Κεφάλαιο 13 Δοσοληψίες

Σύνοψη

Στο παρόν κεφάλαιο θα παρουσιασθούν βασικά στοιχεία όσον αφορά τις δοσοληψίες, την έννοια της σειριοποιησιμότητας και των διαφόρων επιπέδων απομόνωσης.

Προαπαιτούμενη γνώση

Η ύλη των προηγούμενων κεφαλαίων του βιβλίου.

13.1 Εισαγωγικές Έννοιες

Με τον όρο **δοσοληψία,** εννοούμε μια μονάδα εργασίας που μπορεί να έχει πρόσβαση σε διάφορα δεδομένα, διαβάζοντας ή ενημερώνοντάς τα. Είναι μια σειρά από ενέργειες, οι οποίες διαβάζουν ή γράφουν αντικείμενα της βάσης. Όταν εκτελείται μια δοσοληψία, η βάση δεδομένων μπορεί να αλλάζει. Με το πέρας της δοσοληψίας όμως, η βάση πρέπει να παραμένει σταθερή.

Δύο είναι τα βασικά προβλήματα με τις δοσοληψίες:

1. Τι θα γίνει αν κατά τη διάρκεια της εκτέλεσης, πέσει το σύστημα;
2. Τι θα γίνει αν δύο δοσοληψίες επιχειρούν να μεταβάλλουν το ίδιο αντικείμενο ταυτοχρόνως;

Για να διασφαλίσει την ακεραιότητα των δεδομένων το σύστημα βάσης δεδομένων πρέπει να εξασφαλίσει:

1. **Ατομικότητα**: είτε **όλες** οι πράξεις της δοσοληψίας επιτυγχάνουν, είτε **όλες** αποτυγχάνουν.
2. **Συνέπεια**: στο τέλος της δοσοληψίας, η βάση πρέπει να είναι σε συνεπή μορφή.
3. **Απομόνωση**: ακόμα κι αν τρέχουν πολλές δοσοληψίες ταυτόχρονα, **κάθε δοσοληψία πρέπει να νομίζει ότι τρέχει μόνη της.**
4. **Μονιμότητα**: αν η δοσοληψία επιτύχει, πρέπει το αποτέλεσμά της να επιβιώνει, ακόμα κι αν αποτύχει το σύστημα.

Το παραπάνω είναι διεθνώς γνωστό σαν **ACID test**

(A)tomicity	**Ατομικότητα**
(C)onsistency	**Συνέπεια**
(I)solation	**Απομόνωση**
(D)urability	**Μονιμότητα**

Καταστάσεις δοσοληψιών

- **Active:** στο ξεκίνημα και κατά τη διάρκειά της. Η δοσοληψία μένει σ' αυτή την κατάσταση ενώ εκτελείται
    - **Failed:** όταν το DBMS αντιληφθεί ότι η δοσοληψία δεν μπορεί να συνεχίσει
- **Aborted:** όταν η αποτυχημένη δοσοληψία έχει αναιρεθεί από το σύστημα και η ΒΔ είναι σε συνεπή μορφή

- **Committed:** όταν η δοσοληψία επιτύχει και η ΒΔ είναι σε συνεπή μορφή.
- **Partially committed:** όταν έχει εκτελεστεί η τελευταία εντολή της δοσοληψίας

Συμπεραίνουμε λοιπόν, ότι η δοσοληψία τελειώνει είτε με μια δήλωση **COMMIT** που ολοκληρώνει επιτυχώς τη δοσοληψία, μονιμοποιώντας τις αλλαγές στη βάση δεδομένων, είτε με δήλωση **ABORT (ROLLBACK για την MySQL)**.

Σειριοποιησιμότητα

Βασικό ζητούμενο είναι σε κάθε δοσοληψία να διατηρείται η συνέπεια της βάσης. Η σειριακή εκτέλεση των δοσοληψιών (σειριακό χρονοπρόγραμμα) εγγυάται τη συνέπεια της βάσης.

Σειριοποιήσιμο είναι το χρονοπρόγραμμα που εγγυημένα έχει το ίδιο αποτέλεσμα με ένα πλήρες σειριακό χρονοπρόγραμμα. Έτσι, αν μας δοθεί ένα χρονοπρόγραμμα, πρέπει να μπορέσουμε να αποφανθούμε αν είναι σειριοποιήσιμο ή όχι και αυτό μπορούμε να το πετύχουμε με δυο τεχνικές: την **σειριοποιησιμότητα συγκρούσεων** και την **σειριοποιησιμότητα όψεως.** Στο εξής θα αγνοούμε το processing στη μνήμη και θα μας απασχολούν μόνο οι read και write αλληλεπιδράσεις με τη βάση δεδομένων. Εδώ υποθέτουμε ότι οι δοσοληψίες μπορούν να εκτελέσουν τους αυθαίρετους υπολογισμούς τους στα δεδομένα σε τοπικούς buffer ανάμεσα σε read και write.

Σειριοποιησιμότητα συγκρούσεων

Έστω η εντολή I_1 της δοσοληψίας T_1 και η εντολή I_2 της T_2. Τότε αυτές συγκρούονται αν και μόνο αν και οι δυο έχουν πρόσβαση σε ένα αντικείμενο Q και τουλάχιστον μια απ' αυτές γράφει το Q.

Παράδειγμα

1. I_1 = **read** (Q) , I_2 = **read** (Q) Δεν Συγκρούονται
2. I_1 = **read** (Q) , I_2 = **write** (Q) Συγκρούονται
3. I_1 = **write** (Q) , I_2 = **read** (Q) Συγκρούονται
4. I_1 = **write** (Q) , I_2 = **read** (Q) Δεν Συγκρούονται

Αν ένα χρονοπρόγραμμα S μετασχηματιστεί σε ένα χρονοπρόγραμμα S' από μια σειρά εναλλασσόμενων - μη συγκρουόμενων εντολών, λέμε ότι τα S και S' είναι **ισοδύναμα συγκρούσεων.** Δηλαδή, δύο χρονοπρογράμματα είναι ισοδύναμα συγκρούσεων, αν για κάθε σύγκρουση οι συγκρουόμενες εντολές έχουν την ίδια σειρά στα δύο προγράμματα.

Ένα χρονοπρόγραμμα είναι **σειριοποιήσιμο συγκρούσεων,** αν είναι ισοδύναμο συγκρούσεων με ένα σειριακό (δηλαδή, αν όλες οι συγκρουόμενες πράξεις έχουν την ίδια σειρά που θα είχαν σε ένα σειριακό).

Για την καλύτερη κατανόηση των παραπάνω, σημειώνουμε τυπικά ότι:

- στο σειριακό χρονοπρόγραμμα, κάθε δοσοληψία «ξεμπερδεύει» ξεχωριστά (σε απομόνωση) με κάθε αντικείμενο και μετά το «αναλαμβάνει» μια άλλη.
- σε ένα σειριοποιήσιμο, το να ΜΗΝ υπάρχει σύγκρουση σημαίνει ότι το χρονοπρόγραμμα «ξεμπερδεύει» με τα αντικείμενα με την ίδια σειρά ανά δοσοληψία, με την οποία θα το έκανε και το σειριακό.

Σειριοποιησιμότητα όψεων

Έστω ότι S και S΄ είναι δυο χρονοπρογράμματα με το ίδιο set δοσοληψιών. Τα S και S΄ λέγονται **ισοδύναμα όψης** αν συμπίπτουν οι παρακάτω τρεις συνθήκες:

- Για κάθε δεδομένο αντικειμένου Q, αν η δοσοληψία T_1 διαβάζει την αρχική τιμή του Q στο χρονοπρόγραμμα S, τότε η δοσοληψία Ti πρέπει επίσης να διαβάζει την αρχική τιμή του Q στο χρονοπρόγραμμα S΄.
- Για κάθε δεδομένο αντικειμένου Q, αν η δοσοληψία T_1 εκτελέσει την **read**(*Q)* στο χρονοπρόγραμμα S και αυτή η τιμή παράχθηκε από τη δοσοληψία T_1, τότε πρέπει στο χρονοπρόγραμμα S΄ η δοσοληψία T_1 επίσης να διαβάζει την τιμή του Q που επίσης παράχθηκε από τη δοσοληψία T_1.
- Για κάθε δεδομένο αντικειμένου Q, η δοσοληψία που εκτελεί την τελική λειτουργία **write**(Q) στο χρονοπρόγραμμα *S* πρέπει να εκτελεί και την τελική **write**(*Q*) λειτουργία στο χρονοπρόγραμμα *S* ‘.

Ένα χρονοπρόγραμμα S είναι **σειριοποιήσιμο όψης** όταν είναι ισοδύναμο όψης με ένα σειριακό χρονοπρόγραμμα. Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, είναι σειριοποιήσιμο όψεως . Το αντίστροφο όμως δεν ισχύει.

Το παρακάτω χρονοπρόγραμμα είναι σειριοποιήσιμο όψης αλλά όχι σειριοποιήσιμο σύγκρουσης (ισοδύναμο όψης σε σειριακό χρονοπρόγραμμα T3, T4, T6)

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο όψεως, και ΔΕΝ είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, περιέχει **τυφλές εγγραφές** (writes που δεν έχει προηγηθεί read γι’ αυτές στην δοσοληψία τους).

Παραδείγματα με MySQL

Αρχικά δημιουργήστε ένα πίνακα στη μηχανή αποθήκευσης InnoDB με όνομα T1 και γνωρίσματα L1, K1 που δείχνουν το pin λογαριασμού και το υπόλοιπο του κάθε λογαριασμού αντιστοίχως.

Απάντηση:

```
CREATE TABLE T1 (
        L1 INT NOT NULL,
        K1 INT,
PRIMARY KEY (L1)
)ENGINE=InnoDB;
```

και ας εισάγουμε σ΄ αυτόν τις παρακάτω εγγραφές :

```
INSERT INTO T1 VALUES(006, 600);
INSERT INTO T1 VALUES(007, 900);
INSERT INTO T1 VALUES(008, 1000);
INSERT INTO T1 VALUES(009, 1100);
```

Δίνοντας τώρα την εντολή:

```
SELECT * FROM T1;
```

βλέπουμε για το table T1 τον παρακάτω πίνακα που δείχνει πως έχουν μέχρι τώρα οι εγγραφές στο T1:

L1	K1
006	600
007	900
008	1000
009	1100

Το πρότυπο ISO καθορίζει ότι μια δοσοληψία της SQL ξεκινάει αυτόματα με μια δήλωση **εκκίνησης-δοσοληψίας** η οποία εκτελείται αυτόματα από ένα χρήστη ή ένα πρόγραμμα (πχ. από τις δηλώσεις **SELECT, INSERT, UPDATE**). Επίσης οι αλλαγές που γίνονται από μια δοσοληψία δεν είναι ορατές από άλλες δοσοληψίες που εκτελούνται ταυτόχρονα μέχρι την ολοκλήρωσή της.

Ας πειραματιστούμε τώρα με μια μόνο transaction:

Παράδειγμα

Ο κάτοχος του λογαριασμού με pin 009 έκανε κατάθεση και πλέον το υπόλοιπο του λογαριασμό του αντιστοιχεί σε 10000 ευρώ. Υλοποιήστε την ενημέρωση

Απάντηση:

```
START TRANSACTION;
UPDATE T1
SET K1=10000
WHERE K1=1100;
```

Με την παραπάνω πράξη ενημέρωσης της T1 αντικαθιστούμε την τιμή 1100 της k1 με την τιμή 10000. Αυτό φαίνεται από τον παρακάτω πίνακα που προκύπτει αν δώσουμε πάλι μια select μέσα στην transaction.

SELECT * **FROM** T1;

L1	K1
006	600
007	900
008	1000
009	10000

Όμως μια δοσοληψία μπορεί να ολοκληρωθεί με έναν από τους ακόλουθους τρόπους :

- Με μια δήλωση **COMMIT** που ολοκληρώνει επιτυχώς τη δοσοληψία, μονιμοποιώντας τις αλλαγές στη βάση δεδομένων. Μια νέα δοσοληψία ξεκινάει μετά τη δήλωση COMMIT με την επόμενη δήλωση εκκίνησης –δοσοληψίας
- Με μια δήλωση **ROLLBACK** η οποία ακυρώνει τη δοσοληψία, διαγράφοντας κάθε αλλαγή που έχει γίνει από τη δοσοληψία. Μια νέα δοσοληψία ξεκινάει μετά τη δήλωση ROLLBACK με την επόμενη δήλωση εκκίνησης –δοσοληψίας
- Για την προγραμματιζόμενη SQL, η επιτυχής ολοκλήρωση ενός προγράμματος ολοκληρώνει την τελική δοσοληψία με επιτυχία, ακόμη και αν η δήλωση COMMIT δεν έχει εκτελεστεί.
- Για την προγραμματιζόμενη SQL, η μη ομαλή ολοκλήρωση του προγράμματος ακυρώνει τη δοσοληψία.

Αν δηλαδή στη συνέχεια δώσουμε την εντολή ROLLBACK η εκτέλεση της δοσοληψίας θα ακυρωθεί και οι αλλαγές που έχουν γίνει απ’ αυτήν δεν θα αποθηκευτούν στη βάση. Ενώ αν δώσουμε COMMIT θα δούμε ότι οι αλλαγές από τη δοσοληψία θα μονιμοποιηθούν στη βάση δεδομένων και η T1 θα κλείσει επιτυχώς.

Παράδειγμα

Ο κάτοχος του λογαριασμού με pin κάρτας 007 βρίσκεται σε ένα τραπεζικό κατάστημα και προσπαθεί να κάνει κατάθεση μεταβάλλοντας το υπόλοιπο σε 10500ευρώ ενώ την ίδια στιγμή ο γιος του προσπαθεί να κάνει ανάληψη από ένα ATM και να μεταβάλλει τον ίδιο λογαριασμό από 900 σε 400 ευρώ. Υλοποιήστε την παραπάνω συναλλαγή

Απάντηση:

Ξεκινάμε την πρώτη δοσοληψία

START TRANSACTION;

UPDATE T1

SET K1=10500

WHERE K1=900;

Και συνεχίζουμε με την δεύτερη

START TRANSACTION;

(Αν εκτελέσουμε ως εδώ ένα Select * from T1; βλέπουμε ότι οι αλλαγές που έκανε η πρώτη transaction δεν είναι ορατές στη δεύτερη).

UPDATE T1

SET K1=10800

WHERE K1=900;

Ας δούμε τώρα πάλι με ένα Select * from T1; αν η δεύτερη δοσοληψία πραγματοποιήθηκε.

Το αποτέλεσμα που πήραμε φαίνεται στον παρακάτω στον πίνακα όπου διαπιστώνουμε ότι η **δεύτερη** δοσοληψία **δεν** έχει πραγματοποιηθεί!

L1	K1
006	600
007	10500
008	1000
009	10000

Αυτό οφείλεται σε μια τεχνική ελέγχου του ταυτοχρονισμού που ονομάζεται **κλείδωμα.** Η βασική ιδέα είναι απλή : όταν μια συναλλαγή χρειάζεται διαβεβαίωση ότι κάποιο αντικείμενο για το οποίο ενδιαφέρεται δε θα μεταβληθεί με κάποιον απρόβλεπτο τρόπο ενώ "η συναλλαγή έχει γυρισμένη την πλάτη της", τότε η συναλλαγή **αποκτά ένα κλείδωμα** σε αυτό το αντικείμενο. Το αποτέλεσμα του κλειδώματος είναι να αποκλειστούν οι άλλες συναλλαγές από το αντικείμενο, και έτσι να μην μπορούν να το μεταβάλουν. Η πρώτη συναλλαγή έχει έτσι τη δυνατότητα να πραγματοποιήσει την επεξεργασία της, γνωρίζοντας με βεβαιότητα ότι το δεδομένο αντικείμενο θα παραμείνει σε σταθερή κατάσταση για όσο χρόνο επιθυμεί η συναλλαγή.

Για να εκτελεστεί η δεύτερη transaction πρέπει να κλείσει πρώτα η α συναλλαγή με μια δήλωση COMMIT ή ROLLBACK. Τότε θα δούμε ότι η δεύτερη ξεμπλοκάρει και αρχίζει να εκτελείται και η ίδια. Η βάση δεδομένων την κάνει αυτόματα ROLLBACK(όπως αναφέρθηκε παραπάνω).

Γνωρίζοντας όμως τα παραπάνω προσπαθήστε να υλοποιήσετε και την παρακάτω δοσοληψία:

Παράδειγμα

Δυο άτομα την ίδια χρονική στιγμή προσπαθούν να κάνουν μια κατάθεση στο λογαριασμό με αριθμό pin 006 ο ένας μετατρέποντάς τον σε 10.700 ευρώ και ο άλλος στον λογαριασμό 007 μετατρέποντάς τον σε 11.500 ευρώ.

Απάντηση:

Εισάγουμε την πρώτη δοσοληψία
START TRANSACTION;
UPDATE T1
SET K1=10700
WHERE K1=600;
Εκτελούμε μια **COMMIT;** για να κλείσει η TR1 και να μονιμοποιηθούν οι αλλαγές στη βάση δεδομένων.
Κάνουμε εδώ μια
SELECT * **FROM** T1;
για να δούμε την κατάσταση που βρίσκεται το table T1

L1	K1
006	10700
007	10500
008	1000
009	10000

Ξεκινάμε τώρα την TR2
START TRANSACTION;
UPDATE T1
SET K1=11500
WHERE K1=10500;

Κάνοντας εδώ μια SELECT βλέπουμε όπως φαίνεται από τον πίνακα ότι πλέον η ενημέρωση στη βάση από την TR2 έχει γίνει:

L1	K1
006	11500
007	10500
008	1000
009	10000

Για να κλείσουμε τη δοσοληψία εισάγουμε μια δήλωση COMMIT ή ROLLBACK ανάλογα με το αν θέλουμε να διατηρήσουμε τις αλλαγές από την τρέχουσα δοσοληψία ή όχι. Ας προσέξουμε ότι οι παραπάνω δοσοληψίες υλοποιούνται στο K1 που είναι γνώρισμα μη πρωτεύοντος κλειδιού. Προσπαθήστε τώρα να πραγματοποιήσετε ενημέρωση σε transaction σε γνώρισμα πρωτεύοντος κλειδιού.

Παράδειγμα

Οι κάτοχοι των καρτών με pin 006 και 007 έχασαν την κάρτα τους και μόλις το αντιλήφθησαν έτρεξαν να αλλάξουν κωδικό. Ο ένας το μετατρέπει σε 001 ενώ εκείνος που είχε το 007 το μετατρέπει σε 002.

Απάντηση :

START TRANSACTION;
UPDATE T1
SET L1=001
WHERE L1=006;
Εκτελώντας τη δήλωση SELECT * FROM T1; βεβαιωνόμαστε ότι οι αλλαγές έχουν γίνει

L1	K1
001	11500
007	10500
008	1000
009	10000

Προχωράμε στην εκτέλεση της δεύτερης δοσοληψίας

START TRANSACTION;
UPDATE T1
SET L1=002
WHERE L1=007;
Ας εκτελέσουμε μια SELECT * FROM T1;

Το αποτέλεσμα που προκύπτει είναι ο παρακάτω πίνακας 6.7 όπου και διαπιστώνουμε ότι οι αλλαγές από την TR2 έχουν γίνει

L1	K1
001	11500
002	10500
008	1000
009	10000

Οι δυο δοσοληψίες εκτελούνται ταυτόχρονα όταν τα γνωρίσματα που ενημερώνουμε αποτελούν πρωτεύοντα κλειδιά. Εδώ **αποκτούν κλείδωμα μόνο οι συγκεκριμένες εγγραφές** ενώ στην περίπτωση που δεν δρούμε σε πρωτεύον κλειδί αποκτά κλείδωμα όλος ο πίνακας.

Πρόβλημα αδιεξόδου (deadlock)

Παράδειγμα

Δημιουργήστε έναν πίνακα ο οποίος θα περιέχει δυο γνωρίσματα, το ένα από τα δυο θα αντιπροσωπεύει τον κωδικό πτήσης μιας συγκεκριμένης αεροπορικής εταιρίας και το άλλο την ώρα αναχώρησης της συγκεκριμένης πτήσης.

Απάντηση :

```
CREATE TABLE F1 (
        Id INT ,
        hour  TIME,
PRIMARY KEY (Id));
```

Ας εισάγουμε σ' αυτόν μερικές τιμές

```
INSERT INTO T1 VALUES(533, 645);

INSERT INTO T1 VALUES(740, 2005);

INSERT INTO T1 VALUES(927, 615);

INSERT INTO T1 VALUES(348, 1700);
```

Παράδειγμα

Λόγω άσχημων καιρικών συνθηκών έχουμε ακύρωση των πτήσεων με id=740, id=348. Οι παραπάνω πτήσεις αντικαθίστανται η μια από την 632 με ώρα αναχώρησης 21:45 και η άλλη από την πτήση 283 και ώρα αναχώρησης 19:00 αντιστοίχως. Προσπαθήστε να υλοποιήσετε τις παραπάνω συναλλαγές.

Απάντηση:

Ξεκινάμε με τη δημιουργία της πρώτης συναλλαγής

```
START TRANSACTION;
UPDATE F1
SET Id=632
WHERE Id=740;
UPDATE F1
SET hour=2145
WHERE hour=2005;
```

Εκτελούμε και την δεύτερη

```
START TRANSACTION;
UPDATE F1
SET hour=1900
WHERE hour=1700;
```

```
UPDATE F1
SET  Id=283
WHERE Id=348;
```

Σ' αυτή την περίπτωση πέφτουμε σε ένα αδιέξοδο γιατί κάθε μια από τις δοσοληψίες περιμένει την άλλη να απελευθερώσει ένα κλείδωμα για να μπορέσει να προχωρήσει.

Το αδιέξοδο (deadlock) είναι μια κατάσταση όπου δυο ή περισσότερες συναλλαγές είναι **ταυτόχρονα** σε κατάσταση αναμονής, και η κάθε μια περιμένει μια από τις άλλες να απελευθερώσει ένα κλείδωμα, πριν μπορέσει να προχωρήσει. Αν συμβεί ένα αδιέξοδο είναι επιθυμητό το σύστημα να το εντοπίσει και να το σπάσει. Ο εντοπισμός του αδιεξόδου σημαίνει τον εντοπισμό ενός κύκλου στο **γράφημα αναμονών,** δηλαδή στο γράφημα που δείχνει "ποιος περιμένει ποιον". Σπάσιμο του αδιεξόδου σημαίνει να διαλέξουμε για **θύμα** μια από τις συναλλαγές που βρίσκονται σε αδιέξοδο και να την ανακατασκευάσουμε απελευθερώνοντας έτσι τα κλειδώματά της και επιτρέποντας να προχωρήσει σε κάποια άλλη συναλλαγή. Έτσι εδώ η MySQL εντοπίζει τα deadlock και από μόνη της κάνει rollback τις δυο δοσοληψίες.

Επίπεδο απομόνωσης

Ο όρος **επίπεδο απομόνωσης** χρησιμοποιείται για να υποδηλώσει αυτό που λέμε βαθμό παρεμβολής τον οποίο μπορεί να ανεχθεί μια δεδομένη συναλλαγή όσον αφορά τις ταυτόχρονες συναλλαγές. Στην πραγματικότητα, δεν υπάρχει κανένας λόγος να μη λειτουργεί μια συναλλαγή σε διαφορετικά επίπεδα απομόνωσης την ίδια στιγμή, σε διαφορετικά μέρη της βάσης δεδομένων.

Μια ειδική εντολή που ονομάζεται **SET TRANSACTION** χρησιμοποιείται για να ορίσει κάποια χαρακτηριστικά της νέας συναλλαγής που θα ξεκινήσει. Χαρακτηριστικά αυτής της εντολής είναι ο **τρόπος προσπέλασης** και το ***επίπεδο απομόνωσης.*** Η σύνταξη της εντολής είναι:

SET [GLOBAL | SESSION] **TRANSACTION** [READ ONLY | READ WRITE] ISOLATION LEVEL

{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

Οι προτάσεις READ ONLY και READ WRITE υποδεικνύουν αν μια συναλλαγή είναι μόνο για ανάγνωση ή αν περιλαμβάνει και τις δυο πράξεις. Η προκαθορισμένη επιλογή είναι η READ WRITE αν δεν καθορίζεται καμία. Η δήλωση READ ONLY επιτρέπει σε μια δοσοληψία να εκτελέσει τις δηλώσεις INSERT, UPDATE και DELETE σε προσωρινούς και μόνο πίνακες.

Το επίπεδο απομόνωσης υποδεικνύει το βαθμό αλληλεπίδρασης που επιτρέπεται από άλλες δοσοληψίες κατά την εκτέλεση της συγκεκριμένης δοσοληψίας. Τα πιθανά επίπεδα απομόνωσης είναι τα:

1. **READ UNCOMMITTED** (ανάγνωση ανεπικύρωτων): επιτρέπει σε μια δοσοληψία να βλέπει ενδιάμεσα αποτελέσματα άλλων δοσοληψιών, πριν αυτές ολοκληρωθούν. Εδώ εισάγεται το πρόβλημα της **αναξιόπιστης ανάγνωσης** δηλ μια δοσοληψία μπορεί να διαβάσει μια γραμμή δοσοληψίας που μόλις έχει ενημερωθεί ενώ μετά αυτή κάνει ROLLBACK
2. **READ COMMITTED** (ανάγνωση επικυρωμένων): επιτρέπει σε μια δοσοληψία να βλέπει ενδιάμεσα αποτελέσματα άλλων δοσοληψιών όταν αυτά γίνουν COMMITED.Πρόβλημα **μη επαναλήψιμης ανάγνωσης**. Δηλαδή εδώ μια δοσοληψία διαβάζει αρχικά μια τιμή, μετά αυτή ενημερώνεται από μια commited transaction και όταν μετά πάει να ξαναδιαβάσει την τιμή έχει αλλάξει.
3. **REPEATABLE READ (επαναλήψιμη ανάγνωση) ή SERIALIZABLE** (σειριοποιήσιμο) :επιτρέπουν σε μια δοσοληψία να κάνει σταθερές αναγνώσεις. Όταν διαβάσει κάτι εξακολουθεί να το βλέπει ακόμα και αν μια άλλη δοσοληψία το έχει αλλάξει.(phantoms)
4. Η δήλωση GLOBAL σημαίνει ότι αλλάζεις το επίπεδο απομόνωσης καθολικά, για όλες τις επόμενες συνδέσεις ενώ με τη SESSION μόνο για τις επόμενες συνδέσεις.

Παράδειγμα

Τώρα σύμφωνα με τον παραπάνω πίνακα T1 που αφορά τραπεζικούς λογαριασμούς υλοποιήστε μια δοσοληψία που να προσθέτει 200ευρώ στον λογαριασμό με pin=008 και στη συνέχεια μια δεύτερη η οποία διαβάζει το συνολικό ποσό του λογαριασμού 008 και προσθέτει το 10% αυτού στο λογαριασμό με pin=009

Απάντηση:

Ξεκινάμε την πρώτη δοσοληψία
START TRANSACTION;
΄Ομως εδώ η δεύτερη δοσοληψία θέλουμε να κάνει σταθερές αναγνώσεις γιατί από αυτήν διαβάζει το συνολικό ποσό και το μεταφέρει στον άλλο λογαριασμό. Άρα εδώ πρέπει να ορίσουμε σ'αυτήν μια set transaction με προεπιλογή SERIALIZABLE:
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Εκτελούμε την πράξη ενημέρωσης
UPDATE T1
SET K1=K1+200 **WHERE** K1=1000;
Με **SELECT** * **FROM** T1;
Βλέπουμε από τον παρακάτω πίνακα ότι η ενημέρωση πραγματοποιήθηκε.

L1	K1
001	11500
002	10500
008	1200
009	10000

Και προχωράμε στη δεύτερη:
START TRANSACTION;
UPDATE T1
SET K1=0.1 * K1 + 10000 WHERE K1=10000;(πρόσθεση 10% από Α)
UPDATE T1
SET K1=K1-0.1*K1 WHERE K1=1200; (αφαιρούμε το 0.1*K1)

Το αποτέλεσμα φαίνεται από τον πίνακα που ακολουθεί

L1	K1
001	11500
002	10500
008	1080
009	11080

Διαπιστώνουμε ότι η ενημέρωση έγινε παίρνοντας το σωστό ποσό από την TR1.

Κλείδωμα πίνακα

Τα locks χρησιμοποιούνται είτε για να γίνει ένα είδος συναγωνισμού με τα transactions είτε για να επιταχύνουμε όταν γίνεται ενημέρωση της βάσης.

Η εντολή με την οποία κλειδώνεις και ξεκλειδώνεις ένα πίνακα είναι η εξής

LOCK TABLES

tbl_name [AS *alias*] {READ [LOCAL] | [LOW_PRIORITY] WRITE}

[, *tbl_name* [AS *alias*] {READ [LOCAL] | [LOW_PRIORITY] WRITE}]

UNLOCK TABLES

Μπορούμε να έχουμε είτε **read** είτε **write locks.** Όταν κλειδώνουμε έναν πίνακα, πρέπει να καθορίσουμε αν θέλουμε ένα "read lock" ή ένα "write lock". Το πρώτο εμποδίζει άλλες διεργασίες από το να κάνουν αλλαγές στον πίνακα αλλά επιτρέπει σ' άλλες να διαβάσουν τον πίνακα. Ενώ όταν μια ακολουθία επιτρέψει ένα write lock στον πίνακα μόνο εκείνη μπορεί να διαβάσει απ' αυτόν τον πίνακα και να γράψει σ' αυτόν. Όλες οι άλλες διεργασίες μπλοκάρονται.

Παράδειγμα

Έστω ότι έχουμε δυο clients από τους οποίους ο πρώτος παίρνει ένα read lock στον πίνακα p1 που δείχνει τη βαθμολογία σε κάθε μάθημα. Έπειτα ο δεύτερος προσπαθεί να εισάγει μια εγγραφή στο ίδιο table. Υλοποιήστε το. Τί συμβαίνει;

Απάντηση:

Ξεκινάω με τη δημιουργία πίνακα σε MyISAM :

CREATE TABLE P1 (

ID INT ,

DEG INT,

PRIMARY KEY (ID)

)ENGINE=MyISAM;

Βάζω τιμές στον πίνακα

INSERT INTO P1 VALUES(160, 08);

INSERT INTO P1 VALUES(170, 04);

INSERT INTO P1 VALUES(190, 02);

INSERT INTO P1 VALUES(110, 06);

1os client

Κάνω το read lock

Lock table p1 read;

2os client

INSERT INTO P1 VALUES(185, 10);

Παραπάνω ο 2os client προσπαθεί να ενημερώσει τον πίνακα ρ1 που έχει πάρει read lock όμως παρατηρούμε ότι δεν γίνεται. Για να συμβεί κάτι τέτοιο πρέπει πρώτα ο 1os client να κάνει unlock το table.

Επίσης, εδώ παρατηρούμε ότι ο δεύτερος client μπορεί να κάνει ένα

SELECT * FROM p1;

Δηλαδή, όταν ένας πίνακας πάρει ένα read lock σε μια ακολουθία μπορεί η ίδια και οποιαδήποτε άλλη ακολουθία να διαβάσει αυτόν τον πίνακα αλλά δεν μπορεί να κάνει αλλαγές σ' αυτόν.

Παράδειγμα

Έστω ότι έχουμε δυο clients από τους οποίους ο πρώτος παίρνει ένα write lock στον πίνακα p1 που δείχνει τη βαθμολογία σε κάθε μάθημα. Έπειτα ο δεύτερος προσπαθεί να ενημερώσει το ίδιο table. Υλοποιήστε το. Τι συμβαίνει;

Απάντηση:

1os client

Κάνω το write lock

Lock table p1 write;

2os client

UPDATE P1 SET DEG=05 WHERE DEG=04;

Ο δεύτερος client προσπαθεί να ενημερώσει τον ρ1 αλλά επειδή είναι κλειδωμένος με write lock δεν γίνεται!

Επίσης, αν προσπαθήσει ο δεύτερος να διαβάσει από αυτό το table πάλι δεν μπορεί.

Αν όμως ο πρώτος client κάνει ένα

SELECT * FROM p1;

ή ένα

UPDATE p1 SET deg=05 WHERE deg=04;

Παρατηρούμε ότι γίνεται!

Άρα όταν μια ακολουθία επιτρέψει ένα write lock σε έναν πίνακα μόνο αυτή μπορεί να ενημερώσει ή να διαβάσει αυτόν τον πίνακα. Οποιαδήποτε άλλη ακολουθία μπλοκάρεται. Πρέπει να κάνει πρώτα unlock τον πίνακα.

Έτσι αν μια διεργασία επιτρέψει σε έναν πίνακα να πάρει ένα read lock τότε μπορεί και οποιαδήποτε άλλη να πάρει lock σ' αυτόν τον πίνακα αλλά μόνο read. Ενώ αν ένας πίνακας πάρει write lock σε μια διεργασία τότε καμία άλλη διεργασία δεν μπορεί να πάρει οποιοδήποτε lock σε αυτόν τον πίνακα.

Επίσης μπορούμε να έχουμε **read locαl** και **write locαl** παραμέτρους. Η **read local** επιτρέπει στις μη συγκρουόμενες INSERT δηλώσεις να εκτελούνται ενώ κρατείται το lock.

Τα **LOW_PRIORITY WRITE** locks χρησιμοποιούνται για να επιτρέψουν σε άλλες διεργασίες να αποκτήσουν read locks ενώ οι διεργασίες περιμένουν για το write lock. Τα **LOW_PRIORITY WRITE** locks πρέπει να χρησιμοποιούνται μόνο όταν υπάρχει σιγουριά

Κλείδωμα εγγραφών έναντι κλειδώματος πινάκων

Πλεονεκτήματα του row - locking επιπέδου (κλείδωμα σε εγγραφές):

- Λιγότερες lock συγκρούσεις καθώς προσπελαύνονται διαφορετικές γραμμές σε πολλές ακολουθίες
- Λιγότερες αλλαγές για rollbacks.
- Είναι δυνατό να κλειδώσει μια μόνο γραμμή για πολύ ώρα.

Παρ' όλα αυτά όμως το row - locking έχει και τα εξής **μειονεκτήματα:**

- Καταλαμβάνει περισσότερη μνήμη από τα table-locks επίπεδα

- Είναι πιο αργό από το επίπεδο table-lock όταν χρησιμοποιείται σε ένα μεγάλο μέρος του table γιατί πρέπει να αποκτήσεις πολύ περισσότερα locks
- Είναι πολύ χειρότερο από άλλα locks αν κάνεις συχνά GROUP BY λειτουργίες σε ένα μεγάλο μέρος δεδομένων ή αν πρέπει συχνά να εξετάζεις ολόκληρο το table.
- Με τα υψηλότερου επιπέδου locks μπορείς επίσης πιο εύκολα να υποστηρίξεις locks διαφορετικών τύπων για να συντονίσεις την εφαρμογή επειδή τα υψηλότερα locks είναι λιγότερα από τα row-level.

Όμως, για να επιτύχει μια πολύ υψηλή lock ταχύτητα , η MYSQL χρησιμοποιεί το table-locking , αντί για row-locking, για όλες τις μηχανές αποθήκευσης εκτός από την InnoDB και BDB.

Έτσι, τα table-locks είναι **ανώτερα** από τα row-locks στις ακόλουθες περιπτώσεις :

- Όταν οι περισσότερες δηλώσεις για τα tables είναι read
- Αναγνώσεις και ενημερώσεις σε strict κλειδιά,που ενημερώνεις ή διαγράφεις μια γραμμή που μπορεί να προκαλεσθεί με μια ανάγνωση μοναδικού κλειδιού

UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;

DELETE FROM tbl_name WHERE unique_key_col=key_value;

- Παράλληλες INSERT συνδυασμένες με SELECT δηλώσεις, και πολύ λίγες UPDATE και DELETE δηλώσεις
- Μερικές SCAN και ORDER BY λειτουργίες σε ολόκληρο το table χωρίς εγγραφές.

Για μεγάλα tables, το table locking είναι πολύ καλύτερο από το row για τις περισσότερες εφαρμογές

Δεν παύει όμως και το table locking να μειονεκτεί στις παρακάτω περιπτώσεις:

- Όταν ένας client εισάγει μια δήλωση SELECT που παίρνει πολύ χρόνο να τρέξει.
- Ένας δεύτερος client τότε εισάγει μια δήλωση UPDATE στο ίδιο table. Αυτός ο client θα περιμένει μέχρι η δήλωση SELECT να τελειώσει.
- Ένας άλλος client εισάγει μια άλλη δήλωση SELECT στο ίδιο table. Επειδή η UPDATE έχει μεγαλύτερη προτεραιότητα από τη SELECT, αυτή η SELECT θα περιμένει μέχρι η UPDATE να τελειώσει. Θα περιμένει επίσης μέχρι να ολοκληρωθεί και η πρώτη SELECT.

13.2 Παράδειγμα Εργαστηριακής Άσκησης

Εκφώνηση:	Στην Βάση Δεδομένων του ΑΕΙ της εργαστηριακής άσκησης 4 θα πρέπει να εκτελέσετε τις ακόλουθες ενέργειες: 1. Χρησιμοποιείστε δοσοληψία για την ενημέρωση του βαθμού του φοιτητή «Kouris» στο μάθημα «Baseis Dedomenwn». Αρχικά ακυρώστε την δοσοληψία χρησιμοποιώντας την εντολή rollback και στην συνέχει υποβάλετε τον σωστό βαθμό ολοκληρώνοντας την συνεδρία με την εντολή commit. 2. Σε ένα υποθετικό σενάριο όπου δύο χρήστες τις γραμματείας θα πρέπει ταυτόχρονα να περάσουν βαθμούς μαθημάτων στην βάση δεδομένων και να υπολογίσουν και τον Μέσο Όρο κάποιων φοιτητών, χρησιμοποιήστε τα κατάλληλα επίπεδα απομόνωσης συνεδριών για την ορθή λειτουργία και την λήψη σωστών αποτελεσμάτων.
Ζητούμενα:	Χρήση των δοσοληψιών και κατανόηση των επιπέδων απομόνωσης (isolation levels).

Ζητούμενο 1: Χρησιμοποιείστε δοσοληψία για την ενημέρωση του βαθμού του φοιτητή «Kouris» στο μάθημα «Baseis Dedomenwn». Αρχικά ακυρώστε την δοσοληψία χρησιμοποιώντας την εντολή rollback και στην συνέχει υποβάλετε τον σωστό βαθμό ολοκληρώνοντας την συνεδρία με την εντολή commit.

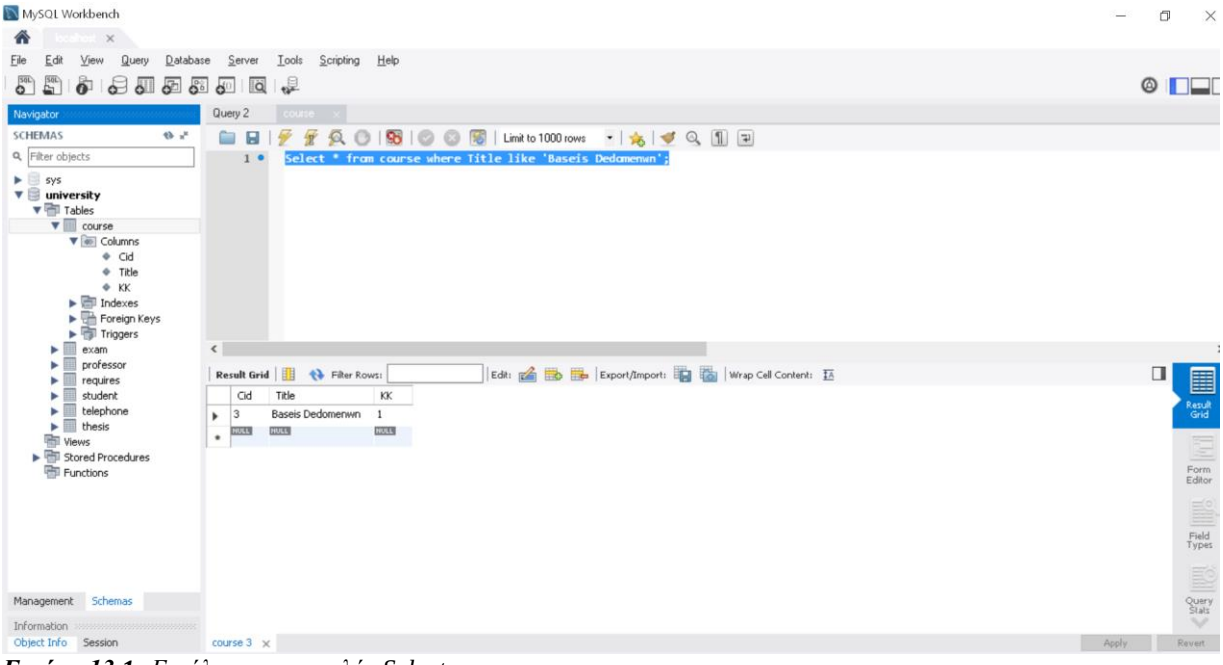
Αρχικά ξεκινάει η ενεργοποίηση της δοσοληψίας και ακύρωση της στην περίπτωση καταχώρησης λάθους βαθμού όπως φαίνεται στο ακόλουθο παράδειγμα:

Επιλέγουμε την Βάση δεδομένων:

```
Use university;
```

Αρχικά βρίσκουμε τον κωδικό του μαθήματος για το οποίο πρέπει να ενημερωθεί ο βαθμός. Ο κωδικός του επίμαχου μαθήματος είναι 3.

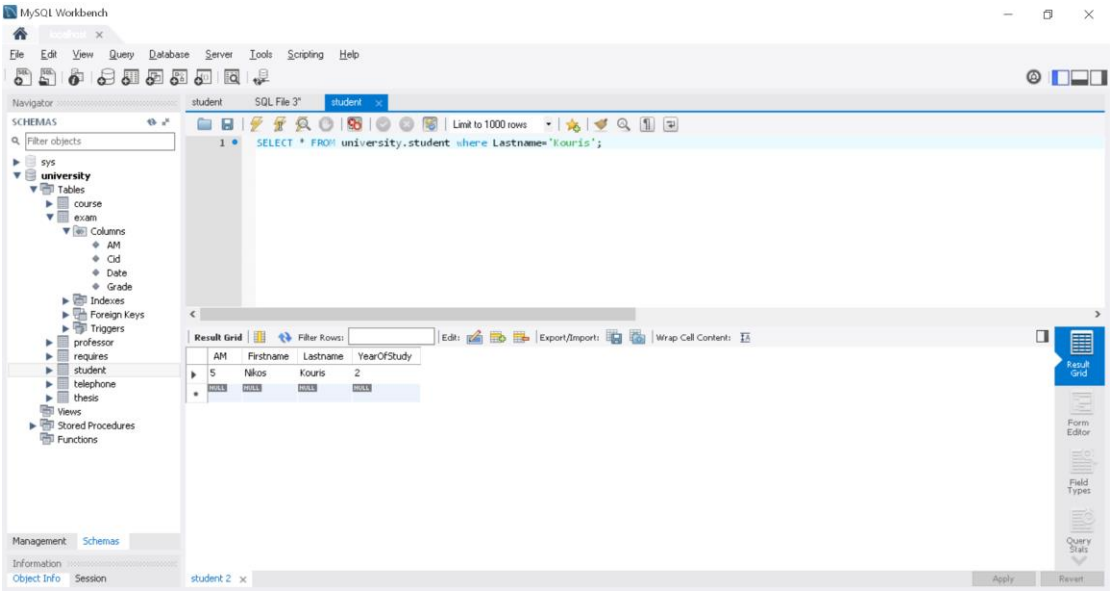
```
Select * from course where Title like 'Baseis Dedomenwn';
```



Εικόνα 13.1: *Εκτέλεση της εντολής Select..*

Στη συνέχεια βρίσκουμε το ΑΜ του μαθητή Kouris. Το ΑΜ του φοιτητή είναι 1.

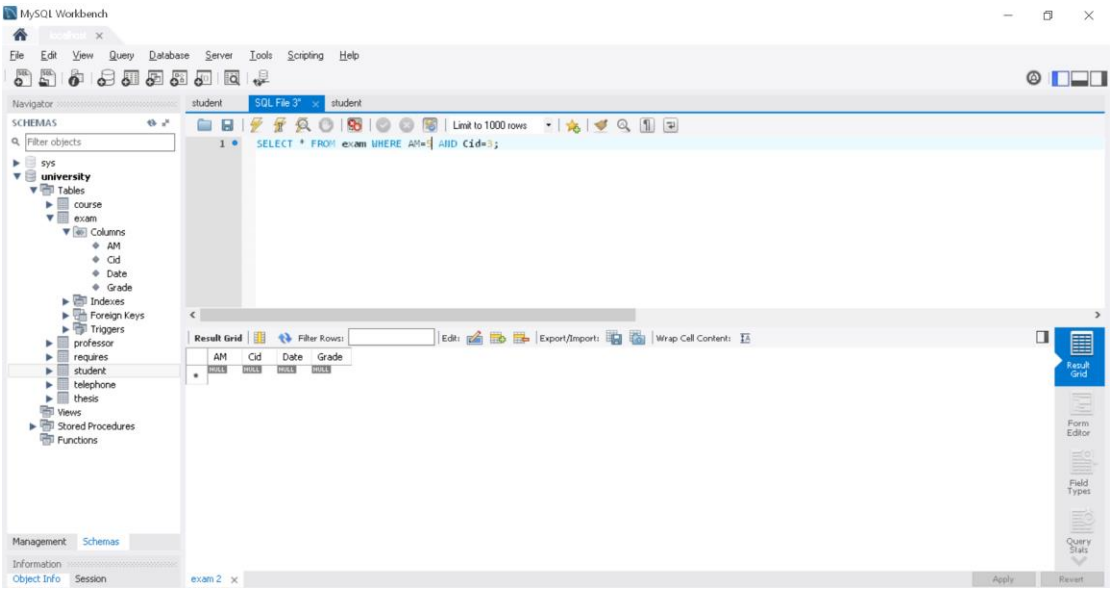
SELECT * FROM university.student where Lastname like Kouris;



Εικόνα 13.2: *Εκτέλεση της εντολής Select..*

Στην συνέχεια εξετάζουμε αν ο φοιτητής έχει ξαναδώσει κάποια εξέταση για το μάθημα.

SELECT * FROM exam WHERE AM=5 AND Cid=3;



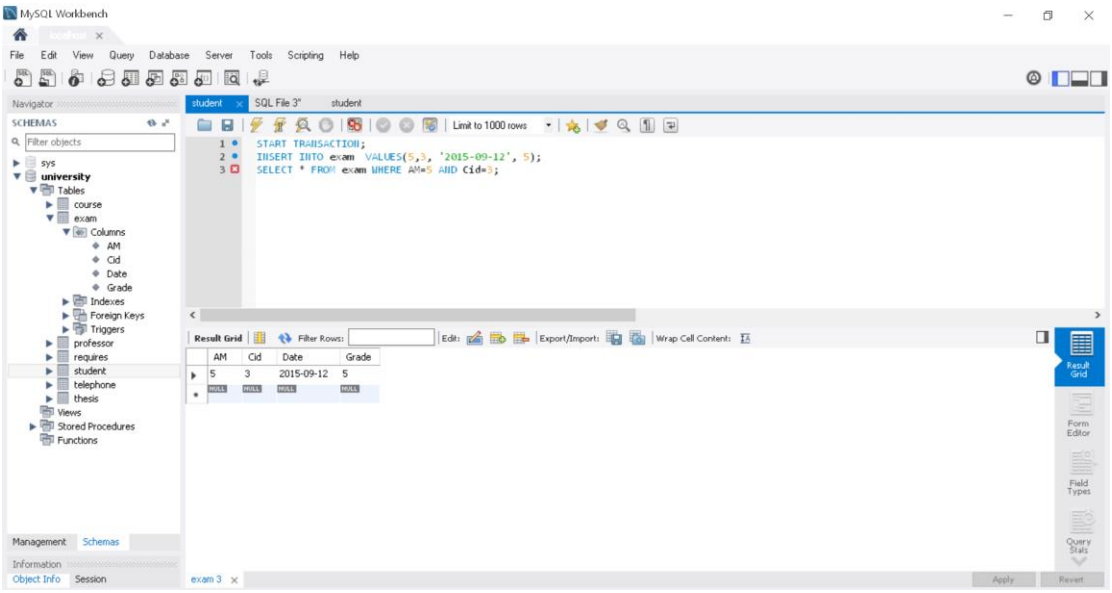
Εικόνα 13.3: *Εκτέλεση της εντολής Select.*

Ξεκινάμε την δοσοληψία που θα κάνει την ενημέρωση γράφουμε την εντολή εισαγωγής νέας εγγραφής και στη συνέχει με χρήση της select βλέπουμε αν η εισαγωγή υπάρχει στο σχήμα exam.

START TRANSACTION;

INSERT INTO exam VALUES(5,3, '2015-09-12', 5);

SELECT * FROM exam WHERE AM=5 AND Cid=3;

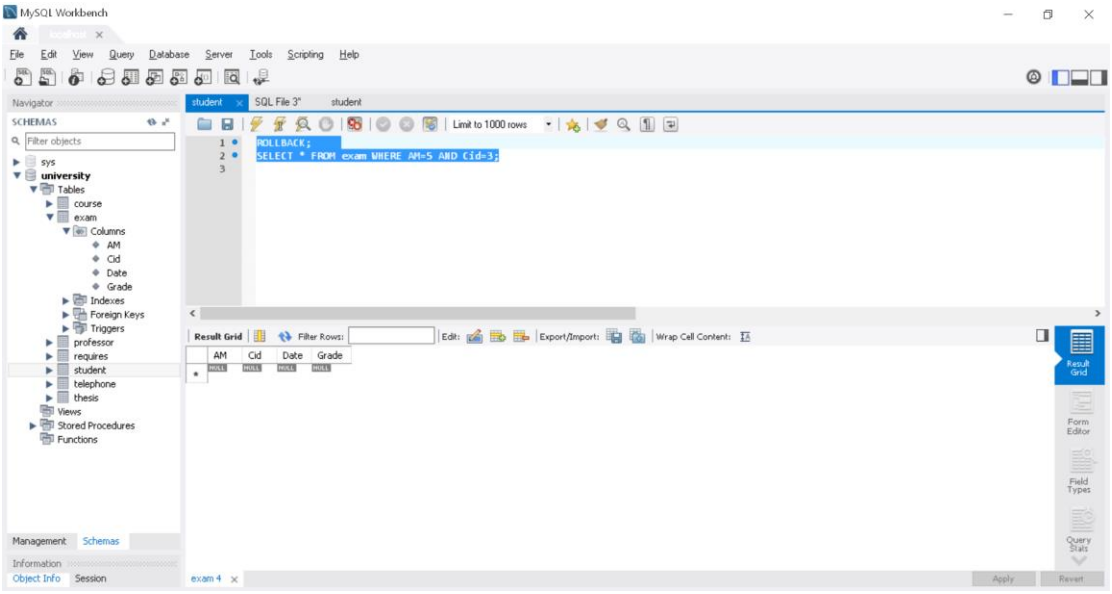


Εικόνα 13.4: *Εκτέλεση εντολών εντός Δοδοληψίας..*

Διαπιστώνουμε ότι ο βαθμός πέντε (5) είναι λάθος οπότε επιλέγουμε να τερματίσουμε την δοσοληψία χωρίς να εκτελεστούν οι αλλαγές στην βάση δεδομένων εκτελώντας την εντολή rollback.

ROLLBACK;

SELECT * FROM exam WHERE AM=5 AND Cid=3;



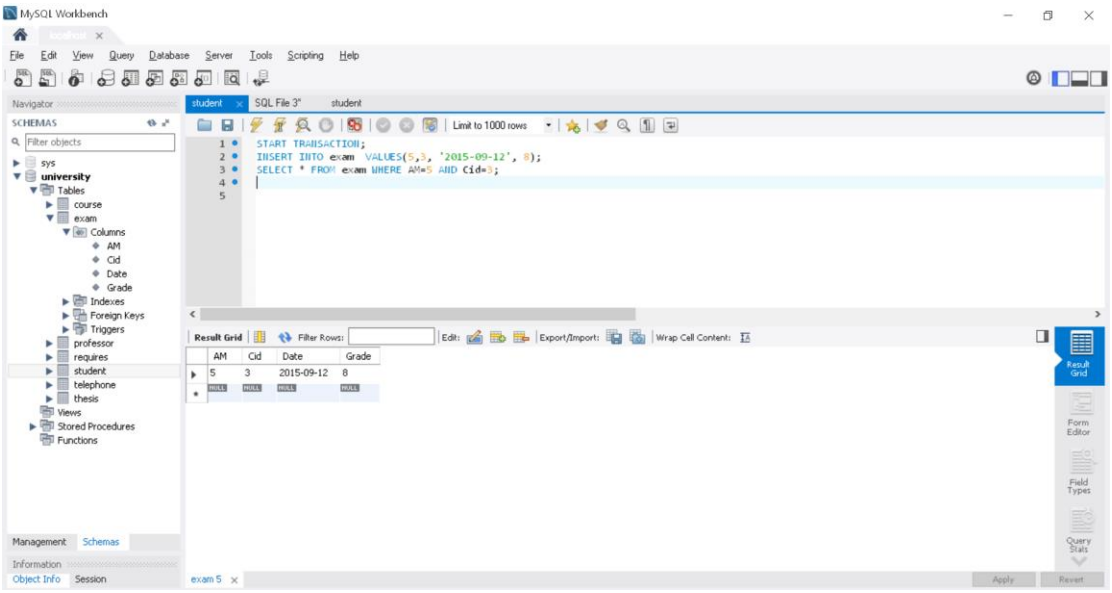
Εικόνα 13.5: *Αναίρεση Δοσοληψίας.*

Παρατηρούμε ότι η διαδικασία ολοκληρώθηκε χωρίς να περάσει η εγγραφή στην βάση δεδομένων. Συνεχίζουμε σε μία νέα συνεδρία όπου χρησιμοποιείται ο σωστός βαθμός 8 στην εξέταση του φοιτητή.

START TRANSACTION;

INSERT INTO exam VALUES (5, 3, '2015-09-12', 8);

SELECT * FROM exam WHERE AM=5 AND Cid=3;

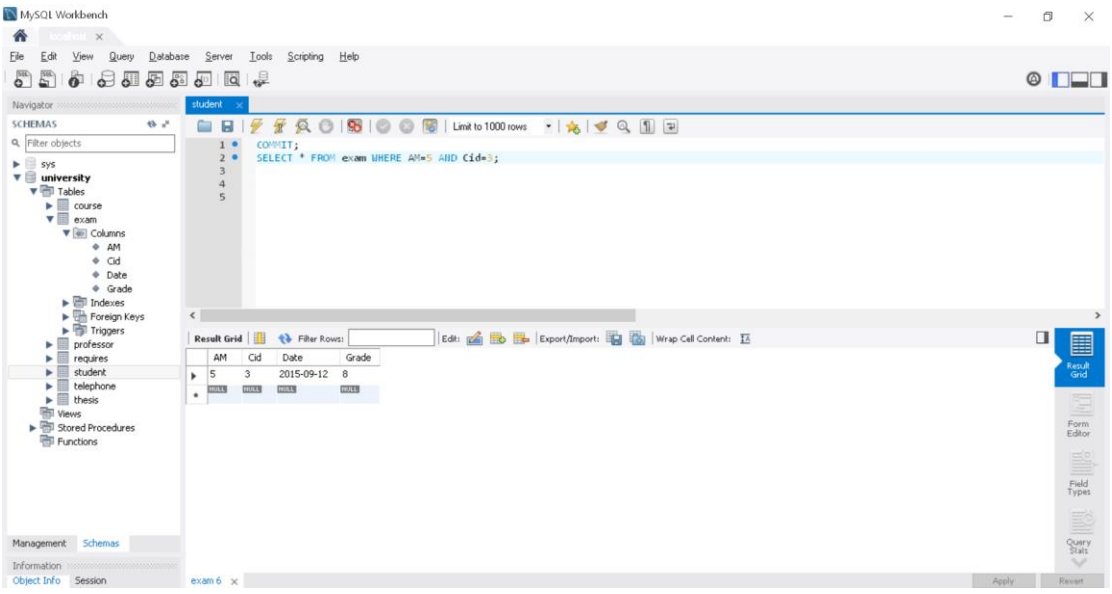


Εικόνα 13.6: *Εκτέλεση Δοσοληψίας..*

Στην συνέχεια ολοκληρώνουμε την συνεδρία με την εντολή commit και οι ενέργειες εκτελούνται τελικά στην βάση δεδομένων.

COMMIT;

SELECT * FROM exam WHERE AM=5 AND Cid=3;

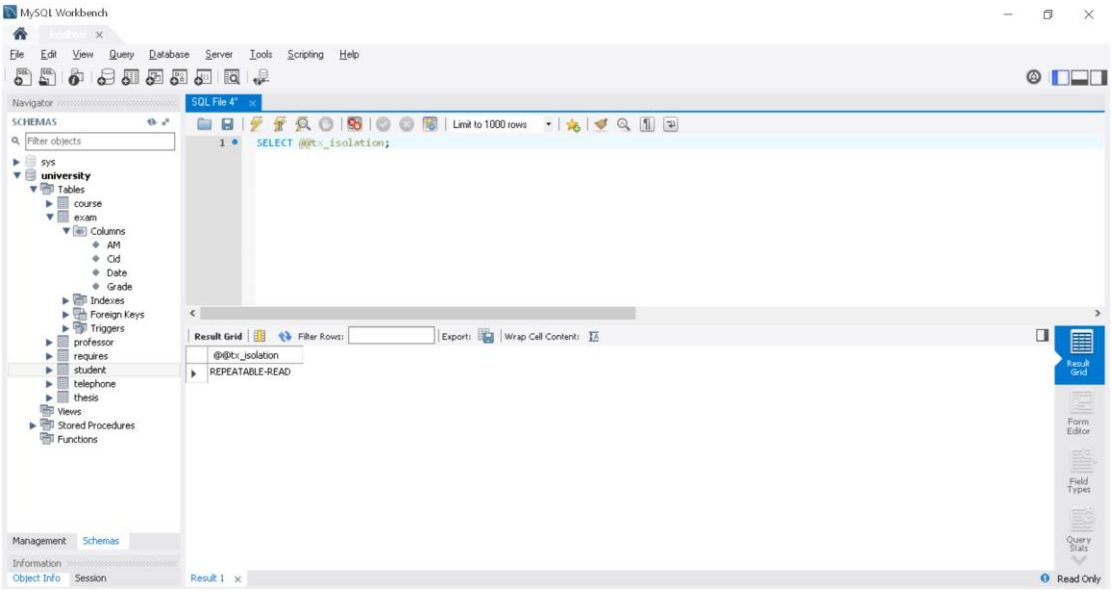


Εικόνα 13.7: *Επιτυχής ολοκλήρωση Δοσοληψίας*

Ζητούμενο 2: Σε ένα υποθετικό σενάριο όπου δύο χρήστες τις γραμματείας θα πρέπει ταυτόχρονα να περάσουν βαθμούς μαθημάτων στην βάση δεδομένων και να υπολογίσουν και τον Μέσο Όρο κάποιων φοιτητών, χρησιμοποιήστε τα κατάλληλα επίπεδα απομόνωσης συνεδριών για την ορθή λειτουργία και την λήψη σωστών αποτελεσμάτων.

Για να δούμε το επίπεδο απομόνωσης δίνουμε την εντολή

SELECT @@tx_isolation;



Εικόνα 13.8: *Επιλογή επιπέδου απομόνωσης.*

Για να ορίσουμε το επίπεδο απομόνωσης μίας δοσοληψίας χρησιμοποιούμε την εντολή:

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{

REPEATABLE READ

| READ COMMITTED

| READ UNCOMMITTED

| SERIALIZABLE

}

Όπως για παράδειγμα:

SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

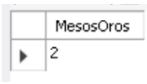
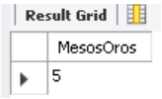
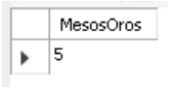
Έστω ότι υπάρχουν δύο χρήστες που θέλουν να εκτελέσουν τις ακόλουθες ενέργειες ταυτόχρονα:

- **Χρήστης Α:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Baseis Dedomenwn χρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,3, '2015-09-12', 8);

- **Χρήστης Β:** Να υπολογίσει τον μέσο όρο του φοιτητή Kouris χρησιμοποιώντας το ακόλουθο ερώτημα: SELECT avg(grade) as MesosOros FROM exam WHERE AM=5

ΣΕΝΑΡΙΟ 1---ISOLATION LEVEL READ UNCOMMITTED

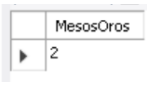
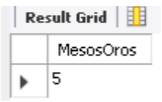
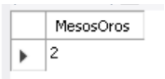
Επιτρέπει στον Χρήστη Β να δει αλλαγές από άλλους χρήστες που δεν έχουν γίνει commit.

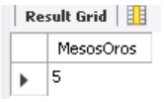
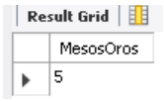
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	
Προσοχή καθώς δίνει ο Χρήστης Α rollback υπάρχει ο κίνδυνος ο Χρήστης Β να συνεχίσει να χρησιμοποιεί τον λανθασμένο Μέσο όρο.	Έστω ότι ο χρήστης δίνει την εντολή rollback για να αναιρέσει την επιλογή του. rollback; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 

Μόνο αν εκτελέσει ξανά το ερώτημα θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά rollback στην ενέργεια του. SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

ΣΕΝΑΡΙΟ 2---ISOLATION LEVEL READ COMMITTED

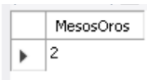
Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit.

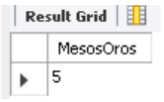
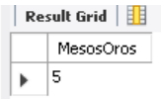
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι δεν βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	

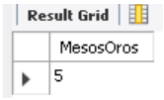
	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
Μόνο αν εκτελέσει ξανά το ερώτημα θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά commit στην ενέργεια του και τα νέα δεδομένα. SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  **Φαινόμενο non-repeatable read το session του Χρήστη Β μέσα στο ίδιο transaction διαβάζει δύο διαφορετικές τιμές.**	

ΣΕΝΑΡΙΟ 3---ISOLATION LEVEL REPEATABLE READ

Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit από τον χρήστη Α αλλά και ταυτόχρονα διατηρεί στην δοσοληψία του την ίδια τιμή. Δηλαδή θα δει τις αλλαγές του χρήστη Α μόλις ο ίδιος ο Χρήστης Β δώσει commit για να ολοκληρώσει την δική του δοσοληψία.

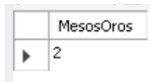
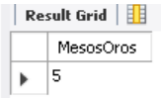
Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SET SESSION TRANSACTION ISOLATION LEVEL **REPEATABLE READ**; SELECT @@tx_isolation; SET autocommit=0; SELECT @@autocommit; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

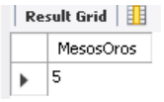
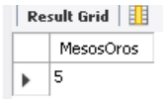
	use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5  Παρατηρούμε ότι δεν βλέπει την εισαγωγή της εγγραφής του Χρήστη Α πριν αυτός δώσει commit.	
	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  **Το φαινόμενο non-repeatable read δεν εμφανίζεται. Το session του Χρήστη Β μέσα στο ίδιο transaction εξακολουθεί να διαβάζει την ίδια τιμή. Μόνο αν εκτελέσει commit και ολοκληρώσει την δική του δοσοληψία θα είναι σε θέση να διαπιστώσει ότι ο Χρήστης Α έκανε τελικά commit στην ενέργεια του και τα νέα δεδομένα.**	

COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	

ΣΕΝΑΡΙΟ 4---ISOLATION LEVEL SERIALIZABLE

Επιτρέπει στον Χρήστη Β να δει μόνο αλλαγές από άλλους χρήστες που έχουν γίνει commit από τον χρήστη Α. Αν κατά την διάρκεια της ενημέρωσης από τον Α ο Β προσπαθήσει να εκτελέσει το ερώτημα του, τότε η δοσολοψία του θα «κλειδώσει» μέχρι να ολοκληρωθεί αυτή του Α.

Ενέργειες Χρήστη Β	**Ενέργειες Χρήστη Α**
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  	
	SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE; use university; start transaction; INSERT INTO exam VALUES(5,3, '2015-09-12', 8); SELECT avg(grade) as MesosOros FROM exam WHERE AM=5; 
SELECT avg(grade) as MesosOros FROM exam WHERE AM=5 **Παρατηρούμε ότι εδώ η δοσοληψία «κλειδώνει» μέχρι να ολοκληρώσει την δοσοληψία του ο Α.**	

	Έστω ότι ο χρήστης δίνει την εντολή commit για να ενημερώσει την βάση δεδομένων. COMMIT; SELECT avg(grade) as MesosOros FROM exam WHERE AM=5;  Result Grid MesosOros 5
 Result Grid MesosOros 5 **Ολοκληρώνεται η εντολή του Χρήστη Β αφού ο Α ολοκλήρωσε την δική του δοσοληψία.**	

13.3 Άλυτες Εργαστηριακές Ασκήσεις

Άσκηση 1

Επεκτείνεται το ζητούμενο 2 της λυμένης εργαστηριακής άσκησης για τρεις χρήστες αυτή τη φορά/ Έστω ότι υπάρχουν τρεις χρήστες που θέλουν να εκτελέσουν τις ακόλουθες ενέργειες ταυτόχρονα:

- **Χρήστης Α:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Baseis Dedomenwn χρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,3, '2015-09-12', 8);
- **Χρήστης Β:** Να υπολογίσει τον μέσο όρο του φοιτητή Kouris χρησιμοποιώντας το ακόλουθο ερώτημα: SELECT avg(grade) as MesosOros FROM exam WHERE AM=5
- **Χρήστης C:** Ενημερώνει τον βαθμό του φοιτητή Kouris στο μάθημα Programmatismos IIχρησιμοποιώντας την εντολή: INSERT INTO exam VALUES(5,6, '2015-09-12', 4);

Δοκιμάστε και τα 4 επίπεδα απομόνωσης όπως και στην λυμένη άσκηση.

Βιβλιογραφία/Αναφορές

R. Elmasri & S.B. Navathe "Θεμελιώδεις Αρχές Συστημάτων ΒΔ - 4η Έκδοση" Κεφάλαιο 17.
R. Ramakrishnan & J. Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., New York, NY, USA. Κεφάλαιο 16.
TRANSACTIONS http://dev.mysql.com/doc/refman/5.7/en/commit.html