, date_of_birth, age, role,
years_of_experience)


        elif user_query_choice == '4':
            break


        else:
            print("Invalid choice, please try again.")


# Admin Menu for Custom Queries
def execute_admin_query(host, port, user, password, database):
    conn = connect_db(host, port, user, password, database)
    cursor = conn.cursor()
    while True:
        query = input("Enter your SQL query (or type 'exit' to log out):
").strip()
        if query.lower() == 'exit':
            break


        try:
            cursor.execute(query)
            conn.commit()
            results = cursor.fetchall()
            for row in results:
                print(row)
        except mysql.connector.Error as err:
            print(f"Error: {err}")


    cursor.close()
    conn.close()
```

```
# Backup the database
def backup_database(host, port, user, password, database):
    backup_file = input("Enter the backup file name (e.g., backup.sql): ")
    command = f"mysqldump -u {user} -p{password} -h {host} --port={port}
{database} > {backup_file}"
    os.system(command)
    print("Database backup completed successfully!")


# Restore the database
def restore_database(host, port, user, password, database):
    backup_file = input("Enter the backup file name (e.g., backup.sql): ")
    command = f"mysql -u {user} -p{password} -h {host} --port={port}
{database} < {backup_file}"
    os.system(command)
    print("Database restore completed successfully!")


# Main script logic
def main():
    print("--Welcome--")


    host = None
    port = None
    user = None
    password = None


    while True:
        print("\nMain Menu:")
        print("1. Create a new database")
        print("2. Connect to an existing database")
        print("3. Exit")
```

```
        choice = input("Enter your choice: ")


        if choice == '1':
            conn, host, port, user, password = connect_server()
            database_name = create_database(conn, host, port, user,
password)
        elif choice == '2':
            host = input("Enter MySQL server host (e.g., '127.0.0.1'): ")
            port = input("Enter MySQL server port (e.g., '3306'): ")
            user = input("Enter MySQL server user (e.g., 'root'): ")
            password = getpass.getpass("Enter MySQL server password: ")
            database_name = input("Enter the name of the existing
database: ")
            conn = connect_db(host, port, user, password, database_name)
            conn.close()

            # Prompt for admin credentials
            print("Please enter admin credentials to proceed:")
            admin_username = input("Enter admin username: ")
            admin_password = getpass.getpass("Enter admin password: ")
            admin_id, role = authenticate_user(host, port, user, password,
database_name, admin_username, admin_password)

            if admin_id and role == 'admin':
                print("Admin authenticated successfully!")
            else:
                print("Authentication failed, please try again.")
                continue

        elif choice == '3':
            print("Goodbye!")
            break
```

```
        else:
            print("Invalid choice, please try again.")
            continue

        while True:
            print("\nUser Menu:")
            print("1. Log in as user")
            print("2. Log in as admin")
            print("3. Create an account")
            print("4. Exit")

            user_choice = input("Enter your choice: ")

            if user_choice == '1':
                username = input("Enter username: ")
                user_password = getpass.getpass("Enter password: ")

                user_id, role = authenticate_user(host, port, user,
password, database_name, username, user_password)

                if user_id:
                    print(f"Welcome, {username}!")

                    while True:
                        execute_user_query(host, port, user, password,
database_name, user_id, role)
                        break
                else:
                    print("Authentication failed, please try again.")

            elif user_choice == '2':
                username = input("Enter admin username: ")
```

```
                user_password = getpass.getpass("Enter admin password: ")

                user_id, role = authenticate_user(host, port, user,
password, database_name, username, user_password)

                if user_id and role == 'admin':
                    print(f"Welcome, {username} (admin)!")

                    while True:
                        print("\nAdmin Menu:")
                        print("1. Backup database")
                        print("2. Restore database")
                        print("3. Execute custom query")
                        print("4. Log out")

                        admin_choice = input("Enter your choice: ")

                        if admin_choice == '1':
                            backup_database(host, port, user, password,
database_name)

                        elif admin_choice == '2':
                            restore_database(host, port, user, password,
database_name)

                        elif admin_choice == '3':
                            execute_admin_query(host, port, user,
password, database_name)

                        elif admin_choice == '4':
                            print("Logging out...")
                            break
```

```
                        else:
                            print("Invalid choice, please try again.")
                else:
                    print("Authentication failed or you are not an admin,
please try again.")


            elif user_choice == '3':
                while True:
                    print("--Creating a cook account--")
                    phone_number = input("Enter phone number: ")
                    username = input("Enter new username: ")
                    user_password = getpass.getpass("Enter new password:
")
                    success, message = create_cook_account(host, port,
user, password, database_name, phone_number, username, user_password)
                    print(message)
                    if success:
                        break


            elif user_choice == '4':
                print("Goodbye!")
                break


            else:
                print("Invalid choice, please try again.")


if __name__ == "__main__":
    main()
```

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.

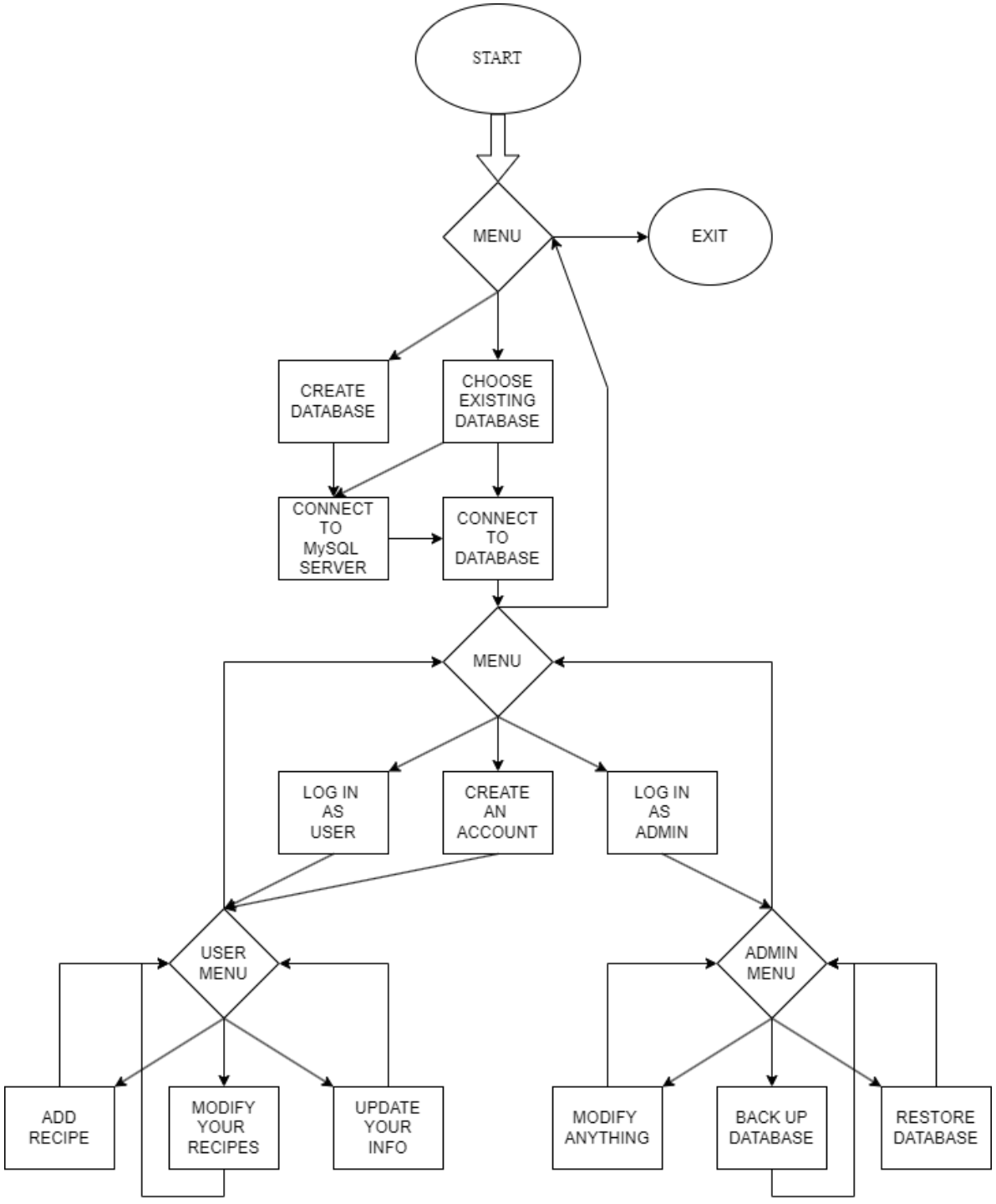
3)Download and install Visual Studio Code. Also install the Python extension by Microsoft.

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



Call OverallUpdateRecipeNutritionalValues,GenerateAnnualCompetition,UpdateGrades in this order and run the Winner Query to get the winner.