ΣΧΕΔΙΑΣΜΟΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

PROJECT – ΜΕΡΟΣ Β

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΓΕΩΡΓΙΟΣ ΧΕΙΡΜΠΟΣ

ΑΜ : 3130230

Ζητούμενο αυτής της εργασίας ήταν μελέτη χρήσεως δεικτών-indexes πάνω στου πίνακες της βάσης με σκοπό βελτιστοποίησης εκτέλεσης των ζητούμενων queries και επιπλέον αν μας συμφέρει τελικά η δημιουργία δεικτών στη βάση.

Σχολια(!):

Η εκτέλεση των script έγινε με την χρήση του παρακάτω SSD. Samsung SSD 840 EVO 2.5 inch.
http://www.samsung.com/global/business/semiconductor/minisite/SSD/M2M/html/ssd840evo/specifications.html
Κάθε εκτέλεση γίνοταν πάντοτε έπειτα από επανεκκίνηση της διεργασίας mysqld.exe και προσθεση εντολής SQL_NO_CACHE σε καθε select για να βεβαιώσουμε ότι το benchmarking ειναι "καθαρό". Τυχόν αποκλίσεις στους χρόνους πιθανολογείται να υπάρχουν.

Γιατί αυτά τα indexes?

Τα indexes τα οποία επέλεξα ήταν πάνω στους πίνακες balance και collateral, διαλεγοντας από τον balance ολες τις στήλες με unique και από τον collateral (collateral_relation_type).
Ο λόγος είναι ότι τα trending ζητούμενα από τα queries βρίσκονται ή/και εξαρτώνται από αυτούς τους πίνακες. Οπότε με αυτόν τον τρόπο θα ελαχιστοποιήσουμε τον χρόνο που χρειάζεται να αναζητήσουμε ζητούμενα δεδομένα από τους πίνακες. Αυτό θα επαληθευτεί στα στιγμιότυπα που παρουσιάζονται παρακάτω. Το unique προστέθηκε αφού διαπιστώθηκε ότι προσφέρει ένα μικρό ποσοστό μείωσης του κόστους εκτέλεσης των subqueries των ερωτημάτων επηρεάζοντας ελάχιστα τελικό χρόνος κια κόστος αυτών.

QUERY 1

Το πρώτο query το υλοποίησα με 3 διαφορετικούς τρόπους.

1)(best perfomance)

```
Select SQL_NO_CACHE count(Selection2.account_id) from
   (Select collateral.account_id, sum(collateral_amount) as sumcom, Selection1.maxbal from collateral
use index (collateralIndex),
      (Select account_id, max(balance_value) as maxbal from balance
      where balance_type = 'Capital'
      group by account_id) as Selection1
   where Selection1.account_id = collateral.account_id
```

```
    and collateral_relation_type = '3'
    group by collateral.account_id) as Selection2
where Selection2.maxbal > sumcom;
```

Ακολουθεί πλάνο εκτέλεσης για best query:

1)χωρίς index.

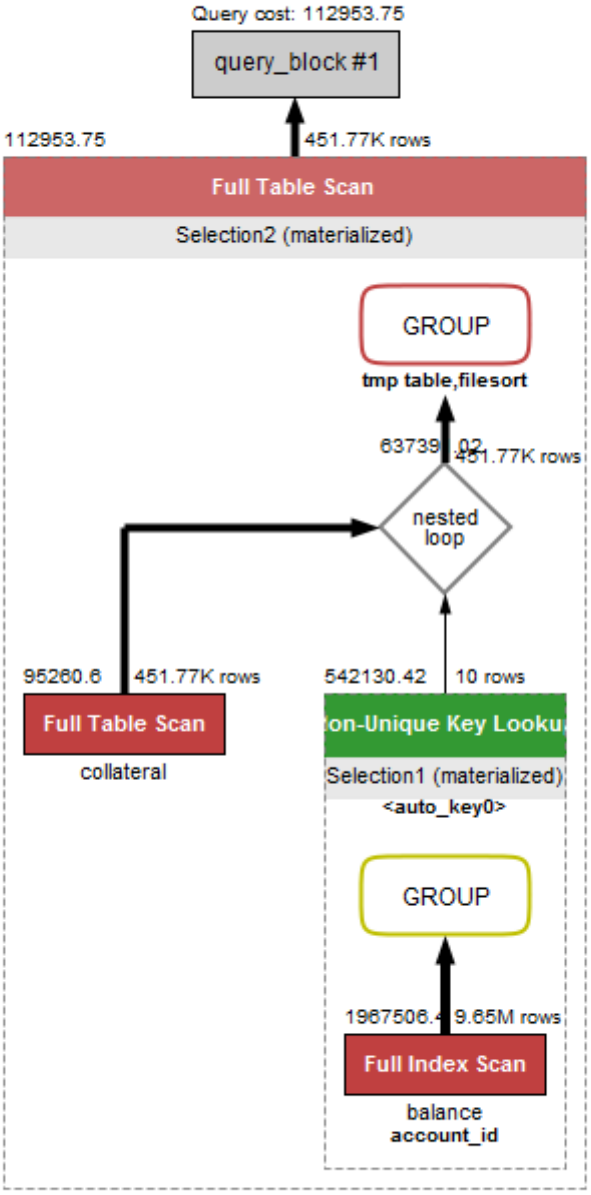
2)index οχι unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

3)index οχι unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

4)index unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

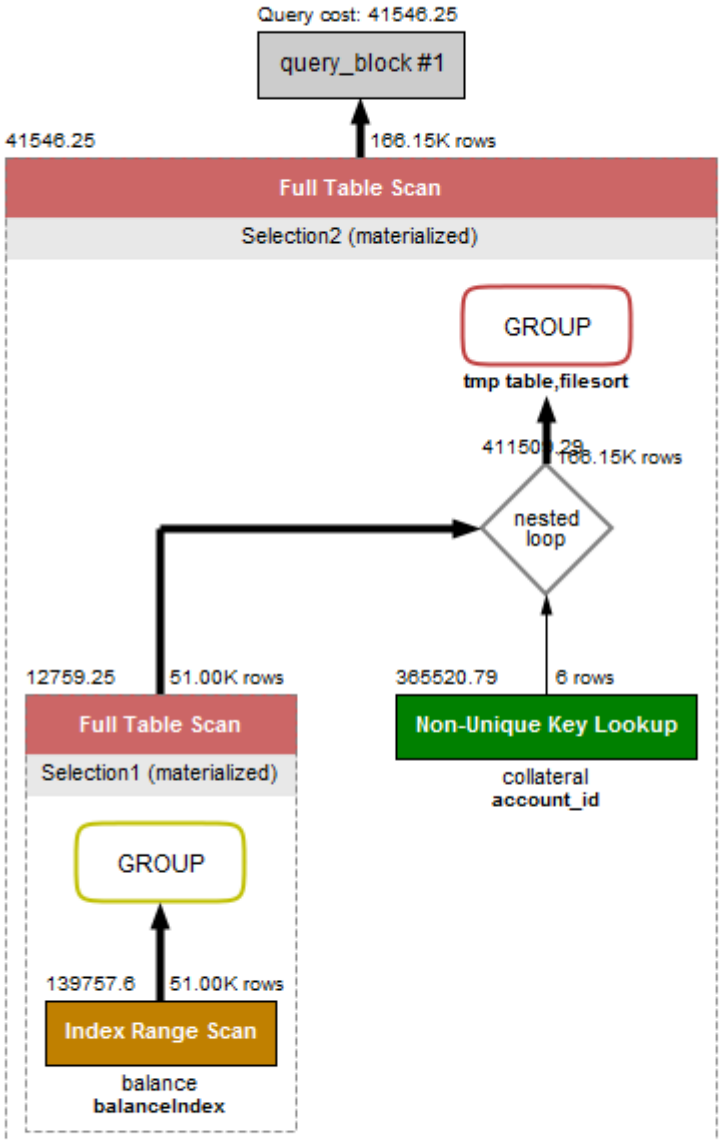
5)index unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

QUERY 1 - BEST - NO INDEX



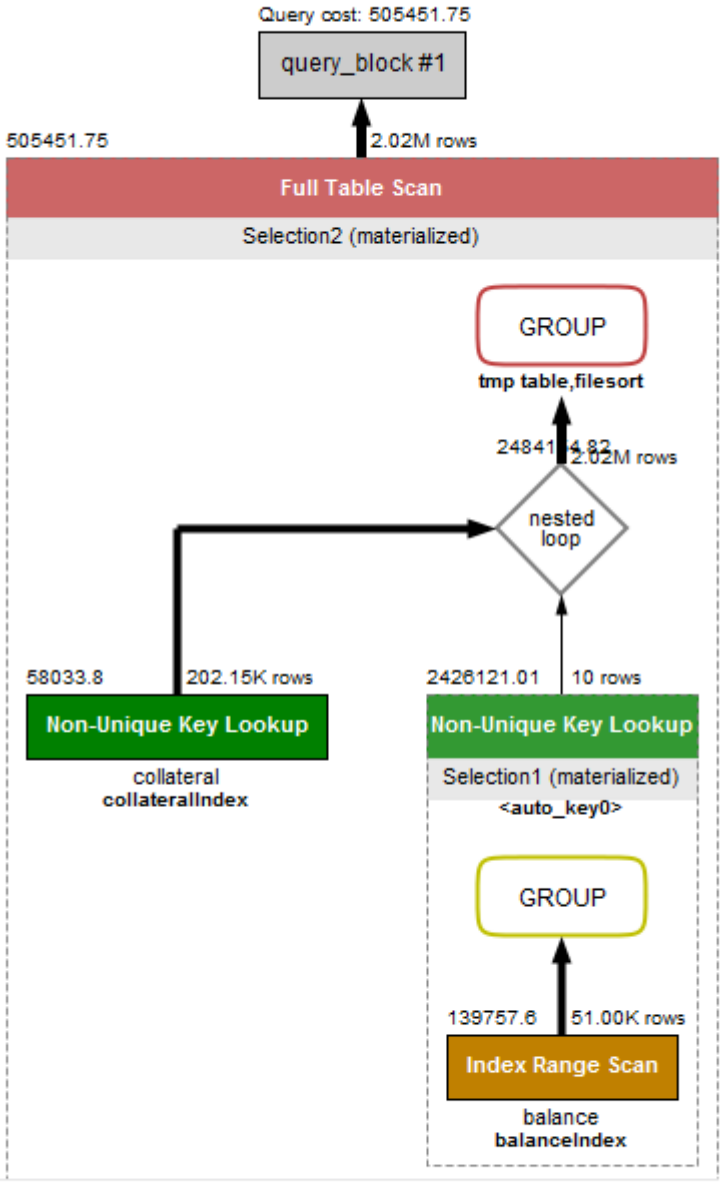
✓	4	15:35:53	Select SQL_NO_CACH	1 row 33.547 sec / 0.000 sec
✓	5	15:37:22	EXPLAIN Select SQL_N	OK 0.000 sec

QUERY 1- BEST -INDEX - NO UNIQUE - NO ENF



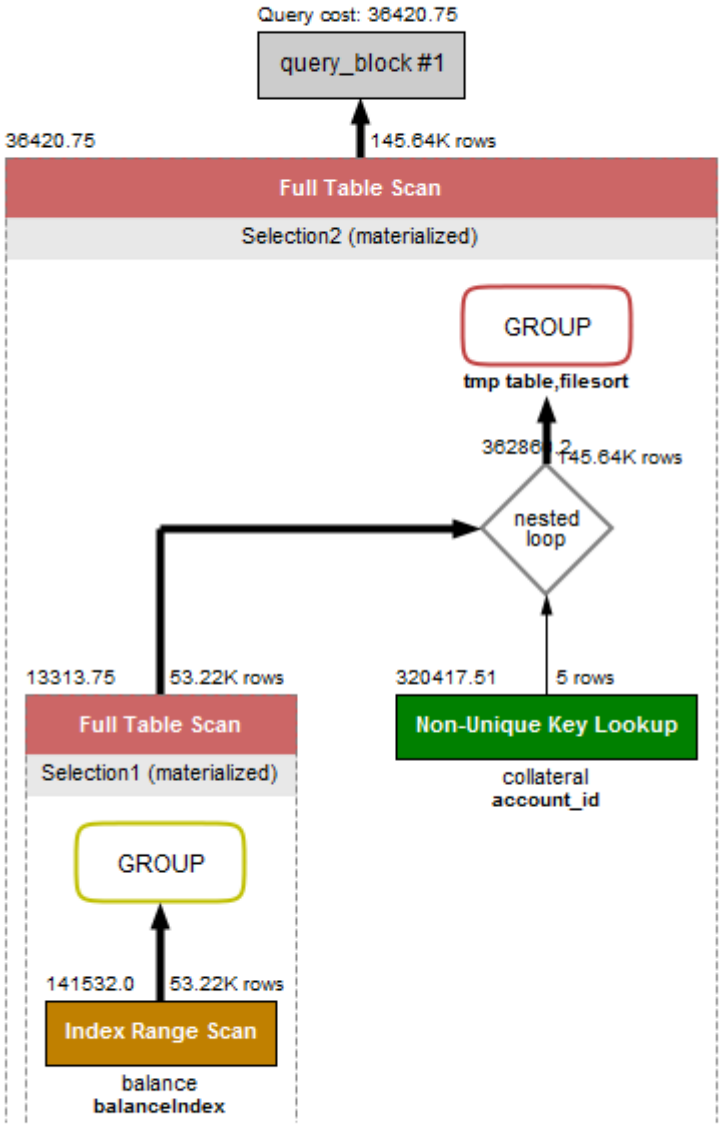
#	Time	Action	Message	Duration / Fetch
1	16:30:30	Select SQL_NO_CACHE coun	1 row(s) returnec	20.312 sec / 0.000 sec
2	16:30:53	EXPLAIN Select SQL_NO_CA	OK	0.000 sec

QUERY 1- BEST -INDEX - NO UNIQUE - ENF (COLLATERAL)



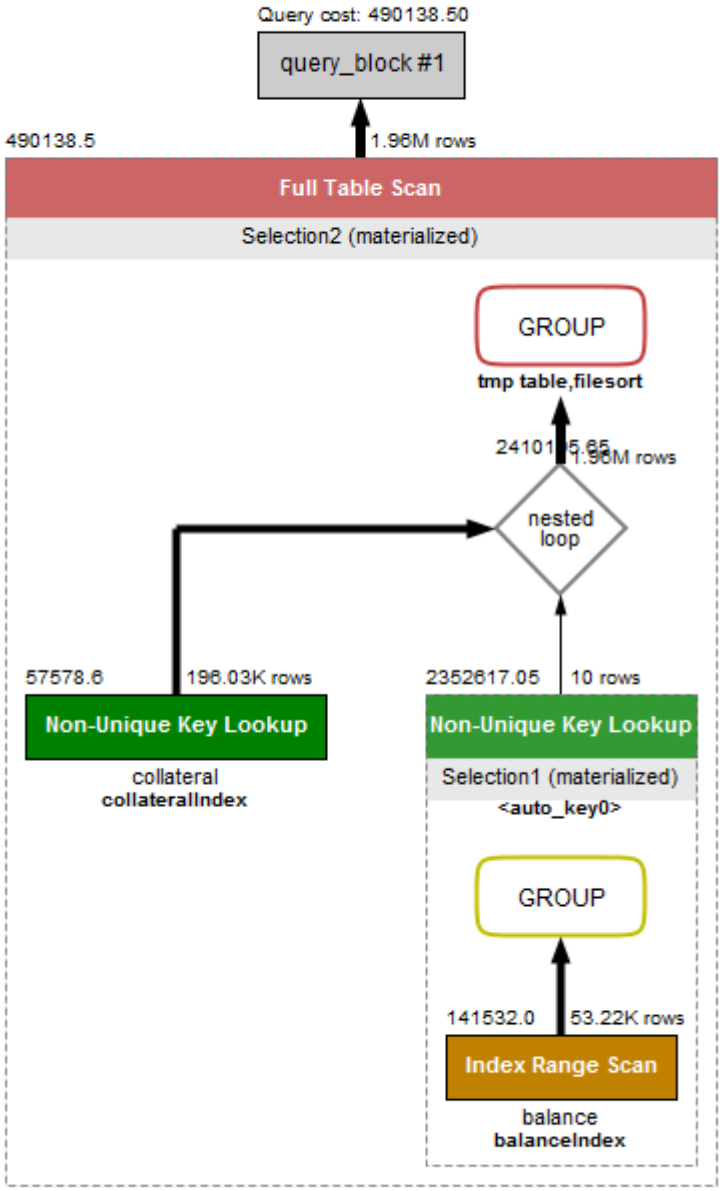
#	Time	Action	Message	Duration / Fetch
1	16:32:02	Select SQL_NO_CACHE count	1 row(s) retur	9.015 sec / 0.000 sec
2	16:32:13	EXPLAIN Select SQL_NO_CAC	OK	0.000 sec

QUERY 1 - BEST - INDEX - UNIQUE (BALANCE) - NO ENF



✔	4	17:14:47	Select SQL_NO_CACHE cou	1 row(s)	20.141 sec / 0.000 sec
✔	5	17:15:10	EXPLAIN Select SQL_NO_C	OK	0.000 sec

QUERY 1- BEST -INDEX - UNIQUE (BALANCE) - ENF(COLLATERAL



✔	7	17:16:07	Select SQL_NO_CA(	1 row(s)	9.360 sec / 0.000 sec
✔	8	17:16:18	EXPLAIN Select SQL	OK	0.000 sec

Παρατηρήσεις:
Απο τα sources των πλάνων εκτέλεσης βλέπουμε βελτίωση στον χρόνος εκτέλεσης με την χρήση του balanceIndex 10sec. Ο βελτιστοποιητής αγνοεί τον collateralIndex. Όταν γίνεται εξαναγκαστικη χρήση του index με την χρήση use index () στο query μας καταλήγουμε να κερδίσουμε ακόμα 10 sec.
Αρχικά το προφανές απο τον κώδικα του πλάνου εκτέλεσης είναι ότι με την χρήση του unique το query μας διαβάζει μερικά εκατομύρια γραμμές λιγότερες ανά κάθε join. Ένα περίεργο pattern που παρατηρείται είναι ότι παρόλο που το κόστος του ερωτήματος αυξάνεται αρκετά με την επιβαλλόμενη χρήση του collateralIndex ο χρόνος μειώθηκε αρκετά. Είτε το συγκρίνουμε με την περίπτωση που δεν εχουμε index είτε με την χρήση μόνο του balanceIndex είναι τουλάχιστον 5πλασιο το κόστος για να πετύχουμε βέλτιστο χρόνο.
(Επειδή και σε άλλα query παραλείπεται από τον optimizer το collateralIndex, κάθε φορά που εδινα εντολή να χρησιμοποιηθεί το κόστος του query ανέβαινε παρόλο που ο χρόνος μειωνόταν. Υποθέτω πως ο optimizer δίνει προτεραιότητα στο εννοιολογικό κόστος και όχι τόσο πολύ στον πραγματικό χρόνο εκτέλεσης.)

==========

2)**(better perfomance)**

```
Select SQL_NO_CACHE count(colac) as sumacc from
   (Select sum(collateral_amount) as sumcol, account_id as colac from collateral
   where collateral_relation_type = '3'
   group by colac) as selection1,
   (Select account_id as balac , max(balance_value) as balance_value from balance
   where balance_type = 'Capital'
group by account_id) as selection2
where selection1.colac = selection2.balac
and selection2.balance_value > selection1.sumcol;
```

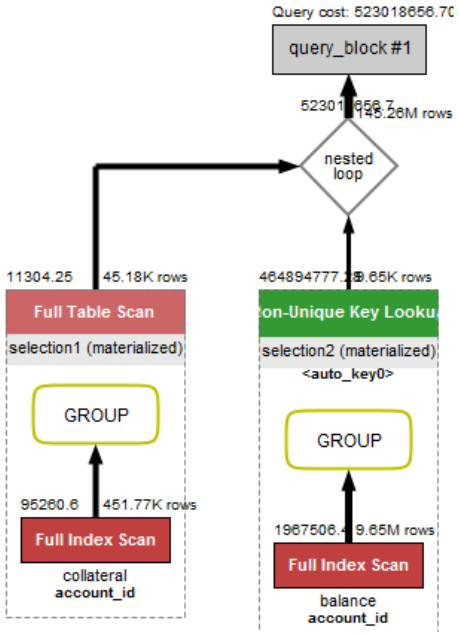
Ακολουθεί πλάνο εκτέλεσης για better query:

1)χωρίς index.

2)index οχι unique στον balance

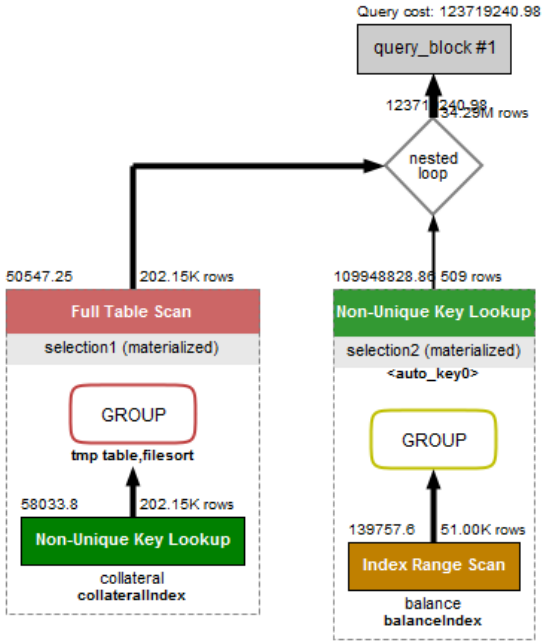
3)index unique στον balance

QUERY 1 - BETTER - NO INDEX



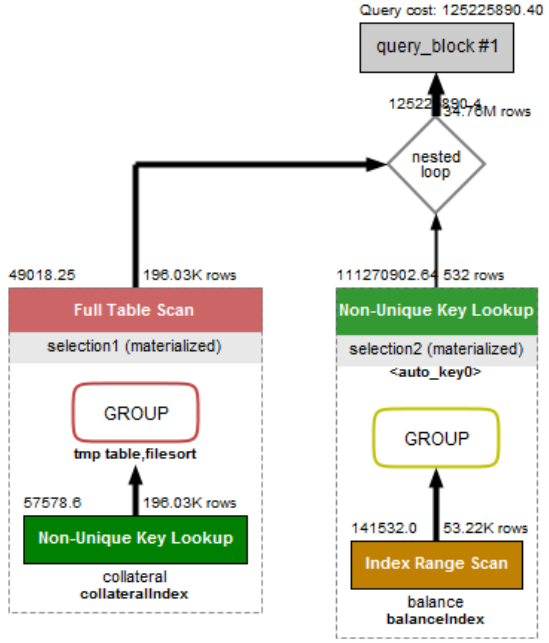
	#	Time	Action	Message	Duration / Fetch
✓	1	15:32:22	Select SQL_NO_CAC	1 row(s) returned	58.672 sec / 0.000 sec
✓	2	15:34:56	EXPLAIN Select SQL	OK	0.000 sec

QUERY 1 BETTER INDEX - NO UNIQUE (BALANCE)



	#	Time	Action	Message	Duration / Fetch
✓	1	16:29:01	Select SQL_NO_CAC	1 row(s) returned	9.578 sec / 0.000 sec
✓	2	16:29:15	EXPLAIN Select SQL	OK	0.000 sec

QUERY 1 BETTER INDEX UNIQUE (BALANCE)



	#	Time	Action	Message	Duration / Fetch
✓	1	17:13:48	Select SQL_NO_CACHE cour	1 row(s) returned	10.188 sec / 0.000 sec
✓	2	17:14:03	EXPLAIN Select SQL_NO_CA	OK	0.000 sec

Αντίθετα με το βέλτιστο query το κόστος εδώ έχει εκτοξευτεί στα ύψη. Στον κώδικα του πλάνου φαίνεται ότι παραγονται δεδομένα ήψους δις.("data_read_per_join": "8G"). Με την χρήση των index (στην περίπτωση αυτή ο optimizer χρησιμοποιεί και τους δυο). Ο χρόνος μειώνεται στο 1/6 του αρχικού και το κόστος περίπου στο 1/5. Με την χρήση του unique κερδίζουμε μερικές χιλιάδες στο κόστος, με αμελητέα διαφορά στον χρόνο. Το query το έχω στην δεύτερη θέση γιατί παρόλο που μπορεί να ανταγωνιστεί σε χρόνο το άνω, το κόστος του είναι πάρα πολύ μεγαλύτερο.

Σε αυτό το σημείο είναι που μου δημιουργήθηκε η απορία. Πως γίνεται να υπάρχει τόσο μεγάλη δυσαναλογία μεταξύ χρόνου και κόστους σε ενα query. Το κόστος του καλύτερου query έγινε μεγαλύτερο παρόλαυτα ο χρόνος μειώθηκε. Το κόστος του 2ο καλύτερου query υπολογίζεται στα δεκάδες εκατομμύρια και ο χρόνος του απλά είναι διπλασιος (χωρις index). Και τελικά στην βελτιστοποιησή του πάλι το κόστος είναι στα εκατομμύρια αλλα ο χρόνος μειώθηκε στα ίδια επίπεδα με του καλύτερου.

---

3)**(worst perfomance)**

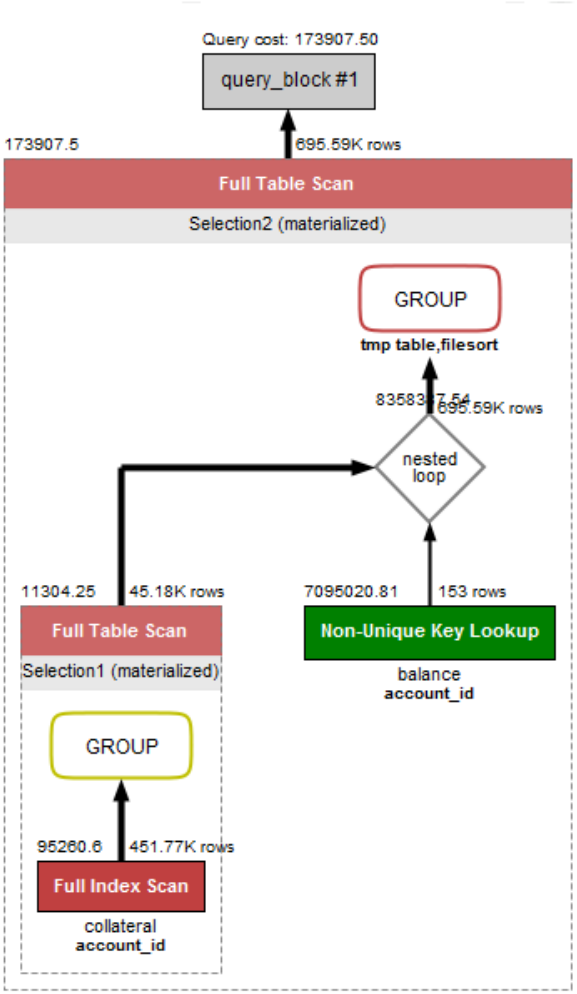
```
Select SQL_NO_CACHE count(Selection2.account_id) from
(Select balance.account_id, Selection1.sumcom , max(balance_value) as maxbal from balance,
   (Select sum(collateral_amount) as sumcom, account_id from collateral
   where collateral_relation_type = '3'
      group by account_id) as Selection1
   where balance.account_id = Selection1.account_id
   and balance_type = 'Capital'
   group by balance.account_id) as Selection2
where Selection2.maxbal > sumcom;
```

Ακολουθεί πλάνο εκτέλεσης για worst query:
1)χωρίς index.

2)index οχι unique στον balance

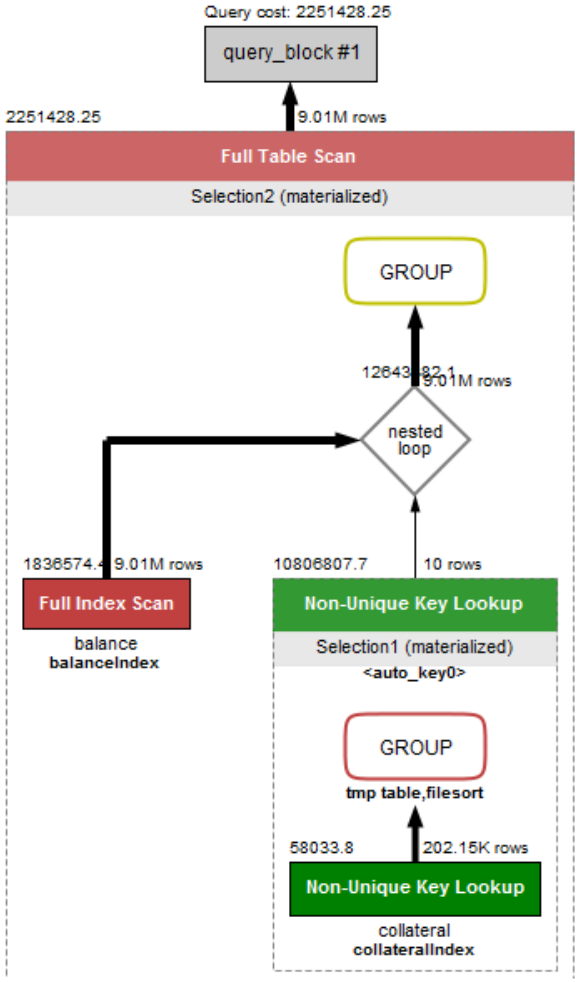
3)index unique στον balance

QUERY 1 - WORST - NO INDEX



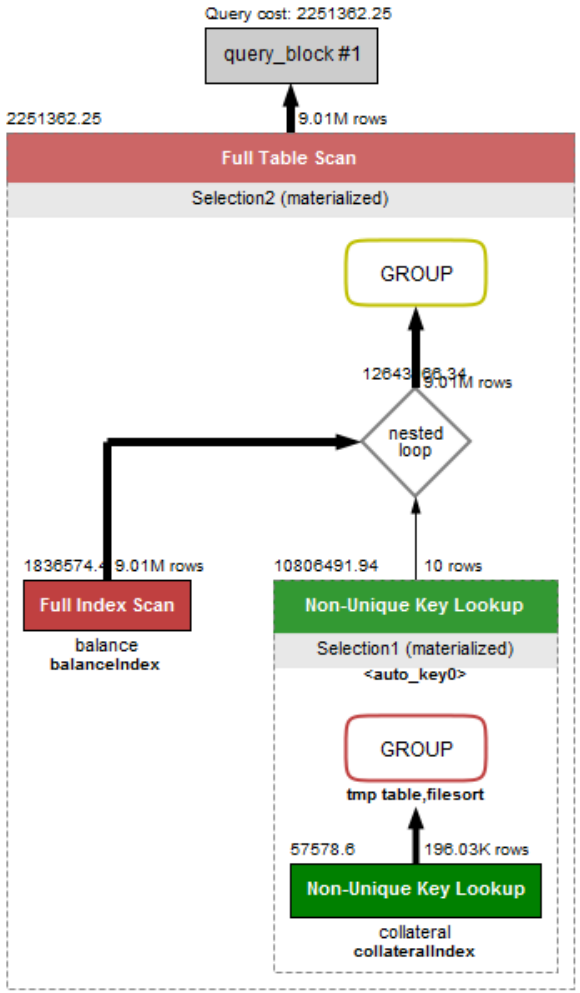
#	Time	Action	Message	Duration / Fetch
1	15:39:31	Select SQL_NO_CACHE co	1 row(s) re	61.484 sec / 0.000 sec
2	15:42:48	EXPLAIN Select SQL_NO_(	OK	0.000 sec

QUERY 1 - WORST - INDEX - NO UNIQUE



#	Time	Action	Message	Duration / Fetch
2	16:34:45	Select SQL_NO..	1 row(s) n	24.844 sec / 0.000 sec
1	16:34:49	EXPLAIN Selec..	OK	0.000 sec

QUERY 1-WORST-INDEX-UNIQUE(BALANCE)



10	17:17:34	Select SQL_NO_CA	1 row(s) ret	24.531 sec / 0.000 sec
11	17:18:00	EXPLAIN Select SQ	OK	0.000 sec

Εδώ δεν υπάρχουν πολλά σχόλια καθώς επαναλαμβόνται τα pattern που προαναφέρθηκαν. Το καλύτερο που πετύχαμε ήταν 20πλασιο κόστος ερωτηματος στο 45% του χρόνου. Το unique απλά μας γλιτώνει λιγο κόστος, αλλά δεν συγκρίνεται με το κόστος χωρις index.

Τελικά:
Best query:
+Καλύτερος χρόνος/κόστος ερωτήματος χωρίς index.
+Καλυτερη κλιμάκωση χρόνου με χρήση πρώτου και δεύτερου index.
+Καλύτερος χρόνος/κοστος ερωτήματος τελικού query.
-Ο optimizer δεν χρησιμοποίησε αμέσως το 2ο index.

Better query:
+Καλύτερο ποσοστό μείωσης χρόνου και κόστους τελικού query.
+Χρήση και των δυο index χωρίς ένδειξη στον optimizer.
+Οι χρόνοι ανταγωνίζονται του καλύτερου query.
-Κοστος οπτικα τεραστιο.
-Μεγάλος αριθμός σάρωσης δεδομένων.

Worst query:
-Ο χρόνος του τελικού query με index ειναι 5 δευτερα μικρότερος από του best query χωρίς index.

Από τις διάφορες υλοποιήσεις του πρώτου query ηδη μπορούμε να συμπεράνουμε μερικά πράγματα. Το κατασκευή του ερωτήματος έχει ιδιαίτερη σημασία. Αν κάποιος ξέρει να φτιάχνει τα σωστά queries μπορεί κάνει την διαφορά στα αν θα χρειαζόμαστε index ή όχι (βλεπε Worst query -). Αντίθέτως ένα μέτριο query μπορεί να βελτιστοιποιηθεί αρκετά (βλεπε better query) και τέλος ένα καλό query μπορεί να κλιμακωθεί και σταδιακά να πετύχει βέλτιστο χρόνο.

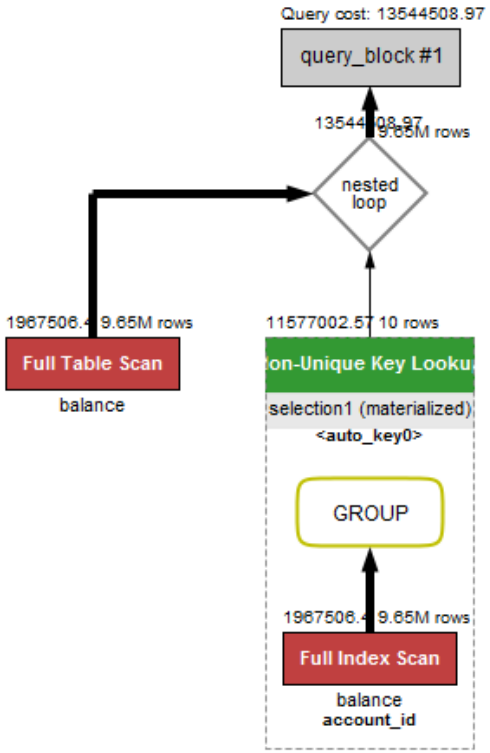
QUERY 2:

```
Select SQL_NO_CACHE avg(balance_value) from balance,
        (Select max(balance_date) maxDate,account_id from balance
        where balance_date - Date('2006-01-01') <= 0
        and balance_type = 'Capital'
        group by account_id) as selection1
where balance.balance_date = selection1.maxDate
and balance.account_id = selection1.account_id
and balance.balance_type = 'Capital';
```

Ακολουθεί πλάνο εκτέλεσης για query:

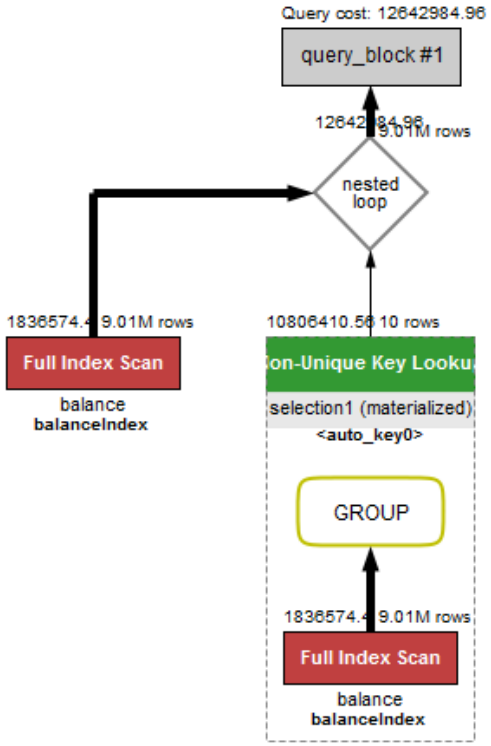
1)χωρίς index.
2)index οχι unique στον balance
3)index unique στον balance

QUERY 2 - NO INDEX



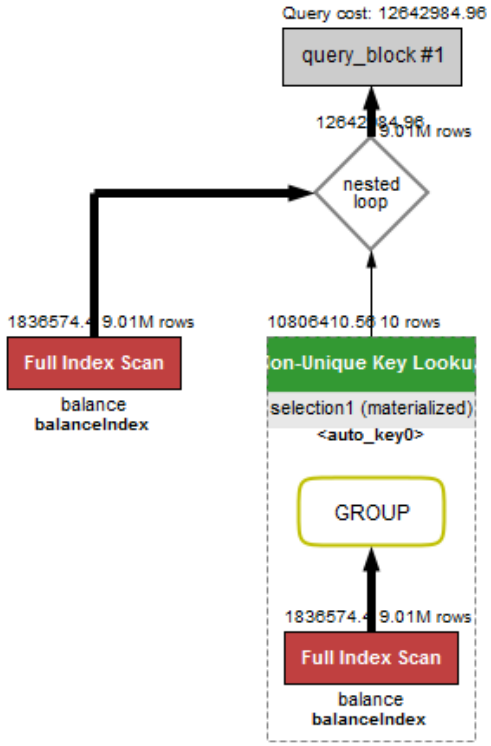
✓	4	15:43:39	Select SQL_NO_CACHE av	1 row(	41.969 sec / 0.000 sec
✓	5	15:45:17	EXPLAIN Select SQL_NO_(	OK	0.000 sec

QUERY 2 - INDEX - NO UNIQUE(BALANCE)



	#	Time	Action	Message	Duration / Fetch
✓	1	16:35:43	Select SQL_NO_CACHE avg(bal	1 row(s) ret	27.125 sec / 0.000
✓	2	16:36:13	EXPLAIN Select SQL_NO_CACH	OK	0.000 sec

QUERY 2 - INDEX - UNIQUE(BALANCE)



✓	13	17:18:49	Select SQL_NO_CACHE avg	1 row(	28.734 sec / 0.000 sec
✓	14	17:19:20	EXPLAIN Select SQL_NO_C	OK	0.000 sec

Αρχικά προφανές γίνεται ότι η χρήση του unique δεν άλλαξε τίποτα από όταν δεν το χρησιμοποιήσαμε. Η διαφορά έγινε στο ότι διαβάζουμε καποια εκατομύρια λιγότερα δεδομένα (βαση του κώδικα του πλάνου). Αυτό οφείλεται στην κατασκευή του index. Αρχικά επειδή χρησιμοποιείται μόνο το account_id (foreignkey) ως unclustered index θα υπάρχει ανάγνωση αρκετών δεδομένων κάτω από αυτό το κλειδί. Λόγο της μορφής του ερωτήματος το index θα μας κάνει απαλοιφή αρκετών αχρείαστων δεδομένων κατα το select. Τελικά γλιτώνουμε περίπου 1000000 μονάδες στο κόστος και το ερώτημα εκτελείται σε 14 δευτερόλεπτα λιγότερο.

QUERY 3-SIMPLE:

```
Select SQL_NO_CACHE count(distinct collateral_id) from
        (Select collateral_id,account.account_id as accnid from collateral use index
(collateralIndex),account,contract,customer
        where collateral_relation_type = '2'
        and collateral.contract_id = account.contract_id
   and account.contract_id = contract.contract_id
        and contract.customer_id = customer.customer_id
        and collateral_amount > 1000000
        and YEAR(curdate()) - YEAR(customer.birth_date) > 60) as Selection1,
   (Select max(balance_value) as maxbal, account_id from balance
        where balance_type = 'Capital'
   group by account_id) as Selection2
where Selection1.accnid = Selection2.account_id
and Selection2.maxbal <= 500000;
```

Ακολουθεί πλάνο εκτέλεσης για query:

1)χωρίς index.

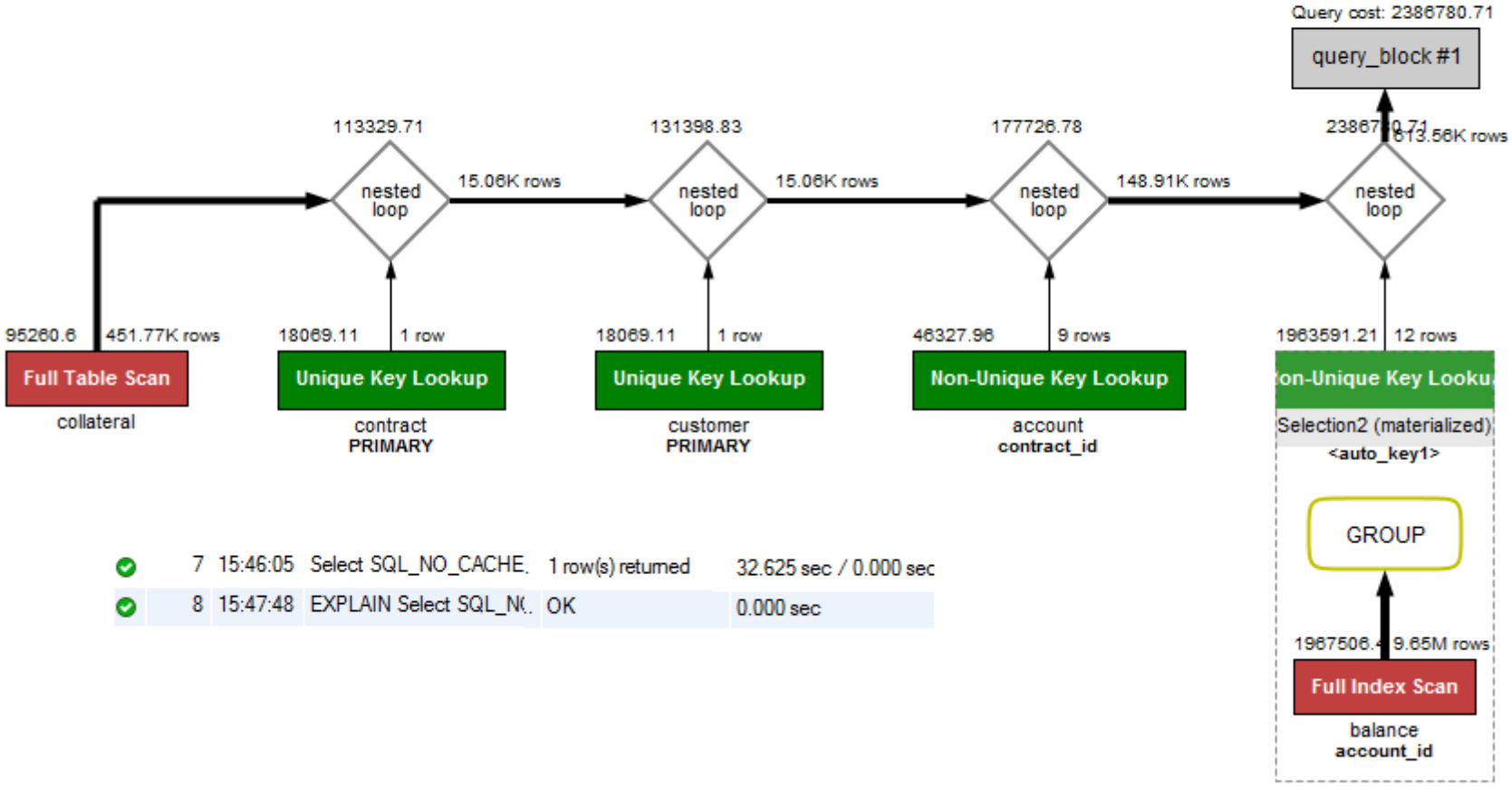
2)index οχι unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

3)index οχι unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

4)index unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

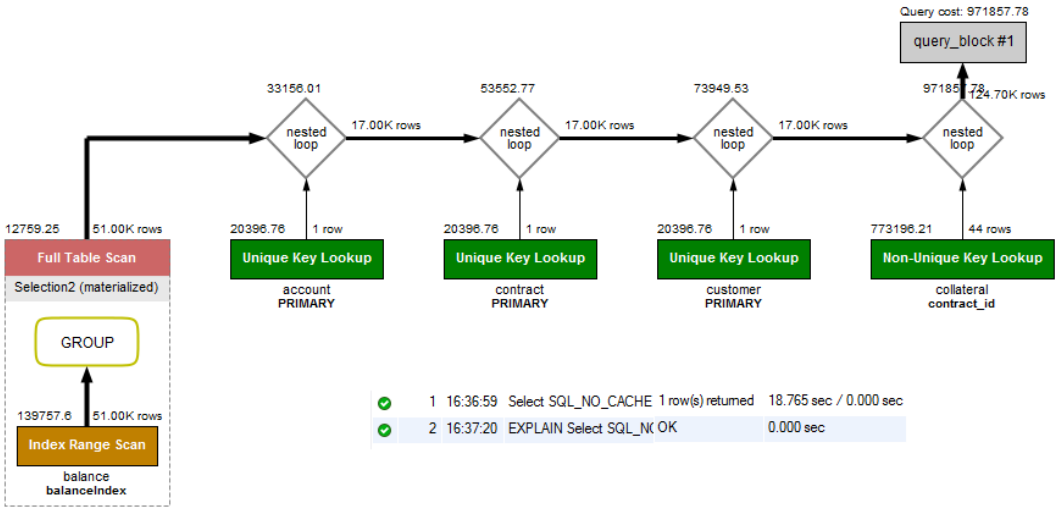
5)index unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

QUERY 3 - SIMPLE - NO INDEX



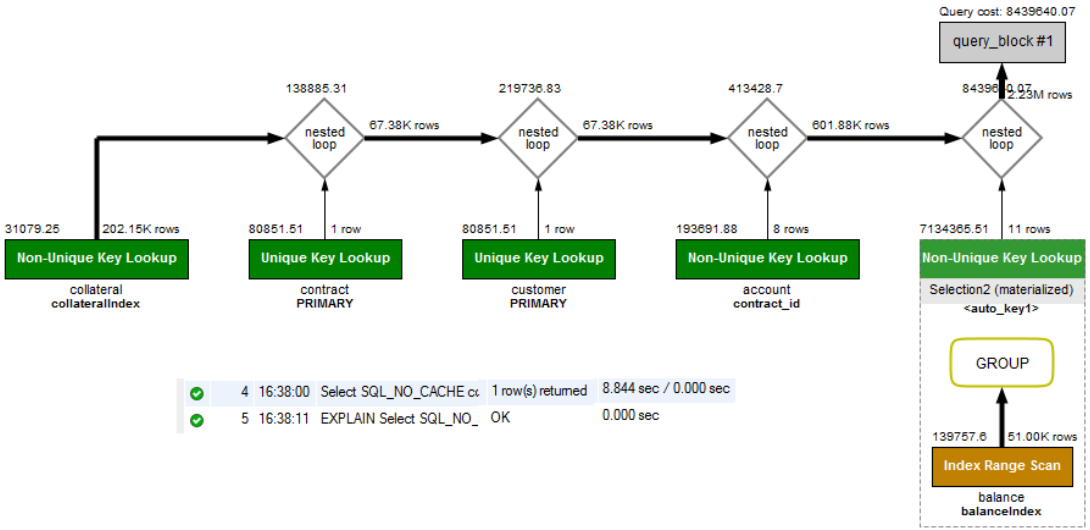
✓	7	15:46:05	Select SQL_NO_CACHE.	1 row(s) returned	32.625 sec / 0.000 sec
✓	8	15:47:48	EXPLAIN Select SQL_N(..	OK	0.000 sec

QUERY 3 - SIMPLE - INDEX - NO UNIQUE - NO ENF



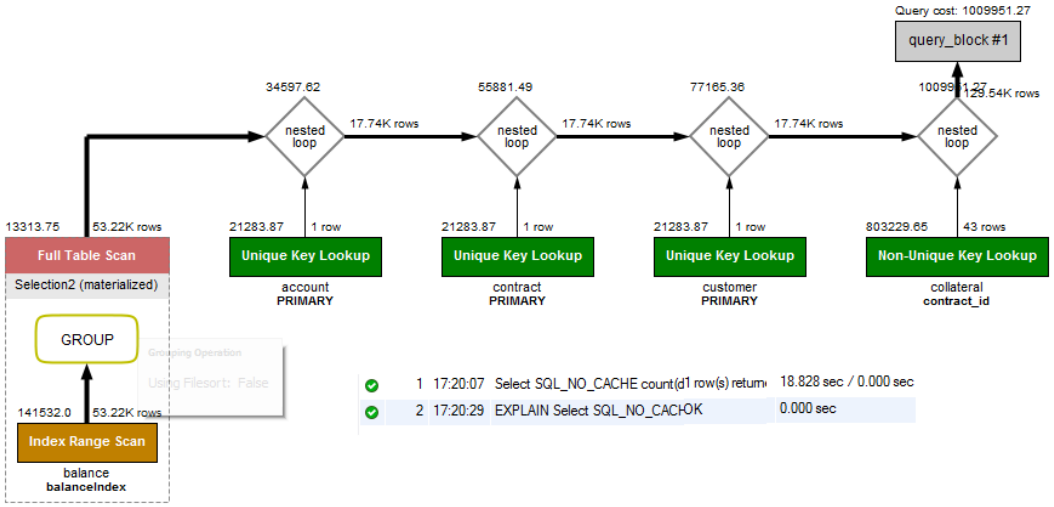
✓	1	16:36:59	Select SQL_NO_CACHE	1 row(s) returned	18.765 sec / 0.000 sec
✓	2	16:37:20	EXPLAIN Select SQL_N(	OK	0.000 sec

QUERY 3 - SIMPLE - INDEX - NO UNIQUE - ENF(COLLATERAL)



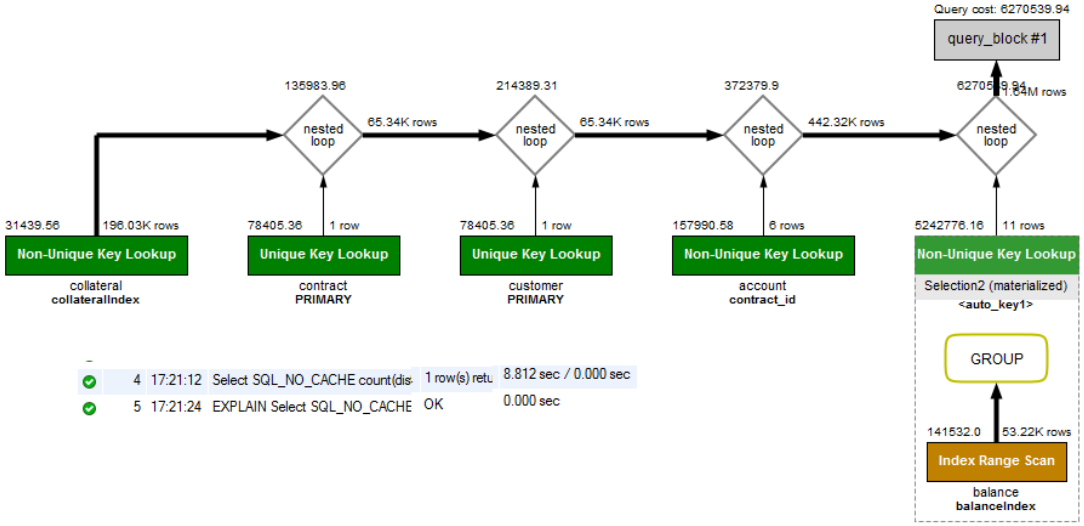
✓	4	16:38:00	Select SQL_NO_CACHE cc	1 row(s) returned	8.844 sec / 0.000 sec
✓	5	16:38:11	EXPLAIN Select SQL_NO_	OK	0.000 sec

QUERY 3 - SIMPLE - INDEX - UNIQUE(BALANCE) - NO ENF



✓	1	17:20:07	Select SQL_NO_CACHE count(d1 row(s) return	18.828 sec / 0.000 sec
✓	2	17:20:29	EXPLAIN Select SQL_NO_CACHOK	0.000 sec

QUERY 3 - SIMPLE - INDEX - UNIQUE(BALANCE) - ENF(COLLATERAL)



✓	4	17:21:12	Select SQL_NO_CACHE count(dis	1 row(s) retu	8.812 sec / 0.000 sec
✓	5	17:21:24	EXPLAIN Select SQL_NO_CACHE	OK	0.000 sec

Η απλή υλοποίηση του 3ου ερωτήματος.
Παλι παρατηρούμε την αγνόηση από την μεριά του optimizer για την χρήση του collateralIndex για την εμφάνιση καλύτερου κόστους. Στην τωρινή περίπτωση το κόστος του ερωτήματος έπεσε στο μισό συγκριτικά με το ερώτημα χωρίς index. Και ο χρόνος μειώθηκε κοντά στο μισό. Λέγοντας για άλλη μια φορά στον optimizer να χρησιμοποιήσει το collateralIndex βλέπουμε μείωση χρόνος -10 sec με 6πλασιο κόστος (έχοντας ως βέλτιστο αποτέλεσμα το UNIQUE(BALANCE)-(COLLATERAL)).
Η αιτια είναι ότι χωρίς το collateralIndex σαρώνονται λιγότερα δεδομένα ανα join (source execution plan).

QUERY 3-BONUS:

```
Select SQL_NO_CACHE count(collateral_id) from
        (Select collateral_id,collateral.contract_id as colconid from collateral,contract,customer
        where collateral_relation_type = '2'
        and collateral.contract_id = contract.contract_id
        and contract.customer_id = customer.customer_id
        and collateral_amount > 1000000
        and YEAR(curdate()) - YEAR(customer.birth_date) > 60) as Selection1,
        (Select sum(Selection3.maxbal) as sumcon,Selection3.acccon from
                (Select max(balance_value) as maxbal,account.contract_id as acccon from
balance,account
                where balance_type = 'Capital'
                and balance.account_id = account.account_id
                group by account.account_id) as Selection3
        group by Selection3.acccon) as Selection2
where Selection1.colconid = Selection2.acccon
and Selection2.sumcon <= 500000;
```

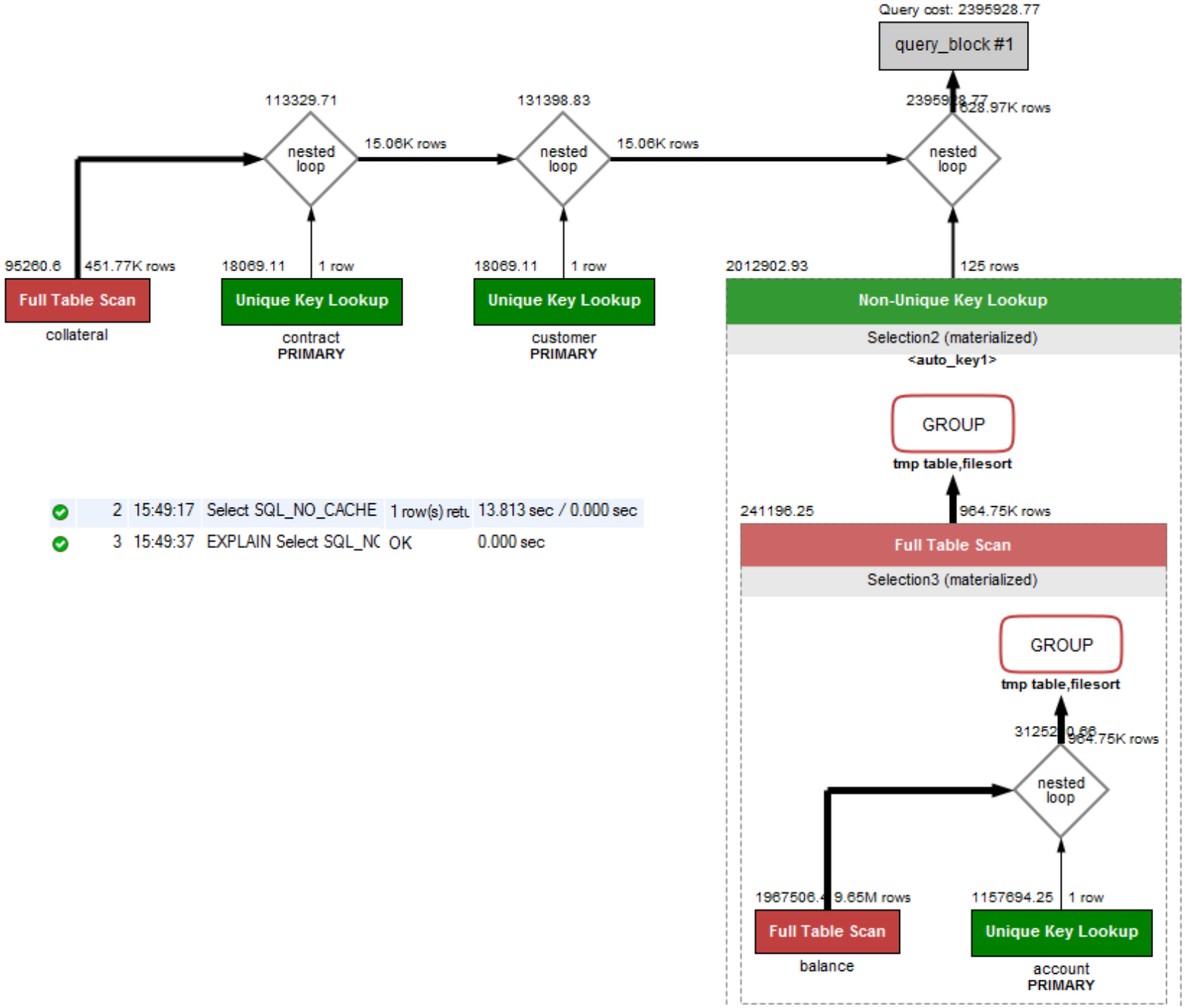
Ακολουθεί πλάνο εκτέλεσης για query:

1)χωρίς index.

2)index οχι unique στον balance

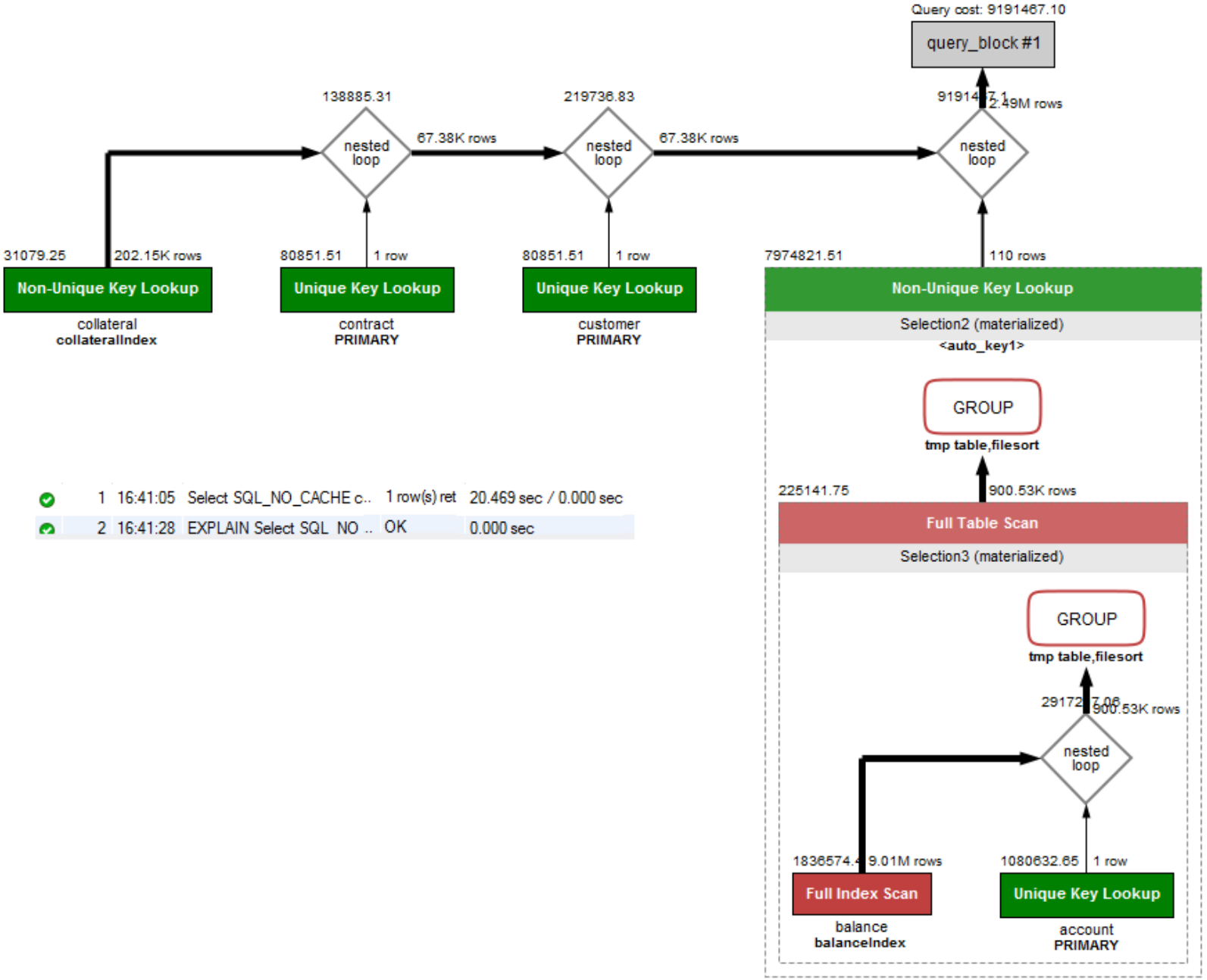
3)index unique στον balance

QUERY 3 - DIF - NO INDEX



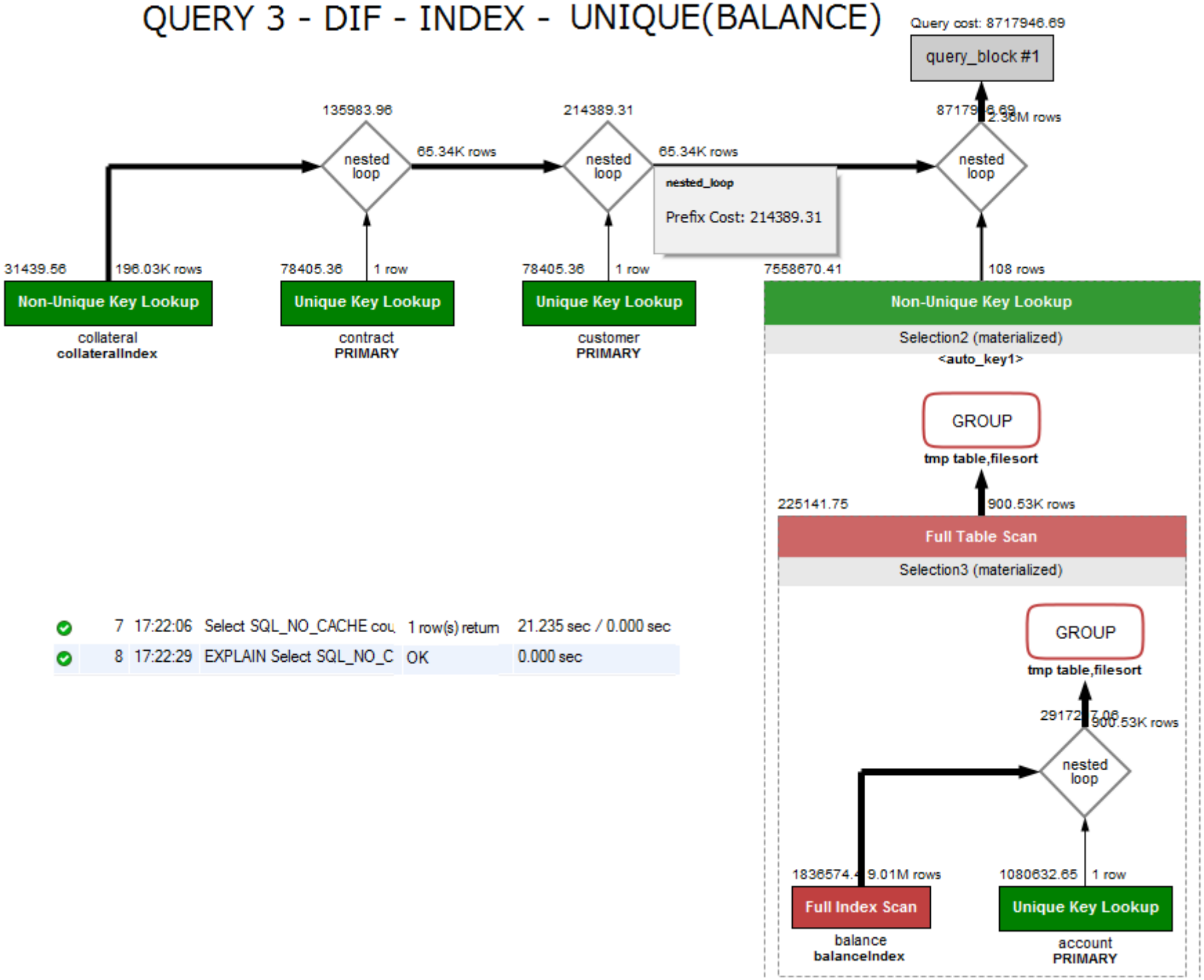
✓	2	15:49:17	Select SQL_NO_CACHE	1 row(s) retu	13.813 sec / 0.000 sec
✓	3	15:49:37	EXPLAIN Select SQL_NC	OK	0.000 sec

QUERY 3 - DIF - INDEX - NO UNIQUE



✔	1	16:41:05	Select SQL_NO_CACHE c..	1 row(s) ret	20.469 sec / 0.000 sec
✔	2	16:41:28	EXPLAIN Select SQL_NO ..	OK	0.000 sec

QUERY 3 - DIF - INDEX - UNIQUE(BALANCE)



✔	7	17:22:06	Select SQL_NO_CACHE cou	1 row(s) return	21.235 sec / 0.000 sec
✔	8	17:22:29	EXPLAIN Select SQL_NO_C	OK	0.000 sec

Εδώ πατατηρείται το αντίθετο από ότι περιμέναμε. Η χρήση των index καθυστερεί περισσότερο την εκτέλεση του ερωτήματος. Ενώ στην αρχή το query ανα join στον collateral διαβαζε 2M αυτό αυξήθηκε στα 10M με την χρήση index. Αυτο συμβαίνει και με τους υπόλοιπους πίνακες, με αποτέλεσμα να καταλήγουμε στο τελικό join με παραπάνω εγγραφές από ότι αρχικά. Η καθυστέρηση και το κόστος μπορεί να δηλώνει ότι είτε η κατασκευή του ερωτήματος προσεγγίστηκε λανθασμένα είτε η εφαρμογή των δεικτών δεν βγάζει το αποτέλεσμα που θα θέλαμε.

(δεν χρησιμοποιήθηκε ignore index καθώς θα αντικρουόταν με τα ζητούμενα της εργασίας. Υποθέτουμε ότι αν δοκιμάζαμε να κανουμε ignore καθε φορα ενα απο τους index να βλέπαμε άλλα αποτελέσματα).

QUERY 4:

```
Select SQL_NO_CACHE count(distinct account_id) from(
        Select max(balance_value) as balance_value,account_id from balance
    where balance_type = 'Interest'
        group by account_id) as selection1,
    (Select collateral.collateral_id,collateral.account_id as collaccid from collateral,real_estate
        where collateral_relation_type = '3'
        and collateral_type = '1'
        and collateral.collateral_id = real_estate.collateral_id
        and real_estate.appreciation_value > 100000) as selection2
where selection1.balance_value <= 10000
and selection1.account_id = selection2.collaccid;
```

Ακολουθεί πλάνο εκτέλεσης για query:

1)χωρίς index.

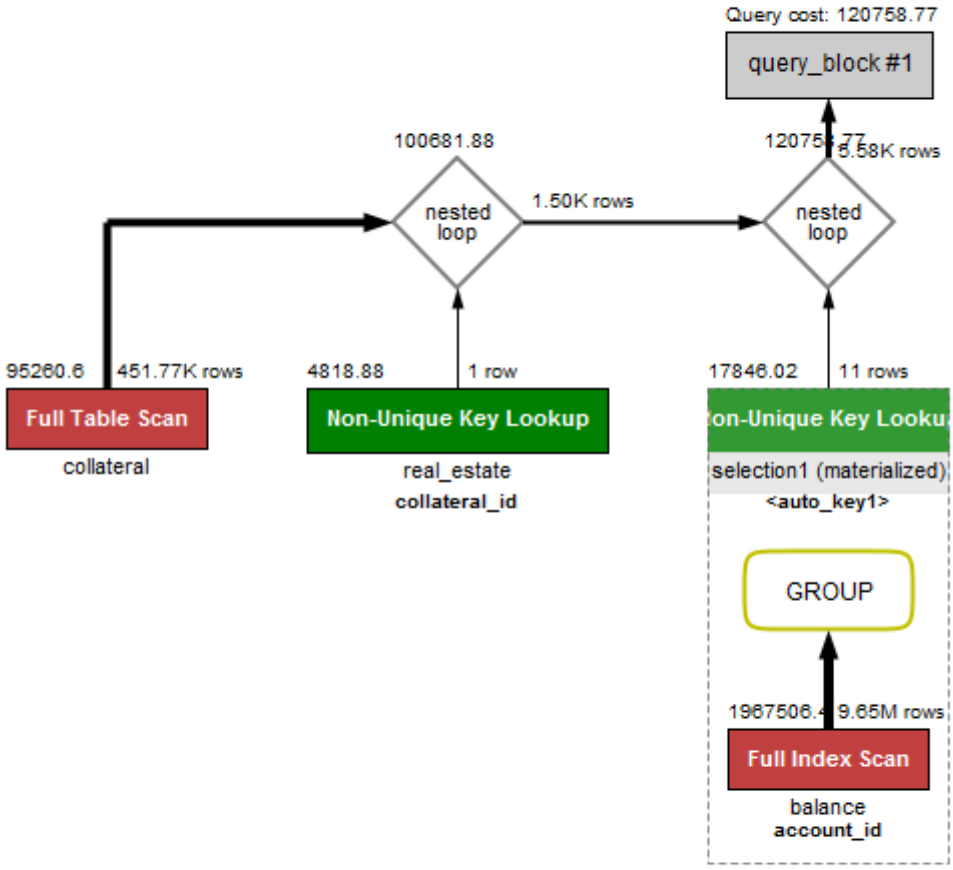
2)index οχι unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

3)index οχι unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

4)index unique στον balance και χωρίς ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

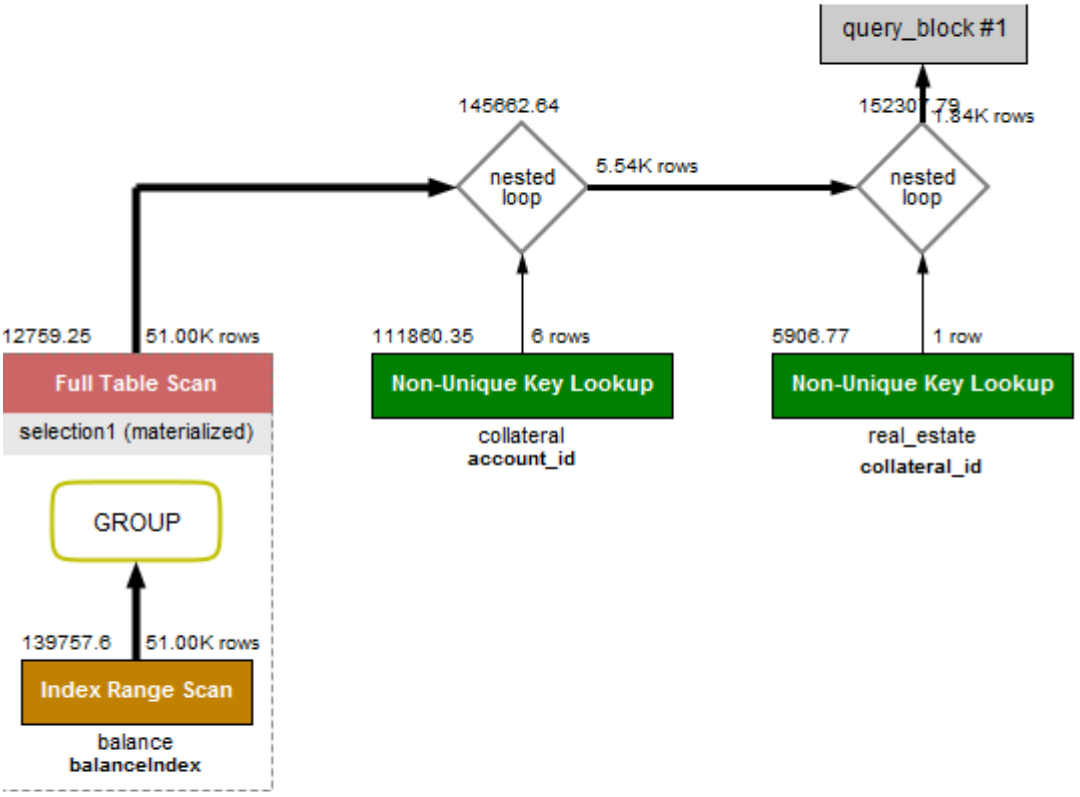
5)index unique στον balance με ένδειξη για τον optimiser να χρησιμοποιήσει index για collateral.

QUERY 4 - NO INDEX



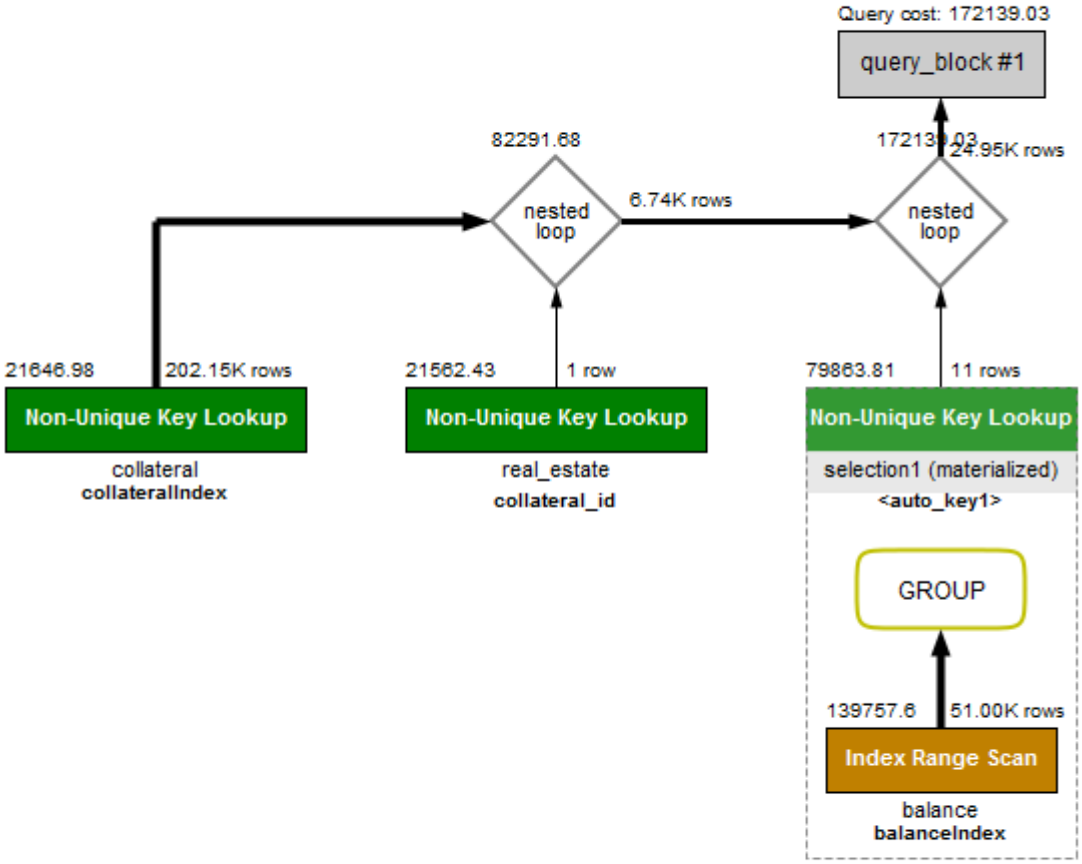
✓	5	15:50:51	Select SQL_NO_CACHE	1 row(	45.641 sec / 0.000 sec
✓	6	15:52:20	EXPLAIN Select SQL_N(	OK	0.000 sec

QUERY 4 - INDEX - NO UNIQUE - NO ENF



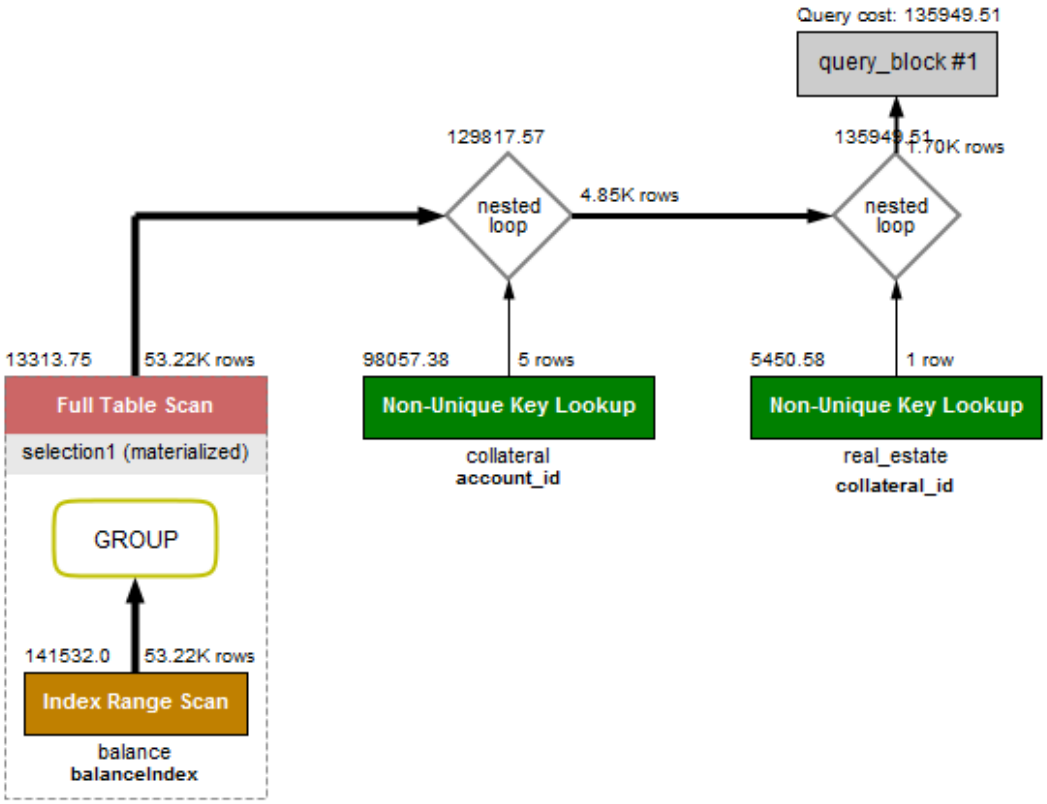
✔	9	16:43:55	Select SQL_NO_CACHE	1 row(s)	7.641 sec / 0.000 sec
✔	10	16:44:10	EXPLAIN Select SQL_N.	OK	0.000 sec

QUERY 4 - INDEX - NO UNIQUE - ENF(COLLATERAL)



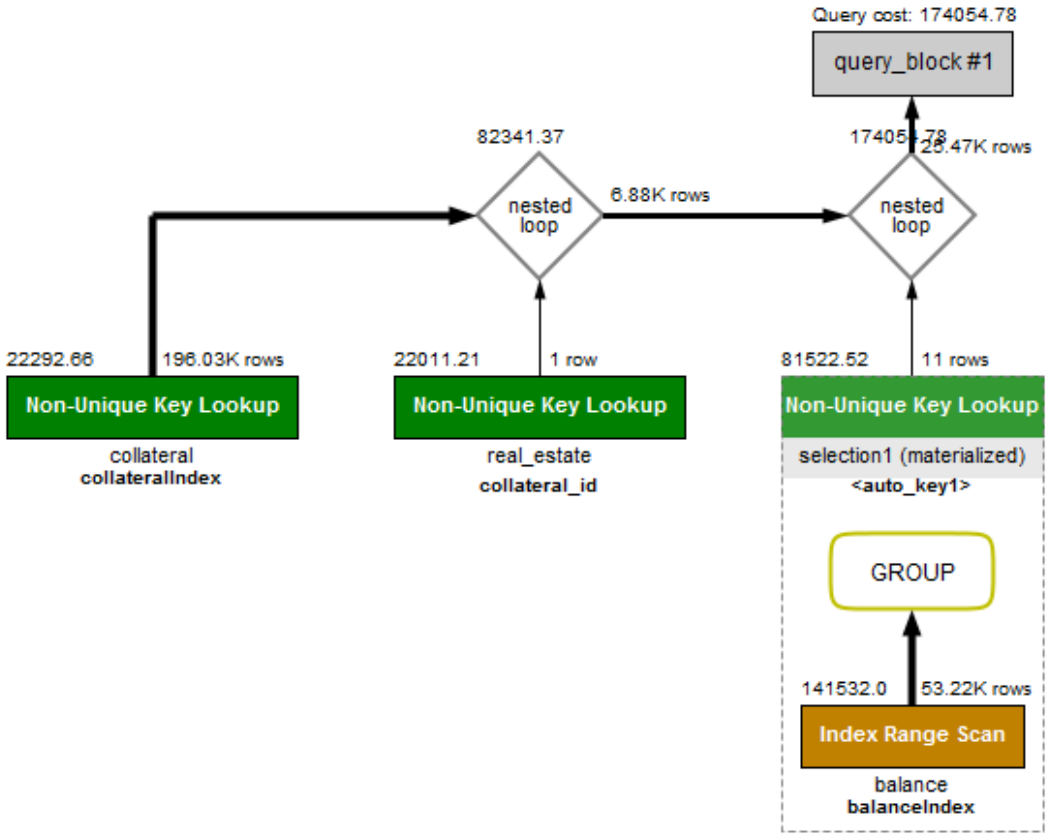
✔	14	16:45:11	Select SQL_NO_CACHE	1 row(s) returned	19.078 sec / 0.000 sec
✔	15	16:45:33	EXPLAIN Select SQL_NO	OK	0.000 sec

QUERY 4 - INDEX - UNIQUE(BALANCE) - NO ENF



✓	10	17:24:12	Select SQL_NO_CACH	1 row(s) returne	6.594 sec / 0.000 sec
✓	11	17:24:20	EXPLAIN Select SQL_I	OK	0.000 sec

QUERY 4 - INDEX - UNIQUE(BALANCE)-ENF(COLLATERAL)



✓	13	17:25:04	Select SQL_NO_CACHE	1 row(s	19.984 sec / 0.000 sec
✓	14	17:25:25	EXPLAIN Select SQL_NC	OK	0.000 sec

Στο τελευταίο query παρατηρούμε μια πιο εξισσοροπημένη κατάσταση όσο αφορά την σχέση κόστους και χρόνου. Όμως τώρα ο optimizer επέλεξε σωστά να μην χρησιμοποιήσει το collateralIndex καθως όχι μόνο επέφερε αύξηση του κόστους άλλά και του χρόνου εκτέλεσης τουλάχιστον 2πλασιου από ότι χωρίς να το χρησιμοποιήσουμε. Από σχήμα και από τον κώδικα φαίνεται ότι τα αποτελέσματα στο join διαφέρουν αρκετά στο ποσο αν συγκρίνουμε το NO ENF με το ENF(COLLATERAL). Η σχέση με το collateralIndex εξαναγκάζεται να πραγματοποιήσει join που τα περισσότερα μας είναι περιττα. Και στο τελικό join φαίνεται από το σχήμα το 1.84K vs 24.95K (για NO UNIQUE) και 1.70K vs 25.47K (UNIQUE(BALANCE).

Ακολουθεί πίνακας χρόνων.

For indexes times:
Query1 – best – index – Unique(balance) – enf(Collateral)
Query2 – index – Unique(balance)
Query3 – simple – index – Unique(balance) – enf(Collateral)
Query3 – dif – index – Unique(balance)
Query4 – index – Unique(balance)

Συνολικός χρόνος δημιουργίας ευρετηρίων	23.5 sec	
	Χωρίς ευρωτήρια	Με ευρετηρια
Χρόνος φόρτωσης νέων δεδομένων	682.25 sec	762.687 sec
Χρονος ερωτήματος 1	33.547 sec	9.36 sec
Χρόνος ερωτήματος 2	41.969 sec	28.734 sec
Χρόνος ερωτήματος 3 (Απλό)	32.625 sec	8.812 sec
Χρόνος ερωτήματος 3 (Bonus)	13.813 sec	21.235 sec
Χρόνος ερωτήματος 4	45.641 sec	6.594 sec

Με βάση τα παραπάνω δεδομένα η απάντηση στο ερώτημα αν συμφέρει ή όχι η δημιουργία index σε βάσεις είναι εξαρτάται. Προφανώς στην περίπτωση μας η απάντηση επιβάλλεται να είναι ότι μας συμφέρει παρατηρώντας τις βελτιώσεις στους χρόνους, αλλά υπάρχουν περιπτώσεις όπως το ερώτημα 1 και 3(Bonus). Εκεί κακή σχεδίαση ενός ερωτήματος μπορεί να μηδενίσει το κέρδος που μπορεί να επιφέρουν τα indexes όπως και η κακή επιλογή index μπορει να φέρει κόστος σε θέματα χρόνου.