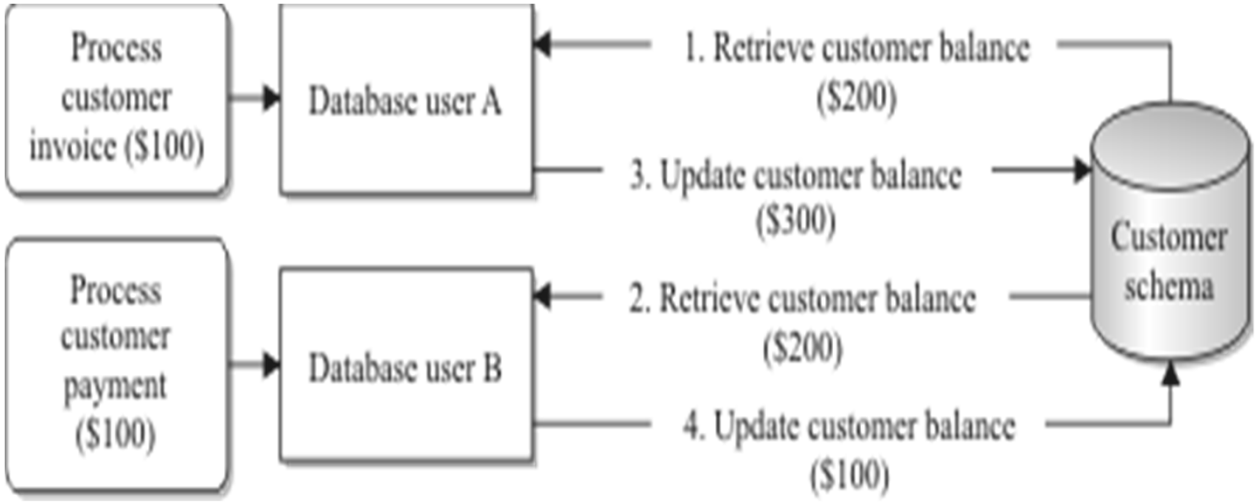
Transaction Management

Chapter 11

The Concurrent Update Problem



- To prevent errors from being introduced when concurrent updates are attempted, the application logic must incorporate the concept of a transaction.

Transaction Management

- *Transaction:* a discrete series of actions that must be either completely processed or not processed at all.
  - Also called a *Unit of Work*
- Predates RDBMSs – transactions were formally supported in many hierarchical and network DBMSs
  - Freely sharing data online led to the concurrent update problem
- Some RDBMSs support nested transactions (e.g. Microsoft SQL Server), while others (e.g. Oracle) do not.

Transaction Properties (ACID)

- *Atomicity:* must remain whole; must completely succeed or completely fail
  - *Rollback*: reverses all changes made by a transaction
  - *Commit*: makes changes made by a transaction permanent
- *Consistency:* should transform database from one consistent state to another
- *Isolation:* should carry out its work independent of any other transaction
- *Durability:* changes made by completed transactions should be permanently stored (*persistent* in OO terminology)

Transaction Support in SQL

- ANSI/ISO SQL Standard specifies the syntax and behavior of SQL transactions
  - RDBMS vendors implemented transactions prior to the development of the SQL Standard, so there are a number of departures from the standard.
  - SQL Server and Oracle are presented here as exceptions
  - MySQL conforms closely to the standard

Standard SQL Transaction Syntax

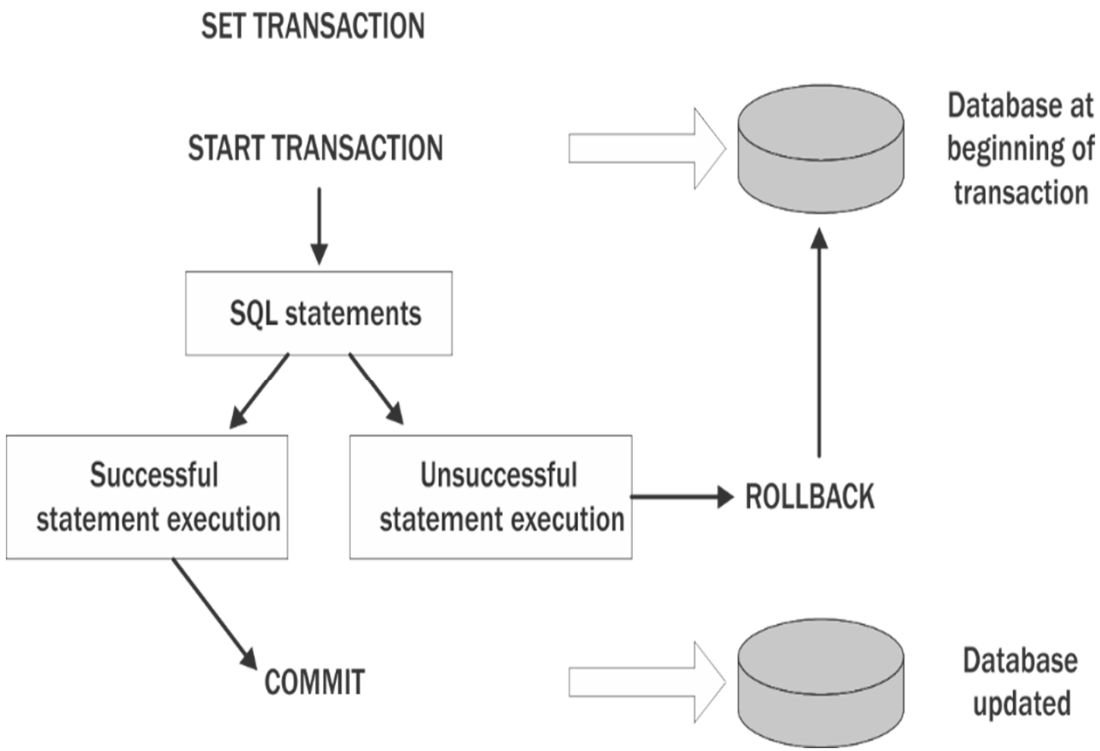
- Seven Transaction Control Statements:
  - **SET TRANSACTION** Sets the properties of the next transaction to be executed
  - **START TRANSACTION** Sets the properties of a transaction and starts the transaction
  - **SET CONSTRAINTS** Sets the constraint mode within a current transaction (controls whether constraints are applied immediates or deferred until the commit point)
  - **SAVEPOINT** Creates a savepoint within a transaction. A ***savepoint*** marks a place within the transaction that acts as a stopping point when you roll back a transaction.

Standard SQL Transaction Syntax

- Seven Transaction Control Statements (continued):
  - **RELEASE SAVEPOINT** Releases a savepoint
  - **ROLLBACK** Terminates a transaction and rolls back any changes to the beginning of the transaction or to a savepoint
  - **COMMIT** Terminates a transaction and commits all changes to the database

  - SET TRANSACTION is intended for implicitly started transactions, while START TRANSACTION explicitly starts a transaction. Both forms set transaction properties.

How SQL Transaction Statements Work Together



Data Phenomena

- **Dirty Reads**
  - Can occur when:
    - One transaction modifies data
    - A second transaction sees those modifications before they're actually committed to the database
    - The first transaction rolls back the modifications, returning the database to its original state.
    - However, the second transaction, having read the modified data, might have taken action based on the incorrect data.

Data Phenomena

- **Nonrepeatable Reads**
  - Occur when:
    - One transaction reads data from a table,
    - Another transaction then updates that data
    - The first transaction rereads the data, only to discover that the data has changed.
  - As a result, the first read is not repeatable.

Data Phenomena

- **Phantom Reads:**
  - Can occur when:
    - A transaction reads a table based on some sort of search condition, then
    - A second transaction updates the data in the table
    - Then the first transaction attempts to reread the data, only this time different rows are returned because of how the search condition is defined.
    - Phantom read refers to whether a subsequent read returns the same rows (as opposed to having rows added or removed from the results of the subsequent read)
    - Nonrepeatable read refers to whether the rows returned have different column data than the prior read of the same rows of data.

Data Phenomena

- **Phantom Reads vs. Nonrepeatable Read**
  - Phantom read refers to whether a subsequent read returns the same rows
    - (as opposed to having rows added or removed from the results of the subsequent read)
  - Nonrepeatable read refers to whether the rows returned have different column data than the prior read of the same rows of data.

Transaction ISOLATION LEVEL Controls Data Phenomena

ISOLATION LEVEL	Dirty Read	Nonrepeatable Read	Phantom Read
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

- Tradeoff: the more restrictive the isolation level, the higher the performance penalty

Examples of SQL Transaction Statements

```
START TRANSACTION
      ISOLATION LEVEL SERIALIZABLE;

UPDATE EMPLOYEE
   SET SALARY = SALARY * 3.5;
-- intending to raise salaries by 3.5%

SELECT SUM(SALARY)
  FROM EMPLOYEE;
-- User sees the new sum and recognizes the error.

ROLLBACK;
-- Rolls back the previous changes and ends trans.
```

Examples of SQL Transaction Statements

```
START TRANSACTION
      ISOLATION LEVEL SERIALIZABLE;

UPDATE EMPLOYEE
   SET SALARY = SALARY * .035;

SELECT SUM(SALARY)
  FROM EMPLOYEE;
-- User sees the new sum is correct.

COMMIT;
-- Commits the changes and ends transaction
```

Locking Mechanisms

- *Lock:* a control placed in the database to reserve data so that only one database session may update it
- Locks normally placed by SQL DML (INSERT, UPDATE, DELETE) statements
- Locks normally released with COMMIT and ROLLBACK

Locking Mechanisms (Cont.)

- *Lock Granularity:* the amount of data held by a lock (see next slide)
- *Lock Escalation:* raises locks to higher levels of granularity as the database session invokes more and more locks
- Many RDBMSs have parameters that control locking behavior

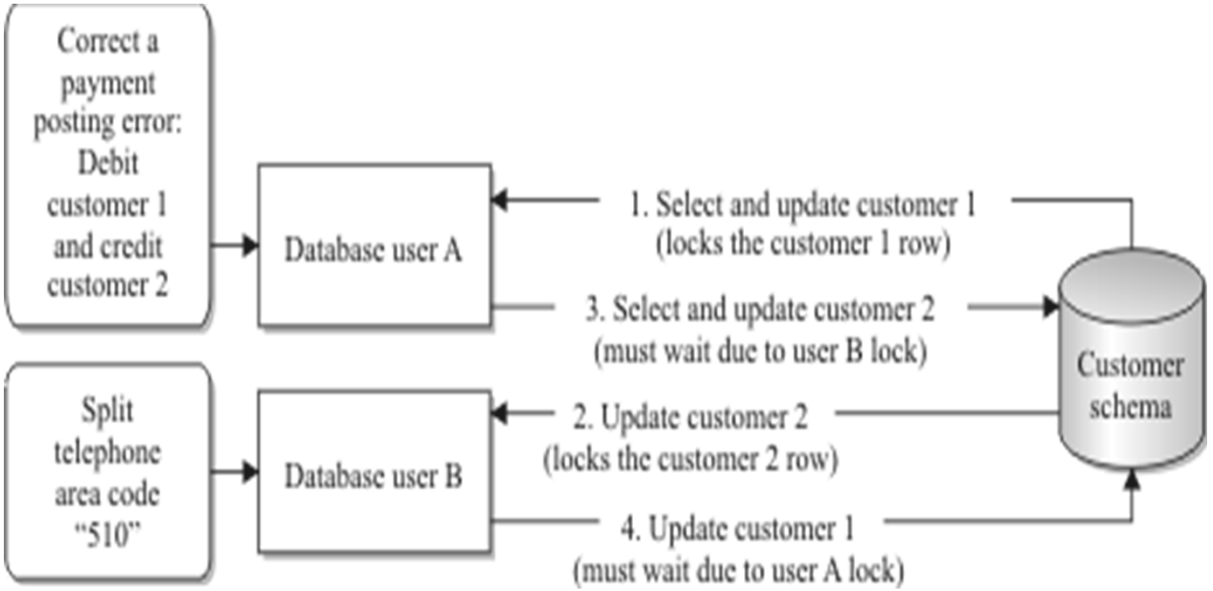
Lock Granularity

- Database
- File
- Block or Page
- Row
- Column

Locks: A Double-Edged Sword

- Locks solve the concurrent update problem by preventing two sessions from executing conflicting changes to the same data
- Locks also cause contention as sessions must wait for locked resources to be freed
- *Deadlock:* a situation where two or more database sessions have locked some data and each then requests a lock on data that the other has already locked

The Deadlock



Transaction Support in Microsoft SQL Server

- Autocommit Mode (default behavior): each SQL statement is automatically committed as it completes
- Explicit Mode:
  - Transaction started with BEGIN TRANSACTION
  - Transaction ended successfully with COMMIT
  - Transaction ended unsuccessfully with ROLLBACK
- Implicit Mode:
  - Mode set with SET IMPLICIT_TRANSACTIONS ON
  - Transaction started implicitly by certain SQL statements
  - Once started, transaction continues until a COMMIT or ROLLBACK

Transaction Support in Oracle

- Implicit Mode (default behavior):
  - Start of database session starts a transaction
  - Transaction continues until:
    - COMMIT issued
    - ROLLBACK issued
    - Database session ends
    - DDL statement issued
- Autocommit Mode:
  - SET AUTOCOMMIT ON
  - SET AUTOCOMMIT OFF

Logical Locking for Database Applications

- Database locking does not work when web applications are involved
  - Most web applications are stateless
  - Users can start a transaction and abandon it, leaving resources locked for a long time
- In this situation, the application layer must handle locking in a logical manner
  - Usually known as ***application locking***
  - Works even in AUTOCOMMIT mode

Last Update Timestamp Method

- Table design includes LAST_UPDATE column
  - TIMESTAMP with precision of 4 or more

EMPLOYEE ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION	LAST_UPDATE
145	John	Russell	14000	0.4	2016-01-05 15:32:00.2341
146	Karen	Partners	13500	0.3	2016-01-10 08:22:15.8472
147	Alberto	Errazuriz	12000	0.3	2016-02-22 01:24:00.7463
148	Gerard	Cambrault	11000	0.3	2016-03-01 10:45:53.0185
149	Eleni	Alotkey	10500	0.2	2016-02-22 01:23:12.5043
150	Peter	Tucker	10000	0.3	2016-02-22 01:23:12.6843
151	David	Bernstein	9500	0.25	2016-03-01 10:45:53.1985
152	Peter	Hall	9000	0.25	2016-01-05 16:08:01.2341
153	Christopher	Olsen	8000	0.2	2016-01-10 08:16:55.5602

Last Update Timestamp Method

- Implementation Rules (Every Application Must Follow):
  - Any select with an intent to update the table must include the LAST_UPDATE column.
  - Any update of a row in the table must also set the LAST_UPDATE column to the current timestamp.
    - SQL provides pseudo column CURRENT_TIMESTAMP
  - Any update or delete must include a WHERE clause predicate that compares the current value of the LAST_UPDATE column to the last value selected for the LAST_UPDATE column for the same row.
    - Updated row will cause a NOT FOUND condition.

Last Update Timestamp Method

- SQL Example:

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, LAST_UPDATE
  FROM EMPLOYEE
  INTO :EmpID, :FName, :LName, :LUpdate
 WHERE EMPLOYEE_ID = 146;

UPDATE EMPLOYEE
   SET LAST_NAME   = 'Chang',
       LAST_UPDATE = CURRENT_TIMESTAMP
 WHERE EMPLOYEE_ID = :EmpID    -- predicate on the primary key column
   AND LAST_UPDATE = :LUpdate;   -- predicate on the last update column
```

Versioning Method

- Uses ROW_VERSION (typically INTEGER) instead of LAST_UPDATE (TIMESTAMP):

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, ROW_VERSION
  FROM EMPLOYEE
  INTO :EmpID, :FName, :LName, :RowVersion
 WHERE EMPLOYEE_ID = 146;

UPDATE EMPLOYEE
   SET LAST_NAME   = 'Chang',
       ROW_VERSION = ROW_VERSION + 1
 WHERE EMPLOYEE_ID = :EmpID
   AND ROW_VERSION = :RowVersion;
```

Session ID Method

- Relies on a unique session ID
  - Value provided by the application or database server
  - Table includes a column where Session ID can be written

EMPLOYEE ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION	LOCK_SESSION_ID
145	John	Russell	14000	0.4	
146	Karen	Partners	13500	0.3	
147	Alberto	Errazuriz	12000	0.3	847389
148	Gerard	Cambrault	11000	0.3	
149	Eleni	Alotkey	10500	0.2	
150	Peter	Tucker	10000	0.3	
151	David	Bernstein	9500	0.25	
152	Peter	Hall	9000	0.25	
153	Christopher	Olsen	8000	0.2	

Session ID Method

- Every Application Must Follow These Rules:
  - Any update to the table must check (with a WHERE clause predicate) to be sure that the LOCK_SESSION_ID column is either null or contains the session number of the current database session.
  - Any database session (connection) that intends to update the table must update the row first to place the session ID in the LOCK_SESSION_ID column.
    - Even this update must check the column before updating it.

Session ID Method

- Rules (Continued):
  - The row data may be read (with the intent to apply updates) only after the LOCK_SESSION_ID update has been applied.
  - The update of the business data in the table row must also clear the Session ID column
    - Logically releasing the application lock) by setting the column value to NULL.
    - This update must also include a WHERE clause predicate that compares the current value of the LOCK_SESSION_ID column to the current session ID.
    - If row not locked by current session, NOT FOUND results.

Session ID Method Example

```
UPDATE EMPLOYEE
   SET LOCK_SESSION_ID = @@SPID
 WHERE EMPLOYEE_ID = 146
   AND LOCK_SESSION_ID IS NULL;
-- row not found if already locked

SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME
  FROM EMPLOYEE
  INTO @EmpID, @FName, @LName
 WHERE EMPLOYEE_ID = 146
   AND LOCK_SESSION_ID = @@SPID;
-- row not found if no session lock
```

Session ID Method Example

```
UPDATE EMPLOYEE
   SET LAST_NAME       = 'Chang',
       LOCK_SESSION_ID = null
-- clears session lock
 WHERE EMPLOYEE_ID = @EmpID
-- predicate on the primary key column
   AND LAST_UPDATE = @@SPID;
-- not found if no session lock
```