

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

**Εργαστήριο Βάσεων Δεδομένων**

**Εξαμηνιαία Εργασία:**

**Ανάπτυξη Βάσης Δεδομένων για Διαγωνισμό Μαγειρικής**

Ομάδα Project: 102

Αναγνώστου Νικόλαος: el21818

Μάκρας Ηλίας: el21207

Μαρκουλιδάκης Γεώργιος: el21050

Περιεχόμενα

[**1.** **Σχεδίαση της Βάσης** 2](#_Toc167623437)

[**1.1** **Εφαρμογές που χρησιμοποιήθηκαν** 2](#_Toc167623438)

[**1.2** **Διάγραμμα ER** 2](#_Toc167623439)

[**1.3** **Σχεσιακό Διάγραμμα** 3](#_Toc167623440)

[**2.** **Υλοποίηση της Βάσης** 4](#_Toc167623441)

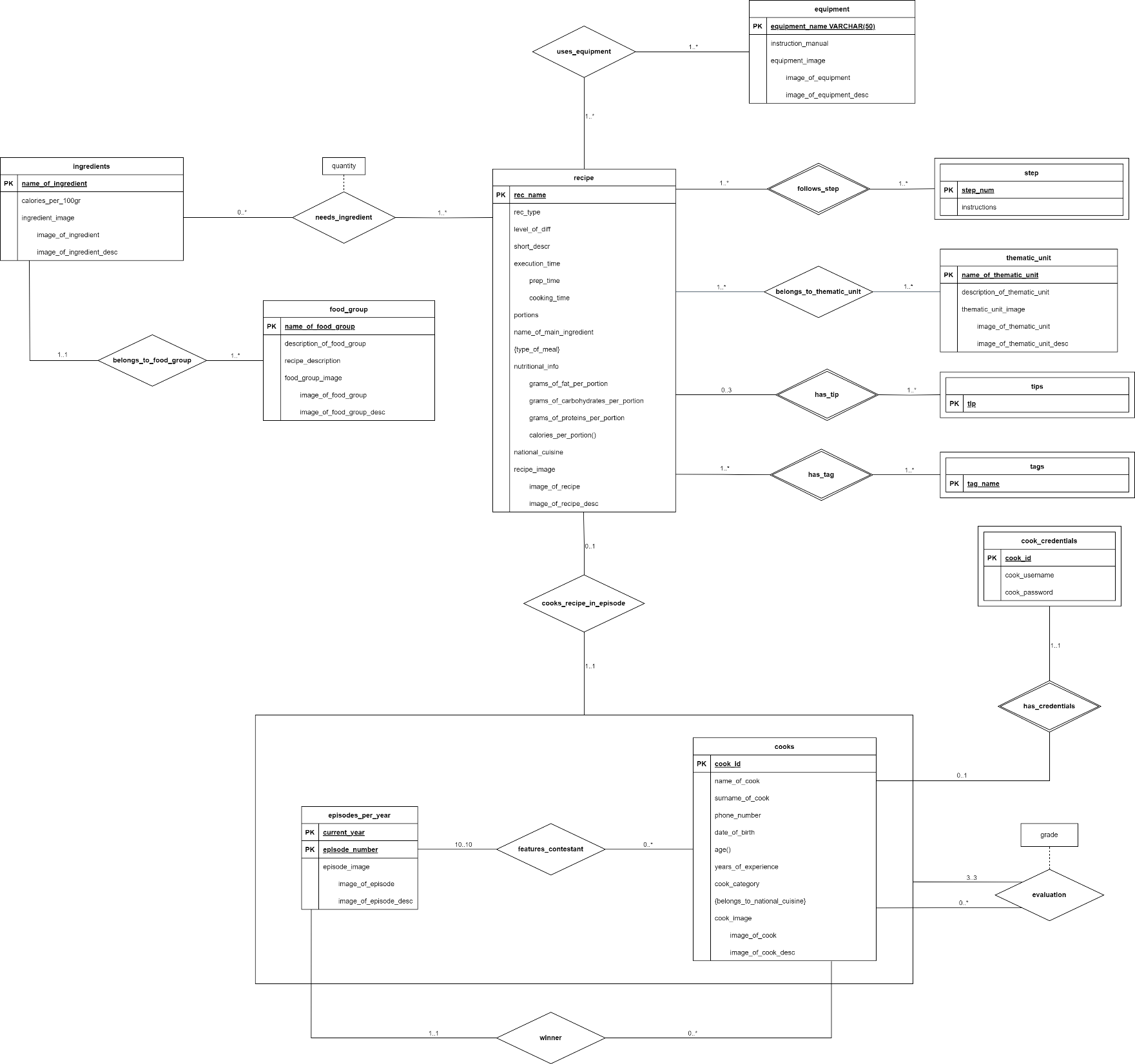
# **Σχεδίαση της Βάσης**

## **Εφαρμογές που χρησιμοποιήθηκαν**

Η εργασία μας σχεδιάστηκε και υλοποιήθηκε με τη χρήση του MySQL Workbench για την εκτέλεση των SQL scripts που δημιουργούν τη βάση και εκτελούν την κλήρωση του διαγωνισμού, καθώς και των queries.

Τα δεδομένα που εισήχθησαν στην βάση δημιουργήθηκαν με κώδικα Python που περιλαμβάνει τις βιβλιοθήκες faker και faker\_food, οι οποίες λειτούργησαν ως γεννήτριες dummy data.

## **Διάγραμμα ER**

Για την ορθή σχεδίαση της βάσης μας δημιουργήσαμε την αναπαράσταση των δεδομένων σε μοντέλο οντοτήτων-συσχετίσεων (ER). Το μοντέλο που σχεδιάσαμε φαίνεται στην ακόλουθη εικόνα:

Η εικόνα φαίνεται καλύτερα στο αντίστοιχο αρχείο στο GitHub Repository της εργασίας, στον σύνδεσμο [*https://github.com/nikosanag/DB\_Project*](https://github.com/nikosanag/DB_Project), με όνομα αρχείου *ER-DIAGRAM.png*.

Στο παραπάνω διάγραμμα έχουμε επιλέξει να χρησιμοποιήσουμε σε όλες τις σχέσεις τους συμβολισμούς άνω και κάτω ορίου, για λόγους συνέπειας. Για τη δημιουργία του διαγράμματος έχουμε λάβει τις εξής παραδοχές:

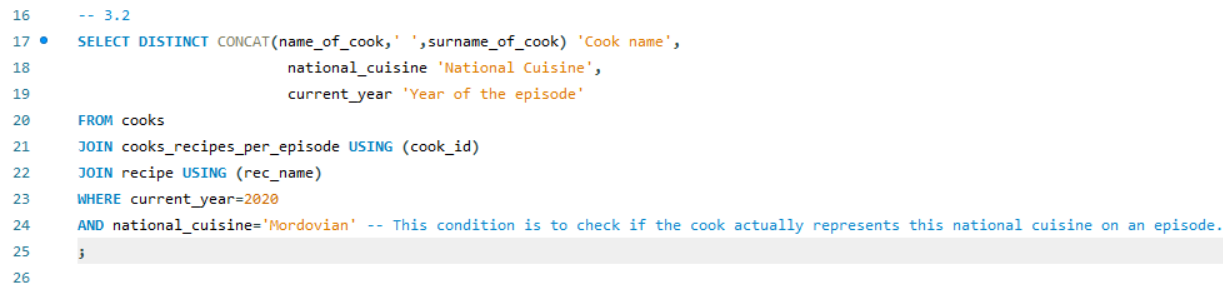
## **Σχεσιακό Διάγραμμα**

# **Υλοποίηση της Βάσης**

Στο

Indexes/Ευρετήρια

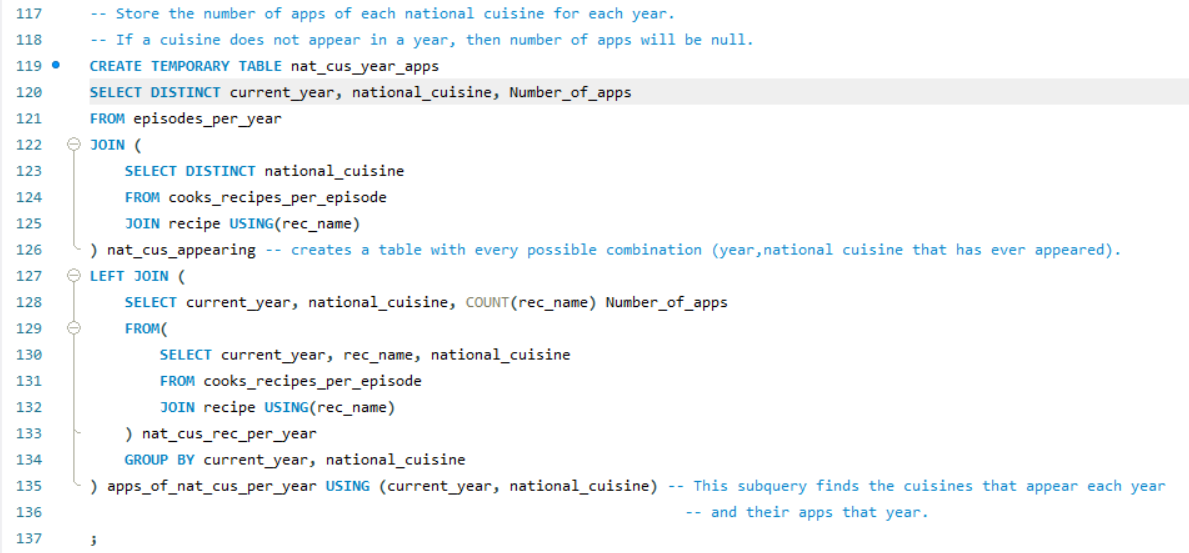
Όπως γνωρίζουμε, για όποια attributes έχουν καθοριστεί ως κλειδιά, είτε Primary Keys είτε Foreign Keys, δημιουργούνται αυτόματα και indexes από τη MySQL. Όπως είναι κατανοητό, τα περισσότερα queries θα βασίζονται σε αυτά τα attributes και συνεπώς δε θα απαιτούν επιπρόσθετα indexes. Υπάρχουν όμως κάποια queries που θα μπορούσαν να επωφεληθούν από τη χρήση κάποιων indexes. Συγκεκριμένα στο query 3.2 έχουμε τη συνθήκη: national\_cuisine = 'Ζητούμενη εθνική κουζίνα'.



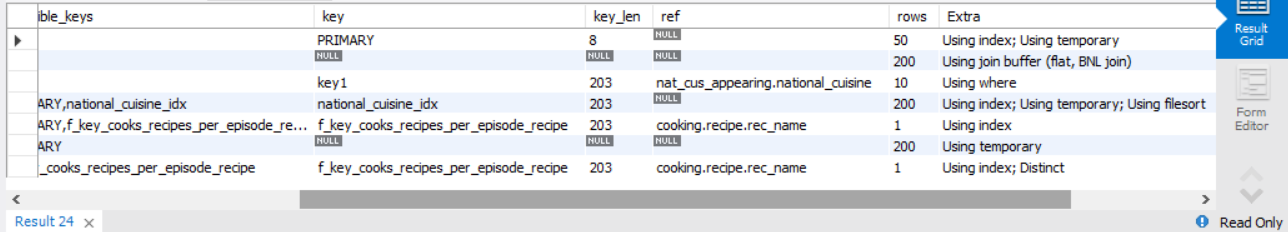
Τα indexes είναι κατάλληλα για τέτοιες συνθήκες αναζήτησης και συνεπώς θα ήταν καλή πρακτική να ορίσουμε ένα index για το table recipe στο attribute national\_cuisine.



Το συγκεκριμένο index χρησιμοποιείται και στο query 3.10, στο πρώτο subquery, ως συνθήκη JOIN.



Όπως μπορούμε να δούμε με τη βοήθεια της εντολής EXPLAIN, ο optimizer της SQL επιλέγει να χρησιμοποιήσει το συγκεκριμένο index.



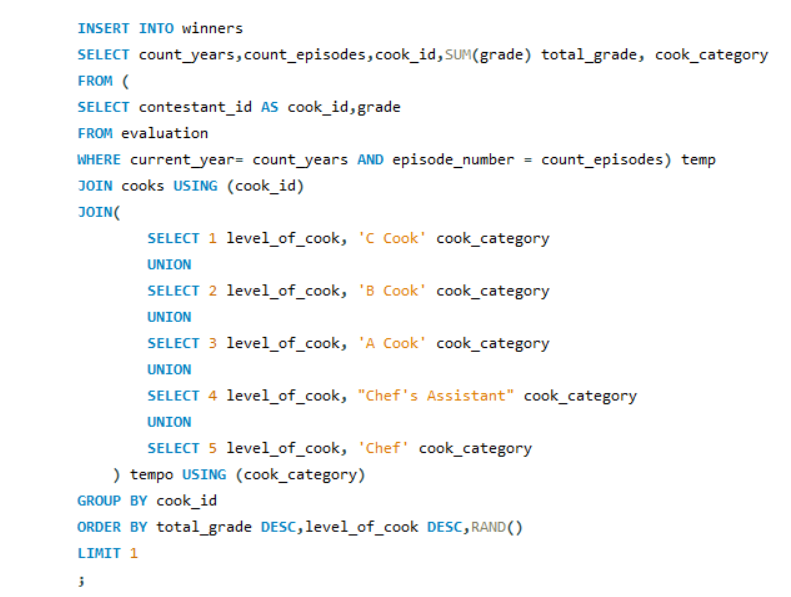
Επιπλέον, στο query 3.13 κάνουμε JOIN χρησιμοποιώντας το attribute cook\_category του cook.



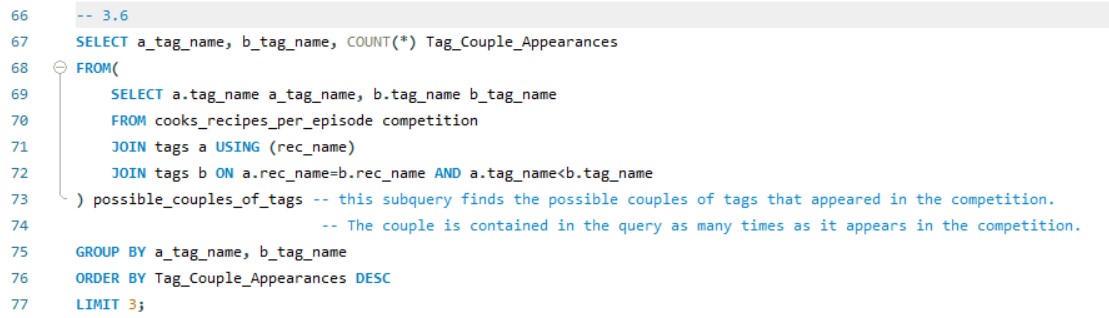
Μπορούμε να επισπεύσουμε τη διαδικασία του JOIN αυτού μέσω ενός index στο cook\_category:



Το συγκεκριμένο index, με την ίδια ακριβώς λογική, θα βοηθήσει και σε ένα subquery της κλήρωσης το οποίο επιλέγει τον νικητή του κάθε επεισοδίου:



Εναλλακτικά Query Plans

3.6) 

Σε αυτό το query ο optimizer της SQL επιλέγει να χρησιμοποιήσει το index του attribute rec\_name, τόσο στον πίνακα competition, όσο και στους δύο πίνακες tags. Η σειρά του JOIN επιλέγεται έτσι ώστε πρώτα να σκαναριστεί το table με τις λιγότερες γραμμές. Στα επόμενα tables του JOIN θα γίνει χρήση indexes και θα αποφευχθεί το σκανάρισμα των περισσότερων γραμμών. Παρακάτω φαίνονται τα αντίστοιχα traces που μπορούμε να δούμε μέσω της εντολής EXPLAIN και επιλέγοντας το FORMAT «JSON». Με κόκκινο φαίνονται τα indexes που χρησιμοποιούνται και με μπλε οι γραμμές που σκαναρίστηκαν από κάθε table. Όπως βλέπουμε το competition σκανάρεται 500 φορές (όσες και οι γραμμές του), ενώ τα δύο tables tags μόλις 2 φορές.

{

"query\_block": {

"select\_id": 1,

"filesort": {

"sort\_key": "count(0) desc",

"temporary\_table": {

"table": {

"table\_name": "competition",

"access\_type": "range",

"possible\_keys": ["f\_key\_cooks\_recipes\_per\_episode\_recipe"],

"key": "f\_key\_cooks\_recipes\_per\_episode\_recipe",

"key\_length": "203",

"used\_key\_parts": ["rec\_name"],

"rows": 500,

"filtered": 100,

"attached\_condition": "competition.rec\_name is not null and competition.rec\_name is not null",

"using\_index": true

},

"table": {

"table\_name": "a",

"access\_type": "ref",

"possible\_keys": ["PRIMARY", "f\_key\_tags\_recipe"],

"key": "f\_key\_tags\_recipe",

"key\_length": "202",

"used\_key\_parts": ["rec\_name"],

"ref": ["cooking.competition.rec\_name"],

"rows": 2,

"filtered": 100,

"using\_index": true

},

"table": {

"table\_name": "b",

"access\_type": "ref",

"possible\_keys": ["PRIMARY", "f\_key\_tags\_recipe"],

"key": "f\_key\_tags\_recipe",

"key\_length": "202",

"used\_key\_parts": ["rec\_name"],

"ref": ["cooking.competition.rec\_name"],

"rows": 2,

"filtered": 100,

"attached\_condition": "a.tag\_name < b.tag\_name",

"using\_index": true

}

}

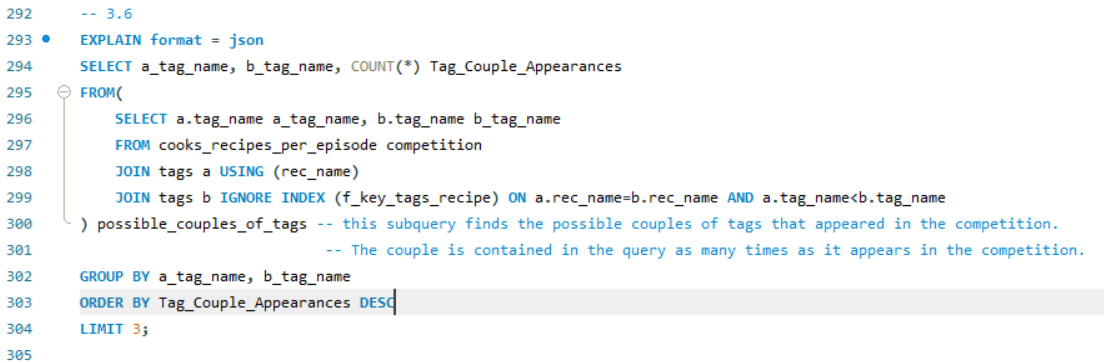
}

}

}

,

Εάν εξαναγκάσουμε τον optimizer της SQL να αγνοήσει το index του rec\_name, μέσω της εντολής IGNORE INDEX, στον πίνακα tags b, ο optimizer θα αναγκαστεί να χρησιμοποιήσει το index του Primary Key του tags, που όμως δεν είναι κατάλληλο στη συγκεκριμένη περίπτωση λόγω της συνθήκης: a.tag\_name < b.tag\_name.

 Για αυτό θα επιλέξει ως πρώτο table στο JOIN το tags b, παρόλο που έχει περισσότερες γραμμές από το competition. Έτσι καταφέρνει να εξετάσει αυτήν τη συνθήκη μόνο 2 φορές , για τις 2 γραμμές που ικανοποίησαν τα JOINS.

{

"query\_block": {

"select\_id": 1,

"filesort": {

"sort\_key": "count(0) desc",

"temporary\_table": {

"table": {

"table\_name": "b",

"access\_type": "index",

"possible\_keys": ["PRIMARY"],

"key": "PRIMARY",

"key\_length": "404",

"used\_key\_parts": ["tag\_name", "rec\_name"],

"rows": 1028,

"filtered": 100,

"using\_index": true

},

"table": {

"table\_name": "competition",

"access\_type": "ref",

"possible\_keys": ["f\_key\_cooks\_recipes\_per\_episode\_recipe"],

"key": "f\_key\_cooks\_recipes\_per\_episode\_recipe",

"key\_length": "203",

"used\_key\_parts": ["rec\_name"],

"ref": ["cooking.b.rec\_name"],

"rows": 1,

"filtered": 100,

"using\_index": true

},

"table": {

"table\_name": "a",

"access\_type": "ref",

"possible\_keys": ["PRIMARY", "f\_key\_tags\_recipe"],

"key": "f\_key\_tags\_recipe",

"key\_length": "202",

"used\_key\_parts": ["rec\_name"],

"ref": ["cooking.b.rec\_name"],

"rows": 2,

"filtered": 100,

"attached\_condition": "a.tag\_name < b.tag\_name",

"using\_index": true

}

}

}

}

}

Τέλος, ένα ακόμα ενδιαφέρον query plan θα ήταν να εξαναγκάσουμε επιπλέον τον optimizer να εκτελέσει τα JOINS με τη συγκεκριμένη σειρά που έχουν γραφτεί τα tables στον κώδικα. Αυτό το επιτυγχάνουμε μέσω της εντολής STRAIGHT\_JOIN.

photo

Αυτή τη φορά ο optimizer δεν μπορεί να αποφύγει τη χρήση του index (tag\_name,rec\_name) του Primary Key για το JOIN με τη συνθήκη a.tag\_name < b.tag\_name. Έτσι καταφεύγει στη χρήση της τεχνικής Block Nested Loop Join. Δηλαδή αποθηκεύει σε κάθε εξωτερική λούπα σε έναν buffer μερικά υποψήφια tuples των άλλων 2 tables και τα εξετάζει ένα προς ένα στην εσωτερική λούπα. Τα traces φαίνονται παρακάτω:

{

"query\_block": {

"select\_id": 1,

"filesort": {

"sort\_key": "count(0) desc",

"temporary\_table": {

"table": {

"table\_name": "competition",

"access\_type": "range",

"possible\_keys": ["f\_key\_cooks\_recipes\_per\_episode\_recipe"],

"key": "f\_key\_cooks\_recipes\_per\_episode\_recipe",

"key\_length": "203",

"used\_key\_parts": ["rec\_name"],

"rows": 500,

"filtered": 100,

"attached\_condition": "competition.rec\_name is not null",

"using\_index": true

},

"table": {

"table\_name": "a",

"access\_type": "ref",

"possible\_keys": ["PRIMARY", "f\_key\_tags\_recipe"],

"key": "f\_key\_tags\_recipe",

"key\_length": "202",

"used\_key\_parts": ["rec\_name"],

"ref": ["cooking.competition.rec\_name"],

"rows": 2,

"filtered": 100,

"using\_index": true

},

"block-nl-join": {

"table": {

"table\_name": "b",

"access\_type": "index",

"possible\_keys": ["PRIMARY"],

"key": "PRIMARY",

"key\_length": "404",

"used\_key\_parts": ["tag\_name", "rec\_name"],

"rows": 1028,

"filtered": 100,

"using\_index": true

},

"buffer\_type": "flat",

"buffer\_size": "256Kb",

"join\_type": "BNL",

"attached\_condition": "b.rec\_name = competition.rec\_name and a.tag\_name < b.tag\_name"

}

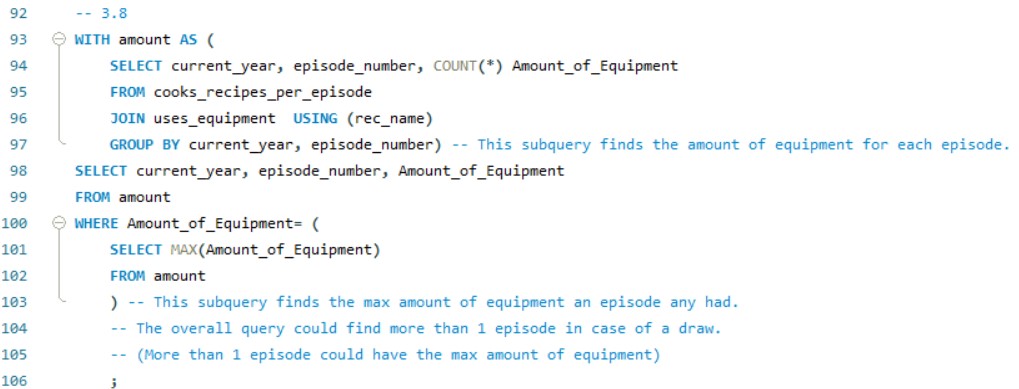
}

}

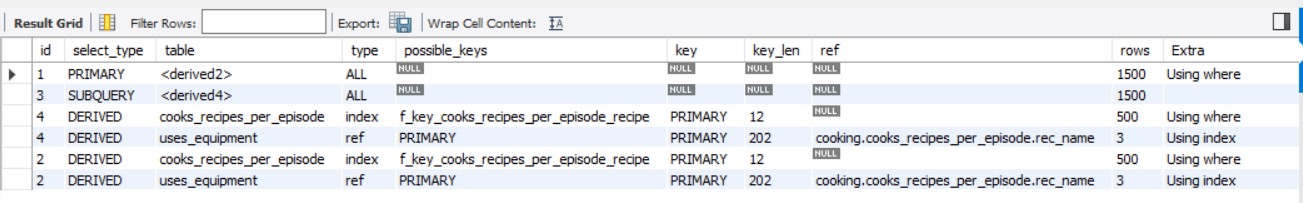
}

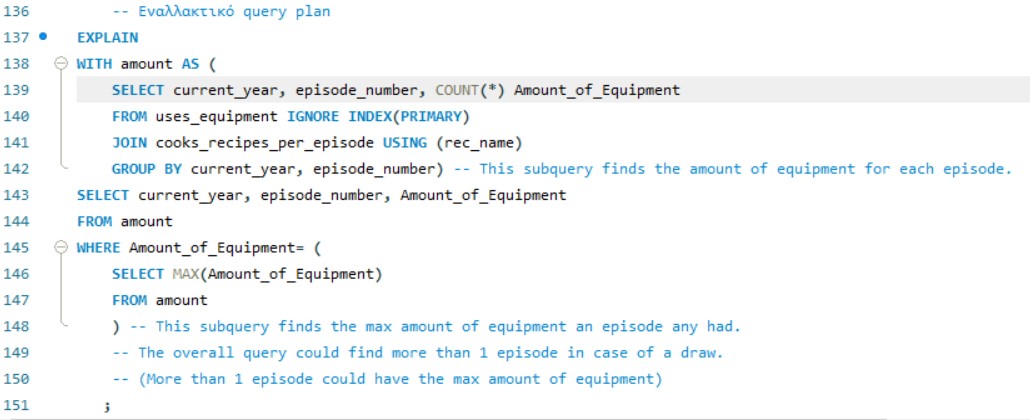
}

Φυσικά η συγκεκριμένη παρέμβαση μειώνει αρκετά την αποδοτικότητα του query, όπως φαίνεται από τις 1028 γραμμές που σκανάρονται στο τελευταίο table.

3.8) 

Σε αυτό το query ο optimizer επιλέγει να κάνει το JOIN του πρώτου subquery χρησιμοποιώντας το Primary Key του uses\_equipment, δηλαδή index στο rec\_name. Έτσι σκανάρει πρώτα το table cooks\_recipes\_per\_episode που έχει λιγότερες γραμμές (500 όπως φαίνεται παρακάτω) και στη συνέχεια χρησιμοποιεί το index για το uses\_equipment, σκανάροντας μόνο 3 γραμμές.



Ένα εναλλακτικό query plan θα ήταν να επιβάλλουμε στον optimizer να αγνοήσει το Primary Key του uses\_equipment. 

Αυτό θα οδηγήσει τον optimizer στο να χρησιμοποιήσει το index στο rec\_name του cooks\_recipes\_per\_episode, σκανάροντας πρώτα το uses\_equipment, το οποίο όπως φαίνεται παρακάτω έχει 1393 γραμμές, μειώνοντας την αποδοτικότητα του query.

