ΠΕΡΙΕΧΟΜΕΝΑ:

1. Συναλλαγές και COMMIT
2. Χειρισμός Λαθών και ROLLBACK
    1. Εντοπισμός Λαθών
3. Σημεία Επαναφοράς
4. Προβλήματα Συγχρονισμού

Ευάγγελος Μ.

Σμαραγδένιος Χορηγός Μαθήματος

Κωνσταντίνος Μπλέτσης

Σμαραγδένιος Χορηγός Μαθήματος

ΜΑΘΗΜΑ 2.6: Συναλλαγές

1. Συναλλαγές και COMMIT

SQL psounis

Μία Συναλλαγή (Transaction):

- είναι μια **ομαδοποίηση εντολών SQL** που θα τρέξουν **όλες μαζί** στον server.
- μπορούν να είναι μέρος:
  - ενός script
  - μίας stored procedure

Γιατί είναι χρήσιμες οι συναλλαγές:

- Γενικά στη βάση μας θα είναι συνδεδεμένοι πολλοί χρήστες.
- Κάθε χρήστης έχει το δικό του νήμα εκτέλεσης ενεργειών. Κάθε φορά μία ενέργεια πραγματοποιείται στη βάση. Οι ενέργειες των χρηστών εναλλάσσονται αλγοριθμικά.
- Οι εντολές που τρέχουν οι διαφορετικοί χρήστες μπορεί να έχουν απροσδόκητα αποτελέσματα, π.χ.
  - Κάποιος χρήστης ξεκινά εισαγωγές εγγραφών σε σχετιζόμενους πίνακες και ολοκληρώνει μερικές. Ένας άλλος χρήστης διαγράφει μία απαραίτητη στον πρώτο χρήστη εγγραφή. Ο πρώτος χρήστης είναι μετέωρος.
- Τα transactions λύνουν το πρόβλημα ως εξής: εκτελούν ομάδες εντολών στο πνεύμα **«Κάντα όλα, ή μην κάνεις τίποτα απολύτως»**

Πότε είναι χρήσιμες οι συναλλαγές:

- Όταν γίνονται πολλά INSERT, UPDATE ή DELETE επί σχετιζόμενων πινάκων, αλλά η αποτυχία κάποιου από αυτά θα προκαλούσε **ασυνέπεια δεδομένων στη βάση (data integrity failure)**

Ορισμός transaction:

```
START TRANSACTION;
...
COMMIT;
```

- Οι εντολές που ομαδοποιούνται έπονται του START TRANSACTION.
- Δίνουμε την εντολή να εκτελεστούν οι εντολές με την COMMIT.

Παράδειγμα 1: DB: emp, script: transaction.sql

```
CREATE PROCEDURE test()
BEGIN
    START TRANSACTION;

    INSERT INTO departments (dept_no, dept_name)
       VALUES (1, 'it');

    INSERT INTO employees (emp_no, birth_date, first_name,
                           last_name, gender, hire_date)
       VALUES(1, '1970-01-01', 'Tywin', 'Lannister', 'M', '2020-01-01');

   INSERT INTO dept_manager(dept_no,emp_no,from_date,to_date)
       VALUES(1, 1, '2020-01-01', '2021-01-01');

    COMMIT;
END
```

Παράδειγμα 2: DB: emp, script: transaction2.sql (βλ.βίντεο)

ΜΑΘΗΜΑ 2.6: Συναλλαγές — 2. Χειρισμός Λαθών και ROLLBACK

SQL psounis

Η MySQL προσφέρει **διαχείριση λαθών** σε μία αποθηκευμένη διαδικασία ως εξής:

- Δηλώνεται (στην αρχή της διαδικασίας) ένας χειριστής ως:

```
DECLARE action HANDLER FOR condition statement;
```

- **action:**
  - **EXIT** (έξοδος) ή **CONTINUE** (συνέχεια) της διαδικασίας εφόσον προκύψει λάθος
- **condition:**
  - **Κωδικός λάθους** της MySQL
  - **SQLSTATE** που ομαδοποιεί τα εξής:
    - **SQLWARNING**: προειδοποίηση (εμφανίζεται, αλλά ο κώδικας τρέχει)
    - **SQLEXCEPTION**: Μη αναστρέψιμο λάθος
    - **NOTFOUND**: (αφορά cursors και αποτυχία σε SELECT…INTO)
- **statement:**
  - **εντολή της MySQL** (συνήθως SET μεταβλητής για μετέπειτα χειρισμό)
  - μπορεί όμως να είναι και ολόκληρο **μπλοκ κώδικα** μέσα σε BEGIN…END.

Κωδικοί Λάθους:

- Υπάρχουν αρκετές εκατοντάδες κωδικοί λάθους
- Συνήθεις: 1045 (access denied), 1062 (πρωτεύον κλειδί), 1064 (συντακτικό λάθος), 2008 (out of memory), κ.λπ.

- Πλήρης λίστα των λαθών βρίσκεται στη σελίδα: https://dev.mysql.com/doc/refman/8.0/en/error-handling.html

Η εντολή: `ROLLBACK;`

- Επαναφέρει τη βάση στην κατάσταση που ήταν στην αρχή της συναλλαγής.

- (Συνεπώς μπορούμε να κάνουμε έλεγχο λαθών και αν προέκυψε λάθος στο transaction, να επαναφέρουμε τη βάση στην προηγούμενη κατάσταση)

Παράδειγμα 3: DB: emp, script: errorhandling.sql

```
CREATE PROCEDURE test()
BEGIN
    DECLARE err TINYINT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET err = 1;
    START TRANSACTION;
    …
    IF err = 1 THEN
        ROLLBACK;
        SELECT 'An error occurred' AS message;
    ELSE
        COMMIT;
        SELECT 'OK' AS message;
    END IF;
END
```

- Ο εντοπισμός του σημείου που έγινε το λάθος, είναι επίπονη διαδικασία.
- Η MySQL δεν προσφέρει μηχανισμούς αποσφαλμάτωσης (debugging).
- **Ωστόσο παρέχεται η δυνατότητα να εμφανισουμε πληροφορίες για το λάθος στην τελευταία εντολή που εκτελέσαμε.**

Η εντολή:

```
SHOW ERRORS;
```

- Εμφανίζει πληροφορίες λάθους στην τελευταία εντολή που εκτελέσαμε.

Για να αποθηκεύσουμε (π.χ. σε μεταβλητές) τις πληροφορίες λαθών, χρησιμοποιούμε το script:

```
GET DIAGNOSTICS CONDITION 1
  @var1 = MYSQL_ERRNO, @var2 = MESSAGE_TEXT;
```

Παράδειγμα 4: DB: emp, script: error.sql

```
...
SHOW errors;
GET DIAGNOSTICS CONDITION 1
      @P1 = MYSQL_ERRNO, @P2 = MESSAGE_TEXT;
SELECT @P1, @P2;
```

- Σε μία stored procedure, ο μηχανισμός λειτουργεί μόνο στο μπλοκ κώδικα του χειριστή του λάθους.

Παράδειγμα 5: DB: emp, script: errorhandlingdebug.sql

```
CREATE PROCEDURE test()
BEGIN
    DECLARE current_error VARCHAR(100);
    DECLARE error_string VARCHAR(300) DEFAULT '';
    DECLARE error_number INT;
    DECLARE error_message VARCHAR(50);
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      BEGIN
        GET DIAGNOSTICS CONDITION 1
          error_number = MYSQL_ERRNO, error_message = MESSAGE_TEXT;
        SET current_error = CONCAT(error_number, '-', error_message, '  ');
      END;
    START TRANSACTION;
    SET current_error='';
    INSERT INTO departments(dept_no, dept_name)
            VALUES (1, 'it');
    IF LENGTH(current_error) > 0 THEN
        SET error_string = CONCAT(error_string, 'Q1: ', error_message, ' ‘);
    END IF;
    ...
    IF LENGTH(error_string)>0 THEN
        ROLLBACK;
        SELECT error_string AS message;
    ELSE
        COMMIT;
        SELECT 'OK' AS message;
    END IF;
END PROCEDURE
```

3. Σημεία Επαναφοράς

- Μπορούμε να επαναφέρουμε τη βάση δεδομένων όχι μόνο στην αρχή της συναλλαγής, αλλά και σε κάποιο ενδιάμεσο σημείο της συναλλαγής.
  - Ορίζουμε το **σημείο επαναφοράς (savepoint)** σε κάποιο ενδιάμεσο σημείο της συναλλαγής
  - Επαναφέρουμε τη βάση σε αυτό το σημείο με την εντολή ROLLBACK TO SAVEPOINT

- Ορίζουμε ένα σημείο επαναφοράς με την εντολή:

```
SAVEPOINT savepoint_name;
```

- προκαλούμε την επαναφορά της βάσης στο σημείο επαναφοράς με την εντολή:

```
ROLLBACK TO SAVEPOINT savepoint_name;
```

Παράδειγμα 6: DB: emp, script: savepoint.sql

```
CREATE PROCEDURE test()
BEGIN
    DECLARE err TINYINT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET err = 1;

    START TRANSACTION;

    INSERT INTO employees(emp_no, birth_date, first_name,
                          last_name, gender, hire_date)
     VALUES(2, '1990-01-01', 'Tyrion', 'Lannister', 'M', '2020-01-01');
    SAVEPOINT point1;

    INSERT INTO dept_manager (dept_no, emp_no,
                              from_date, to_date)
    VALUES (1, 1, '2020-01-01', '2021-01-01');

     IF err = 1 THEN
            ROLLBACK TO SAVEPOINT point1;
            SELECT 'An error occurred' AS message;
     ELSE
            COMMIT;
            SELECT 'OK' AS message;
     END IF;
END
```

Άσκηση 1: DB: emp

Κατασκευάστε stored procedure με όνομα insert_employee_with_salary

- Παίρνει ορίσματα τα: first_name, last_name, gender, birth_date, hire_date, salary, from_date, to_date
- Εισάγει μία νέα εγγραφή στον πίνακα employees
- Εισάγει μία νέα εγγραφή στον πίνακα salary
- Σε περίπτωση λάθους να γίνεται rollback αμέσως μετά την εισαγωγή στον πίνακα employees.

- Τα σημεία επαναφοράς είναι πραγματικά χρήσιμα σε πολύ μεγάλες stored procedure με ουσιαστικά διακριτές ενέργειες.

4. Προβλήματα Συγχρονισμού

- 4 συνηθισμένα προβλήματα συγχρονισμού (concurrency problems):
  - **Χαμένες Ενημερώσεις (Lost Updates):** Προκαλείται όταν δύο transactions ενημερώνουν την ίδια εγγραφή. Θα ισχύσει η ενημέρωση που έγινε τελευταία
  - **Βρώμικες Αναγνώσεις (Dirty Reads):**
    - Η Α αλλάζει μια γραμμή
    - Η Β επιλέγει τη γραμμή πριν η Α κάνει commit
    - Η Α κάνει rollback
    - Η Β έχει δεδομένα που δεν υπάρχουν στη βάση
  - **Διαφορετικές Αναγνώσεις (Nonrepeatable Reads):**
    - Δύο αναγνώσεις της ίδιας γραμμής επιστρέφουν διαφορετικά δεδομένα, επειδή μια άλλη συναλλαγή αλλάζει τα δεδομένα της γραμμής
  - **Phantom Reads:**
    - Η Α ενημερώνει μία στήλη σε όλο τον πίνακα
    - Η Β εισάγει μία νέα εγγραφή
    - Η Α στο πέρας της δεν έχει ενημερώσει όλο τον πίνακα.
- Τέτοια προβλήματα είναι αναμενόμενα σε πραγματικά συστήματα.
  - Η MySQL προσφέρει προστασία έναντι αυτων με τα **επίπεδα απομόνωσης (isolation levels)** των συναλλαγών.

Επίπεδα Απομόνωσης:

- **READ UNCOMMITTED:** Επιτρέπει τα πάντα
- **READ COMMITTED:** Αποτρέπει τις βρώμικες ενημερώσεις
- **REPEATABLE READS:** Δεν αποτρέπει τα phantom reads
- **SERIALIZABLE:** Αποτρέπει τα πάντα

Default είναι το REPEATABLE READS. Ωστόσο μπορούμε να το αλλάξουμε με το συντακτικό:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL level;
```

Παράδειγμα 7: DB: emp

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Γενικές Συμβουλές για τις Συναλλαγές:

- Οι συναλλαγές δεν πρέπει να είναι σύντομες και να μη μένουν ανοικτές για πολύ χρόνο.
- Τα SELECT καλό θα είναι να μένουν εκτός συναλλαγών
- Εξαιρετικά χρονοβόρες συναλλαγές (όπως π.χ. μαζικές ενημερώσεις), θα πρέπει να γίνονται σε χρόνο που το σύστημα έχει μικρό πλήθος ενεργών χρηστών (ώρες μη αιχμής).