Module 4

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 4 Study Guide and Deliverables

Background Concepts Readings:	• Coronel & Morris, chapter 6 and chapter 9 (only sections 9-1 through 9-3, and 9-8 through 9-9)
Optional SQL Readings:	• *12th Edition:* Coronel & Morris, sections 8.4 through 8.8 of chapter 8 • *13th Edition:* Coronel & Morris, sections 8.6 through 8.8 of chapter 8
Assignments:	• Term Project Iteration 4 due **Tuesday, February 14 at 6:00 AM ET** • Lab 4 due **Tuesday, February 14 at 6:00 AM ET**
Live Classroom:	• **Tuesday, February 7 from 8:00-9:30 PM ET** • **Wednesday, February 8 from 8:00-9:30 PM ET**

Which First?

Read the book chapters before reading the online lectures.

The categorizations of SQL functions provided by the textbook—date and time, numeric, string, and conversion—are widely accepted, yet these categorizations by no means identify the only kinds of functions available. Many modern DBMS functions take many different types of parameters and return many different types of values, and these functions cannot be easily categorized. For example, functions in some modern DBMS can take table rows or entire tables as parameters. Modern DBMS also support the creation of new, custom functions which may not be accurately described by the aforementioned categories. Lastly, different DBMS implement many functions differently and so there is little uniformity between different DBMS.

Section 9.4 in the textbook teaches conceptual design with one major difference with what is taught in this course. The textbook includes SQL constraints in conceptual design, resulting in conceptual ERDs with SQL constraints visible, while in this course we teach you that conceptual design should

be entirely independent of any particular model, including the relational model. In this course, we recommend to avoid using SQL-based constraints in conceptual design.

Lecture 10 - Database Design

Introduction

metcs669_module09 video cannot be displayed here

Data are raw facts stored in a database. These raw facts are processed in transactions and transformed into useful information which can be used for decision making by managers. The database is part of a larger whole known as an *information system*, which handles the storage, collection, retrieval and analysis of data. Information systems are composed of people, hardware, software, one or more databases, application programs, and procedures. *Systems analysis* is the process that establishes the need for and extent of an information system. The process of creating this information system is known as *systems development*. Within the framework of systems development, applications transform data into information. The need for this transformation and the careful planning for the storage of data and information is considered during design. Database design is a critical aspect of information system design.

An information system performs three sets of services:

- It supports data collection, storage, and retrieval.
- It facilitates the transformation of data into information.
- It provides the tools and support for the management of data and information.

Basically, a database is a fact (data) repository that serves an information system. Because the representation and management of data is critical to information systems, poor database design thwarts the transformation of data to information and impedes or prevents the efficient management of data and information by the entire information system.

The transformation of data into information is accomplished through application programs. It is impossible to produce good information from poor data; and, no matter how sophisticated the application programs are, it is difficult or sometimes impossible for applications programs to overcome the effects of bad database design. Application programmers can, by devoting considerable time and money, overcome many database design defects, but it is far better to provide a design that supports applications programs well. In short: **Good database design is crucial to the success of an information system.**

Database design must yield a database that:

- has no uncontrolled data duplication, thus preventing data anomalies and the attendant lack of data integrity.
- provides efficient data access.
- serves the needs of the information system.

The last point deserves emphasis: even the best-designed database lacks value if it fails to meet information system objectives. In short, good database designers must pay close attention to the information system requirements.

Systems design and database design are usually tightly intertwined and are often performed in parallel. Therefore, database and systems designers must cooperate and coordinate to yield the best possible information system.

Test Yourself 4.1

Which of the following is correct? (Please select all of the following that are correct.)

The database is part of a larger system called the information system, which includes human beings.

This is true. Information systems are composed of people, hardware, software, one or more databases, application programs, and procedures.

Systems analysis is the process that establishes the need for and extent of an information system.

This is true. The process of creating this information system is known as systems development.

The sole function of an information system is to support data collection, storage, and retrieval.

This is false. An information system performs three sets of services. These are supporting data collection, storage, and retrieval; facilitating the transformation of data to information; and providing the tools and support for the management of data and information.

A database serves the needs of the information system.

This is true. Good database design is crucial to the success of an information system.

Systems design and database design are generally performed in complete isolation from each other.

This is false. Systems design and database design are usually tightly intertwined and are often performed in parallel. Coordination and cooperation between the involved designers are needed to result in the optimal information system.

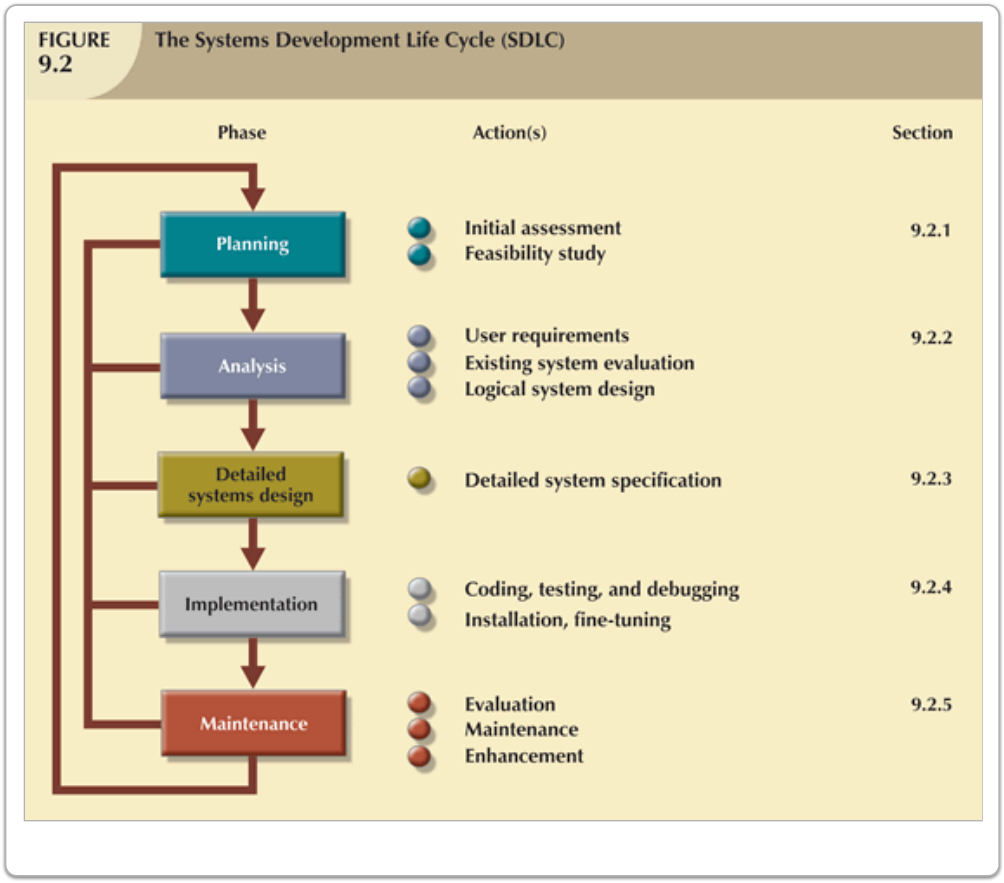
Developing Successful Information Systems

The *Systems Development Life Cycle*, or SDLC, is an idealized model of the life history of an information system. The SDLC described in the text is based on an early and in practice unworkable model of system and database design and development commonly termed the "waterfall model." The fundamental flaw of the waterfall model is that we do not in practice understand the requirements, or the appropriate solution, when we are beginning, but only discover the real requirements as we proceed with the development. Real modern software development is better described as a complex iterative process in which the requirements, design, implementation, and evaluation phases are repeated in a more or less orderly sequence, with the results of the evaluation used to update the requirements and design. In the best approaches the cycles in the iterative development are replanned frequently, often daily, as more is learned about the requirements and the solutions that satisfy those requirements. A currently popular form of iterative development which is more ad hoc is termed "agile" development, implying in part that the plan and even the process itself are continually redefined. The SDLC provides the big picture within which database design and development can be mapped out and evaluated. There are many ways to describe the SDLC. One of the simplest is to describe the SDLC as having five primary phases:

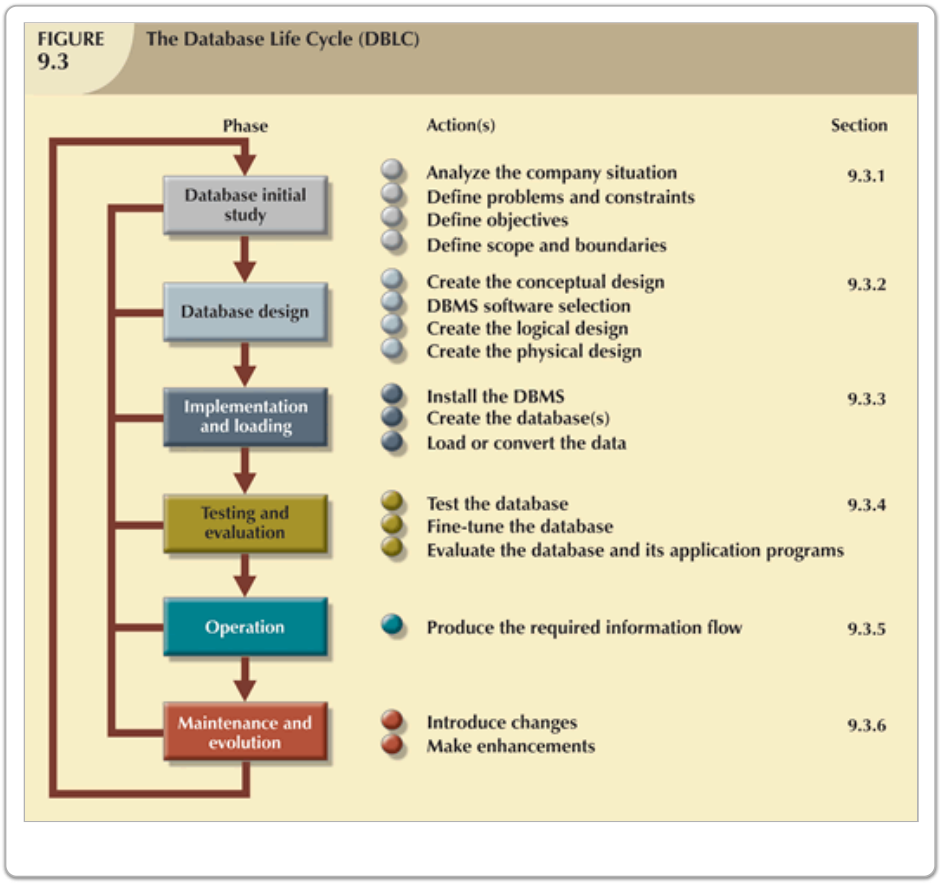
1. The *planning phase* provides a general overview of the enterprise and its objectives
2. The *analysis phase* looks at the problems defined during the first phase. It seeks to understand the precise requirements of the end user and develop an understanding of how those requirements fit with the overall information system.
3. The *detailed systems design phase* is when the designer completes the technical specifications for the system.
4. The *implementation phase*, is used for the installation of hardware, applications, and DBMS software. Any custom-developed software is written during this phase as well.
5. The *systems maintenance phase* is when the system is kept operational and modifications are made.

The SDLC is essential to the successful definition, design, deployment and maintenance of an information system.

The SDLC traces the history (life cycle) of an information system. The Database Life Cycle (DBLC) traces the history (life cycle) of a database system. Since we know that the database serves the information system, it is not surprising that the two life cycles conform to the same basic phases.



Compare the above SDLC with the Database Life Cycle shown below:



Test Yourself 4.2

Which of the following is true regarding systems development? (Please select all of the following that are true.)

Modern software development is best described as a five phase process during which each phase is addressed once and completed in order.

This is false. Modern software development is a complex iterative process. The results of later phases are used to update the results of earlier phases.

A form of development referred to as “agile” development forces an adherence to a more concrete process to improve efficiency.

This is false. With “agile” development, the plan and process may be continually redefined.

There is a single correct way to describe the systems development life cycle (SDLC). This description is provided by the course textbook.

This is false. There are many ways to describe the SDLC. One of the simplest is to describe it as having five primary phases (Planning, Analysis, Detailed systems design, Implementation, and Maintenance).

The Database Life Cycle (DBLC) can be described as having similar phases to the SDLC.

This is true. The DBLC traces the history (life cycle) of a database system. As the database system serves the needs of the information system, it should not be a surprise that the phases are similar.

According to the provided descriptions, changes are made in the Maintenance phase of the SDLC and the Maintenance and evolution phase of the DBLC.

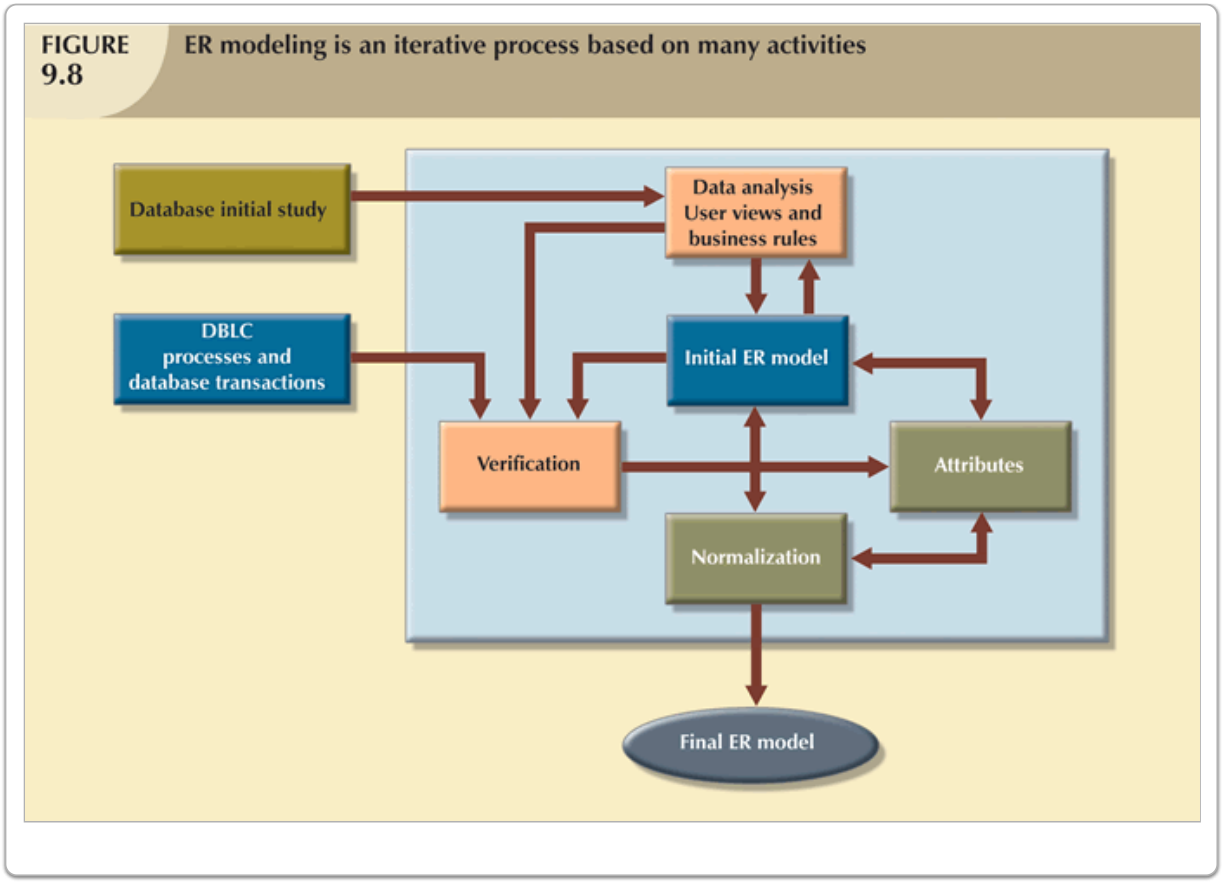
This is true. This is an example of similarity.

Features of Most Successful Databases

The Database Life Cycle, or DBLC, describes the life history of the database within the information system. It provides a framework for the evaluation, revision, and maintenance of the database. There are several different ways of describing and implementing the DBLC, but one of the most common and useful describes the DBLC as consisting of six phases or activities:

1. The *database initial study phase* is when designers examine the current system and try to determine what problems exist and what opportunities are present to improve the system.. If there is no current system, then the initial study phase defines the data and database requirements for the proposed new system.
2. The *database design phase* involves creating or updating the conceptual database model, selecting the DBMS software, and then creating the logical and physical designs.
3. The *implementation and loading phase* deals with installing the DBMS, creating the database, and loading the required data.
4. The *testing and evaluation phase* ensures the database is working as expected.
5. The *operation phase* produces the expected information flow.
6. The *maintenance and evaluation phase* keeps the system operating and up-to-date. The maintenance and evaluation phase can be decades long and it can include bounded revisiting of the five steps above.

Also note the tight integration between the two life cycles and ERM below.



Test Yourself 4.3

Which of the following is true regarding database design? (Please select all of the following that are true.)

The Database Life Cycle (DBLC) describes the life history of the database within the information system.

This is true. It provides a framework for the evaluation, revision, and maintenance of the database.

There is a single correct way to describe the database life cycle (DBLC). This description is provided by the course textbook.

This is false. There are several different ways of describing and implementing the DBLC, but one of the most common and useful describes the DBLC as consisting of six phases or activities (Database initial study, Database design, Implementation and loading, Testing and evaluation, Operation, and Maintenance and evaluation).

The DBLC is an iterative process.

This is true. This is similar to the SDLC.

Once a conceptual database model is created, it should not be changed.

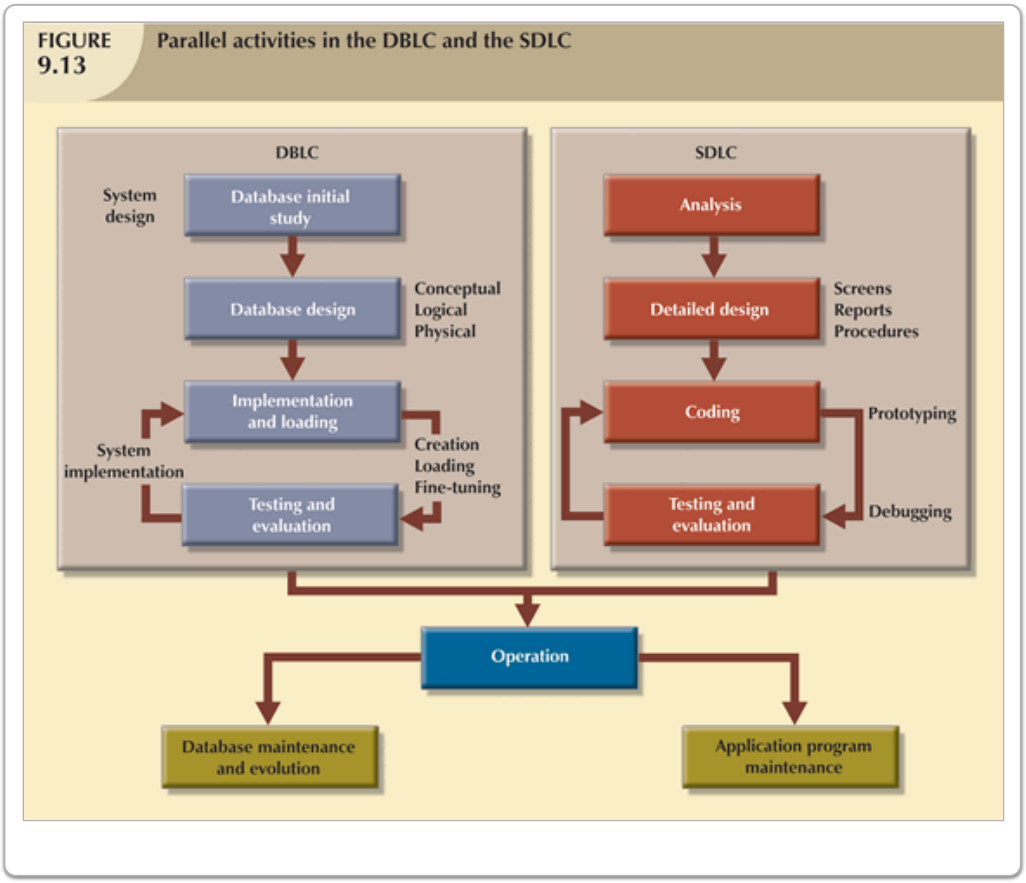
This is false. The conceptual database model may be updated multiple times.

The maintenance and evaluation phase of the DBLC is always the shortest phase.

This is false. The maintenance and evaluation phase can be decades long and it can include bounded revisiting of the other phases.

Conducting Evaluation and Revision within SDLC and DBLC Frameworks

Both the SDLC and the DBLC provide an orderly set of processes for defining, creating, and maintaining information systems and databases. The SDLC focuses on the overall information system while the DBLC is specifically tailored for database system implementation. Both life cycles have initial phases that allow problems or needs to be defined. These needs are carefully studied in subsequent phases prior to developing an appropriate solution. After the systems are in place, both have later phases that enable users to suggest changes and have them implemented. In the SDLC, this takes place in the maintenance phase. In the DBLC, this is the maintenance and evolution phase. Both systems encourage post-implementation evaluation based on audits and testing. In this way, the systems can continue to improve and remain in operation for longer time periods.



Test Yourself 4.4

Which of the following is true regarding the SDLC and the DBLC? (Please select all of the following that are true.)

The SDLC and the DBLC are distinct unrelated processes with little to no similarity.

This is false. These are related processes with much similarity.

There are later phases or stages in both during which changes or updates may be made.

This is true. The maintenance phase of a SDLC can be considered similar to the maintenance and evaluation phase of a DBLC.

Both encourage post-implementation evaluation based on audits and testing.

This is true. Both the database system and the larger information system can continue to improve and remain in operation for extended periods.

Database Design Strategies

There are several database design strategies, which are often best used in combination. Two classical strategies are the top-down and bottom-up approaches. The top-down approach starts by identifying data sets. Data elements or attributes are defined for each of the identified entities. In the bottom-up approach, data elements are identified first, and then grouped together into data sets. In other words, the attributes are identified and then the entities are determined.

These general approaches to database design can be influenced by factors such as the scope and size of the system, the company's management style, or the company's structure. Depending on these factors, the database design may be *centralized* or *decentralized*. The centralized design approach is productive when the data component is relatively simple. Centralized design is usually carried out by a single person or small team. A *decentralized design* approach works better in large team environments and in situations where complex, diverse and distributed data components are present.

Advanced Topic: Island Growing

There is a variant of the top down design strategy sometimes termed *island growing* that is useful when one or more subsets of the data are already well understood. For example, there may be an operational database with a solid design for some of the data. In these situations it is often helpful to identify the needed additional data associated with these existing islands. When using island growing it is important to develop an overall high level data model that spans the islands, to avoid duplication, maintain consistency, and facilitate integration of the islands into a coherent overall design. Island growing is particularly helpful when extending existing database systems or integrating database systems. Island growing conducted piecemeal during the maintenance phase without a coherent overall model can result in an incoherent design that gradually degenerates into inconsistent isolated islands with duplicated data. Island growing from different database systems is often useful in decentralized design.

To recap there are two basic approaches to database design: top-down and bottom-up.

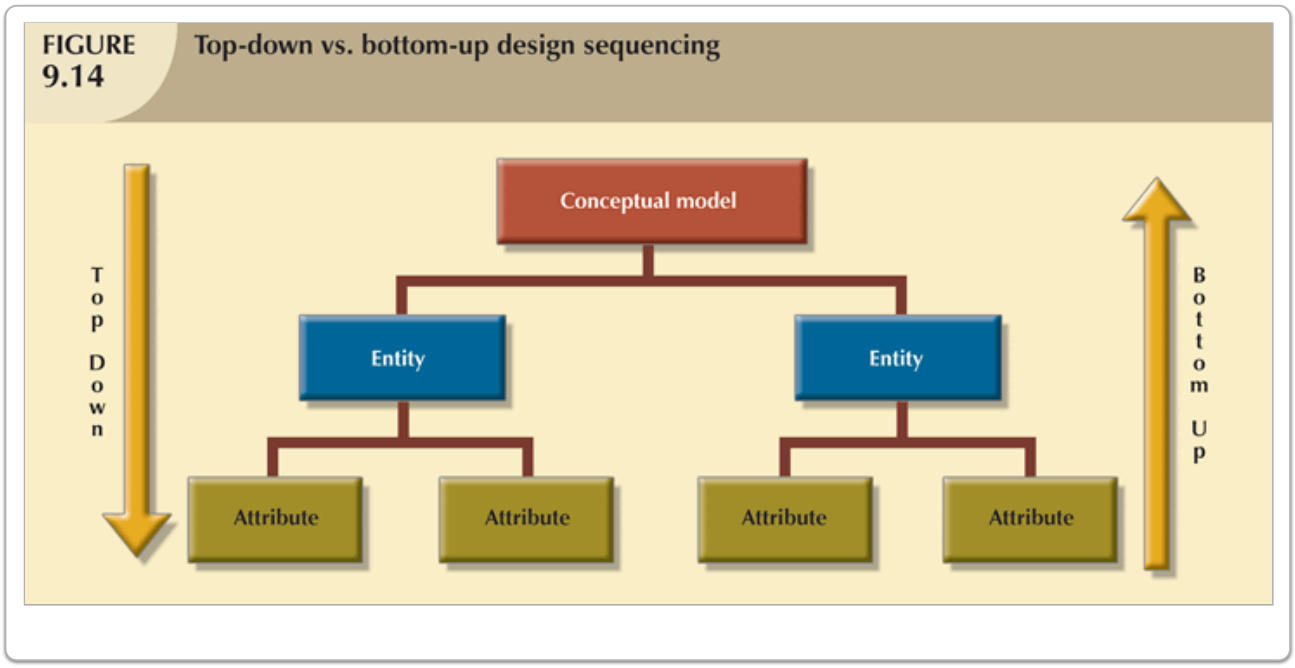
Top down design begins by identifying the different entity types and the definition of each entity's attributes. In other words, top-down design:

- starts by defining the required data sets and then,
- defines the data elements for each of those data sets.

Bottom-up design:

- first defines the required attributes and then,

- groups the attributes to form entities.



From our experience the best approach integrates top-down and bottom-up activities, with complementary interplay between these approaches, and frequent interaction with the users and others who understand the data well.

Test Yourself 4.5

Which of the following is true regarding the design of a database? (Please select all of the following that are true.)

The top-down approach starts by identifying data sets.

This is true. As opposed to the bottom-up approach, in which data elements are identified first, and then grouped together into data sets. (Top-down: entities then attributes; Bottom-up: attributes then entities)

Database design is either top-down or bottom-up, but never both.

This is false. Optimal outcome may result when these approaches are used together.

The database for a very small company with modest data component complexity is more likely to be designed using a centralized approach as opposed to a decentralized approach.

This is true. The centralized design approach is productive when the data component is relatively simple. (A decentralized approach works better in situations where complex, diverse and distributed data components are present.)

Centralized design of a database would typically be carried out by fewer people than decentralized design of a database.

This is true. Centralized design is usually carried out by a single person or small team as opposed to decentralized design which would typically involve a large team.

Summary

Databases are central to information systems. Systems analysis defines the business needs for information systems as well as their scope. Systems development is the set of processes that define, design, develop, deploy, and maintain information systems. The Systems Development Lifecycle (SDLC) traces the life cycle of applications within an information system. The traditional SDLC includes planning, analysis, detailed design, implementation, and deployment activities, which may be staged sequentially or incrementally. The Database Life Cycle (DBLC) traces the life cycle of a database within an information system. The DBLC can be described in terms of phases such as initial study, design, implementation and loading, testing and evaluation, and maintenance and evaluation. The DBLC activities are usually conducted iteratively, with iterations also within complex activities such as design. Conceptual database design can be conducted in either top down (entities first) or bottom-up (attributes first) strategies. For most database environments the best approach is to combine these strategies. Conceptual database design can be conducted in a centralized fashion, with a single small team designing the database, or in a distributed fashion, where multiple teams work on different portions of the design. The choice of either approach, and their combination within an integrating design framework, depends on the complexity of the design, how easily it can be factored into subschemas or databases, and the organizational environment.

Lecture 11 - Transaction Management and Concurrency Control

Introduction

metcs669_module10 video cannot be displayed here

A *transaction* is any action that reads from a database, writes to it, or does both. A transaction might consist of a simple action like a SELECT statement or something more complex like a series of UPDATE statements. A transaction usually represents a real-world event and transactions must be a logical unit of work. An entire transaction must be executed. If only a portion of it can be executed, then the entire transaction must be aborted. A transaction takes a database from one consistent state to another. A consistent database is one in which all data integrity constraints are satisfied. In multiuser databases transactions have five properties: atomicity, consistency, isolation, durability, and serializability. *Atomicity* means that if any operations in a transaction are completed then all operations must be completed and that any incomplete transactions must be entirely aborted. Consistency means that

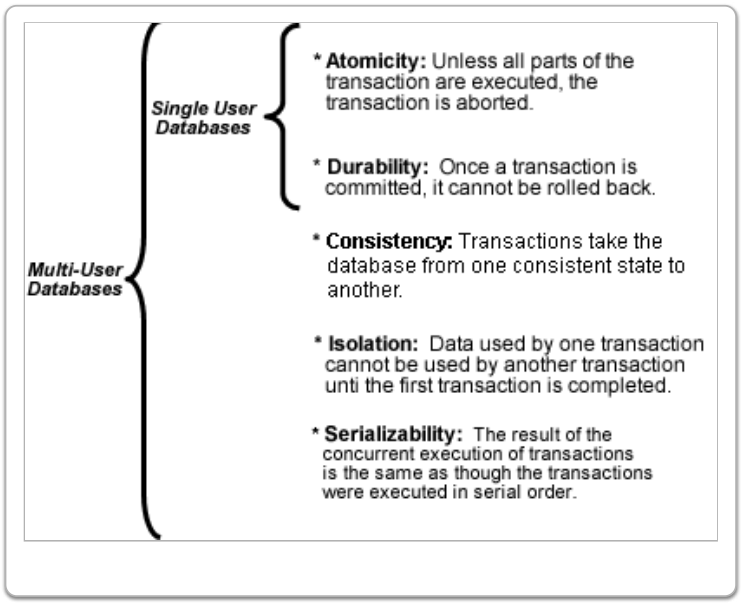
transactions take the database from one consistent state to another. *Isolation* means that other database sessions are isolated from the effects of transactions until execution of the current transaction is complete. *Serializability* requires that transactions execute as if they were conducted in serial order. This is particularly important in multi-user and distributed database environments. *Durability* refers to the fact that a consistent database state must be permanent. Many people find it convenient to remember Atomicity, Consistency, Isolation, Durability, and Serializability by their acronym ACIDS.

Historic Note: Serializability

You may encounter earlier descriptions of the basic transaction properties that do not include serializability. Some early DBMS did not support serializability and it is sometimes possible to configure a modern DBMS so that the serializability requirement, which is computationally expensive, is relaxed. It is better for users to not have to consider whether the DBMS may execute the steps in one of their transactions in a different order, so the preferred modern definition of full transaction support in DBMS includes serializability.

- A transaction usually *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 five properties:



Test Yourself 4.6

Which of the following are true regarding transactions? (Please check all of the following that are true.)

The SQL statements grouped together to form a transaction are selected by the DBMS based on memory availability and performance optimization.

This is false. Transactions must be a logical unit of work and usually represent a real-world event.

A transaction contains 2 UPDATE statements, one INSERT statement, and one DELETE statement. The DELETE statement cannot be executed. As a result, the entire transaction must be aborted.

This is true. This demonstrates the atomicity property of a transaction. Any incomplete transactions must be entirely aborted.

Serializability demands that the results of executing transaction A at the same time as transaction B are the same as the results would be if one of the transactions was executed after the other was completed.

This is true. With the serializability property, the result of the concurrent execution of requests is the same as though the requests were executed in serial order.

A user is executing transaction X that updates the balance of customer C. Another user wants to execute transaction Y that also updates the balance of customer C. The isolation property dictates that transaction Y cannot access the balance of customer C until transaction X is complete.

This is true. The property of isolation dictates that data used by one transaction cannot be used by another transaction until the first transaction is completed.

In the case of a database system that only allows a single user session, transaction isolation is guaranteed.

This is true. In this case, only a single transaction is executed at a time. Transaction serializability is also guaranteed in this case.

Managing Database Transactions

Database transactions are managed with SQL. The American National Standards Institute (ANSI) has defined standards that govern database transactions. Transactions are supported by two SQL statements commands: COMMIT and ROLLBACK. The COMMIT command permanently records all of the changes in a transaction within the database. The ROLLBACK command aborts all changes and returns the database to its previous consistent state. ANSI standards require that once initiated, a transaction must continue through all succeeding SQL statements until one of four events occur: a COMMIT statement is reached; a ROLLBACK statement is reached; the end of a program is successfully reached and all changes are permanently recorded to the database; or the program is abnormally terminated and the database is returned to its prior state. The DBMS uses a transaction log to keep track of all transactions that change the database.

The transaction log plays a key role in each transaction. It is a special file that contains a description of all the database transactions executed by the DBMS. In most DBMS the database transaction log plays a central role in maintaining database concurrency control. The transaction log is central to maintaining integrity in all DBMS.

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE

341	101	Null	352	START	****Start Transaction				
351	101	341	363	UDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	****End of Transaction				

TRL_ID= Transaction log record ID
TRX_NUM=Transaction number
PTR= Pointer to a transaction log record ID
(Note: The transaction number is automatically assigned by the DBMS.)

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 other storage device to prevent the failure caused by a media error. In Oracle and other industrial-strength DBMS the transaction logs can be written redundantly to multiple storage devices, and this is normally done for important databases.

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.

Test Yourself 4.7

Which of the following are correct regarding database transactions? (Please check all of the following that are correct.)

The COMMIT command permanently saves all changes in a transaction within the database.

This is true. This is the ANSI defined function of the COMMIT command. Transactions are supported by two SQL statements commands: COMMIT and ROLLBACK. The ROLLBACK command aborts all changes and returns the database to its previous consistent state.

While a transaction is being executed, the end of a program is successfully reached without reaching a COMMIT statement. Since no COMMIT statement was reached, the database must be returned to its prior state.

This is false. According to ANSI standards, successful program termination is equivalent to a COMMIT statement. All changes would be permanently recorded to the database.

While a transaction is being executed, the program is abnormally terminated. This will trigger the same action that would have occurred if a COMMIT statement had been reached.

This is false. According to ANSI standards, the abnormal termination of a program should ROLLBACK the transaction. The database would be returned to its previous consistent state.

The transaction log functions as a source of information for returning the database to a previous consistent state.

This is true. The transaction log stores a description of all transactions that alter the database. It serves as a source of information for recovery or ROLLBACK of the database.

Multiple copies of the same transaction log may be stored.

This is true. This is typical for important databases.

Concurrency Control and the Role it Plays

Introduction

The coordination of simultaneous execution of transactions in a multiprocessing database system is known as *concurrency control*. The main objective of concurrency control is to ensure that the steps in transactions are executed in an order which is functionally equivalent to the order they were requested by the user or application. This concept is called *serializability*. This is particularly important in multiprocessing database environments. The three main problems regarding concurrency are: lost updates; uncommitted data; and inconsistent retrievals. The scheduler is the DBMS software subsystem that manages the concurrent execution of transactions. The scheduler interleaves the execution of database operations to ensure that concurrent transaction executions occur appropriately. This is done using locking, time stamping, or optimistic methods. The scheduler also ensures the computer's processing and I/O systems are used efficiently.

Concurrency control is the activity of coordinating the simultaneous execution of transactions in a multiprocessing or multiuser database management system. The objective of concurrency control is to ensure the serializability of transactions in a multiuser database management system. The *scheduler* is the software subsystem in the DBMS that is responsible for 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.

A scheduler plays a key role in 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 multiple 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.

Your textbook illustrates the correct sequence of a transaction to maintain database integrity. A sample is reproduced below. Notice that *PROD_QOH* in the second transaction correctly reads the value as 35, because the previous write to *PROD_QOH* was rolled back.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH=35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK*****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH= 35 -30	
7	T2	Write PROD_QOH	5

Test Yourself 4.8

Suppose that the execution of transaction A, transaction B, and transaction C are requested in that order by three different users. These transactions are executed concurrently. Which of the following are true regarding this situation? (Please check all of the following that are true.)

If ACIDS-compliant concurrency control is maintained, the concurrent execution of transactions A, B, and C will be functionally equivalent to the execution of transactions A, B, and C separately in that order.

This is true. This is the main objective of concurrency control, and is also the definition of serializability.

Serializability is ensured by the scheduler.

This is true. The scheduler interleaves the execution of database operations to ensure that concurrent transaction executions occur appropriately.

The scheduler ensures that the computer's processing and I/O systems are used efficiently.

This is true. This is another responsibility of the scheduler.

If transaction isolation is NOT maintained, an update from one transaction may be overwritten by an update from another transaction.

This is true. This problem is referred to as a lost or buried update.

If isolation is NOT maintained, transaction A may read some data updated by transaction B before it is committed and some data updated by transaction B after it is committed.

This is true. This problem is referred to as *inconsistent retrieval*. Another problem that may occur if isolation is not maintained is having one transaction read another transaction's uncommitted data; this is termed a *dirty read*.

Locking Methods and How They Work

Introduction

A lock guarantees exclusive use of a data item to a current transaction. A transaction acquires a lock prior to data access. When the transaction is complete, the lock is released making the data item available to subsequent transactions. Data consistency cannot be guaranteed during a transaction; therefore locks prevent other transactions from reading inconsistent data. Any multiuser DBMS (with a few exceptions) automatically initiates and enforces locking procedures. Lock information is managed by the *lock manager*. *Lock granularity* indicates the scale of the database data structures to which the locks apply. Locks can be applied on the database, table, page, row, or field level. A DBMS may use different lock types. Two common types are binary locks and shared or exclusive locks. Binary locks have two states. The first state is locked (1) and the second is unlocked (0). Database objects locked by binary locks are unavailable to other transactions and unlocked objects are open to any transaction. Modern DBMS have shared and exclusive locks, which improve concurrency compared to the older binary locks. A *shared lock* exists when concurrent transactions are granted READ access. These locks produce no conflict for read-only transactions. Shared locks are issued when a transaction wants to perform a read and an exclusive lock is not held on the data item. Exclusive locks are used when a transaction wants to update unlocked data.

To recap, a lock is a mechanism used in concurrency control that a database transaction uses to restrict the access of other transactions to a data element. 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.

In a DBMS with only binary locks 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 executing.

In a modern DBMS such as Oracle, a lock and the data that it protects 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 reserved for the transaction that locked the object so that no other transaction can read or change the locked data. 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 (neither shared nor exclusive) are currently held on that data item by any other transaction.*

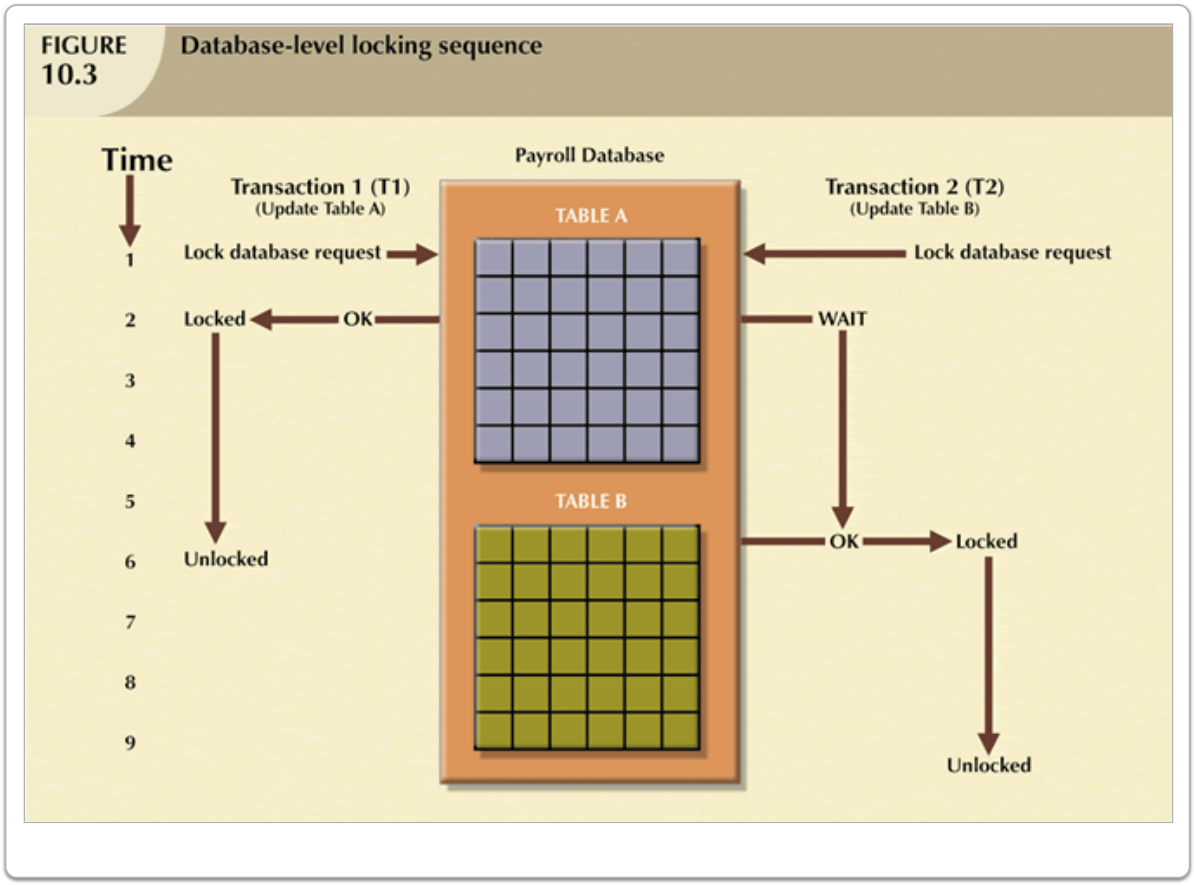
Table 10.9 Read/Write Conflict Scenarios:
Conflicting Database Operations Matrix

	TRANSACTIONS		
	T1	**T2**	**RESULT**
Operations	Read	Read	No Conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

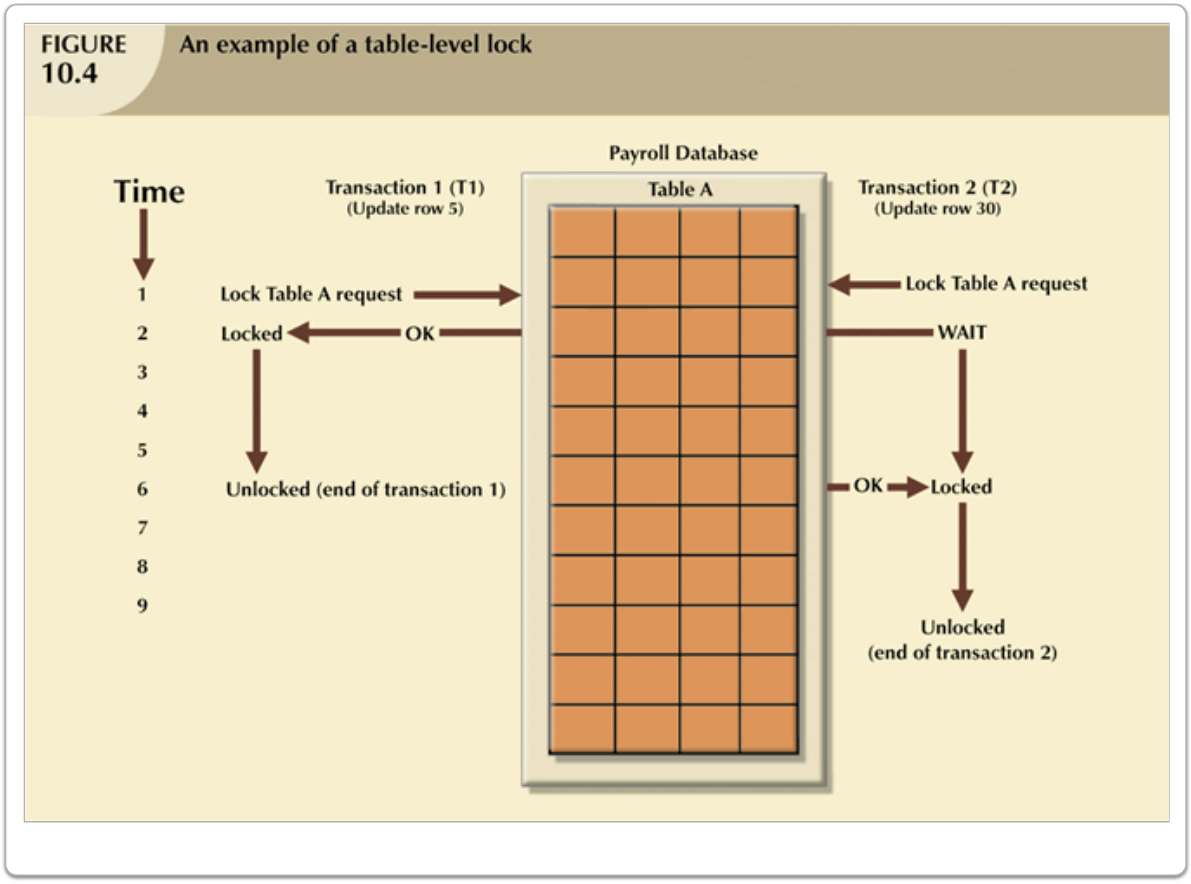
To understand the reasons for having an exclusive lock, look at its counterpart, the shared lock. Shared locks are appropriate when concurrent transactions are granted READ access on the basis of a common lock, because concurrent transactions based on a READ cannot produce a conflict.

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. The diagram above illustrates how conflicts can occur.

In modern DBMS locking can take place at the database level, table level, block (or page) level and row level. Let us compare a database lock with a table lock.



The following is an example of a table level lock.



There are tradeoffs that influence the optimum selection of the lock granularity. Coarser locks lock more data, which reduces the overhead of procuring and releasing locks. It also reduces the number of locks needed. Since locks are a memory-resident structure built on operating system primitives called semaphores, it is important to minimize the number of locks needed. Accordingly coarser locks such as block-level locks are used when the DBMS needs to do something like a table scan, which is examining all of the data in a table. There is a problem with larger granularity locks, which is that they can unnecessarily reduce concurrency. For example if a transaction has a block-level lock on a portion of a table when it needed to access a single record in the table, then this would also lock other rows in the same block in the table, preventing other transactions from accessing those other rows, and accordingly reducing concurrency and database performance. There is also a tradeoff from using too small a lock granularity. For example, if a DBMS were to lock an individual data item, then it may need to procure many locks to support the update of a single row. Since many database operations influence multiple attributes in a row, and the cost per lock is significant, no modern mainstream DBMS support locks on individual data items, and the smallest common lock granularity is the row level lock.

Test Yourself 4.9

Which of the following are correct regarding database locking? (Please check all of the following that are correct.)

In a DBMS with only binary locks, transaction A, which only reads data element X, has a lock on data element X. Transaction B, which only reads data element X, can access data element X while it is locked by transaction A.

This is false. A binary lock is either locked or unlocked. Whether a transaction reads or updates is irrelevant with a binary lock. Transaction B could not access data element X while transaction A still has the lock.

Transaction A has a shared lock on data element X. Transaction B, which only reads data element X, requests a shared lock on data element X. A shared lock is issued to transaction B.

This is true. A shared lock is issued to a transaction when there is no exclusive lock held on the data element. Both transactions can read data element X without conflict.

Transaction A has a shared lock on data element X. Transaction B, which updates data element X, requests an exclusive lock on data element X. An exclusive lock is issued to transaction B.

This is false. An exclusive lock is issued to a transaction when there is no lock held on the data element. In this situation transaction A has a shared lock, so transaction B will need to wait until the lock is released.

Transaction A has an exclusive lock on data element X. Transaction B, which only reads data element X, requests a shared lock on data element X. A shared lock is issued to transaction B.

This is false. A shared lock is issued to a transaction when there is no exclusive lock held on the data element. Transaction A has an exclusive lock on data element X, so transaction B must wait for the lock to be released.

Transaction A has an exclusive row level lock on a row in table T. Transaction B requests an exclusive row level lock on a different row in table T. The lock is issued to transaction B.

This is true. The granularity of the lock is at the row level, so the lock is issued. If transaction A had an exclusive table-level lock on table T and transaction B requested an exclusive lock on the same table, it would not be issued.

Database Recovery Management to Maintain Integrity

Database recovery management restores a database from a given state, usually inconsistent, to a previously consistent state. Recovery techniques are based on the atomic transaction property, which states all portions of the transaction must be treated as a single unit of work. All portions of a transaction must be completed to produce a consistent database. If any transaction operation cannot be completed, the transaction must be aborted and all changes made to the database made by the transaction must be rolled back. Failures other than those at the transaction level might also occur. Software, hardware, programming, and other conditions can also result in a database failure. For these reasons, a DBMS often provides automatic database backup capability. These backups save databases to secondary storage devices such as hard disks or RAID storage. The level of backup varies, with categories such as full backup, differential backup, and transaction log backup.

Backup and recovery are critical functions that allow today's DBMS to maintain its integrity. A DBMS may provide functions that allow the database administrator to perform and schedule automatic database backups to permanent secondary storage devices, such as disks or tapes.

From the database point of view, there are three levels of backup:

- A full backup of the database, or dump of the database. This is a complete backup of the entire database. A full backup was traditionally performed overnight or on weekends, though with modern storage it can be performed at any time without interrupting operations. A full backup requires that there be no data that is not written to durable storage. Accordingly the database is *quiesced* at the moment of the backup. This operation flushes all modified database blocks in memory to durable

storage. A full backup stores all of the data in the database, so creating a full backup takes the longest time and requires the most storage. Recovering a database to the database state when the backup was made does not require recovery of the transaction logs. Depending on the size of the database and the speed of the hardware a full backup can take from a fraction of a second to many days.

- In a differential (or incremental) backup of the database, only the modifications to the database since the last full backup are copied to the backup device. The differential backup will back up only the data that has changed since the last full backup and that has not been backed up before. Depending on the amount of changed data and the hardware, an incremental backup can be completed within a time ranging from several seconds to overnight. In most applications it is problematic when an incremental backup takes longer than overnight.
- A backup of the transaction log only - backs up all the transaction log records that are not reflected in a previous backup copy of the database. Transaction logs are so important for complete database recovery that transaction logs are normally written to multiple storage devices as part of the transaction commit operation.

Early DBMSs has to be shut down so that they could be backed up. You may have encountered this as an announcement that a system will be down for backup. Today many businesses run nonstop, and these outages can cause expensive business interruptions. Consequently, modern DBMSs, such as Oracle, support *hot backup* that permits a database to be backed up from the transaction logs while transactions are being processed. There is also advanced backup technology in storage systems such as EMC's Symmetrix that snapshots a database in a second or so, after which the snapshot can be backed up offline while database operations continue.

How do you restore a database?

Suppose that you are administering a database and something happens that destroys the data on the storage devices used to store the tables and other durable objects in the database. How do you recover the database? The first step is to identify and locate the most recent full backup. If you have been prepared you have a copy of the primary backup on an independent storage device on the premises, though perhaps not in the same building. You will also have an offsite copy of the full backup stored in a secure location. You will next identify and locate the most recent incremental backup, which should also be both on-site and off-site. Then you identify and locate copies of all of the transaction logs since that most recent incremental backup. You should have several of these, both the ones created by the DBMS and transaction logs saved as part of the incremental backup process. After preparing space for the restoration of the database you will then restore the primary backup. This can take a long time if the database is large and the backup is made on a slow device such as a tape drive. Restoring the largest databases from primary backup can take a month. When you have restored from the primary you next restore from the latest incremental backup. This will restore all transactions that committed between the commit point of the primary backup and the commit point of the incremental backup. Next you replay the transactions in the transaction logs that were written after the commit point of the last incremental, but before the commit point to which you wish to restore the database. Note that this restore commit point may not be the last commit point, because what destroyed the database may have been an erroneous or malicious operation, and you want to restore to the commit point before that operation. DBMS vendors provide tools for editing transaction logs to handle these situations.

A full recovery of a large database using the standard procedures outline above can force a considerable business interruption. Accordingly critical databases are implemented using distributed database technologies such as replication and mirroring. These technologies reduce or eliminate the downtime for the restore. These and other more advanced topics are covered in CS779 Advanced Database Management.

Test Yourself 4.10

Which of the following are correct regarding database recovery? (Please check all of the following that are correct.)

In general a differential backup takes longer than a full backup.

This is false. A full backup stores all of the data in the database, so creating a full backup takes the longest time and requires the most storage. In a differential backup of the database, only the modifications to the database since the last full backup are copied.

A transaction log backup may contain records of transactions not reflected in any full or differential backup.

This is true. If the transaction log backup is done after a full or differential backup and any transactions were executed between the backups, then those transactions would only be in the transaction log and transaction log backup. A transaction log backup backs up all the transaction log records that are not reflected in a previous backup copy of the database.

In many commercial DBMS it is possible to back up a database from the transaction logs while transactions are being processed.

This is true. This is referred to as a hot backup. Oracle and other leading commercial DBMS support this, but not all DBMS.

Recovery of a database to the last commit point may require the use of a full backup, a differential backup, and transaction logs.

This is true. These would be used in that order.

Recovery of the database is always to the last commit point.

This is false. You may not want to restore the database to the last commit point, because the reason for the restore may be to recover a database corrupted by an erroneous or malicious operation. Recovery of the database is implemented by restoring to a commit point before the operation that corrupted the database.

Summary

A *transaction* is a collection of operations against a database. Transactions usually represent events, such as sales, in the enterprise. Transactions are *atomic*, meaning that all operations of a transaction must be completed, or none. Transactions take databases from consistent states where all integrity constraints are satisfied to new consistent states. In modern DBMS transactions have the five ACIDS properties: atomicity, consistency, isolation, durability, and serializability. Consistency implies both that the database integrity constraints are satisfied. Isolation means that the results of a transaction are not visible to other transactions until the first transaction is committed, and they will then only be visible to transactions that start after the commit point. Durability means that committed changes are preserved even in the event of recoverable system or database failures. Serializability means that transactions execute in a fashion that is functionally equivalent to having the transaction's components executed in the order that they are presented to the DBMS.

Transactions are supported in SQL with COMMIT and ROLLBACK. Changes to the database are recorded in the transaction logs, which are critical for recovering the transactions in the event of system failures. Modern DBMS execute many transactions concurrently, with the DBMS scheduler responsible for assuring isolation and serializability, using locking, time stamping and optimistic concurrency control methods.

Locks are software objects used to control simultaneous access to database data objects. Early DBMS used binary locks, which have two states: locked or unlocked. If a transaction has a binary lock on a data object, then another transaction cannot access the object until the first transaction releases the lock. Binary locks have been replaced by shared/exclusive locks, which support higher levels of concurrency. A shared/exclusive lock on a database data object can have three states: unlocked, locked exclusive by one transaction, or locked shared by one or more transactions. Exclusive locks are like binary locks, and they are used when a transaction will change the locked data object. Shared locks are used when a transaction wants to read a data object. Because most data accesses are reads and any number of transactions can hold a shared lock, this improves concurrency.

Schedulers use two-phase locking to assure serializability. In two-phase locking a transaction procures all needed locks, performs all operations, and then frees all locks. Deadlocks occur when two or more transactions each hold locks needed by the others before they can proceed. Deadlocks are managed differently by different DBMS. Time stamping plays an important role in all concurrency control designs, including those based on locks. Modern high performance DBMS such as Oracle use optimistic concurrency control, because it performs better.

Database recovery restores the database to a previous consistent state. Database recovery is usually based on primary database backups, incremental backups, and transaction logs.

Lecture 12 - Database Programming

Database Programming with PL/SQL

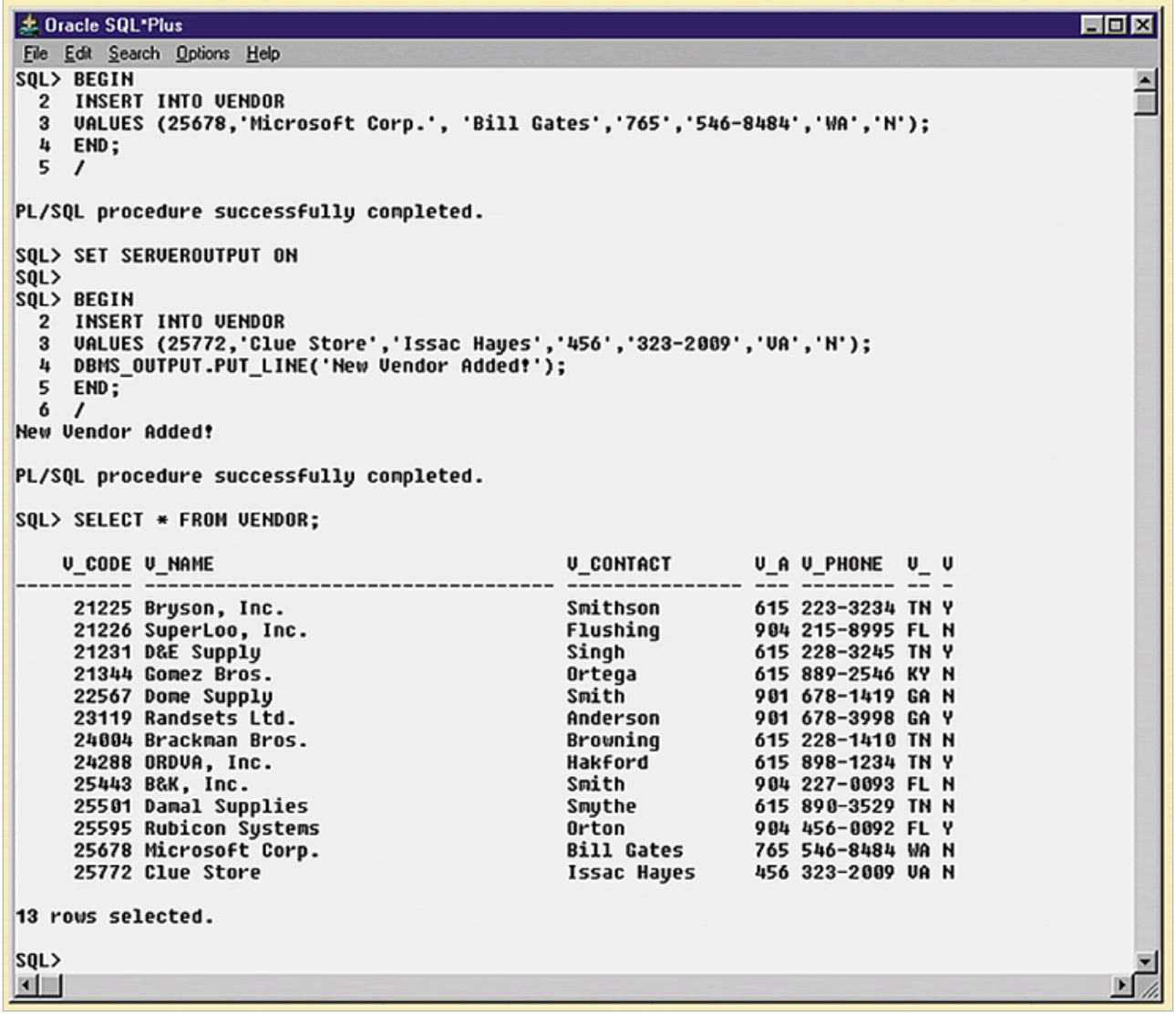
Introduction

All industrial-strength DBMS include a database programming language. PL/SQL is the programming language developed by Oracle. Database programming languages such as TransactSQL, which is supported by Microsoft SQL Server and Sybase, are similar. Database programs are used to write triggers, stored procedures, stored functions, and object methods. Database programming is important for databases that support large numbers of applications, for advanced database security, for efficient execution of conditional operations, and for data-intensive and compute-intensive operations such as loading data warehouses.

PL/SQL Blocks

A *PL/SQL block* is a piece of code that is sent to the database, where it is executed once, and not saved for future use the way a stored procedure or function is stored. A PL/SQL block is *anonymous*, meaning that it has no name. A PL/SQL block can't be called again, so we don't need to give it a name by which we can be refer to it in future calls. A PL/SQL block can begin with the word DECLARE or with the word BEGIN, and it ends with the word END. The following figure from the text illustrates some very simple PL/SQL blocks.

FIGURE 8.28 Anonymous PL/SQL block examples

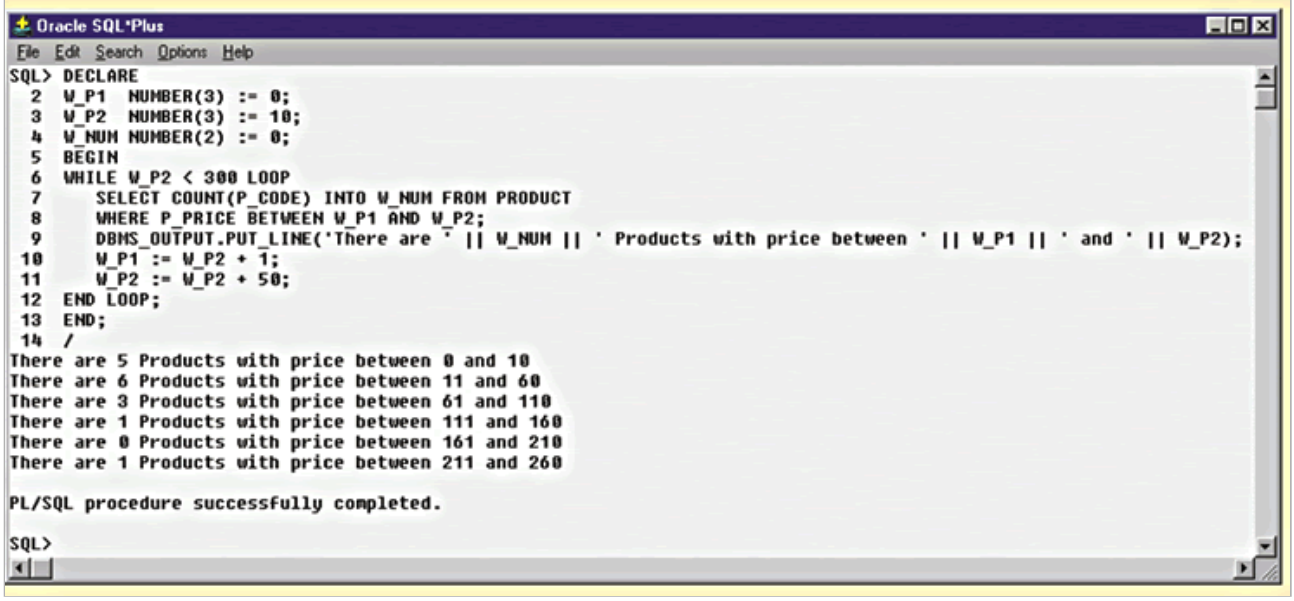


The example at the top just illustrates that Oracle will accept a single INSERT statement even if it is preceded by BEGIN and followed by END, and that when the block has executed Oracle and SQLPlus tell us that a procedure has successfully completed, even though it is not a procedure but just a PL/SQL block. The second example shows us that we can include a call to the PUT_LINE procedure in the DBMS_OUTPUT built-in package inside the PL/SQL block. Such a procedure call is not legal as a SQL statement, so this tells us that we can do more inside of a PL/SQL block than we can in just SQL. Note the line SET SERVEROUTPUT ON. This enables the output of the DBMS_OUTPUT package. Without this line the block would have executed just fine, but the line "New Vendor Added!" would not have appeared in SQLPlus.

A More Complex Example

We now examine a PL/SQL block of more interesting complexity, from the text:

FIGURE 8.29 Anonymous PL/SQL block with variables and loops



Note that the variables declared in the declarations section are initialized; this is a good practice. Notice the SELECT...INTO syntax. There is no standard output such as the user screen for PL/SQL to select a result set into, so SELECT statements in PL/SQL must always have the INTO clause. Notice the BETWEEN clause; this is shorthand for two WHERE clauses specifying the upper and lower bounds. Notice the two vertical bar characters in line 9; these two characters together comprise the SQL string concatenation operation. If you have been attentive you will realize that this session must have executed SET DBMS_OUTPUT ON before this code was executed, because the DBMS_OUTPUT.PUT_LINE procedure results in output to the SQLPlus screen. Notice the PL/SQL WHILE loop, which executes everything between the LOOP and END LOOP as long as the condition (W_P2 < 300 in this case) is true.

This topic is covered in more detail in Advanced Database Management (MET CS 779) and you will find an assignment this week that takes you step-by-step through a PL/SQL programming example.

Test Yourself 4.11

Which of the following is correct regarding PL/SQL? (Please select all of the following that are correct.)

All industrial-strength DBMS use PL/SQL.

This is false. All industrial-strength DBMS include a database programming language. PL/SQL is the programming language developed by Oracle. There are similar database programming languages used by other DBMS.

An anonymous PL/SQL block can be called, just like a predefined function, in multiple SQL statements without rewriting all of its steps.

This is false. An anonymous PL/SQL block cannot be called multiple times. (There is no name to call it with, as opposed to a stored procedure that is named and saved.)

A PL/SQL block must begin with the word BEGIN and end with the word END.

This is false. A PL/SQL block may begin with the word DECLARE or the word BEGIN.

Variables may be used inside a PL/SQL block.

This is true. Loops may be used inside a PL/SQL block also.(The ":=" symbols in the example are being used to assign values to variables inside the block.)

The variable PLAYER_NUM is declared and initialized in the declarations section of a PL/SQL block. The statement "SELECT COUNT(*) INTO PLAYER_NUM FROM PLAYER" inside the PL/SQL block (Between the word BEGIN and the word END) would assign a value equal to the number of rows in the player table to the variable PLAYER_NUM.

This is true. This is the correct syntax for accomplishing this within a PL/SQL block.

How to Create and Use Triggers and Stored Procedures

A persistent stored module (PSM) is a block of code (containing standard SQL statements and procedural extensions) that is stored and executed at the DBMS server. The PSM may represent business logic that can be encapsulated, stored, and shared among multiple database users. A *trigger* is procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a given data manipulation event. A trigger is associated with a database table and executed in response to modifications to that table. The trigger can be configured to execute before, during or after a database change. A *stored procedure* is a named collection of procedural and SQL statements. As with database triggers, stored procedures are stored in the database. One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions. Stored procedures can substantially reduce network traffic and increase performance. When used properly, they can also help reduce code duplication by means of code isolation and code sharing, minimizing the chance of errors and the cost of application development and maintenance. Like procedures in other programming languages, stored procedures can take and return arguments.

Although persistent stored modules can implement business logic, particularly logic that applies to all data in the database, the inclusion of extensive business logic in the database can compromise the layering that is fundamental to the architecture of modern tiered applications. In modern tiered system design the functionality is allocated to layers depending on the role each component in the overall application system. In strictly layered applications the roles of the database are maintaining data integrity and data persistence, and business logic is localized in the business logic layer.

These topics are covered further in MET CS 779 Advanced Database Management and in MET CS 674 Database Security.

Test Yourself 4.12

Which of the following is true? (Please select all of the following that are true.)

A persistent stored module is executed at the DBMS server.

This is true. A persistent stored module (PSM) is a block of code that is stored and executed at the DBMS server. Triggers and stored procedures are persistent stored modules.

A trigger is executed when called or referred to by name in a SQL or PL/SQL statement.

This is false. A trigger is associated with a database table and executed in response to modifications to that table.

A stored procedure can be executed by a command that refers to it by name.

This is true. A stored procedure is a named collection of procedural and SQL statements. (Executed by referring to that name as opposed to a trigger.)

Stored procedures can return values.

This is true. Stored procedures can take and return arguments.

The use of a stored procedure can decrease the amount of code that needs to be sent to a DBMS server to accomplish a database task or transaction.

This is true. The stored procedure is stored and executed at the DBMS server. The code required to accomplish the task or transaction specified by the stored procedure does not have to be transmitted each time the task or transaction occurs. (Only the command or statement that executes the stored procedure needs to be sent to the DBMS server.) This can result in a significant difference in transmission requirements. (Reduce network traffic)

How to Create and Use Updatable Views

An updatable view updates attributes in the base tables that are used in the view. Not all views are updatable and restrictions exist on updatable views. The most common updatable view restrictions are: you cannot use GROUP BY expressions or aggregate functions in updatable views and you cannot use set operators such as UNION, INTERSECT, and MINUS. Most restrictions are based on the use of JOINs or group operators in views. One easy way to determine if a view can be used to update a base table is to examine the view's output. If the primary key columns of the base table you want to update still have unique values in the view, the base table may be updatable. Whether a view is updatable depends on the DBMS and even on the version of the DBMS, so it is wise to verify that your DBMS supports updating a particular view, and not make assumptions.

Views based on joins and "advanced SQL" commands may not be updatable, depending on the DBMS. If you wish to update a value presented by a view and there is not an unambiguous mapping of the value to a single value in a single table, then that value cannot be updated in the view in any DBMS. For example, a value in a view computed by an aggregate function cannot be updated in any DBMS.

Test Yourself 4.13

Which of the following is correct regarding views? (Please check all of the following that are correct.)

Making changes to data in an updatable view can cause data stored in a table to change.

This is true. An updatable view allows you to make data changes in a table that the view is based on by changing the corresponding data in the view.

All views are updatable by definition.

This is false. Not all views are updatable. Common restrictions are views based on aggregate functions and on set operators such as UNION, INTERSECT, and MINUS.

A view containing only aggregate values has to be updatable.

This is false. If a view contained an average or a sum of a base table column and you changed this value in the view, how would the DBMS determine which individual fields in that column to change and what values to change these to?

Whether a particular view is updatable may depend on the DBMS.

This is true. While some views, such as those based on aggregates, are theoretically not updatable, because there is no place to change the data, even the best DBMS do not support updates of some view types that are theoretically updatable. The restrictions may even vary with the database version.

Surrogate Keys in Practice

We previously learned that a *surrogate* key is a primary key with programmatically generated values that have no intrinsic meaning in the table or schema containing the key. Often for consistency and simplicity, a surrogate key spans only one column that has an integer datatype, with a maximum value corresponding to the maximum number of rows that will ever exist in the table. While there are many ways to programmatically generate a surrogate key's values, probably the most common method is to utilize the database management system's support for generating unique integers.

Sequences

Modern relational DBMS that support the ANSI SQL standards, including Oracle and SQL Server, provide for *sequences*, the preferred construct for generating surrogate keys. A sequence is a separate schema object that multiple users may use to generate unique integers. Just as tables are schema objects designed to store data, sequences are schema objects designed to generate unique integers. Typically, a different sequence is used to create values for each differentsurrogate key. For example, if we have a *Customer* table which has a *customer_id* surrogate key, we might create a sequence named *Customer_seq* or *Customer_id_seq*, and use that sequence to generate the values for the *customer_id* column. As long as we always use that sequence to generate *customer_id* values, the values will always be unique in the context of the *Customer* table.

To create a sequence, we use the command:

```
CREATE SEQUENCE sequence_name MINVALUE 1;
```

This command creates a sequence that starts with a value of 1. And by default, as we expect, each new value is obtained by incrementing the previous value by 1. The first few values generated from the sequence would thus be 1, 2, 3, 4, and you can probably guess what the remaining numbers will be.

Following the Customer example above, the command to create the sequence would be:

```
CREATE SEQUENCE Customer_seq MINVALUE 1;
```

The syntax for obtaining the next unique value from a sequence differs per DBMS. For Oracle, we use a dot (.) followed by the keyword *NEXTVAL*. We can use the value from the sequence any place in a SQL command where an integer value is legal; however, in practice the sequence values are used in conjunction with INSERT commands. For example:

```
INSERT INTO Customer (customer_id, ...)
VALUES (Customer_seq.NEXTVAL, ...);
```

The syntax for obtaining the next unique value from a sequence in SQL Server is using the words *NEXT VALUE FOR* followed by the sequence name. For example:

```
INSERT INTO Customer (customer_id, …)
VALUES (NEXT VALUE FOR Customer_seq, …);
```

In Oracle, if you need to reference the last value you had obtained from a sequence, you may use the keyword *CURRVAL*. For example, if we need to create a unique value for *customer_id*, then reference that unique value in another statement, we could do so as follows:

```
INSERT INTO Customer (customer_id, ...)
VALUES (Customer_seq.NEXTVAL, ...); --Generate the unique integer

INSERT INTO Address (customer_id, ...)
VALUES (Customer_seq.CURRVAL, ...); --Recall the unique integer
```

SQL Server has no direct, counterpart command to atomically retrieve the current value from a sequence.

SQL Server Identity Columns

In SQL Server versions prior to SQL Server 2012, sequences are not supported; *identity columns* are the preferred method of generating surrogate key values in these versions. An identity column in SQL Server automatically generates unique integers for a row when it is inserted. The column is identified as an identity column when the table is created. There may be only one identity column per table, and that column's datatype must be one of the many available integer datatypes. When a row is inserted, no value is specified for the identity column, and SQL Server automatically inserts the next unique value.

For example, imagine a *Customer* table that has been created with a *customer_id* identity column, and a *last_name* column. To generate the next unique value, you would simply insert a row with no value specified for *customer_id*:

```
INSERT INTO Customer (last_name)
VALUES ('Smith'); --The customer_id value will generated by SQL Server
```

In SQL Server 2012 and later versions, the use of sequences is recommended over the use of identity columns, as sequences offer better performance and are more configurable.