DAVID M. KROENKE and DAVID J. AUER

DATABASE CONCEPTS, 4th Edition

# Database Administration





## Chapter Objectives

- Understand the need for and importance of database administration & Learn different ways of processing a database
- Understand the need for concurrency control, security, and backup and recovery
- Learn typical problems that can occur when multiple users process a database concurrently & understand the use of locking and the problem of deadlock
- Learn the difference between optimistic and pessimistic locking & learn the meaning of *ACID transaction*
- Learn the four 1992 ANSI standard isolation levels

## Chapter Objectives (continued)

- Understand the need for security and learn a generalized model of database security & see the difference between DBMS and application security.
- See the difference between recovery via reprocessing and recovery via rollback/roll-forward & the nature of the tasks required for recovery using rollback/roll-forward.
- Learn basic administrative and managerial DBA functions & understand distributed database processing.
- Understand the concept of object-relational databases.





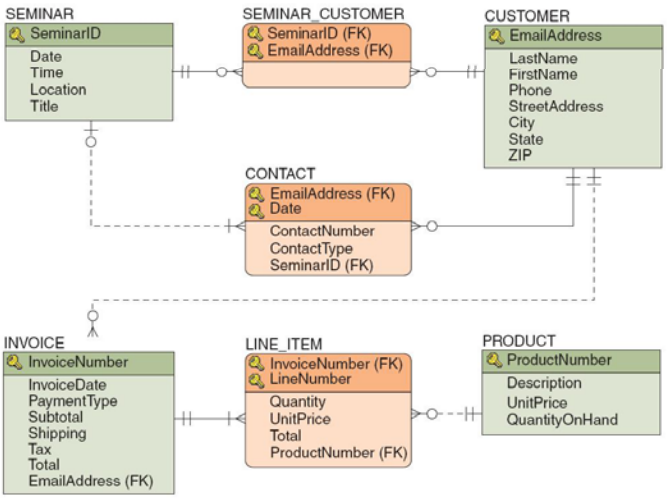
DAVID M. KROENKE and DAVID J. AUER

# DATABASE CONCEPTS, 4th Edition

## Data Administration

## Database Administration

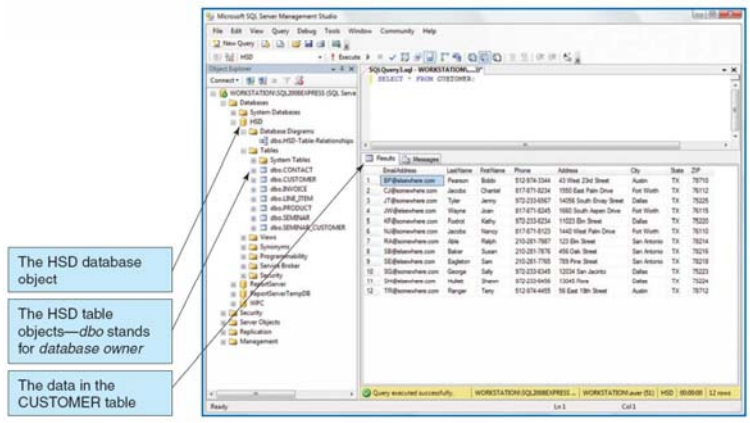
# Heather Sweeney Designs: Database Design



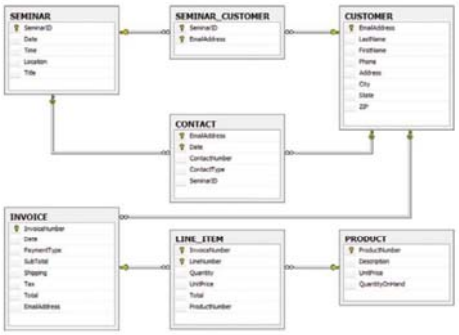





# Heather Sweeney Designs: HSD Database in SQL Server 2008



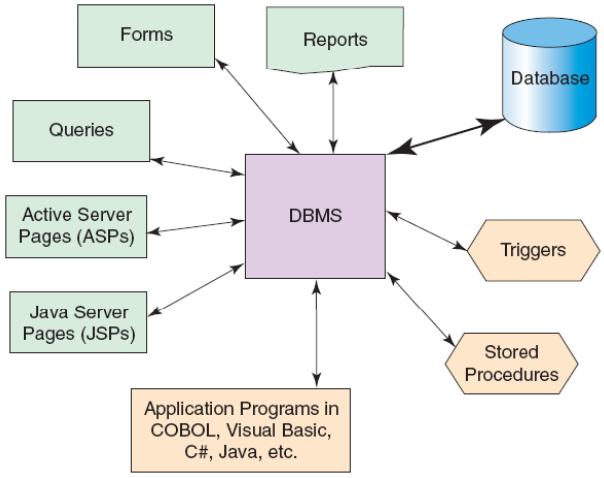
# Heather Sweeney Designs:
## HSD Database Diagram in SQL Server 2008








# The Database Processing Environment



## Control, Security and Reliability

- Three necessary database administration functions;
  - Concurrency control
  - Security
  - Backup and Recovery





## Concurrency Control

- Concurrency control ensures that one user's actions do not adversely impact another user's actions, at the core of concurrency is accessibility
- In one extreme, data becomes inaccessible once a user touches the data
  - This ensures that data considered for update is not shown
- In the other extreme, data is always readable
  - The data is even readable when it is locked for update

## Concurrency Control (continued)

- Interdependency
  - Changes required by one user may impact others
- Concurrency
  - People or applications may try to update the same information at the same time
- Record retention
  - When information should be discarded

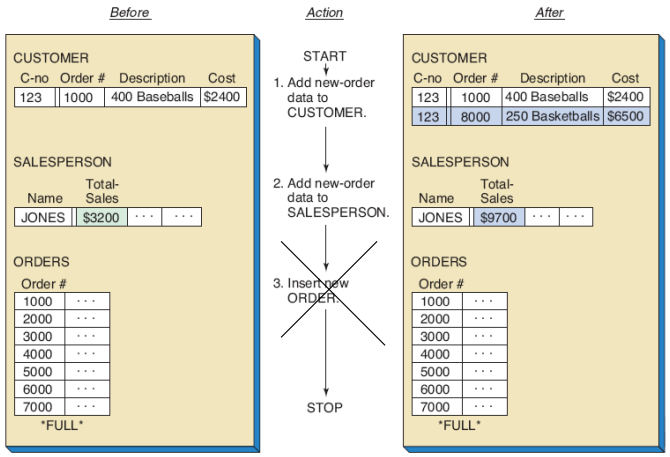




## Need for Atomic Transactions

- A database operation typically involves several transactions
- These transactions are *atomic* and are sometimes called logical units of work (LUW)
- Before an operation is *committed* to the database, all LUWs must successfully complete
  - If one or more LUW is unsuccessful, a *rollback* is performed and no changes are saved to the database

# Transaction Example I
## Recording a New Order

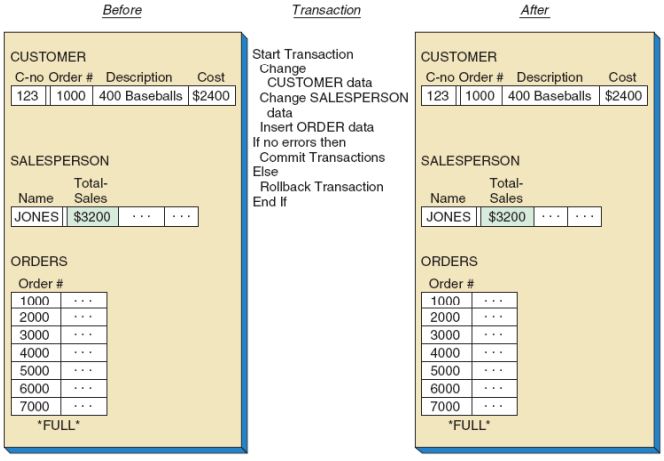



(a) Two out of Three Activities Successfully Completed, Resulting in Database Anomalies





# Transaction Example II




(b) No Change Made Because Entire Transaction Not Successful

# Concurrent Processing Example


User A

1. Read Item 100.
2. Change Item 100.
3. Write Item 100.

User B

1. Read Item 200.
2. Change Item 200.
3. Write Item 200.

Order of processing at database server

1. Read Item 100 for A.
2. Read Item 200 for B.
3. Change Item 100 for A.
4. Write Item 100 for A.
5. Change Item 200 for B.
6. Write Item 200 for B.

No Problem as Two Different Items to be processed at the server requested by two different users.





# Lost Update Problem


- If two or more users are attempting to update the same piece of data at the same time, it is possible that one update may overwrite another update

## Lost Update Problem Example

User A

1. Read Item 100 (assume item count is 10).
2. Reduce count of items by 5.
3. Write Item 100.

User B

1. Read Item 100 (assume item count is 10).
2. Reduce count of items by 3.
3. Write Item 100.

Order of processing at database server

1. Read Item 100 (for A).
2. Read Item 100 (for B).
3. Set item count to 5 (for A).
4. Write Item 100 for A.
5. Set item count to 7 (for B).
6. Write Item 100 for B.

Problem as same Item to be processed at the server requested by two different users and the sequence will matter

Note: The change and write in steps 3 and 4 are lost.





## Concurrency Issues

- Dirty reads
  - One transaction reads a changed record that has not been committed to the database by the second transaction (which he may discard later).
- Inconsistent reads / Non-repeatable read
  - One transaction re-reads a data and finds that the data has changed by another transaction.
- Phantom reads
  - One transaction re-reads a data and finds that a new row/record has been added by another transaction.

## Resource Locking

- To avoid concurrency issues, resource locking will disallow transactions from reading, modifying and/or writing to a data set that has been *locked*

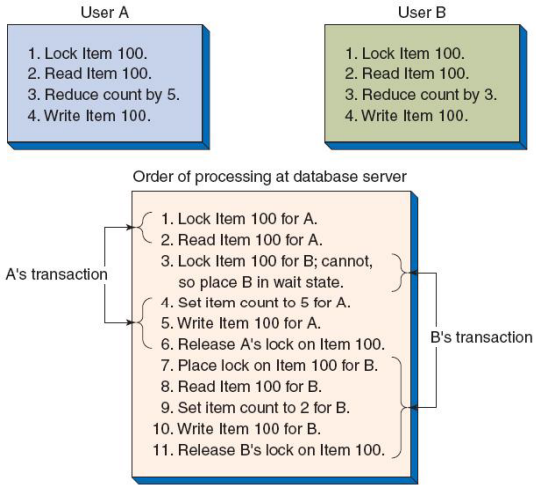




## Implicit versus Explicit Resource Locking

- Implicit locks are issued automatically by the DBMS based on an activity
- Explicit locks are issued by users requesting exclusive rights to the data

## Concurrent Processing with Explicit Locking Example








## Serializable Transactions

- When two or more transactions are processed concurrently, the results in the database should be logically consistent with the results that would have been achieved had the transactions been processed in an *arbitrary serial fashion*
- A scheme for processing concurrent transactions in this way is said to be serializable

## Two-Phased Locking

- One way to achieve serializable transactions is by using *two-phased locking*.
- Two-phased locking lets locks be obtained and released as they are needed:
  1. A growing phase, when the transaction continues to request additional locks.
  2. A shrinking phase, when the transaction begins to release the locks.





## Deadlock

- As a transaction begins to lock resources, it may have to wait for a particular resource to be released by another transaction
- On occasions, two transactions may indefinitely wait on each other to release resources—This condition is known as a deadlock or the deadly embrace

# Deadlock Example

User A

1. Lock paper.
2. Take paper.
3. Lock pencils.

User B

1. Lock pencils.
2. Take pencils.
3. Lock paper.

Order of processing at database server

1. Lock paper for User A.
2. Lock pencils for User B.
3. Process A's requests; write paper record.
4. Process B's requests; write pencil record.
5. Put A in wait state for pencils.
6. Put B in wait state for paper.

** Locked **

Can you think a way so the above deadlock should not happen ?





# Optimistic Locking vs Pessimistic Locking

- Optimistic Locking (Hoping no conflict will occur)
  - Read data
  - Process transaction
  - Issue update
  - Look for conflict
  - If conflict occurred, rollback and repeat
  - Else commit
- Pessimistic Locking (just the oposite)
  - Lock required resources
  - Read data
  - Process transaction
  - Issue commit
  - Release locks

## Optimistic Locking Example

```
SELECT    PRODUCT.Name, PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.Name = 'Pencil'

OldQuantity = PRODUCT.Quantity

Set NewQuantity = PRODUCT.Quantity – 5

{process transaction – take exception action if NewQuantity < 0, etc.

Assuming all is OK: }

LOCK PRODUCT {at some level of granularity}

UPDATE    PRODUCT
SET       PRODUCT.Quantity = NewQuantity
WHERE     PRODUCT.Name = 'Pencil'
      AND PRODUCT.Quantity = OldQuantity

UNLOCK    PRODUCT

{check to see if update was successful;
if not, repeat transaction}
```





## Pessimistic Locking Example

```
LOCK      PRODUCT {at some level of granularity}

SELECT    PRODUCT.Name, PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.Name = 'Pencil'

Set NewQuantity = PRODUCT.Quantity – 5

{process transaction – take exception action if NewQuantity < 0, etc.

Assuming all is OK: }

UPDATE    PRODUCT
SET       PRODUCT.Quantity = NewQuantity
WHERE     PRODUCT.Name = 'Pencil'

UNLOCK    PRODUCT

{no need to check if update was successful}
```

## Marking Transaction Boundaries Example

```
BEGIN TRANSACTION;

SELECT      PRODUCT.Name, PRODUCT.Quantity
FROM        PRODUCT
WHERE       PRODUCT.Name = 'Pencil'

Old Quantity = PRODUCT.Quantity

Set NewQuantity = PRODUCT.Quantity – 5

(process part of transaction – take exception action if NewQuantity < 0, etc.)

UPDATE      PRODUCT
SET         PRODUCT.Quantity = NewQuantity
WHERE       PRODUCT.Name = 'Pencil'

(continue processing transaction) . . .


IF transaction has completed normally      THEN
      COMMIT TRANSACTION
ELSE
      ROLLBACK TRANSACTION
END IF

Continue processing other actions not part of this transaction . . .
```





## Consistent Transactions

- Consistent transactions are often referred to by the acronym *ACID*
  - Atomic
  - Consistent
  - Isolated
  - Durable

## ACID: Atomic

- An atomic transaction is one in which all of the database actions occur or none of them do.
- A transaction consists of a series of steps. Each step must be successful for the transaction to be saved
- This ensures that the transaction completes everything it intended to do before saving the changes





## ACID: Durable

- A durable transaction is one in which all committed changes are permanent

## ACID: Consistent

- No other transactions are permitted on the records until the current transaction finishes
- This ensures that the transaction integrity has statement level consistency among all records





## ACID: Isolation

- Within multiuser environments, different transactions may be operating on the same data
- As such, the sequencing of uncommitted updates, rollbacks, and commits continuously change the data content
- The 1992 ANSI SQL standard defines four isolation levels that specify which of the concurrency control problems are allowed to occur

## 1992 ANSI SQL Isolation levels

| | | Isolation Level | | | |
|---|---|---|---|---|---|
| | | Read Uncommitted | Read Committed | Repeatable Read | Serializable |
| Problem Type | Dirty Read | Possible | Not possible | Not possible | Not possible |
| | Nonrepeatable Read | Possible | Possible | Not possible | Not possible |
| | Phantom Read | Possible | Possible | Possible | Not possible |





## Cursors

- A cursor is a pointer into a set of rows that are the result set from an SQL SELECT statement
- Cursors are usually defined using SELECT statements

```
DECLARE CURSOR TransCursor AS
        SELECT *
        FROM   SALE_TRANSACTION
        WHERE  PurchasePrice > '10000';
```

## Cursor Types

- Forward only cursor
- Scrollable cursors
  1. Static cursor
  2. Keyset cursor
  3. Dynamic cursor



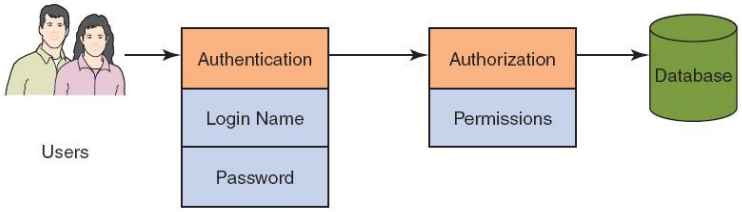


## Cursor Type Descriptions

| CursorType | Description | Comments |
|---|---|---|
| Forward only | Application can only move forward through the recordset. | Changes made by other cursors in this transaction or in other transactions will be visible only if they occur on rows ahead of the cursor. |
| Static | Application sees the data as they were at the time the cursor was opened. | Changes made by this cursor are visible. Changes from other sources are not visible. Backward and forward scrolling are allowed. |
| Keyset | When the cursor is opened, a primary key value is saved for each row in the recordset. When the application accesses a row, the key is used to fetch the current values for the row. | Updates from any source are visible. Inserts from sources outside this cursor are not visible (there is no key for them in the keyset). Inserts from this cursor appear at the bottom of the recordset. Deletions from any source are visible. Changes in row order are not visible. If the isolation level is dirty read, then committed updates and deletions are visible; otherwise, only committed updates and deletions are visible. |
| Dynamic | Changes of any type and from any source are visible. | All inserts, updates, deletions, and changes in recordset order are visible. If the isolation level is dirty read, then uncommitted changes are visible. Otherwise, only committed changes are visible. |

## Database Security

- Database Security strives to ensure that
  - Only authenticated users
  - Perform authorized activities








## Processing Rights and Responsibilities

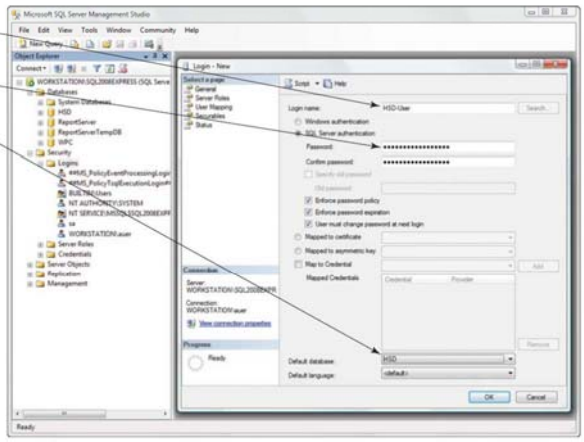
- Processing rights define who is permitted to do what and when.
- The individuals performing these activities have full responsibility for the implications of their actions.
- Individuals are identified by a username and a password.

## User Accounts in SQL Server 2005

The user's DMBS login name

The user's DBMS password

The HSD database







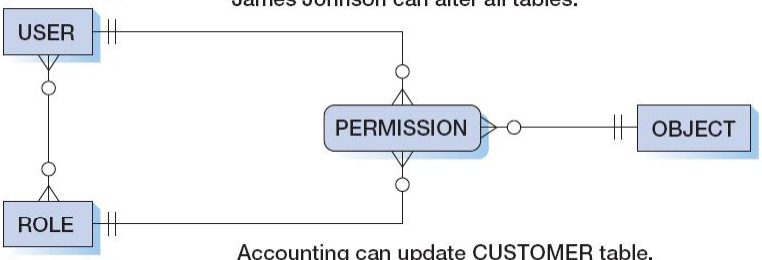
## Granting Permissions

- Database users are known as an individual and as a member of one or more roles
- Granting access and processing rights/privileges may be granted to an individual and/or a role
- Users possess the compilation of rights granted to the individual and all the roles for which they are members

## A Model of DBMS Security

Eleanore Wu
James Johnson
Richard Ent

Eleanore Wu can execute MonthEnd stored procedure
James Johnson can alter all tables.




Accounting can update CUSTOMER table.

Accounting
Tellers
Shop Managers
Unknown Public





## Processing Rights at Heather Sweeney Designs

DATABASE RIGHTS GRANTED

| Table | Administrative Assistants | Management | System Administrator |
|---|---|---|---|
| SEMINAR | Read, Insert, Change | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| CUSTOMER | Read, Insert, Change | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| SEMINAR_CUSTOMER | Read, Insert, Change, Delete | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| CONTACT | Read, Insert, Change | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| INVOICE | Read, Insert, Change | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| LINE_ITEM | Read, Insert, Change, Delete | Read, Insert, Change, Delete | Grant Rights, Modify Structure |
| PRODUCT | Read, Insert, Change | Read, Insert, Change, Delete | Grant Rights, Modify Structure |

# Database Security Guidelines

- Run the DBMS behind a firewall
- Apply the latest operating system and DBMS service packs and patches
- Limit DBMS functionality to needed features
- Protect the computer that runs the DBMS
- Manage accounts and passwords





# Database Backup and Recovery

- Common causes of database failures
  1. Hardware failures
  2. Programming bugs
  3. Human errors/mistakes
  4. Malicious actions
- Since these issues are impossible to completely avoid, recovery procedures are essential.

## Recovery via Reprocessing

- In reprocessing, all activities since the backup was performed are redone
- This is a brunt-force technique
- This procedure is costly in the effort involved in re-entering the data
- This procedure is risky in that human error is likely and in that paper record-keeping may not be accurate





## Recovery via Rollback and Rollforward

- Most database management systems provide a mechanism to record activities into a log file
  - To undo a transaction the log must contain a copy of every database record before it was changed
    - ✓ Such records are called before-images
    - ✓ A transaction is undone by applying before-images of all its changes to the database
  - To redo a transaction the log must contain a copy of every database record (or page) after it was changed
    - ✓ These records are called after-images
    - ✓ A transaction is redone by applying after-images of all its changes to the database
- The log file is then used for recovery via *rollback* or *rollforward*

## Example Transaction Log

| Relative Record Number | Transaction ID | Reverse Pointer | Forward Pointer | Time | Type of Operation | Object | Before-Image | After-Image |
|---|---|---|---|---|---|---|---|---|
| 1 | OT1 | 0 | 2 | 11:42 | START | | | |
| 2 | OT1 | 1 | 4 | 11:43 | MODIFY | CUST 100 | (old value) | (new value) |
| 3 | OT2 | 0 | 8 | 11:46 | START | | | |
| 4 | OT1 | 2 | 5 | 11:47 | MODIFY | SP AA | (old value) | (new value) |
| 5 | OT1 | 4 | 7 | 11:47 | INSERT | ORDER 11 | | (value) |
| 6 | CT1 | 0 | 9 | 11:48 | START | | | |
| 7 | OT1 | 5 | 0 | 11:49 | COMMIT | | | |
| 8 | OT2 | 3 | 0 | 11:50 | COMMIT | | | |
| 9 | CT1 | 6 | 10 | 11:51 | MODIFY | SP BB | (old value) | (new value) |
| 10 | CT1 | 9 | 0 | 11:51 | COMMIT | | | |

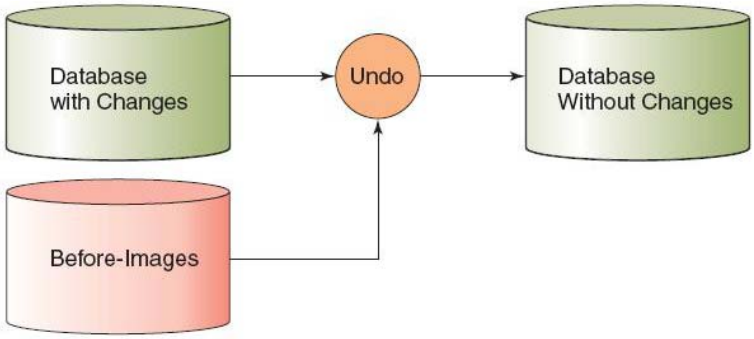




## Rollback

- Log files save activities in sequence order
- It is possible to undo activities in reverse order that they were originally executed
- This is performed to correct/undo erroneous or malicious transaction(s) after a database is recovered from a full backup

## Rollback Example



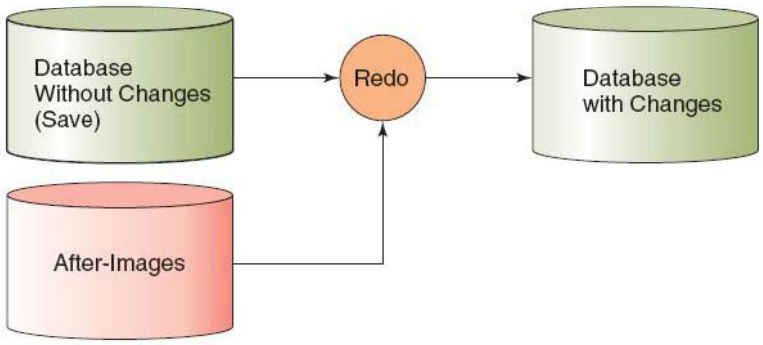





## Rollforward

- Activities recorded in the log files may be replayed
- In doing so, all activities are re-applied to the database
- This procedure is used to resynchronize restored database data by adding transactions to the last full backup

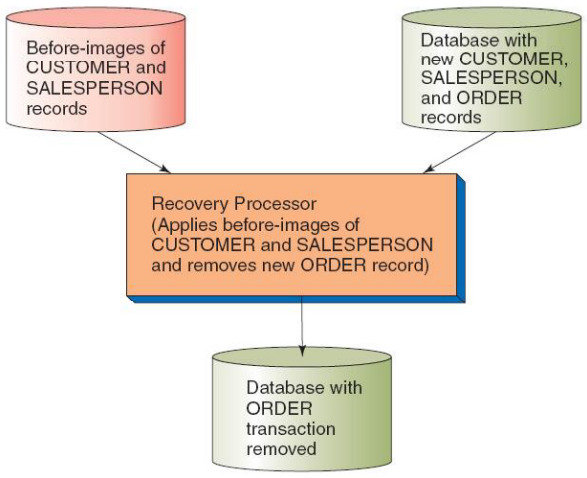
## Rollforward Example








## Recovery Example



## Additional DBA Responsibilities

- The DBA needs to ensure that a system exists to gather and record user reported errors and other problems
  - A means needs to be devised to prioritize those errors and problems and to ensure that they are corrected accordingly
- The DBA needs to create and manage a process for controlling the database configuration
  - Procedures for recording change requests
  - Conducting user and developer reviews of such requests
  - Creating projects and tasks
- The DBA is responsible for ensuring that appropriate documentation is maintained
  - Database structure
  - Concurrency control
  - Security
  - Backup and recovery
  - Applications used

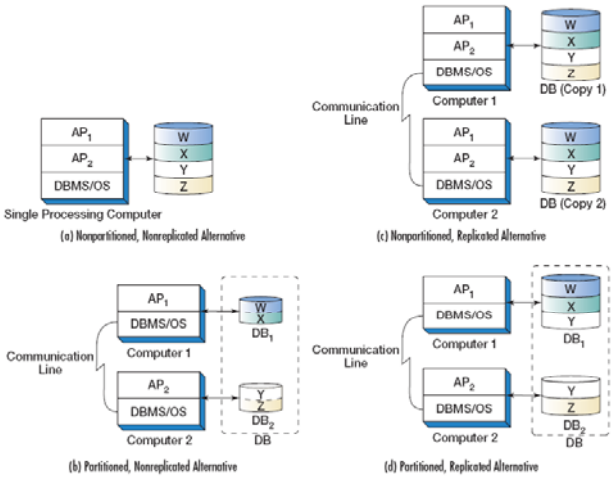




## Distributed Database Processing

- A distributed database is a database that is distributed on more than one compute.
- A database is can be distributed by:
  1. Partition
  2. Replication
  3. Both partition and replication
- This is fairly straight forward for read-only replicas, but it can be very difficult for other installations

## Type of Distributed Databases








## Object-Relational Database Management

- Object-oriented programming (OOP) is based on objects, and OOP is now used as the basis of many computer programming languages:
  - Java
  - VisualBasic.Net
  - C++
  - C#

## Objects

- Object classes have
  1. Identifiers & Properties
     - ✓ These are data items associated with the object
  2. Methods
     - ✓ These are programs that allow the object to perform tasks
- The only difference between entity classes and object classes is the methods





## Object Persistence

- Object persistence means that values of the object properties are storable and retrievable
- Object persistence can be achieved by various techniques
  - A main technique is database technology
  - Relational databases can be used, but require substantial programming

## OODBMS

- Object-Oriented DBMSs (OODBMSs) have been developed
  - Never achieved commercial success
    - ✓ It would be too expensive to transfer existing data from relational and other legacy databases
    - ✓ Therefore, the OODBMSs were *not* cost justifiable





## Object-Relational DBMSs

- Some relational DBMS vendors have added object-oriented features to their products:
  - Oracle
- These products are known as object-relational DBMSs and support object-relational databases