ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

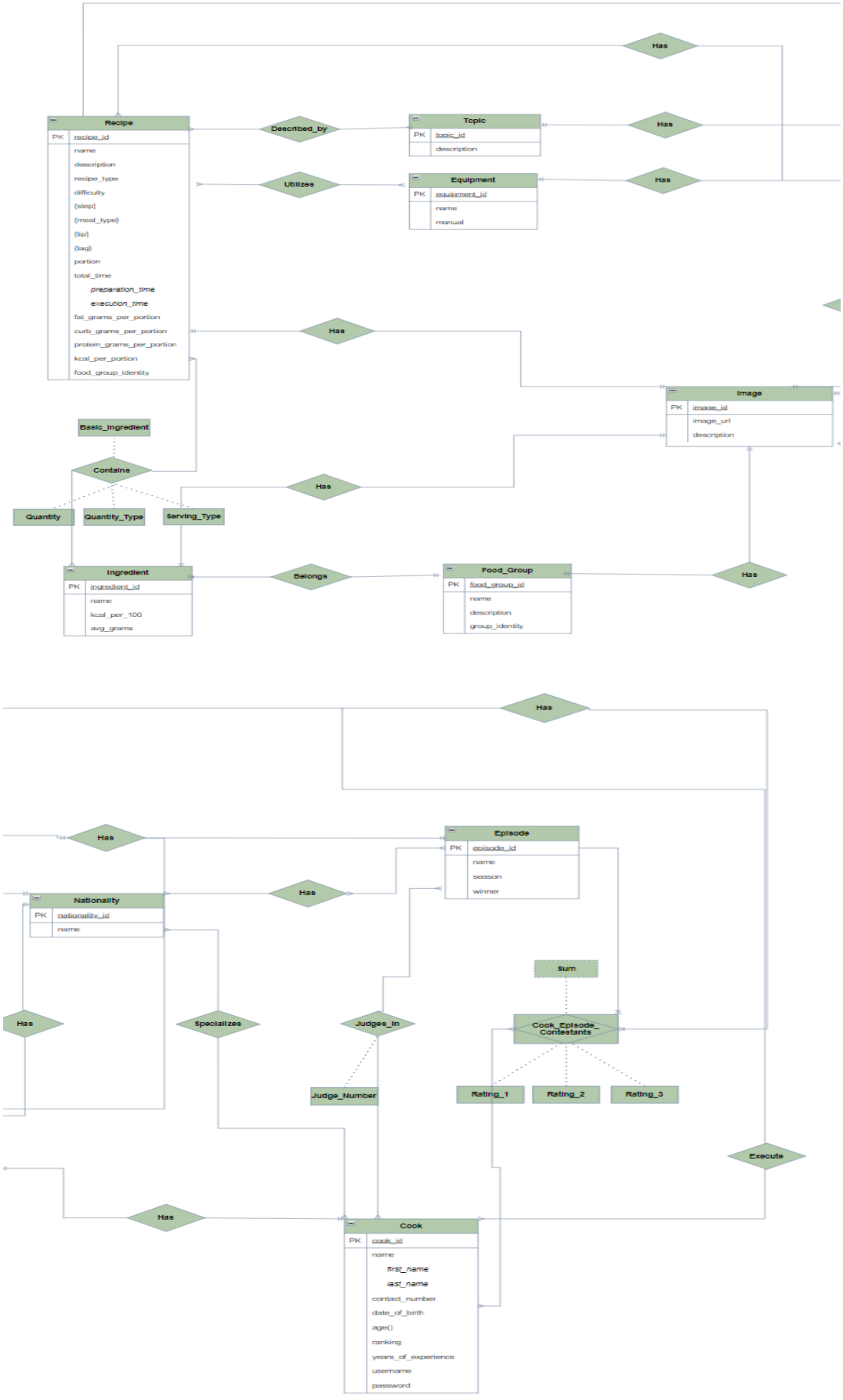
ΑΝΑΦΟΡΑ ΕΞΑΜΗΝΙΑΙΑΣ ΕΡΓΑΣΙΑΣ
ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ
2023-2024

Επιμέλεια

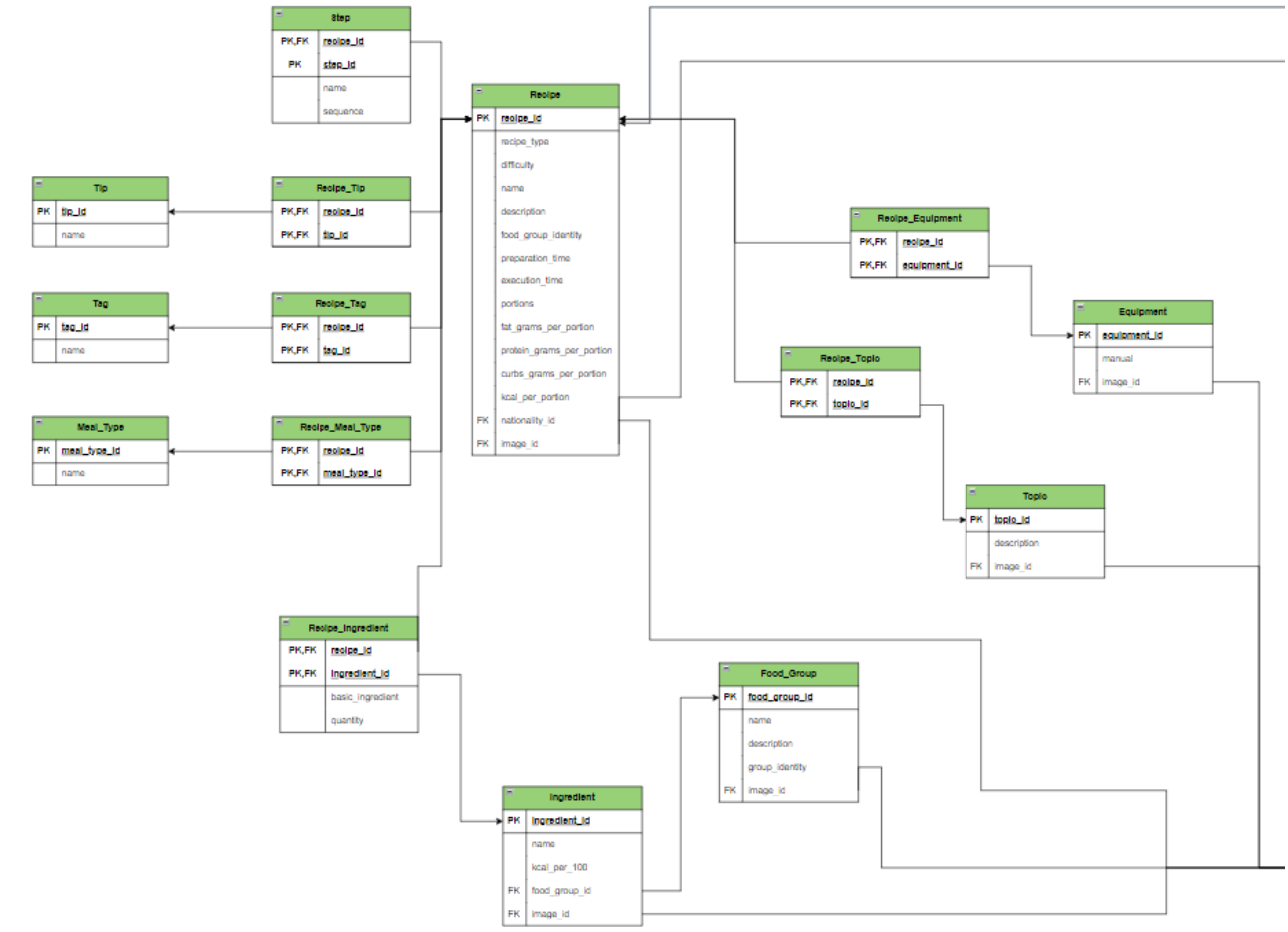
Ομάδα 121

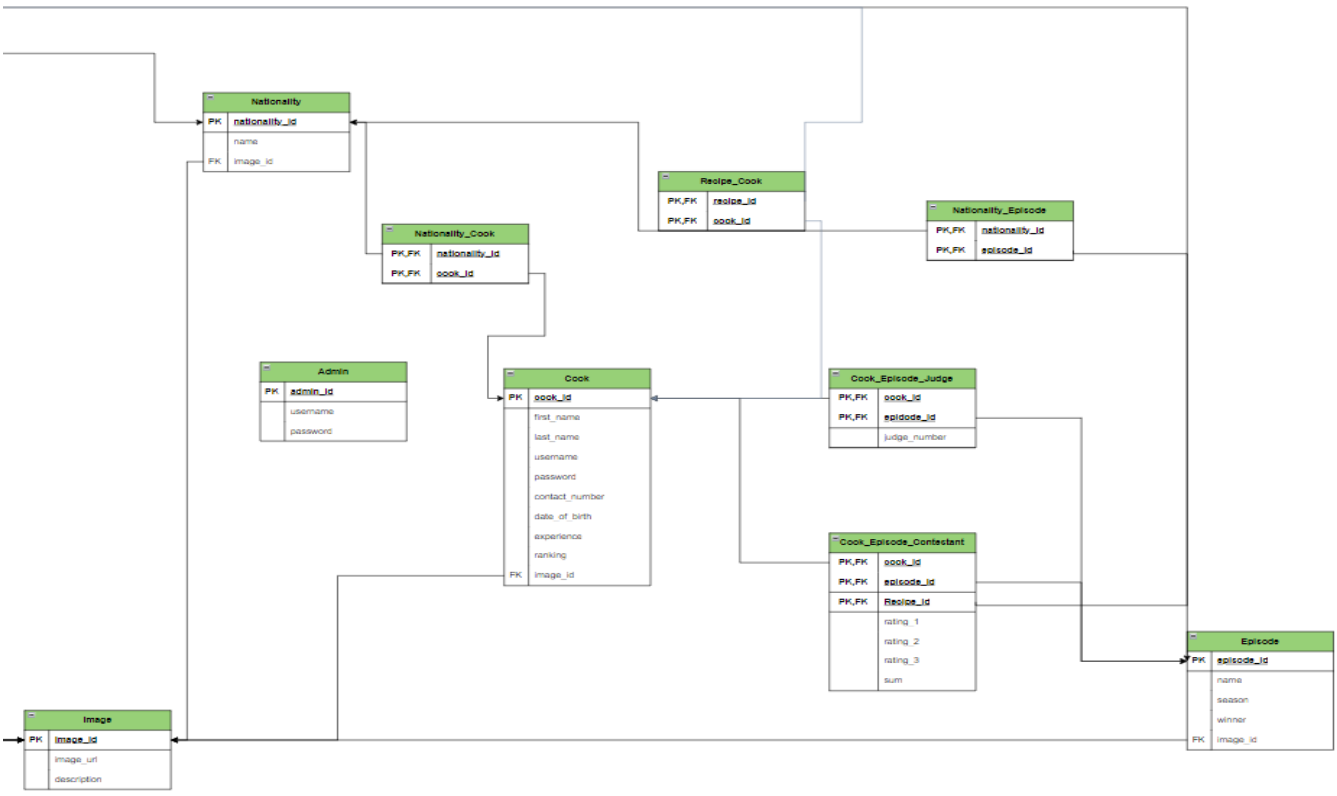
Αζάς Λεωνίδας / ge20117
Ζώρζος Αχιλλέας / ge20026
Μπόθος Βουτεράκος Νικόλαος / el20158

1. ER Diagram


Has
Recipe
PK recipe_id
name
description
recipe_type
difficulty
{step}
{meal_type}
{tip}
{tag}
portion
total_time
preparation_time
execution_time
fat_grams_per_portion
curb_grams_per_portion
protein_grams_per_portion
kcal_per_portion
food_group_identity
Described_by
Topic
PK topic_id
description
Has
Utilizes
Equipment
PK equipment_id
name
manual
Has
Has
Image
PK image_id
image_url
description
Basic_Ingredient
Contains
Has
Quantity
Quantity_Type
Serving_Type
Ingredient
PK ingredient_id
name
kcal_per_100
avg_grams
Belongs
Food_Group
PK food_group_id
name
description
group_identity
Has
Has
Episode
PK episode_id
name
season
winner
Has
Nationality
PK nationality_id
name
Has
Sum
Has
Specializes
Judges_In
Cook_Episode_Contestants
Judge_Number
Rating_1
Rating_2
Rating_3
Execute
Has
Cook
PK cook_id
name
first_name
last_name
contact_number
date_of_birth
age()
ranking
years_of_experience
username
password

2. Relational Diagram





GitHub Repository:

Το git repo του συγκεκριμένου project βρίσκεται στον παρακάτω σύνδεσμο:

https://github.com/nikolasbv/Cooking_Contest_DB.git

Ανάλυση των Διαγραμμάτων :

Η βάση δεδομένων του διαγωνισμού μαγειρικής περιλαμβάνει πολλαπλές οντότητες. Μεταξύ των κυριότερων συγκαταλέγονται οι συνταγές, οι μάγειρες και τα επεισόδια. Οι δευτερεύουσας σημασίας, αλλά ουσιώδεις για την απρόσκοπτη λειτουργία της βάσης, είναι τα συστατικά κάθε συνταγής, ο εξοπλισμός που χρησιμοποιήθηκε κατά την παρασκευή τους, οι θεματικές κατηγορίες στις οποίες ανήκουν, οι εθνικές κουζίνες που εκπροσωπούνται, οι ομάδες τροφίμων που ομαδοποιούν τα συστατικά και οι εικόνες που επιλέχθηκαν για κάθε οντότητα.

Επιπλέον, στη βάση υπάρχουν καταχωρημένοι 2 τύποι χρηστών για την εφαρμογή: Ο Διαχειριστής (Admin) και οι Μάγειρες (Users).

Επεξήγηση των οντοτήτων και των σχέσεων μεταξύ τους :

- Ο πίνακας **Recipe** περιέχει όλες τις συνταγές καταχωρημένες στη βάση και κάθε μια από αυτές συνδέεται με μια εικόνα (**Image)** μέσω του Foreign Key : image_id και με μια εθνική κουζίνα (**Nationality)** μέσω του Foreign Key : nationality_id. Κάποια ιδιαίτερα attributes που περιέχει ο πίνακας **Recipe** είναι το *recipe_type* που δέχεται δύο τύπους συνταγών (μαγειρικής, ζαχαροπλαστικής) και το *food_group_identity* που συνδυάζει κάθε συνταγή με μια ομάδα τροφίμων ανάλογα το basic_ingredient που την χαρακτηρίζει.

- Ο πίνακας **Nationality** περιέχει όλες τις εθνικές κουζίνες στη βάση, με κάθε μια από αυτές να συνδέεται με μια εικόνα (**Image)** μέσω του Foreign Key : image_id.

- Ο πίνακας **Meal_type** αποτελείται από διάφορες μορφές γεύματος (π.χ. πρωινό, πρόγευμα, γεύμα, απογευματινό, δείπνο) στις οποίες μπορεί να ανήκει κάποια συνταγή και δεν χρειάζεται Foreign Key.

- Ομοίως οι πίνακες **Tag**, **Tip** και **Step** δεν απαιτούν Foreign key.

- Ο πίνακας **Tag** αποτελείται από ετικέτες για μορφή/ τύπο γεύματος (πχ brunch, quick-lunch, κρύο πιάτο κτλ).

- Ο πίνακας **Tip** εμπεριέχει διάφορες χρηστικές συμβουλές για την πραγματοποίηση μιας συνταγής.

- Ο πίνακας **Step** περιέχει βήματα για την παροχή οδηγιών και βοήθειας στην ολοκλήρωση μιας συνταγής. Επίσης για να εκτελεστούν σειριακά υπάρχει το attribute *sequence* που τοποθετεί σε μια σειρά τα βήματα μιας συνταγής.

- Στον πίνακα **Ingredient** εισάγουμε τα υλικά τα οποία χρειάζονται οι συνταγές και κάθε ένα από αυτά συνδέεται με μια εικόνα (**Image**) μέσω του Foreign Key : image_id και με μια ομάδα τροφίμων (**Food_Group**) μέσω του Foreign key : food_group_id.

- Στον πίνακα **Equipment** αποθηκεύουμε των εξοπλισμό που χρειάζονται οι συνταγές. Κάθε εξάρτημα συνδέεται με μια εικόνα (**Image)** μέσω του Foreign Key : image_id.

- Την ίδια λογική με παραπάνω(Equipment), χρησιμοποιούμε και στον πίνακα **Topic** που εμπεριέχει τις θεματικές ενότητες κάθε συνταγής (π χ συνταγές του χωρίου, ριζότο συνταγές, πασχαλινά γλυκά).

- Ο πίνακας **Food_Group** όπως αναφέραμε και προηγουμένως αποτελείται από διάφορες ομάδες τροφίμων και κάθε μια από αυτές συνδέεται με μια εικόνα (**Image**) μέσω του Foreign Key : image_id.

- Με τον πίνακα **Cook** αναπαριστούμε των πληθυσμό των μαγείρων που μπορούμε να επιλέξουμε στον διαγωνισμό, τυχαία σε κάθε επεισόδιο. Κάθε μάγειρας στέλνει μια φωτογραφία για να συμμετάσχει στο διαγωνισμό και αυτή αποθηκεύεται στον πίνακα **Image** μέσω του Foreign key: image_id. Μέσα στον πίνακα **Cook** υπάρχουν τα attributes *username* και *password* με τα οποία ο κάθε μάγειρας χρήστης της εφαρμογής μπορεί να έχει πρόσβαση στο σύστημα. Επίσης, έχει την δυνατότητα να επεξεργαστεί όλα τα στοιχεία των συνταγών που του έχουν ανατεθεί, να προσθέσει νέες συνταγές, και να επεξεργαστεί τα προσωπικά του στοιχεία.

- Με τον πίνακα **Nationality_Cook** συνδέουμε τον κάθε μάγειρα με τις εθνικές κουζίνες που έχει εξειδίκευση. Ο εν λόγω πίνακας συνδέεται με τον **Cook** μέσω του Foreign Key : cook_id και με το **Nationality** μέσω του Foreign Key : nationality_id.

- Ο πίνακας **Episode** αποτελείται από τα επεισόδια του εν λόγω διαγωνισμού και συνδέεται με τον πίνακα **Image** μέσω του Foreign Key: image_id . Το *name* του κάθε επεισοδίου περιέχει τιμές από το 1 μέχρι το 10. Ενώ, το κάθε διαφορετικό έτος που γυρίζονται τα επεισόδια πρεσβεύεται από το attribute *season.* Το season παίρνει ακέραιες τιμές και αντιπροσωπεύει το date στη βάση, μιας και δεν γνωρίζουμε την ακριβή ημερομηνία που γυρίζονται τα επεισόδια κατά την διάρκεια του έτους. Άρα για παράδειγμα το 13 επεισόδιο χαρακτηρίζεται ως το 3 επεισόδιο της 2 σεζόν.

- Με τον πίνακα **Nationality_Episode** συνδέουμε το κάθε επεισόδιο με τις εθνικές κουζίνες που συμμετέχουν σε αυτό. Ο εν λόγω πίνακας συνδέεται με το **Episode** μέσω του Foreign Key : episode_id και με το **Nationality** μέσω του Foreign Key : nationality_id.

- Ο πίνακας **Administrator** περιέχει τους διαχειριστές της εφαρμογής που έχουν την δυνατότητα να καταχωρούν και να τροποποιούν όλα τα απαιτούμενα στοιχεία. Μπορούν να δημιουργήσουν αντίγραφο ασφαλείας για όλη τη βάση (backup) και να επαναφέρουν το σύστημα από αυτό (restore).

Οι παρακάτω πίνακες συνδέουν τον κάθε μάγειρα με κάθε επεισόδιο που συμμετάσχει, είτε ως διαγωνιζόμενος (Contestant) είτε ως κριτής (Judge) :

- **Cook_Episode_Judge** : συνδέεται με το **Episode** μέσω του Foreign Key : episode_id και με το **Cook** μέσω του Foreign Key : cook_id. Εφόσον υπάρχουν τρεις κριτές ανά επεισόδιο, αποφασίσαμε να δημιουργήσουμε το attribute *judge_number* για να αριθμήσουμε τον κάθε κριτή (από το 1 έως το 3).

- **Cook_Episode_Contestant** (Associative Entity): συνδέεται με το **Episode** μέσω του Foreign Key : episode_id, με το **Cook** μέσω του Foreign Key : cook_id και με το **Recipe** μέσω του Foreign Key : recipe_id. Μέσα σε αυτό το entity είναι αποθηκευμένα τα τρία rating για κάθε συνδυασμό μάγειρα/συνταγή/επεισόδιο , ως attributes.

Καθώς μια συνταγή μπορεί να έχει πολλαπλά βήματα, χρηστικές συμβουλές, μορφές γεύματος, ετικέτες, υλικά, εξοπλισμό, μάγειρες, επεισόδια, θεματικές ενότητες (multivalued attributes) δημιουργούμε τους παρακάτω πίνακες:

- Ο πίνακας **Recipe_Step** συνδέει την κάθε συνταγή με τα βήματα που αν ακολουθήσουμε θα ολοκληρωθεί. Για αυτόν τον λόγο συνδέεται με το **Recipe** μέσω του Foreign Key : recipe_id και με το **Step** μέσω του Foreign Key : step_id.

- Ο πίνακας **Recipe_Tip** συνδέει την κάθε συνταγή με τις χρηστικές συμβουλές που προσφέρουν βοήθεια στη διαδικασία ολοκλήρωσης της συνταγής . Για αυτόν τον λόγο συνδέεται με το **Recipe** μέσω του Foreign Key : recipe_id και με το **Tip** μέσω του Foreign Key : tip_id.

- Τα ίδια ακριβώς ισχύουν και για τους πίνακες **Recipe_Meal_Type** , **Recipe_Tag** , **Recipe_Ingredient** , **Recipe_Equipment** , **Recipe_Topic** , **Recipe_Cook** .

- Στον πίνακα **Recipe_Ingredient** χρησιμοποιούμε το attribute *quantity_type* που δέχεται τρεις τύπους ποσοτήτων κάθε υλικού σε μια συνταγή (γραμμάρια, ποσότητα πχ σε κούπες /κουταλιές, μη αριθμήσιμη πχ λίγο/ πολύ). Το attribute *serving_type* δείχνει το μέσο με το οποίο έγινε το serving της ποσότητας.

2.2 Indexes

Οι δείκτες δημιουργούνται σε στήλες που χρησιμοποιούνται συχνά και σε όρους WHERE, JOIN, ORDER BY κ.λπ. Αυτό επιταχύνει τη διαδικασία αναζήτησης δεδομένων. Παρακάτω δίνονται τα indexes που χρησιμοποιήθηκαν στη βάση:

```
-- Queries 1, 2
-- Index για την ενίσχυση του φιλτραρίσματος και του JOIN συνταγών κατά εθνικότητα.
-- Ενισχύει την αποδοτικότητα των JOIN και των όρων WHERE που βασίζονται στην εθνικότητα
-- και χρειάζονται τα recipe_id για JOIN με άλλους πίνακες.
CREATE INDEX idx_recipe_on_nationality_id_recipe_id ON Recipe(nationality_id, recipe_id);

-- Queries 1, 2, 6, 8, 9, 11, 12, 14, 15
-- Index για την επιτάχυνση των JOIN σε queries που περιλαμβάνουν διαγωνιζόμενους
-- επεισοδίων. Απαραίτητος για τη βελτίωση της απόδοσης σε σενάρια όπου πολλαπλά JOIN
-- βασίζονται στο φιλτράρισμα κατά episode_id και recipe_id.
CREATE INDEX idx_cec_on_episode_id_recipe_id ON Cook_Episode_Contestants(episode_id,
recipe_id);

-- Queries 2, 5, 10
-- Index που βοηθά στην ταχεία συσχέτιση επεισοδίων με τις σεζόν τους,
```

```sql
-- κάτι που είναι χρήσιμο για τη λειτουργία JOIN στα Sub-query που ομαδοποιούν δεδομένα
-- κατά σεζόν.
CREATE INDEX idx_episode_on_episode_id_season ON Episode(episode_id, season);

-- Queries 3, 7
-- Index που βοηθάει στη μείωση του φόρτου αναζήτησης συνταγών για κάθε μάγειρα, ειδικά
-- όταν γίνεται φιλτράρισμα μέσω ενός ενδεχομένως μεγάλου αριθμού συμμετοχών διαγωνιζομένων
-- επεισοδίων.
CREATE INDEX idx_cec_on_cook_id_recipe_id ON Cook_Episode_Contestants(cook_id, recipe_id);

-- Queries 5, 7, 11, 13
-- Index που επιτρέπει στη βάση δεδομένων να ομαδοποιεί γρήγορα τα δεδομένα κατά cook_id
-- και να έχει αποδοτική πρόσβαση στο episode_id, κάτι που είναι κρίσιμο για τη λειτουργία
-- JOIN  στα Sub-query που ομαδοποιούν δεδομένα κατά judges και τα επεισόδια που
-- συμμετέχουν και την πιθανώς επακόλουθη ομαδοποίηση κατά cook_id ή/και σεζόν.
CREATE INDEX idx_cej_on_cook_id_episode_id ON Cook_Episode_Judge(cook_id, episode_id);

-- Query 6
-- Index για την ταχύτερη πρόσβαση και το JOIN μεταξύ πινάκων Recipe_Tag.
-- Το συγκεκριμένο index βελτιστοποιεί τις λειτουργίες που αφορούν την ανάκτηση
-- δεδομένων με βάση το recipe_id και το tag_id, κάτι που είναι κρίσιμο για την αποδοτική
-- ένωση των ετικετών συνταγών στο εσωτερικό των subqueries.
CREATE INDEX idx_recipe_tag_on_recipe_id_tag_id ON Recipe_Tag(recipe_id, tag_id);

-- Query 8
-- Index που υποστηρίζει την ταχεία ανάκτηση δεδομένων κατά τη
-- διάρκεια των συνενώσεων και βελτιστοποιεί την καταμέτρηση του εξοπλισμού ανά επεισόδιο.
CREATE INDEX idx_re_on_recipe_id_equipment_id ON Recipe_Equipment(recipe_id, equipment_id);

-- Query 10
-- Index που επιταχύνει την ανάκτηση δεδομένων εθνικότητας που συνδέονται με
-- συγκεκριμένα επεισόδια, κάτι που είναι απαραίτητο για την αρχική λειτουργία JOIN στο
-- query. Βοηθάει στην ελαχιστοποίηση του χρόνου αναζήτησης κατά τη διάρκεια των JOIN
-- και του φιλτραρίσματος στο episode_id.
CREATE INDEX idx_ne_on_episode_id_nationality_id ON Nationality_Episode(episode_id,
nationality_id);

-- Query 12
-- Index που βελτιστοποιεί την αναζήτηση της δυσκολίας συνταγών κατά τα operations JOIN και
-- GROUP BY, επιταχύνοντας τον υπολογισμό των μέσων δυσκολιών ανά επεισόδιο.
CREATE INDEX idx_recipe_on_recipe_id_difficulty ON Recipe(recipe_id, difficulty);

-- Query 13
-- Index που βελτιώνει την απόδοση των JOIN μεταξύ του πίνακα Cook_Episode_Judge και
-- του πίνακα Cook. Εξασφαλίζει ταχεία αναζήτηση για συγκέντρωση δεδομένων βάσει
-- episode_id, κάτι που είναι σημαντικό για το GROUP BY του Query.
CREATE INDEX idx_cej_on_episode_id_cook_id ON Cook_Episode_Judge(episode_id, cook_id);
```

2.3 Triggers

Για την σωστότερη λειτουργία του διαγωνισμού σε καταστάσεις insert, delete, update έχουμε εισάγει στη βάση πολλαπλά triggers, όπως βλέπουμε παρακάτω:

TRIGGER *check_tip_limit_before_insert;*
TRIGGER *add_step_in_middle_of_recipe;*
TRIGGER *food_group_identity_on_recipe;*
TRIGGER *food_group_identity_on_recipe_update;*
TRIGGER *pick_winner_in_episode;*
TRIGGER *match_season_with_unique_episode;*
TRIGGER *nationality_episode_regulator;*
TRIGGER *contestant_episode_regulator;*
TRIGGER *calculate_kcal_per_portion_dynamically;*
TRIGGER *update_kcal_per_portion_dynamically;*
TRIGGER *delete_kcal_per_portion_dynamically;*
TRIGGER *update_kcal_on_ingredient_kcal_change;*
TRIGGER *update_kcal_on_portions_change;*
TRIGGER *three_different_judges_per_episode;*
TRIGGER *check_consecutive_episodes_per_contestant;*
TRIGGER *check_consecutive_episodes_per_judge;*
TRIGGER *check_consecutive_episodes_per_nationality;*
TRIGGER *check_consecutive_episodes_per_recipe;*
TRIGGER *link_recipe_to_cook_after_insert;*
TRIGGER *check_user_before_edit_on_cook;*
TRIGGER *check_user_before_edit_on_recipe;*
TRIGGER *check_user_before_add_on_recipe_ingredient;*
TRIGGER *check_user_before_edit_on_recipe_ingredient;*
TRIGGER *check_user_before_add_on_recipe_equipment;*
TRIGGER *check_user_before_edit_on_recipe_equipment;*
TRIGGER *check_user_before_add_on_recipe_topic;*
TRIGGER *check_user_before_edit_on_recipe_topic;*
TRIGGER *check_user_before_add_on_recipe_tag;*
TRIGGER *check_user_before_edit_on_recipe_tag;*
TRIGGER *check_user_before_add_on_recipe_tip;*
TRIGGER *check_user_before_edit_on_recipe_tip;*
TRIGGER *check_user_before_add_on_recipe_meal_type;*
TRIGGER *check_user_before_edit_on_recipe_meal_type*
TRIGGER *check_user_before_add_on_step;*
TRIGGER *check_user_before_edit_on_step;*
TRIGGER *check_user_before_delete_on_recipe_ingredient;*
TRIGGER *check_user_before_delete_on_recipe_equipment;*
TRIGGER *check_user_before_delete_on_recipe_topic;*
TRIGGER *check_user_before_delete_on_recipe_tag;*
TRIGGER *check_user_before_delete_on_recipe_tip;*
TRIGGER *check_user_before_delete_on_recipe_meal_type;*
TRIGGER *check_user_before_delete_on_step;*

Ανάλυση των Trigger

Παρατηρούμε τα παρακάτω:

- Υπάρχουν τα triggers check_user που ελέγχουν το αν είναι επιτρεπτή η επεξεργασία δεδομένων του κάθε μάγειρα από τον χρήστη που έχει συνδεθεί στη βάση.
- Υπάρχουν τα trigger check_consecutive_episodes που ελέγχουν τις συνεχόμενες εμφανίσεις μάγειρων, κριτών, εθνικοτήτων, συνταγών σε επεισόδια.
- Υπάρχουν τα trigger kcal_per_portion που υπολογίζουν δυναμικά τον συνολικό αριθμό θερμίδων ανά μερίδα για κάθε συνταγή.
- Υπάρχουν τα trigger food_group_identity για τον χαρακτηρισμό της κάθε συνταγής, με βάση την ομάδα τροφίμων που βρίσκεται το βασικό της συστατικό.
- Υπάρχουν trigger, check_tip_limit_before_insert που ελέγχει το όριο έως τρία tips per recipe, pick_winner_in_episode που διαλέγει τον νικητή του κάθε επεισοδίου με βάση το μεγαλύτερο άθροισμα ratings και σε περίπτωση ισοβαθμίας τα κριτήρια τα οποία έχουν αναφερθεί στην εκφώνηση και match_season_with_unique_episode που οριοθετεί 10 επεισόδια από το 1 έως το 10 για κάθε σεζόν.

DDL Script

Οι εντολές DDL χρησιμοποιούνται για τη διαμόρφωση της δομής της βάσης δεδομένων. Πιο συγκεκριμένα, στο DDL script δημιουργούνται όλοι οι πίνακες, τα triggers, τα check constraints και τα procedures της βάσης.

Ενδεικτικά βλέπουμε παρακάτω την δημιουργία του πίνακα Cook :

```
CREATE TABLE IF NOT EXISTS Cook (

    cook_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    contact_number VARCHAR(10) NOT NULL,
    date_of_birth DATE NOT NULL,
    years_of_experience INT NOT NULL,
    ranking ENUM("chef", "sous chef", "cook A", "cook B", "cook C") NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(50) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (cook_id),
    CONSTRAINT fk_cook_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Στο φάκελο Code στο git repo αλλά και στο τέλος του report είναι τοποθετημένο το παρακάτω DDL script (cooking_contest_db.sql) :

https://github.com/nikolasbv/Cooking_Contest_DB/blob/main/SQL_Code/cooking_contest_db.sql

Για την διαχείριση των διαφορετικών τύπων χρηστών (Users, Admin) έχουν δημιουργηθεί ROLES στα οποία έχουν δοθεί διαφορετικά δικαιώματα ανάλογα με τον τύπο χρήστη που τους αντιστοιχεί. Συγκεκριμένα, οι περιορισμοί υπάρχουν για τους users, ενώ οι διαχειριστές έχουν πρόσβαση σε ολόκληρη τη βάση. Συνήθως, η διαχείριση των χρηστών θα συνέβαινε αποκλειστικά στο front-end, καθώς ανάλογα με τον χρήστη πολύ εύκολα ρυθμίζεται σε τι έχει πρόσβαση. Όσον αφορά το back up και το restore της βάσης, θα δινόταν πρόσβαση στο front-end στον διαχειριστή, για να εκτελέσει εντολές από το mysqldump program, ώστε να κρατήσουν αντίγραφα της βάσης, αλλά και από το mysqladmin για την εύρυθμη λειτουργία της.

DML Script

Οι εντολές DML χρησιμοποιούνται για τη διαχείριση των δεδομένων μέσα στη βάση. Επομένως το DML script αποτελείται από τα αρχικά δεδομένα που εισάγουμε μέσω ενός insert script στη βάση αλλά και των queries που ζητήθηκαν στην εκφώνηση της εργασίας.

Για την εισαγωγή των δεδομένων στη βάση δημιουργήθηκε ένα notebook databases_data_generator.ipynb, μέσω του οποίου αξιοποιούμε διάφορα datasets (τα οποία βρίσκονται στο git repo), προκειμένου να παράξουμε τα δεδομένα της βάσης. Επιπροσθέτως, δημιουργήθηκε το python script episode_creator.py, το οποίο με βάση τα υπάρχοντα δεδομένα της βάσης πραγματοποιεί την κλήρωση ενός επεισοδίου του διαγωνισμού και εισάγει αυτόματα τα αντίστοιχα δεδομένα στη βάση. Για την δημιουργία δεδομένων της βάσης και για τον διαγωνισμό

δημιουργήθηκε το contest_creator.py που καλεί 100 φορές μια τροποποιημένη έκδοση του episode_creator.py (προκειμένου τα data που προκύπτουν να παράγουν επιθυμητά output σε ορισμένα queries) και ταυτόχρονα αποθηκεύει τα insert data σε ένα ξεχωριστό insert_contest_data.sql.

Στον φάκελο SQL_Code στο git repo είναι τοποθετημένο το συνολικό DML script με όλα τα δεδομένα (insert_data.sql) :

https://github.com/nikolasbv/Cooking_Contest_DB/blob/main/SQL_Code/insert_data.sql

Ενώ στον φάκελο Data_Generation υπάρχουν όλα τα scripts και τα αρχεία που αναφέρθηκαν για την δημιουργία των δεδομένων, καθώς και το episode_creator για την κλήρωση του διαγωνισμού:

https://github.com/nikolasbv/Cooking_Contest_DB/tree/main/Data_Generation

3. Queries

Ακολουθεί η υλοποίηση των queries που ζητούνται, ενώ υπάρχουν και στο git repo μέσα στον φάκελο SQL_Code στο αρχείο queries.sql:

https://github.com/nikolasbv/Cooking_Contest_DB/blob/main/SQL_Code/queries.sql

Σε ορισμένα queries υπάρχουν δύο υλοποιήσεις, καθώς δεν ήμασταν σίγουροι για το ποια από τις δύο υλοποιήσεις ζητούσε η εκφώνηση και ανάλογα με το context άλλαζαν και τα δεδομένα που επέστρεφε το κάθε query. Θα υπάρξει επιπλέον εξήγηση στα queries που συμβαίνει αυτό.

3.1. Μέσος Όρος Αξιολογήσεων (σκορ) ανά μάγειρα και Εθνική κουζίνα.

```
WITH CookRatings AS (
    SELECT cec.cook_id, (cec.rating_1 + cec.rating_2 + cec.rating_3) AS total_rating
    FROM Cook_Episode_Contestants cec
),
CookAverage AS (
    SELECT cr.cook_id, AVG(cr.total_rating) AS average_rating
    FROM CookRatings cr
    GROUP BY cr.cook_id
),
NationalityRatings AS (
    SELECT r.nationality_id, cr.total_rating
    FROM CookRatings cr
    JOIN Recipe r ON cr.cook_id = r.recipe_id
),
NationalityAverage AS (
    SELECT nr.nationality_id, AVG(nr.total_rating) AS average_nationality_rating
    FROM NationalityRatings nr
    GROUP BY nr.nationality_id
)
SELECT
    'Cook' AS Type, ca.cook_id AS ID, ca.average_rating AS Average_Rating
```

```sql
FROM CookAverage ca

UNION ALL

SELECT 'Nationality' AS Type, na.nationality_id AS ID, na.average_nationality_rating AS
Average_Rating
FROM NationalityAverage na
ORDER BY Type, ID;
```

3.2. Για δεδομένη Εθνική κουζίνα και έτος, ποιοι μάγειρες ανήκουν σε αυτήν και ποιοι μάγειρες συμμετείχαν σε επεισόδια;

```sql
SET @NationalityID = 103;
SET @Season = 4;

WITH CooksByNationality AS (
    SELECT nc.cook_id
    FROM Nationality_Cook nc
    WHERE nc.nationality_id = @NationalityID
), CooksInEpisodes AS (
    SELECT DISTINCT cec.cook_id
    FROM Cook_Episode_Contestants cec
    JOIN Episode e ON cec.episode_id = e.episode_id
    JOIN Recipe r ON cec.recipe_id = r.recipe_id
    WHERE e.season = @Season AND r.nationality_id = @NationalityID
)

SELECT * FROM (
    SELECT 'Nationality Linked Cook' AS Type, c.cook_id, c.first_name, c.last_name
    FROM CooksByNationality cbn
    JOIN Cook c ON cbn.cook_id = c.cook_id

    UNION

    SELECT 'Episode Participating Cook' AS Type, c.cook_id, c.first_name, c.last_name
    FROM CooksInEpisodes cie
    JOIN Cook c ON cie.cook_id = c.cook_id
) AS results
ORDER BY Type, cook_id;
```

3.3. Βρείτε τους νέους μάγειρες (ηλικία < 30 ετών) που έχουν τις περισσότερες συνταγές.

```sql
#IF WE ARE REFFERING TO RECIPE_COOK (EVERY RECIPE HE CAN MAKE)
SELECT Cook.cook_id, Cook.first_name, Cook.last_name, COUNT(*) AS recipe_count,
TIMESTAMPDIFF(YEAR, Cook.date_of_birth, CURDATE()) AS age
FROM Cook
INNER JOIN Recipe_Cook ON Cook.cook_id = Recipe_Cook.cook_id
WHERE TIMESTAMPDIFF(YEAR, Cook.date_of_birth, CURDATE()) < 30
GROUP BY Cook.cook_id, Cook.first_name, Cook.last_name
ORDER BY COUNT(*) DESC;

#IF WE ARE REFFERING TO COOK_EPISODE_CONTESTANTS (RECIPE_ID ATTRIBUTE) (RECIPES HE WAS
#ASSIGNED IN EPISODES)
```

```sql
SELECT Cook.cook_id, Cook.first_name, Cook.last_name, COUNT(DISTINCT cec.recipe_id) AS
recipe_count, TIMESTAMPDIFF(YEAR, Cook.date_of_birth, CURDATE()) AS age
FROM Cook
INNER JOIN Cook_Episode_Contestants cec ON Cook.cook_id = cec.cook_id
WHERE TIMESTAMPDIFF(YEAR, Cook.date_of_birth, CURDATE()) < 30
GROUP BY Cook.cook_id, Cook.first_name, Cook.last_name
ORDER BY recipe_count DESC;
```

Στο συγκεκριμένο query, εάν θέλουμε να επιλέξουμε από τις συνταγές που ο κάθε μάγειρας μπορεί να εκτελέσει (υλοποίηση 1), τότε πρέπει να χρησιμοποιήσουμε το table Recipe_Cook, ενώ αν θέλουμε να επιλέξουμε τις συνταγές που έχουν ανατεθεί σε κάποιο επεισόδιο (υλοποίηση 2), τότε πρέπει να χρησιμοποιήσουμε το table Cook_Episode_Contestants, από το οποίο μπορούμε να λάβουμε για συγκεκριμένο επεισόδιο και μάγειρα την συνταγή που το ανατέθηκε.

3.4. Βρείτε τους μάγειρες που δεν έχουν συμμετάσχει ποτέ σε ως κριτές σε κάποιο επεισόδιο.

```sql
SELECT Cook.cook_id, Cook.first_name, Cook.last_name
FROM Cook
WHERE Cook.cook_id NOT IN (SELECT cook_id FROM Cook_Episode_Judge);
```

3.5. Ποιοι κριτές έχουν συμμετάσχει στον ίδιο αριθμό επεισοδίων σε διάστημα ενός έτους με περισσότερες από 3 εμφανίσεις;

```sql
#ONLY FROM THE SAME SEASON
WITH JudgeAppearances AS (
    SELECT cej.cook_id,e.season,COUNT(*) AS appearances
    FROM Cook_Episode_Judge cej
    JOIN Episode e ON cej.episode_id = e.episode_id
    GROUP BY cej.cook_id, e.season
    HAVING COUNT(*) > 3
)
SELECT j1.cook_id,j1.season,j1.appearances
FROM JudgeAppearances j1
JOIN JudgeAppearances j2 ON j1.season = j2.season AND j1.appearances = j2.appearances AND
j1.cook_id <> j2.cook_id
GROUP BY j1.cook_id, j1.season, j1.appearances
ORDER BY j1.season, j1.appearances;

#FROM ALL THE SEASONS SEASON
WITH JudgeAppearances AS (
    SELECT cej.cook_id,e.season,COUNT(*) AS appearances
    FROM Cook_Episode_Judge cej
    JOIN Episode e ON cej.episode_id = e.episode_id
    GROUP BY cej.cook_id, e.season
    HAVING COUNT(*) > 3
)
SELECT j1.cook_id,j1.season,j1.appearances
FROM JudgeAppearances j1
JOIN JudgeAppearances j2 ON j1.appearances = j2.appearances AND j1.cook_id <> j2.cook_id
GROUP BY j1.cook_id, j1.season, j1.appearances
ORDER BY j1.appearances DESC, j1.season;
```

Στο συγκεκριμένο query έχουμε δύο υλοποιήσεις, όπου στην πρώτη επιλέγουμε τους κριτές που έχουν συμμετάσχει στον ίδιο αριθμό επεισοδίων μέσα στην ίδια σεζόν, ενώ στην δεύτερη επιλέγουμε για κριτές που έχουν συμμετάσχει στο ίδιο αριθμό επεισοδίων στο διάστημα μιας σεζόν αλλά για οποιαδήποτε σεζόν του διαγωνισμού.

3.6. Πολλές συνταγές καλύπτουν περισσότερες από μια ετικέτες. Ανάμεσα σε ζεύγη πεδίων (π.χ. brunch και κρύο πιάτο) που είναι κοινά στις συνταγές, βρείτε τα 3 κορυφαία (top-3) ζεύγη που εμφανίστηκαν σε επεισόδια6 Για το ερώτημα αυτό η απάντηση σας θα πρέπει να περιλαμβάνει εκτός από το ερώτημα (query), εναλλακτικό Query Plan (πχ με force index), τα αντίστοιχα traces και τα συμπεράσματα σας από την μελέτη αυτών.

```sql
SELECT tag1, tag2, COUNT(DISTINCT cec.episode_id) AS appearances
FROM (
    SELECT
        r.recipe_id,
        LEAST(t1.name, t2.name) AS tag1,
        GREATEST(t1.name, t2.name) AS tag2
    FROM
        Recipe_Tag rt1
    INNER JOIN Recipe_Tag rt2 ON rt1.recipe_id = rt2.recipe_id AND rt1.tag_id < rt2.tag_id
    INNER JOIN Tag t1 ON rt1.tag_id = t1.tag_id
    INNER JOIN Tag t2 ON rt2.tag_id = t2.tag_id
    INNER JOIN Recipe r ON rt1.recipe_id = r.recipe_id
    GROUP BY
        r.recipe_id, tag1, tag2
) AS pairs
INNER JOIN Cook_Episode_Contestants cec ON pairs.recipe_id = cec.recipe_id
GROUP BY
    tag1, tag2
ORDER BY
    appearances DESC
LIMIT 3;
```

Για να δημιουργήσουμε το εναλλακτικό query plan, πρώτα εκτελούμε το παραπάνω query με την εντολή EXPLAIN, προκειμένου να πάρουμε τα αντίστοιχα traces και να δούμε πως τρέχει ο optimizer της mysql το query αυτό.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	100	Using temporary; Using filesort
1	PRIMARY	cec	ref	fk3_recipe_episode_cook	fk3_recipe_episode_cook	4	pairs.recipe_id	1	Using index
2	DERIVED	r	index	PRIMARY,idx_recipe_on_recipe_id_difficulty	fk_recipe_image	4	NULL	100	Using index; Using temporary; Using filesort
2	DERIVED	rt1	ref	PRIMARY,fk_recipe_tag,idx_recipe_tag_on_recipe_id_tag_id	PRIMARY	4	contest.r.recipe_id	1	Using index
2	DERIVED	t1	eq_ref	PRIMARY	PRIMARY	4	contest.rt1.tag_id	1	
2	DERIVED	rt2	ref	PRIMARY,fk_recipe_tag,idx_recipe_tag_on_recipe_id_tag_id	PRIMARY	4	contest.r.recipe_id	1	Using where; Using index
2	DERIVED	t2	eq_ref	PRIMARY	PRIMARY	4	contest.rt2.tag_id	1	

7 rows in set (0.002 sec)

Στη συνέχεια, δημιουργούμε το εναλλακτικό query plan, χρησιμοποιώντας forced indexing για τα indexes idx_recipe_tag_on_recipe_id_tag_id και idx_recipe_on_recipe_id_difficulty, το οποίο φαίνεται παρακάτω.

```sql
--Force Indexing
SELECT tag1, tag2, COUNT(DISTINCT cec.episode_id) AS appearances
FROM (
    SELECT
        r.recipe_id,
        LEAST(t1.name, t2.name) AS tag1,
        GREATEST(t1.name, t2.name) AS tag2
    FROM Recipe_Tag rt1 FORCE INDEX (idx_recipe_tag_on_recipe_id_tag_id)
    INNER JOIN Recipe_Tag rt2
        FORCE INDEX (idx_recipe_tag_on_recipe_id_tag_id)
        ON rt1.recipe_id = rt2.recipe_id AND rt1.tag_id < rt2.tag_id
    INNER JOIN Tag t1 ON rt1.tag_id = t1.tag_id
    INNER JOIN Tag t2 ON rt2.tag_id = t2.tag_id
    INNER JOIN Recipe r FORCE INDEX (idx_recipe_on_recipe_id_difficulty)
        ON rt1.recipe_id = r.recipe_id
    GROUP BY r.recipe_id, tag1, tag2
) AS pairs
INNER JOIN Cook_Episode_Contestants cec ON pairs.recipe_id = cec.recipe_id
GROUP BY
    tag1, tag2
ORDER BY
    appearances DESC
LIMIT 3;
```

Εκτελούμε το παραπάνω query με την εντολή EXPLAIN και λαμβάνουμε τα παρακάτω αποτελέσματα.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	100	Using temporary; Using filesort
1	PRIMARY	cec	ref	fk3_recipe_episode_cook	fk3_recipe_episode_cook	4	pairs.recipe_id	1	Using index
2	DERIVED	r	index	idx_recipe_on_recipe_id_difficulty	idx_recipe_on_recipe_id_difficulty	8	NULL	100	Using index; Using temporary; Using filesort
2	DERIVED	rt1	ref	idx_recipe_tag_on_recipe_id_tag_id	idx_recipe_tag_on_recipe_id_tag_id	4	contest.r.recipe_id	1	Using index
2	DERIVED	t1	eq_ref	PRIMARY	PRIMARY	4	contest.rt1.tag_id	1	
2	DERIVED	rt2	ref	idx_recipe_tag_on_recipe_id_tag_id	idx_recipe_tag_on_recipe_id_tag_id	4	contest.r.recipe_id	1	Using where; Using index
2	DERIVED	t2	eq_ref	PRIMARY	PRIMARY	4	contest.rt2.tag_id	1	

7 rows in set (0.001 sec)

Παρατηρούμε πως τα indexes τα οποία επιβάλαμε έχουν όντως χρησιμοποιηθεί, με αποτέλεσμα να διαλέγουμε πιθανώς το βέλτιστο access path για τα joins που τα χρησιμοποιούνε, όμως τα rows που τελικά χρησιμοποιούνται από το query παραμένουν τα ίδια και άρα δεν μπορούμε να πούμε πως βελτιώθηκε η απόδοση. Παρατηρούμε παρόλα αυτά πως ο χρόνος εκτέλεσης είναι ο μισός (0.002 sec στο πρώτο και 0.001 sec στο δεύτερο), γεγονός που πιθανώς να δηλώνει μια μικρή βελτίωση του query.

3.7. Βρείτε όλους τους μάγειρες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον μάγειρα με τις περισσότερες συμμετοχές σε επεισόδια.

```sql
--ONLY CONTESTANTS
WITH CookParticipations AS (
    SELECT ce.cook_id, c.first_name, c.last_name, COUNT(*) AS appearances
    FROM Cook_Episode_Contestants ce
    JOIN Cook c ON ce.cook_id = c.cook_id
    GROUP BY ce.cook_id, c.first_name, c.last_name
```

```
), MaxParticipation AS (
    SELECT MAX(appearances) AS max_appearances
    FROM CookParticipations
)

SELECT 'Maximum Appearances' AS Description, cp.cook_id, cp.first_name, cp.last_name,
cp.appearances AS Appearances
FROM CookParticipations cp
WHERE cp.appearances = (SELECT max_appearances FROM MaxParticipation)

UNION ALL

SELECT 'At least 5 fewer appearances' AS Description, cp.cook_id, cp.first_name,
cp.last_name, cp.appearances AS Appearances
FROM CookParticipations cp
JOIN MaxParticipation mp ON cp.appearances <= (mp.max_appearances - 5) AND cp.appearances <
mp.max_appearances
ORDER BY Appearances DESC;


--BOTH CONTESTANTS AND JUDGES (MAX IS FOR TOTAL APPEARANCES)
WITH ContestantCounts AS (
    SELECT ce.cook_id, COUNT(*) AS contestant_appearances
    FROM Cook_Episode_Contestants ce
    GROUP BY ce.cook_id
),
JudgeCounts AS (
    SELECT cej.cook_id, COUNT(*) AS judge_appearances
    FROM Cook_Episode_Judge cej
    GROUP BY cej.cook_id
),
CombinedCounts AS (
    SELECT
        c.cook_id,
        c.first_name,
        c.last_name,
        COALESCE(cc.contestant_appearances, 0) AS contestant_appearances,
        COALESCE(jc.judge_appearances, 0) AS judge_appearances,
        (COALESCE(cc.contestant_appearances, 0) + COALESCE(jc.judge_appearances, 0)) AS
total_appearances
    FROM Cook c
    LEFT JOIN ContestantCounts cc ON c.cook_id = cc.cook_id
    LEFT JOIN JudgeCounts jc ON c.cook_id = jc.cook_id
),
MaxParticipation AS (
    SELECT MAX(total_appearances) AS max_appearances
    FROM CombinedCounts
)

SELECT CASE WHEN cp.total_appearances = mp.max_appearances THEN 'Maximum Appearances'
            ELSE 'At least 5 fewer appearances' END AS Description,
       cp.cook_id,
```

```
       cp.first_name,
       cp.last_name,
       cp.contestant_appearances,
       cp.judge_appearances,
       cp.total_appearances
FROM CombinedCounts cp
CROSS JOIN MaxParticipation mp
WHERE cp.total_appearances = mp.max_appearances OR cp.total_appearances <=
(mp.max_appearances - 5)
ORDER BY (cp.total_appearances = mp.max_appearances) DESC, total_appearances DESC;
```

Σε αυτό το query έχουμε δύο υλοποιήσεις, όπου στην πρώτη υλοποίηση ψάχνουμε για μάγειρες που συμμετείχαν στον διαγωνισμό μόνο ως διαγωνιζόμενοι, ενώ στην δεύτερη υλοποίηση ψάχνουμε για μάγειρες που συμμετείχαν είτε ως διαγωνιζόμενοι είτε ως κριτές.

3.8. Σε ποιο επεισόδιο χρησιμοποιήθηκαν τα περισσότερα εξαρτήματα (εξοπλισμός); Ομοίως με ερώτημα 3.6, η απάντηση σας θα πρέπει να περιλαμβάνει εκτός από το ερώτημα (query), εναλλακτικό Query Plan (πχ με force index), τα αντίστοιχα traces και τα συμπεράσματα σας από την μελέτη αυτών.

```
--ONLY UNIQUE EQUIPMENT
WITH EpisodeEquipmentCounts AS (
    SELECT e.episode_id, e.name AS episode_name, e.season, COUNT(DISTINCT re.equipment_id)
AS total_unique_equipment
    FROM Episode e
    JOIN Cook_Episode_Contestants cec ON e.episode_id = cec.episode_id
    JOIN Recipe_Equipment re ON cec.recipe_id = re.recipe_id
    GROUP BY e.episode_id, e.name, e.season
)

SELECT eec.episode_id, eec.episode_name AS episode, eec.season, eec.total_unique_equipment
FROM EpisodeEquipmentCounts eec
WHERE eec.total_unique_equipment = (
    SELECT MAX(total_unique_equipment) FROM EpisodeEquipmentCounts
)
ORDER BY eec.total_unique_equipment DESC;


--TOTAL EQUIPMENT (NOT UNIQUE)
WITH EpisodeEquipmentCounts AS (
    SELECT e.episode_id, e.name AS episode_name, e.season, COUNT(re.equipment_id) AS
total_equipment
    FROM Episode e
    JOIN Cook_Episode_Contestants cec ON e.episode_id = cec.episode_id
    JOIN Recipe_Equipment re ON cec.recipe_id = re.recipe_id
    GROUP BY e.episode_id, e.name, e.season
)

SELECT eec.episode_id, eec.episode_name AS episode, eec.season, eec.total_equipment
FROM EpisodeEquipmentCounts eec
```

```
WHERE eec.total_equipment = (
    SELECT MAX(total_equipment) FROM EpisodeEquipmentCounts
)
ORDER BY eec.total_equipment DESC;
```

Για το query αυτό έχουμε δύο υλοποιήσεις, όπου στην πρώτη υλοποίηση ψάχνουμε για τον συνολικό αριθμό μοναδικού εξοπλισμού που εμφανίστηκε ανά επεισόδιο, ενώ στην δεύτερη ψάχνουμε για τον συνολικό αριθμό εξοπλισμού γενικά (όπου αν χρησιμοποιήθηκε παραπάνω από μια φορά ο ίδιος εξοπλισμός, κάθε φορά μετράει στο συνολικό αποτέλεσμα). Για το εναλλακτικό query plan χρησιμοποιήθηκε η δεύτερη υλοποίησης του συγκεκριμένου query.

Για το εναλλακτικό query plan επαναλαμβάνουμε την ίδια διαδικασία. Ακολουθεί το παραπάνω query εκτελεσμένο μαζί με την εντολή EXPLAIN.

```
+------+-------------+------------+------+---------------------------------------------------------------+-------------------------------+---------+----------------------+------+----------------------------------------------
--------+
| id   | select_type | table      | type | possible_keys                                                 | key                           | key_len | ref                  | rows | Extra
        |
+------+-------------+------------+------+---------------------------------------------------------------+-------------------------------+---------+----------------------+------+----------------------------------------------
--------+
|    1 | PRIMARY     | <derived2> | ALL  | NULL                                                          | NULL                          | NULL    | NULL                 |  300 | Using where
        |
|    3 | SUBQUERY    | <derived4> | ALL  | NULL                                                          | NULL                          | NULL    | NULL                 |  300 |
        |
|    4 | DERIVED     | e          | ALL  | PRIMARY,idx_episode_on_episode_id_season                      | NULL                          | NULL    | NULL                 |  100 | Using temporary; Using f
ilesort |
|    4 | DERIVED     | cec        | ref  | PRIMARY,fk3_recipe_episode_cook,idx_cec_on_episode_id_recipe_id | idx_cec_on_episode_id_recipe_id | 4       | contest.e.episode_id |    1 | Using index
        |
|    4 | DERIVED     | re         | ref  | idx_re_on_recipe_id_equipment_id                              | idx_re_on_recipe_id_equipment_id | 4       | contest.cec.recipe_id |    3 | Using index
        |
|    2 | DERIVED     | e          | ALL  | PRIMARY,idx_episode_on_episode_id_season                      | NULL                          | NULL    | NULL                 |  100 | Using temporary; Using f
ilesort |
|    2 | DERIVED     | cec        | ref  | PRIMARY,fk3_recipe_episode_cook,idx_cec_on_episode_id_recipe_id | idx_cec_on_episode_id_recipe_id | 4       | contest.e.episode_id |    1 | Using index
        |
|    2 | DERIVED     | re         | ref  | idx_re_on_recipe_id_equipment_id                              | idx_re_on_recipe_id_equipment_id | 4       | contest.cec.recipe_id |    3 | Using index
        |
+------+-------------+------------+------+---------------------------------------------------------------+-------------------------------+---------+----------------------+------+----------------------------------------------
--------+
8 rows in set (0.002 sec)
```

Χρησιμοποιώντας ως forced indexes idx_episode_on_episode_id_season, idx_re_on_recipe_id_equipment_id, idx_cec_on_episode_id_recipe_id, δημιουργούμε το εναλλακτικό query plan και εκτελούμε ξανά το query με την εντολή EXPLAIN.

```
--Force Indexing
EXPLAIN
WITH EpisodeEquipmentCounts AS (
    SELECT e.episode_id, e.name AS episode_name, e.season, COUNT(re.equipment_id) AS
total_equipment
    FROM Episode e
    FORCE INDEX (idx_episode_on_episode_id_season)
    JOIN Cook_Episode_Contestants cec FORCE INDEX (idx_cec_on_episode_id_recipe_id) ON
e.episode_id = cec.episode_id
    JOIN Recipe_Equipment re FORCE INDEX (idx_re_on_recipe_id_equipment_id) ON
cec.recipe_id = re.recipe_id
    GROUP BY e.episode_id, e.name, e.season
)

SELECT eec.episode_id, eec.episode_name AS episode, eec.season, eec.total_equipment
FROM EpisodeEquipmentCounts eec
WHERE eec.total_equipment = (
    SELECT MAX(total_equipment) FROM EpisodeEquipmentCounts
)
ORDER BY eec.total_equipment DESC;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	300	Using where
3	SUBQUERY	<derived4>	ALL	NULL	NULL	NULL	NULL	300	
4	DERIVED	e	ALL	idx_episode_on_episode_id_season	NULL	NULL	NULL	100	Using temporary; Using filesort
4	DERIVED	cec	ref	idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
4	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index
2	DERIVED	e	ALL	idx_episode_on_episode_id_season	NULL	NULL	NULL	100	Using temporary; Using filesort
2	DERIVED	cec	ref	idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
2	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index

8 rows in set (0.001 sec)

Τα αποτελέσματα για το query plan για το ερώτημα 3.8 είναι πολύ παρόμοια με αυτά του query plan του ερωτήματος 3.6. Συγκεκριμένα, αν και εφαρμόστηκαν τα forced indexes που χρησιμοποιήσαμε, ο αριθμός γραμμών δεν άλλαξε προκειμένου να πούμε ότι βελτιώθηκε η απόδοση του query, μειώθηκε όμως ο χρόνος εκτέλεσης από 0.002 sec σε 0.001 sec.

Προκειμένου να βελτιώσουμε παραπάνω την αποδοτικότητα του query με βάση τις παρατηρήσεις μας, δημιουργούμε ένα index που ενώνει το episode_id με το season (όπως και το index που χρησιμοποιήσαμε μόλις) αλλά και με το name του επεισοδίου, προκειμένου να αποφύγουμε την χρήση filesort.

```
CREATE INDEX idx_episode_grouping ON Episode(episode_id, name, season);
```

Παρατηρούμε πλέον, πως και με force index και χωρίς, ο optimizer πλέον δεν χρησιμοποιεί filesort και ο χρόνος εκτέλεσης είναι και στις δύο περιπτώσεις 0.001 sec.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	300	Using where
3	SUBQUERY	<derived4>	ALL	NULL	NULL	NULL	NULL	300	
4	DERIVED	e	index	PRIMARY,idx_episode_on_episode_id_season,idx_episode_grouping	idx_episode_grouping	12	NULL	100	Using index
4	DERIVED	cec	ref	PRIMARY,fk3_recipe_episode_cook,idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
4	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index
2	DERIVED	e	index	PRIMARY,idx_episode_on_episode_id_season,idx_episode_grouping	idx_episode_grouping	12	NULL	100	Using index
2	DERIVED	cec	ref	PRIMARY,fk3_recipe_episode_cook,idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
2	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index

8 rows in set (0.001 sec)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	300	Using where
3	SUBQUERY	<derived4>	ALL	NULL	NULL	NULL	NULL	300	
4	DERIVED	e	index	idx_episode_grouping	idx_episode_grouping	12	NULL	100	Using index
4	DERIVED	cec	ref	idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
4	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index
2	DERIVED	e	index	idx_episode_grouping	idx_episode_grouping	12	NULL	100	Using index
2	DERIVED	cec	ref	idx_cec_on_episode_id_recipe_id	idx_cec_on_episode_id_recipe_id	4	contest.e.episode_id	1	Using index
2	DERIVED	re	ref	idx_re_on_recipe_id_equipment_id	idx_re_on_recipe_id_equipment_id	4	contest.cec.recipe_id	3	Using index

8 rows in set (0.001 sec)

3.9. Λίστα με μέσο όρο αριθμού γραμμάριων υδατανθράκων στο διαγωνισμό ανά έτος;

```
SELECT season,

       AVG(Recipe.curbs_grams_per_portion) AS avg_carbs_per_portion
FROM Episode
JOIN Cook_Episode_Contestants ON Episode.episode_id = Cook_Episode_Contestants.episode_id
JOIN Recipe ON Cook_Episode_Contestants.recipe_id = Recipe.recipe_id
GROUP BY season;
```

3.10. Ποιες Εθνικές κουζίνες έχουν τον ίδιο αριθμό συμμετοχών σε διαγωνισμούς, σε διάστημα δύο συνεχόμενων ετών, με τουλάχιστον 3 συμμετοχές ετησίως ;

```sql
WITH NationalityAppearances AS (
    SELECT ne.nationality_id, e.season, COUNT(*) AS appearances
    FROM Nationality_Episode ne
    JOIN Episode e ON ne.episode_id = e.episode_id
    GROUP BY ne.nationality_id, e.season
    HAVING COUNT(*) >= 3
), ConsecutiveSeasons AS (
    SELECT n1.nationality_id, n1.season AS season1, n2.season AS season2, n1.appearances +
n2.appearances AS total_appearances
    FROM NationalityAppearances n1
    JOIN NationalityAppearances n2 ON n1.nationality_id = n2.nationality_id AND n2.season =
n1.season + 1
), FilteredNationalities AS (
    SELECT cs1.nationality_id, cs1.season1, cs1.season2, cs1.total_appearances
    FROM ConsecutiveSeasons cs1
    JOIN ConsecutiveSeasons cs2 ON cs1.total_appearances = cs2.total_appearances AND
cs1.nationality_id <> cs2.nationality_id
)

SELECT DISTINCT fn.nationality_id AS Nationality_id, n.name AS Name, fn.season1,
fn.season2, fn.total_appearances
FROM FilteredNationalities fn
JOIN Nationality n ON n.nationality_id=fn.nationality_id
ORDER BY fn.total_appearances DESC, fn.season1;
```

3.11. Βρείτε τους top-5 κριτές που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα μάγειρα. (όνομα κριτή, όνομα μάγειρα και συνολικό σκορ βαθμολόγησης)

```sql
WITH JudgeRatings AS (
    SELECT
        cec.cook_id AS contestant_cook_id,
        cec.episode_id,
        cec.rating_1,
        cec.rating_2,
        cec.rating_3
    FROM Cook_Episode_Contestants cec
),
Judges AS (
    SELECT
        cej.cook_id AS judge_cook_id,
        cej.episode_id,
        cej.judge_number
    FROM Cook_Episode_Judge cej
),
JudgeContestantRatings AS (
    SELECT
        j.judge_cook_id,
        jr.contestant_cook_id,
        CASE
            WHEN j.judge_number = 1 THEN jr.rating_1
            WHEN j.judge_number = 2 THEN jr.rating_2
            WHEN j.judge_number = 3 THEN jr.rating_3
```

```
        END AS rating
    FROM Judges j
    JOIN JudgeRatings jr ON j.episode_id = jr.episode_id
),
TotalRatings AS (
    SELECT
        j.judge_cook_id,
        j.contestant_cook_id,
        SUM(j.rating) AS total_rating
    FROM JudgeContestantRatings j
    GROUP BY j.judge_cook_id, j.contestant_cook_id
),
JudgeNames AS (
    SELECT
        c.cook_id,
        c.first_name AS judge_first_name,
        c.last_name AS judge_last_name
    FROM Cook c
),
ContestantNames AS (
    SELECT
        c.cook_id,
        c.first_name AS contestant_first_name,
        c.last_name AS contestant_last_name
    FROM Cook c
)
SELECT
    jn.cook_id AS judge_cook_id,
    jn.judge_first_name,
    jn.judge_last_name,
    cn.cook_id AS contestant_cook_id,
    cn.contestant_first_name,
    cn.contestant_last_name,
    tr.total_rating
FROM TotalRatings tr
JOIN JudgeNames jn ON tr.judge_cook_id = jn.cook_id
JOIN ContestantNames cn ON tr.contestant_cook_id = cn.cook_id
ORDER BY tr.total_rating DESC
LIMIT 5;
```

3.12. Ποιο ήταν το πιο τεχνικά δύσκολο, από πλευράς συνταγών, επεισόδιο του διαγωνισμού ανά έτος;

```
WITH EpisodeDifficulties AS (
    SELECT
        e.episode_id,
        e.name AS episode_name,
        e.season,
        AVG(r.difficulty) AS average_recipe_difficulty
    FROM Episode e
    JOIN Cook_Episode_Contestants cec ON e.episode_id = cec.episode_id
    JOIN Recipe r ON cec.recipe_id = r.recipe_id
    GROUP BY e.episode_id, e.name, e.season
),
MaxDifficultiesPerSeason AS (
```

```
    SELECT
        season,
        MAX(average_recipe_difficulty) AS max_difficulty
    FROM EpisodeDifficulties
    GROUP BY season
)
SELECT
    ed.season,
    ed.episode_name,
    ed.episode_id,
    ed.average_recipe_difficulty
FROM EpisodeDifficulties ed
JOIN MaxDifficultiesPerSeason mds ON ed.season = mds.season AND
ed.average_recipe_difficulty = mds.max_difficulty
ORDER BY ed.season, ed.average_recipe_difficulty DESC;
```

3.13. Ποιο επεισόδιο συγκέντρωσε τον χαμηλότερο βαθμό επαγγελματικής κατάρτισης (κριτές και μάγειρες);

```
WITH CookRankValues AS (
    SELECT
        cej.episode_id,
        CASE
            WHEN c.ranking = 'chef' THEN 5
            WHEN c.ranking = 'sous chef' THEN 4
            WHEN c.ranking = 'cook A' THEN 3
            WHEN c.ranking = 'cook B' THEN 2
            WHEN c.ranking = 'cook C' THEN 1
            ELSE 0
        END AS ranking_value
    FROM Cook_Episode_Judge cej
    JOIN Cook c ON cej.cook_id = c.cook_id
    UNION ALL
    SELECT
        cec.episode_id,
        CASE
            WHEN c.ranking = 'chef' THEN 5
            WHEN c.ranking = 'sous chef' THEN 4
            WHEN c.ranking = 'cook A' THEN 3
            WHEN c.ranking = 'cook B' THEN 2
            WHEN c.ranking = 'cook C' THEN 1
            ELSE 0
        END AS ranking_value
    FROM Cook_Episode_Contestants cec
    JOIN Cook c ON cec.cook_id = c.cook_id
),
EpisodeRankings AS (
    SELECT
        episode_id,
        AVG(ranking_value) AS average_ranking
    FROM CookRankValues
    GROUP BY episode_id
)
SELECT
    e.episode_id,
```

```
    e.name AS episode_name,
    e.season,
    er.average_ranking
FROM EpisodeRankings er
JOIN Episode e ON er.episode_id = e.episode_id
ORDER BY er.average_ranking ASC
LIMIT 1;
```

3.14. Ποια θεματική ενότητα έχει εμφανιστεί τις περισσότερες φορές στο διαγωνισμό;

```
SELECT rt.topic_id, t.name AS topic_name, COUNT(cec.episode_id) AS appearances
FROM Recipe_Topic rt
JOIN Topic t ON rt.topic_id = t.topic_id
JOIN Cook_Episode_Contestants cec ON cec.recipe_id = rt.recipe_id
GROUP BY rt.topic_id, t.name
ORDER BY appearances DESC
LIMIT 1;
```

3.15. Ποιες ομάδες τροφίμων δεν έχουν εμφανιστεί ποτέ στον διαγωνισμό;

```
WITH UsedFoodGroups AS (
    SELECT DISTINCT r.food_group_identity
    FROM Recipe r
    JOIN Cook_Episode_Contestants cec ON r.recipe_id = cec.recipe_id
)
SELECT fg.food_group_id, fg.name AS food_group_name
FROM Food_Group fg
LEFT JOIN UsedFoodGroups ufg ON fg.group_identity = ufg.food_group_identity
WHERE ufg.food_group_identity IS NULL;
```

Database Guide

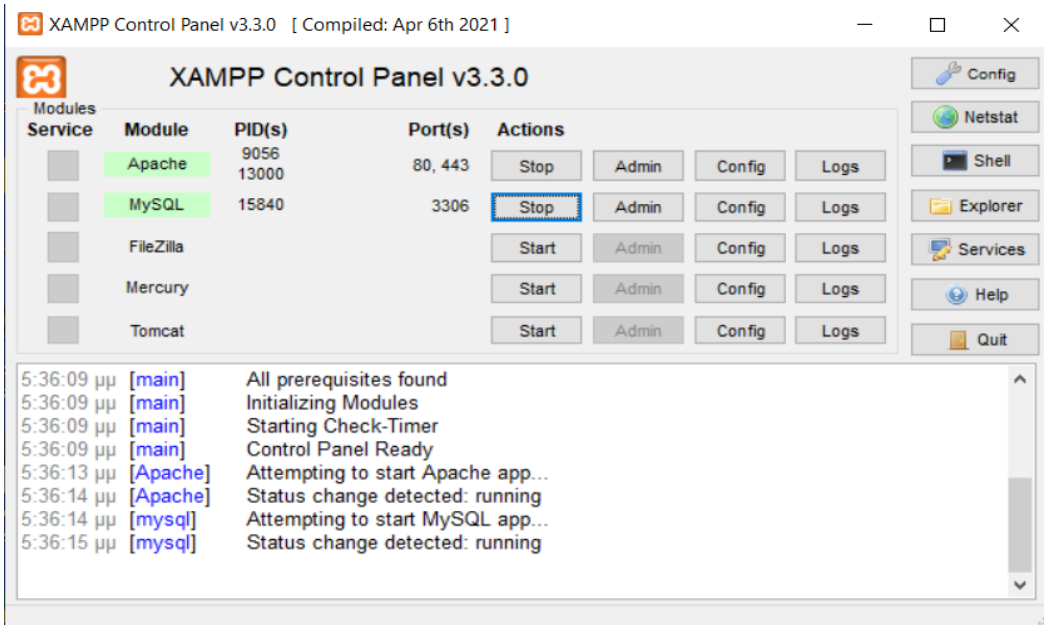
Στη συνέχεια θα δείξουμε αναλυτικά τον τρόπο με τον οποίο μπορεί να γίνει η εγκατάσταση και η λειτουργία της βάσης στο προσωπικό μας υπολογιστή. Ο οδηγός που ακολουθεί προορίζεται για λειτουργικό Windows.

1) Κατεβάστε το XAMPP αν δεν το έχετε ήδη κατεβάσει από στον υπολογιστή σας. Για περισσότερες πληροφορίες ακολουθήστε τον ακόλουθο σύνδεσμο: https://www.apachefriends.org/download.html

2) Επιλέξτε τον παρακάτω σύνδεσμο που θα σας προωθήσει στον φάκελο SQL_Code του git repo μας, όπου μπορείτε να κατεβάσετε τα αρχεία cooking_contest_db.sql και insert_data.sql:

https://github.com/nikolasbv/Cooking_Contest_DB/tree/main/SQL_Code

3) Ανοίξτε το Control Panel του XAMPP και ξεκινήστε τα modules "MySQL" και "Apache". Όπως φαίνεται στην παρακάτω εικόνα.



4) Ανοίξτε ένα terminal και μεταφερθείτε στο dir /path/to/xampp/mysql/bin χρησιμοποιώντας την εντολή cd.Στην περίπτωση που χρησιμοποιήσατε τις προτεινόμενες ρυθμίσεις κατά την εγκατάσταση τότε το ζητούμενο directory θα βρίσκεται στο C:\xampp\mysql\bin και επομένως αρκεί να εκτελέσετε στο terminal την εντολή:

```
cd C:/xampp/mysql/bin
```

5) Συνδεθείτε στο root χρήστη εκτελόντας την παρακάτω εντολή και πατώντας Enter:

```
mysql -u root -p
```

6) Στη συνέχεια, εισάγετε το password του root user σας, ενώ στην περίπτωση που δεν έχετε αλλάξει τον κωδικό πατήστε απλά Enter.

7) Εισάγετε τα αρχεία cooking_contest_db.sql και insert_data.sql που κατεβάσατε νωρίτερα στην mysql,

είτε εκτελώντας:

```
source /path/to/cooking_contest_db.sql
source /path/to/insert_data.sql
```

είτε αντιγράφοντας το περιεχόμενο του κάθε αρχείου στο cli της mysql στο οποίο βρίσκεστε.

8) Στην περίπτωση που επιθυμείτε να πραγματοποιήσετε κλήρωση διαγωνισμού θα χρειαστεί να τρέξετε το αρχείο episode_creator.py, το οποίο βρίσκεται στον παρακάτω σύνδεσμο του git repo και για την εκτέλεση του χρειάζεται να έχετε τα requirements που αναγράφονται στο requirements.txt επίσης στο git repo:

episode_creator.py:

https://github.com/nikolasbv/Cooking_Contest_DB/blob/main/Data_Generation/episode_creator.py

requirements.txt:

https://github.com/nikolasbv/Cooking_Contest_DB/blob/main/requirements.txt

**Σημείωση: Προκειμένου να λειτουργήσει σωστά το episode_creator.py (αλλά και το databases_data_generator.ipynb) πρέπει είτε να αλλαχθούν κατάλληλα τα file paths που χρησιμοποιούνται μέσα στο αρχείο, είτε να γίνει clone το git repo του συγκεκριμένου project και να εκτελεστούν τα αντίστοιχα αρχεία στους φακέλους που ήδη βρίσκονται.*

Τέλος, ακολουθεί ολόκληρο το DDL script όπως αναφέρθηκε παραπάνω στην αναφορά.

FULL DDL

```sql
DROP DATABASE contest;
CREATE DATABASE IF NOT EXISTS contest;
USE contest;

CREATE TABLE IF NOT EXISTS Image (
    image_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    image_url VARCHAR(255) NOT NULL,
    description VARCHAR(255) NOT NULL,
    PRIMARY KEY (image_id)
);

CREATE TABLE IF NOT EXISTS Nationality (
    nationality_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (nationality_id),
    CONSTRAINT fk_nationality_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe (
    recipe_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    recipe_type ENUM("pastry", "cooking") NOT NULL,
    difficulty INT NOT NULL,
    name VARCHAR(50) NOT NULL,
    description TEXT NOT NULL,
    preparation_time INT UNSIGNED NOT NULL,
    execution_time INT UNSIGNED NOT NULL,
    portions INT UNSIGNED NOT NULL,
    fat_grams_per_portion DECIMAL(7,2) UNSIGNED,
    protein_grams_per_portion DECIMAL(7,2) UNSIGNED,
    curbs_grams_per_portion DECIMAL(7,2) UNSIGNED,
    kcal_per_portion DECIMAL(7,2) UNSIGNED NOT NULL DEFAULT 0,
    image_id INT UNSIGNED NOT NULL,
    nationality_id INT UNSIGNED NOT NULL,
    food_group_identity VARCHAR(50),
    PRIMARY KEY (recipe_id),
    CONSTRAINT fk_recipe_nationality
        FOREIGN KEY (nationality_id) REFERENCES Nationality(nationality_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_recipe_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT CHK_difficulty
        CHECK (difficulty BETWEEN 1 AND 5),
    CONSTRAINT CHK_portions
        CHECK (portions > 0)
);

CREATE TABLE IF NOT EXISTS Step (
    recipe_id INT UNSIGNED NOT NULL,
    step_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name TEXT NOT NULL,
```

```
    sequence INT UNSIGNED NOT NULL,
    PRIMARY KEY (step_id, recipe_id),
    CONSTRAINT fk_step_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Tag (
    tag_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    PRIMARY KEY (tag_id)
);

CREATE TABLE IF NOT EXISTS Tip (
    tip_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    PRIMARY KEY (tip_id)
);

CREATE TABLE IF NOT EXISTS Meal_type (
    meal_type_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    PRIMARY KEY (meal_type_id)
);

CREATE TABLE IF NOT EXISTS Recipe_Tag (
    recipe_id INT UNSIGNED NOT NULL,
    tag_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (recipe_id, tag_id),
    CONSTRAINT fk_tag_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_recipe_tag
        FOREIGN KEY (tag_id) REFERENCES Tag(tag_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Tip (
    recipe_id INT UNSIGNED NOT NULL,
    tip_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (recipe_id, tip_id),
    CONSTRAINT fk_tip_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_recipe_tip
        FOREIGN KEY (tip_id) REFERENCES Tip(tip_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Meal_Type (
    recipe_id INT UNSIGNED NOT NULL,
    meal_type_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (recipe_id, meal_type_id),
    CONSTRAINT fk_meal_type_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    CONSTRAINT fk_recipe_meal_type
        FOREIGN KEY (meal_type_id) REFERENCES Meal_type(meal_type_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Food_Group (
    food_group_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    description VARCHAR(255) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    group_identity VARCHAR(50) NOT NULL,
    PRIMARY KEY (food_group_id),
    CONSTRAINT fk_food_group_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Ingredient (
    ingredient_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    kcal_per_100 DECIMAL(7,2) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    food_group_id INT UNSIGNED NOT NULL,
    avg_grams INT UNSIGNED,
    PRIMARY KEY (ingredient_id),
    CONSTRAINT fk_ingredient_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_ingredient_food_group
        FOREIGN KEY (food_group_id) REFERENCES Food_Group(food_group_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Ingredient (
    recipe_id INT UNSIGNED NOT NULL,
    ingredient_id INT UNSIGNED NOT NULL,
    basic_ingredient BOOLEAN NOT NULL DEFAULT FALSE,
    quantity_type ENUM('grams', 'serving', 'non_numeric') NOT NULL,
    quantity VARCHAR(50) NOT NULL,
    serving_type VARCHAR(50),
    PRIMARY KEY (recipe_id, ingredient_id),
    CONSTRAINT fk1_ingredient_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_recipe_ingredient
        FOREIGN KEY (ingredient_id) REFERENCES Ingredient(ingredient_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Equipment (
    equipment_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    manual TEXT NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (equipment_id),
    CONSTRAINT fk_equipment_image
```

```
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Equipment (
    recipe_id INT UNSIGNED NOT NULL,
    equipment_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (equipment_id, recipe_id),
    CONSTRAINT fk1_equipment_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_recipe_equipment
        FOREIGN KEY (equipment_id) REFERENCES Equipment(equipment_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Topic (
    topic_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    description VARCHAR(50) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (topic_id),
    CONSTRAINT fk_topic_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Topic (
    recipe_id INT UNSIGNED NOT NULL,
    topic_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (topic_id, recipe_id),
    CONSTRAINT fk1_topic_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_recipe_topic
        FOREIGN KEY (topic_id) REFERENCES Topic(topic_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Administrator (
    admin_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(50) NOT NULL,
    PRIMARY KEY (admin_id)
);

CREATE TABLE IF NOT EXISTS Cook (
    cook_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    contact_number VARCHAR(10) NOT NULL,
    date_of_birth DATE NOT NULL,
    years_of_experience INT NOT NULL,
    ranking ENUM("chef", "sous chef", "cook A", "cook B", "cook C") NOT NULL,
```

```
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(50) NOT NULL,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (cook_id),
    CONSTRAINT fk_cook_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Nationality_Cook (
    nationality_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (nationality_id, cook_id),
    CONSTRAINT fk1_cook_nationality
        FOREIGN KEY (nationality_id) REFERENCES Nationality(nationality_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_nationality_cook
        FOREIGN KEY (cook_id) REFERENCES Cook(cook_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Recipe_Cook (
    recipe_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (recipe_id, cook_id),
    CONSTRAINT fk1_cook_recipe
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_recipe_cook
        FOREIGN KEY (cook_id) REFERENCES Cook(cook_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Episode (
    episode_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name INT NOT NULL,
    season INT NOT NULL,
    winner INT UNSIGNED,
    image_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (episode_id),
    CONSTRAINT fk_episode_image
        FOREIGN KEY (image_id) REFERENCES Image(image_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT CHK_episode_number
        CHECK (name BETWEEN 1 AND 10)
);

CREATE TABLE IF NOT EXISTS Nationality_Episode (
    nationality_id INT UNSIGNED NOT NULL,
    episode_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (nationality_id, episode_id),
    CONSTRAINT fk1_episode_nationality
        FOREIGN KEY (nationality_id) REFERENCES Nationality(nationality_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_nationality_episode
        FOREIGN KEY (episode_id) REFERENCES Episode(episode_id)
```

```
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Cook_Episode_Judge (
    episode_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    judge_number INT NOT NULL,
    PRIMARY KEY (episode_id, cook_id),
    CONSTRAINT fk1_judge_episode
        FOREIGN KEY (episode_id) REFERENCES Episode(episode_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_episode_judge
        FOREIGN KEY (cook_id) REFERENCES Cook(cook_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS Cook_Episode_Contestants (
    episode_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    recipe_id INT UNSIGNED NOT NULL,
    rating_1 INT NOT NULL,
    rating_2 INT NOT NULL,
    rating_3 INT NOT NULL,
    PRIMARY KEY (episode_id, cook_id, recipe_id),
    CONSTRAINT fk1_cook_episode
        FOREIGN KEY (episode_id) REFERENCES Episode(episode_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk2_episode_cook
        FOREIGN KEY (cook_id) REFERENCES Cook(cook_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk3_recipe_episode_cook
        FOREIGN KEY (recipe_id) REFERENCES Recipe(recipe_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE INDEX idx_recipe_on_nationality_id_recipe_id ON Recipe(nationality_id, recipe_id);
CREATE INDEX idx_cec_on_episode_id_recipe_id ON Cook_Episode_Contestants(episode_id,
recipe_id);
CREATE INDEX idx_episode_on_episode_id_season ON Episode(episode_id, season);
CREATE INDEX idx_cec_on_cook_id_recipe_id ON Cook_Episode_Contestants(cook_id, recipe_id);
CREATE INDEX idx_cej_on_cook_id_episode_id ON Cook_Episode_Judge(cook_id, episode_id);
CREATE INDEX idx_re_on_recipe_id_equipment_id ON Recipe_Equipment(recipe_id, equipment_id);
CREATE INDEX idx_ne_on_episode_id_nationality_id ON Nationality_Episode(episode_id,
nationality_id);
CREATE INDEX idx_recipe_on_recipe_id_difficulty ON Recipe(recipe_id, difficulty);
CREATE INDEX idx_cej_on_episode_id_cook_id ON Cook_Episode_Judge(episode_id, cook_id);


DROP ROLE 'Cook_User';
DROP ROLE 'Administrator';

CREATE ROLE "Cook_User";
CREATE ROLE "Administrator";

GRANT ALL PRIVILEGES ON contest.* TO "Administrator";
```

```
GRANT SELECT, INSERT, UPDATE ON contest.Recipe TO "Cook_User";
GRANT SELECT, UPDATE ON contest.Cook TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Recipe TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Step TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Meal_type TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Tip TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Tag TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Meal_type TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Tip TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Tag TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Recipe_Cook TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Equipment TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Ingredient TO "Cook_User";
GRANT SELECT, INSERT, UPDATE, DELETE ON contest.Recipe_Topic TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Image TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Nationality TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Ingredient TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Equipment TO "Cook_User";
GRANT SELECT, INSERT, UPDATE ON contest.Topic TO "Cook_User";
GRANT SELECT ON contest.Cook_Episode_Contestants TO "Cook_User";
GRANT SELECT ON contest.Episode TO "Cook_User";


DELIMITER //

DROP TRIGGER IF EXISTS check_tip_limit_before_insert;
DROP TRIGGER IF EXISTS add_step_in_middle_of_recipe;
DROP TRIGGER IF EXISTS food_group_identity_on_recipe;
DROP TRIGGER IF EXISTS food_group_identity_on_recipe_update;
DROP TRIGGER IF EXISTS pick_winner_in_episode;
DROP TRIGGER IF EXISTS match_season_with_unique_episode;
DROP TRIGGER IF EXISTS nationality_episode_regulator;
DROP TRIGGER IF EXISTS contestant_episode_regulator;
DROP TRIGGER IF EXISTS calculate_kcal_per_portion_dynamically;
DROP TRIGGER IF EXISTS update_kcal_per_portion_dynamically;
DROP TRIGGER IF EXISTS delete_kcal_per_portion_dynamically;
DROP TRIGGER IF EXISTS update_kcal_on_ingredient_kcal_change;
DROP TRIGGER IF EXISTS update_kcal_on_portions_change;
DROP TRIGGER IF EXISTS three_different_judges_per_episode;
DROP TRIGGER IF EXISTS check_consecutive_episodes_per_contestant;
DROP TRIGGER IF EXISTS check_consecutive_episodes_per_judge;
DROP TRIGGER IF EXISTS check_consecutive_episodes_per_nationality;
DROP TRIGGER IF EXISTS check_consecutive_episodes_per_recipe;
DROP TRIGGER IF EXISTS link_recipe_to_cook_after_insert;
DROP TRIGGER IF EXISTS check_user_before_edit_on_cook;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_ingredient;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_ingredient;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_equipment;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_equipment;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_topic;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_topic;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_tag;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_tag;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_tip;
```

```
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_tip;
DROP TRIGGER IF EXISTS check_user_before_add_on_recipe_meal_type;
DROP TRIGGER IF EXISTS check_user_before_edit_on_recipe_meal_type;
DROP TRIGGER IF EXISTS check_user_before_add_on_step;
DROP TRIGGER IF EXISTS check_user_before_edit_on_step;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_ingredient;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_equipment;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_topic;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_tag;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_tip;
DROP TRIGGER IF EXISTS check_user_before_delete_on_recipe_meal_type;
DROP TRIGGER IF EXISTS check_user_before_delete_on_step;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_tip_limit_before_insert
BEFORE INSERT ON Recipe_Tip
FOR EACH ROW
BEGIN
    DECLARE tip_count INT;
    SELECT COUNT(*) INTO tip_count FROM Recipe_Tip WHERE recipe_id = NEW.recipe_id;
    IF tip_count >= 3 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot add more than 3 tips per
recipe';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER add_step_in_middle_of_recipe
BEFORE INSERT ON Step
FOR EACH ROW
BEGIN
    IF (SELECT NEW.sequence-MAX(sequence) FROM Step WHERE NEW.recipe_id = recipe_id)>1
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot skip step in recipe';
        END IF;
    IF (SELECT sequence FROM Step WHERE recipe_id = NEW.recipe_id AND sequence =
NEW.sequence)
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This step already exists';
        END IF;
END;
//

DELIMITER ;

DELIMITER //

CREATE TRIGGER delete_step_in_middle_of_recipe
```

```
BEFORE DELETE ON Step
FOR EACH ROW
BEGIN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete existing step';
END;
//

DELIMITER ;

DELIMITER //

CREATE TRIGGER food_group_identity_on_recipe
BEFORE INSERT ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE temp_food_group_id INT;
    DECLARE temp_food_group VARCHAR(50);
    IF NEW.basic_ingredient
        THEN
        IF EXISTS (SELECT basic_ingredient FROM Recipe_Ingredient
        WHERE recipe_id  = NEW.recipe_id and basic_ingredient = TRUE)
            THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This recipe already has a
basic ingredient';
        ELSE
            SELECT food_group_id INTO temp_food_group_id FROM Ingredient
            WHERE ingredient_id = NEW.ingredient_id;
            SELECT group_identity INTO temp_food_group FROM Food_Group
            WHERE food_group_id = temp_food_group_id;

            UPDATE Recipe
            SET food_group_identity = temp_food_group
            WHERE recipe_id = NEW.recipe_id;
        END IF;
    END IF;
END;
//

DELIMITER ;

DELIMITER //

CREATE TRIGGER food_group_identity_on_recipe_update
BEFORE UPDATE ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE prev_basic_ingredient INT DEFAULT NULL;
    IF NEW.basic_ingredient THEN

        SELECT ingredient_id INTO prev_basic_ingredient FROM Recipe_Ingredient
        WHERE recipe_id  = NEW.recipe_id and basic_ingredient = TRUE;

        IF prev_basic_ingredient THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This recipe already has a basic
ingredient';
        END IF;
    END IF;
```

```
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER pick_winner_in_episode
AFTER INSERT ON Cook_Episode_Contestants
FOR EACH ROW
BEGIN
    DECLARE current_winner INT;
    DECLARE total_rating INT;

    SELECT Cook_Episode_Contestants.cook_id, SUM(rating_1 + rating_2 + rating_3)
    INTO current_winner, total_rating
    FROM Cook_Episode_Contestants
    INNER JOIN Cook ON Cook.cook_id = Cook_Episode_Contestants.cook_id
    WHERE episode_id = NEW.episode_id
    GROUP BY Cook_Episode_Contestants.cook_id
    ORDER BY SUM(rating_1 + rating_2 + rating_3) DESC, Cook.ranking DESC
    LIMIT 1;

    UPDATE Episode
    SET winner = current_winner
    WHERE episode_id = NEW.episode_id;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER match_season_with_unique_episode
BEFORE INSERT ON Episode
FOR EACH ROW
BEGIN
    DECLARE episode_count INT ;
    IF EXISTS (
        SELECT 1 FROM Episode
        WHERE season = NEW.season
        AND name = NEW.name
    ) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot add episode with duplicate name
in this season';
    END IF;
        SELECT count(*) INTO episode_count FROM Episode WHERE season = NEW.season;
    IF episode_count >=10 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot add another episode in this
season';
    END IF;

END;
//
```

```
DELIMITER ;


DELIMITER //

CREATE TRIGGER nationality_episode_regulator
BEFORE INSERT ON Nationality_Episode
FOR EACH ROW
BEGIN
    DECLARE num_of_nationalities INT;
    SELECT COUNT(*) INTO num_of_nationalities
    FROM Nationality_Episode
    WHERE episode_id = NEW.episode_id;
    IF num_of_nationalities >= 10
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Episode already has maximum
Nationalities';
    END IF;

    IF EXISTS(SELECT nationality_id FROM Nationality_Episode
    WHERE episode_id = NEW.episode_id AND nationality_id = NEW.nationality_id)
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Nationality already in this Episode';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER contestant_episode_regulator
BEFORE INSERT ON Cook_Episode_Contestants
FOR EACH ROW
BEGIN
    DECLARE num_of_contestants INT;
    SELECT COUNT(*) INTO num_of_contestants
    FROM Cook_Episode_Contestants
    WHERE episode_id = NEW.episode_id;
    IF num_of_contestants >= 10
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Episode already has maximum
Contestants';
    END IF;

    IF EXISTS (SELECT cook_id FROM Cook_Episode_Contestants
    WHERE episode_id = NEW.episode_id AND cook_id = NEW.cook_id)
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Contestant already in this Episode';
    END IF;

    IF EXISTS(SELECT cook_id FROM Cook_Episode_Judge
    WHERE cook_id = NEW.cook_id AND episode_id = NEW.episode_id)
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cook is a Judge in this Episode';
    END IF;

END;
//
```

```
DELIMITER ;


DELIMITER //

CREATE TRIGGER calculate_kcal_per_portion_dynamically
AFTER INSERT ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE temp_total_kcal DECIMAL(7,2) DEFAULT 0;
    DECLARE temp_portions INT;
    DECLARE temp_avg_grams INT;
    DECLARE temp_kcal_per_100 INT;
    DECLARE converted_quantity INT;

    SELECT portions INTO temp_portions FROM Recipe WHERE recipe_id = NEW.recipe_id;
    SELECT avg_grams, kcal_per_100 INTO temp_avg_grams, temp_kcal_per_100
    FROM Ingredient WHERE ingredient_id = NEW.ingredient_id;

    IF NEW.quantity_type IN ('grams', 'serving') THEN
        SET converted_quantity = CAST(NEW.quantity AS UNSIGNED);

        IF NEW.quantity_type = 'grams' THEN
            SET temp_total_kcal = (converted_quantity / 100) * temp_kcal_per_100 /
temp_portions;
        ELSEIF NEW.quantity_type = 'serving' THEN
            SET temp_total_kcal = ((temp_avg_grams * converted_quantity / 100) *
temp_kcal_per_100) / temp_portions;
        END IF;

        UPDATE Recipe
        SET kcal_per_portion = kcal_per_portion + temp_total_kcal
        WHERE recipe_id = NEW.recipe_id;
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER update_kcal_per_portion_dynamically
AFTER UPDATE ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE old_total_kcal DECIMAL(7,2) DEFAULT 0;
    DECLARE new_total_kcal DECIMAL(7,2) DEFAULT 0;
    DECLARE temp_portions INT;
    DECLARE temp_avg_grams INT;
    DECLARE temp_kcal_per_100 INT;
    DECLARE old_converted_quantity INT;
    DECLARE new_converted_quantity INT;

    IF OLD.quantity <> NEW.quantity OR OLD.quantity_type <> NEW.quantity_type THEN
        SELECT portions INTO temp_portions FROM Recipe WHERE recipe_id = OLD.recipe_id;
```

```
        SELECT avg_grams, kcal_per_100 INTO temp_avg_grams, temp_kcal_per_100
        FROM Ingredient WHERE ingredient_id = OLD.ingredient_id;

        IF OLD.quantity_type IN ('grams', 'serving') THEN
            SET old_converted_quantity = CAST(OLD.quantity AS UNSIGNED);

            IF OLD.quantity_type = 'grams' THEN
                SET old_total_kcal = (old_converted_quantity / 100) * temp_kcal_per_100 /
temp_portions;
            ELSEIF OLD.quantity_type = 'serving' THEN
                SET old_total_kcal = ((temp_avg_grams * old_converted_quantity / 100) *
temp_kcal_per_100) / temp_portions;
            END IF;
        END IF;

        IF NEW.quantity_type IN ('grams', 'serving') THEN
            SET new_converted_quantity = CAST(NEW.quantity AS UNSIGNED);

            IF NEW.quantity_type = 'grams' THEN
                SET new_total_kcal = (new_converted_quantity / 100) * temp_kcal_per_100 /
temp_portions;
            ELSEIF NEW.quantity_type = 'serving' THEN
                SET new_total_kcal = ((temp_avg_grams * new_converted_quantity / 100) *
temp_kcal_per_100) / temp_portions;
            END IF;
        END IF;

        UPDATE Recipe
        SET kcal_per_portion = kcal_per_portion - old_total_kcal + new_total_kcal
        WHERE recipe_id = OLD.recipe_id;
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER delete_kcal_per_portion_dynamically
AFTER DELETE ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE deleted_kcal DECIMAL(7,2) DEFAULT 0;
    DECLARE temp_portions INT;
    DECLARE temp_avg_grams INT;
    DECLARE temp_kcal_per_100 INT;
    DECLARE converted_quantity INT;

    SELECT portions INTO temp_portions FROM Recipe WHERE recipe_id = OLD.recipe_id;
    SELECT avg_grams, kcal_per_100 INTO temp_avg_grams, temp_kcal_per_100
    FROM Ingredient WHERE ingredient_id = OLD.ingredient_id;

    IF OLD.quantity_type IN ('grams', 'serving') THEN

        SET converted_quantity = CAST(OLD.quantity AS UNSIGNED);
```

```
        IF OLD.quantity_type = 'grams' THEN
            SET deleted_kcal = (converted_quantity / 100) * temp_kcal_per_100 /
temp_portions;
        ELSEIF OLD.quantity_type = 'serving' THEN
            SET deleted_kcal = ((temp_avg_grams * converted_quantity / 100) *
temp_kcal_per_100) / temp_portions;
        END IF;

        UPDATE Recipe
        SET kcal_per_portion = kcal_per_portion - deleted_kcal
        WHERE recipe_id = OLD.recipe_id;
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER update_kcal_on_ingredient_kcal_change
AFTER UPDATE ON Ingredient
FOR EACH ROW
BEGIN
    DECLARE temp_recipe_id INT;
    DECLARE old_kcal DECIMAL(10,2) DEFAULT 0;
    DECLARE new_kcal DECIMAL(10,2) DEFAULT 0;
    DECLARE temp_quantity INT;
    DECLARE temp_avg_grams INT;
    DECLARE temp_portions INT;
    DECLARE done BOOLEAN DEFAULT FALSE;
    DECLARE quantity_type ENUM('grams', 'serving', 'non_numeric');

    DECLARE cur CURSOR FOR
        SELECT recipe_id, Recipe_Ingredient.quantity, avg_grams, quantity_type
        FROM Recipe_Ingredient
        JOIN Ingredient ON Recipe_Ingredient.ingredient_id = Ingredient.ingredient_id
        WHERE Ingredient.ingredient_id = NEW.ingredient_id AND quantity_type IN ('grams',
'serving');

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    IF OLD.kcal_per_100 <> NEW.kcal_per_100 THEN
        OPEN cur;

        recipe_loop: LOOP
            FETCH cur INTO temp_recipe_id, temp_quantity, temp_avg_grams, quantity_type;
            IF done THEN
                LEAVE recipe_loop;
            END IF;

            SELECT portions INTO temp_portions FROM Recipe WHERE recipe_id =
temp_recipe_id;

            IF quantity_type = 'grams' THEN
```

```
                SET old_kcal = (CAST(temp_quantity AS UNSIGNED) / 100) * OLD.kcal_per_100 /
temp_portions;
                SET new_kcal = (CAST(temp_quantity AS UNSIGNED) / 100) * NEW.kcal_per_100 /
temp_portions;
            ELSEIF quantity_type = 'serving' THEN
                SET old_kcal = ((temp_avg_grams * CAST(temp_quantity AS UNSIGNED) / 100) *
OLD.kcal_per_100) / temp_portions;
                SET new_kcal = ((temp_avg_grams * CAST(temp_quantity AS UNSIGNED) / 100) *
NEW.kcal_per_100) / temp_portions;
            END IF;

            UPDATE Recipe
            SET kcal_per_portion = kcal_per_portion - old_kcal + new_kcal
            WHERE recipe_id = temp_recipe_id;

        END LOOP;

        CLOSE cur;
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER restrict_update_kcal_on_portions_change
BEFORE UPDATE ON Recipe
FOR EACH ROW
BEGIN
    IF OLD.portions <> NEW.portions
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot change recipe portions';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER three_different_judges_per_episode
BEFORE INSERT ON Cook_Episode_Judge
FOR EACH ROW
BEGIN
        DECLARE temp_judge_count INT;
        DECLARE temp_judge_number INT;
        IF EXISTS(SELECT cook_id FROM Cook_Episode_Judge
        WHERE cook_id = NEW.cook_id AND episode_id = NEW.episode_id)
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cook is already a Judge in this
Episode' ;
        END IF;
         IF EXISTS(SELECT cook_id FROM Cook_Episode_Contestants
         WHERE cook_id = NEW.cook_id AND episode_id = NEW.episode_id)
```

```
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cook already competes in this
Episode' ;
       END IF;
      SELECT COUNT(*) INTO temp_judge_count
      FROM Cook_Episode_Judge
      WHERE episode_id = NEW.episode_id;
      IF temp_judge_count >= 3
        THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Episode already has maximum
Judges';
      END IF;


       SET temp_judge_number = temp_judge_count + 1;
       SET NEW.judge_number = temp_judge_number ;

    END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_consecutive_episodes_per_contestant
BEFORE INSERT ON Cook_Episode_Contestants
FOR EACH ROW
BEGIN
    DECLARE prev_episode_id INT DEFAULT 0;
    DECLARE second_last_episode_id INT DEFAULT 0;
    DECLARE third_last_episode_id INT DEFAULT 0;
    DECLARE count_episodes INT DEFAULT 0;

    SELECT episode_id INTO prev_episode_id
    FROM Cook_Episode_Contestants
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 0;

    SELECT episode_id INTO second_last_episode_id
    FROM Cook_Episode_Contestants
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 1;

    SELECT episode_id INTO third_last_episode_id
    FROM Cook_Episode_Contestants
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 2;

    SELECT COUNT(*) INTO count_episodes
    FROM Cook_Episode_Contestants
    WHERE cook_id = NEW.cook_id;

    IF (count_episodes >= 3 AND NEW.episode_id = prev_episode_id + 1 AND prev_episode_id =
second_last_episode_id + 1 AND second_last_episode_id = third_last_episode_id + 1) THEN
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cook cannot participate in more than 3
consecutive episodes as a contestant';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_consecutive_episodes_per_judge
BEFORE INSERT ON Cook_Episode_Judge
FOR EACH ROW
BEGIN
    DECLARE prev_episode_id INT DEFAULT 0;
    DECLARE second_last_episode_id INT DEFAULT 0;
    DECLARE third_last_episode_id INT DEFAULT 0;
    DECLARE count_episodes INT DEFAULT 0;

    SELECT episode_id INTO prev_episode_id
    FROM Cook_Episode_Judge
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 0;

    SELECT episode_id INTO second_last_episode_id
    FROM Cook_Episode_Judge
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 1;

    SELECT episode_id INTO third_last_episode_id
    FROM Cook_Episode_Judge
    WHERE cook_id = NEW.cook_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 2;

    SELECT COUNT(*) INTO count_episodes
    FROM Cook_Episode_Judge
    WHERE cook_id = NEW.cook_id;

    IF (count_episodes >= 3 AND NEW.episode_id = prev_episode_id + 1 AND prev_episode_id =
second_last_episode_id + 1 AND second_last_episode_id = third_last_episode_id + 1) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cook cannot participate in more than 3
consecutive episodes as a judge';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_consecutive_episodes_per_nationality
```

```
BEFORE INSERT ON Nationality_Episode
FOR EACH ROW
BEGIN
    DECLARE prev_episode_id INT DEFAULT 0;
    DECLARE second_last_episode_id INT DEFAULT 0;
    DECLARE third_last_episode_id INT DEFAULT 0;
    DECLARE count_episodes INT DEFAULT 0;

    SELECT episode_id INTO prev_episode_id
    FROM Nationality_Episode
    WHERE nationality_id = NEW.nationality_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 0;

    SELECT episode_id INTO second_last_episode_id
    FROM Nationality_Episode
    WHERE nationality_id = NEW.nationality_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 1;

    SELECT episode_id INTO third_last_episode_id
    FROM Nationality_Episode
    WHERE nationality_id = NEW.nationality_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 2;

    SELECT COUNT(*) INTO count_episodes
    FROM Nationality_Episode
    WHERE nationality_id = NEW.nationality_id;

    IF (count_episodes >= 3 AND NEW.episode_id = prev_episode_id + 1 AND prev_episode_id =
second_last_episode_id + 1 AND second_last_episode_id = third_last_episode_id + 1) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Nationality cannot appear in more than
3 consecutive episodes';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_consecutive_episodes_per_recipe
BEFORE INSERT ON Cook_Episode_Contestants
FOR EACH ROW
BEGIN
    DECLARE prev_episode_id INT DEFAULT 0;
    DECLARE second_last_episode_id INT DEFAULT 0;
    DECLARE third_last_episode_id INT DEFAULT 0;
    DECLARE count_episodes INT DEFAULT 0;

    SELECT episode_id INTO prev_episode_id
    FROM Cook_Episode_Contestants
    WHERE recipe_id = NEW.recipe_id
    ORDER BY episode_id DESC
```

```
    LIMIT 1 OFFSET 0;

    SELECT episode_id INTO second_last_episode_id
    FROM Cook_Episode_Contestants
    WHERE recipe_id = NEW.recipe_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 1;

    SELECT episode_id INTO third_last_episode_id
    FROM Cook_Episode_Contestants
    WHERE recipe_id = NEW.recipe_id
    ORDER BY episode_id DESC
    LIMIT 1 OFFSET 2;

    SELECT COUNT(*) INTO count_episodes
    FROM Cook_Episode_Contestants
    WHERE recipe_id = NEW.recipe_id;

    IF (count_episodes >= 3 AND NEW.episode_id = prev_episode_id + 1 AND prev_episode_id =
second_last_episode_id + 1 AND second_last_episode_id = third_last_episode_id + 1) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Recipe cannot appear in more than 3
consecutive episodes';
    END IF;
END;
//

DELIMITER ;


DELIMITER //

CREATE TRIGGER link_recipe_to_cook_after_insert
AFTER INSERT ON Recipe
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);

    IF current_cook_id IS NOT NULL THEN
        INSERT INTO Recipe_Cook (recipe_id, cook_id)
        VALUES (NEW.recipe_id, current_cook_id);
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_cook
BEFORE UPDATE ON Cook
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
```

```
    DECLARE current_username VARCHAR(255);
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SET error_message = CONCAT("Unauthorized attempt to modify Cook with cook_id: ",
CAST(OLD.cook_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR));

        IF OLD.cook_id <> current_cook_id THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe
BEFORE UPDATE ON Recipe
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;
```

```
DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_ingredient
BEFORE UPDATE ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_recipe_ingredient
BEFORE INSERT ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
```

```
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_equipment
BEFORE UPDATE ON Recipe_Equipment
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //
```

```
CREATE TRIGGER check_user_before_add_on_recipe_equipment
BEFORE INSERT ON Recipe_Equipment
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_topic
BEFORE UPDATE ON Recipe_Topic
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;
```

```
        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                   ".\nAction attempted by user: ", current_username,
                                   " with cook_id: ", CAST(current_cook_id AS CHAR),
                                   ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_recipe_topic
BEFORE INSERT ON Recipe_Topic
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                   ".\nAction attempted by user: ", current_username,
                                   " with cook_id: ", CAST(current_cook_id AS CHAR),
                                   ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_tag
BEFORE UPDATE ON Recipe_Tag
FOR EACH ROW
```

```
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_recipe_tag
BEFORE INSERT ON Recipe_Tag
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
```

```
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_tip
BEFORE UPDATE ON Recipe_Tip
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_recipe_tip
BEFORE INSERT ON Recipe_Tip
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
```

```
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_recipe_meal_type
BEFORE UPDATE ON Recipe_Meal_Type
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
```

```
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_recipe_meal_type
BEFORE INSERT ON Recipe_Meal_Type
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_edit_on_step
BEFORE UPDATE ON Step
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
```

```
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_add_on_step
BEFORE INSERT ON Step
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = NEW.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(NEW.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //
```

```
DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_recipe_ingredient
BEFORE DELETE ON Recipe_Ingredient
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_recipe_equipment
BEFORE DELETE ON Recipe_Equipment
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
```

```
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_recipe_topic
BEFORE DELETE ON Recipe_Topic
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //
```

```
CREATE TRIGGER check_user_before_delete_on_recipe_tag
BEFORE DELETE ON Recipe_Tag
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_recipe_tip
BEFORE DELETE ON Recipe_Tip
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;
```

```
        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                   ".\nAction attempted by user: ", current_username,
                                   " with cook_id: ", CAST(current_cook_id AS CHAR),
                                   ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_recipe_meal_type
BEFORE DELETE ON Recipe_Meal_Type
FOR EACH ROW
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                   ".\nAction attempted by user: ", current_username,
                                   " with cook_id: ", CAST(current_cook_id AS CHAR),
                                   ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;


DELIMITER //

CREATE TRIGGER check_user_before_delete_on_step
BEFORE DELETE ON Step
FOR EACH ROW
```

```
BEGIN
    DECLARE current_cook_id INT;
    DECLARE current_username VARCHAR(255);
    DECLARE is_cook_linked BOOLEAN DEFAULT FALSE;
    DECLARE error_message VARCHAR(255);

    SELECT cook_id INTO current_cook_id FROM Cook WHERE username = SUBSTRING_INDEX(USER(),
'@', 1);
    SET current_username = SUBSTRING_INDEX(USER(), '@', 1);

    IF current_cook_id IS NOT NULL THEN
        SELECT EXISTS(
            SELECT 1 FROM Recipe_Cook
            WHERE cook_id = current_cook_id AND recipe_id = OLD.recipe_id
        ) INTO is_cook_linked;

        SET error_message = CONCAT("Unauthorized attempt to modify Recipe with recipe_id:
", CAST(OLD.recipe_id AS CHAR),
                                ".\nAction attempted by user: ", current_username,
                                " with cook_id: ", CAST(current_cook_id AS CHAR),
                                ". Cook not linked to this recipe.");

        IF NOT is_cook_linked THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = error_message;
        END IF;
    END IF;
END //

DELIMITER ;
```