IM Ch10 Trans Mgt and Concurrency Ctrl Ed12

Database Systems (Charles Sturt University)

Chapter 10

Transaction Management and Concurrency Control

Discussion Focus

Why does a multi-user database environment give us a special interest in transaction management and concurrency control?

Begin by exploring what a transaction is, what its components are, and why it must be managed carefully even in a single-user database environment. Then explain why a multi-user database environment makes transaction management even more critical.

Emphasize the following points:

- A transaction *represents a real-world event* such as the sale of a product.
- A transaction must be a *logical unit of work*. That is, no portion of a transaction stands by itself. For example, the product sale has an effect on inventory and, if it is a credit sale, it has an effect on customer balances.
- A transaction must *take a database from one consistent state to another*. Therefore, all parts of a transaction must be executed or the transaction must be aborted. (A consistent state of the database is one in which all data integrity constraints are satisfied.)

All transactions have four properties: **A**tomicity, **C**onsistency, **I**solation, and **D**urability. (These four properties are also known as the **ACID** test of transactions.) In addition, multiple transactions must conform to the property of serializability. Table IM10.1 provides a good summary of transaction properties.

Table IM10.1 Transaction Properties.

Multi-user Databases	**Single-user Databases**	**atomicity**: Unless all parts of the executed, the transaction is aborted
		consistency. Indicates the permanence of the database's consistent state.
		durability: Once a transaction is committed, it cannot be rolled back
		isolation: Data used by one transaction cannot be used by another transaction until the first transaction is completed.
		serializability: The result of the concurrent execution of transactions is the same as though the transactions were executed in serial order.

Note that SQL provides transaction support through

COMMIT (permanently saves changes to disk)

and

ROLLBACK (restores the previous database state)

Each SQL transaction is composed of several *database requests*, each one of which yields I/O operations. A *transaction log* keeps track of all transactions that modify the database. The transaction log data may be used for recovery (ROLLBACK) purposes.

Next, explain that concurrency control is the management of *concurrent* transaction execution. (Therefore, a single-user database does not require concurrency control!) Specifically, explore the concurrency control issues concerning lost updates, uncommitted data, and inconsistent databases. Note that multi-user DBMSs use a process known as a *scheduler* to enforce concurrency control.

Since concurrency control is made possible through the use of *locks*, examine the various levels and types of locks. Because the use of locks creates a possibility of producing *deadlocks*, discuss the detection, prevention, and avoidance strategies that enable the system to manage deadlock conditions.

Answers to Review Questions

1. Explain the following statement: a transaction is a logical unit of work.

A transaction is a logical unit of work that must be entirely completed of aborted; no intermediate states are accepted. In other words, a transaction, composed of several database requests, is treated by the DBMS as a *unit* of work in which all transaction steps must be fully completed if the transaction is to be accepted by the DBMS.

Acceptance of an incomplete transaction will yield an inconsistent database state. To avoid such a state, the DBMS ensures that all of a transaction's database operations are completed before they are committed to the database. For example, a credit sale requires a minimum of three database operations:

1. An invoice is created for the sold product.
2. The product's inventory quantity on hand is reduced.
3. The customer accounts payable balance is increased by the amount listed on the invoice.

If only parts 1 and 2 are completed, the database will be left in an inconsistent state. Unless all three parts (1, 2, and 3) are completed, the entire sales transaction is canceled.

2. **What is a consistent database state, and how is it achieved?**

   A consistent database state is one in which all data integrity constraints are satisfied. To achieve a consistent database state, a transaction must take the database from one consistent state to another. (See the answer to question 1.)

3. **The DBMS does not guarantee that the semantic meaning of the transaction truly represents the real-world event. What are the possible consequences of that limitation? Give an example.**

   The database is designed to verify the syntactic accuracy of the database commands given by the user to be executed by the DBMS. The DBMS will check that the database exists, that the referenced attributes exist in the selected tables, that the attribute data types are correct, and so on. Unfortunately, the DBMS is not designed to guarantee that the syntactically correct transaction accurately represents the real-world event.

   For example, if the end user sells 10 units of product 100179 (Crystal Vases), the DBMS cannot detect errors such as the operator entering 10 units of product 100197 (Crystal Glasses). The DBMS will execute the transaction, and the database will end up in a *technically consistent state* but in a *real-world inconsistent state* because the wrong product was updated.

4. **List and discuss the five transaction properties.**

   The five transaction properties are:

Atomicity	requires that *all* parts of a transaction must be completed or the transaction is aborted. This property ensures that the database will remain in a consistent state.
Consistency	Indicates the permanence of the database consistent state**.**
Isolation	means that the data required by an executing transaction cannot be accessed by any other transaction until the first transaction finishes. This property ensures data consistency for concurrently executing transactions.
Durability	indicates that the database will be in a *permanent* consistent state after the execution of a transaction. In other words, once a consistent state is reached, it cannot be lost.
Serializability	means that a series of concurrent transactions will yield the same result as if they were executed one after another.

   All five transaction properties work together to make sure that a database maintains data integrity and consistency for either a single-user or a multi-user DBMS.

5. What does serializability of transactions mean?

Serializability of transactions means that a series of concurrent transactions will yield the same result as if they were executed one after another

6. What is a transaction log, and what is its function?

The transaction log is a special DBMS table that contains a description of all the database transactions executed by the DBMS. The database transaction log plays a crucial role in maintaining database concurrency control and integrity.

The information stored in the log is used by the DBMS to recover the database after a transaction is aborted or after a system failure. The transaction log is usually stored in a different hard disk or in a different media (tape) to prevent the failure caused by a media error.

7. What is a scheduler, what does it do, and why is its activity important to concurrency control?

The scheduler is the DBMS component that establishes the order in which concurrent database operations are executed. The scheduler interleaves the execution of the database operations (belonging to several concurrent transactions) to ensure the *serializability* of transactions. In other words, the scheduler guarantees that the execution of concurrent transactions will yield the same result as though the transactions were executed one after another. The scheduler is important because it is the DBMS component that will ensure transaction serializability. In other words, the scheduler allows the concurrent execution of transactions, giving end users the impression that they are the DBMS's only users.

8. What is a lock, and how, in general, does it work?

A lock is a mechanism used in concurrency control to guarantee the exclusive use of a data element to the transaction that owns the lock. For example, if the data element X is currently locked by transaction T1, transaction T2 will not have access to the data element X until T1 releases its lock.

Generally speaking, a data item can be in only two states: locked (being used by some transaction) or unlocked (not in use by any transaction). To access a data element X, a transaction T1 first must request a lock to the DBMS. If the data element is not in use, the DBMS will lock X to be used by T1 exclusively. No other transaction will have access to X while T1 is executed.

9. What are the different levels of lock granuality?

Lock granularity refers to the size of the database object that a single lock is placed upon. Lock granularity can be:

Database-level, meaning the entire database is locked by one lock.
Table-level, meaning a table is locked by one lock.
Page-level, meaning a diskpage is locked by one lock.
Row-level, meaning one row is locked by one lock.
Field-level, meaning one field in one row is locked by one lock.

10. Why might a page-level lock be preferred over a field-level lock?

Smaller lock granualarity improves the concurrency of the database by reducing contention to lock database objects. However, smaller lock granularity also means that more locks must be maintained and managed by the DBMS, requiring more processing overhead and system resources for lock management. Concurrency demands and system resource usage must be balanced to ensure the best overall transaction performance. In some circumstances, page-level locks, which require fewer system resources, may produce better overall performance than field-level locks, which require more system resources.

11. What is concurrency control, and what is its objective?

Concurrency control is the activity of coordinating the simultaneous execution of transactions in a multiprocessing or multi-user database management system. The objective of concurrency control is to ensure the serializability of transactions in a multi-user database management system. (The DBMS's scheduler is in charge of maintaining concurrency control.)

Because it helps to guarantee data integrity and consistency in a database system, concurrency control is one of the most critical activities performed by a DBMS. If concurrency control is not maintained, three serious problems may be caused by concurrent transaction execution: lost updates, uncommitted data, and inconsistent retrievals.

12. What is an exclusive lock, and under what circumstances is it granted?

An exclusive lock is one of two lock types used to enforce concurrency control. (A lock can have three states: unlocked, shared (read) lock, and exclusive (write) lock. The "shared" and "exclusive" labels indicate the nature of the lock.)

An exclusive lock exists when access to a data item is specifically reserved for the transaction that locked the object. The exclusive lock must be used when a potential for conflict exists, e.g., when one or more transactions must update (WRITE) a data item. Therefore, an exclusive lock is issued only when a transaction must WRITE (update) a data item *and no locks are currently held on that data item by any other transaction*.

To understand the reasons for having an exclusive lock, look at its counterpart, the shared lock. Shared locks are appropriate when concurrent transactions are granted READ access on the basis of a common lock, because concurrent transactions based on a READ cannot produce a conflict.

A shared lock is issued when a transaction must read data from the database *and no exclusive locks are held* on the data to be read.

13. What is a deadlock, and how can it be avoided? Discuss several strategies for dealing with deadlocks.

Base your discussion on Chapter 10's Section 10-3d, Deadlocks. Start by pointing out that, although locks prevent serious data inconsistencies, their use may lead to two major problems:

1. The transaction schedule dictated by the locking requirements may not be serializable, thus causing data integrity and consistency problems.
2. The schedule may create *deadlocks*. Database deadlocks are the equivalent of a traffic gridlock in a big city and are caused by two transactions waiting for each other to unlock data.

Use Table 10.13 in the text to illustrate the scenario that leads to a deadlock.) The table has been reproduced below for your convenience.

TABLE 10.13 How a Deadlock Condition is Created

TIME	TRANSACTION	REPLY	LOCK STATUS		
0			**Data X**		**Data Y**
1	T1:LOCK(X)	OK	Unlocked		Unlocked
2	T2: LOCK(Y)	OK	Locked		Unlocked
3	T1:LOCK(Y)	WAIT	Locked		Locked
4	T2:LOCK(X)	WAIT	Locked	**Deadlock**	Locked
5	T1:LOCK(Y)	WAIT	Locked		Locked
6	T2:LOCK(X)	WAIT	Locked		Locked
7	T1:LOCK(Y)	WAIT	Locked		Locked
8	T2:LOCK(X)	WAIT	Locked		Locked
9	T1:LOCK(Y)	WAIT	Locked		Locked
…	……..	……..	……		……..…
…	……..	……..	……		……..…
…	……..	……..	……		……..…
…	……..	……..	……		……

Deadlock

In a real world DBMS, many more transactions can be executed simultaneously, thereby increasing the probability of generating deadlocks. Note that deadlocks are possible only if one of the transactions wants to obtain an exclusive lock on a data item; no deadlock condition can exist among shared locks.

Three basic techniques exist to control deadlocks:

Deadlock Prevention

A transaction requesting a new lock is aborted if there is a possibility that a deadlock may occur. If the transaction is aborted, all the changes made by this transaction are rolled back and all locks are released. The transaction is then re-scheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.

Deadlock Detection

The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and rescheduled) and the other transaction continues. Note particularly the discussion in Section 10-4a, Wait/Die and Wound/Wait Schemes.

Deadlock Avoidance

The transaction must obtain all the locks it needs before it can be executed. This technique avoids rollback of conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases the response times.

The best deadlock control method depends on the database environment. For example, if the probability of deadlocks is low, deadlock detection is recommended. However, if the probability of deadlocks is high, deadlock prevention is recommended. If response time is not high on the system priority list, deadlock avoidance may be employed.

14. What are some disadvantages of time-stamping methods for concurrency control?

The disadvantages are: 1) each value stored in the database requires two additional time stamp fields – one for the last time the field was read and one for the last time it was updated, 2) increased memory and processing overhead requirements, and 3) many transactions may have to be stopped, rescheduled, and restamped.

15. Why might it take a long time to complete transactions when an optimistic approach to concurrency control is used?

Because the optimistic approach makes the assumption that conflict from concurrent transactions is unlikely, it does nothing to avoid conflicts or control the conflicts. The only test for conflict occurs during the validation phase. If a conflict is detected, then the entire transaction restarts. In an environment with few conflicts from concurrency, this type of single checking scheme works well. In an environment where conflicts are common, a transaction may have to be restarted numerous times before it can be written to the database.

16. What are the three types of database critical events that can trigger the database recovery process? Give some examples for each one.

Backup and recovery functions constitute a very important component of today's DBMSs. Some DBMSs provide functions that allow the database administrator to perform and schedule automatic database backups to permanent secondary storage devices, such as disks or tapes.

Critical events include:

- *Hardware/software failures*. hard disk media failure, a bad capacitor on a motherboard, or a failing memory bank. Other causes of errors under this category include application program or operating system errors that cause data to be overwritten, deleted, or lost.
- *Human-caused incidents*. This type of event can be categorized as unintentional or intentional.
    - An unintentional failure is caused by carelessness by end-users. Such errors include deleting the wrong rows from a table, pressing the wrong key on the keyboard, or shutting down the main database server by accident.

  - Intentional events are of a more severe nature and normally indicate that the company data are at serious risk. Under this category are security threats caused by hackers trying to gain unauthorized access to data resources and virus attacks caused by disgruntled employees trying to compromise the database operation and damage the company.
- *Natural disasters*. This category includes fires, earthquakes, floods, and power failures.

17. What are the four ANSI transaction isolation levels? What type of reads does each level allow?

The four ANSI transaction isolation levels are 1) read uncommitted, 2) read committed, 3) repeatable read, and 4) Serializable. These levels allow different "questionable" reads. A read is questionable if it can produce inconsistent results. Read uncommitted isolation will allow dirty reads, non-repeatable reads and phantom reads. Read committed isolation will allow non-repeatable reads and phantom reads. Repeatable read isolation will allow phantom reads. Serializable does not allow any questionable reads.

Problem Solutions

1. Suppose you are a manufacturer of product ABC, which is composed of parts A, B, and C. Each time a new product is created, it must be added to the product inventory, using the PROD_QOH in a table named PRODUCT. And each time the product ABC is created, the parts inventory, using PART_QOH in a table named PART, must be reduced by one each of parts A, B, and C. The sample database contents are shown in Table P10.1

Table P10.1 The Database for Problem 1

Table name: PRODUCT

PROD_CODE	PROD_QOH
ABC	1,205

Table name: PART

PART_CODE	PART_QOH
A	567
B	98
C	549

Given that information, answer Questions a through e.

a. How many database requests can you identify for an inventory update for both PRODUCT and PART?

Depending in how the SQL statements are written, there are two correct answers: 4 or 2.

b. Using SQL, write each database request you identified in step a.

The database requests are shown in the following table.

Four SQL statements	Two SQL statements
UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC'	UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC'

UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'B' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'C'	UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' OR PART_CODE = 'B' OR PART_CODE = 'C'

c. Write the complete transaction(s).

The transactions are shown in the following table.

Four SQL statements	Two SQL statements
BEGIN TRANSACTION UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'B' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'C' COMMIT;	BEGIN TRANSACTION UPDATE PRODUCT SET PROD_QOH = PROD_OQH + 1 WHERE PROD_CODE = 'ABC' UPDATE PART SET PART_QOH = PART_OQH - 1 WHERE PART_CODE = 'A' OR PART_CODE = 'B' OR PART_CODE = 'C' COMMIT;

d. Write the transaction log, using Table 10.1 as your template.

We assume that product 'ABC' has a PROD_QOH = 23 at the start of the transaction and that the transaction is representing the addition of 1 new product. We also assume that PART components "A", "B" and "C" have a PROD_QOH equal to 56, 12, and 45 respectively.

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
1	1A3	NULL	2	START	**START TRANSACTION				
2	1A3	1	3	UPDATE	PRODUCT	'ABC'	PROD_QOH	23	24
3	1A3	2	4	UPDATE	PART	'A'	PART_QOH	56	55
4	1A3	3	5	UPDATE	PART	'B'	PART_QOH	12	11
5	1A3	4	6	UPDATE	PART	'C'	PART_QOH	45	44
6	1A3	5	NULL	COMMIT	** END TRANSACTION				

e. Using the transaction log you created in Step d, trace its use in database recovery.

Begin with the last trl_id (trl_id 6) for the transaction (trx_num 1A3) and work backward using the prev_ptr to identify the next step to undo moving from the end of the transaction back to the beginning.
Trl_ID 6: Nothing to change because it is a end of transaction marker.
Trl_ID 5: Change PART_QOH from 44 to 45 for ROW_ID 'C' in PART table.
Trl_ID 4: Change PART_QOH from 11 to 12 for ROW_ID 'B' in PART table.
Trl_ID 3: Change PART_QOH from 55 to 56 for ROW_ID 'A' in PART table.
Trl_ID 2: Change PROD_QOH from 24 to 23 for ROW_ID 'ABC' in PRODUCT table.
Trl_ID 1: Nothing to change because it is a beginning of transaction marker.

2. Describe the three most common problems with concurrent transaction execution. Explain how concurrency control can be used to avoid those problems.

The three main concurrency control problems are triggered by *lost updates, uncommitted data*, and *inconsistent retrievals*. These control problems are discussed in detail in Section 10-2. Note particularly Section 10-2a, Lost Updates, Section 10-2b, Uncommitted Data, and Section 10-2c, Inconsistent Retrievals.

3. What DBMS component is responsible for concurrency control? How is this feature used to resolve conflicts?

Severe database integrity and consistency problems can arise when two or more concurrent transactions are executed. In order to avoid such problems, the DBMS must exercise concurrency control. The DBMS's component in charge of concurrency control is the *scheduler*. The scheduler is discussed in Section 10-2d. Note particularly the Read/Write conflict scenarios illustrated with the help of Table 10.11, Read/Write Conflict Scenarios: Conflicting Database Operations Matrix.

4. Using a simple example, explain the use of binary and shared/exclusive locks in a DBMS.

Discuss Section 10-3, Concurrency Control with Locking Methods. Binary locks are discussed in Section 10-3b, Lock Types.

5. Suppose that your database system has failed. Describe the database recovery process and the use of deferred-write and write-through techniques.

Recovery restores a database from a given state, usually inconsistent, to a previously consistent state. Depending on the type and the extent of the failure, the recovery process ranges from a minor short-term inconvenience to a major long-term rebuild action. Regardless of the extent of the required recovery process, recovery is not possible without backup.

The database recovery process generally follows a predictable scenario:

1. Determine the type and the extent of the required recovery.
2. If the entire database needs to be recovered to a consistent state, the recovery uses the most recent backup copy of the database in a known consistent state.
3. The backup copy is then *rolled forward* to restore all subsequent transactions by using the transaction log information.
4. If the database needs to be recovered, but the committed portion of the database is usable, the recovery process uses the transaction log to "undo" all the transactions that were not committed.

Recovery procedures generally make use of *deferred-write* and *write-thru* techniques. In the case of the deferred-write or deferred-update, the transaction operations do not immediately update the database. Instead:

- All changes (previous and new data values) are first written to the transaction log
- The database is updated only after the transaction reaches its commit point.
- If the transaction fails before it reaches its commit point, no changes (no roll-back or undo) need to be made to the database because the database was never updated.

In contrast, if the *write-thru* or *immediate-update* technique is used:

- The database is immediately updated by transaction operations during the transaction's execution, even before the transaction reaches its commit point.
- The *transaction log* is also updated, so if a transaction fails, the database uses the log information to *roll back* ("undo") the database to its previous state.

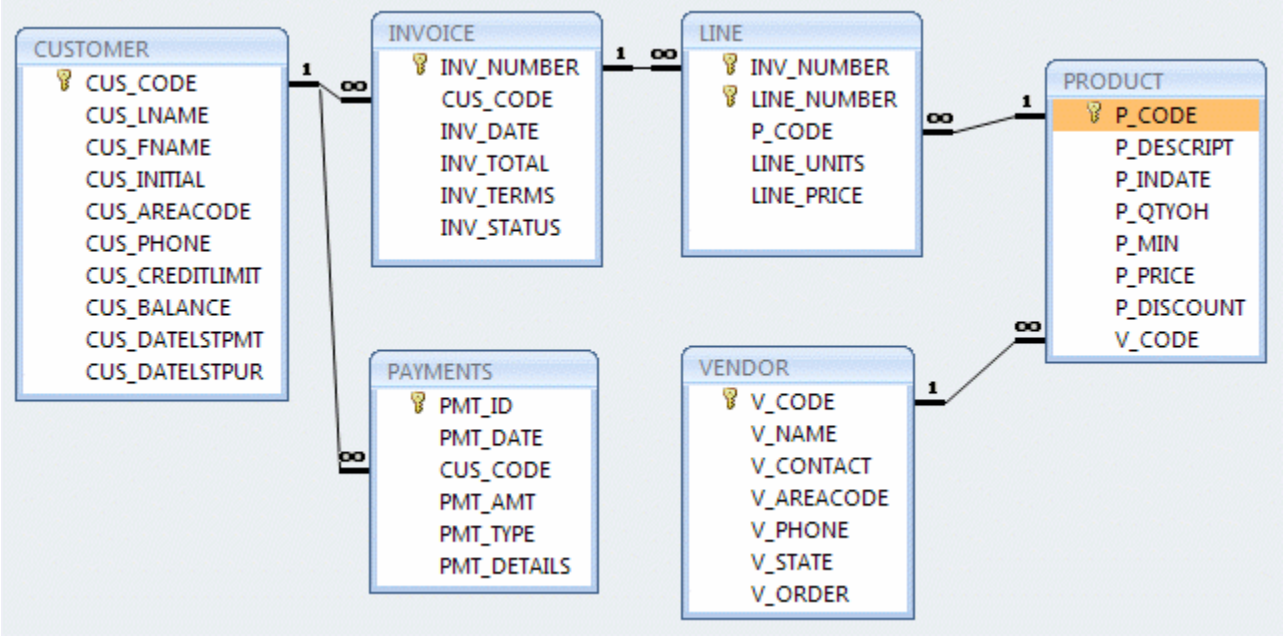
ONLINE CONTENT

The Ch10_ABC_Markets database is available at www.cengagebrain.com. This database is stored in Microsoft Access format.

6. **ABC Markets sell products to customers. The relational diagram shown in Figure P10.6 represents the main entities for ABC's database. Note the following important characteristics:**
   - **A customer may make many purchases, each one represented by an invoice.**
     - **The CUS_BALANCE is updated with each credit purchase or payment and represents the amount the customer owes.**
     - **The CUS_BALANCE is increased (+) with every credit purchase and decreased (-) with every customer payment.**
     - **The date of last purchase is updated with each new purchase made by the customer.**
     - **The date of last payment is updated with each new payment made by the customer.**
   - **An invoice represents a product purchase by a customer.**
     - **An INVOICE can have many invoice LINEs, one for each product purchased.**
     - **The INV_TOTAL represents the total cost of invoice including taxes.**
     - **The INV_TERMS can be "30," "60," or "90" (representing the number of days of credit) or "CASH," "CHECK," or "CC."**
     - **The invoice status can be "OPEN," "PAID," or "CANCEL."**
   - **A product's quantity on hand (P_QTYOH) is updated (decreased) with each product sale.**
   - **A customer may make many payments. The payment type (PMT_TYPE) can be one of the following:**
     - **"CASH" for cash payments.**
     - **"CHECK" for check payments**
     - **"CC" for credit card payments**
   - **The payment details (PMT_DETAILS) are used to record data about check or credit card payments:**
     - **The bank, account number, and check number for check payments**
     - **The issuer, credit card number, and expiration date for credit card payments.**

Note: Not all entities and attributes are represented in this example. Use only the attributes indicated.

FIGURE P10.6 The ABC Markets Relational Diagram



Using this database, write the SQL code to represent each one of the following transactions. Use BEGIN TRANSACTION and COMMIT to group the SQL statements in logical transactions.

a. On May 11, 2016, customer '10010' makes a credit purchase (30 days) of one unit of product '11QER/31' with a unit price of $110.00; the tax rate is 8 percent. The invoice number is 10983, and this invoice has only one product line.

a. BEGIN TRANSACTION
b. INSERT INTO INVOICE
    i. VALUES (10983, '10010', '11-May-2016', 118.80, '30', 'OPEN');
c. INSERT INTO LINE
    i. VALUES (10983, 1, '11QER/31', 1, 110.00);
d. UPDATE PRODUCT
    i. SET P_QTYOH = P_QTYOH – 1
    ii. WHERE P_CODE = '11QER/31';
e. UPDATE CUSTOMER
f. SET CUS_DATELSTPUR = '11-May-2016', CUS_BALANCE = CUS_BALANCE +118.80
g. WHERE CUS_CODE = '10010';
h. COMMIT;

b. On June 3, 2016, customer '10010' makes a payment of $100 in cash. The payment ID is 3428.

a. BEGIN TRANSACTION
b. INSERT INTO PAYMENTS
VALUES (3428, '03-Jun-2016', '10010', 100.00, 'CASH', 'None');
UPDATE CUSTOMER;

```
SET CUS_DATELSTPMT = '03-Jun-2016', CUS_BALANCE = CUS_BALANCE -100.00
WHERE CUS_CODE = '10010';
COMMIT
```

7. Create a simple transaction log (using the format shown in Table 10.13) to represent the actions of the two previous transactions.

The transaction log is shown in Table P10.7

Table P10.7 The ABC Markets Transaction Log

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
987	101	Null	1023	START	* Start Trx.				
1023	101	987	1026	INSERT	INVOICE	10983			10983, 10010, 11-May-2016, 118.80, 30, OPEN
1026	101	1023	1029	INSERT	LINE	10983, 1			10983, 1, 11QER/ 31, 1, 110.00
1029	101	1026	1031	UPDATE	PRODUCT	11QER/31	P_QTYOH	47	46
1031	101	1029	1032	UPDATE	CUSTOMER	10010	CUS_BALANCE	345.67	464.47
1032	101	1031	1034	UPDATE	CUSTOMER	10010	CUS_DATELSTPUR	5-May-2014	11-May-2016
1034	101	1032	Null	COMMIT	* End Trx. *				
1089	102	Null	1091	START	* Start Trx.				
1091	102	1089	1095	INSERT	PAYMENT	3428			3428, 3-Jun-2016, 10010, 100.00, CASH, None
1095	102	1091	1096	UPDATE	CUSTOMER	10010	CUS_BALANCE	464.47	364.47
1096	102	1095	1097	UPDATE	CUSTOMER	10010	CUS_DATELSTPMT	2-May-2014	3-Jun-2016
1097	102	1096	Null	COMMIT	* End Trx.				

Note: Because we have not shown the table contents, the "before" values in the transaction can be assumed. The "after" value must be computed using the assumed "before" value, plus or minus the transaction value. Also, in order to save some space, we have combined the "after" values for the INSERT statements into a single cell. Actually, each value could be entered in individual rows.

8. Assuming that pessimistic locking is being used, but the two-phase locking protocol is not, create a chronological list of the locking, unlocking, and data manipulation activities that would occur during the complete processing of the transaction described in Problem 6a.

Time	Action
1	Lock INVOICE
2	Insert row 10983 into INVOICE
3	Unlock INVOICE
4	Lock LINE
5	Insert tow 10983, 1 into LINE
6	Unlock LINE
7	Lock PRODUCT

8	Update PRODUCT 11QER/31, P_QTYOH from 47 to 46
9	Unlock PRODUCT
10	Lock CUSTOMER
11	Update CUSTOMER 10010, CUS_BALANCE from 345.67 to 464.47
12	Update CUSTOMER 10010, CUS_DATELSTPUR from 05-May-2016 to 11-May-2016
13	Unlock CUSTOMER

9. Assuming that pessimistic locking with the two-phase locking protocol is being used, create a chronological list of the locking, unlocking, and data manipulation activities that would occur during the complete processing of the transaction described in Problem 6a.

Time	Action
1	Lock INVOICE
2	Lock LINE
3	Lock PRODUCT
4	Lock CUSTOMER
5	Insert row 10983 into INVOICE
6	Insert tow 10983, 1 into LINE
7	Update PRODUCT 11QER/31, P_QTYOH from 47 to 46
8	Update CUSTOMER 10010, CUS_BALANCE from 345.67 to 464.47
9	Update CUSTOMER 10010, CUS_DATELSTPUR from 05-May-2016 to 11-May-2016
10	Unlock INVOICE
11	Unlock LINE
12	Unlock PRODUCT
13	Unlock CUSTOMER

10. Assuming that pessimistic locking is being used, but the two-phase locking protocol is not, create a chronological list of the locking, unlocking, and data manipulation activities that would occur during the complete processing of the transaction described in Problem 6b.

Time	Action
1	Lock PAYMENT
2	Insert row 3428 into PAYMENT
3	Unlock PAYMENT
4	Lock CUSTOMER
5	Update CUSTOMER 10010, CUS_BALANCE from 464.47 to 364.47
6	Update CUSTOMER 10010, CUS_DATELSTPMT from 02-May-2016 to 03-Jun-2016
7	Unlock CUSTOMER

11. Assuming that pessimistic locking with the two-phase locking protocol is being used, create a chronological list of the locking, unlocking, and data manipulation activities that would occur during the complete processing of the transaction described in Problem 6b.

Time	Action
1	Lock PAYMENT
2	Lock CUSTOMER
3	Insert row 3428 into PAYMENT
4	Update CUSTOMER 10010, CUS_BALANCE from 464.47 to 364.47
5	Update CUSTOMER 10010, CUS_DATELSTPMT from 02-May-2016 to 03-Jun-2016
6	Unlock PAYMENT
7	Unlock CUSTOMER

Additional Problems and Answers

The following problems are designed to give your students additional practice … or you can use them as test questions.

1. **Write the SQL statements that might be used in transaction management and explain how they work.**

   The following transaction registers the credit sale of a product to a customer.

```
                                                    Comment
BEGIN TRANSACTION                                   Start transaction

INSERT INTO INVOICE                                 Add record to invoice
    (INV_NUM, INV_DATE, ACCNUM, TOTAL)
        VALUES (1020,'15-MAR-2016','60120010',3500);

UPDATE INVENTRY                                     Update the quantity on hand of the
    SET ON_HAND = ON_HAND – 100                     product
        WHERE PROD_CODE = '345TYX';

UPDATE ACCREC                                       Update the customer balance
    SET BALANCE = BALANCE + 3500                    account
        WHERE ACCNUM = '60120010';
COMMIT;
```

   The credit sale transaction must do *all* of the following:
   1. Create the invoice record.
   2. Update the inventory data.
   3. Update the customer account data.

   In SQL, the transaction begins automatically with the first SQL statement, or the user can start with the BEGIN TRANSACTION statement. The SQL transaction ends when
   - the last SQL statement is found and/or the program ends
   - the user cancels the transaction
   - COMMIT or ROLLBACK statements are found

   The DBMS will ensure that all SQL statements are executed and completed before committing all work. If, for any reason, one or more of the SQL statements in the transaction cannot be completed, the entire transaction is aborted and the database is rolled back to its previous consistent state.

2. Starting with a consistent database state, trace the activities that are required to execute a set of transactions to produce an updated consistent database state.

The following example traces the execution of problem 1's credit sale transaction. We will assume that the transaction is based on a currently consistent database state. We will also assume that the transaction is the only transaction being executed by the DBMS.

Time	Transaction	Table	Operation	Comment
0				Database is in a consistent state.
1	Write	INVOICE	INV_NUM = 1020	INSERT Invoice number into the INVOICE table
2	Read	INVENTORY	ON_HAND = 134	
3			ON_HAND = 134 - 100	UPDATE the quantity on hand of product 345TYX
4	Write		ON_HAND = 34	
5	Read	ACCREC	ACC_BALANCE = 900	
6			ACC_BALANCE = 900 + 3500	
7	Write		ACC_BALANCE = 4400	
8	COMMIT			Permanently saves all changes to the database. The database is in a consistent state.

3. Write an example of a database request.

A database transaction is composed of *one or more* database requests. A database request is the equivalent of an SQL statement, i.e. SELECT, INSERT, UPDATE, PROJECT, etc. Some database transactions are as simple as:

```
SELECT   *
FROM     ACCOUNT;
```

A database request can include references to one or more tables. For example, the request

```
SELECT   ACCT_NUM, CUSTOMER.CUS_NUM, INV_NUM, INV_DATE, INV_TOTAL
FROM     ACCOUNT, INVOICE
WHERE    ACCOUNT.ACCT_NUM     = INVOICE.ACCT_NUM AND
         ACCCOUNT.ACCT_NUM    = '60120010'
```

will list all the invoices for customer '60120010'. Note that the preceding query accesses two tables: ACCOUNT and INVOICE. Note also that, if an attribute shows up in different places (as a primary key and as a foreign key), its source must be specified to avoid an ambiguity error message.

A database request may update a database or insert a new row in a table. For example the database request

```
INSERT INTO INVOICE (INV_NUM, INV_DATE, ACCT_NUM, INV_TOTAL)
    VALUES (1020,'10-Feb-2016','60120010',3500);
```

will insert a new row into the INVOICE table.

4. **Trace the use of the transaction log in database recovery.**

The following transaction log traces the database transaction explained in problem 1.

Transaction Log

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
1	101	NULL	2	* Start TRX *					
2	101	1	3	INSERT	INVOICE				1020, '10-Feb-2016', '60120010', 3500
3	101	2	4	UPDATE	PRODUCT	345TYX	PROD_ON_HAND	134	34
4	101	3	5	UPDATE	ACCOUNT	60120010	ACCT_BALANCE	900	4,400
5	101	4	NULL	COMMIT					

* The TID (Transaction ID) is automatically assigned by the DBMS

The transaction log maintains a record of all database transactions that changed the database. For example, the preceding transaction log records

- the insertion of a new row to the INVOICE table
- the update to the P_ON_HAND attribute for the row identified by '345TYX' in the PRODUCT table
- and the update of ACCT_BALANCE attribute for the row identified by '60120010' in the ACCOUNT table.

The transaction log also records the transaction's beginning and end in order to help the DBMS to determine the operations that must be rolled back if the transaction is aborted. Note: Only the current transaction may be rolled back, not all the previous transactions.

If the database must be recovered, the DBMS will:

- Change the BALANCE attribute of the row '60120010' from 4400 to 900 in the ACCREC table.
- Change the ON_HAND attribute of the row '345TYX' from 34 to 134 in the INVENTORY table.
- Delete row 1020 of the INVOICE table.

5. **Suppose you are asked to evaluate a DBMS in terms of lock granularity and the different locking levels. Create a simple database environment in which these features would be important.**

Lock granularity describes the different lock levels supported by a DBMS. The lock levels are:

Database-level

- The DBMS locks the entire database. If transaction T1 locks the database, transaction T2 cannot access any database tables until T1 releases the lock.

- Database-level locks work well in a batch processing environment when the bulk of the transactions are executed off-line and a batch process is used to update the master database at off-peak times such as midnight, weekends, etc.

Table-level

- The DBMS locks an entire table within a database. This lock level prevents access to any row by a transaction T2 while transaction T1 is using the table. However, two transactions can access the database as long as they access different tables.
- Table-level locks are only appropriate when table sharing is not an issue. For example, if researchers pursue unrelated research projects within a research database, the DBMS will be able to allow such users to access their data without affecting the work of other users.

Page-level

- The DBMS will lock an entire *disk-page*. A disk-page or page is the equivalent of a *disk-block*, which may be described as a (referenced) section of a disk. A page has a fixed size, such as 4K, 8K, 16K, etc. A table may span several pages, and a page may contain several rows of one or more tables. Page-level locks are (currently) the most frequently used of the multi-user DBMS locking devices.
- Page-level locks are particularly appropriate in a multi-user DBMS system in which data sharing is a crucial component. For example, page-level locks are common in accounting systems, sales systems, and payroll systems. In fact, just about any business DBMS application that runs in a multi-user environment benefits from page-level locking.

Row-level

- The row-level lock is much less restrictive than the locks we have just discussed. Row-level locks permit the DBMS to allow concurrent transactions to access different rows of the same table, even if these rows are located on the same page. Although the row-level locking approach improves the availability of data, its management requires high overhead cost because a lock exists for each row in each table of the database.
- Row-level locking is yet to be implemented in most of the currently available DBMS systems. Consequently, row-level locks are mostly a curiosity at this point. Nevertheless, its very high degree of shareability makes it a potential option for multi-user database applications like the ones that currently use page-level locking.

Field-level

- The field-level locking approach allows concurrent transactions to access the same row as long as they use different attributes within that row. Although field-level locking clearly yields the most flexible multi-user data access, it requires too much computer overhead to be practical at this point.