Εξαμηνιαία εργασία του μαθήματος Βάσεις Δεδομένων

Ομαδα 123

Τσαπικούνης Γεώργιος 03119070
Χρήστος Σκάρλος 03119911
Αντιγόνη Μαρία Καρανίκα 03120159

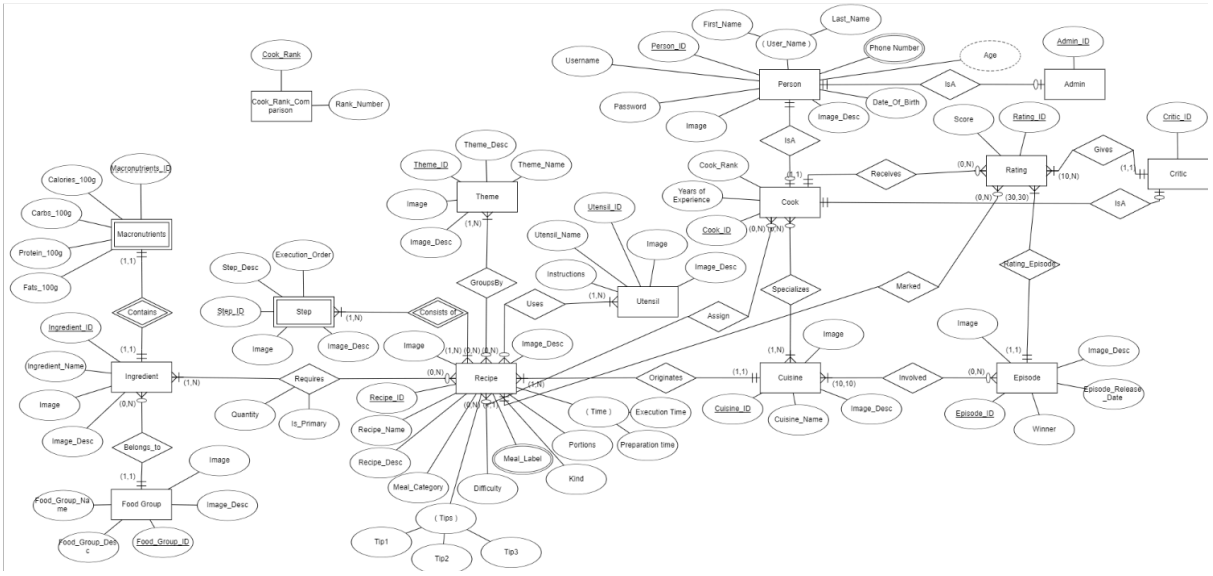
Εισαγωγή

Στην παρούσα εξαμηνιαία εργασία καλούμαστε να υλοποιήσουμε μια εφαρμογή για την αποθήκευση πληροφοριών και δεδομένων ενός διαγωνισμού μαγειρικής που αποτελείται από 10 επεισόδια ετησίως. Η εφαρμογή αυτή φροντίζει να βρίσκει τον νικητή κάθε επεισοδίου, με βάση τις βαθμολογίες που έλαβαν οι μάγειρες από τους κριτές. Η διαχείριση και αποθήκευση των δεδομένων έχει πραγματοποιηθεί μέσω της βάσης δεδομένων που αναπτύξαμε σε MariaDB.

Για τη λειτουργία της εφαρμογής έχουμε δημιουργήσει δύο ρόλους (Admin, Cook) στους οποίους έχουμε αναθέσει διαφορετικά δικαιώματα. Αρχικά, ένας χρήστης οποιαδήποτε ρόλου χρειάζεται να συνδεθεί με βάση το username και τον κωδικό του για να έχει πρόσβαση στην εφαρμογή.
Ο διαχειριστής (Admin) μπορεί να καταχωρήσει και να τροποποιήσει όλα τα στοιχεία και να δημιουργήσει αντίγραφο ασφαλείας για όλη τη βάση (backup) και να επαναφέρει το σύστημα από αυτό (restore).
Ένας χρήστης Cook, από την άλλη, είναι πιο περιορισμένος. Συγκεκριμένα, μπορεί να επεξεργαστεί μόνο τα στοιχεία του και τις συνταγές που του έχουν ανατεθεί, με τη δυνατότητα να προσθέσει ακόμα και καινούριες.

Όλα τα απαραίτητα αρχεία για την εργασία βρίσκονται στο παρακάτω σύνδεσμο:
https://github.com/hjmjko/ntua_db_project

1) ER Διάγραμμα

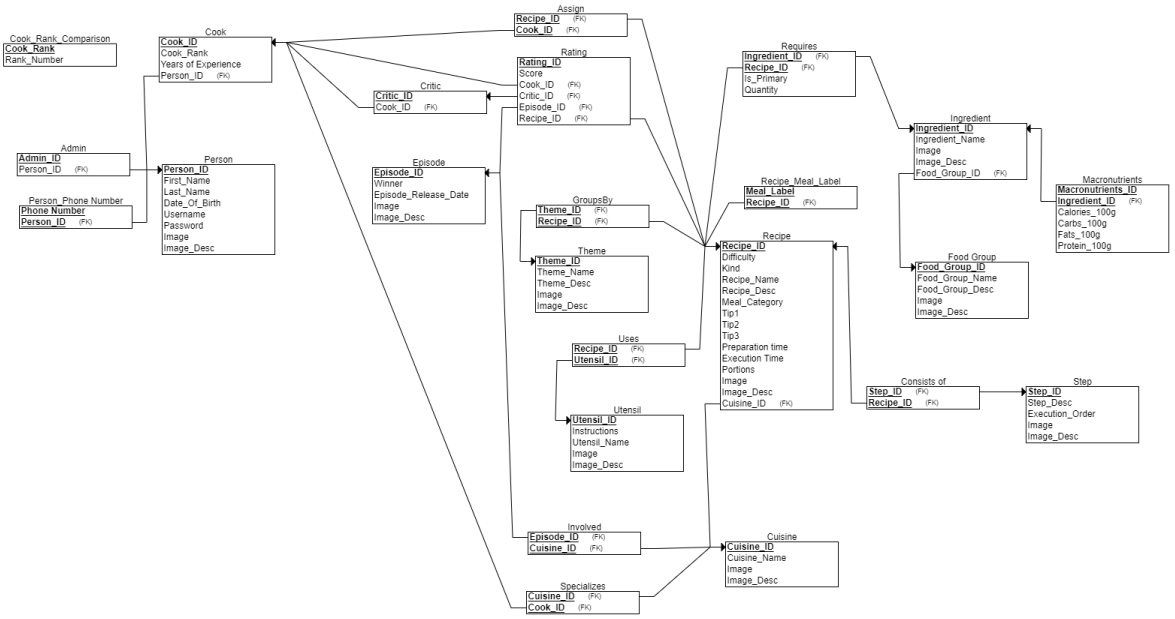


Τα cardinalities σημειώνονται δίπλα από κάθε οντότητα. Για να γίνει πιο κατανοητός ο συμβολισμός που έχουμε ακολουθήσει ας δούμε τη σχέση Gives που συνδέει την οντότητα Rating με την οντότητα Critic. Ένας κριτής μπορεί να δώσει από 10 μέχρι Ν κριτικές (εφόσον σε ένα επεισόδιο θα δώσει αναγκαστικά μία κριτική και στους 10 μάγειρες που διαγωνίζονται) και μια κριτική θα δοθεί από έναν κριτή.

Ο κύκλος συμβολίζει ότι μια οντότητα είναι προαιρετική στη μεταξύ τους σχέση και η γραμμή συμβολίζει ότι είναι υποχρεωτική. Για παράδειγμα, για τη σχέση Receives που συνδέει τις οντότητες Cook, Rating ισχύει ότι μια κριτική θα αντιστοιχεί αναγκαστικά σε κάποιο μάγειρα, ενώ ένας μάγειρας δεν είναι απαραίτητο ότι έχει λάβει κάποια κριτική, εφόσον μπορεί να μην έχει λάβει ακόμη μέρος στον διαγωνισμό.

Η οντότητα Cook_Rank_Comparison δεν συνδέεται με κάποια άλλη οντότητα, καθώς τη χρησιμοποιούμε μόνο ως look up table για να αντιστοιχούμε την ιεραρχία των μαγείρων σε ένα νούμερο.

2) Σχεσιακό Διάγραμμα



Από το ER προκύπτει το παραπάνω σχεσιακό διάγραμμα. Ένας από τους πιο σημαντικούς πίνακες που προκύπτουν είναι ο Rating, καθώς τον χρησιμοποιούμε σχεδόν σε όλα τα queries που καλούμαστε να υλοποιήσουμε. Ο πίνακας αυτός μας πληροφορεί για την κριτική που έδωσε ένας κριτής σε έναν μάγειρα, όταν μαγείρεψε μια συγκεκριμένη συνταγή σε ένα ορισμένο επεισόδιο.

Στη συνέχεια, από το σχεσιακό διάγραμμα προκύπτουν τα tables της βάσης μας, τα οποία φαίνονται στο αρχείο my_ddl.sql.

Jupyter Notebooks

Μέσα στο notebook helper_functions.ipynb υπάρχουν βοηθητικές συναρτήσεις που χρησιμοποιούνται για μέρη της εφαρμογής. Οι συγκεκριμένες συναρτήσεις ακολουθούν την λογική ότι θα καλούνταν από την εφαρμογή, σε περίπτωση που αυτή υπήρχε, και θα αποτελούσαν την διεπαφή μεταξύ εφαρμογής και βάσης.

Για παράδειγμα η συνάρτηση `fill_episode()` πραγματοποιεί την κλήρωση του διαγωνισμού και εισάγει τα δεδομένα στην βάση εξασφαλίζοντας πάντα τους περιορισμούς για μέγιστο 3 διαδοχικές συμμετοχές σε επεισόδιο για μάγειρες/κριτές/εθνικές κουζίνες/συνταγές, αλλά και αναμενόμενους περιορισμούς όπως ένας μάγειρας δεν μπορεί να είναι κριτής και μάγειρας στο ίδιο επεισόδιο. Με κατάλληλες παραμέτρους χρησιμοποιείται για την δημιουργία dummy data.

Με τις συναρτήσεις `create_admins()` και `create_cooks()` δημιουργούνται δυναμικά οι χρήστες τις βάσεις και σε επίπεδο εφαρμογής θα καλούνταν όταν θέλουμε να εισάγουμε κάποιον χρήστη στη βάση.

Roles

Η βάση διαθέτει δύο διαφορετικούς ρόλους, Admin και Cook. Κάθε χρήστης που παίρνει τον ρόλο του διαχειριστή διαθέτει εκτεταμένα privileges λόγω του `ALL PRIVILEGES`, το οποίο περιλαμβάνει όλα τα βασικά όπως `SELECT, INSERT, UPDATE`. Επιπλέον δίνονται στοχευμένα τα privileges `RELOAD, PROCESS, LOCK TABLES, BINLOG MONITOR` τα οποία απαιτούνται από το εργαλείο mariabackup για δημιουργία backup της βάσης και restore από αυτό .

Ο ρόλος Cook έχει δημιουργηθεί έτσι ώστε να μην έχει κανένα γενικό privilege και επομένως να μην μπορεί να επέμβει με κανέναν άμεσο τρόπο στη βάση. Ωστόσο του έχει δωθεί στοχευμένα EXECUTE privileges σε ορισμένές PROCEDURES οι οποίες αποτελούν το σύνολο των αποδεκτών ενεργειών του σύμφωνα με τις απαιτήσεις τις εκφώνησεις. Οι διεργασίες αυτές εκτελούνται με τα privileges του δημιουργού τους, δηλαδή του Admin. Στόχος είναι οι μάγειρες να μην μπορούν να επέμβουν στην βάση αλλά να εκτελούν μέσω της εφαρμογής και του UI αυτές τις συγκεκριμένες διεργασίες.

Οι χρήστες που ανατίθονται στους ρόλους δημιουργούνται δυναμικά από τις παραπάνω συνρτήσεις `create_admins()` και `create_cooks()`, σύμφωνα με τα dummy data στην συγκεκριμένη περίπτωση και πρακτικά μέσω της εφαρμογής αν υπήρχε.

Indexes

Για την βελτίωση της απόδοσης της βάσης μας χρησιμοποιήσαμε τα εξής:

```
-- To remove triple duplicates because of critics per cook
CREATE INDEX Unique_Ratings ON Rating (Recipe_ID, Cook_ID, Episode_ID);
-- Because Score is sometimes used as counter or average or sum
CREATE INDEX Score ON Rating (Score);
-- Year is frequently used on Episode_Release_Date and joined on
CREATE INDEX Episode_Release_Date ON Episode (Episode_Release_Date);
```

Το Unique_Ratings index είναι χρήσιμο για τη βάση μας, καθώς κατά την υλοποίηση των περισσότερων queries ήταν απαραίτητο να γίνει ένα GROUP BY Recipe_ID, Cook_ID, Episode_ID, έτσι ώστε να λάβουμε υπόψη μας τον κάθε μάγειρα που συμμετέχει σε ένα επεισόδιο μόνο μια φορά, εφόσον χωρίς αυτό θα εμφανιζόταν τρεις φορές λόγω των τριών κριτών που λαμβάνει τη βαθμολογία.

Το Score μας είναι χρήσιμο γιατί πολλές φορές το χρησιμοποιούμε ως μετρητή εμφανίσεων ή για τους υπολογισμούς των μέσων τιμών και των αθροισμάτων.

Τέλος, στα περισσότερα queries χρησιμοποιούμε το έτος που διαδραματίζεται κάθε επεισόδιο, κυρίως σε JOIN, οπότε γι αυτό ορίσαμε το index Episode_Release_Date.

Τα ερωτήματα που υποστηρίζει η βάση μας

- ***Ερώτημα 3.1/query1:***

```
SELECT Person.First_Name, Person.Last_Name, ROUND(AVG(Rating.Score), 1) AS CookAverageScore
FROM Rating
JOIN Cook ON Rating.Cook_ID = Cook.Cook_ID
JOIN Person ON Cook.Person_ID = Person.Person_ID
GROUP BY Rating.Cook_ID;

SELECT Cuisine.Cuisine_Name, ROUND(AVG(Rating.Score), 1) AS CuisineAverageScore
FROM Rating
JOIN Recipe ON Rating.Recipe_ID = Recipe.Recipe_ID
JOIN Cuisine ON Recipe.Cuisine_ID = Cuisine.Cuisine_ID
GROUP BY Cuisine.Cuisine_Name;
```

Το πρώτο ερώτημα το χωρίσαμε σε 2 query . Όπου το πρώτο μας εμφανίζει τον μέσο όρο των βαθμολογιών ανά μάγειρα. Ενώ στο δεύτερο υπολογίζουμε τον μέσο όρο των βαθμολογιών ανά Εθνική κουζίνα.

- ***Ερώτημα 3.2/query2:***

```
-- DONE!
WITH CooksPerCuisine AS (
    SELECT Person.Person_ID, Person.First_Name, Person.Last_Name, Cuisine.Cuisine_Name
    FROM Specializes
    JOIN Cuisine ON Specializes.Cuisine_ID = Cuisine.Cuisine_ID
    JOIN Cook ON Specializes.Cook_ID = Cook.Cook_ID
    JOIN Person ON Cook.Person_ID = Person.Person_ID
),
CooksPerYear AS (
    SELECT Person.Person_ID, Person.First_Name, Person.Last_Name, YEAR(Episode.Episode_Release_Date) AS EpisodeYear
    FROM Rating
    JOIN Episode ON Rating.Episode_ID = Episode.Episode_ID
    JOIN Cook ON Rating.Cook_ID = Cook.Cook_ID
    JOIN Person ON Cook.Person_ID = Person.Person_ID
    GROUP BY Rating.Cook_ID, YEAR(Episode.Episode_Release_Date)
    ORDER BY EpisodeYear ASC
)
SELECT CooksPerCuisine.Person_ID, CooksPerCuisine.First_Name, CooksPerCuisine.Last_Name, CooksPerCuisine.Cuisine_Name,
       CooksPerYear.EpisodeYear
FROM CooksPerCuisine
LEFT JOIN CooksPerYear ON CooksPerCuisine.Person_ID = CooksPerYear.Person_ID
ORDER BY CooksPerYear.EpisodeYear ASC;
```

Στο δεύτερο query χρησιμοποιούμε τα WITH () AS ώστε αρχικά να υπολογίσουμε έναν πίνακα με όλους τους μάγειρες και τις αντίστοιχες κουζίνες στις οποίες ειδικεύονται. Έπειτα υπολογίζουμε έναν πίνακα πάλι με όλους τους μάγειρες και τα επεισόδια στα οποία έχουν πάρει μέρος αυτοί. Στο τέλος, κάνουμε SELECT από τους δύο πίνακες που φτιάξαμε όλους τους μάγειρες μαζί με τα επεισόδια στα οποία συμμετείχαν και τις κουζίνες στις οποίες ειδικεύονται.

- ***Ερώτημα 3.3/query3:***

```sql
-- DONE!
WITH CookedRecipes AS (
    SELECT Person.Person_ID, Person.First_Name, Person.Last_Name, Person.Age, Rating.Cook_ID, Rating.Recipe_ID, Rating.Episode_ID
    FROM Rating
    JOIN Cook ON Rating.Cook_ID = Cook.Cook_ID
    JOIN Person ON Cook.Person_ID = Person.Person_ID
    GROUP BY Rating.Recipe_ID, Rating.Cook_ID, Rating.Episode_ID
)
SELECT CookedRecipes.Person_ID, CookedRecipes.First_Name, CookedRecipes.Last_Name, CookedRecipes.Age,
       COUNT(DISTINCT CookedRecipes.Recipe_ID) as Recipes_Per_Cook
FROM CookedRecipes
WHERE CookedRecipes.Age < 30
GROUP BY CookedRecipes.Person_ID
ORDER BY Recipes_Per_Cook DESC, Age ASC;
```

Σε αυτό το ερώτημα υπολογίζουμε πρώτα έναν πίνακα ο οποίος περιέχει όλους τους μάγειρες, τα επεισόδια που έχουν πάρει μέρος και τις συνταγές του κάθε μάγειρα. Έπειτα, από αυτόν τον πίνακα μετράμε το πόσες συνταγές έχει κάθε μάγειρας και έπειτα διαλέγουμε αυτούς οι οποίοι είναι μικρότεροι από 30 χρονών και τους τοποθετούμε στον πίνακα με φθίνουσα σειρά σύμφωνα με τον αριθμό των συνταγών που έχει ο καθένας τους, ώστε στην αρχή του πίνακα να εμφανίζονται αυτοί με τις περισσότερες συνταγές .

- ***Ερώτημα 3.4/query4:***

```sql
-- DONE!
(
    SELECT First_Name, Last_Name
    FROM Cook
    JOIN Person ON Cook.Person_ID = Person.Person_ID
)
EXCEPT
(
    SELECT First_Name, Last_Name
    FROM Critic
    JOIN Cook ON Critic.Cook_ID = Cook.Cook_ID
    JOIN Person ON Cook.Person_ID = Person.Person_ID
);
```

Εδώ υπολογίζουμε πρώτα όλους τους μάγειρες σε έναν πίνακα και μετά κάνουμε EXCEPT τον πίνακα που υπολογίζουμε από κάτω, ο οποίος υπολογίζει όλους τους μάγειρες οι οποίοι έχουν συμμετάσχει και ως κριτές σε κάποιο επεισόδιο. Αφήνοντας στο τέλος μόνο τους μάγειρες οι οποίοι δεν έχουν συμμετάσχει ποτέ ως κριτές.

- ***Ερώτημα 3.5/query5:***

```sql
-- DONE!
WITH CriticAppearancesPerYear AS (
    SELECT Critic_ID, COUNT(Critic_ID) AS AppearancesPerYear, EpisodeYear
    FROM (
        SELECT Cook_ID, Critic_ID, Episode.Episode_ID, YEAR(Episode_Release_Date) AS EpisodeYear
        FROM Rating
        JOIN Episode ON Rating.Episode_ID = Episode.Episode_ID
        GROUP BY Critic_ID, Episode_ID -- To avoid dividing by magic number 10
    ) AS EpisodeRatings
    GROUP BY Critic_ID, EpisodeYear
    HAVING AppearancesPerYear > 3
)
SELECT CriticAppearancesPerYear.Critic_ID, Person.First_Name, Person.Last_Name, CriticAppearancesPerYear.AppearancesPerYear,
 CriticAppearancesPerYear.EpisodeYear
FROM CriticAppearancesPerYear
JOIN Critic ON CriticAppearancesPerYear.Critic_ID = Critic.Critic_ID
JOIN Cook ON Critic.Cook_ID = Cook.Cook_ID
JOIN Person ON Cook.Person_ID = Person.Person_ID
ORDER BY CriticAppearancesPerYear.AppearancesPerYear ASC;
```

Στο 5ο ερώτημα αρχικά υπολογίζουμε έναν πίνακα με τις συμμετοχές κάθε κριτή ανά χρόνο διαλέγοντας, όμως, μόνο αυτούς που έχουν πάνω από τρεις εμφανίσεις.Στη συνέχεια έχοντας τον αριθμό συμμετοχών και το id του κάθε κριτή βρίσκουμε και τα ονόματά τους . Έτσι, έχουμε όλους τους κριτές που έχουν τον ίδιο αριθμό συμμετοχών ανά χρόνο με πάνω από 3 συμμετοχές.

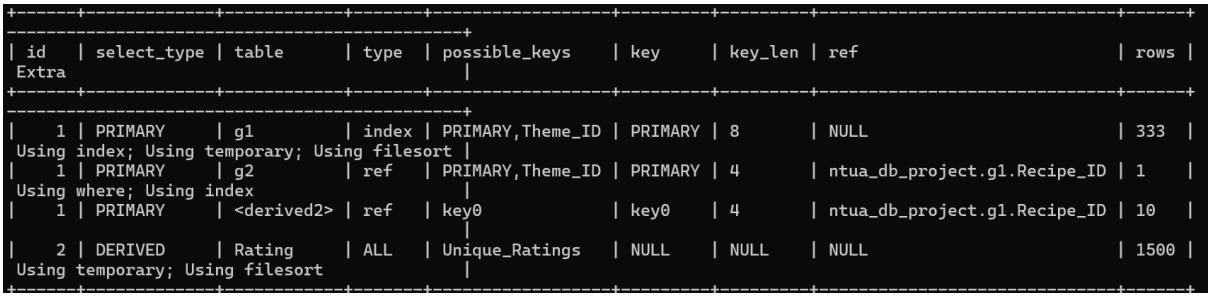
- ***Ερώτημα 3.6/query6:***

```sql
-- DONE!! Original query plan
WITH Episode_Ratings AS(
    SELECT *
    FROM Rating
    GROUP BY Recipe_ID, Cook_ID,Episode_ID
),
Theme_Pairs AS(
    SELECT g1.Recipe_ID, g1.Theme_ID AS G1Theme, g2.Theme_ID AS G2Theme
    FROM Groups_By g1 JOIN Groups_By g2 ON g1.Recipe_ID = g2.Recipe_ID
    WHERE g1.Theme_ID < g2.Theme_ID
)
SELECT COUNT(*) AS Theme_Pair_Appearances, Theme_Pairs.G1Theme, Theme_Pairs.G2Theme
FROM Episode_Ratings
JOIN Theme_Pairs ON Episode_Ratings.Recipe_ID = Theme_Pairs.Recipe_ID
GROUP BY Theme_Pairs.G1Theme, Theme_Pairs.G2Theme
ORDER BY Theme_Pair_Appearances DESC, RAND()
LIMIT 3;

-- DONE!! Alternate query plan
WITH Episode_Ratings AS(
    SELECT *
    FROM Rating
    FORCE INDEX FOR GROUP BY (Unique_Rating)
    GROUP BY Recipe_ID, Cook_ID,Episode_ID
),
Theme_Pairs AS(
    SELECT g1.Recipe_ID, g1.Theme_ID AS G1Theme, g2.Theme_ID AS G2Theme
    FROM Groups_By g1
    FORCE INDEX (Theme_ID)
    JOIN Groups_By g2 FORCE INDEX (Theme_ID) ON g1.Recipe_ID = g2.Recipe_ID
    WHERE g1.Theme_ID < g2.Theme_ID
)
SELECT COUNT(*) AS Theme_Pair_Appearances, Theme_Pairs.G1Theme, Theme_Pairs.G2Theme
FROM Episode_Ratings
JOIN Theme_Pairs ON Episode_Ratings.Recipe_ID = Theme_Pairs.Recipe_ID
GROUP BY Theme_Pairs.G1Theme, Theme_Pairs.G2Theme
ORDER BY Theme_Pair_Appearances DESC, RAND()
LIMIT 3;
```

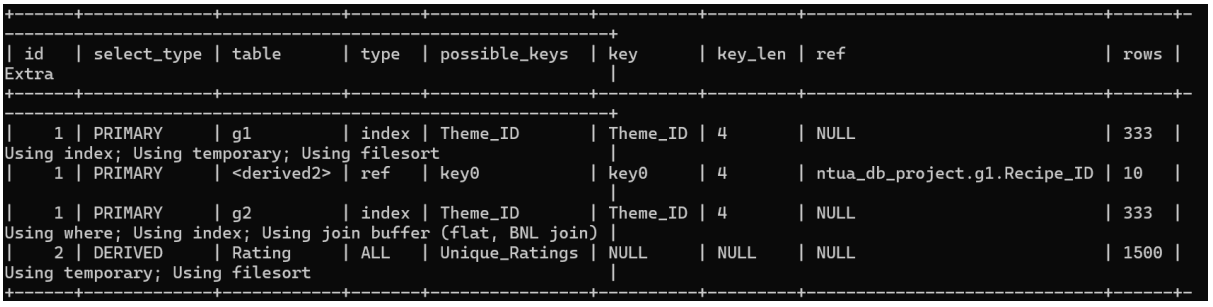
Εδώ αρχικά βρίσκουμε τις βαθμολογίες για κάθε συνταγή που έχει μαγειρέψει κάθε μάγειρας σε κάθε επεισόδιο.Στη συνέχεια, υπολογίζουμε όλα τα πιθανά ζευγάρια θεμάτων κάθε συνταγής . Έπειτα, κάνουμε COUNT, ώστε να βρούμε τον αριθμό των φορών που υπήρξαν σε επεισόδια οι συνταγές με αυτά τα θέματα . Στο τέλος, τα τοποθετούμε κατά φθίνουσα σειρά και κρατάμε τις πρώτες 3 γραμμές, ώστε να έχουμε το top 3 των συμμετοχών που ζητάμε.

Με χρήση της εντολής EXPLAIN προκύπτουν τα εξής traces:



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	g1	index	PRIMARY,Theme_ID	PRIMARY	8	NULL	333	Using index; Using temporary; Using filesort
1	PRIMARY	g2	ref	PRIMARY,Theme_ID	PRIMARY	4	ntua_db_project.g1.Recipe_ID	1	Using where; Using index
1	PRIMARY	<derived2>	ref	key0	key0	4	ntua_db_project.g1.Recipe_ID	10	
2	DERIVED	Rating	ALL	Unique_Ratings	NULL	NULL	NULL	1500	Using temporary; Using filesort

Δημιουργούμε ένα εναλλακτικό Query Plan με τη χρήση του FORCE INDEX (Theme_ID) και προκύπτουν τα εξής traces:



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	g1	index	Theme_ID	Theme_ID	4	NULL	333	Using index; Using temporary; Using filesort
1	PRIMARY	<derived2>	ref	key0	key0	4	ntua_db_project.g1.Recipe_ID	10	
1	PRIMARY	g2	index	Theme_ID	Theme_ID	4	NULL	333	Using where; Using index; Using join buffer (flat, BNL join)
2	DERIVED	Rating	ALL	Unique_Ratings	NULL	NULL	NULL	1500	Using temporary; Using filesort

Παρατηρούμε ότι στη δεύτερη περίπτωση που το αναγκάσαμε να χρησιμοποιήσει το index Theme_ID, το οποίο δεν είναι το βέλτιστο σε αυτή την περίπτωση, τα rows αυξήθηκαν για το table g2 και άρα η απόδοση μειώνεται.

- ***Ερώτημα 3.7/query7:***

```sql
-- DONE!
WITH EpisodeRatings AS (
    SELECT Person.Person_ID, Person.First_Name, Person.Last_Name, Rating.Cook_ID
    FROM Rating
    JOIN Cook ON Rating.Cook_ID = Cook.Cook_ID
    JOIN Person ON Cook.Person_ID = Person.Person_ID
    GROUP BY Rating.Recipe_ID, Rating.Cook_ID, Rating.Episode_ID
),
AppearancesPerCook AS (
    SELECT EpisodeRatings.Cook_ID, COUNT(*) AS AppearancesCounter
    FROM EpisodeRatings
    GROUP BY EpisodeRatings.Cook_ID
),
TotalMaxAppearances AS (
    SELECT MAX(AppearancesPerCook.AppearancesCounter)  AS MaxAppearances
    FROM AppearancesPerCook
)
SELECT AppearancesPerCook.Cook_ID, EpisodeRatings.First_Name, EpisodeRatings.Last_Name, AppearancesPerCook.AppearancesCounter,
       TotalMaxAppearances.MaxAppearances
FROM AppearancesPerCook
JOIN EpisodeRatings ON AppearancesPerCook.Cook_ID = EpisodeRatings.Cook_ID
CROSS JOIN TotalMaxAppearances
WHERE AppearancesPerCook.AppearancesCounter <= TotalMaxAppearances.MaxAppearances - 5
GROUP BY AppearancesPerCook.Cook_ID, AppearancesPerCook.AppearancesCounter
ORDER BY AppearancesPerCook.AppearancesCounter DESC;
```

Σε αυτό το ερώτημα, αρχικά αντιστοιχίζουμε κάθε μάγειρα με όλες τις συμμετοχές του στον διαγωνισμό. Έπειτα, μετράμε πόσες συμμετοχές είχε κάθε μάγειρας στον διαγωνισμό και αμέσως μετά από αυτόν τον πίνακα βρίσκουμε τον μεγαλύτερο αριθμό συμμετοχών που έχει ένας μάγειρας . Τέλος, κάνουμε JOIN τους παραπάνω πίνακες που βρήκαμε, ώστε να βρούμε εν τέλει μόνο τους μάγειρες, οι οποίοι είχαν τουλάχιστον 5 λιγότερες συμμετοχές από τον μάγειρα με τις περισσότερες .

- ***Ερώτημα 3.8/query8:***

```sql
-- DONE!
WITH RecipesCooked AS (
    SELECT Uses.Utensil_ID, Rating.Episode_ID
    FROM Rating
    JOIN Recipe ON Rating.Recipe_ID = Recipe.Recipe_ID
    JOIN Uses ON Recipe.Recipe_ID = Uses.Recipe_ID
    -- JOIN Utensil ON Uses.Utensil_ID = Utensil.Utensil_ID
    GROUP BY Rating.Recipe_ID, Rating.Cook_ID, Rating.Episode_ID, Uses.Utensil_ID
),
UtensilsPerEpisode AS (
    SELECT RecipesCooked.Episode_ID, COUNT(*) AS TimesUtensilUsed
    FROM RecipesCooked
    GROUP BY RecipesCooked.Episode_ID
    ORDER BY RecipesCooked.Episode_ID
),
MaxTimesUtensilUsed AS (
    SELECT MAX(TimesUtensilUsed) AS MaxCounter
    FROM UtensilsPerEpisode
)
SELECT UtensilsPerEpisode.Episode_ID, UtensilsPerEpisode.TimesUtensilUsed
FROM UtensilsPerEpisode JOIN MaxTimesUtensilUsed ON UtensilsPerEpisode.TimesUtensilUsed = MaxTimesUtensilUsed.MaxCounter;
```

Εδώ αρχικά βρίσκουμε ποια ακριβώς εξαρτήματα χρησιμοποιήθηκαν σε κάθε επεισόδιο. Στη συνέχεια, υπολογίζουμε τον αριθμό των εξαρτημάτων που χρησιμοποιήθηκαν σε κάθε επεισόδιο. Και στο τέλος, από αυτόν τον πίνακα βρίσκουμε το επεισόδιο στο οποίο χρησιμοποιήθηκαν τα περισσότερα εξαρτήματα.

Με χρήση της εντολής EXPLAIN προκύπτουν τα εξής traces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived5>	ALL	NULL	NULL	NULL	NULL	22	
1	PRIMARY	<derived4>	ref	key0	key0	9	UtensilsPerEpisode.TimesUtensilUsed	2	
5	DERIVED	<derived6>	ALL	NULL	NULL	NULL	NULL	22	Using temporary; Using filesort
6	DERIVED	Uses	index	PRIMARY	PRIMARY	8	NULL	1	Using index; Using temporary; Using filesort
6	DERIVED	Recipe	eq_ref	PRIMARY	PRIMARY	4	ntua_db_project.Uses.Recipe_ID	1	Using index
6	DERIVED	Rating	ref	Unique_Ratings	Unique_Ratings	4	ntua_db_project.Uses.Recipe_ID	22	Using index
4	DERIVED	<derived3>	ALL	NULL	NULL	NULL	NULL	22	
3	DERIVED	<derived2>	ALL	NULL	NULL	NULL	NULL	22	Using temporary; Using filesort
2	DERIVED	Uses	index	PRIMARY	PRIMARY	8	NULL	1	Using index; Using temporary; Using filesort
2	DERIVED	Recipe	eq_ref	PRIMARY	PRIMARY	4	ntua_db_project.Uses.Recipe_ID	1	Using index
2	DERIVED	Rating	ref	Unique_Ratings	Unique_Ratings	4	ntua_db_project.Uses.Recipe_ID	22	Using index

Με τη χρήση ενός STRAIGHT_JOIN προκύπτουν τα εξής traces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived5>	ALL	NULL	NULL	NULL	NULL	22	
1	PRIMARY	<derived4>	ref	key0	key0	9	UtensilsPerEpisode.TimesUtensilUsed	2	
5	DERIVED	<derived6>	ALL	NULL	NULL	NULL	NULL	22	Using temporary; Using filesort
6	DERIVED	Uses	index	PRIMARY	PRIMARY	8	NULL	1	Using index; Using temporary; Using filesort
6	DERIVED	Rating	ref	Unique_Ratings	Unique_Ratings	4	ntua_db_project.Uses.Recipe_ID	22	Using index
6	DERIVED	Recipe	eq_ref	PRIMARY	PRIMARY	4	ntua_db_project.Uses.Recipe_ID	1	Using index
4	DERIVED	<derived3>	ALL	NULL	NULL	NULL	NULL	22	
3	DERIVED	<derived2>	ALL	NULL	NULL	NULL	NULL	22	Using temporary; Using filesort
2	DERIVED	Uses	index	PRIMARY	PRIMARY	8	NULL	1	Using index; Using temporary; Using filesort
2	DERIVED	Rating	ref	Unique_Ratings	Unique_Ratings	4	ntua_db_project.Uses.Recipe_ID	22	Using index
2	DERIVED	Recipe	eq_ref	PRIMARY	PRIMARY	4	ntua_db_project.Uses.Recipe_ID	1	Using index

Παρατηρούμε ότι δεν υπάρχει κάποια διαφορά στις γραμμές, δηλαδή με τη χρήση του STRAIGHT_JOIN σε αυτή την περίπτωση δεν καταφέραμε ούτε να βελτιώσουμε την επίδοση, αλλά ούτε και να τη μειώσουμε.

- ***Ερώτημα 3.9/query9:***

```sql
-- DONE!
WITH CookedRecipes AS (
    SELECT Recipe_ID, Episode.Episode_ID, YEAR(Episode_Release_Date) AS EpisodeYear
    FROM Rating JOIN Episode ON Rating.Episode_ID = Episode.Episode_ID
    -- Used to remove duplicate recipes due to ratings for x critics.
    -- Cook matters in case requirements change and 2 cooks can cook same recipe in episode
    GROUP BY Recipe_ID, Cook_ID, Episode_ID
)
SELECT ROUND(AVG(ROUND((Requires.Quantity / 100) * Macronutrients.Carbs_100g, 2)), 2) AS AverageCarbs, CookedRecipes.EpisodeYear
FROM CookedRecipes
JOIN Requires ON CookedRecipes.Recipe_ID = Requires.Recipe_ID
JOIN Macronutrients ON Requires.Ingredient_ID = Macronutrients.Ingredient_ID
GROUP BY CookedRecipes.EpisodeYear;
```

Εδώ αρχικά, βρίσκουμε όλες τις συνταγές που έχουν μαγειρευτεί στον διαγωνισμό. Και μετά με JOIN αντιστοιχίζουμε την ποσότητα Carbs ανά συνταγή και ανά χρόνο, βρίσκοντας έτσι τον μέσο όρο Carbs ανά χρόνο στον διαγωνισμό.

- ***Ερώτημα 3.10/query10:***

```sql
WITH YearPairs AS (
    SELECT DISTINCT YEAR(e1.Episode_Release_Date) AS Year1, YEAR(e2.Episode_Release_Date) AS Year2
    FROM Episode e1
    JOIN Episode e2 ON YEAR(e2.Episode_Release_Date) = YEAR(e1.Episode_Release_Date) + 1
),
YearCuisines AS (
    SELECT YEAR(e.Episode_Release_Date) AS EpisodeYear, i.Cuisine_ID, COUNT(*) AS CuisineAppearancesperYear
    FROM Episode e
    JOIN Involved i ON e.Episode_ID = i.Episode_ID
    GROUP BY YEAR(e.Episode_Release_Date), i.Cuisine_ID
)
SELECT c.Cuisine_Name, Year1Cuisines.CuisineAppearancesperYear + Year2Cuisines.CuisineAppearancesperYear AS Total_Participations,
     YearPairs.Year1, YearPairs.Year2
FROM YearPairs
JOIN YearCuisines AS Year1Cuisines ON YearPairs.Year1 = Year1Cuisines.EpisodeYear
JOIN YearCuisines AS Year2Cuisines ON YearPairs.Year2 = Year2Cuisines.EpisodeYear
    AND Year1Cuisines.Cuisine_ID = Year2Cuisines.Cuisine_ID
JOIN Cuisine c ON Year1Cuisines.Cuisine_ID = c.Cuisine_ID
WHERE Year1Cuisines.CuisineAppearancesperYear >= 3 AND Year2Cuisines.CuisineAppearancesperYear >= 3
-- GROUP BY Total_Participations, Year1, Year2
-- HAVING COUNT(*) > 1
ORDER BY Year1 ASC, Year2 ASC, Total_Participations ASC;
```

Αρχικά βρίσκουμε όλα τα διαδοχικά ζευγάρια χρόνων και το πλήθος των εμφανίσεων κάθε κουζίνας ανά χρόνο στον διαγωνισμό. Συνδυάζοντας τα δύο παραπάνω αποτελέσματα με την προϋπόθεση ότι οι χρόνοι είναι διαδοχικοί επιλέγουμε τις κουζίνες που είχαν παραπάνω από 3 εμφανίσεις και στους δύο χρόνους ξεχωριστά.

- ***Ερώτημα 3.11/query11:***

```sql
-- DONE!
WITH ScoreCookPerCritic AS (
    SELECT Cook_ID, Critic_ID, SUM(Score) AS SumScore
    FROM Rating
    GROUP BY Cook_ID, Critic_ID
),
RankedCritics AS (
    SELECT Cook_ID, Critic_ID, SumScore, ROW_NUMBER() OVER (PARTITION BY Cook_ID ORDER BY SumScore DESC, RAND()) AS Rank
    FROM ScoreCookPerCritic
)
SELECT Cook_ID, Critic_ID, SumScore
FROM RankedCritics
WHERE Rank <= 5;
```

Βρίσκουμε το άθροισμα όλων των βαθμολογιών που έχει δώσει κάθε κριτής σε κάθε μάγειρα και με την χρήση του ROW_NUMBER() OVER (PARTITION…) τα διατάσουμε ανά μάγειρα σύμφωνα με το παραπάνω άθροισμα. Σε περίπτωση ισοβαθμίας η σειρά καθορίζεται τυχαία. Τέλος διαλέγοντας για Rank <=5 παίρνουμε τους top 5 κριτές από τους οποίους κάθε μάγειρας έλαβε τις μεγαλύτερες βαθμολογίες συνολικά.

- ***Ερώτημα 3.12/query12:***

```sql
-- DONE!
WITH EpisodeRatings AS (
    SELECT e.Episode_ID, e.Episode_Release_Date, re.Difficulty
    FROM Episode e
    JOIN Rating ra ON ra.Episode_ID = e.Episode_ID
    JOIN Recipe re ON re.Recipe_ID = ra.Recipe_ID
    GROUP BY ra.Recipe_ID, ra.Cook_ID, ra.Episode_ID
),
AVG_DIFF AS (
SELECT EpisodeRatings.Episode_ID, YEAR(EpisodeRatings.Episode_Release_Date) AS EpisodeYear,
       ROUND(AVG(EpisodeRatings.Difficulty), 1) AS avg_difficulty
FROM EpisodeRatings
GROUP BY EpisodeYear, EpisodeRatings.Episode_ID
),
MAX_DIFF AS (
SELECT Episode_ID, EpisodeYear, avg_difficulty
, ROW_NUMBER() OVER (PARTITION BY EpisodeYear ORDER BY avg_difficulty DESC, RAND()) AS Ranking
    FROM AVG_DIFF
)
SELECT Episode_ID, EpisodeYear, avg_difficulty
FROM MAX_DIFF
WHERE Ranking = 1;
```

Αρχικά επιλέγουμε μοναδικά κάθε rating αφαιρώντας τις επαναλήψεις λόγω των 3 κριτών μέσω GROUP BY. Έπειτα βρίσκουμε την μέση βαθμολογία ανά χρόνο και από αυτήν επιλέγουμε την μέγιστη. Με χρήση του ROW_NUMBER αποδίδουμε ένα ranking στις γραμμές του πίνακα που ανήκουν στον ίδιο χρόνο και παίρνουμε εκείνο με ranking 1, δηλαδή την μέγιστη δυσκολία, ανά έτος. Κάθε βήμα αποθηκεύεται ως ένας προσωρινός πίνακας, Common Table Expression (CTE), με χρήση του keyword WITH.

- ***Ερώτημα 3.13/query13:***

```sql
-- DONE!
WITH EpisodeRatings AS (
    SELECT Rating.Cook_ID, Rating.Episode_ID
    FROM Rating
    GROUP BY Rating.Cook_ID, Rating.Episode_ID
),
CookRanks AS(
SELECT EpisodeRatings.Episode_ID, SUM(Cook_Rank_Comparison.Rank_Number) AS CookRankSum
FROM EpisodeRatings
JOIN Cook ON EpisodeRatings.Cook_ID  = Cook.Cook_ID
JOIN Cook_Rank_Comparison ON Cook.Cook_Rank  = Cook_Rank_Comparison.Cook_Rank
GROUP BY EpisodeRatings.Episode_ID
),
CookCriticRanks AS (
    SELECT CriticRating.Episode_ID, SUM(Cook_Rank_Comparison.Rank_Number) AS CriticRankSum
    FROM (SELECT * FROM Rating GROUP BY Rating.Critic_ID, Rating.Episode_ID) AS CriticRating
    JOIN Critic ON CriticRating.Critic_ID = Critic.Critic_ID
    JOIN Cook ON Critic.Cook_ID  = Cook.Cook_ID
    JOIN Cook_Rank_Comparison ON Cook.Cook_Rank  = Cook_Rank_Comparison.Cook_Rank
    GROUP BY CriticRating.Episode_ID
)
SELECT SUM(CookRanks.CookRankSum + CookCriticRanks.CriticRankSum) AS TotalSum, CookRanks.Episode_ID
FROM CookRanks
JOIN CookCriticRanks ON CookRanks.Episode_ID = CookCriticRanks.Episode_ID
GROUP BY CookRanks.Episode_ID
ORDER BY TotalSum DESC, RAND()
LIMIT 1;
```

Αρχικά, βρίσκουμε τους μάγειρες και σε ποια επεισόδια πήραν μέρος. Έπειτα βρίσκουμε το άθροισμα επαγγελματικής κατάρτισης των μαγείρων ανά επεισόδιο και στην συνέχεια το άθροισμα επαγγελματικής κατάρτισης των κριτών ανά επεισόδιο. Τέλος, με τη βοήθεια αυτών των δύο πινάκων βρίσκουμε το συνολικό άθροισμα επαγγελματικής κατάρτισης ανά επεισόδιο.

- ***Ερώτημα 3.14/query14:***

```sql
-- DONE!
WITH ThemesAppearances AS (
    -- Divide by 3 because each recipe is graded by 3 critics
    SELECT Theme.Theme_ID, Theme.Theme_Name, COUNT(*)/3 AS Counter
    FROM Rating JOIN Groups_By ON Rating.Recipe_ID = Groups_By.Recipe_ID JOIN Theme ON Groups_By.Theme_ID = Theme.Theme_ID
    GROUP BY Theme_ID, Theme_Name
    ORDER BY Counter DESC
),
MaxCount AS (
    SELECT MAX(Counter) AS MaxCounter
    FROM ThemesAppearances
)
SELECT Theme_ID, Theme_Name, Counter
FROM ThemesAppearances JOIN MaxCount ON ThemesAppearances.Counter = MaxCount.MaxCounter;
```

Εδώ αρχίζουμε υπολογίζοντας πόσες φορές έχει εμφανιστεί μια θεματική ενότητα στον διαγωνισμό. Και στο τέλος από αυτό τον πίνακα βρίσκουμε το MAX ώστε να βρούμε την θεματική ενότητα που έχει εμφανιστεί τις περισσότερες φορές στον διαγωνισμό.

- ***Ερώτημα 3.15/query15:***

```sql
-- DONE!
(
    SELECT Food_Group_ID, Food_Group_Name
    FROM Food_Group
)
EXCEPT
(
    SELECT DISTINCT Food_Group.Food_Group_ID, Food_Group.Food_Group_Name FROM
    Rating JOIN Requires ON Rating.Recipe_ID = Requires.Recipe_ID
    JOIN Ingredient ON Requires.Ingredient_ID = Ingredient.Ingredient_ID
    JOIN Food_Group ON Ingredient.Food_Group_ID = Food_Group.Food_Group_ID
)
ORDER BY Food_Group_ID ASC;
```

Αρχικά, επιλέγουμε όλες τις ομάδες τροφίμων που έχουμε στην βάση και από αυτό τον πίνακα κάνουμε EXCEPT τον παρακάτω πίνακα ο οποίος έχει όλες τις ομάδες τροφίμων οι οποίες έχουν εμφανιστεί στον διαγωνισμό. Έτσι στο τέλος καταλήγουμε με τις ομάδες τροφίμων που δεν έχουν εμφανιστεί ποτέ στον διαγωνισμό.

Προαπαιτούμενα

Η εργασία έχει υλοποιηθεί με MariaDB ως RDBMS και χρήση MariaDB Connector/Python μέσω Jupyter Notebook για ορισμένα τμήματα της εφαρμογής, όπως κλήρωση διαγωνισμού, δυναμική δημιουργία Users και εισαγωγή ορισμένων dummy data. Για την εγκατάσταση των βιβλιοθηκών που απαιτούνται στην εκτέλεση των notebooks έχει δημιουργηθεί ένα περιβάλλον Anaconda με τα αντίστοιχα πακέτα το οποίο παρέχεται ως environment αρχείο `environment.yml`.

Αρχικά εγκαθιστούμε τα βασικά προαπαιτούμενα ακολουθώντας όποια prompt προκύψουν κατά την διάρκεια της εγκατάστασης σύμφωνα και με τις οδηγίες MariaDB Ubuntu Installation.

```
sudo chmod 774 ./requirements.sh
```

```
sudo ./requirements.sh
```

Σε περίπτωση που το Anaconda δεν είναι εγκατεστημένο μπορεί να εγκατασταθεί ακολουθώντας τα βήματα που παρουσιάζονται στην σελίδα Anaconda Linux Installation.

Δημιουργούμε έναν χρήστη με πολλαπλά privileges με τον οποίο θα γίνεται η αρχικοποίηση της βάσης. Ο χρήστης root δεν έχει αρχικά password άμα δεν έχουμε θέσει έναν, οπότε πατάμε απλά Enter.

```
sudo mariadb -u root
```

```
GRANT ALL PRIVILEGES ON *.* TO `myadmin`@`localhost` IDENTIFIED BY
'password' WITH GRANT OPTION;
```

```
GRANT SUPER ON *.* TO 'myadmin'@'localhost' WITH GRANT OPTION;
exit;
```

Αφού έχει εκκινήσει ο server θέλουμε να αρχικοποιήσουμε την βάση με δεδομένα και αντίστοιχους ρόλους Admin και Cook. Καθώς γίνεται χρήση πολλών administrative privileges μπορούμε για ευκολία να συνδεθούμε ως root ή ως κάποιος ισάξιος χρήστης. Στην συγκεκριμένη περίπτωση θα χρησιμοποιούμε τον χρήστη myadmin που δημιουργήσαμε παραπάνω.

```
mariadb -u myadmin -p
```

```
source ./Scripts/my_ddl.sql;
source ./Scripts/dummy_data.sql;
```

Σε νέο bash terminal εκτελούμε τις παρακάτω εντολές για την δημιουργία του anaconda environment και την εκτέλεση των jupyter notebooks.

```
conda env create -f ./environment.yml
```

```
conda activate ntua_db_test
```

```
jupyter notebook ./dummy_data.ipynb
```

Αρκεί η εκτέλεση όλων των κελιών μόνο του notebook `dummy_data.ipynb` για την εκτέλεση των συναρτήσεων. Μέσα στο notebook θα ζητηθεί το username και το password του χρήστη με τον οποίο θέλουμε να συνδεθούμε στην βάση. Στην συγκεκριμένη περίπτωση χρησιμοποιούμε τον χρήστη `'myadmin'@'localhost'` που ορίσαμε παραπάνω.

Τέλος επιστρέφουμε στο terminal του MariaDB για να εισάγουμε μερικά επιπλέον δεδομένα ώστε να υπάρχουν οι προϋποθέσεις που εξασφαλίζουν ορατό αποτέλεσμα για όλα τα ζητούμενα queries.

```
source ./Scripts/extra_dummy_data.sql;
```