Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εξάμηνο: 6ο

Μάθημα: Βάσεις Δεδομένων

Αναφορά Εξαμηνιαίας Εργασίας

Θέμα: Υλοποίηση βάσης δεδομένων για δημοφιλή διαγωνισμό μαγειρικής

GitHub repo link: https://github.com/Yassin1-prog/cooking-competition-DBMS

Ομάδα: Project 25
Συνεργάτες:

Αχμέντ Γιασίν 03121161 Κουμπιάς Ιωάννης - Χρυσοβαλάντης 03121053

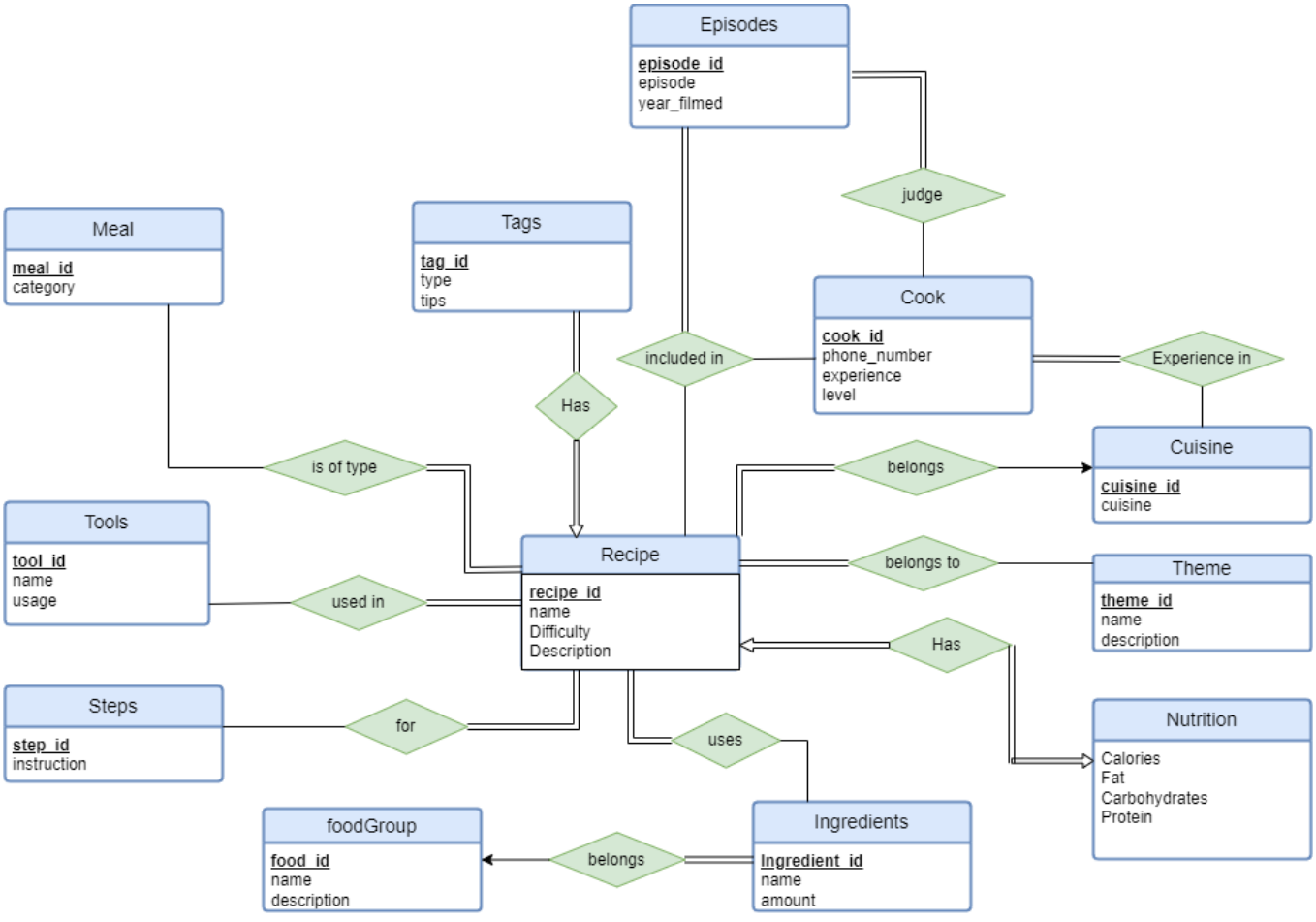
Εισαγωγή

Με την παρούσα αναφορά παρουσιάζουμε την βάση δεδομένων που υλοποιήσαμε για έναν δημοφιλή διαγωνισμό μαγειρικής. Το σύστημά μας είναι πλήρως ικανό να αποθηκεύσει και να διαχειριστεί τις πληροφορίες που απαιτούνται για την λειτουργία του διαγωνισμού σχετικά με τις συνταγές, τα υλικά, τον εξοπλισμό, τους μάγειρες και τα επεισόδια.

Προδιαγραφές υλοποίησης

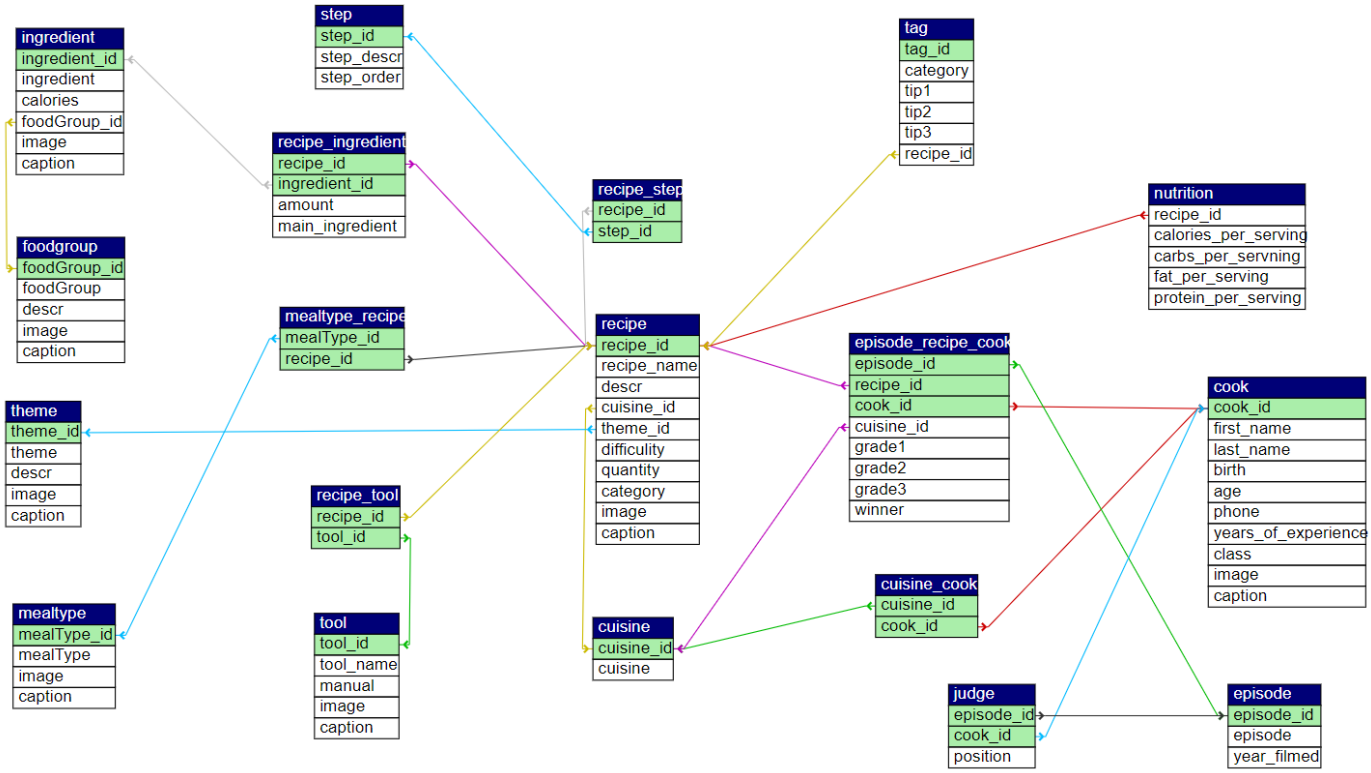
- Κάθε ετικέτα μπορεί να έχει μέχρι 3 tips, σε περίπτωση που χρειάζονται περισσότερα από 3 για μια συνταγή θα πρέπει να φτιαχτεί επιπλέον ετικέτα για αυτή.
- Θεωρούμε ότι τα protein/fat/carbs per serving κάθε συνταγής δίνονται έτοιμα από τους διοργανωτές του διαγωνισμού και εμείς αρκεί να υπολογίσουμε τις θερμίδες per serving.
- Οι θερμίδες συνταγής (calories) θεωρείται ακέραιος αριθμός (στρογγυλοποιημένος), όπως στη πραγματικότητα.
- Για υλικά όπως αλάτι/πιπέρι με ασαφείς ποσότητες (λίγο αλάτι) θεωρούμε ότι δε συμπεριλαμβάνονται στις διατροφικές πληροφορίες μιας συνταγής (εφόσον περιέχουν αμελητέες ποσότητες θερμιδών).
- Θερμίδες όλων των υλικών θα δίνονται ανα 100γρ/100ml.
- Ένα άτομο μπορεί να βρίσκεται σε περισσότερα από 3 συνεχόμενα επεισόδια με τη προυπόθεση ότι δεν ήταν μόνο μάγειρας ή μόνο κριτής.
- Ο παραπάνω περιορισμός (3 συνεχόμενα επεισόδια) ανανεώνεται με την αλλαγή χρονιάς.

Διάγραμμα Οντοτήτων – Συσχετίσεων (ER Diagram)



Σχεσιακό Διάγραμμα (Relational Schema)

Παρακάτω φαίνεται το σχεσιακό διάγραμμα της βάσης όπως προέκυψε από την κατασκευή του συστήματος μέσω του DDL script. Στο σχεσιακό διάγραμμα διακρίνουμε τόσο τα tables που αναφέρονται σε οντότητες της βάσης, όσο και tables που υλοποιούν τις σχέσεις οι οποίες περιγράφονται στο E-R Diagram. Τέλος στο schema φαίνονται και τα primary και foreign keys που χρησιμοποιήθηκαν στον κάθε πίνακα (πράσινα πεδία):



Constraints Βάσης Δεδομένων

- Οι μερίδες (quantity) κάθε συνταγής πρέπει να είναι θετικός ακέραιος αριθμός
- Η σειρά εκτέλεσης ενός βήματος (step_order) πρέπει να είναι θετικός ακέραιος αριθμός.
- Οι θερμίδες ανα μερίδα ενός συνταγής πρέπει να είναι θετικός ακέραιος αριθμός.
- Ένας μάγειρας δε μπορεί να είναι κάτω των 18 ετών.
- Ελέγχουμε ότι η ηλικία ενός μάγειρα συμφωνεί με την ημερομηνία γεννησής του.
- Οι βαθμολογίες που δίνονται από τους κριτές πρέπει να ανήκουν στη κλίμακα 1-5.
- Ελέγχουμε ότι η επαγγελματική κατάρτιση ενός μάγειρα ανήκει στις αποδεκτές τιμές.
- Ελέγχουμε τη σειρά ενός κριτή και επιβεβαιώνουμε ότι είναι στο Range 1-3 μέσω Check αλλιώς σε περίπτωση invalid value θα εισηχθεί empty by default.
- Κάθε συνταγή μπορεί να έχει μόνο ένα βασικό υλικό.
- Ενας κριτής/μάγειρας/κουζίνα/συνταγή δε μπορούν να συμμετέχουν συνεχόμενα σε πάνω από 3 επεισόδια.
- Ένας μάγειρας μπορεί να μαγειρέψει μόνο τις συνταγές που ανήκουν στις εθνικές κουζίνες στις οποίες έχει εμπειρία.
- Ένας κριτής δε μπορεί να είναι και μάγειρας στο ίδιο επεισόδιο.
- Η κλίμακα δυσκολίας μιας συνταγής είναι στο διάστημα 1 εως 5.
- Τα έτη εμπειρίας ενός μάγειρα να είναι μη αρνητικός ακέραιος αριθμός.

- Σε ένα επεισόδιο μπορεί να υπάρχει μόνο ένας νικητής.

Όλα τα προηγούμενα εξασφαλίζονται με τη χρήση SQL constraints και διαφόρων triggers τα οποία υπάρχουν στο DDL script.

Ευρετήρια Βάσης Δεδομένων

Είναι πολύ σημαντικό να λαμβάνουμε υπόψιν ότι στον MariaDB server με InnoDB engine δημιουργούνται αυτόματα clustered indices στα primary keys columns του κάθε table. Αυτό είναι πολύ χρήσιμο αφού τα primary keys στη βάση μας είναι ID's (auto_incremented) και χρησιμοποιούνται πολύ συχνά στα διάφορα queries ([WHERE], [JOIN ON], [ORDER BY]) καθώς και στα triggers τα οποία επιβάλλουν τους περιορισμούς της βάσης μας. Έτσι η ύπαρξη αυτών των indices οδηγεί στη γρήγορη πρόσβαση στα διάφορα primary keys και ως αποτέλεσμα αυξάνει την αποδοτικότητα της βάσης μας.

Στη συνέχεια καλή τακτική είναι να ορίσουμε index για τα foreign keys που δεν είναι primary keys σε άλλο table. Ωστόσο στη υλοποίησή μας δεν υφίσταται αυτό το ενδεχόμενο. Αξίζει να σημειωθεί ότι τα foreign keys έχουν αυτομάτως non unique index, σε αντίθεση με τα primary keys των οποίων τα index είναι unique. Η διαχείρισή τους από τον MariaDB optimizer όμως είναι διαφορετική, με τα unique index να είναι πιο γρήγορα, όπως στο παράδειγμα ενός Insert για το οποίο δε θα χρειαστεί έλεγχος uniqueness πριν να εγκριθεί, ή ενός Select που μόλις βρει τη ζητούμενη τιμή δε θα συνεχίσει την αναζήτηση αφού γνωρίζει ότι είναι μοναδική. Φυσικά αν ο όγκος δεδομένων στη βάση μας είναι σχετικά μικρός οι παραπάνω διαφορές θα είναι αμελητέες.

Επίσης χρήσιμη είναι και η ύπαρξη ορισμένων column attributes που είναι Unique και άρα αποκτούν αυτόματα indexes ,όπως **mealType, theme, cuisine**, αφού ερωτήσεις τύπου «ποιες συνταγές είναι για πρωινό» ή «ποιες συνταγές ανήκουν στη μεξικάνικη κουζίνα» θα είναι πολύ συχνές. Στην εργασία μας αυτό επιβεβαιώνεται από queries όπως το 3.14 που χρησιμοποιεί ORDER BY COUNT(theme).

Ως προς τα composite primary keys που επίσης αυτόματα έχουν το δικό τους index, ισχύει ότι αν υπάρχει για παράδειγμα το primary key (a,b,c,d) τότε το index του θα μπορεί να χρησιμοποιηθεί σε queries με φιλτράρισμα πάνω στο (a), (a,b), (a,b,c). Για τα άλλα όμως υποσύνολα όπως (b,c) ή (b) θα πρέπει να δημιουργηθεί νέο αντοίστιχο index. Στη βάση μας όλα τα composite primary keys αποτελούνται από foreign keys και άρα δεν υφίσταται το συγκεκριμένο ζήτημα.

Τέλος δημιουργούμε indexes σε columns που χρησιμοποιούνται συχνά σε queries στο φιλτάρισμα, join και order by όπως για παράδειγμα το year_filmed του episode table καθώς και το age του cook table.

- year_filmed του table episode (χρήση στο φιλτράρισμα επεισοδίων)
- age του table cook (χρήση στο φιλτράρισμα μαγείρων)

Άρα τελικά η βάση δεδομένων μας περιέχει διάφορα indexes αυτόματα δημιουργημένα επάνω σε primary/foreign keys (single και composite), επάνω σε unique columns καθώς και κάποια που φτιάξαμε εμείς, τα οποία όλα αναφέρονται παραπάνω. Έτσι εξασφαλίσαμε ότι οι δείκτες που δημιουργήσαμε είναι απαραίτητοι και συμβάλλουν στην αποτελεσματική διαχείριση των δεδομένων χωρίς να επιβαρύνουν το σύστημα.

DDL script

Το DDL script που κατασκευάζει τα tables του συστήματος καθώς και τα σχετικά indexes, constraints, stored procedures και triggers βρίσκεται στο GitHub repository στο path «SQL Code/db_schema.sql». Επίσης ο κώδικας παρουσιάζεται και στο τέλος της αναφοράς.

DML script – Τυχαία εισαγωγή επεισοδίων

Το DML script που γεμίζει την βάση με mock data για τις διάφορες οντότητές της και τις σχέσεις τους βρίσκεται στο GitHub repository στο path «SQL Code/insert_data.sql». Αξίζει να σημειωθεί πως η εισαγωγή των επεισοδίων έχει γίνει με ξεχωριστό script το οποίο είναι γραμμένο σε Python 3.11.1 και χρειάζεται τη βιβλιοθήκη «mysql-connector» (βλ. GitHub) που υλοποιεί instances από επεισόδια τύχαιων χαρακτηριστικών (εθνικές κουζίνες, μάγειρες που συμμετέχουν και συνταγές που εκτελούν). Κατόπιν αφού βεβαιωθεί πως ανταποκρίνονται στους περιορισμούς που θέτει η εκφώνηση όσον αφορά το συγκεκριμένο ζήτημα (κανόνας των 3 διαδοχικών), τα εισάγει στην βάση με κατάλληλα INSERT queries. Οι εισαγωγές που πραγματοποιήθηκαν εκ των προτέρων, έχουν εισηχθεί στο παραπάνω DML μέσω export από το phpmyadmin συνεπώς δεν απαιτείται η εκτέλεση του python προγράμματος για την ορθή προσθήκη των απαραίτητων δεδομένων.

Εξακρίβωση ταυτότητας – δικαιωμάτων χρήστη (Authentication)

Στο αρχείο «applicationUsers.sql» δείχνουμε ενδεικτικά πως θα μπορούσαμε να δημιουργήσουμε τους χρήστες της εφαρμογής μέσω κώδικα SQL. Δημιουργούμε ένα table user που αποθηκεύει τα login στοιχεία όλων των χρήστων της βάσης. Έπειτα δημιουργούμε δύο views τα οποία περιέχουν τα προσωπικά στοιχεία του μάγειρα καθώς και όλες τις συνταγές που του έχουν ανατεθεί σε επεισόδια στο διαγωνισμό (στην περίπτωση του admin αυτά τα πεδία θα είναι κενά). Τέλος μέσω της εντολής GRANT δίνουμε τα κατάλληλα δικαιώματα στον κάθε μάγειρα ώστε να μπορεί να δεί και να αλλάξει τα στοιχεία που αναφέρονται μόνο σε αυτόν.

Σημείωση/Πιθανή ιδέα επέκτασης: Σε περίπτωση που ζητούνταν full stack εφαρμογή τότε στο backend θα μπορούσαμε κάθε φορά που έκανε sign up ένας χρήστης να τον κάναμε insert στο table users και μετά να τρέχαμε αυτόματα το υπόλοιπο SQL script εξασφαλίζοντας έτσι το integrity και το consistency του συτήματός μας. Με τον τρόπο αυτό θα παραμένει ανεπηρέαστη η βάση από τις αλλαγές που μπορεί να κάνει ο κάθε μάγειρας.

Οδηγίες Εγκατάστασης

Σύνδεσμος για το GitHub repository: https://github.com/Yassin1-prog/cooking-competition-DBMS

Κατεβάστε τα απαραίτητα αρχεία που προαναφέρθηκαν:

«db_schema.sql», «insert_data.sql» , «add_episodes.py», «applicationUsers.sql»

Για την χρήση της βάσης δεδομένων είναι απαραίτητη η εγκατάσταση του DBMS MariaDB. Συνδεθείτε στο DBMS εκτελώντας στο bin φάκελο του web development stack της επιλογής σας το command:

```
./mysql -u root -p
```

Αν επιθυμείτε να διαχειριστείτε τη βάση. Στη συνέχεια για τη δημιουργία του database χρησιμοποιείστε το command:

```
source < path to db_schema.sql >
```

Στη συνέχεια για να γεμίσετε τη βάση με mock data εκτελέστε το command:

```
source < path to insert_data.sql >
```

Η βάση δεδομένων είναι έτοιμη για χρήση. Στην περίπτωση που θέλετε να προσθέσετε επιπλέον επεισόδια στην βάση αρκεί να τρέξετε το πρόγραμμα «add_episodes.py» εκτελώντας στο terminal:

```
python < path to add_episodes.py >
```

Στο prompt που θα ακολουθήσει εισάγετε το έτος έναρξης «προβολής» των επεισοδίων και το έτος λήξης. Για παράδειγμα εάν επιθυμείτε να εισάγετε επεισόδια για τα έτη 2035, 2036 και 2037 θα εισάγετε το 2035 ως start year και το 2037 ως end year. Από την άλλη εάν θέλετε να εισάγετε το 2038 τότε θα το εισάγετε ως start year και ως end year. Προσοχή στην εγκυρότητα των δεδομένων εισαγωγής! Το script μπορείτε να το εκτελέσετε όσες φορές θέλετε αρκεί τα εισαγόμενα έτη να αυξάνονται σε τιμή, ώστε να διατηρείται η χρονική εγκυρότητα της λειτουργίας του διαγωνισμού. Τέλος στην περίπτωση που επιθυμείτε να ελέγξετε τη συμπεριφορά της βάσης στο σενάριο που κάποιος μάγειρας του διαγωνισμού συνδεθεί σε αυτήν ως user μπορείτε να εκτελέσετε στο MariaDB το command:

```
source < path to applicationUsers.sql >
```

Έτσι θα δημιουργήσετε δύο users, έναν admin με δικαιώματα διαχειριστή στη βάση και καθόλου προσωπικές πληροφορίες ή συνταγές και έναν μάγειρα cook_1 ο οποίος αντιστοιχεί στον μάγειρα που εισήχθη στην βάση με cook_id ίσο με 1. Αναλόγως με ποιον user επιλέξετε να συνδεθείτε στο database τα δικαιώματα σας θα είναι διαφορετικά κατά τα ζητούμενα της εκφώνησης. Τα passwords των χρηστών φαίνονται στο ίδιο αρχείο SQL στα queries εισαγωγής των users.

Queries

```sql
/*3.1*/
SELECT c.cook_id, concat(c.first_name, " ", c.last_name) cook_name, cu.cuisine,
AVG((ep_rc.grade1+ep_rc.grade2+ep_rc.grade3)/3) avg_grade
FROM episode_recipe_cook ep_rc
INNER JOIN cook c
ON ep_rc.cook_id = c.cook_id
INNER JOIN cuisine cu
ON ep_rc.cuisine_id = cu.cuisine_id
GROUP BY ep_rc.cook_id,ep_rc.cuisine_id;

/*3.2*/
SELECT ep.year_filmed, ep.episode, cu.cuisine, concat(c.first_name," ",c.last_name)
cook_name, c.cook_id
FROM episode ep
INNER JOIN episode_recipe_cook ep_rc
ON ep.episode_id = ep_rc.episode_id
INNER JOIN cuisine cu
ON ep_rc.cuisine_id = cu.cuisine_id
INNER JOIN cook c
ON ep_rc.cook_id = c.cook_id;

/*3.3*/
SELECT c.cook_id, concat(c.first_name, " ", c.last_name) as cook, c.age, COUNT(erc.cook_id)
as recipe_count FROM cook c
INNER JOIN episode_recipe_cook erc ON erc.cook_id = c.cook_id
WHERE c.age < 30
GROUP BY erc.cook_id
ORDER  BY COUNT(erc.cook_id) DESC;
```

```sql
/*3.4*/
SELECT c.cook_id, concat(c.first_name, " ", c.last_name) not_judge FROM cook c
WHERE c.cook_id NOT IN (SELECT j.cook_id FROM judge j);

/*3.5*/
WITH JudgeAppearances AS (
    SELECT
        j.cook_id,
        e.year_filmed,
        COUNT(j.episode_id) AS appearances
    FROM
        judge j
    JOIN
        episode e ON j.episode_id = e.episode_id
    GROUP BY
        j.cook_id, e.year_filmed
    HAVING
        COUNT(j.episode_id) > 3
),
JudgeSameAppearances AS (
    SELECT
        ja1.cook_id AS judge1_id,
        ja2.cook_id AS judge2_id,
        ja1.year_filmed,
        ja1.appearances
    FROM
        JudgeAppearances ja1
    JOIN
        JudgeAppearances ja2
    ON
        ja1.year_filmed = ja2.year_filmed AND ja1.appearances = ja2.appearances AND
ja1.cook_id < ja2.cook_id
)
SELECT
    concat(j1.first_name," ",j1.last_name) as judge_1,
    concat(j2.first_name," ",j2.last_name) as judge_2,
    jsa.year_filmed,
    jsa.appearances
FROM
    JudgeSameAppearances jsa
JOIN
    cook j1 ON jsa.judge1_id = j1.cook_id
JOIN
    cook j2 ON jsa.judge2_id = j2.cook_id
ORDER BY
    jsa.year_filmed, jsa.appearances DESC;

/*3.6*/
```

```sql
WITH pairs AS (
    SELECT t1.category category1, t2.category category2, ep_rc.recipe_id
    FROM tag t1
    INNER JOIN tag t2
    ON t1.recipe_id = t2.recipe_id
    INNER JOIN episode_recipe_cook ep_rc
    ON t1.recipe_id = ep_rc.recipe_id
    WHERE t1.tag_id<t2.tag_id)
SELECT count(recipe_id) as appearances, category1, category2 FROM pairs
GROUP BY category1,category2
ORDER BY count(recipe_id) DESC
LIMIT 3;

-- Alternative query plan with ignore index (execute with explain to see the traces)
WITH pairs AS (
    SELECT t1.category AS category1, t2.category AS category2, ep_rc.recipe_id
    FROM tag t1 IGNORE INDEX (PRIMARY)
    INNER JOIN tag t2 IGNORE INDEX (fk_Recipe_Tag)
    ON t1.recipe_id = t2.recipe_id
    INNER JOIN episode_recipe_cook ep_rc IGNORE INDEX (fk_Recipe_Episode)
    ON t1.recipe_id = ep_rc.recipe_id
    WHERE t1.tag_id < t2.tag_id
)
SELECT COUNT(recipe_id), category1, category2
FROM pairs
GROUP BY category1, category2
ORDER BY COUNT(recipe_id) DESC
LIMIT 3;



/*3.7*/
SELECT c.cook_id, concat(c.first_name, " ", c.last_name) cook_name, COUNT(x.cook_id) AS
participations FROM
cook c INNER JOIN episode_recipe_cook x
ON c.cook_id = x.cook_id
GROUP BY cook_id
HAVING ((SELECT COUNT(cook_id) FROM episode_recipe_cook GROUP BY(cook_id) ORDER BY
COUNT(cook_id) DESC LIMIT 1) - participations ) >= 5
ORDER BY participations DESC;

/*3.8 taking into account equal scores*/
WITH amountTool AS (
    SELECT COUNT(rt.recipe_id) as amount, rt.recipe_id FROM recipe_tool rt GROUP BY recipe_id
),
ToolsPerEpisode AS (
    SELECT SUM(att.amount) as total, e.episode, erc.episode_id, e.year_filmed FROM amountTool
att
```

```
    INNER JOIN episode_recipe_cook erc ON erc.recipe_id = att.recipe_id
    INNER JOIN episode e ON e.episode_id = erc.episode_id
    GROUP BY erc.episode_id
),
MaxTools AS (
    SELECT tpe.episode, tpe.year_filmed, tpe.total, MAX(tpe.total) as app
    FROM ToolsPerEpisode tpe
)
SELECT tpe.episode, tpe.year_filmed, tpe.total
FROM MaxTools mt, ToolsPerEpisode tpe WHERE tpe.total = mt.app;

-- Alternative query plan with force index
WITH amountTool AS (
    SELECT COUNT(rt.recipe_id) as amount, rt.recipe_id
    FROM recipe_tool rt
    FORCE INDEX (PRIMARY)
    GROUP BY rt.recipe_id
),
ToolsPerEpisode AS (
    SELECT SUM(att.amount) as total, e.episode, erc.episode_id, e.year_filmed
    FROM amountTool att
    INNER JOIN episode_recipe_cook erc FORCE INDEX (PRIMARY)
    ON erc.recipe_id = att.recipe_id
    INNER JOIN episode e FORCE INDEX (PRIMARY)
    ON e.episode_id = erc.episode_id
    GROUP BY erc.episode_id
),
MaxTools AS (
    SELECT tpe.episode, tpe.year_filmed, tpe.total, MAX(tpe.total) as app
    FROM ToolsPerEpisode tpe
)
SELECT tpe.episode, tpe.year_filmed, tpe.total
FROM MaxTools mt, ToolsPerEpisode tpe
WHERE tpe.total = mt.app;


/*3.9*/
SELECT AVG(n.carbs_per_servning) AS "average gramms of carbs", e.year_filmed FROM
episode e INNER JOIN episode_recipe_cook x
ON e.episode_id = x.episode_id
INNER JOIN nutrition n
ON n.recipe_id = x.recipe_id
GROUP BY e.year_filmed;

/*3.10*/
WITH cuisineIncluded AS (
    SELECT COUNT(c.cuisine) as entries, c.cuisine, e.year_filmed, c.cuisine_id
    FROM cuisine c INNER JOIN episode_recipe_cook erc ON erc.cuisine_id = c.cuisine_id
```

```
    INNER JOIN episode e ON e.episode_id = erc.episode_id
    GROUP BY c.cuisine_id, e.year_filmed
    HAVING entries >= 3
),
consecutive_years AS (
    SELECT
        ci1.cuisine_id,
        ci1.cuisine,
        ci1.year_filmed AS year1,
        ci1.entries AS entries1,
        ci2.year_filmed AS year2,
        ci2.entries AS entries2
    FROM
        cuisineIncluded ci1
    INNER JOIN
        cuisineIncluded ci2
        ON ci1.cuisine_id = ci2.cuisine_id
        AND ci2.year_filmed = ci1.year_filmed + 1
)
SELECT cy1.cuisine, cy2.cuisine, (cy1.entries1 + cy1.entries2) as entries, cy1.year1,
cy1.year2
FROM consecutive_years cy1, consecutive_years cy2 WHERE
cy1.year1 = cy2.year1 AND cy1.year2 = cy2.year2 AND cy1.entries1 + cy1.entries2 =
cy2.entries1 + cy2.entries2 AND cy1.cuisine_id < cy2.cuisine_id;

/*3.11*/
SELECT
    j.cook_id AS judge_id,
    concat(judge_cook.first_name, " ", judge_cook.last_name) as judge,
    erc.cook_id AS cook_id,
    concat(cook.first_name, cook.last_name) as cook,
    SUM(CASE
        WHEN j.position = '1' THEN erc.grade1
        WHEN j.position = '2' THEN erc.grade2
        WHEN j.position = '3' THEN erc.grade3
    END) AS total_score
FROM
    judge j
JOIN
    episode_recipe_cook erc ON j.episode_id = erc.episode_id
JOIN
    cook judge_cook ON j.cook_id = judge_cook.cook_id
JOIN
    cook cook ON erc.cook_id = cook.cook_id
GROUP BY
    j.cook_id, erc.cook_id
ORDER BY
    total_score DESC
```

```sql
LIMIT 5;

/*3.12*/
WITH episode_difficulity as (
    SELECT SUM(r.difficulity) as level, erc.episode_id as episode,  e.year_filmed as year
FROM recipe r
    INNER JOIN episode_recipe_cook erc ON r.recipe_id = erc.recipe_id
    INNER JOIN episode e ON e.episode_id = erc.episode_id
    GROUP BY erc.episode_id
    ORDER BY SUM(r.difficulity) DESC
)
SELECT ed.episode, ed.year, MAX(ed.level) as relative_level_indicator  FROM
episode_difficulity ed
GROUP BY ed.year
ORDER BY ed.year;

/*3.13*/
WITH cook_experience AS (
    SELECT
        cook_id,
        CASE class
            WHEN 'chef' THEN 5
            WHEN 'chef assistant' THEN 4
            WHEN 'A' THEN 3
            WHEN 'B' THEN 2
            WHEN 'C' THEN 1
            ELSE 0
        END AS experience
    FROM cook
),
episode_cook_experience AS (
    SELECT
        erc.episode_id,
        ep.episode,
        ep.year_filmed,
        SUM(ce.experience) AS total_cook_experience
    FROM
        episode_recipe_cook erc
    JOIN
        cook_experience ce ON erc.cook_id = ce.cook_id
    JOIN
        episode ep ON ep.episode_id = erc.episode_id
    GROUP BY
        erc.episode_id
),
episode_judge_experience AS (
    SELECT
        j.episode_id,
```

```
        ep.episode,
        ep.year_filmed,
        SUM(ce.experience) AS total_judge_experience
    FROM
        judge j
    JOIN
        cook_experience ce ON j.cook_id = ce.cook_id
    JOIN
        episode ep ON ep.episode_id = j.episode_id
    GROUP BY
        j.episode_id
),
episode_total_experience AS (
    SELECT
        ece.episode_id,
        ece.episode,
        ece.year_filmed,
        (ece.total_cook_experience + eje.total_judge_experience) AS total_experience
    FROM
        episode_cook_experience ece
    JOIN
        episode_judge_experience eje ON ece.episode_id = eje.episode_id
)
SELECT
    ete.episode,
    ete.year_filmed,
    ete.total_experience as relative_experience_indicator
FROM
    episode_total_experience ete
ORDER BY
    ete.total_experience ASC
LIMIT 1;

/*3.14*/
SELECT t.theme, COUNT(t.theme) as appearances FROM theme t
INNER JOIN recipe r ON r.theme_id = t.theme_id
INNER JOIN episode_recipe_cook erc ON erc.recipe_id = r.recipe_id
GROUP BY t.theme_id
ORDER BY COUNT(t.theme) DESC
LIMIT 1;

/*3.15*/
SELECT f.foodGroup unused_food_groups
FROM foodGroup f
WHERE f.foodGroup_id NOT IN (
    SELECT f.foodGroup_id
    FROM foodGroup f
    INNER JOIN ingredient i
```

```
    ON f.foodGroup_id = i.foodGroup_id
    INNER JOIN recipe_ingredient ri
    ON i.ingredient_id = ri.ingredient_id
    INNER JOIN episode_recipe_cook ep_rc
    ON ri.recipe_id = ep_rc.recipe_id
);
```

Εναλλακτικά Query Plans

Query 3.6

Κάνουμε explain το αρχικό query 3.6 ώστε να πάρουμε τα traces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t1	ALL	PRIMARY,fk_Recipe_Tag	NULL	NULL	NULL	74	Using temporary; Using filesort
1	SIMPLE	t2	ref	PRIMARY,fk_Recipe_Tag	fk_Recipe_Tag	4	master_chef.t1.recipe_id	1	Using index condition
1	SIMPLE	ep_rc	ref	fk_Recipe_Episode	fk_Recipe_Episode	4	master_chef.t1.recipe_id	10	Using index

Στη συνέχεια αλλάζουμε το query όπως φαίνεται παρακάτω:

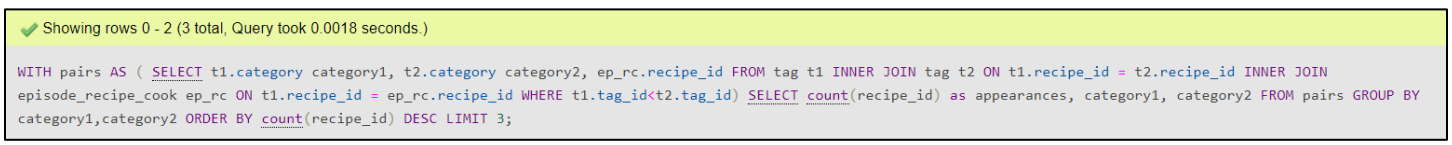
```
-- Alternative query plan with ignore index (execute with explain to see the traces)
WITH pairs AS (
    SELECT t1.category AS category1, t2.category AS category2, ep_rc.recipe_id
    FROM tag t1 IGNORE INDEX (PRIMARY)
    INNER JOIN tag t2 IGNORE INDEX (fk_Recipe_Tag)
    ON t1.recipe_id = t2.recipe_id
    INNER JOIN episode_recipe_cook ep_rc IGNORE INDEX (fk_Recipe_Episode)
    ON t1.recipe_id = ep_rc.recipe_id
    WHERE t1.tag_id < t2.tag_id
)
SELECT COUNT(recipe_id), category1, category2
FROM pairs
GROUP BY category1, category2
ORDER BY COUNT(recipe_id) DESC
LIMIT 3;
```

Κάνουμε explain και λαμβάνουμε τα παρακάτω αποτελέσματα:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t1	ALL	NULL	NULL	NULL	NULL	74	Using temporary; Using filesort
1	SIMPLE	t2	ALL	PRIMARY	NULL	NULL	NULL	74	Range checked for each record (index map: 0x1)
1	SIMPLE	ep_rc	index	NULL	fk_Cook_Episode	4	NULL	1500	Using where; Using index; Using join buffer (flat...

Παρατηρούμε ότι όταν αναγκάζουμε το query optimizer να αγνοήσει τα indexes τότε αναγκάζεται να κάνει table scan και βλέπουμε από τα traces του δεύτερου query ότι επεξεργάζεται πολλά περισσότερα rows.

Εκτελώντας τα 2 διαφορετικά queries παρατηρούμε τα execution times:



Showing rows 0 - 2 (3 total, Query took 0.0063 seconds.)

```
WITH pairs AS ( SELECT t1.category AS category1, t2.category AS category2, ep_rc.recipe_id FROM tag t1 IGNORE INDEX (PRIMARY, fk_Recipe_Tag) INNER JOIN tag t2 IGNORE INDEX (fk_Recipe_Tag) ON t1.recipe_id = t2.recipe_id INNER JOIN episode_recipe_cook ep_rc IGNORE INDEX (fk_Recipe_Episode) ON t1.recipe_id = ep_rc.recipe_id WHERE t1.tag_id < t2.tag_id ) SELECT COUNT(recipe_id), category1, category2 FROM pairs GROUP BY category1, category2 ORDER BY COUNT(recipe_id) DESC LIMIT 3;
```

Όπως ήταν αναμενόμενο αγνοώντας τα indexes η ταχύτητα του query μας είναι πολύ χαμηλότερη.

Query 3.8

Κάνουμε explain το αρχικό query 3.8 ώστε να πάρουμε τα traces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived4>	ALL	NULL	NULL	NULL	NULL	1870	Using where
1	PRIMARY	<derived5>	ref	key0	key0	20	mt.app	10	
5	DERIVED	<derived6>	ALL	NULL	NULL	NULL	NULL	187	Using temporary; Using filesort
5	DERIVED	erc	ref	PRIMARY,fk_Recipe_Episode	fk_Recipe_Episode	4	att.recipe_id	10	Using index
5	DERIVED	e	eq_ref	PRIMARY	PRIMARY	4	master_chef.erc.episode_id	1	
6	DERIVED	rt	index	PRIMARY	PRIMARY	8	NULL	187	Using index; Using temporary; Using filesort
4	DERIVED	<derived3>	ALL	NULL	NULL	NULL	NULL	1870	
3	DERIVED	<derived2>	ALL	NULL	NULL	NULL	NULL	187	Using temporary; Using filesort
3	DERIVED	erc	ref	PRIMARY,fk_Recipe_Episode	fk_Recipe_Episode	4	att.recipe_id	10	Using index
3	DERIVED	e	eq_ref	PRIMARY	PRIMARY	4	master_chef.erc.episode_id	1	
2	DERIVED	rt	index	PRIMARY	PRIMARY	8	NULL	187	Using index; Using temporary; Using filesort

Στη συνέχεια αλλάζουμε το query όπως φαίνεται παρακάτω:

```
-- Alternative query plan with force index
WITH amountTool AS (
    SELECT COUNT(rt.recipe_id) as amount, rt.recipe_id
    FROM recipe_tool rt
    FORCE INDEX (PRIMARY)
    GROUP BY rt.recipe_id
),
ToolsPerEpisode AS (
    SELECT SUM(att.amount) as total, e.episode, erc.episode_id, e.year_filmed
    FROM amountTool att
    INNER JOIN episode_recipe_cook erc FORCE INDEX (PRIMARY)
    ON erc.recipe_id = att.recipe_id
    INNER JOIN episode e FORCE INDEX (PRIMARY)
    ON e.episode_id = erc.episode_id
    GROUP BY erc.episode_id
),
MaxTools AS (
    SELECT tpe.episode, tpe.year_filmed, tpe.total, MAX(tpe.total) as app
    FROM ToolsPerEpisode tpe
)
SELECT tpe.episode, tpe.year_filmed, tpe.total
FROM MaxTools mt, ToolsPerEpisode tpe
WHERE tpe.total = mt.app;
```

Κάνουμε explain και λαμβάνουμε τα παρακάτω αποτελέσματα:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived4>	ALL	NULL	NULL	NULL	NULL	7791	Using where
1	PRIMARY	<derived5>	ref	key0	key0	20	mt.app	10	
5	DERIVED	e	index	PRIMARY	PRIMARY	4	NULL	150	
5	DERIVED	erc	ref	PRIMARY	PRIMARY	4	master_chef.e.episode_id	5	Using index
5	DERIVED	<derived6>	ref	key0	key0	4	master_chef.erc.recipe_id	10	
6	DERIVED	rt	index	PRIMARY	PRIMARY	8	NULL	187	Using index; Using temporary; Using filesort
4	DERIVED	<derived3>	ALL	NULL	NULL	NULL	NULL	7791	
3	DERIVED	e	index	PRIMARY	PRIMARY	4	NULL	150	
3	DERIVED	erc	ref	PRIMARY	PRIMARY	4	master_chef.e.episode_id	5	Using index
3	DERIVED	<derived2>	ref	key0	key0	4	master_chef.erc.recipe_id	10	
2	DERIVED	rt	index	PRIMARY	PRIMARY	8	NULL	187	Using index; Using temporary; Using filesort

Τρέχουμε τα δύο queries και παρατηρούμε τα execution times:

Showing rows 0 - 24 (52 total, Query took 0.0108 seconds.)

```
WITH amountTool AS ( SELECT COUNT(rt.recipe_id) as amount, rt.recipe_id FROM recipe_tool rt GROUP BY recipe_id ), ToolsPerEpisode AS ( SELECT SUM(att.amount) as total,
e.episode, erc.episode_id, e.year_filmed FROM amountTool att INNER JOIN episode_recipe_cook erc ON erc.recipe_id = att.recipe_id INNER JOIN episode e ON e.episode_id =
erc.episode_id GROUP BY erc.episode_id ), MaxTools AS ( SELECT tpe.episode, tpe.year_filmed, tpe.total, MAX(tpe.total) as app FROM ToolsPerEpisode tpe ) SELECT
tpe.episode, tpe.year_filmed, tpe.total FROM MaxTools mt, ToolsPerEpisode tpe WHERE tpe.total = mt.app;
```

Showing rows 0 - 24 (52 total, Query took 0.0052 seconds.)

```
WITH amountTool AS ( SELECT COUNT(rt.recipe_id) as amount, rt.recipe_id FROM recipe_tool rt FORCE INDEX (PRIMARY) GROUP BY rt.recipe_id ), ToolsPerEpisode AS ( SELECT
SUM(att.amount) as total, e.episode, erc.episode_id, e.year_filmed FROM amountTool att INNER JOIN episode_recipe_cook erc FORCE INDEX (PRIMARY) ON erc.recipe_id =
att.recipe_id INNER JOIN episode e FORCE INDEX (PRIMARY) ON e.episode_id = erc.episode_id GROUP BY erc.episode_id ), MaxTools AS ( SELECT tpe.episode, tpe.year_filmed,
tpe.total, MAX(tpe.total) as app FROM ToolsPerEpisode tpe ) SELECT tpe.episode, tpe.year_filmed, tpe.total FROM MaxTools mt, ToolsPerEpisode tpe WHERE tpe.total = mt.app;
```

Παρατηρούμε ότι το εναλλακτικό query plan (με Force Index) εκτελείται σε υποδιπλάσιο χρόνο από το αρχικό μας query.

Το προηγούμενο ήταν αναμενόμενο λαμβάνοντας υπόψιν τα traces που δημιουργήσαμε παραπάνω. Συγκεκριμένα, στο δεύτερο query περισσότερα derived χρησημοποιούν join type ref (index access using non unique keys) σε σχέση με το πρώτο όπου είναι πιο συχνά τα join type ALL (full table scan). Ωστόσο, παρόλο που το πρώτο query συνολικά επεξεργάζεται λιγότερα rows, ο κύριος λόγος που το εναλλακτικό query plan είναι πιο αποδοτικό είναι η επιλογή των indexes που θα χρησημοποιηθούν. Στη δεύτερη περίπτωση αναγκάζουμε το query optimizer να χρησιμοποιεί πάντα (στα βασικά tables και όχι τα derived) το primary index και όχι για παράδειγμα το fk_recipe_episode. Αυτό έχει αποτέλεσμα αυξημένη αποδοτικότητα, αφού το primary index είναι unique και clustered σε σχέση με το foreign key index το οποίο θα έχει duplicates και δεν είναι άμεσο.Έτσι έχοντας αρκετά joins στο συγκεκριμένο query γίνεται εμφανής ο λόγος που το δεύτερο είναι πιο γρήγορο.

DDL Script

```
CREATE DATABASE IF NOT EXISTS master_chef;

USE master_chef;

CREATE TABLE IF NOT EXISTS theme (
    theme_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    theme VARCHAR(45) NOT NULL UNIQUE,
    descr VARCHAR(100) NOT NULL,
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS cuisine (
    cuisine_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    cuisine VARCHAR(45) NOT NULL UNIQUE,
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS recipe (
    recipe_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    recipe_name VARCHAR(45) NOT NULL,
    descr TEXT NOT NULL,
    cuisine_id INT UNSIGNED NOT NULL,
    theme_id INT UNSIGNED NOT NULL,
    difficulity INT UNSIGNED NOT NULL CHECK (difficulity BETWEEN 1 AND 5),
    quantity INT UNSIGNED NOT NULL CHECK(quantity > 0),
    category VARCHAR(45) NOT NULL,                    -- vegan if the main ingredient is
vegetables, or see food if its fish and etc..
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL,
    CONSTRAINT fk_Recipe_Theme FOREIGN KEY(theme_id)
        REFERENCES theme (theme_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Recipe_Cuisine FOREIGN KEY(cuisine_id)
        REFERENCES cuisine (cuisine_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS mealType (                     -- mealType like lunch,
breakfast etc..
    mealType_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    mealType VARCHAR(45) NOT NULL UNIQUE,
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL)
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS mealType_recipe (              -- a recipe can belong to many
mealTypes and a Mealtype has a lot of recipes
    mealType_id INT UNSIGNED NOT NULL,
    recipe_id INT UNSIGNED NOT NULL,
    PRIMARY KEY(mealType_id,recipe_id),
    CONSTRAINT fk_Meal_Recipe FOREIGN KEY(mealType_id)
        REFERENCES mealType (mealType_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Recipe_Meal FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS tag (
    tag_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    category VARCHAR(45) NOT NULL,                  -- like brunch, quick lunch etc.
    tip1 VARCHAR(100),                              -- if a recipe has more than 3 tips then
more tags could be made
    tip2 VARCHAR(100),
    tip3 VARCHAR(100),
    recipe_id INT UNSIGNED NOT NULL,
    CONSTRAINT fk_Recipe_Tag FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS tool (
    tool_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    tool_name VARCHAR(45) NOT NULL,
    manual TEXT NOT NULL,                           -- usage instruction
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS step (
    step_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    step_descr VARCHAR(100) NOT NULL,
    step_order INT UNSIGNED NOT NULL CHECK(step_order > 0))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS recipe_tool (
    recipe_id INT UNSIGNED NOT NULL,
    tool_id INT UNSIGNED NOT NULL,
    PRIMARY KEY(recipe_id,tool_id),
    CONSTRAINT fk_Recipe_Tool FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Tool_Recipe FOREIGN KEY(tool_id)
        REFERENCES tool (tool_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS recipe_step (
    recipe_id INT UNSIGNED NOT NULL,
    step_id INT UNSIGNED NOT NULL,
    PRIMARY KEY (recipe_id , step_id),
    CONSTRAINT fk_Recipe_Step FOREIGN KEY (recipe_id)
        REFERENCES recipe (recipe_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Step_Recipe FOREIGN KEY (step_id)
        REFERENCES step (step_id)
        ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS foodGroup (
    foodGroup_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,    -- consult wikipidea, i think
there might be 12 different food groups
    foodGroup VARCHAR(45),
    descr TEXT,
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS ingredient (
    ingredient_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    ingredient VARCHAR(45),
    calories INT UNSIGNED NOT NULL,                        -- per 100g for foods that get
weighet in gr, per 100ml for liquid, pepper,salt have 0 , per quantity for things like eggs
etc.
    foodGroup_id INT UNSIGNED NOT NULL,
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL,
    CONSTRAINT fk_Ingredient_FoodGroup FOREIGN KEY(foodGroup_id)
        REFERENCES foodGroup (foodGroup_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;



CREATE TABLE IF NOT EXISTS recipe_ingredient (
    recipe_id INT UNSIGNED NOT NULL,
    ingredient_id INT UNSIGNED NOT NULL,
    amount INT UNSIGNED NOT NULL,                          -- per 100gr/100ml/quantity
    main_ingredient BOOLEAN NOT NULL DEFAULT FALSE,     -- based on that ingredient we set
the category attribute in the recipe table
    PRIMARY KEY(recipe_id, ingredient_id),
    CONSTRAINT fk_Recipe_Ingredient FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Ingredient_Recipe FOREIGN KEY(ingredient_id)
        REFERENCES ingredient (ingredient_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS nutrition (
    recipe_id INT UNSIGNED NOT NULL,
    calories_per_serving INT UNSIGNED NOT NULL CHECK(calories_per_serving > 0),    -- this
one shouldnt be included in the mock data but instead calculated dynamically by the sum of
all ingredients multiplied byy their calories of the recipe
    carbs_per_servning DECIMAL(5,2) NOT NULL,     -- THERE IS A TYPO, CARBS PER
****SERVNING****
    fat_per_serving DECIMAL(5,2) NOT NULL,
    protein_per_serving DECIMAL(5,2) NOT NULL,
    CONSTRAINT fk_Recipe_Nutrition FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;



CREATE TABLE IF NOT EXISTS cook (
    cook_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(45) NOT NULL,
    last_name VARCHAR(45) NOT NULL,
    birth DATE NOT NULL,
    age INT UNSIGNED NOT NULL CHECK(age >= 18),
    phone VARCHAR(45) NOT NULL,
    years_of_experience INT UNSIGNED NOT NULL,
    class ENUM('C','B','A','chef assistant','chef'),
    image VARCHAR(100) NOT NULL,
    caption VARCHAR(60) NOT NULL,
    CHECK(class in ('C','B','A','chef assistant','chef')))
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS cuisine_cook (
    cuisine_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    PRIMARY KEY(cuisine_id,cook_id),
    CONSTRAINT fk_Cuisine_Cook FOREIGN KEY(cuisine_id)
        REFERENCES cuisine (cuisine_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Cook_Cuisine FOREIGN KEY(cook_id)
        REFERENCES cook (cook_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS episode (
    episode_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    episode INT UNSIGNED NOT NULL,
    year_filmed INT UNSIGNED NOT NULL)
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS episode_recipe_cook (
    episode_id INT UNSIGNED NOT NULL,
    recipe_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    cuisine_id INT UNSIGNED NOT NULL,
    grade1 INT UNSIGNED NOT NULL CHECK(grade1 >= 1 and grade1 <=5),
    grade2 INT UNSIGNED NOT NULL CHECK(grade2 >= 1 and grade2 <=5),
    grade3 INT UNSIGNED NOT NULL CHECK(grade3 >= 1 and grade3 <=5),
    winner BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY(episode_id, recipe_id, cook_id),
    CONSTRAINT fk_Recipe_Episode FOREIGN KEY(recipe_id)
        REFERENCES recipe (recipe_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Cook_Episode FOREIGN KEY(cook_id)
        REFERENCES cook (cook_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Cuisine_Episode FOREIGN KEY(cuisine_id)
        REFERENCES cuisine (cuisine_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Episode_Recipe_Cook FOREIGN KEY(episode_id)
        REFERENCES episode (episode_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS judge (
    episode_id INT UNSIGNED NOT NULL,
    cook_id INT UNSIGNED NOT NULL,
    position ENUM('1', '2', '3') NOT NULL,
    PRIMARY KEY(episode_id,cook_id),
    CHECK(position IN ('1', '2', '3')),
    CONSTRAINT fk_Judge_Episode FOREIGN KEY(episode_id)
        REFERENCES episode (episode_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Judge_Cook FOREIGN KEY(cook_id)
        REFERENCES cook (cook_id) ON DELETE RESTRICT ON UPDATE CASCADE)
ENGINE = InnoDB;



CREATE INDEX idx_year_filmed ON episode (year_filmed);  -- used with GROUP and JOIN and ORDER
BY in some queries

CREATE INDEX idx_age_cook ON cook (age);                -- used in filtering with WHERE

-- no need for more indexes as most of our queries utilize primary and unique indexes that
got created automatically from InnoDB
```

```
DELIMITER //
CREATE TRIGGER atmost_one_mainIngredient
BEFORE INSERT ON recipe_ingredient
FOR EACH ROW
BEGIN
    DECLARE counter_mainIngredient INT;
    IF NEW.main_ingredient = TRUE THEN
        SELECT COUNT(*) INTO counter_mainIngredient
        FROM recipe_ingredient
        WHERE recipe_id = NEW.recipe_id AND main_ingredient = TRUE;
        IF counter_mainIngredient > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'A recipe can have at most one main ingredient';
        END IF;
    END IF;
END //
DELIMITER ;

-- update instead of insert
DELIMITER //
CREATE TRIGGER atmost_one_mainIngredient1
BEFORE UPDATE ON recipe_ingredient
FOR EACH ROW
BEGIN
    DECLARE counter_mainIngredient INT;
    IF NEW.main_ingredient = TRUE THEN
        SELECT COUNT(*) INTO counter_mainIngredient
        FROM recipe_ingredient
        WHERE recipe_id = NEW.recipe_id AND main_ingredient = TRUE AND ingredient_id !=
NEW.ingredient_id;
        IF counter_mainIngredient > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'A recipe can have at most one main ingredient';
        END IF;
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER famous_judge
BEFORE INSERT  ON judge
FOR EACH ROW
BEGIN
    DECLARE nbrOfParticipation INT;

    IF (NEW.episode_id % 10) >= 4 THEN
        SELECT COUNT(*) INTO nbrOfParticipation
        FROM judge
```

```
        WHERE episode_id BETWEEN  NEW.episode_id - 3 AND NEW.episode_id
        AND cook_id = NEW.cook_id;
    END IF;

    IF nbrOfParticipation >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A Judge cannot be present in more than 3 consecutive episodes';
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER cook_or_judge
BEFORE INSERT  ON judge
FOR EACH ROW
BEGIN
    DECLARE nbrOfParticipation INT;

    SELECT COUNT(*) INTO nbrofParticipation
    FROM episode_recipe_cook
    WHERE episode_id = NEW.episode_id  AND cook_id = NEW.cook_id;

    IF nbrOfParticipation > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A Judge cannot be cooking as well';
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER cook_cuisine_ability
BEFORE INSERT  ON episode_recipe_cook
FOR EACH ROW
BEGIN
    DECLARE canCook INT;

    SELECT COUNT(*) INTO canCook
    FROM cuisine_cook
    WHERE cuisine_id = NEW.cuisine_id
    AND cook_id = NEW.cook_id;

    IF canCook < 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A cook can only cook cuisine in his expertice';
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER cook_overuse
BEFORE INSERT ON episode_recipe_cook
FOR EACH ROW
BEGIN
    DECLARE nbrOfParticipation INT;

    IF (NEW.episode_id % 10) >= 4 THEN
        SELECT COUNT(*) INTO nbrOfParticipation
        FROM episode_recipe_cook
        WHERE episode_id BETWEEN NEW.episode_id - 3 AND NEW.episode_id
        AND cook_id = NEW.cook_id;
    END IF;

    IF nbrOfParticipation >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A Cook cannot be present in more than  3 consecutive episodes';
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER recipe_overuse
BEFORE INSERT ON episode_recipe_cook
FOR EACH ROW
BEGIN
    DECLARE nbrOfParticipation INT;

    IF (NEW.episode_id % 10) >= 4 THEN
        SELECT COUNT(*) INTO nbrOfParticipation
        FROM episode_recipe_cook
        WHERE episode_id BETWEEN NEW.episode_id - 3 AND NEW.episode_id
        AND recipe_id = NEW.recipe_id;
    END IF;

    IF nbrOfParticipation >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A Recipe cannot be present in more than 3 consecutive episodes';
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER cuisine_overuse
BEFORE INSERT ON episode_recipe_cook
FOR EACH ROW
BEGIN
    DECLARE nbrOfParticipation INT;
```

```
    IF (NEW.episode_id % 10) >= 4 THEN
        SELECT COUNT(*) INTO nbrOfParticipation
        FROM episode_recipe_cook
        WHERE episode_id BETWEEN NEW.episode_id - 3 AND NEW.episode_id
        AND cuisine_id = NEW.cuisine_id;
    END IF;

    IF nbrOfParticipation >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A Cuisine cannot be present in more than 3 consecutive episodes';
    END IF;
END //
DELIMITER ;


-- THE BEFORE UPDATE ON for the above triggers isnt required since it doesnt make sense for
anyone to go and change what has arleady happened
-- besides it has almost the same logic as the above triggers so creating them wouldnt be an
issue
-- but they would negatively impact our database efficiency so they will be ignored
-- **********************************
-- **********************************


DELIMITER //

CREATE PROCEDURE CalculateCaloriesPerServing()
BEGIN
    DECLARE recipeId INT;
    DECLARE totalCalories INT;

    -- Declare cursor to iterate over each recipe
    DECLARE recipeCursor CURSOR FOR
        SELECT DISTINCT recipe_id FROM recipe_ingredient;

    -- Declare NOT FOUND handler for the cursor, will be used to know when to terminate the
loop
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET recipeId = NULL;

    -- Open the cursor
    OPEN recipeCursor;

    -- Initialize totalCalories
    SET totalCalories = 0;

    -- Loop through each recipe
```

```
    recipeLoop: LOOP
        FETCH recipeCursor INTO recipeId;
        IF recipeId IS NULL THEN
            LEAVE recipeLoop;
        END IF;

        -- Calculate total calories for the recipe
        SET totalCalories = (
            SELECT SUM((ri.amount * i.calories / 100) / r.quantity)
            FROM recipe_ingredient ri
            JOIN ingredient i ON ri.ingredient_id = i.ingredient_id
            JOIN recipe r ON r.recipe_id = ri.recipe_id
            WHERE ri.recipe_id = recipeId
        );

        -- Update the calories_per_serving in the nutrition table
        UPDATE nutrition
        SET calories_per_serving = totalCalories
        WHERE recipe_id = recipeId;
    END LOOP;

    -- Close the cursor
    CLOSE recipeCursor;

END //

DELIMITER ;


/* Now it would be way more efficient to have an argument in the previous stored procedure
that specifies the recipe_id in nutrition table
and creat triggers that call the procedure after an insert/update. like below */

DELIMITER //

CREATE PROCEDURE CalculateCaloriesPerServingForRecipe(IN recipeId INT)
BEGIN
    DECLARE totalCalories DECIMAL(10,2);

    -- Calculate total calories for the recipe
    SET totalCalories = (
        SELECT SUM((ri.amount * i.calories / 100) / r.quantity)
        FROM recipe_ingredient ri
        JOIN ingredient i ON ri.ingredient_id = i.ingredient_id
        JOIN recipe r ON r.recipe_id = ri.recipe_id
        WHERE ri.recipe_id = recipeId
    );
```

```
    -- Update the calories_per_serving in the nutrition table
    UPDATE nutrition
    SET calories_per_serving = totalCalories
    WHERE recipe_id = recipeId;
END //
DELIMITER ;
DELIMITER //
-- Trigger after inserting a new ingredient for a recipe **
CREATE TRIGGER after_recipe_ingredient_insert
AFTER INSERT ON recipe_ingredient
FOR EACH ROW
BEGIN
    CALL CalculateCaloriesPerServingForRecipe(NEW.recipe_id);
END //

-- Trigger after updating an ingredient for a recipe  **
CREATE TRIGGER after_recipe_ingredient_update
AFTER UPDATE ON recipe_ingredient
FOR EACH ROW
BEGIN
    CALL CalculateCaloriesPerServingForRecipe(NEW.recipe_id);
END //

-- Trigger after deleting an ingredient for a recipe  **
CREATE TRIGGER after_recipe_ingredient_delete
AFTER DELETE ON recipe_ingredient
FOR EACH ROW
BEGIN
    CALL CalculateCaloriesPerServingForRecipe(OLD.recipe_id);
END //
DELIMITER ;
DELIMITER //
CREATE PROCEDURE updateEpisodeWinners()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE current_episode_id INT UNSIGNED;
    DECLARE max_grade INT UNSIGNED;
    DECLARE max_grade_cook_id INT UNSIGNED;

    -- Cursor to iterate over each episode
    DECLARE episode_cursor CURSOR FOR
        SELECT episode_id FROM episode;

    -- Handler to manage the end of the cursor loop
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open the cursor
    OPEN episode_cursor;
```

```
    -- Loop through each episode
    episode_loop: LOOP
        FETCH episode_cursor INTO current_episode_id;

        IF done THEN
            LEAVE episode_loop;
        END IF;

        -- Find the cook_id with the highest total grades in the current episode
        SELECT cook_id, (grade1 + grade2 + grade3) AS total_grade INTO max_grade_cook_id,
max_grade
        FROM episode_recipe_cook
        WHERE episode_id = current_episode_id
        ORDER BY total_grade DESC
        LIMIT 1;

        -- Update the winner column to TRUE for the cook with the highest total grade in the
current episode
        UPDATE episode_recipe_cook
        SET winner = TRUE
        WHERE episode_id = current_episode_id AND cook_id = max_grade_cook_id;
    END LOOP;

    -- Close the cursor
    CLOSE episode_cursor;
END //
DELIMITER ;
DELIMITER //
CREATE TRIGGER atmost_one_winner
BEFORE INSERT ON episode_recipe_cook
FOR EACH ROW
BEGIN
    DECLARE counter_winner INT;
    IF NEW.winner = TRUE THEN
        SELECT COUNT(*) INTO counter_winner
        FROM episode_recipe_cook
        WHERE episode_id = NEW.episode_id AND winner = TRUE AND cook_id != new.cook_id;
        IF counter_winner > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'There can only be one winner in a single episode';
        END IF;
    END IF;
END //
DELIMITER ;
```