**Chapter 9**

**Database Administration**

**Database Administration Tasks**

**DBA Responsibilities**

**Concurrency Control**

**Transactions**

**Locks**

**Data read problems**

**Transaction Isolation levels**

**Cursor types**

**Database security**

**SQL Data Control Language**

**Application Security**

**SQL Injection Attack**

**Database recovery**

**Rollback/Rollforward**

**Checkpoint**

**Transaction Log**

**Managing and Maintaining the DBMS/Data repository**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Database Administration**

• All large and small databases need database administration.

• Data administration refers to a function concerning all of an organization’s data assets.

• Database administration (DBA) refers to a person or office specific to a single database and its applications.

**Database Administration Tasks**

• Manage database Structure

• Control Concurrent Processing

• Manage processing rights and responsibilities

• Develop database security

• Provide for database recovery

• Manage the DBMS

• Maintain the data repository

**DBA Responsibilities**

• Participate in Database and Application Development

- Assist in the requirements analysis stage and data model creation

- Play an active role in database design and creation

• Facilitate Changes to database Structure

|  |  |  |
| --- | --- | --- |
| - Seek communitywide solutions | - Assess impact on all users | - Provide configuration control forum |
| - Be prepared for problems after changes are made | - Maintain documentation |  |

**Concurrency Control**

• **Concurrency control** ensures that one user’s work does not inappropriately influence another user’s work.

– No single concurrency control technique is ideal for all circumstances.

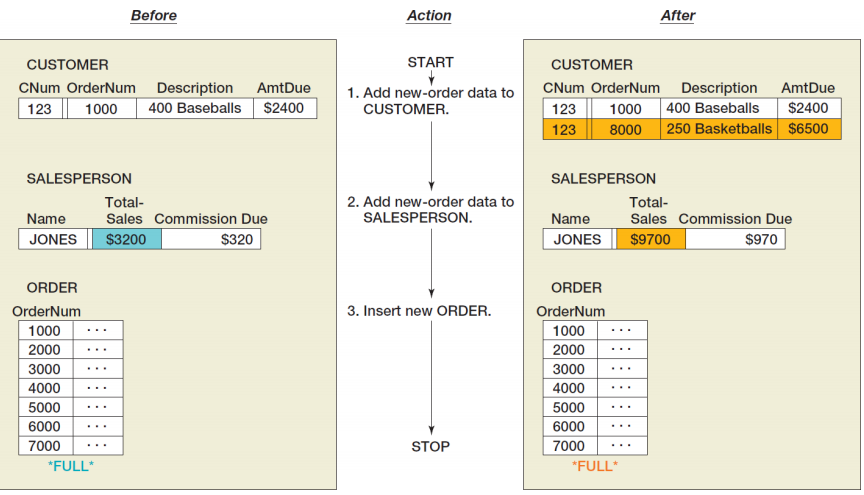
– Trade-offs need to be made between level of protection and throughput.

**Atomic Transactions**

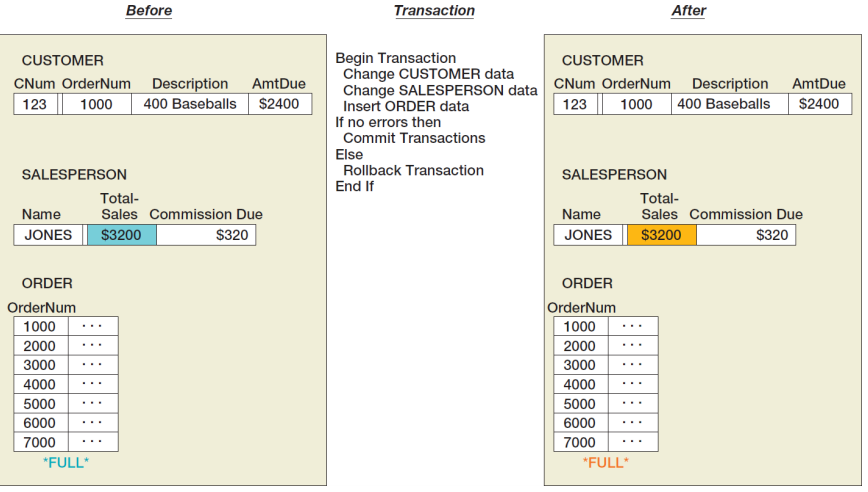
• A **transaction**, or **logical unit of work (LUW),** is a series of actions taken against the database that occurs as an atomic unit:

– Either all actions in a transaction occur or none of them do.

**Errors introduced without Atomic Transactions**



**Errors Prevented with Atomic Transaction**



**Concurrent Transaction**

• **Concurrent transactions** refer to two or more transactions that appear to users as they are being processed against a database at the same time.

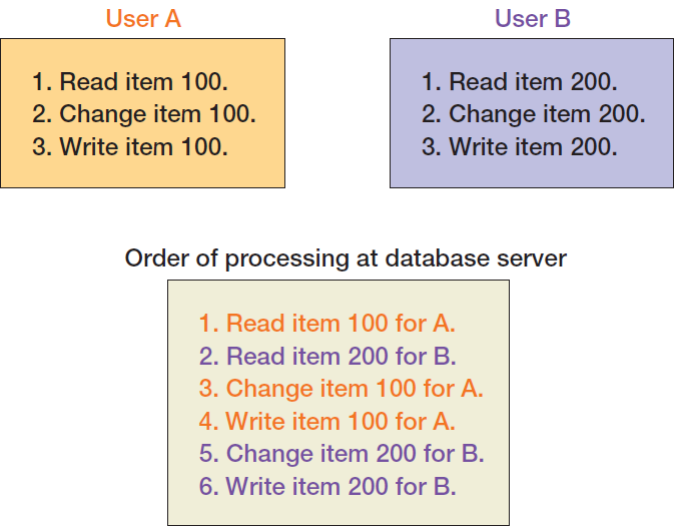
• In reality, CPU can execute only one instruction at a time.

– **Transactions are interleaved**:

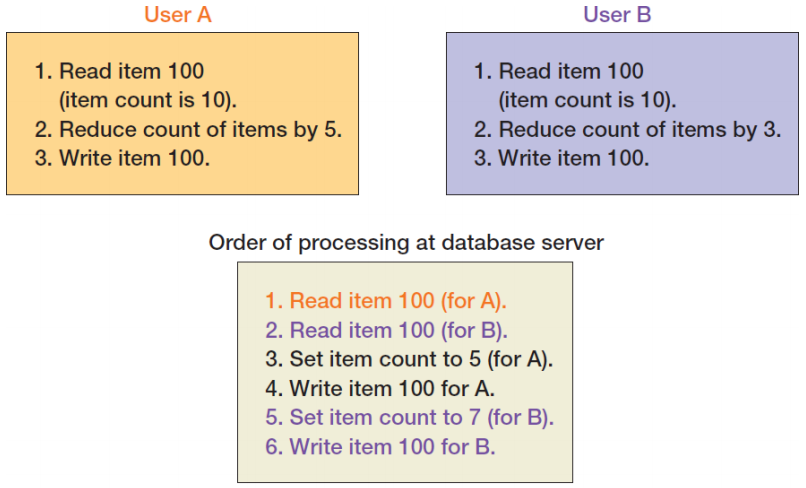
• The operating system quickly switches CPU services among tasks so that some portion of each of them is carried out in a given interval.

• Concurrency problems are lost updates and inconsistent reads.

**Concurrent Transaction Processing**



**Lost Update Problem**



**Resource Locking**

• **Resource locking** prevents multiple applications from obtaining copies of the same record when the record is about to be changed.

**Lock Terminology**

• **Implicit locks** are locks placed by the DBMS.

• **Explicit locks** are issued by the application program.

• **Lock granularity** refers to size of a locked resource:

– Rows, page, table, and database level.

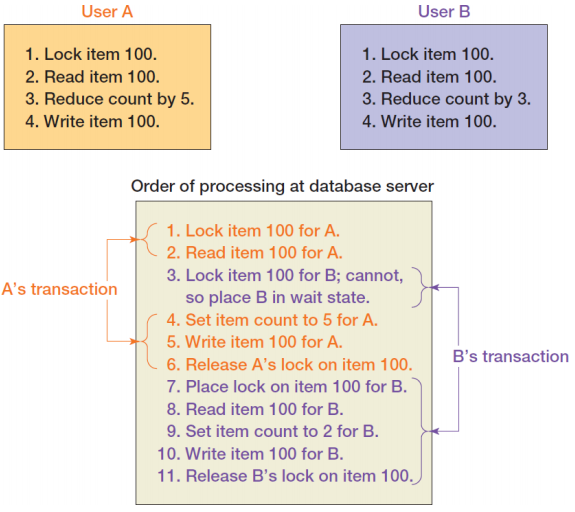
• Large granularity is easy to manage but frequently causes conflicts.

• Types of lock:

– An **exclusive lock** prohibits other users from reading the locked resource.

– A **shared lock** allows other users to read the locked resource, but they cannot update it.

**Concurrent Processing with Explicit Locks**



**Serializable Transactions**

• **Serializable transactions** refer to two transactions that run concurrently and generate results that are consistent with the results that would have occurred if they had run separately.

• **Two-phased locking** is one of the techniques used to achieve serializability.

**Two-phased locking**

• **Two-phased locking** – Transactions are allowed to obtain locks as necessary (**growing phase**). – Once the first lock is released (shrinking phase), no other lock can be obtained.

• A special case of two-phased locking

|  |  |  |
| --- | --- | --- |
| – Locks are obtained throughout the transaction. | – No lock is released until the COMMIT or ROLLBACK command is issued. | – This strategy is more restrictive but easier to implement than two-phase locking. |

**Deadlock**

• **Deadlock**, or the deadly embrace, occurs when two transactions are each waiting on a resource that the other transaction holds.

• Preventing deadlock:

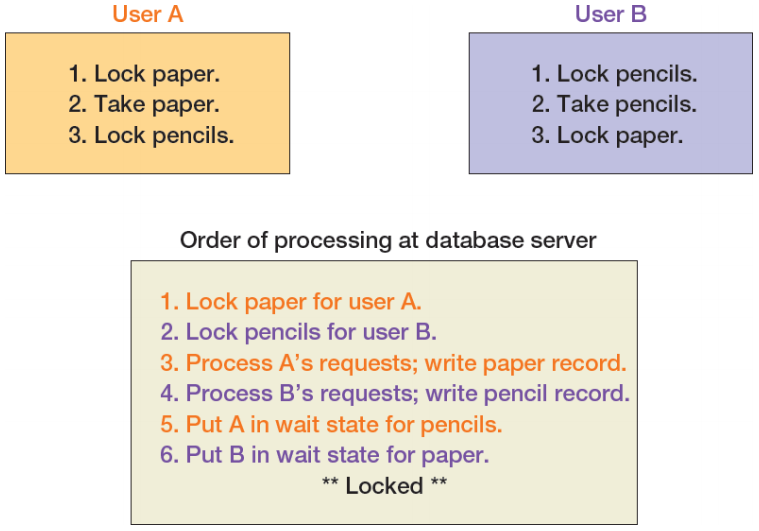
– Allow users to issue all lock requests at one time.

– Require all application programs to lock resources in the same order.

• Breaking deadlock:

– Almost every DBMS has algorithms for detecting deadlock.

– When deadlock occurs, DBMS aborts one of the transactions and rolls back partially completed work.



**Optimistic versus Pessimistic Locking**

• **Optimistic locking** assumes that no transaction conflict will occur and is preferred for the Internet and for many intranet applications.

– DBMS processes a transaction; checks whether conflict occurred:

• If not, the transaction is finished.

• If so, the transaction is repeated until there is no conflict.

• **Pessimistic locking** assumes that conflict will occur.

– Locks are issued before transaction is processed, and then the locks are released.

**Declaring Lock Characteristics**

• Most application programs do not explicitly declare locks due to its complication.

• Instead, they mark transaction boundaries and declare locking behavior they want the DBMS to use.

– Transaction boundary markers (syntax varies with DBMS):

• **SQL BEGIN TRANSACTION statement**

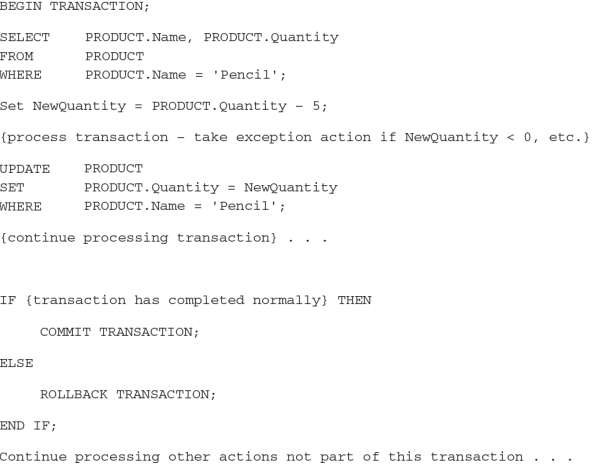
**• SQL COMMIT TRANSACTION statement**

**• SQL ROLLBACK TRANSACTION statement**

• Advantage:

– If the locking behavior needs to be changed, only the lock declaration need be changed, not the application program.

**Marking Transaction Boundaries**



**Implicit and Explicit COMMIT TRANSACTION**

• Microsoft SQL Server 2014 and MySQL 5.6 automatically commit changes after a transaction.

– This is an implicit COMMIT.

• Oracle Database requires an explicit COMMIT statement:

UPDATE CUSTOMER SET AreaCode = ‘425’ WHERE ZIPCode = ‘98050’; COMMIT;

**ACID TRANSACTION**

• Acronym **ACID** transaction is one that is **A**tomic, **C**onsistent, **I**solated, and **D**urable.

• **Atomic** means either all or none of the database actions occur.

• **Durable** means database committed changes are permanent.

• **Consistency** means either statement level or transaction level consistency.

– **Statement level consistency**: each statement independently processes rows consistently

– **Transaction level consistency**: all rows impacted by either of the SQL statements are protected from changes during the entire transaction

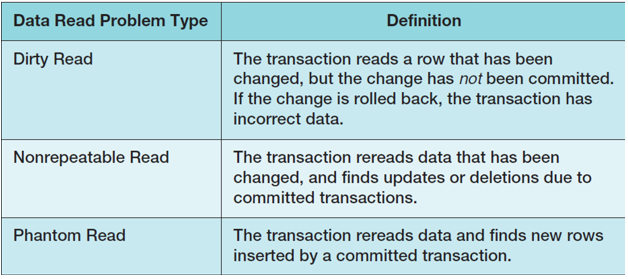
• With transaction level consistency, a transaction may not see its own changes.

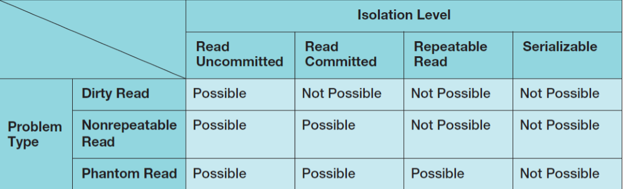
• **Isolation** means application programmers are able to declare the type of isolation level and to have the DBMS manage locks so as to achieve that level of isolation.

• SQL-92 defines four **transaction isolation levels**:

|  |  |  |  |
| --- | --- | --- | --- |
| – Read uncommitted | – Read committed | – Repeatable read | – Serializable |

**Data Read Problems Level**



**Transaction Isolation Level** 

**Cursor type**

• A **cursor** is a pointer into a set of records.

• It can be defined using SELECT statements.

• Four cursor types:

• **Forward only**: the application can only move forward through the recordset.

- Scrollable cursors can be scrolled forward and backward through the recordset.

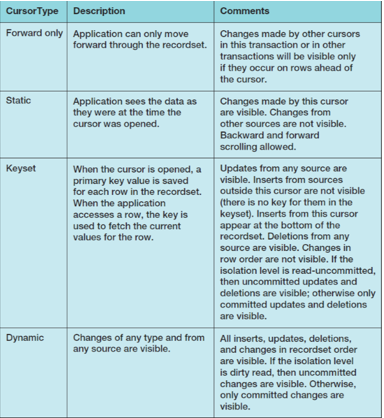
• **Static**: processes a snapshot of the relation that was taken when the cursor was opened

• **Keyset**: combines some features of static cursors with some features of dynamic cursors

• **Dynamic**: a fully featured cursor

• Choosing appropriate isolation levels and cursor types is critical to database design.

**Cursor Summary**



**Database Security**

• **Database security** ensures that only authorized users can perform authorized activities at authorized times.

• Developing database security:

– Determine users’ processing rights and responsibilities.

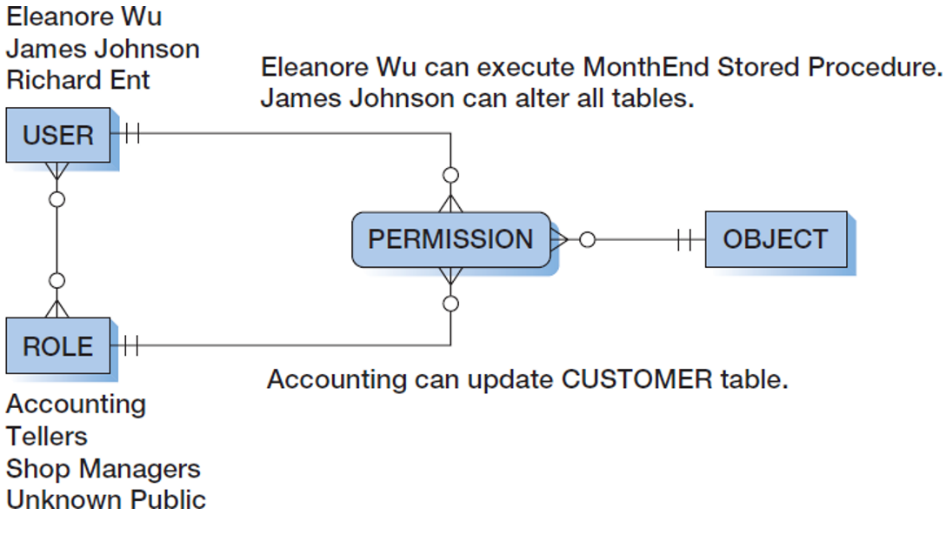
– Enforce security requirements using security features from both DBMS and application programs.

• DBMS products provide security facilities.

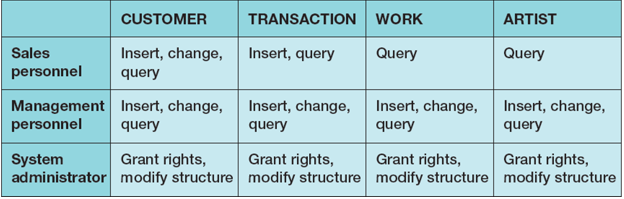
• They limit certain actions on certain objects to certain users or groups (also called roles).

• Almost all DBMS products use some form of username and password security.

**DBMS Security Model**



**View Ridge Gallery Security Model**



**DBMS Security Guidelines**

• Run DBMS behind a firewall, but plan as though the firewall has been breached

• Apply the latest operating system and DBMS service packs and fixes

• Use the least functionality possible

– Support the fewest network protocols possible

– Delete unnecessary or unused system stored procedures

– Disable default logins and guest users, if possible

– Unless required, never allow all users to log on to the DBMS interactively

• Protect the computer that runs the DBMS

– No user allowed to work at the computer that runs the DBMS

– DBMS computer physically secured behind locked doors

– Access to the room containing the DBMS computer should be recorded in a log

• Manage accounts and passwords

– Use a low privilege user account for the DBMS service

– Protect database accounts with strong passwords

– Monitor failed login attempts

– Frequently check group and role memberships

– Audit accounts with null passwords

– Assign accounts the lowest privileges possible

– Limit DBA account privileges

• Planning

– Develop a security plan for preventing and detecting security problems

– Create procedures for security emergencies and practice them

**SQL Data Control Language (DCL)**

Permissions can be managed using **SQL Data Control Language (DCL):**

– The SQL GRANT statement is used to assign permissions to users and groups so that the users or groups can perform various operations on the data in the database.

– The SQL REVOKE statement is used to take existing permissions away from users and groups.

**Application Security**

• If DBMS security features are inadequate, additional security code could be written in application program.

– Application security in Internet applications is often provided on the Web server computer.

• However, you should use the DBMS security features first.

– The closer the security enforcement is to the data, the less chance there is for infiltration.

– DBMS security features are faster, cheaper, and probably result in higher quality results than developing your own.

**SQL Injection Attack**

• An **SQL injection attack** occurs when data from the user is used to modify an SQL statement.

• User input that can modify an SQL statement must be carefully edited to ensure that only valid input has been received and that no additional SQL syntax has been entered.

• Example: users are asked to enter their names into a Web form textbox:

– User input: 'Benjamin Franklin' OR TRUE

– Resulting effective user input:

SELECT \* FROM EMPLOYEE WHERE EMPLOYEE.Name = ‘Benjamin Franklin’ OR TRUE;

**Database Recovery**

• In the event of system failure, that database must be restored to a usable state as soon as possible.

• Two recovery techniques:

– Recovery via reprocessing

– Recovery via rollback/rollforward

**Recovery via Reprocessing**

• **Recovery via reprocessing**: the database goes back to a known point (database save) and reprocesses the workload from there.

• Unfeasible strategy because:

– The recovered system may never catch up if the computer is heavily scheduled.

– Asynchronous events, although concurrent transactions, may cause different results.

**Rollback/RollForward**

• **Recovery via rollback/rollforward**:

– Periodically save the database and keep a database change log since the save.

• Database log contains records of the data changes in chronological order.

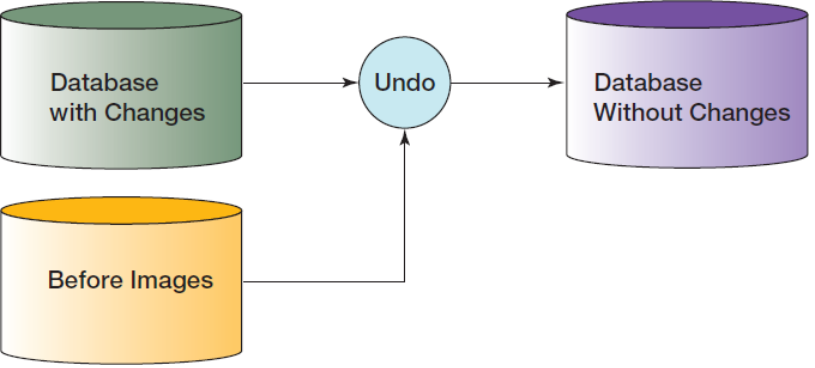
• When there is a failure, either rollback or rollforward is applied.

– **Rollback**: undo the erroneous changes made to the database and reprocess valid transactions

– **Rollforward**: restore database using saved data and valid transactions since the last save

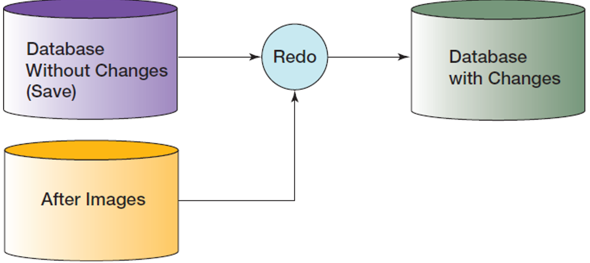
**Rollback**

Before-image: a copy of every database record (or page) before it was changed



**Rollforward**

After-image: a copy of every database record (or page) after it was changed



**Checkpoint**

• A **checkpoint** is a point of synchronization between the database and the transaction log.

– DBMS refuses new requests, finishes processing outstanding requests, and writes its buffers to disk.

– The DBMS waits until the writing is successfully completed  the log and the database are synchronized.

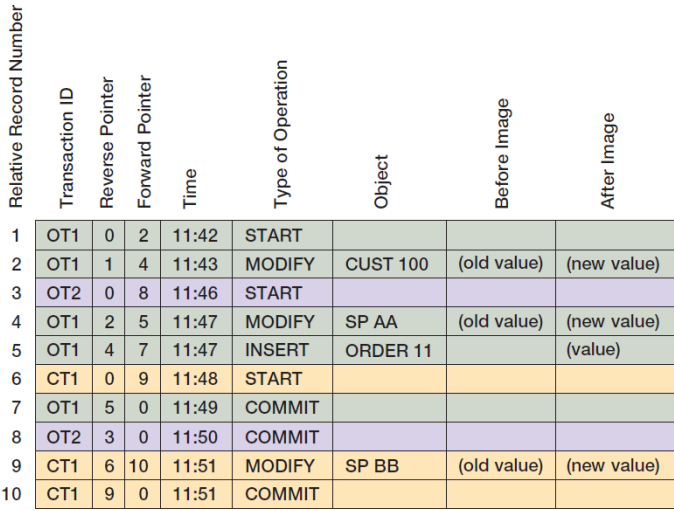
• Checkpoints speed up database recovery process.

– Database can be recovered using after-images since the last checkpoint.

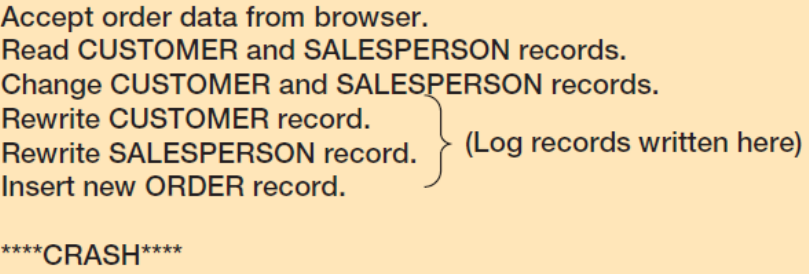
– Checkpoint can be done several times per hour.

• Most DBMS products automatically checkpoint themselves.

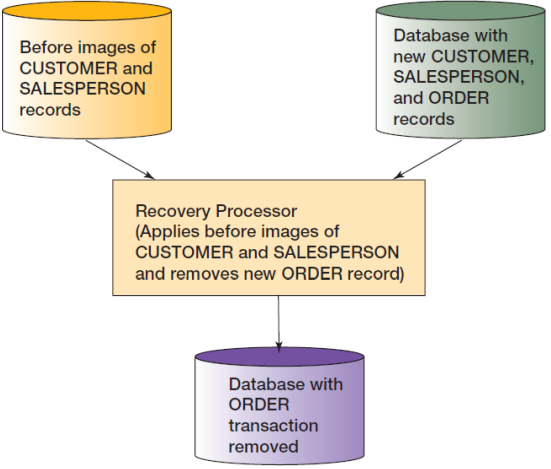
**Transaction Log**



**Database Recovery A Processing Problem Occurs**



**Database Recovery: Recovery Processing**



**Managing the DBMS**

|  |  |  |
| --- | --- | --- |
| • Generate database application performance | • Investigate user performance complaints | • Assess need for changes in database structure |
| • modify database structure | • Evaluate and implement new DBMS features | • Tune the DBMS |

**Maintaining the Data repository**

• DBA is responsible for maintaining the data repository.

• Data repositories are collections of metadata about users, databases, and its applications.

• The repository may be:

– Virtual, as it is composed of metadata from many different sources: DBMS, code libraries, Webpage generation and editing tools, etc.

– An integrated product from a CASE tool vendor or from other companies.

– Active – part of the systems development process

– Passive – documentation only made when someone has time

• The best repositories are active, and they are part of the system development process.