Βάσεις Δεδομένων

Εργαστηριακό Project

Σύστημα Διαχείρισης Διαγωνισμού Μαγειρικής

Βέργαδου Παναγιώτα
el21186

Καραπιδάκης Γεώργιος
el21038

Μπίλιας Ευστάθιος
el20800

26 Μαΐου 2024

ΣΗΜΜΥ
Εθνικό Μετσόβιο Πολυτεχνείο

Περιεχόμενα

1 Εισαγωγή

Η παρούσα αναφορά αποτελεί κομμάτι της εξαμηνιαίας εργασία του μαθήματος ***Βάσεις Δεδομένων*** του 6ου εξαμήνου για το ακαδημαϊκό έτος 2023-2024.
Για το μάθημα της παρούσας χρονιάς, κληθήκαμε να υλοποιήσουμε μια βάση δεδομένων (ΒΔ), η οποία θα εξυπηρετεί έναν διαγωνισμό μαγειρικής, σύμφωνα με τους κανόνες που δοθηκαν στην εκφώνηση της άσκησης. Στην αναφορά περιλαμβάνονται η παράθεση και η αναλυτική περιγραφή των ER και σχεσιακών διαγραμμάτων, τα script DDL και DML για την σχεδίαση/setup και την ορθή λειτουργία της ΒΔ, αντίστοιχα και τα βήματα, οι τυχούσες απαιτούμενες βιβλιοθήκες για την εγκατάσταση της εφαρμογής μας εξαρχής, καθώς και σχολιασμός για την ευκολότερη κατανόηση των εκάστοτε παραδοχών ή μεθόδων που ακολουθήθηκαν για την καλύτερη δυνατή απάντηση των εκάστοτε ερωτημάτων. Παρατίθεται ο σύνδεσμος για το github repository[1], όπου έχουν συμπεριληφθεί όλα τα ζητούμενα, μαζί με τον πηγαίο κώδικα υλοποίησης της δομής και των ερωτημάτων.

2 «Στήσιμο» της Εφαρμογής

Κατά την σχεδίαση της εφαρμογής θέλαμε εύκολο deployment οπουδήποτε χωρίς να υπάρχουν προβλήματα με τις εκδόσεις της MySQL του καθενός. Για αυτό επιλέξαμε τη χρήση Docker Containers με την δυνατότητα επέκτασης αυτών για την υλοποίηση του application layer. Το παράδειγμα που παρέχουμε έχει τα απαραίτητα containers για έναν Apache Web Server με PHP και MySQL. Εμείς στα πλαίσια της άσκησης χρειαζόμαστε μόνο τη βάση δεδομένων MySQL. Για να στηθεί το σύστημα αρκεί κανείς μέσα στο directory του repository που κατέβασε να τρέξει τις παρακάτω εντολές στο τερματικό του εφόσον πρώτα έχει εγκαταστήσει το docker και το docker compose.

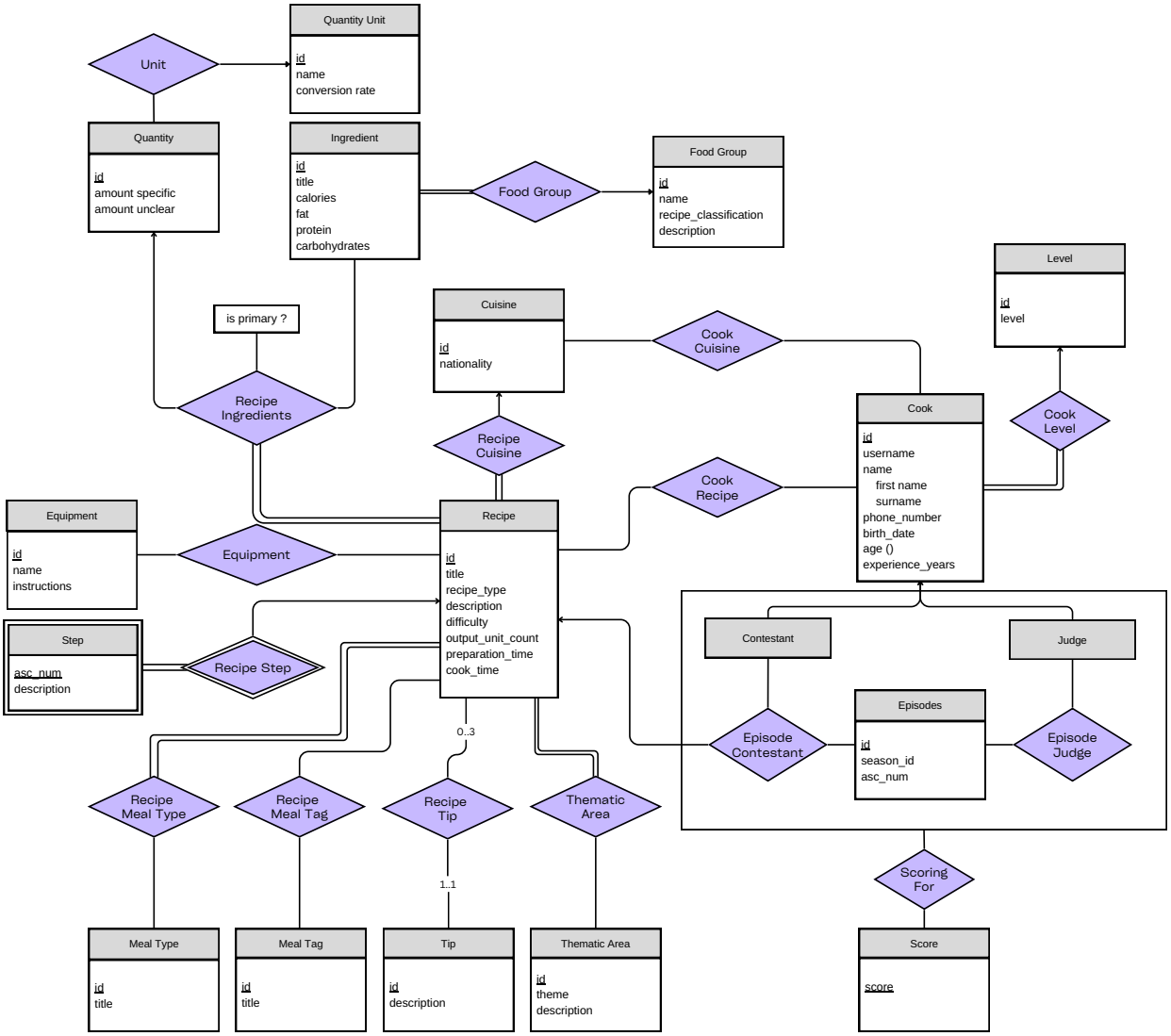
```
$ docker compose build
$ docker compose up -d
```

Σε αυτό το σημείο έχει δημιουργηθεί ένα container MySQL που ακούει στη θύρα 8306 και έχει δημιουργηθεί ένας root user με κωδικό τη λέξη password. Οπότε μπορεί κανείς να στήσει το σύστημα αν συνδεθεί στη βάση και τρέξει τα δύο scripts το `ddl.sql` και `dml.sql` .

[1] `https://github.com/kkgiorgos/NTUA-Database-ChefMaesters-Project.git`

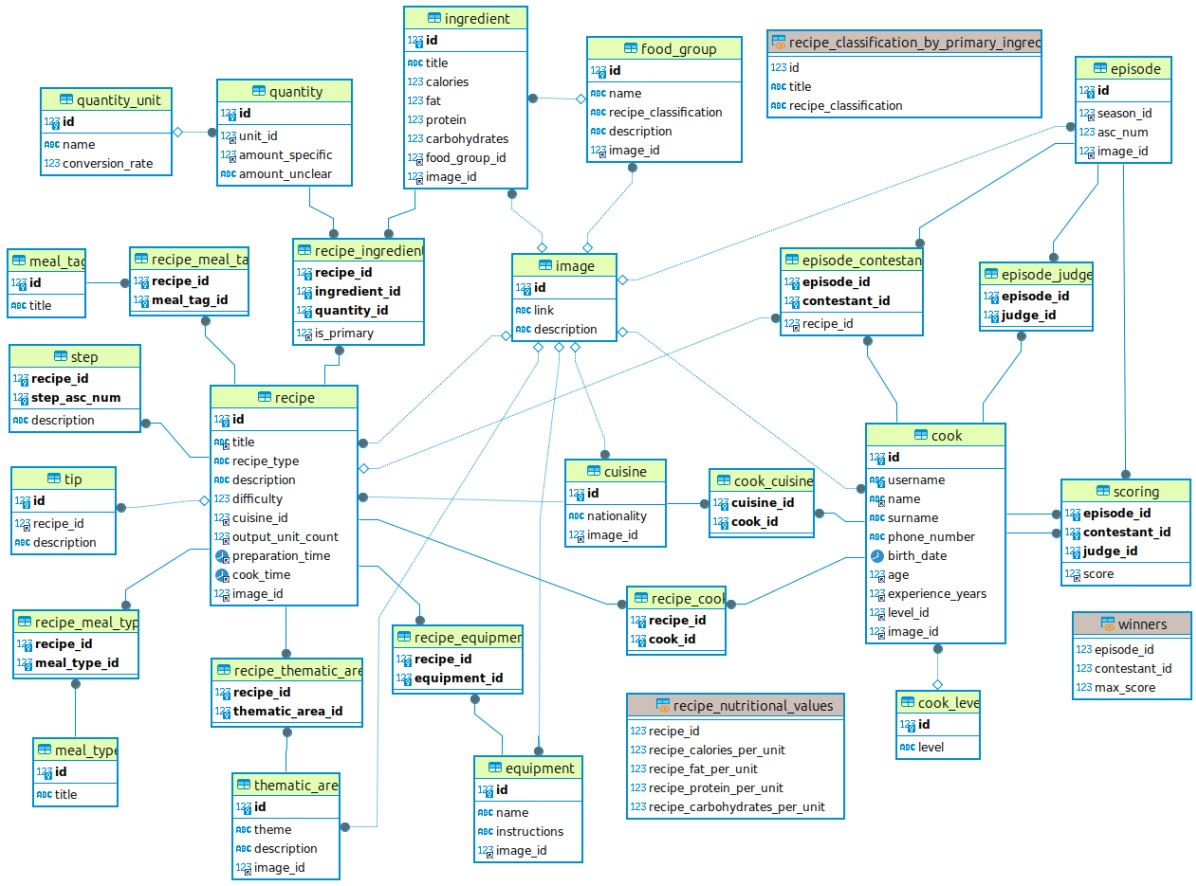
3 ER Model (Μοντέλο Οντοτήτων-Συσχετίσεων)

Σε αυτή την παράγραφο εξετάζεται η σχεδίαση του σχήματος μιας ΒΔ, εστιάζοντας στο E-R Model, μια αφηρημένη λογική αναπαράσταση, όπου ορίζονται οι οντότητες (entities) που μετέχουν στην προκειμένη βάση, καθώς κι οι συσχετίσεις μεταξύ των (relationships). Το μοντελο αυτό, περιλαμβάνει τις έννοιες (i) των συνόλων οντοτήτων, (ii) των συνόλων συσχετίσεων και (iii) των συνόλων ιδιοτήτων. Το E-R Model έχει μια σχετική, διαγραμματική αναπαράσταση: το διάγραμμα E-R, όπου αναπτύσσεται και απεικονίζεται λογικά η συνολική δομή της εκάστοτε υπό επεξεργασία βάσης, με απλό τρόπο, λόγος ο οποίος συνέβαλε στην ευρεία χρήση αυτών των διαγραμμάτων. Ένα τέτοιο διάγραμμα παρατίθεται ακριβώς από κάτω:



4 Σχεσιακό Διάγραμμα και ανάπτυξη της ΒΔ

Μια άλλη αφηρημένη λογική αναπαράσταση είναι το σχεσιακό μοντέλο. Η δόμηση των σχεσιακών ΒΔ αποσκοπεί στην παραγωγή ενός συνόλου σχεσιακών σχημάτων, τα οποία αδειοδοτούν την αποθήκευση χωρίς περιττή επαναληψιμότητα, αλλά και την εύκολη ανάκτηση πληροφοριών. Τα δύο μοντέλα E-R και σχεσιακό, ακολουθουν παρόμοιες αρχές, οπότε η μετατροπή από το ένα στο άλλο είναι σχετικά απλή. Για κάθε σύνολο οντοτήτων και για κάθε σύνολο συσχετίσεων στη σχεδίαση μιας ΒΔ υπάρχει ένα μοναδικό σχεσιακό σχήμα, όπου δίνεται το όνομα του αντίστοιχου συνόλου οντοτήτων ή συνόλου συσχετίσεων. Το σχεσιακό μοντέλο χρησιμοποιεί ένα σύνολο από πίνακες για τα δεδομένα και τις συσχετίσεις μεταξύ αυτών των δεδομένων, με κάθε πίνακα να φέρει μοναδικό όνομα. Γενικά, μία γραμμή ενός πίνακα αντιπροσωπεύει μια συσχέτιση μεταξύ ενός συνόλου τιμών. Ο όρος σχέση χρησιμοποιείται αναφερόμενος σε έναν πίνακα, ενώ ο όρος tuple χρησιμοποιείται ως αναφορά σε μια γραμμή και ο όρος ιδιότητα σε μία στήλη του πίνακα, αντίστοιχα. Στο σχεσιακό διάγραμμα κάθε σχέση εμφανίζεται σε ένα πλαίσιο με το όνομα της σχέση στην κορυφή, εδώ εμφανίζονται με πράσινο χρώμα, και με τις ιδιότητες να ακολουθούν σε μορφή στήλης από κάτω.



Οι οντότητες του συστήματος μπορούν να διαχωριστούν σε τρεις κατηγορίες. Αυτές που έχουν σχέση με τις συνταγές, τα υλικά και τα επεισόδια.

Αρκετές οντότητες χρειάζονται φωτογραφίες. Για τον σκοπό αυτό δημιουργήθηκε ξεχωριστός πίνακας για τις φωτογραφίες όλου του συστήματος.

```
CREATE TABLE image (
    id int,
    link text,
    description text,
    primary key (id)
);
```

4.1 Συνταγές

Κάθε συνταγή ανήκει σε μία εθνική κουζίνα. Για τον στόχο αυτό δημιουργήθηκε σχέση εθνικών κουζινών (`cuisine`)

```
CREATE TABLE cuisine (
    id int,
    nationality varchar(100) not null,
    image_id int,
    primary key (id),
    foreign key (image_id) references image(id)
);
```

Ακολουθούν οι συνταγές (`recipe`) που περιλαμβάνουν όλα τα attributes που απαιτούνται για να χαρακτηριστε.

```
CREATE TABLE recipe (
    id int,
    title varchar(100) not null,
    recipe_type enum('Cooking', 'Baking'),
    description text,
    difficulty int,
    cuisine_id int not null,
    output_unit_count int,
    preparation_time time,
    cook_time time,
    image_id int,
    primary key (id),
    constraint difficulty_chk check (difficulty between 1 and 5),
    foreign key (cuisine_id) references cuisine(id),
    foreign key (image_id) references image(id)
);
```

Εκτός αυτών όμως, υπάρχουν αρκετές σχέσεις many-to-many ή many-to-one που δεν μπορούν να υλοποιηθούν απευθείας πάνω στη σχέση της συνταγής.

Η μορφή του γεύματος (`meal_type`), δηλαδή αν είναι Πρωινό, Μεσημεριανό, κλπ, απαιτεί τέτοια σχέση, καθώς μπορεί μια συνταγή να ανήκει σε πολλές μορφές γεύματος.

```
CREATE TABLE meal_type(
    id int,
    title varchar(100) not null,
    primary key (id)
);

CREATE TABLE recipe_meal_type(
    recipe_id int,
    meal_type_id int,
    primary key (recipe_id, meal_type_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (meal_type_id) references meal_type(id)
);
```

Αντίστοιχα οι πιθανώς απεριόριστες ετικέτες (`meal_tag`) για μορφές/τύπους γεύματος (π.χ Brunch) θέλουν αντίστοιχη μεταχείριση.

```
CREATE TABLE meal_tag(
    id int,
    title varchar(100) not null,
    primary key (id)
);

CREATE TABLE recipe_meal_tag(
    recipe_id int,
    meal_tag_id int,
    primary key (recipe_id, meal_tag_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (meal_tag_id) references meal_tag(id)
);
```

Ο διαγωνισμός θέλει να μπορεί να ομαδοποιεί τις συνταγές σε θεματικές ενότητες με δικά του κριτήρια (`thematic_area`) οι οποίες ουσιαστικά ακολουθούν την ίδια λογική με τα tags.

```
CREATE TABLE thematic_area(
    id int,
    theme varchar(100) not null,
    description text,
    image_id int,
    primary key (id),
    foreign key (image_id) references image(id)
);

CREATE TABLE recipe_thematic_area(
    recipe_id int,
    thematic_area_id int,
    primary key (recipe_id, thematic_area_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (thematic_area_id) references thematic_area(id)
);
```

Επιπλέον, χρειαζόμαστε το πολύ τρεις χρηστικές συμβουλές (`tip`) ανα συνταγή (tips) οι οποίες θεωρούμε δεν είναι γενικές αλλά εξαρτώνται από την συνταγή (δηλαδή είναι σχέση many-to-one). Ο περιορισμός το πολύ 3 tips ανα συνταγή υλοποιείται με trigger.

```
CREATE TABLE tip(
    id int,
    recipe_id int not null,
    description text not null,
    primary key (id),
    foreign key (recipe_id) references recipe(id)
);

CREATE TRIGGER ins_recipe_tip BEFORE INSERT ON tip FOR EACH ROW
BEGIN
    IF ( (SELECT count(*)
        FROM tip t
        WHERE t.recipe_id = new.recipe_id) = 3) THEN

        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Maximum Tips Allowed is 3!!!';
    END IF;
END
```

Θεωρούμε ότι οι συνταγές χρειάζονται εξοπλισμό (`equipment`), όμως προφανώς ο εξοπλισμός είναι κάτι γενικό το οποίο μοιράζονται όλες οι συνταγές, με κάθε εξάρτημα να μπορεί να χρησιμοποιηθεί πολλές φορές και κάθε συνταγή να χρειάζεται συνήθως παραπάνω από ένα εξαρτήματα, επομένως συνδέεται με τις συνταγές μέσω σχέσης many-to-many.

```
CREATE TABLE equipment(
    id int,
    name varchar(100) not null,
    instructions text,
    image_id int,
    primary key (id),
    foreign key (image_id) references image(id)
);

CREATE TABLE recipe_equipment(
    recipe_id int,
    equipment_id int,
    primary key (recipe_id, equipment_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (equipment_id) references equipment(id)
);
```

Κάθε συνταγή αποτελείται απο βήματα (`steps`) τα οποία χαρακτηρίζονται από τον αύξοντα αριθμό τους εντός της συνταγής.

```
CREATE TABLE step(
    recipe_id int,
    step_asc_num int,
    description text not null,
    primary key (recipe_id, step_asc_num),
    foreign key (recipe_id) references recipe(id)
);
```

4.2 Υλικά

Κάθε συνταγή αποτελείται από υλικά (`ingredients`). Τα υλικά, όπως και ο εξοπλισμός είναι κάτι γενικό που το μοιράζονται όλες οι συνταγές. Επιπλέον, κάθε υλικό ανήκει σε μία (και μόνο μία) ομάδα τροφίμων (`food_group`).

```
CREATE TABLE food_group(
    id int,
    name varchar(100) not null,
    recipe_classification varchar(100),
    description text,
    image_id int,
    primary key (id),
    foreign key (image_id) references image(id)
);
```

Θεωρούμε ότι η διατροφική αξία κάθε υλικού είναι αποθηκευμένη για ποσότητα 100g.

```
CREATE TABLE ingredient(
    id int,
    title varchar(100) not null,
    calories int,
    fat int,
    protein int,
    carbohydrates int,
    food_group_id int not null,
    image_id int,
    primary key (id),
    foreign key (food_group_id) references food_group(id),
    foreign key (image_id) references image(id)
);
```

Προφανώς κάθε συνταγή έχει πολλά υλικά, καθένα από τα οποία εμφανίζεται σε μία ορισμένη ποσότητα (`quantity`). Οι ποσότητες μπορεί να είναι είτε σαφώς ορισμένες (ένας αριθμός) με μία μονάδα μέτρησης (`quantity_unit`) ή ασαφώς ορισμένες (ένα περιγραφικό κείμενο ορισμού της ποσότητας).

```
CREATE TABLE quantity_unit(
    id int,
    name varchar(100) not null,
    conversion_rate decimal(11, 9) not null,
    primary key (id)
);

CREATE TABLE quantity(
    id int,
    unit_id int,
    amount_specific decimal(6,2),
    amount_unclear varchar(100),
    primary key (id),
    foreign key (unit_id) references quantity_unit(id)
);
```

Προφανώς, μια συνταγή δεν μπορεί να έχει ταυτόχρονα το ίδιο υλικό και σαφώς και ασαφώς ορισμένο. Για τον λόγο αυτό υλοποιήθηκε trigger που κάνει αυτόν τον έλεγχο. Για τις σαφώς ορισμένες ποσότητες θέλουμε να μπορούμε να κάνουμε τη μετατροπή σε αναλογία των 100g που αντιστοιχεί στις πληροφορίες που έχουμε για τα υλικά.

```
CREATE TRIGGER ins_quantity BEFORE INSERT ON quantity FOR EACH ROW
BEGIN
        IF (new.amount_unclear IS NOT NULL
            AND (new.amount_specific IS NOT NULL
                OR new.unit_id IS NOT NULL)) THEN

                SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = 'Amount is unclear, specific should
                         ↪  not be defined!!!';
        ELSEIF (new.amount_specific IS NOT NULL
            AND new.unit_id IS NULL) THEN

                SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = 'Specific amount requires unit
                         ↪  type!!!';
        ELSEIF (new.unit_id IS NOT NULL
            AND new.amount_specific IS NULL) THEN

                SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = 'Specific amount unit defined but no
                         ↪  amount given!!!';
        ELSEIF (new.amount_unclear IS NULL
            AND new.amount_specific IS NULL
            AND new.unit_id IS NULL) THEN

                SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = 'No quantity defined';
        END IF;
END
```

Τέλος, για κάθε συνταγή υπάρχει ακριβώς ένα υλικό που ορίζεται ως βασικό, η ομάδα τροφίμων του οποίου καθορίζει τον χαρακτηρισμό της συνταγής. Ο περιορισμός αυτός υλοποιείται πάλι με χρήση trigger.

```
CREATE TABLE recipe_ingredients (
    recipe_id int,
    ingredient_id int,
    quantity_id int,
    is_primary bool not null,
    primary key (recipe_id, ingredient_id, quantity_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (ingredient_id) references ingredient(id),
    foreign key (quantity_id) references quantity(id)
);

CREATE TRIGGER ins_recipe_ingredient BEFORE INSERT ON recipe_ingredients FOR
 ↪  EACH ROW
BEGIN
    IF (new.is_primary = TRUE AND
        (SELECT count(*)
            FROM recipe_ingredients ri
            WHERE ri.recipe_id = new.recipe_id
                AND ri.is_primary = TRUE) = 1) THEN

            SIGNAL SQLSTATE '45000'
                    SET MESSAGE_TEXT = 'Only one primary ingredient allowed per
                     ↪  recipe!!!';
    END IF;
END
```

Επομένως μπορούμε να φτιάξουμε ένα view για τα διατροφικά χαρακτηριστικά των συνταγών και

ένα για τον χαρακτηρισμό των συνταγών με βάση το κύριο συστατικό.

```
CREATE VIEW recipe_nutritional_values AS
SELECT
    r.id as recipe_id,
    sum(q.amount_specific * qu.conversion_rate * i.calories /
     ↪  r.output_unit_count) recipe_calories_per_unit,
    sum(q.amount_specific * qu.conversion_rate * i.fat  / r.output_unit_count)
     ↪  recipe_fat_per_unit,
    sum(q.amount_specific * qu.conversion_rate * i.protein /
     ↪  r.output_unit_count) recipe_protein_per_unit,
    sum(q.amount_specific * qu.conversion_rate * i.carbohydrates /
     ↪  r.output_unit_count) recipe_carbohydrates_per_unit
FROM recipe r
INNER JOIN recipe_ingredients ri ON r.id = ri.recipe_id
INNER JOIN ingredient i ON ri.ingredient_id = i.id
INNER JOIN quantity q ON ri.quantity_id = q.id
INNER JOIN quantity_unit qu ON qu.id = q.unit_id
GROUP BY r.id;

CREATE VIEW recipe_classification_by_primary_ingredient AS
SELECT r.id, r.title, fg.recipe_classification
FROM recipe r
INNER JOIN recipe_ingredients ri ON r.id = ri.recipe_id
INNER JOIN ingredient i ON ri.ingredient_id = i.id
INNER JOIN food_group fg ON i.food_group_id = fg.id
WHERE ri.is_primary = TRUE;
```

4.3 Διαγωνισμός

Ο διαγωνισμός, εκτός από συνταγές, χρειάζεται και μάγειρες. Αρχικά, οι μάγειρες θεωρείται έχουν έναν (μόνο) χαρακτηρισμό επαγγελματικής κατάρτισης (`cook_level`). Οι πιθανοί χαρακτηρισμοί αποθηκεύονται σε ξεχωριστό πίνακα (ακολουθούμε την ίδια λογική με τις ομάδες τροφίμων).

```
CREATE TABLE cook_level (
    id int,
    level varchar(30) not null,
    primary key (id)
);
```

Οι μάγειρες εκτός από απλές εγγραφές στο σύστημα είναι και χρήστες αυτού. Για τον σκοπό αυτό εκτός από τα υπόλοιπα στοιχεία που τους χαρακτηρίζουν έχουν και ένα πεδίο username το οποίο αντιστοιχεί στο username του αντίστοιχου χρήστη της βάσης. Αυτό το πεδίο θα αξιοποιηθεί αργότερα σε ειδικά views στα οποία έχουν πρόσβαση οι μάγειρες ως χρήστες. Εκτός των βασικών στοιχείων, οι μάγειρες έχουν εξειδίκευση σε πολλές εθνικές κουζίνες. Ως χρήστες τους δίνεται η δυνατότητα προσθήκης νέων συνταγών. Το σύστημα διατηρεί σε ξεχωριστό πίνακα ποιες συνταγές έχει ένας μάγειρας. Αυτές είναι τόσο οι νέες συνταγές που προστέθηκαν από αυτόν όσο και αυτές που του ανατέθηκαν κατά τη διάρκεια των επεισοδίων. Προφανώς με κατάλληλα views ο μάγειρας ως χρήστης βλέπει και επεξεργάζεται πληροφορίες μόνο για τις συνταγές που έχει.

```
CREATE TABLE cook (
    id int,
    username varchar(32) UNIQUE NOT NULL,
    name varchar(100),
    surname varchar(100),
    phone_number varchar(20),
    birth_date date,
    age int,
    experience_years int,
    level_id int not null,
    image_id int,
    primary key (id),
    foreign key (level_id) references cook_level(id),
    foreign key (image_id) references image(id)
);

CREATE TABLE recipe_cook (
    recipe_id int,
    cook_id int,
    primary key (recipe_id, cook_id),
    foreign key (recipe_id) references recipe(id),
    foreign key (cook_id) references cook(id)
);

CREATE TABLE cook_cuisine (
    cuisine_id int,
    cook_id int,
    primary key (cuisine_id, cook_id),
    foreign key (cuisine_id) references cuisine(id),
    foreign key (cook_id) references cook(id)
);
```

Ο διαγωνισμός οργανώνεται σε έτη (σεζόν). Κάθε σεζόν αποτελείται από 10 διαφορετικά επεισόδια (`episode`), όμως το σύστημα επιτρέπει οσαδήποτε πολλά να δημιουργηθούν από τον χρήστη, καθώς μπορεί να θελήσει να δημιουργηθεί ένα special επεισόδιο πέραν των 10. Κάθε επεισόδιο του διαγωνισμού έχει ένα γενικό αναγνωριστικό για χρήση με άλλες σχέσεις.

```
CREATE TABLE episode (
    id int auto_increment,
    season_id int not null,
    asc_num int not null,
    image_id int,
    primary key (id),
    foreign key (image_id) references image(id)
);
```

Σε κάθε επεισόδιο επιλέγουμε 10 διαφορετικές εθνικές κουζίνες. Με βάση αυτές επιλέγουμε 10 μάγειρες ως διαγωνιζόμενους (`episode_contestants`) (το σύστημα απαγορεύει περισσότερους με χρήση trigger) με βάση την εξειδίκευση τους στις επιλεγμένες κουζίνες και αναθέτουμε στον καθένα από μία συνταγή της αντίστοιχης εθνικής κουζίνας. Το trigger απαγορεύει επίσης να έχουμε σε πάνω από 3 συνεχόμενα επεισόδια τις ίδιες εθνικές κουζίνες καθώς επίσης και τους ίδιους διαγωνιζόμενους.

```
CREATE TABLE episode_contestants (
    episode_id int,
    contestant_id int,
    recipe_id int,
    primary key (episode_id, contestant_id),
    foreign key (episode_id) references episode(id),
    foreign key (contestant_id) references cook(id),
    foreign key (recipe_id) references recipe(id)
);
```

Εκτός από διαγωνιζόμενους σε κάθε επεισόδιο χρειάζεται να επιλέξουμε τρεις μάγειρες κριτές (`episode_judges`). Υπάρχουν trigger για να βεβαιωθούν ότι ένας κριτής δεν είναι στο ίδιο επεισόδιο διαγωνιζόμενος και το αντίστροφο. Επιπλέον στο trigger εξασφαλίζουμε ότι ένας κριτής δεν εμφανίζεται πάνω από 3 φορές συνεχόμενα και ότι δεν προσπαθούμε να εισάγουμε κριτή που έχει ήδη εισαχθεί.

```
CREATE TABLE episode_judges (
    episode_id int,
    judge_id int,
    primary key (episode_id, judge_id),
    foreign key (episode_id) references episode(id),
    foreign key (judge_id) references cook(id)
);
```

Ακολουθούν τα triggers για τους διαγωνιζόμενους και τους κριτές.

```
CREATE TRIGGER ins_episode_contestant BEFORE INSERT ON episode_contestants FOR
 ↪  EACH ROW
BEGIN
    IF ((SELECT count(*) FROM episode_contestants ec WHERE ec.episode_id =
     ↪  new.episode_id) = 10) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Maximum Contestants in Episode Allowed is
             ↪  10!!!';
    END IF;

    DROP TEMPORARY TABLE IF EXISTS last_three_episodes;
    CREATE TEMPORARY TABLE last_three_episodes
        SELECT e.id
        FROM episode e
        ORDER BY e.id DESC
        LIMIT 3;

    IF ( (SELECT count(*)
        FROM episode_contestants ec
        INNER JOIN  last_three_episodes le ON le.id = ec.episode_id
        WHERE ec.contestant_id = new.contestant_id) > 2 ) THEN

        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Same contestant cannot appear 3 times in a
             ↪  row!!!';
    END IF;

    IF ( SELECT count(*)
        FROM episode_contestants ec
        INNER JOIN last_three_episodes le ON le.id = ec.episode_id
        INNER JOIN recipe r ON r.id = recipe_id
        WHERE r.cuisine_id IN
            (SELECT r.cuisine_id FROM recipe r WHERE r.id = new.recipe_id)
        > 2 ) THEN

        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Same cuisine cannot appear 3 times in a row!!!';
    END IF;

    IF ( (SELECT count(*)
        FROM episode_judges ej
        WHERE ej.judge_id = new.contestant_id
        AND ej.episode_id = new.episode_id) > 0 ) THEN

    SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Contestant cannot appear as judge in the same
             ↪  episode!!!';
    END IF;

    DROP TEMPORARY TABLE IF EXISTS last_three_episodes;
END
```

```
CREATE TRIGGER ins_episode_judge BEFORE INSERT ON episode_judges FOR EACH ROW
BEGIN
    IF ((SELECT count(*) FROM episode_judges ej WHERE ej.episode_id =
     ↪  new.episode_id) = 3) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Maximum Judges in Episode Allowed is 3!!!';
    END IF;

    DROP TEMPORARY TABLE IF EXISTS last_three_episodes;
    CREATE TEMPORARY TABLE last_three_episodes
        SELECT e.id
        FROM episode e
        ORDER BY e.id DESC
        LIMIT 3;

    IF ( (SELECT count(*)
        FROM episode_judges ej
        INNER JOIN  last_three_episodes le ON le.id = ej.episode_id
        WHERE ej.judge_id = new.judge_id) > 2 ) THEN

        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Same judge cannot appear 3 times in a row!!!';
    END IF;

    IF ( (SELECT count(*)
        FROM episode_contestants ec
        WHERE ec.contestant_id = new.judge_id
        AND ec.episode_id = new.episode_id) > 0 ) THEN

    SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Judge cannot appear as contestant in the same
             ↪  episode!!!';
    END IF;

    DROP TEMPORARY TABLE IF EXISTS last_three_episodes;
END
```

Τέλος, για κάθε συνδυασμό επεισοδίου, διαγωνιζόμενου και κριτή πρέπει να αποθηκευτεί και η βαθμολογία (`scoring`). Προφανώς, οι προηγούμενοι πίνακες έχουν την ίδια πληροφορία με αυτόν, αλλά δεν μας πειράζει καθώς θεωρούμε ότι η διαδικασία κλήρωσης των επεισοδίων προηγείται της βαθμολόγησης και θέλουμε ξεχωριστά την κατάσταση των επεισοδίων.

```
CREATE TABLE scoring (
    episode_id int,
    contestant_id int,
    judge_id int,
    score int not null,
    primary key (episode_id, contestant_id, judge_id),
    foreign key (episode_id) references episode(id),
    foreign key (contestant_id) references cook(id),
    foreign key (judge_id) references cook(id),
    constraint score_chk check (score between 1 and 5)
);
```

Για την υλοποίηση των κληρώσεων των διαγωνισμών δημιουργήθηκαν 3 procedures. Το πρώτο δημιουργεί ένα τυχαίο έγκυρο επεισόδιο. Το δεύτερο δημιουργεί όσα επεισόδια θέλουμε και τα βάζει να συνεχίζουν μετά από το τελευταίο που έχει αποθηκευτεί στο σύστημα. Και το τελευταίο γεμίζει με τυχαίες βαθμολογίες όσα επεισόδια δεν έχουν βαθμολογηθεί. Η υλοποίηση φαίνεται στο αρχείο `ddl.sql` , και παραλείπεται καθώς δεν προσφέρει κάτι. Παραθέτουμε απλά τα αποτυπώματα τους.

```
PROCEDURE generate_episode(IN episode_season_id int, IN episode_asc_num int)

PROCEDURE generate_episodes (IN episode_count int)

PROCEDURE fill_episode_scoring ()
```

Τέλος φτιάχνουμε `VIEW` για τους νικητές του κάθε επεισοδίου. Δυστυχώς, δεν ήταν εύκολο να διαχειριστούμε τις ισοπαλίες οπότε επιλέγουμε όλους τους διαγωνιζόμενους που ήρθαν σε ισοπαλία στο συγκεκριμένο επεισόδιο. Θα μπορούσε να υλοποιηθεί εύκολα η λογική της κατάταξης σε περίπτωση ισοπαλίας στο application layer.

```
CREATE VIEW winners AS
WITH scores AS (
    SELECT s.episode_id AS ep_id, s.contestant_id AS contestant_id, SUM(s.score)
     ↪  AS tot_score
    FROM scoring s
    GROUP BY s.episode_id, s.contestant_id
), max_scores AS (
    SELECT ss.ep_id AS episode_id, MAX(ss.tot_score) AS max_score
    FROM scores ss
    GROUP BY ss.ep_id
) SELECT ms.episode_id, s.contestant_id, ms.max_score
FROM max_scores ms
INNER JOIN scores s ON ms.episode_id = s.ep_id
    AND ms.max_score = s.tot_score;
```

4.4 Διαχείριση χρηστών

Σε αυτό το σημείο μπορούμε να εξηγήσουμε τον τρόπο που υλοποιήθηκαν οι χρήστες μάγειρες. Προφανώς ο διαχειριστής είναι ένας χρήστης με όλα τα δικαιώματα πάνω στη συγκεκριμένη βάση.

Δημιουργήθηκε ρόλος για τους χρήστες στον οποίο δόθηκαν τα απαραίτητα permissions και όταν εισάγονται οι νέοι μάγειρες τότε δημιουργούνται οι αντίστοιχοι χρήστες της βάσης και τους ανατίθενται τα κατάλληλα δικαιώματα.

Τα περισσότερα `SELECT, UPDATE` επιτράπηκαν πάνω σε `VIEWS` που περιορίζουν τις συνταγές μόνο σε αυτές που ανήκουν στον συγκεκριμένο μάγειρα. Τα `INSERT` επιτρέπονται στους αρχικούς πίνακες. Από τα επεισόδια δίνεται η δυνατότητα εύρεσης πληροφοριών μόνο για τα επεισόδια στα οποία συμμετείχε ο συγκεκριμένος μάγειρας ως διαγωνιζόμενος. Θα μπορούσαν να δημιουργηθούν και αντίστοιχα `VIEW` για τους κριτές ώστε να μπορούν να επεξεργάζονται βαθμολογίες αλλά αυτό δεν υλοποιήθηκε. Σε όλα τα στοιχεία που δεν αναφέρονται σε συγκεκριμένη συνταγή αλλά είναι γενικά ο χρήστης έχει δικαιώματα `SELECT` , εκτός από τα υλικά και τον εξοπλισμό στα οποία έχει δικαιώματα και για `INSERT` καθώς μπορεί να θέλει να προσθέσει πράγματα για μια συνταγή του που δεν υπήρχαν πριν. Ακολουθούν αναλυτικά τα δικαιώματα πάνω σε πίνακες και όψεις που δόθηκαν στους χρήστες. Η αναλυτική δημιουργία των `VIEW` παραλείπεται. Παραθέτουμε μόνο το `cook_user_info_view` που επιστρέφει τις πληροφορίες για τον συγκεκριμένο χρήστη, με βάση αυτό όλα τα άλλα `VIEW` περιορίζουν τα αποτελέσματα τους. Επίσης, εδώ υλοποιούμε και το trigger κατά την προσθήκη συνταγής ώστε να ανατίθεται και στον μάγειρα που τη δημιούργησε.

```
CREATE VIEW cook_user_info_vw AS
SELECT c.*
FROM cook c
WHERE c.username = SUBSTRING_INDEX(USER(), '@', 1);

CREATE TRIGGER ins_recipe AFTER INSERT ON recipe FOR EACH ROW
BEGIN
    INSERT INTO recipe_cook (recipe_id, cook_id)
        SELECT new.id, cuiv.id
        FROM cook_user_info_vw cuiv
        LIMIT 1;
END
```

```
CREATE ROLE IF NOT EXISTS chefmaesters_admin_user, chefmaesters_cook_user;
GRANT ALL PRIVILEGES ON chefmaesters.* TO chefmaesters_admin_user;

GRANT SELECT ON quantity_unit TO chefmaesters_cook_user;
GRANT SELECT ON food_group TO chefmaesters_cook_user;
GRANT SELECT ON meal_tag TO chefmaesters_cook_user;
GRANT SELECT ON meal_type TO chefmaesters_cook_user;
GRANT SELECT, INSERT ON equipment TO chefmaesters_cook_user;
GRANT SELECT, INSERT ON ingredient TO chefmaesters_cook_user;
GRANT SELECT ON cuisine TO chefmaesters_cook_user;
GRANT SELECT ON cook_level TO chefmaesters_cook_user;

GRANT SELECT, UPDATE ON cook_user_info_vw TO chefmaesters_cook_user;
GRANT SELECT ON cook_user_episode_view TO chefmaesters_cook_user;
GRANT SELECT ON cook_user_episode_contestants_view TO chefmaesters_cook_user;
GRANT SELECT ON cook_user_episode_judges_view TO chefmaesters_cook_user;
GRANT SELECT ON cook_user_scoring_view TO chefmaesters_cook_user;

GRANT SELECT, UPDATE ON cook_user_recipe_vw TO chefmaesters_cook_user;
GRANT INSERT ON recipe TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_tip_vw TO chefmaesters_cook_user;
GRANT INSERT ON tip TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_step_vw TO chefmaesters_cook_user;
GRANT INSERT ON step TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_recipe_meal_tag_vw TO chefmaesters_cook_user;
GRANT INSERT ON recipe_meal_tag TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_recipe_ingredients_vw TO
 ↪  chefmaesters_cook_user;
GRANT INSERT ON recipe_ingredients TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_quantity_vw TO chefmaesters_cook_user;
GRANT INSERT ON quantity TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_recipe_equipment_vw TO chefmaesters_cook_user;
GRANT INSERT ON equipment TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_recipe_thematic_area_vw TO
 ↪  chefmaesters_cook_user;
GRANT INSERT ON recipe_thematic_area TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_recipe_meal_type_vw TO chefmaesters_cook_user;
GRANT INSERT ON recipe_meal_type TO chefmaesters_cook_user;
GRANT SELECT, UPDATE ON cook_user_cook_cuisine_vw TO chefmaesters_cook_user;
GRANT INSERT ON cook_cuisine TO chefmaesters_cook_user;
```

4.5 Ευρετήρια (Indices)

Με τη δημιουργία των παραπάνω πινάκων το σύστημα δημιουργεί αυτόματα ευρετήρια πάνω σε όλα τα primary keys και σε όλα τα foreign keys, εκτός από αυτά που είναι το πρώτο μέρος ενός συνδυασμού που οδηγεί στο primary key. Επιπλέον, δημιουργεί αυτόματα ευρετήριο σε πεδία που είναι candidate keys, δηλαδή σε attributes που έχουν περιορισμούς `UNIQUE NOT NULL` (π.χ. username).

Για να επιταχύνουμε τα range queries δημιουργούμε ευρετήρια σε αριθμητικά πεδία. Για να επιταχύνουμε ταξινομήσεις δημιουργούμε ευρετήρια σε ονόματα ή τίτλους. Τέλος δημιουργούμε ευρετήρια στο πεδίο που χαρακτηρίζει το βασικό υλικό μιας συνταγής, για να κάνουμε γρήγορα `JOIN` στα βασικά υλικά και στο πεδίο *season id* του επεισοδίου για να γίνονται γρήγορα τα `GROUP BY` .

```
CREATE INDEX recipe_title_index ON recipe(title) USING BTREE;
CREATE INDEX prep_time_index ON recipe(preparation_time) USING BTREE;
CREATE INDEX cook_time_index ON recipe(cook_time) USING BTREE;
CREATE INDEX unit_count_index ON recipe(output_unit_count) USING BTREE;

CREATE INDEX recipe_id_index ON recipe_ingredients(recipe_id) USING BTREE;
CREATE INDEX is_primary_index ON recipe_ingredients(is_primary) USING BTREE;

CREATE INDEX quantity_amount_index ON quantity(amount_specific) USING BTREE;

CREATE INDEX cook_names_index ON cook(name, surname) USING BTREE;
CREATE INDEX cook_age_index ON cook(age) USING BTREE;
CREATE INDEX cook_exp_years_index ON cook(experience_years) USING BTREE;

CREATE INDEX season_index ON episode(season_id) USING BTREE;
CREATE INDEX score_index ON scoring(score) USING BTREE;
```

5 Queries

Τα ζητούμενα queries που υλοποιήθηκαν είναι τα ακόλουθα. Ο χρήστης μπορεί να τα τρέξει και του επιστρέφονται αποτελέσματα. (Προσοχή εδώ δεν περιορίζουμε τον χρήστη σε έναν απλό μάγειρα, αλλά μπορεί να δει όλη τη βάση).

Ερώτηση 5.1 *Μέσος Όρος Αξιολογήσεων (σκορ) ανά μάγειρα και Εθνική κουζίνα.*

Σε αυτό το Query παράγεται ο μέσος όρος αξιολογήσεων κάθε μάγειρα κι ο μέσος όρος κάθε εθνικής κουζίνας ξεχωριστά, εξού και τα δύο ξεχωριστά block κώδικα. Με την εντολή `AVG()` υπλογίζονται οι επί μέρους μέσοι όροι. Το `EXPLAIN SELECT` φέρνει μόνο τα ζητούμενα δεδομένα, ενώ παράλληλα βοηθάει στον συνδυασμό δεδομένων από διαφορετικές τιμές. Οι εντολές `GROUP BY` και `ORDER BY` ταξινομούν τις προκύπτουσες απατνήσεις βάσει πλήρους ονόματος μάγειρα ή εθνικότητας.

```
EXPLAIN SELECT
    c.id AS cook_id,
    CONCAT(c.name, ' ', c.surname) AS cook_name,
    AVG(s.score) AS average_rating_cook
FROM scoring AS s
JOIN episode_contestants AS ec ON s.episode_id = ec.episode_id AND
 ↪  s.contestant_id = ec.contestant_id
JOIN cook AS c ON ec.contestant_id = c.id
GROUP BY c.id
ORDER BY c.name, c.surname;

SELECT
    cu.id AS cuisine_id,
    cu.nationality AS national_cuisine,
    AVG(s.score) AS average_rating_cuisine
FROM scoring AS s
JOIN episode_contestants AS ec ON s.episode_id = ec.episode_id AND
 ↪  s.contestant_id = ec.contestant_id
JOIN cook AS c ON ec.contestant_id=c.id
JOIN cook_cuisine AS cc ON c.id=cc.cook_id
JOIN cuisine AS cu ON cc.cuisine_id = cu.id
GROUP BY cu.id
ORDER BY cu.nationality;
```

Ερώτηση 5.2 *Για δεδομένη Εθνική κουζίνα και έτος, ποιοι μάγειρες ανήκουν σε αυτήν και ποιοι μάγειρες συμμετείχαν σε επεισόδια;*

Εδώ, ζητείται να υλοποιηθεί ένα query το οποίο παράγει ποιοι μάγειρες ειδικεύονται σε μια συγκεκριμένη Εθνική Κουζίνα.

```
-- Find the cooks that know the specific national cuisine
SELECT
    c.id AS cook_id,
    c.name AS cook_name,
    c.surname AS cook_surname
FROM cook AS c
JOIN cook_cuisine AS cc ON c.id = cc.cook_id
WHERE cc.cuisine_id = :cuisine_id;
```

Σε μία πρώτη ανάγνωση, θέλουμε να βρούμε το ποιοι μάγειρες συμμετείχαν σε επεισόδια ενός συγκεκριμένου έτους.

```sql
-- Find the cooks that participated in a given season's episodes
SELECT
    c.id AS cook_id,
    c.name AS cook_name,
    c.surname AS cook_surname
FROM cook AS c
JOIN episode_contestants AS ec ON c.id = ec.contestant_id
JOIN episode AS e ON ec.episode_id = e.id
WHERE e.season_id = :season_id;
```

Σε μια δεύτερη ανάγνωση, μπορεί να θεωρηθεί ότι ζητάται να επιστραφεί ποιοι μάγειρες συμμετείχαν στα επεισόδια ενός συγκεκριμένου κύκλου, εκπροσωπώντας μια εθνική κουζίνα. Οπότε, στο τρίτο `SELECT` βρίσκονται ποιοι παίκτες που εκπροσωπούν κάθε κουζίνα συμμετείχαν σε επεισόδια της σεζόν.

```sql
-- Find the cooks that participated in a given season's episodes under a
 ↪  specific cuisine
SELECT
    c.id AS cook_id,
    c.name AS cook_name,
    c.surname AS cook_surname
FROM cook AS c
JOIN cook_cuisine AS cc ON c.id = cc.cook_id
JOIN episode_contestants AS ec ON c.id = ec.contestant_id
JOIN episode AS e ON ec.episode_id = e.id
WHERE cc.cuisine_id= :cuisine_id AND e.season_id= :season_id;
```

Ερώτηση 5.3 *Βρείτε τους νέους μάγειρες (ηλικία < 30 ετών) που έχουν τις περισσότερες συνταγές.*

Στο παρόν ερώτημα ζητάται να βρεθούν οι νεότεροι μάγειρες, οι οποίοι δεν έχουν υπερβεί τα 30 έτη κι έχουν τις περισσότερες συνταγές. Γι'αυτό το λόγο πρώτα βρίσκονται οι μάγειρες που ικανοποιούν αυτό το ηλικιακό όριο, και μετράμε τιο πλήθος των συνταγών που αντιστοιχούν στον καθένα. Μετά ταξινομούνται σε φθίνουσα σειρά με βάση αυτό το πλήθος ώστε πρώτος να είναι ο μάγειρας με τον μεγαλύτερο αριθμό συνταγών και τελευταίος αυτός με τον μικρότερο.

```sql
SELECT
    c.id,
    c.name,
    c.surname,
    c.age,
    count(rc.recipe_id) as recipe_count
FROM cook c
INNER JOIN recipe_cook rc ON c.id = rc.cook_id
WHERE c.age < 30
GROUP BY c.id
ORDER BY recipe_count DESC;
```

Ερώτηση 5.4 *Βρείτε τους μάγειρες που δεν έχουν συμμετάσχει ποτέ ως κριτές σε κάποιο επεισόδιο.*

Για να βρεθούν οι μάγειρες, που δεν έχουν επιλεγεί ποτέ στο διαγωνισμό στη θέση του κριτή, ελέγχεται το πλήθος εμφανίσεων του καθενός ως κριτής.

```
SELECT c.id, c.name, c.surname
FROM cook c
LEFT JOIN episode_judges ej ON c.id = ej.judge_id
GROUP BY c.id
HAVING COUNT(ej.judge_id) = 0;
```

Ερώτηση 5.5 *Ποιοι κριτές έχουν συμμετάσχει στον ίδιο αριθμό επεισοδίων σε διάστημα ενός έτους με περισσότερες από 3 εμφανίσεις;*

Βάσει της θεωρίας του μαθήματος, το WITH στην SQL ορίζει μια προσωρινή σχέση, με φάσμα ισχύς μόνον μέσα στο sub-query στο οποίο υπάρχει το `WITH` . Εδώ, ορίζεται η προσωρινή σχέση `episode counts` , η οποία κρατάει του κριτές και μπορεί να χρησιμοποιηθεί σε οποιοδήποτε σημείο του κώδικα, που έχουν συμμετάσχει με αυτό το ρόλο σε επεισόδια πάνω από τρεις φορές. Μελετώνται όλα τα πιθανά ζεύγη από κριτές, διασφαλίζοντας ότι ποτέ δεν θα συγκριθεί κάποιος με τον εαυτό του. Για κάθε ένα από αυτά κρατάμε τους κριτές με το ίδιο episode count. Στη συγκεκριμένη υλοποίηση υποθέτουμε ότι αναζητούμε ζεύγη κριτών που είχαν τις ίδιες εμφανίσεις στην ίδια σεζόν.

```
-- This version assumes we are looking for judge pairs with the same amount of
 ↪  episode appearances within the same season
WITH episode_counts AS (
    SELECT ej.judge_id, e.season_id, COUNT(ej.episode_id) AS episode_count
    FROM episode_judges ej
    JOIN episode e ON ej.episode_id = e.id
    GROUP BY ej.judge_id, e.season_id
    HAVING COUNT(ej.episode_id) > 3
)
SELECT a.judge_id AS judge1_id, b.judge_id AS judge2_id, a.episode_count
FROM episode_counts a
JOIN episode_counts b ON a.season_id = b.season_id AND a.episode_count =
 ↪  b.episode_count AND a.judge_id < b.judge_id
ORDER BY a.episode_count DESC;
```

Μια εναλλακτική ερμηνεία, θέλει να κρατάμε όσα ζεύγη κριτών έχουν το ίδιο episode count, σε δύο οποιεσδήποτε σεζόν.

```
-- This version assumes we are looking for judge pairs with the same amount of
 ↪  episode appearances in any two seasons (not necessarily the same)
WITH episode_counts AS (
    SELECT ej.judge_id, e.season_id, COUNT(ej.episode_id) AS episode_count
    FROM episode_judges ej
    JOIN episode e ON ej.episode_id = e.id
    GROUP BY ej.judge_id, e.season_id
    HAVING COUNT(ej.episode_id) > 3
)
SELECT a.judge_id AS judge1_id, b.judge_id AS judge2_id, a.episode_count
FROM episode_counts a
JOIN episode_counts b ON a.episode_count = b.episode_count AND a.judge_id <
 ↪  b.judge_id
ORDER BY a.episode_count DESC;
```

Ερώτηση 5.6 *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 επεισόδια 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.*

Η βασική εκτέλεση του query προαπαιτεί να βρούμε σε έναν προσωρινό πίνακα τα tags κάθε συνταγής. Παίρνοντας το καρτεσιανό γινόμενο δύο τέτοιων πινάκων (κρατώντας λεξικογραφικά ζεύγη από tags) ανα συνταγή μπορούμε να κάνουμε `GROUP` ανά ζεύγος ετικετών και να μετρήσουμε τις εμφανίσεις κρατώντας τα 3 μεγαλύτερα αποτελέσματα.

```
WITH recipe_tags_joined AS
(
    SELECT r.id AS recipe_id, mt.id AS tag_id, mt.title AS tag_title
    FROM recipe_meal_tag rmt
    INNER JOIN recipe r ON rmt.recipe_id = r.id
    INNER JOIN meal_tag mt ON mt.id = rmt.meal_tag_id
)
SELECT rtj1.tag_title AS tag1, rtj2.tag_title AS tag2, COUNT(rtj1.tag_id) AS
 ↪  appearances
FROM recipe_tags_joined rtj1
INNER JOIN recipe_tags_joined rtj2 ON rtj1.recipe_id = rtj2.recipe_id AND
 ↪  rtj1.tag_id < rtj2.tag_id
GROUP BY rtj1.tag_id, rtj2.tag_id
ORDER BY appearances DESC
LIMIT 3;
```

Κάνοντας `EXPLAIN` βλέπουμε το ακόλουθο trace για το πώς επιλέγει η βάση να εκτελέσει το ερώτημα.

```
+----+-------------+-------+------------+--------+---------------------+
| id | select_type | table | partitions | type   | possible_keys       |
+----+-------------+-------+------------+--------+---------------------+
|  1 | SIMPLE      | r     | NULL       | index  | PRIMARY             |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY             |
|  1 | SIMPLE      | rmt   | NULL       | ref    | PRIMARY,meal_tag_id |
|  1 | SIMPLE      | mt    | NULL       | eq_ref | PRIMARY             |
|  1 | SIMPLE      | rmt   | NULL       | ref    | PRIMARY,meal_tag_id |
|  1 | SIMPLE      | mt    | NULL       | eq_ref | PRIMARY             |
+----+-------------+-------+------------+--------+---------------------+

+------------+---------+-----------------------------+------+
| key        | key_len | ref                         | rows |
+------------+---------+-----------------------------+------+
| cuisine_id | 4       | NULL                        |  990 |
| PRIMARY    | 4       | chefmaesters.r.id           |    1 |
| PRIMARY    | 4       | chefmaesters.r.id           |    4 |
| PRIMARY    | 4       | chefmaesters.rmt.meal_tag_id |    1 |
| PRIMARY    | 4       | chefmaesters.r.id           |    4 |
| PRIMARY    | 4       | chefmaesters.rmt.meal_tag_id |    1 |
+------------+---------+-----------------------------+------+

+----------+-----------------------------------------------+
| filtered | Extra                                         |
+----------+-----------------------------------------------+
|   100.00 | Using index; Using temporary; Using filesort |
|   100.00 | Using index                                   |
|   100.00 | Using index                                   |
|   100.00 | NULL                                          |
|   100.00 | Using index                                   |
|   100.00 | Using where                                   |
+----------+-----------------------------------------------+
```

Βλέπουμε ότι γενικά τρέχει αποδοτικά το συγκεκριμένο query καθώς υπάρχουν indices πάνω σε όλα τα operations που γίνονται. Θα μπορούσαμε να δοκιμάσουμε να αλλάξουμε τη σειρά των join εξαναγκάζοντας τη σειρά που γράφουμε με την εντολή `STRAIGHT_JOIN` για να δούμε πόσο πολύ περισσότερες γραμμές θα είχαμε. Δεν μπορούμε όμως να αλλάξουμε το query plan για να το κάνουμε καλύτερο στην συγκεκριμένη περίπτωση.

```
EXPLAIN WITH recipe_tags_joined AS
(
    SELECT r.id AS recipe_id, mt.id AS tag_id, mt.title AS tag_title
    FROM recipe_meal_tag rmt
    STRAIGHT_JOIN recipe r ON rmt.recipe_id = r.id
    STRAIGHT_JOIN meal_tag mt ON mt.id = rmt.meal_tag_id
)
SELECT rtj1.tag_title AS tag1, rtj2.tag_title AS tag2, COUNT(rtj1.tag_id) AS
 ↪  appearances
FROM recipe_tags_joined rtj1
INNER JOIN recipe_tags_joined rtj2 ON rtj1.recipe_id = rtj2.recipe_id AND
 ↪  rtj1.tag_id < rtj2.tag_id
GROUP BY rtj1.tag_id, rtj2.tag_id
ORDER BY appearances DESC
LIMIT 3;
```

Βλέπουμε τώρα πολύ περισσότερες γραμμές να πρέπει να επεξεργασθούν, σε αντίθεση με πριν, επομένως αν είχαμε αρκετά περισσότερα δεδομένα, θα υπήρχε σοβαρή καθυστέρηση. Βλέπουμε λοιπόν, ότι ο optimizer της βάσης επιλέγει την σωστή σειρά των join operations για να μειώσει όσο το δυνατόν περισσότερο τις γραμμές.

```
+----+-------------+-------+------------+--------+---------------------+
| id | select_type | table | partitions | type   | possible_keys       |
+----+-------------+-------+------------+--------+---------------------+
|  1 | SIMPLE      | rmt   | NULL       | index  | PRIMARY,meal_tag_id |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY             |
|  1 | SIMPLE      | mt    | NULL       | eq_ref | PRIMARY             |
|  1 | SIMPLE      | rmt   | NULL       | ref    | PRIMARY,meal_tag_id |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY             |
|  1 | SIMPLE      | mt    | NULL       | eq_ref | PRIMARY             |
+----+-------------+-------+------------+--------+---------------------+

+-------------+---------+-----------------------------+------+
| key         | key_len | ref                         | rows |
+-------------+---------+-----------------------------+------+
| meal_tag_id | 4       | NULL                        | 3537 |
| PRIMARY     | 4       | chefmaesters.rmt.recipe_id  |    1 |
| PRIMARY     | 4       | chefmaesters.rmt.meal_tag_id|    1 |
| PRIMARY     | 4       | chefmaesters.rmt.recipe_id  |    4 |
| PRIMARY     | 4       | chefmaesters.rmt.recipe_id  |    1 |
| PRIMARY     | 4       | chefmaesters.rmt.meal_tag_id|    1 |
+-------------+---------+-----------------------------+------+

+----------+-----------------------------------------------+
| filtered | Extra                                         |
+----------+-----------------------------------------------+
|   100.00 | Using index; Using temporary; Using filesort |
|   100.00 | Using index                                   |
|   100.00 | NULL                                          |
|   100.00 | Using index                                   |
|   100.00 | Using index                                   |
|   100.00 | Using where                                   |
+----------+-----------------------------------------------+
```

Ερώτηση 5.7 *Βρείτε όλους τους μάγειρες που συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον μάγειρα με τις περισσότερες συμμετοχές σε επεισόδια.*

Και πάλι σε αυτό το query παρατηρείται η χρήση της εντολής `WITH`. Η προσωρινή σχέση κρατάει το πλήθος των επεισοδίων στα οποία εμφανίστηκε ο κάθε μάγειρας. Με την εντολή `WHERE` φιλτράρονται για να δοθούν στην τελική απάντηση όσοι συμμετείχαν τουλάχιστον 5 λιγότερες φορές από τον μάγειρα με τις περισσότερες συμμετοχές σε επεισόδια κι αυτοί παρατίθενται στον χρήστη σε φθίνουσα σειρά.

```
WITH contestant_appearance_counts AS (
    SELECT ec.contestant_id AS contestant_id, count(ec.contestant_id) AS
     ↪  contestant_appearances
    FROM episode_contestants ec
    GROUP BY ec.contestant_id
)
SELECT cac.contestant_id, cac.contestant_appearances
FROM contestant_appearance_counts cac
WHERE cac.contestant_appearances >= (SELECT MAX(contestant_appearances) FROM
 ↪  contestant_appearance_counts) - 5
ORDER BY cac.contestant_appearances DESC;
```

Ερώτηση 5.8 *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.*

Το συγκεκριμένο ερώτημα είναι αρκετά απλό, καθώς απαιτεί μόνο ένα `JOIN` και ένα `GROUP BY` . Προφανώς, ο optimizer βρίσκει τη βέλτιστη λύση, που φαίνεται παρακάτω.

```
SELECT r.id AS id, r.title AS recipe_title, COUNT(re.equipment_id) AS
 ↪  equipment_count
FROM recipe_equipment re
INNER JOIN recipe r ON r.id = re.recipe_id
GROUP BY r.id
ORDER BY equipment_count DESC
LIMIT 1;
```

```
+----+-------------+-------+------------+-------+
| id | select_type | table | partitions | type  |
+----+-------------+-------+------------+-------+
|  1 | SIMPLE      | r     | NULL       | index |
|    |             |       |            |       |
|  1 | SIMPLE      | re    | NULL       | ref   |
+----+-------------+-------+------------+-------+

+-------------------------------------------------------------------+
| possible_keys                                                     |
+-------------------------------------------------------------------+
| PRIMARY,cuisine_id,image_id,recipe_title_index,prep_time_index,   |
|       cook_time_index,unit_count_index                            |
| PRIMARY,recipe_id_index                                           |
+-------------------------------------------------------------------+

+---------+---------+------------------+------+
| key     | key_len | ref              | rows |
+---------+---------+------------------+------+
| PRIMARY | 4       | NULL             |  990 |
|         |         |                  |      |
| PRIMARY | 4       | chefmaesters.r.id |    3 |
+---------+---------+------------------+------+

+----------+--------------------------------+
| filtered | Extra                          |
+----------+--------------------------------+
|   100.00 | Using temporary; Using filesort |
|          |                                |
|   100.00 | Using index                    |
+----------+--------------------------------+
```

Μπορούμε όμως να αναγκάσουμε τον optimizer να αγνοήσει εντελώς τα indices οπότε θα δούμε πόσο περισσότερες γραμμές χρειάζεται να ελέγξουμε τώρα. Σε κάθε περίπτωση τα queries μας είναι πολύ μικρά και χωράνε στη μνήμη οπότε δεν κερδίζουμε πολλά με τη χρήση των indices αλλά αν θεωρήσουμε ότι η βάση αυτή θα επεκταθεί και θα περιλαμβάνει δεκάδες χιλιάδες εγγραφές από συνταγές και εξοπλισμό τότε θα μπορούσε να φανεί η αξία των indices.

```
EXPLAIN SELECT r.id AS id, r.title AS recipe_title, COUNT(re.equipment_id) AS
 ↪  equipment_count
FROM recipe_equipment re
USE INDEX ()
INNER JOIN recipe r
USE INDEX ()
ON r.id = re.recipe_id
GROUP BY r.id
ORDER BY equipment_count DESC
LIMIT 1;
```

```
+----+-------------+-------+------------+------+---------------+
| id | select_type | table | partitions | type | possible_keys |
+----+-------------+-------+------------+------+---------------+
|  1 | SIMPLE      | r     | NULL       | ALL  | NULL          |
|  1 | SIMPLE      | re    | NULL       | ALL  | NULL          |
+----+-------------+-------+------------+------+---------------+

+------+---------+------+------+----------+
| key  | key_len | ref  | rows | filtered |
+------+---------+------+------+----------+
| NULL | NULL    | NULL |  990 |   100.00 |
| NULL | NULL    | NULL | 2134 |     0.16 |
+------+---------+------+------+----------+

+-----------------------------------------------+
| Extra                                         |
+-----------------------------------------------+
| Using temporary; Using filesort               |
| Using where; Using join buffer (hash join) |
+-----------------------------------------------+
```

Εδώ βλέπουμε την ουσιαστική διαφορά πώς ελέγχονται όλες οι γραμμές. Βέβαια και πριν όλες ελέγχονταν ουσιαστικά απλά δεν το ήξερε ο optimizer. Εδώ έχει αλλάξει ο τρόπος που γίνεται το `JOIN` . Ενώ πριν γινόταν με χρήση του index τώρα γίνεται με hash join.

Ερώτηση 5.9 *Λίστα με μέσο όρο αριθμού γραμμάριων υδατανθράκων στο διαγωνισμό ανά έτος;*

Κατά την υλοποίηση αυτού του query υπολογίζεται ο μέσος όρος ποσότητας υδατανθράκων ανά μονάδα για κάθε σεζόν. Αξιοποιούμε το `view` για τα θρεπτικά συστατικά και με τα κατάλληλα `JOIN` επιλέγουμε όλες τις συνταγές για κάθε επεισόδιο. Τα ζητούμε αποτελέσματα είναι ανά έτος γι'αυτό και ο μέσος όρος υπολογίζεται ανα `GROUP` των season.

```
SELECT season_id, AVG(rnv.recipe_carbohydrates_per_unit) AS
 ↪  average_carbohydrates_per_season
FROM episode_contestants ec
INNER JOIN recipe_nutritional_values rnv ON ec.recipe_id = rnv.recipe_id
INNER JOIN episode e ON ec.episode_id = e.id
GROUP BY e.season_id;
```

Ερώτηση 5.10 *Ποιες Εθνικές κουζίνες έχουν τον ίδιο αριθμό συμμετοχών σε διαγωνισμούς, σε διάστημα δύο συνεχόμενων ετών, με τουλάχιστον 3 συμμετοχές ετησίως;*

Στο πρώτο σκέλος, υποτίθεται ότι η ισότητα των συμμετοχών αφορά αθροιστικά τα δύο έτη κι όχι το κάθε έτος ξεχωριστά, ενώ οι εξεταζόμενες σεζόν για την κάθε κουζίνα δεν απαιτείται να είναι οι ίδιες, αλλά οποιεσδήποτε δύο συνεχόμενες που ικανοποιούν την ισότητα. Στην εσωτερική προσωρινή

σχέση υπολογίζονται οι εμφανίσεις για κάθε εθνική κουζίνα οι οποίες ικανοποιούν τον περιορισμό των τουλάχιστον τριών συμμετοχών ετησίως. Ενώ στην εξωτερική προσωρινή σχέση κρατάμε τον συνολικό αριθμό εμφανίσεων κάθε κουζίνας μεταξύ ζευγαριών σε συναπτά έτη.

```
-- Assume the seasons are not necessarily identical
WITH total_valid_appearances AS (
    WITH valid_appearances AS (
        SELECT
            e.season_id AS season_id,
            c.id AS cuisine_id,
            count(c.id) AS yearly_appearances
        FROM episode_contestants ec
        INNER JOIN recipe r ON ec.recipe_id = r.id
        INNER JOIN cuisine c ON c.id = r.cuisine_id
        INNER JOIN episode e ON ec.episode_id = e.id
        GROUP BY e.season_id, c.id
        HAVING yearly_appearances > 3
    )
    SELECT va1.cuisine_id AS cuisine_id, va1.season_id AS season_id,
     ↪  (va1.yearly_appearances + va2.yearly_appearances) AS total_appearances
    FROM valid_appearances va1
    INNER JOIN valid_appearances va2 ON va1.cuisine_id = va2.cuisine_id
    WHERE va1.season_id = va2.season_id - 1
)
SELECT tva1.cuisine_id, tva1.season_id, tva2.cuisine_id, tva2.season_id,
 ↪  tva1.total_appearances
FROM total_valid_appearances tva1
INNER JOIN total_valid_appearances tva2 ON tva1.total_appearances =
 ↪  tva2.total_appearances AND tva1.cuisine_id < tva2.cuisine_id
ORDER BY total_appearances DESC;
```

Μια άλλη ερμηνεία απαιτεί οι συνθήκες να ισχύουν για τις ίδιες σεζόν για τα ζεύγη των κουζινών.

```
-- Assume the seasons must be the same
WITH total_valid_appearances AS (
    WITH valid_appearances AS (
        SELECT
            e.season_id AS season_id,
            c.id AS cuisine_id,
            count(c.id) AS yearly_appearances
        FROM episode_contestants ec
        INNER JOIN recipe r ON ec.recipe_id = r.id
        INNER JOIN cuisine c ON c.id = r.cuisine_id
        INNER JOIN episode e ON ec.episode_id = e.id
        GROUP BY e.season_id, c.id
        HAVING yearly_appearances > 3
    )
    SELECT va1.cuisine_id AS cuisine_id, va1.season_id AS season_id,
     ↪  (va1.yearly_appearances + va2.yearly_appearances) AS total_appearances
    FROM valid_appearances va1
    INNER JOIN valid_appearances va2 ON va1.cuisine_id = va2.cuisine_id
    WHERE va1.season_id = va2.season_id - 1
)
SELECT tva1.cuisine_id, tva2.cuisine_id, tva1.total_appearances
FROM total_valid_appearances tva1
INNER JOIN total_valid_appearances tva2 ON tva1.total_appearances =
 ↪  tva2.total_appearances AND tva1.season_id = tva2.season_id AND
 ↪  tva1.cuisine_id < tva2.cuisine_id
ORDER BY total_appearances DESC;
```

Τέλος, σύμφωνα με άλλη ερμηνεία βρίσκουμε τα ζεύγη από κουζίνες που είχαν τον ίδιο αριθμό εμφανίσεων για κάθε χρονιά, δύο χρόνια στη σειρά.

```
-- Assume that we are looking for the cuisines that had the same number of
 ↪  appearances two years in a row
WITH valid_appearances AS (
    SELECT
        e.season_id AS season_id,
        c.id AS cuisine_id,
        count(c.id) AS yearly_appearances
    FROM episode_contestants ec
    INNER JOIN recipe r ON ec.recipe_id = r.id
    INNER JOIN cuisine c ON c.id = r.cuisine_id
    INNER JOIN episode e ON ec.episode_id = e.id
    GROUP BY e.season_id, c.id
    HAVING yearly_appearances > 3
)
SELECT va1.cuisine_id, va1.season_id AS from_season, va2.season_id AS to_season,
 ↪  va1.yearly_appearances AS common_appearances
FROM valid_appearances va1
INNER JOIN valid_appearances va2 ON va1.yearly_appearances =
 ↪  va2.yearly_appearances AND va1.cuisine_id = va2.cuisine_id
WHERE va1.season_id = va2.season_id - 1
ORDER BY common_appearances DESC;
```

Ερώτηση 5.11 *Βρείτε τους top-5 κριτές που έχουν δώσει συνολικά την υψηλότερη βαθμολόγηση σε ένα μάγειρα. (όνομα κριτή, όνομα μάγειρα και συνολικό σκορ βαθμολόγησης)*

Το παρόν query ψάχνει τους 5 υψηλότερους συνδυασμούς κριτών-διαγωνιζόμενων από άποψη βαθμολογίας. Υπολογίζει το συνολικό σκορ που ο κάθε κριτής έχει δώσει σε κάθε διαγωνιζόμενο και ταξινομεί τα αποτελέσματα σε φθίνουσα σειρά, περιορίζοντας το αποτέλεσμα στους πέντε πρώτους.

```
SELECT
    (CONCAT(cc.name, ' ', cc.surname)) as contestant_full_name,
    (CONCAT(cj.name, ' ', cj.surname)) as judge_full_name,
    sum(s.score) as total_score
FROM scoring s
INNER JOIN cook cc ON cc.id = s.contestant_id
INNER JOIN cook cj ON cj.id = s.judge_id
GROUP BY s.contestant_id, s.judge_id
ORDER BY total_score DESC, s.contestant_id, s.judge_id ASC
LIMIT 5;
```

Ερώτηση 5.12 *Ποιο ήταν το πιο τεχνικά δύσκολο, από πλευράς συνταγών, επεισόδιο του διαγωνισμού ανά έτος;*

Στο query αυτό, η προσωρινή σχέση υπολογίζει το βαθμό δυσκολίας του κάθε επεισοδίου, συναρτήσει του βαθμού δυσκολίας όλως των συνταγών που συμπεριλήφθησαν στο επεισόδιο. Στις γραμμές κώδικα που ακολουθούν, υπολογίζεται ο μέγιστος αριθμός δυσκολίας που εμφανίστηκε σε επεισόδιο για κάθε σεζόν. Εδώ ο συμβολισμός `.*` σημαίνει πάρε όλες τις στήλες.

```
WITH total_episode_difficulty AS (
    SELECT e.*, sum(r.difficulty) as total_difficulty
    FROM episode e
    INNER JOIN episode_contestants ec ON e.id = ec.episode_id
    INNER JOIN recipe r ON r.id = ec.recipe_id
    GROUP BY e.id
)
SELECT ted2.*
FROM (
    SELECT ted.season_id AS id, max(ted.total_difficulty) AS total_difficulty
    FROM total_episode_difficulty ted
    GROUP BY ted.season_id
) AS max_season_difficulties
INNER JOIN total_episode_difficulty ted2 ON
    ted2.season_id = max_season_difficulties.id AND
    max_season_difficulties.total_difficulty = ted2.total_difficulty;
```

Ερώτηση 5.13 *Ποιο επεισόδιο συγκέντρωσε τον χαμηλότερο βαθμό επαγγελματικής κατάρτισης (κριτές και μάγειρες);*

Στο ερώτημα αυτό, υπολογίζεται το άθροισμα του επιπέδου των κριτών και των διαγωνιζόμενων. Το `tjl` υπολογίζει το συνολικό επίπεδο των κριτών για κάθε επεισόδιο, ενώ το `tcl` επιστρέφει το αντίστοιχο μέγεθος για τους διαγωνιζομένους. Στο κατώτερο μέρος του κώδικα, συνδυάζονται τα αποτελέσματα των δύο subqueries κρατώντας το συνολικό επίπεδο όλων των μαγείρων που συμμετέχουν σε κάθε επεισόδιο. Τέλος, υπολογίζεται το επεισόδιο με το χαμηλότερο βαθμό επαγγελματικής κατάρτισης.

```
SELECT e2.*, tjl.total_judge_level+tcl.total_contestant_level as total_level
FROM
(
    SELECT e.id as id, sum(cl.id) as total_judge_level
    FROM episode_judges ej
    INNER JOIN cook c ON ej.judge_id = c.id
    INNER JOIN cook_level cl ON c.level_id = cl.id
    INNER JOIN episode e ON e.id = ej.episode_id
    GROUP BY e.id
) AS tjl,
(
    SELECT e.id as id, sum(cl.id) as total_contestant_level
    FROM episode_contestants ec
    INNER JOIN cook c ON ec.contestant_id = c.id
    INNER JOIN cook_level cl ON c.level_id = cl.id
    INNER JOIN episode e ON e.id = ec.episode_id
    GROUP BY e.id
) AS tcl
INNER JOIN episode e2 ON tcl.id = e2.id
WHERE tcl.id = tjl.id
ORDER BY total_level ASC
LIMIT 1;
```

Ερώτηση 5.14 *Ποια θεματική ενότητα έχει εμφανιστεί τις περισσότερες φορές στο διαγωνισμό;*

Εδώ αντιστοιχίζονται συνταγές σε θεματικές ενότητες, ενώ με το count καταγράφονται οι εμφανίσεις έκαστου id του thematic area. Η θεματική περιοχή με τον μεγαλύτερο αριθμό count είναι αυτή με τις περισσότερες εμφανίσεις στο διαγωνισμό.

```
SELECT ta.*, count(ta.id) as appearances
FROM episode_contestants ec
INNER JOIN recipe_thematic_area rta ON rta.recipe_id = ec.recipe_id
INNER JOIN thematic_area ta ON rta.thematic_area_id = ta.id
GROUP BY ta.id
ORDER BY count(ta.id) DESC
LIMIT 1;
```

Ερώτηση 5.15 *Ποιες ομάδες τροφίμων δεν έχουν εμφανιστεί ποτέ στον διαγωνισμό;*

Το query στην προκειμένη περίπτωση παίρνει όλες τις στήλες του πίνακα ”food group” που δεν ανήκουν σε ένα subquery μέσα στο `WHERE` . Το subquery βρίσκει όλες τις ομάδες τροφίμων που εμφανίστηκαν στον διαγωνισμό.

```
SELECT fg.*
FROM food_group fg
WHERE fg.id NOT IN (
    SELECT DISTINCT fg.id
    FROM episode_contestants ec
    INNER JOIN recipe_ingredients ri ON ec.recipe_id = ri.recipe_id
    INNER JOIN ingredient i ON ri.ingredient_id = i.id
    INNER JOIN food_group fg ON i.food_group_id = fg.id
);
```