

**Ανοικτά Ακαδημαϊκά Μαθήματα**

**Πανεπιστήμιο Δυτικής Αττικής**

**Βάσεις Δεδομένων ΙΙ**

**Ενότητα 8:** Διαχείριση δοσοληψιών στο προϊόν mySQL – Επίπεδα απομόνωσης (Transactions – Isolation levels)

Χ. Σκουρλάς

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

|  |  |
| --- | --- |
| Το περιεχόμενο του μαθήματος διατίθεται με άδεια Creative Commons εκτός και αν αναφέρεται διαφορετικά | Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους. |

Περιεχόμενα

[1. Διαχείριση δοσοληψιών στο προϊόν mySQL – Επίπεδα απομόνωσης (Transactions – Isolation levels) 3](#_Toc401506382)

[1.1 Θέμα 1: Προσομοίωση «Τυφλής αντικατάστασης» (Blind Overwriting) 4](#_Toc401506383)

[1.2 Θέμα 2: Επανάλαβε το Θέμα 1 χρησιμοποιώντας απευθείας “sensitive updates” χωρίς να χρησιμοποιήσεις τοπικές μεταβλητές -using “sensitive updates” in SELECT-UPDATE scenarios without local variables- 7](#_Toc401506384)

[1.3 Θέμα 3: Διατήρηση των S-κλειδαριών(S-locks, Shared Locks) 9](#_Toc401506385)

[1.4 Θέμα 4: Επανάληψη με χρήση isolation level REPEATABLE READ. 13](#_Toc401506386)

[1.5 Θέμα 5: Ανταγωνισμός σε δύο “resources” με διαφορετική σειρά 17](#_Toc401506387)

[1.6 Θέμα 6: Απόπειρα εμφάνισης dirty read 19](#_Toc401506388)

[1.7 Θέμα 7: The non-repeatable read anomaly 22](#_Toc401506389)

[Άσκηση 24](#_Toc401506390)

[1.8 Θέμα 8: Απόπειρα εμφάνισης insert phantom 25](#_Toc401506391)

Στόχος του όγδοου εργαστηρίου είναι η κατανόηση σε βάθος της διαχείρισης δοσοληψιών και ειδικότερα των επιπέδων απομόνωσης. Μετά την επεξεργασία του εργαστηρίου ο ενδιαφερόμενος θα έχει κατανοήσει και θα έχει εμπεδώσει τον τρόπο διαχείρισης επιπέδων απομόνωσης στο προϊόν mySQL.

**Λέξεις Κλειδιά**: Transactions, Transactions processing, Isolation levels, mySQL

# Διαχείριση δοσοληψιών στο προϊόν mySQL – Επίπεδα απομόνωσης (Transactions – Isolation levels)

**Προσοχή το προϊόν της mySQL λειτουργεί όταν συνδεθούμε σε AUTOCOMMIT mode.**

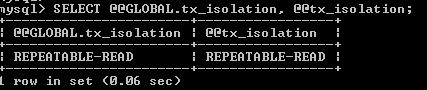
DROP DATABASE IF EXISTS TestDB;

CREATE DATABASE TestDB;

use TestDB;

Το προκαθορισμένο (default) isolation level στη MySQL είναι το REPEATABLE READ:

SELECT @@GLOBAL.tx\_isolation, @@tx\_isolation;



**Αρχικοποίηση του πίνακα *Accounts* table**

USE TestDB;

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

acctID INTEGER NOT NULL PRIMARY KEY,

balance INTEGER NOT NULL,

CONSTRAINT remains\_nonnegative CHECK (balance >= 0)

);

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT;



## Θέμα 1: Προσομοίωση «Τυφλής αντικατάστασης» (Blind Overwriting)

Σημείωση:

Στο βήμα 4 θυμηθείτε ότι το χρονικό όριο της προκαθορισμένης κλειδαριάς στο προϊόν της MySQL (MySQL’s default lock timeout) είναι 90 seconds. Επομένως, η συναλλαγή Α - transaction (client)- πρέπει να προχωρήσει χωρίς καθυστέρηση στο βήμα 5 αμέσως μετά το βήμα 4 της συναλλαγής Β.

**Blind Overwriting problem, application simulated by use of local variables**

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  -- amount to be transfered by A  SET @amountA = 200;  SET @balanceA = 0; -- init value  SELECT balance INTO @balanceA  FROM Accounts  WHERE acctID = 101;  SET @balanceA = @balanceA - @amountA;  SELECT @balanceA; |  |
| 2 |  | SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  -- amount to be transfered by B  SET @amountB = 500;  SET @balanceB = 0; -- init value  SELECT balance INTO @balanceB  FROM Accounts WHERE acctID = 101;  SET @balanceB = @balanceB - @amountB;  SELECT @balanceB; |
| 3 | UPDATE Accounts  SET balance = @balanceA  WHERE acctID = 101; |  |
| 4 |  | UPDATE Accounts  SET balance = @balanceB  WHERE acctID = 101; |
| 5 | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |  |
|  |  | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |
|  | SELECT \* FROM Accounts; |  |
|  |  | SELECT \* FROM Accounts; |

Όλα τα προϊόντα DBMS υποστηρίζουν έλεγχο ταυτοχρονισμού (concurrency control), έτσι ώστε σε όλα τα επίπεδα απομόνωσης (isolation levels) να μην εμφανίζεται η ανωμαλία χαμένης ενημέρωσης (the lost update anomaly). Παρόλα αυτά, υπάρχει πάντα η περίπτωση απρόσεκτα γραμμένου κώδικα της εφαρμογής, όπου επιχειρείται «τυφλή αντικατάσταση» και η οποία οδηγεί στις ίδιες καταστροφικές συνέπειες -με την ανωμαλία ενημέρωσης- από την πίσω πόρτα!

## Θέμα 2: Επανάλαβε το Θέμα 1 χρησιμοποιώντας απευθείας “sensitive updates” χωρίς να χρησιμοποιήσεις τοπικές μεταβλητές -using “sensitive updates” in SELECT-UPDATE scenarios without local variables-

**Φαινόμενο ανταγωνισμού για την ίδια γραμμή**

**- SELECT-UPDATE competition on the same (row) resource -**

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

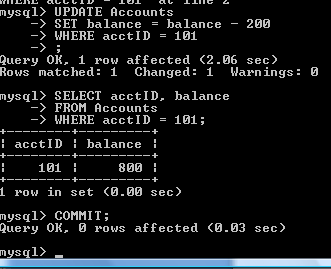
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT WORK;

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  SERIALIZABLE;  SELECT balance FROM Accounts  WHERE acctID = 101; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  SERIALIZABLE;  SELECT balance FROM Accounts  WHERE acctID = 101; |
| 3 | UPDATE Accounts  SET balance = balance - 200  WHERE acctID = 101; |  |
| 4 |  | UPDATE Accounts  SET balance = balance - 500  WHERE acctID = 101; |
| 5 | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |  |
|  |  | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |

Session A



Session B



## Θέμα 3: Διατήρηση των S-κλειδαριών(S-locks, Shared Locks)

**Αρχικοποίηση του πίνακα *Accounts* table**

DROP DATABASE IF EXISTS TestDB;

CREATE DATABASE TestDB;

USE TestDB;

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

acctID INTEGER NOT NULL PRIMARY KEY,

balance INTEGER NOT NULL,

CONSTRAINT remains\_nonnegative CHECK (balance >= 0)

);

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT;

**εναλλακτικά**

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

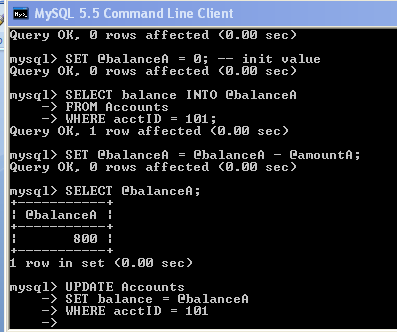
COMMIT WORK;

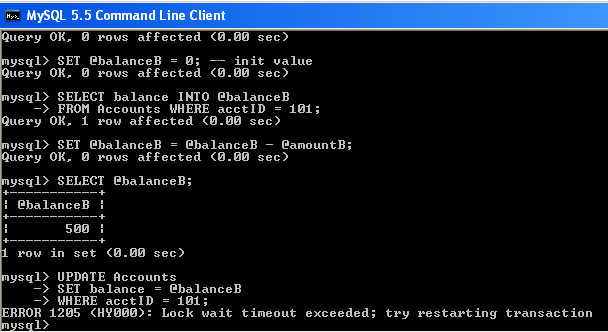
**keeping the S-locks**

**Επανάληψη του θέματος 1 αλλά με χρήση isolation level SERIALIZABLE.**

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  -- amount to be transfered by A  SET @amountA = 200;  SET @balanceA = 0; -- init value  SELECT balance INTO @balanceA  FROM Accounts  WHERE acctID = 101;  SET @balanceA = @balanceA - @amountA;  SELECT @balanceA; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  -- amount to be transfered by B  SET @amountB = 500;  SET @balanceB = 0; -- init value  SELECT balance INTO @balanceB  FROM Accounts WHERE acctID = 101;  SET @balanceB = @balanceB - @amountB;  SELECT @balanceB; |
| 3 | UPDATE Accounts  SET balance = @balanceA  WHERE acctID = 101; |  |
| 4 |  | UPDATE Accounts  SET balance = @balanceB  WHERE acctID = 101; |
| 5 | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |  |
|  |  | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |
|  |  |  |



****

****

## Θέμα 4: Επανάληψη με χρήση isolation level REPEATABLE READ.

**Αρχικοποίηση του πίνακα *Accounts* table**

DROP DATABASE IF EXISTS TestDB;

CREATE DATABASE TestDB;

USE TestDB;

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

acctID INTEGER NOT NULL PRIMARY KEY,

balance INTEGER NOT NULL,

CONSTRAINT remains\_nonnegative CHECK (balance >= 0)

);

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT;

**εναλλακτικά**

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT WORK;

**Repeating exercise, but now using isolation level REPEATABLE READ.**

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  -- amount to be transfered by A  SET @amountA = 200;  SET @balanceA = 0; -- init value  SELECT balance INTO @balanceA  FROM Accounts  WHERE acctID = 101;  SET @balanceA = @balanceA - @amountA;  SELECT @balanceA; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  REPEATABLE READ;  -- amount to be transfered by B  SET @amountB = 500;  SET @balanceB = 0; -- init value  SELECT balance INTO @balanceB  FROM Accounts WHERE acctID = 101;  SET @balanceB = @balanceB - @amountB;  SELECT @balanceB; |
| 3 | UPDATE Accounts  SET balance = @balanceA  WHERE acctID = 101; |  |
| 4 |  | UPDATE Accounts  SET balance = @balanceB  WHERE acctID = 101; |
| 5 | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |  |
|  |  | SELECT acctID, balance  FROM Accounts  WHERE acctID = 101;  COMMIT; |

## Θέμα 5: Ανταγωνισμός σε δύο “resources” με διαφορετική σειρά

**-UPDATE-UPDATE competition on two resources in different order-**

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

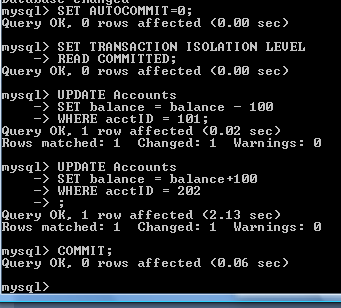
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

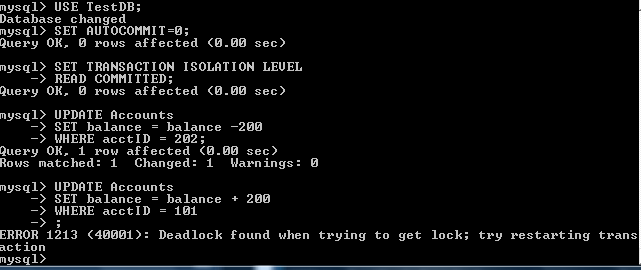
COMMIT WORK;

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  UPDATE Accounts  SET balance = balance - 100  WHERE acctID = 101; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT=0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  UPDATE Accounts  SET balance = balance -200  WHERE acctID = 202; |
| 3 | UPDATE Accounts  SET balance = balance+100  WHERE acctID = 202; |  |
| 4 |  | UPDATE Accounts  SET balance = balance + 200  WHERE acctID = 101; |
| 5 | COMMIT;  SELECT acctID, balance  FROM Accounts  WHERE acctID = 101; |  |
|  |  | COMMIT;  SELECT acctID, balance  FROM Accounts  WHERE acctID = 101; |

Session A



Session B



Το Επίπεδο απομόνωσης δεν παίζει κάποιο ρόλο σε αυτό το σενάριο, αλλά είναι μια καλή πρακτική ο καθορισμός επίπεδου απομόνωσης στην αρχή κάθε συναλλαγής!

## Θέμα 6: Απόπειρα εμφάνισης dirty read

**(An attempt to produce the occurrence of a dirty read situation).**

Transaction A runs in (mySQL’s default) REPEATABLE READ, whereas transaction B is set to run in READ UNCOMMITTED isolation level

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

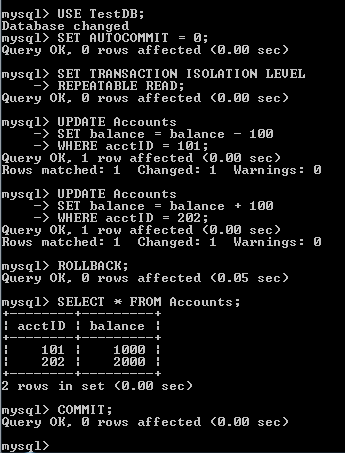
SELECT \* FROM Accounts;

COMMIT WORK;

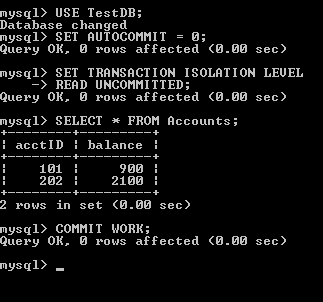
|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  REPEATABLE READ;  UPDATE Accounts  SET balance = balance - 100  WHERE acctID = 101;  UPDATE Accounts  SET balance = balance + 100  WHERE acctID = 202; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  READ UNCOMMITTED;  SELECT \* FROM Accounts;  COMMIT WORK; |
| 3 | ROLLBACK;  SELECT \* FROM Accounts;  COMMIT; |  |

Dirty read problem

Session A



Session B



**Questions**

**A) What if ‘READ UNCOMMITTED’ is replaced ‘READ COMMITTED’ in transaction B?**

**B) What if ‘READ UNCOMMITTED’ is replaced ‘REPEATABLE READ’ in transaction B?**

**C) What if ‘READ UNCOMMITTED’ is replaced ‘SERIALIZABLE’ in transaction B?**

## Θέμα 7: The non-repeatable read anomaly

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

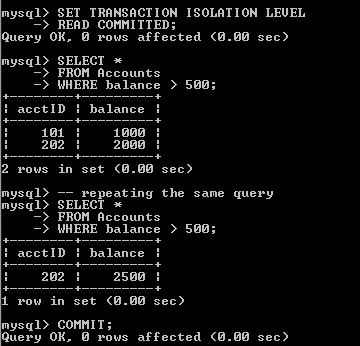
INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

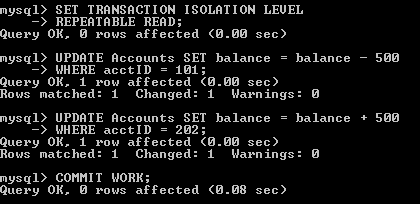
COMMIT WORK;

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  SELECT \*  FROM Accounts  WHERE balance > 500; |  |
| 2 |  | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  REPEATABLE READ;  UPDATE Accounts SET balance = balance - 500  WHERE acctID = 101;  UPDATE Accounts SET balance = balance + 500  WHERE acctID = 202;  COMMIT WORK; |
| 3 | -- repeating the same query  SELECT \*  FROM Accounts  WHERE balance > 500;  COMMIT; |  |

Session A



Session B



Άσκηση

**Τι θα συμβεί αν θέσουμε στην transaction A το επίπεδο απομόνωσης (isolation level) ίσο με REPEATABLE READ;**

## Θέμα 8: Απόπειρα εμφάνισης insert phantom

DELETE FROM Accounts;

INSERT INTO Accounts (acctID,balance) VALUES (101,1000);

INSERT INTO Accounts (acctID,balance) VALUES (202,2000);

SELECT \* FROM Accounts;

COMMIT WORK;

|  |  |  |
| --- | --- | --- |
| Step | Session A | Session B |
| 1 | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  REPEATABLE READ ; |  |
| 2 | -- Accounts having balance > 1000 euros;  SELECT \* FROM Accounts  WHERE balance > 1000; |  |
| 3 |  | USE TestDB;  SET AUTOCOMMIT = 0;  SET TRANSACTION ISOLATION LEVEL  READ COMMITTED;  INSERT INTO Accounts (acctID, balance)  VALUES (303,3000);  COMMIT; |
|  | -- Can we see the new account 303?  SELECT \* FROM Accounts  WHERE balance > 1000;  COMMIT; |  |

**Ερωτήσεις**

**α) Η συναλλαγή Β πρέπει να περιμένει τη συναλλαγή A;**

**β) Είναι η (πρόσφατα εισαχθείσα από τη συναλλαγή Β) γραμμή acctID = 303 ορατή στο περιβάλλον της συναλλαγής Α;**

**γ) Επηρεάζεται το σύνολο αποτελεσμάτων (resultset) του βήματος 4, αν αλλάξετε τη σειρά των βημάτων 2 και 3;**

**Σημειώματα**

**Σημείωμα Αναφοράς**

Copyright Πανεπιστήμιο Δυτικής Αττικής, Χ. Σκουρλάς.

Χ. Σκουρλάς, «Βάσεις Δεδομένων ΙΙ. Ενότητα 8: Διαχείριση δοσοληψιών στο προϊόν mySQL – Επίπεδα απομόνωσης (Transactions – Isolation levels)». Έκδοση: 1.0. Αθήνα 2019. Διαθέσιμο από τη δικτυακή διεύθυνση: [pyles.uniwa.gr](https://ocp.teiath.gr/).

**Σημείωμα Αδειοδότησης**

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[](file:///C:\Users\pantelis\Downloads\%5b1%5d%20http:\creativecommons.org\licenses\by-nc-sa\4.0\)

[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

* που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
* που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
* που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

**Διατήρηση Σημειωμάτων**

* Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
* το Σημείωμα Αναφοράς
* το Σημείωμα Αδειοδότησης
* τη δήλωση Διατήρησης Σημειωμάτων
* το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

**Σημείωμα Χρήσης Έργων Τρίτων**

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

“SQL Transactions” Educational and Training Content, The DBTech VET Teachers (EU LLP Transfer of Innovation) project, 1/10/2012 – 30/9/2014. Retrieved 14 May 2013. <http://www.dbtechnet.org>, διαθέσιμο με άδεια [CC BY-NC-SA 3.0](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.el)