CS 220
Database Systems

Fall 2019

Lecture 6

Triggers

When an action is performed on data, it is possible to check if the manipulation of the data concurs with the underlying business rules, and thus avoids erroneous entries in a table.

For example:

We might want to ship a free item to a client with the order, if it totals more than $1000. A trigger will be built to check the order total upon completion of the order, to see if an extra order line needs to be inserted.

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table.

Trigger Syntax

```
CREATE TABLE people (age INT, name varchar(150));

CREATE TRIGGER agecheck BEFORE INSERT ON people FOR
EACH ROW IF NEW.age < 0 THEN SET NEW.age = 0; END IF;

INSERT INTO people VALUES (-20, 'Sid'), (30, 'Josh');
```

Age	Name
0	Sid
30	Josh

Benefits of Using Triggers in Business

- Faster application development. Because the database stores triggers, you do not have to code the trigger actions into each database application.
- Global enforcement of business rules. Define a trigger once and then reuse it for any application that uses the database.
- Easier maintenance. If a business policy changes, you need to change only the corresponding trigger program instead of each application program.
- Improve performance in client/server environment. All rules run on the server before the result returns.

Transactions

A transaction is a logical unit of work that contains one or more SQL statements.

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for ensuring

1. either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database
2. or that the transaction does not have any effect on the database or any other transactions.

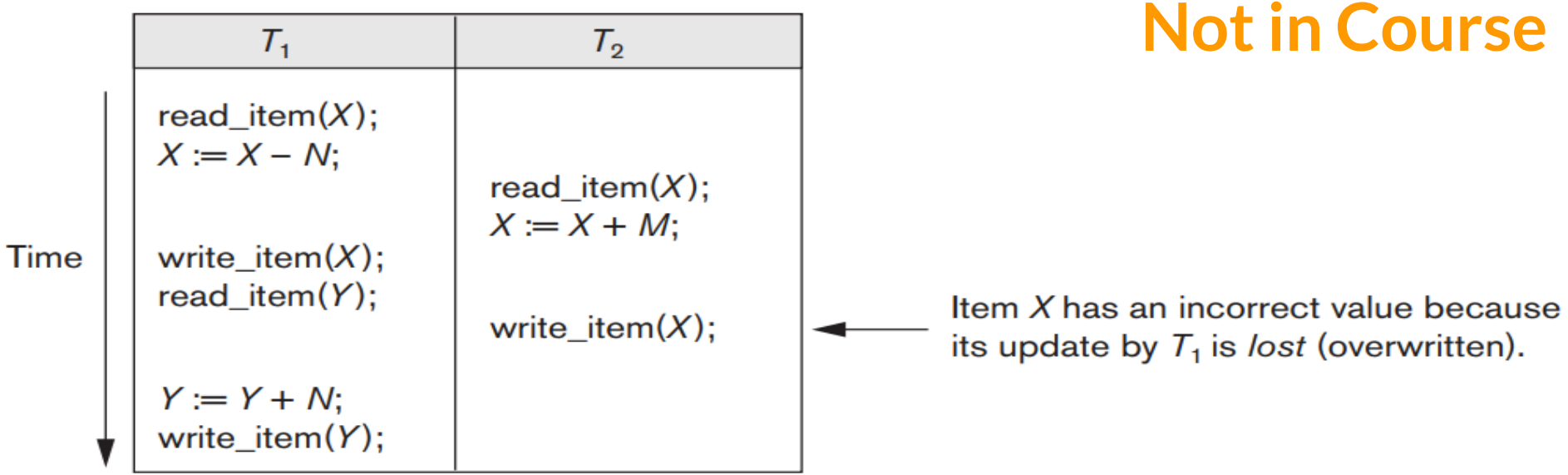
Why Transactions?

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

- **The Lost Update Problem**
- **The Temporary Update (or Dirty Read) Problem**
- **The Incorrect Summary Problem**
- **The Unrepeatable Read Problem**

The Lost Update Problem

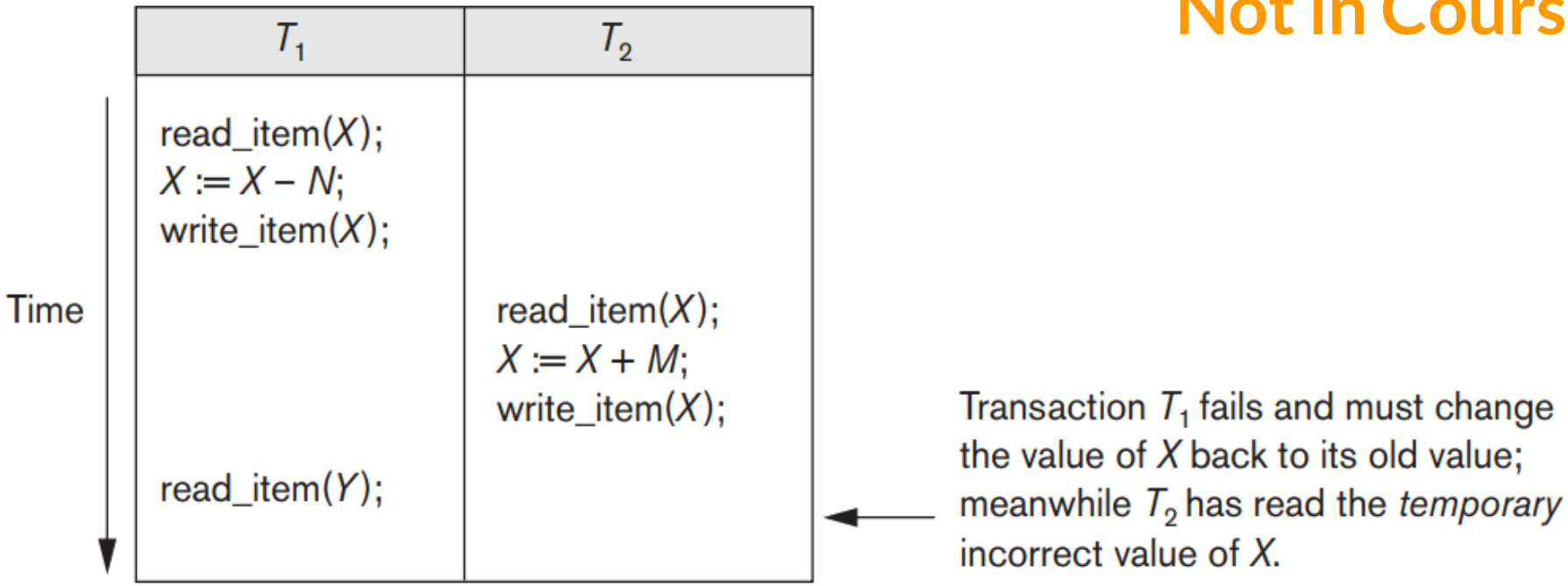
Additional Slides
Not in Course



This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.

The Temporary Update Problem

Additional Slides
Not in Course



This problem occurs when one transaction updates a database item and then the transaction fails for some reason.

The Incorrect Summary Problem

Additional Slides Not in Course

T_1	T_3
	$sum := 0$; read_item(A); $sum := sum + A$; . . .

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$sum := sum + X$;
	read_item(Y);
	$sum := sum + Y$;

read_item(Y);
$Y := Y + N$;
write_item(Y);	

T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

The Unrepeatable Read Problem

Additional Slides Not in Course

Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T′ between the two reads.

Properties of a Transaction

Transactions have the following four standard properties

- **Atomicity** − This ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
- **Consistency** − This ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation** − This enables transactions to operate independently on and transparent to each other.
- **Durability** − This ensures that the result or effect of a committed transaction persists in case of a system failure.

Properties of a Transaction

- When a successful transaction is completed, the **COMMIT** command should be issued so that the changes to all involved tables will take effect.
- If a failure occurs, a **ROLLBACK** command should be issued to return every table referenced in the transaction to its previous state.
    - The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not, because the whole transaction is a logical unit of database processing.
    - If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Transactions in MySQL

By default, MySQL runs with autocommit mode enabled

As you execute a statement table is updated

You can disable autocommit mode

```
SET AUTOCOMMIT=0;
```

Transactions in MySQL

After disabling autocommit mode

- ➢You must use **COMMIT** to store your changes to disk
- ➢Or **ROLLBACK** if you want to ignore the changes you have made since the beginning of your transaction.

```
START TRANSACTION;
    SELECT  /UPDATE  /DELETE
COMMIT;
```

*DDL commands can not be rolled back

Sub Queries

SQL allows a single query to have multiple subqueries nested inside of it.

This allows for more complex queries to be written.

When queries are nested, the outer statement determines the contents of the final result, while the inner SELECT statements are used by the outer statement (often to lookup values for WHERE clauses).

Subqueries can be categorized into two types

- A non-correlated (simple) subquery

Non-correlated Subqueries

A noncorrelated subquery executes independently of the outer query.

The subquery executes first

and then passes its results to the outer query

```
SELECT name, street, city, state FROM addresses
    WHERE state IN (SELECT state FROM states);
```

Three types of non-correlated Subqueries

- Scalar subqueries
- Multiple Row subquery
- Table subquery

Scalar Subqueries/ Single Row Subquery

Returns a single value.

Often value is then used in a comparison.

*If query is written so that it expects a subquery to return a single value, and if it returns multiple values or no values, a run-time error occurs.

Return the employees that are in the 'Accounting' department:

```
SELECT ename
FROM   emp
WHERE  dno = (SELECT dno FROM dept
              WHERE dname = 'Accounting')
```

Multiple Row Subqueries

Returns a multiple rows with one column.

Return all employees who work more hours than average on a single project:

```
SELECT ename
FROM   emp, workson
WHERE  workson.eno = emp.eno AND
       workson.hours > (SELECT AVG(hours) FROM workson)
```

Table Subqueries

Returns multiple rows with one or more columns.

```
SELECT * FROM staff
    WHERE (name,age) = (
        SELECT name,age FROM customer WHERE name='Valerius'
        );

SELECT name,age FROM staff
    WHERE (name,age) IN (
        SELECT name,age FROM customer
        );
```

More on Table Subqueries

A table subquery returns a relation. There are several operators that can be used:

- `EXISTS` *R* - true if *R* is not empty
- *s* `IN` *R* - true if *s* is equal to one of the values of *R*
- *s* > `ALL` *R* - true if *s* is greater than **every** value in *R*
- *s* > `ANY` *R* - true if *s* is greater than **any** value in *R*

Notes:

- 1) Any of the comparison operators (<, <=, =, etc.) can be used.
- 2) The keyword `NOT` can proceed any of the operators.
  - ⇨ Example: *s* `NOT IN` *R*

Using Any or All

Example: Return the employees who make more than all the employees with title 'ME' make.

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary FROM emp
                     WHERE title = 'ME')
```

Subquery Important Rules

- The ORDER BY clause may not be used in a subquery.
- The number of attributes in the SELECT clause in the subquery must match the number of attributes compared to with the comparison operator.

Correlated Subqueries

A nested query is correlated with the outside query if

- ➢ it must be re-computed for every tuple produced by the outside query.
- ➢ it contains a reference to an attribute in the outer query.

Return all employees who have the same name as another employee:

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT eno FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

Correlated Subqueries

Outer query needs to be executed before inner query.

An important point to note is that correlated subqueries are slower queries and one should avoid it as much as possible.

```
SELECT employee_id, manager_id,first_name, last_name
   FROM employees a
   WHERE EXISTS
      (SELECT employeeid
         FROM employees b
         WHERE b.manager_id = a.employee_id)
```

Correlated Subqueries - Another Example

Find all employees whose salary is above average for their department

```
SELECT employee_number, name
 FROM employees emp WHERE salary > (
     SELECT AVG(salary) FROM employees
         WHERE department = emp.department);
```

Summarizing Subqueries

- **Correlated subquery:** inner query depends on outer query
- **Non-correlated query:** inner query or subquery doesn't depends on outer query and run by its own.
- **Correlated subquery:** outer query executed before inner query or subquery
- **Non-correlated query:** inner query executes before outer query.
- Correlated Sub-queries are slower than non correlated subquery and should be avoided in favor of sql joins.

SQL Functions - Google Time

Databases have many built-in functions that can be used when writing queries. Syntax and support varies between systems.

- Date: DATEDIFF, YEAR, GETDATE
- String: CONCAT, UPPER, LEFT, SUBSTRING
- Logical: CASE, IIF, ISNULL
- Aggregate: SUM, COUNT, AVG
- Note: Case-insensitive function names.

Summarizing SQL Query

The general form of the `SELECT` statement is:

SELECT <attribute list>
FROM <table list>
[WHERE (*condition*)]
[GROUP BY <grouping attributes>]
[HAVING <group condition>]
[ORDER BY <attribute list>]
[LIMIT <num> [OFFSET <offset>]]

- Clauses in square brackets ([,]) are optional.
- There are often numerous ways to express the same query in SQL.

Data Control Language

DCL- Data Control Language

DCL commands are used to enforce database security in a multiple user database environment

Two types of DCL commands are GRANT and REVOKE.

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

SQL Privileges

A privilege corresponds to the right to use certain SQL statements such as SELECT, INSERT, etc. on one or more database objects

Privilege	Explanation
SELECT	Provides retrieval privilege
INSERT	Gives insert privilege
UPDATE	Gives update privilege
DELETE	Gives delete privilege

Privileges

- **GRANT** SELECT, INSERT, UPDATE, DELETE
  ON SUPPLIER **TO** user1
- **GRANT** SELECT (PRODNR, PRODNAME)
  ON PRODUCT **TO** user1
- **REVOKE** DELETE
  ON SUPPLIER **FROM** user1

QUESTIONS ???