IT344- Database Management Systems

Study Guide for Final Examination
May 2015

College of Computing and Informatics
Saudi Electronic University

Chapter 17

Disk Storage, Basic File Structure and Hashing

Disk Storage Devices

- Preferred secondary storage device for high storage capacity and low cost.
- Data stored as magnetized areas on magnetic disk surfaces.
- A **disk pack** contains several magnetic disks connected to a rotating spindle.
- Disks are divided into concentric circular **tracks** on each disk **surface**.
  - Track capacities vary typically from 4 to 50 Kbytes or more

Disk Storage Devices (cont.)

- A track is divided into smaller **blocks** or **sectors**
  - because it usually contains a large amount of information
- The division of a track into **sectors** is hard-coded on the disk surface and cannot be changed.
  - One type of sector organization calls a portion of a track that subtends a fixed angle at the center as a sector.
- A track is divided into **blocks**.
  - The block size B is fixed for each system.
    - Typical block sizes range from B=512 bytes to B=4096 bytes.
  - Whole blocks are transferred between disk and main memory for processing.

Records

- Fixed and variable length records
- Records contain fields which have values of a particular type
  - E.g., amount, date, time, age
- Fields themselves may be fixed length or variable length
- Variable length fields can be mixed into one record:
  - Separator characters or length fields are needed so that the record can be "parsed."

Operation on Files

- Typical file operations include:
  - **OPEN**: Readies the file for access, and associates a pointer that will refer to a *current* file record at each point in time.
  - **FIND**: Searches for the first file record that satisfies a certain condition, and makes it the current file record.
  - **FINDNEXT**: Searches for the next file record (from the current record) that satisfies a certain condition, and makes it the current file record.
  - **READ**: Reads the current file record into a program variable.
  - **INSERT**: Inserts a new record into the file & makes it the current file record.
  - **DELETE**: Removes the current file record from the file, usually by marking the record to indicate that it is no longer valid.
  - **MODIFY**: Changes the values of some fields of the current file record.
  - **CLOSE**: Terminates access to the file.
  - **REORGANIZE**: Reorganizes the file records.
    - For example, the records marked deleted are physically removed from the file or a new organization of the file records is created.
  - **READ_ORDERED**: Read the file blocks in order of a specific field of the file.

Ordered Files

- Also called a **sequential** file.
- File records are kept sorted by the values of an *ordering field*.
- Insertion is expensive: records must be inserted in the correct order.
  - It is common to keep a separate unordered *overflow* (or *transaction*) file for new records to improve insertion efficiency; this is periodically merged with the main ordered file.
- A **binary search** can be used to search for a record on its *ordering field* value.
  - This requires reading and searching $\log_2$ of the file blocks on the average, an improvement over linear search.
- Reading the records in order of the ordering field is quite efficient.

RAID Technology (cont.)

- Different raid organizations were defined based on different combinations of the two factors of granularity of data interleaving (striping) and pattern used to compute redundant information.
  - Raid level 0 has no redundant data and hence has the best write performance at the risk of data loss
  - Raid level 1 uses mirrored disks.
  - Raid level 2 uses memory-style redundancy by using Hamming codes, which contain parity bits for distinct overlapping subsets of components. Level 2 includes both error detection and correction.
  - Raid level 3 uses a single parity disk relying on the disk controller to figure out which disk has failed.
  - Raid Levels 4 and 5 use block-level data striping, with level 5 distributing data and parity information across all disks.
  - Raid level 6 applies the so-called P + Q redundancy scheme using Reed-Soloman codes to protect against up to two disk failures by using just two redundant disks.

Chapter 18

Indexing Structures for Files

Indexes as Access Paths

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries <**field value, pointer to record>**, which is ordered by field value
- The index is called an access path on the field.

Indexes as Access Paths (cont.)

- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- Indexes can also be characterized as dense or sparse
  - A **dense index** has an index entry for every search key value (and hence every record) in the data file.
  - A **sparse (or nondense) index**, on the other hand, has index entries for only some of the search values

Types of Single-Level Indexes

- Primary Index
  - Defined on an ordered data file
  - The data file is ordered on a **key field**
  - Includes one index entry *for each block* in the data file; the index entry has the key field value for the *first record* in the block, which is called the *block anchor*
  - A similar scheme can use the *last record* in a block.
  - A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.

Types of Single-Level Indexes

- Clustering Index
  - Defined on an ordered data file
  - The data file is ordered on a *non-key field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
  - Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
  - It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.

Types of Single-Level Indexes

- Secondary Index
  - A secondary index provides a secondary means of accessing a file for which some primary access already exists.
  - The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some **non-ordering field** of the data file that is an indexing field.
    - The second field is either a **block** pointer or a record pointer.
    - There can be *many* secondary indexes (and hence, indexing fields) for the same file.
  - Includes one entry *for each record* in the data file; hence, it is a *dense index*

Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- Most multi-level indexes use B-tree or B+-tree data structures because of the insertion and deletion problem
  - This leaves space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full

Difference between B-tree and B+-tree

- In a B-tree, pointers to data records exist at all levels of the tree
- In a B+-tree, all pointers to data records exists at the leaf-level nodes
- A B+-tree can have less levels (or higher capacity of search values) than the corresponding B-tree

Chapter 19

Algorithms for Query processing and Optimization

0. Introduction to Query Processing (1)

- **Query optimization**:
  - The process of choosing a suitable execution strategy for processing a query.
- Two internal representations of a query:
  - **Query Tree**
  - **Query Graph**

1. Translating SQL Queries into Relational Algebra (1)

- **Query block**:
  - The basic unit that can be translated into the algebraic operators and optimized.
- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.
- **Nested queries** within a query are identified as separate query blocks.
- Aggregate operators in SQL must be included in the extended algebra.

Translating SQL Queries into Relational Algebra (2)

```
SELECT   LNAME, FNAME
FROM     EMPLOYEE
WHERE    SALARY > ( SELECT   MAX (SALARY)
                    FROM     EMPLOYEE
                    WHERE    DNO = 5);
```

SELECT	LNAME, FNAME
FROM	EMPLOYEE
WHERE	SALARY > C

$\pi_{\text{LNAME, FNAME}} (\sigma_{\text{SALARY>C}}(\text{EMPLOYEE}))$

SELECT	MAX (SALARY)
FROM	EMPLOYEE
WHERE	DNO = 5

$\mathcal{F}_{\text{MAX SALARY}} (\sigma_{\text{DNO=5}} (\text{EMPLOYEE}))$

Using Heuristics in Query Optimization (2)

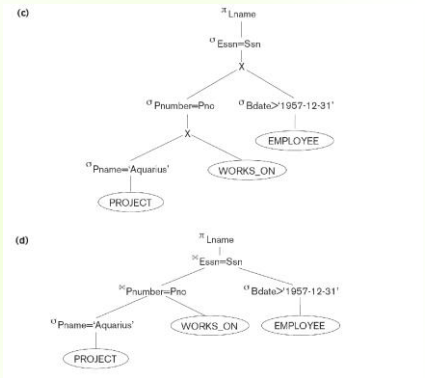
- **Query tree**:
  - A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as **leaf nodes** of the **tree**, and represents the relational algebra operations as internal nodes.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.
- **Query graph**:
  - A graph data structure that corresponds to a relational calculus expression. It does *not* indicate an order on which operations to perform first. There is only a *single* graph corresponding to each query.

Using Heuristics in Query Optimization (5)

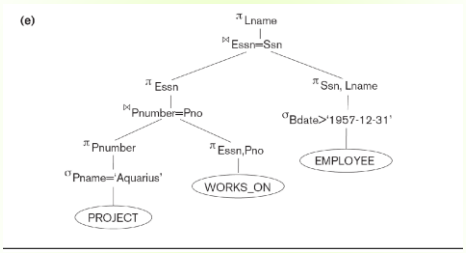
- Heuristic Optimization of Query Trees:
  - The same query could correspond to many different relational algebra expressions — and hence many different query trees.
  - The task of heuristic optimization of query trees is to find a **final query tree** that is efficient to execute.
- Example:

```
Q: SELECT   LNAME
   FROM     EMPLOYEE, WORKS_ON, PROJECT
   WHERE    PNAME = 'AQUARIUS' AND
            PNMUBER=PNO AND ESSN=SSN
            AND BDATE > '1957-12-31';
```

Using Heuristics in Query Optimization (7)



Using Heuristics in Query Optimization (8)



Using Heuristics in Query Optimization (9)

- General Transformation Rules for Relational Algebra Operations:

1. Cascade of σ: A conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations:
   - $\sigma_{c1 \text{ AND } c2 \text{ AND } ... \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$
2. Commutativity of σ: The σ operation is commutative:
   - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
3. Cascade of π: In a cascade (sequence) of π operations, all but the last one can be ignored:
   - $\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) = \pi_{List1}(R)$
4. Commuting σ with π: If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
   - $\pi_{A1, A2, ..., An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, ..., An}(R))$

Using Heuristics in Query Optimization (10)

- General Transformation Rules for Relational Algebra Operations (contd.):

5. Commutativity of (and x): The operation is commutative as is the x operation:
   - $R \bowtie_C S = S \bowtie_C R$; $R \times S = S \times R$
6. Commuting σ with (or x): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:
   - $\sigma_c(R \bowtie S) = (\sigma_c(R)) \bowtie S$
- Alternatively, if the selection condition c can be written as (c1 and c2), where condition c1 involves only the attributes of R and condition c2 involves only the attributes of S, the operations commute as follows:
   - $\sigma_c(R \bowtie S) = (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$

Using Heuristics in Query Optimization (16)

- Query Execution Plans
  - An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.
  - **Materialized evaluation**: the result of an operation is stored as a temporary relation.
  - **Pipelined evaluation**: as the result of an operator is produced, it is forwarded to the next operator in sequence.

Chapter 20

Physical Database Design and Tuning

1. Physical Database Design in Relational Databases (1)

- Factors that Influence Physical Database Design:
  A. **Analyzing the database queries and transactions**
  - For each query, the following information is needed.
    1. The *files* that will be accessed by the query;
    2. The *attributes* on which any *selection* conditions for the query are specified;
    3. The *attributes* on which any *join* conditions or conditions to link multiple tables or objects for the query are specified;
    4. The *attributes* whose values will be *retrieved* by the query.
  - Note: the attributes listed in items 2 and 3 above are candidates for definition of access structures.

Physical Database Design in Relational Databases (2)

- Factors that Influence Physical Database Design (cont.):
  A. **Analyzing the database queries and transactions** (cont.)
  - For each **update** transaction or operation, the following information is needed.
    1. The files that will be updated;
    2. The type of operation on each file (insert, update or delete);
    3. The attributes on which selection conditions for a delete or update operation are specified;
    4. The attributes whose values will be changed by an update operation.
  - Note: the attributes listed in items 3 above are candidates for definition of access structures. However, the attributes listed in item 4 are candidates for avoiding an access structure.

Physical Database Design in Relational Databases (3)

- Factors that Influence Physical Database Design (cont.):
  B. **Analyzing the expected frequency of invocation of queries and transactions**
  - The expected frequency information, along with the attribute information collected on each query and transaction, is used to compile a cumulative list of expected frequency of use for all the queries and transactions.
  - It is expressed as the expected frequency of using each attribute in each file as a selection attribute or join attribute, over all the queries and transactions.
  - **80-20 rule**
    - 20% of the data is accessed 80% of the time

Physical Database Design in Relational Databases (4)

- Factors that Influence Physical Database Design (cont.)
  C. **Analyzing the time constraints of queries and transactions**
  - Performance constraints place further priorities on the attributes that are candidates for access paths.
  - The selection attributes used by queries and transactions with time constraints become higher-priority candidates for primary access structure.

Physical Database Design in Relational Databases (4)

- Factors that Influence Physical Database Design (cont.)
  D. **Analyzing the expected frequencies of update operations**
  - A minimum number of access paths should be specified for a file that is updated frequently.

Physical Database Design in Relational Databases (4)

- Factors that Influence Physical Database Design (cont.)
  E. **Analyzing the uniqueness constraints on attributes**
  - Access paths should be specified on all candidate key attributes — or set of attributes — that are either the primary key or constrained to be unique.

Physical Database Design in Relational Databases (6)

- Physical Database Design Decisions (cont.)
- Denormalization as a design decision for speeding up queries
  - The goal of normalization is to separate the logically related attributes into tables to minimize redundancy and thereby avoid the update anomalies that cause an extra processing overheard to maintain consistency of the database.
  - The goal of denormalization is to improve the performance of frequently occurring queries and transactions. (Typically the designer adds to a table attributes that are needed for answering queries or producing reports so that a join with another table is avoided.)
  - Trade off between update and query performance

2. An Overview of Database Tuning in Relational Systems (1)

- **Tuning**:
  - The process of continuing to revise/adjust the physical database design by monitoring resource utilization as well as internal DBMS processing to reveal bottlenecks such as contention for the same data or devices.
- Goal:
  - To make application run faster
  - To lower the response time of queries/transactions
  - To improve the overall throughput of transactions

An Overview of Database Tuning in Relational Systems (3)

- Problems to be considered in tuning:
  - How to avoid excessive lock contention?
  - How to minimize overhead of logging and unnecessary dumping of data?
  - How to optimize buffer size and scheduling of processes?
  - How to allocate resources such as disks, RAM and processes for most efficient utilization?

Chapter 21

Introduction to Transaction Processing Concepts and Theory

1 Introduction to Transaction Processing (1)

- **Single-User System**:
  - At most one user at a time can use the system.
- **Multiuser System**:
  - Many users can access the system concurrently.
- **Concurrency**
  - **Interleaved processing**:
    - Concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing**:
    - Processes are concurrently executed in multiple CPUs.

Introduction to Transaction Processing (2)

- A **Transaction**:
  - Logical unit of database processing that includes one or more access operations (read -retrieval, write - insert or update, delete).
- A transaction (set of operations) may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program.
- **Transaction boundaries**:
  - Begin and End transaction.
- An **application program** may contain several transactions separated by the Begin and End transaction boundaries.

Introduction to Transaction Processing (3)

SIMPLE MODEL OF A DATABASE (for purposes of discussing transactions):

- **A database** is a collection of named data items
- **Granularity** of data - a field, a record , or a whole disk block (Concepts are independent of granularity)
- Basic operations are **read** and **write**
  - **read_item(X)**: Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
  - **write_item(X)**: Writes the value of program variable X into the database item named X.

Introduction to Transaction Processing (6)

Why Concurrency Control is needed:

- **The Lost Update Problem**
  - This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
- **The Temporary Update (or Dirty Read) Problem**
  - This occurs when one transaction updates a database item and then the transaction fails for some reason (see Section 21.1.4).
  - The updated item is accessed by another transaction before it is changed back to its original value.
- **The Incorrect Summary Problem**
  - If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Introduction to Transaction Processing (12)

Why **recovery** is needed:
(What causes a Transaction to fail)

1. A computer failure (system crash):
2. 2. A transaction or system error:
3. 3. Local errors or exception conditions detected by the transaction:
4. 4. Concurrency control enforcement:
5. 5. Disk failure:
6. 6. Physical problems and catastrophes:

2 Transaction and System Concepts (1)

- A **transaction** is an atomic unit of work that is either completed in its entirety or not done at all.
  - For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- **Transaction states**:
  - Active state
  - Partially committed state
  - Committed state
  - Failed state
  - Terminated State

Transaction and System Concepts (3)

- Recovery manager keeps track of the following operations (cont):
  - **commit_transaction**: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
  - **rollback** (or **abort**): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Transaction and System Concepts (4)

- Recovery techniques use the following operators:
  - **undo**: Similar to rollback except that it applies to a single operation rather than to a whole transaction.
  - **redo**: This specifies that certain *transaction operations* must be *redone* to ensure that all the operations of a committed transaction have been applied successfully to the database.

Transaction and System Concepts (10)

Commit Point of a Transaction:

- **Definition a Commit Point:**
  - A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in the log.
  - Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
  - The transaction then writes an entry [commit,T] into the log.
- **Roll Back of transactions:**
  - Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.

3 Desirable Properties of Transactions (1)

ACID properties:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another.
- **Isolation**: A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary (see Chapter 21).
- **Durability or permanency**: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

Chapter 22

Concurrency Control Techniques

Database Concurrency Control

- 1 Purpose of Concurrency Control
  - To enforce Isolation (through mutual exclusion) among conflicting transactions.
  - To preserve database consistency through consistency preserving execution of transactions.
  - To resolve read-write and write-write conflicts.
- Example:
  - In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Database Concurrency Control

Two-Phase Locking Techniques

- Locking is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction.
- Example:
  - Lock (X). Data item X is locked in behalf of the requesting transaction.
- Unlocking is an operation which removes these permissions from the data item.
- Example:
  - Unlock (X): Data item X is made available to all other transactions.
- Lock and Unlock are Atomic operations.

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Two locks modes:
  - (a) shared (read) (b) exclusive (write).
- Shared mode: shared lock (X)
  - More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.
- Exclusive mode: Write lock (X)
  - Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.
- Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Database Concurrency Control

Two-Phase Locking Techniques: Essential components

- Lock Manager:
  - Managing locks on data items.
- Lock table:
  - Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

- Two Phases:
  - (a) Locking (Growing)
  - (b) Unlocking (Shrinking).
- **Locking (Growing) Phase:**
  - A transaction applies locks (read or write) on desired data items one at a time.
- **Unlocking (Shrinking) Phase:**
  - A transaction unlocks its locked data items one at a time.
- **Requirement:**
  - For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

Database Concurrency Control

Two-Phase Locking Techniques: The algorithm

- Two-phase policy generates two locking algorithms
  - (a) **Basic**
  - (b) **Conservative**
- **Conservative**:
  - Prevents deadlock by locking all desired data items before transaction begins execution.
- **Basic**:
  - Transaction locks data items incrementally. This may cause deadlock which is dealt with.
- **Strict**:
  - A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock**

T'1	T'2	
read_lock (Y);		T1 and T2 did follow two-phase policy but they are deadlock
read_item (Y);		
	read_lock (X);	
	read_item (Y);	
write_lock (X);		
(waits for X)		
	write_lock (Y);	
	(waits for Y)	

- Deadlock (T'1 and T'2)

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock prevention**
  - A transaction locks all data items it refers to before it begins execution.
  - This way of locking prevents deadlock since a transaction never waits for a data item.
  - The conservative two-phase locking uses this approach.

Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock detection and resolution**
  - In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.
  - A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle. One of the transaction o

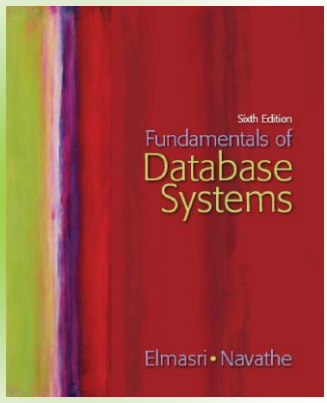
Database Concurrency Control

Dealing with Deadlock and Starvation

- **Deadlock avoidance**
  - There are many variations of two-phase locking algorithm.
  - Some avoid deadlock by not letting the cycle to complete.
  - That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
  - Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.

Chapter 23

Database Recovery Techniques



Database Recovery

1 Purpose of Database Recovery
- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

- Example:
  - If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Database Recovery

2 Types of Failure
- The database may become unavailable for use due to
  - **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
  - **System failure**: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
  - **Media failure**: Disk head crash, power disruption, etc.

Database Recovery

3 Transaction Log
- For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database Recovery

4 Data Update
- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

Database Recovery

6 Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**
- To maintain atomicity, a transaction's operations are redone or undone.
  - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
  - **Redo**: Restore all AFIMs on to disk.
- Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

Database Recovery

Write-Ahead Logging
- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that
  - **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  - **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Database Recovery

7 Checkpointing

- Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:
  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.
- During recovery redo or undo is required to transactions appearing after [checkpoint] record.

Database Recovery

Steal/No-Steal and Force/No-Force

- Possible ways for flushing database cache to database disk:
  1. Steal: Cache can be flushed before transaction commits.
  2. No-Steal: Cache cannot be flushed before transaction commit.
  3. Force: Cache is immediately flushed (forced) to disk.
  4. No-Force: Cache is deferred until transaction commits
- These give rise to four different ways for handling recovery:
  - Steal/No-Force (Undo/Redo)
  - Steal/Force (Undo/No-redo)
  - No-Steal/No-Force (Redo/No-undo)
  - No-Steal/Force (No-undo/No-redo)

Database Recovery

8 Recovery Scheme

- Deferred Update (No Undo/Redo)
  - The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
  - After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Database Recovery

Deferred Update with concurrent users

- This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.

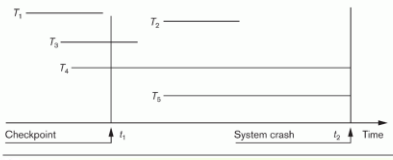


Figure 23.2
An example of a recovery timeline to illustrate the effect of checkpointing.

Database Recovery

Deferred Update with concurrent users

- Two tables are required for implementing this protocol:
  - **Active table**: All active transactions are entered in this table.
  - **Commit table**: Transactions to be committed are entered in this table.
- During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

Database Recovery

Recovery Techniques Based on Immediate Update

- **Undo/No-redo Algorithm**
  - In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
  - For this reason the recovery manager **undoes** all transactions during recovery.
  - No transaction is **redone**.
  - It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Database Recovery

Recovery Techniques Based on Immediate Update

- **Undo/Redo Algorithm** (**Single-user** environment)
  - Recovery schemes of this category apply **undo** and also **redo** for recovery.
  - In a single-user environment no concurrency control is required but a log is maintained under WAL.
  - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
  - The recovery manager performs:
    - **Undo** of a transaction if it is in the **active** table.
    - **Redo** of a transaction if it is in the **commit** table.

Database Recovery

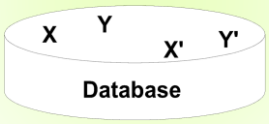
Recovery Techniques Based on Immediate Update

- **Undo/Redo Algorithm** (**Concurrent** execution)
- Recovery schemes of this category applies **undo** and also **redo** to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager checkpointing is used.
- The recovery performs:
  - **Undo** of a transaction if it is in the **active** table.
  - **Redo** of a transaction if it is in the **commit** table.

Database Recovery

Shadow Paging

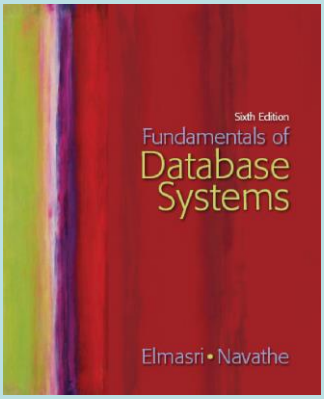
- The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items
X' and Y': Current copies of data items

Chapter 24

Database Security



Introduction to Database Security Issues (2)

- Threats to databases
  - Loss of **integrity**
  - Loss of **availability**
  - Loss of **confidentiality**
- To protect databases against these types of threats four kinds of countermeasures can be implemented:
  - **Access control**
  - **Inference control**
  - **Flow control**
  - **Encryption**

Introduction to Database Security Issues (3)

- A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.
- Two types of database security mechanisms:
  - **Discretionary** security mechanisms
  - **Mandatory** security mechanisms

Introduction to Database Security Issues (4)

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole
  - This function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.

Discretionary Access Control Based on Granting and Revoking Privileges

- The typical method of enforcing **discretionary access control** in a database system is based on the **granting** and **revoking privileges**.

2.1Types of Discretionary Privileges

- The **account level**:
  - At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- The **relation level** (or **table level**):
  - At this level, the DBA can control the privilege to access each individual relation or view in the database.

2.1Types of Discretionary Privileges(2)

- The privileges at the **account level** apply to the capabilities provided to the account itself and can include
  - the **CREATE SCHEMA** or **CREATE TABLE** privilege, to create a schema or base relation;
  - the **CREATE VIEW** privilege;
  - the **ALTER** privilege, to apply schema changes such adding or removing attributes from relations;
  - the **DROP** privilege, to delete relations or views;
  - the **MODIFY** privilege, to insert, delete, or update tuples;
  - and the **SELECT** privilege, to retrieve information from the database by using a **SELECT** query.

2.1Types of Discretionary Privileges(3)

- The second level of privileges applies to the **relation level**
  - This includes **base relations** and virtual (**view**) relations.
- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model where
  - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
  - The **columns** represent **objects** (relations, records, columns, views, operations).
  - Each position **M(i,j)** in the matrix represents the types of privileges (read, write, update) that **subject i** holds on **object j**.

2.1Types of Discretionary Privileges(4)

- To control the granting and revoking of relation privileges, each relation R in a database is assigned and **owner account**, which is typically the account that was used when the relation was created in the first place.
  - The owner of a relation is given all privileges on that relation.
  - In SQL2, the DBA can assign and owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the **CREATE SCHEMA** command.
  - The owner account holder can **pass privileges** on any of the owned relation to other users by **granting** privileges to their accounts.

2.1Types of Discretionary Privileges(5)

- In SQL the following types of privileges can be granted on each individual relation R:
  - **SELECT** (retrieval or read) privilege on R:
    - Gives the account retrieval privilege.
    - In SQL this gives the account the privilege to use the **SELECT** statement to retrieve tuples from R.
  - **MODIFY** privileges on R:
    - This gives the account the capability to modify tuples of R.
    - In SQL this privilege is further divided into **UPDATE**, **DELETE**, and **INSERT** privileges to apply the corresponding SQL command to R.
    - In addition, both the **INSERT** and **UPDATE** privileges can specify that only certain attributes can be updated by the account.

2.1Types of Discretionary Privileges(6)

- In SQL the following types of privileges can be granted on each individual relation R (contd.):
  - **REFERENCES** privilege on R:
    - This gives the account the capability to **reference** relation R when specifying integrity constraints.
    - The privilege can also be **restricted** to specific attributes of R.
- Notice that to create a **view**, the account must have **SELECT** privilege on all relations involved in the view definition.

2.2 Specifying Privileges Using Views

- The mechanism of **views** is an important discretionary authorization mechanism in its own right. For example,
  - If the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B.
  - The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access.

2.3 Revoking Privileges

- In some cases it is desirable to grant a privilege to a user temporarily. For example,
  - The owner of a relation may want to grant the **SELECT** privilege to a user for a specific task and then revoke that privilege once the task is completed.
  - Hence, a mechanism for **revoking** privileges is needed. In SQL, a **REVOKE** command is included for the purpose of **canceling privileges**.

2.4 Propagation of Privileges using the GRANT OPTION

- Whenever the owner A of a relation R grants a privilege on R to another account B, privilege can be given to B with or without the **GRANT OPTION**.
- If the **GRANT OPTION** is given, this means that B can also grant that privilege on R to other accounts.
  - Suppose that B is given the **GRANT OPTION** by A and that B then grants the privilege on R to a third account C, also with **GRANT OPTION**. In this way, privileges on R can **propagate** to other accounts without the knowledge of the owner of R.
  - If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege should automatically be revoked by the system.

2.5 An Example

- Suppose that the DBA creates four accounts
  - A1, A2, A3, A4
- and wants only A1 to be able to create base relations. Then the DBA must issue the following GRANT command in SQL

```
GRANT CREATETAB TO A1;
```

- In SQL2 the same effect can be accomplished by having the DBA issue a **CREATE SCHEMA** command as follows:

```
CREATE SCHAMA EXAMPLE AUTHORIZATION A1;
```

2.5 An Example (2)

- User account A1 can create tables under the schema called **EXAMPLE**.
- Suppose that A1 **creates** the two base relations **EMPLOYEE** and **DEPARTMENT**
  - A1 is then **owner** of these two relations and hence all the relation privileges on each of them.
- Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:

```
GRANT INSERT, DELETE ON
    EMPLOYEE, DEPARTMENT TO A2;
```


2.5 An Example (3)

Figure 24.1
Schemas for the two relations EMPLOYEE and DEPARTMENT.

2.5 An Example (4)

- Suppose that A1 wants to allow A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts.
- A1 can issue the command:

```
GRANT SELECT ON EMPLOYEE, DEPARTMENT
    TO A3 WITH GRANT OPTION;
```

- A3 can grant the **SELECT** privilege on the **EMPLOYEE** relation to A4 by issuing:

```
GRANT SELECT ON EMPLOYEE TO A4;
```

  - Notice that A4 can't propagate the SELECT privilege because GRANT OPTION was not given to A4

2.5 An Example (5)

- Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 can issue:

```
REVOKE SELECT ON EMPLOYEE FROM A3;
```

- The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.

2.5 An Example (6)

- Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege.
  - The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.
- A1 then create the view:

```
CREATE VIEW A3EMPLOYEE AS
  SELECT NAME, BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE DNO = 5;
```

- After the view is created, A1 can grant **SELECT** on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3
    WITH GRANT OPTION;
```


2.5 An Example (7)

- Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE;
- A1 can issue:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO
A4;
```

  - The **UPDATE** or **INSERT** privilege can specify particular attributes that may be updated or inserted in a relation.
  - Other privileges (**SELECT**, **DELETE**) are not attribute specific.

2.6 Specifying Limits on Propagation of Privileges

- Techniques to limit the propagation of privileges have been developed, although they have not yet been implemented in most DBMSs and are not a part of SQL.
  - Limiting **horizontal propagation** to an integer number i means that an account B given the GRANT OPTION can grant the privilege to at most i other accounts.
  - **Vertical propagation** is more complicated; it limits the depth of the granting of privileges.

3 Mandatory Access Control and Role-Based Access Control for Multilevel Security

- The discretionary access control techniques of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems.
- This is an all-or-nothing method:
  - A user either has or does not have a certain privilege.
- In many applications, and **additional security policy** is needed that classifies data and users based on security classes.
  - This approach as **mandatory access control**, would typically be **combined** with the discretionary access control mechanisms.

3.1 Comparing Discretionary Access Control and Mandatory Access Control

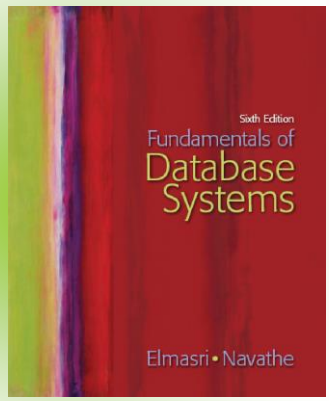
- **Discretionary Access Control (DAC)** policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains.
  - The main drawback of **DAC** models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs.

3.1 Comparing Discretionary Access Control and Mandatory Access Control(2)

- By contrast, mandatory policies ensure a high degree of protection in a way, they prevent any illegal flow of information.
- Mandatory policies have the drawback of being too rigid and they are only applicable in limited environments.
- In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

Chapter 25

Distributed Databases and Client-Server Architectures

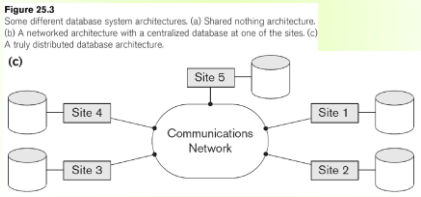


Distributed Database Concepts

- A transaction can be executed by multiple networked computers in a unified manner.
- A **distributed database (DDB)** processes Unit of execution (a transaction) in a distributed manner. A distributed database (DDB) can be defined as
  - A distributed database (DDB) is a collection of multiple logically related database distributed over a computer network, and a distributed database management system as a software system that manages a distributed database while making the distribution transparent to the user.

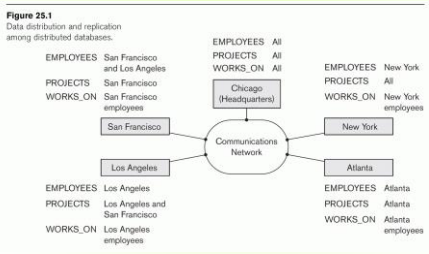
Distributed Database System

- Advantages
  - Management of distributed data with different **levels of transparency**:
    - This refers to the physical placement of data (files, relations, etc.) which is not known to the user (distribution transparency).



Distributed Database System

- Advantages (transparency, contd.)
  - The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally and stored with possible replication as shown below.



Distributed Database System

- Advantages (transparency, contd.)
  - **Distribution and Network transparency**:
    - Users do not have to worry about operational details of the network.
      - There is Location transparency, which refers to freedom of issuing command from any location without affecting its working.
      - Then there is Naming transparency, which allows access to any names object (files, relations, etc.) from any location.

Distributed Database System

- Advantages (transparency, contd.)
  - **Replication transparency**:
    - It allows to store copies of a data at multiple sites as shown in the above diagram.
    - This is done to minimize access time to the required data.
  - **Fragmentation transparency**:
    - Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation).

Distributed Database System

- Other Advantages
  - **Increased reliability and availability**:
    - Reliability refers to system live time, that is, system is running efficiently most of the time. Availability is the probability that the system is continuously available (usable or accessible) during a time interval.
    - A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.

Data Fragmentation, Replication and Allocation

- **Data Fragmentation**
  - Split a relation into logically related and correct parts. A relation can be fragmented in two ways:
    - **Horizontal Fragmentation**
    - **Vertical Fragmentation**

Data Fragmentation, Replication and Allocation

- **Horizontal fragmentation**
  - It is a horizontal subset of a relation which contain those of tuples which satisfy selection conditions.
  - Consider the Employee relation with selection condition (DNO = 5). All tuples satisfy this condition will create a subset which will be a horizontal fragment of Employee relation.
  - A selection condition may be composed of several conditions connected by AND or OR.
  - Derived horizontal fragmentation: It is the partitioning of a primary relation to other secondary relations which are related with Foreign keys.

Data Fragmentation, Replication and Allocation

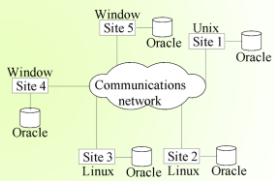
- **Vertical fragmentation**
  - It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation.
  - Consider the Employee relation. A vertical fragment of can be created by keeping the values of Name, Bdate, Sex, and Address.
  - Because there is no condition for creating a vertical fragment, each fragment must include the primary key attribute of the parent relation Employee. In this way all vertical fragments of a relation are connected.

Data Fragmentation, Replication and Allocation

- **Data Replication**
  - Database is replicated to all sites.
  - In full replication the entire database is replicated and in partial replication some selected part is replicated to some of the sites.
  - Data replication is achieved through a replication schema.
- **Data Distribution (Data Allocation)**
  - This is relevant only in the case of partial replication or partition.
  - The selected portion of the database is distributed to the database sites.

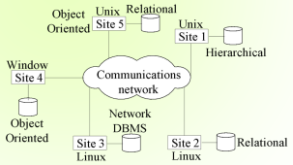
Types of Distributed Database Systems

- Homogeneous
  - All sites of the database system have identical setup, i.e., same database system software.
  - The underlying operating system may be different.
    - For example, all sites run Oracle or DB2, or Sybase or some other database system.
  - The underlying operating systems can be a mixture of Linux, Window, Unix, etc.



Types of Distributed Database Systems

- Heterogeneous
  - Federated: Each site may run different database system but the data access is managed through a single conceptual schema.
    - This implies that the degree of local autonomy is minimum. Each site must adhere to a centralized access policy. There may be a global schema.
  - Multidatabase: There is no one conceptual global schema. For data access a schema is constructed dynamically as needed by the application software.



Types of Distributed Database Systems

- Federated Database Management Systems Issues
  - Differences in data models:
    - Relational, Objected oriented, hierarchical, network, etc.
  - Differences in constraints:
    - Each site may have their own data accessing and processing constraints.
  - Differences in query language:
    - Some site may use SQL, some may use SQL-89, some may use SQL-92, and so on.

Query Processing in Distributed Databases

- Issues
  - Cost of transferring data (files and results) over the network.
    - This cost is usually high so some optimization is necessary.
    - Example relations: Employee at site 1 and Department at Site 2
      - Employee at site 1. 10,000 rows. Row size = 100 bytes. Table size = 10^6 bytes.

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Superssn	Dno

      - Department at Site 2. 100 rows. Row size = 35 bytes. Table size = 3,500 bytes.

Dname	Dnumber	Mgrssn	Mgrstartdate

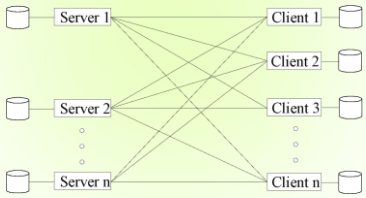
    - Q: For each employee, retrieve employee name and department name Where the employee works.
    - Q: $\Pi_{Fname,Lname,Dname}$ (Employee $\bowtie_{Dno = Dnumber}$ Department)

Query Processing in Distributed Databases

- Result
  - The result of this query will have 10,000 tuples, assuming that every employee is related to a department.
  - Suppose each result tuple is 40 bytes long. The query is submitted at site 3 and the result is sent to this site.
  - Problem: Employee and Department relations are not present at site 3.

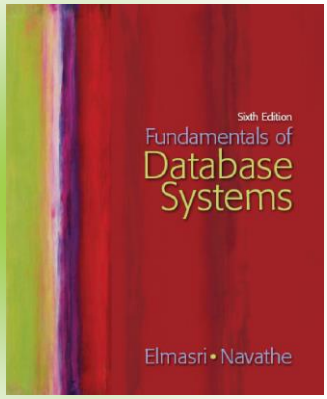
Client-Server Database Architecture

- It consists of clients running client software, a set of servers which provide all database functionalities and a reliable communication infrastructure.



Chapter 26

Enhanced Data Models for Advanced Applications



Active Database Concepts and Triggers

Generalized Model for Active Databases and Oracle **Triggers**

- **Triggers** are executed when a specified condition occurs during insert/delete/update
  - Triggers are action that fire automatically based on these conditions

Event-Condition-Action (ECA) Model

Generalized Model (cont.)

- Triggers follow an Event-condition-action (ECA) model
  - **Event**:
    - Database modification
      - E.g., insert, delete, update),
  - **Condition**:
    - Any true/false expression
      - Optional: If no condition is specified then condition is always true
  - **Action**:
    - Sequence of SQL statements that will be automatically executed

Row-Level versus Statement-level

Generalized Model (cont.)

- Triggers can be
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**
    - Default (when FOR EACH ROW is not specified)
- Row level triggers
  - Executed separately for each affected row
- Statement-level triggers
  - Execute once for the SQL statement,

Condition

Generalized Model (cont.)

- Any true/false condition to control whether a trigger is activated on not
  - Absence of condition means that the trigger will always execute for the even
  - Otherwise, condition is evaluated
    - before the event for BEFORE trigger
    - after the event for AFTER trigger

Action

Generalized Model (cont.)

- Action can be
  - One SQL statement
  - A sequence of SQL statements enclosed between a BEGIN and an END
- Action specifies the relevant modifications

Active Database Concepts and Triggers

Design and Implementation Issues for Active Databases

- An active database allows users to make the following changes to triggers (rules)
  - Activate
  - Deactivate
  - Drop

Active Database Concepts and Triggers

Design and Implementation Issues (cont.)

- Immediate consideration
  - Part of the same transaction and can be one of the following depending on the situation
    - Before
    - After
    - Instead of
- Deferred consideration
  - Condition is evaluated at the end of the transaction
- Detached consideration
  - Condition is evaluated in a separate transaction

Temporal Database Concepts

Time Representation, … (cont.)

- A **calendar** organizes time into different time units for convenience.
  - Accommodates various calendars
    - Gregorian (western)
    - Chinese
    - Islamic
    - Hindu
    - Jewish
    - Etc.

Temporal Database Concepts

Time Representation, … (cont.)

- Transaction time
  - The time when the information from a certain transaction becomes valid
- Bitemporal database
  - Databases dealing with two time dimensions

Temporal Database Concepts

Incorporating Time in Relational Databases Using Tuple Versioning

- Add to every tuple
  - Valid start time
  - Valid end time

Temporal Database Concepts

Incorporating Time in Object-Oriented Databases Using Attribute Versioning

- A single complex object stores all temporal changes of the object
- **Time varying attribute**
  - An attribute that changes over time
    - E.g., age
- **Non-Time varying attribute**
  - An attribute that does **not** changes over time
    - E.g., date of birth

Spatial Databases

Spatial Database Concepts

- Keep track of objects in a multi-dimensional space
  - Maps
  - Geographical Information Systems (**GIS**)
  - Weather
- In general spatial databases are n-dimensional
  - This discussion is limited to 2-dimensional spatial databases

Spatial Databases

Spatial Database Concepts

- Typical Spatial Queries
  - **Range** query: Finds objects of a particular type within a particular distance from a given location
    - E.g., Taco Bells in Pleasanton, CA
  - **Nearest Neighbor** query: Finds objects of a particular type that is nearest to a given location
    - E.g., Nearest Taco Bell from an address in Pleasanton, CA
  - **Spatial joins** or overlays: Joins objects of two types based on some spatial condition (intersecting, overlapping, within certain distance, etc.)
    - E.g., All Taco Bells within 2 miles from I-680.

Spatial Databases

Spatial Database Concepts

- **R-trees**
  - Technique for typical spatial queries
  - Group objects close in spatial proximity on the same leaf nodes of a tree structured index
  - Internal nodes define areas (rectangles) that cover all areas of the rectangles in its subtree.
- **Quad trees**
  - Divide subspaces into equally sized areas

Multimedia Databases

Multimedia Database Concepts

- In the years ahead multimedia information systems are expected to dominate our daily lives.
  - Our houses will be wired for bandwidth to handle interactive multimedia applications.
  - Our high-definition TV/computer workstations will have access to a large number of databases, including digital libraries, image and video databases that will distribute vast amounts of multisource multimedia content.

Multimedia Databases

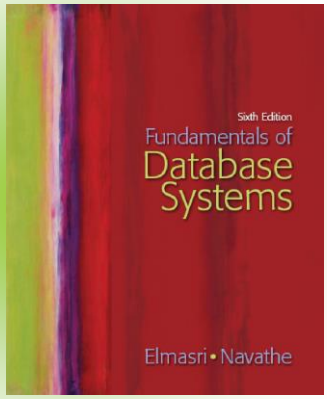
- Types of multimedia data are available in current systems (cont.)
  - **Images**: Includes drawings, photographs, and so forth, encoded in standard formats such as bitmap, JPEG, and MPEG. Compression is built into JPEG and MPEG.
    - These images are not subdivided into components. Hence querying them by content (e.g., find all images containing circles) is nontrivial.
  - **Animations**: Temporal sequences of image or graphic data.

Multimedia Databases

- Nature of Multimedia Applications:
  - Multimedia data may be stored, delivered, and utilized in many different ways.
  - Applications may be categorized based on their data management characteristics.

Chapter 28

Data Mining Concepts



Definitions of Data Mining

- The discovery of new information in terms of patterns or rules from vast amounts of data.
- The process of finding interesting structure in data.
- The process of employing one or more computer learning techniques to automatically analyze and extract knowledge from data.

Data Warehousing

- The data warehouse is a historical database designed for decision support.
- Data mining can be applied to the data in a warehouse to help with certain types of decisions.
- Proper construction of a data warehouse is fundamental to the successful use of data mining.

Knowledge Discovery in Databases (KDD)

- Data mining is actually one step of a larger process known as **knowledge discovery in databases** (KDD).
- The KDD process model comprises six phases
  - Data selection
  - Data cleansing
  - Enrichment
  - Data transformation or encoding
  - Data mining
  - Reporting and displaying discovered knowledge

Goals of Data Mining and Knowledge Discovery (PICO)

- **Prediction**:
  - Determine how certain attributes will behave in the future.
- **Identification**:
  - Identify the existence of an item, event, or activity.
- **Classification**:
  - Partition data into classes or categories.
- **Optimization**:
  - Optimize the use of limited resources.

Association Rules

- Association rules are frequently used to generate rules from **market-basket data**.
  - A market basket corresponds to the sets of items a consumer purchases during one visit to a supermarket.
- The set of items purchased by customers is known as an **itemset**.
- An **association rule** is of the form X=>Y, where X ={x_1, x_2, …., x_n }, and Y = {y_1,y_2, …., y_n} are sets of items, with x_i and y_i being distinct items for all i and all j.
  - For an association rule to be of interest, it must satisfy a minimum support and confidence.

Generating Association Rules

- The general algorithm for generating association rules is a two-step process.
  - Generate all itemsets that have a support exceeding the given threshold. Itemsets with this property are called **large** or **frequent itemsets**.
  - Generate rules for each itemset as follows:
    - For itemset X and Y a subset of X, let Z = X – Y;
    - If support(X)/Support(Z) > minimum confidence, the rule Z=>Y is a valid rule.

Generating Association Rules: The Apriori Algorithm

- The **Apriori algorithm** was the first algorithm used to generate association rules.
  - The Apriori algorithm uses the general algorithm for creating association rules together with downward closure and anti-monotonicity.

Generating Association Rules: Frequent-Pattern Tree Algorithm

- The **Frequent-Pattern Tree** Algorithm reduces the total number of candidate itemsets by producing a compressed version of the database in terms of an FP-tree.
- The FP-tree stores relevant information and allows for the efficient discovery of frequent itemsets.
- The algorithm consists of two steps:
  - Step 1 builds the FP-tree.
  - Step 2 uses the tree to find frequent itemsets.

Classification

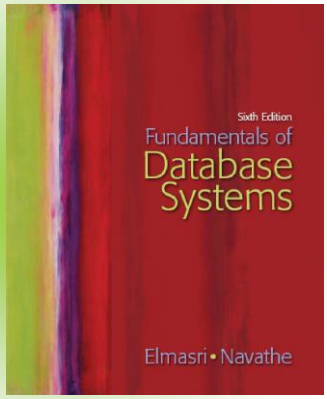
- **Classification** is the process of learning a model that is able to describe different classes of data.
- Learning is **supervised** as the classes to be learned are predetermined.
- Learning is accomplished by using a training set of pre-classified data.
- The model produced is usually in the form of a decision tree or a set of rules.

Clustering

- Unsupervised learning or clustering builds models from data without predefined classes.
- The goal is to place records into groups where the records in a group are highly similar to each other and dissimilar to records in other groups.
- The **k-Means** algorithm is a simple yet effective clustering technique.

Chapter 29

Overview of Data Warehousing and OLAP



Purpose of Data Warehousing

- Traditional databases are not optimized for data access only they have to balance the requirement of data access with the need to ensure integrity of data.
- Most of the times the data warehouse users need only read access but, need the access to be fast over a large volume of data.
- Most of the data required for **data warehouse** analysis comes from multiple databases and these analysis are recurrent and predictable to be able to design specific software to meet the requirements.
- There is a great need for tools that provide decision makers with information to make decisions quickly and reliably based on historical data.
- The above functionality is achieved by Data Warehousing and **Online analytical processing (OLAP)**

Comparison with Traditional Databases

- Data Warehouses are mainly optimized for appropriate data access.
  - Traditional databases are transactional and are optimized for both access mechanisms and integrity assurance measures.
- Data warehouses emphasize more on historical data as their main purpose is to support time-series and trend analysis.
- Compared with transactional databases, data warehouses are nonvolatile.
- In transactional databases transaction is the mechanism change to the database. By contrast information in data warehouse is relatively coarse grained and refresh policy is carefully chosen, usually incremental.

Classification of Data Warehouses

- Generally, Data Warehouses are an order of magnitude larger than the source databases.
- The sheer volume of data is an issue, based on which Data Warehouses could be classified as follows.
  - Enterprise-wide data warehouses
    - They are huge projects requiring massive investment of time and resources.
  - Virtual data warehouses
    - They provide views of operational databases that are materialized for efficient access.
  - Data marts
    - These are generally targeted to a subset of organization, such as a department, and are more tightly focused.

Data Modeling for Data Warehouses

- Traditional Databases generally deal with two-dimensional data (similar to a spread sheet).
  - However, querying performance in a multi-dimensional data storage model is much more efficient.
- Data warehouses can take advantage of this feature as generally these are
  - Non volatile
  - The degree of predictability of the analysis that will be performed on them is high.

Multi-dimensional Schemas

- Multi-dimensional schemas are specified using:
  - **Dimension table**
    - It consists of tuples of attributes of the dimension.
  - **Fact table**
    - Each tuple is a recorded fact. This fact contains some measured or observed variable (s) and identifies it with pointers to dimension tables. The fact table contains the data, and the dimensions to identify each tuple in the data.

Multi-dimensional Schemas

- Two common multi-dimensional schemas are
  - **Star schema:**
    - Consists of a fact table with a single table for each dimension
  - **Snowflake Schema:**
    - It is a variation of star schema, in which the dimensional tables from a star schema are organized into a hierarchy by normalizing them.

Multi-dimensional Schemas

- **Star schema**:
  - Consists of a fact table with a single table for each dimension.

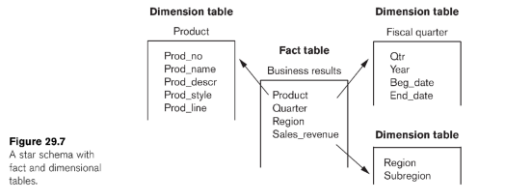


Figure 29.7
A star schema with fact and dimensional tables.

Multi-dimensional Schemas

- **Indexing**
  - Data warehouse also utilizes indexing to support high performance access.
  - A technique called bitmap indexing constructs a bit vector for each value in domain being indexed.
  - Indexing works very well for domains of low cardinality.

Building A Data Warehouse

- The Design of a Data Warehouse involves following steps.
  - Acquisition of data for the warehouse.
  - Ensuring that Data Storage meets the query requirements efficiently.
  - Giving full consideration to the environment in which the data warehouse resides.