Advanced DBMS

Unit 1: The Relational Model of Data and RDBMS Implementation Techniques

Database management system

• Collection of interrelated files and set of programs which allows users to access and modify files

• Primary Goal is to provide a convenient and efficient way to store, retrieve and modify information

Database Applications:

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Relational Database Management System

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

- Data is represented in terms of tuples (rows) in RDBMS.
- Relational database is most commonly used database. It contains number of tables and each table has its own primary key.
- Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

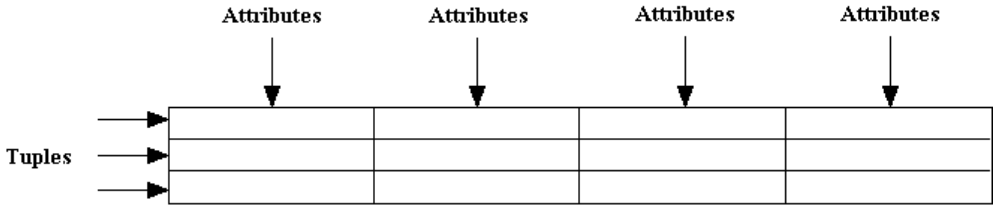
	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	**Normalization is not** present in DBMS.	**Normalization is** present in RDBMS.
4)	DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be **no relation between the tables**.	In RDBMS, data values are stored in the form of tables, so r**elationship** between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS **does not support distributed database**.	RDBMS **supports distributed database**.

8)	DBMS is meant to be for small organization and **deal with small data**. It supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.
9)	Examples of DBMS are file systems, xml etc.	Example of RDBMS are mysql, postgre, sql server, oracle etc.
10)	DBMS is used for simpler business applications	RDBMS is used for more complex applications.

Relational model conformity and Integrity,

The Relation

The Relation is the basic element in a relational data model.



- ~ Relation (file, table) is a two-dimensional table.
- ~ Attribute (i.e. field or data item) is a column in the table.
- ~ Each column in the table has a unique name within that table.
- ~ Each column is homogeneous. Thus the entries in any column are all of the same type (e.g. age, name, employee-number, etc).
- ~ Each column has a domain, the set of possible values that can appear in that column.
- ~ A Tuple (i.e. record) is a row in the table
- ~ Duplicate rows are not allowed
- ~ Cells must be single-valued (but can be variable length). Single valued means the following:
  - ~ Cannot contain multiple values such as 'A1, B2, C3'.
  - ~ Cannot contain combined values such as 'ABC-XYZ' where 'ABC' means one thing and 'XYZ' another.

Query optimization

Query optimization is a process of identifying how an RDBMS can improve on the performance of a query by re-ordering the operations. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

A query is a request for information from a database. It can be as simple as "finding the address of a person," or more complex like "finding the average salary of all the employed married men in California between the ages 30 to 39, that earn less than their wives." Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information.

Since database structures are complex, in most cases, and especially for not-very-simple queries, the needed data for a query can be collected from a database by accessing it in different ways, through different data-structures, and in different orders.

Each different way typically requires different processing time. Processing times of the same query may have large variance, from a fraction of a second to hours, depending on the way selected. The purpose of query optimization, which is an automated process, is to find the way to process a given query in minimum time. The large possible variance in time justifies performing query optimization, though finding the exact optimal way to execute a query, among all possibilities, is typically very complex, time consuming by itself, may be too costly, and often practically impossible. Thus query optimization typically tries to approximate the optimum by comparing several common-sense alternatives to provide in a reasonable time a "good enough" plan which typically does not deviate much from the best possible result.

Concurrency control and Transaction management

Properties of Transactions:

ACID properties:

- Atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.(Either all or nothing)
- 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.
- Durability or permanency: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

A transaction is a logical unit of database processing that includes one or more database access operations—these can include insertion, deletion, modification, or retrieval operations. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL. One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries.

Transaction processing systems

Transaction processing systems are systems with large databases and hundreds of concurrent users that are executing database transactions. Examples of such systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkout, and other similar systems. They require high availability and fast response time for hundreds of concurrent users.
If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction.

Basic operations are read and write

read_item(X): Reads a database item named X into a program variable.
write_item(X): Writes the value of program variable X into the database item named X.

Executing a read_item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing a write item(X) command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Step 4 is the one that actually updates the database on disk. In some cases the buffer is not immediately stored to disk, in case additional changes are to be made to the buffer.

Transaction States:

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

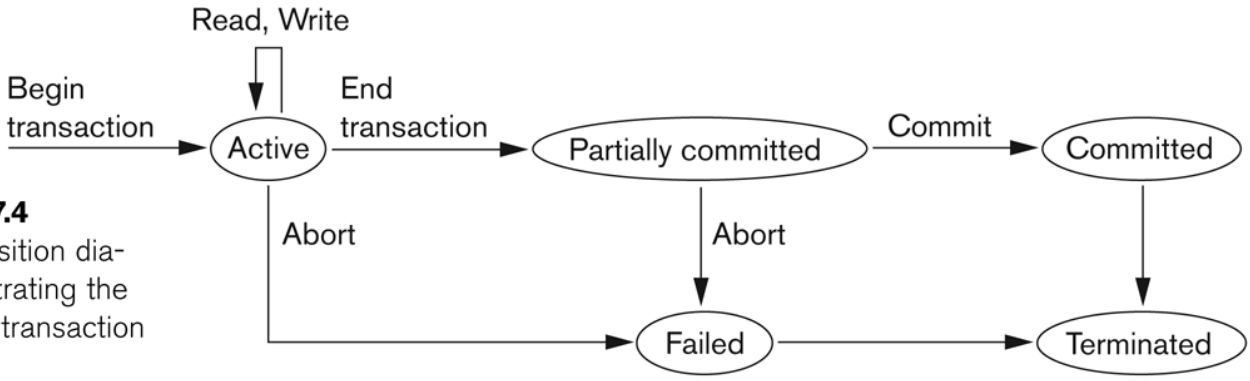


Figure 17.4
State transition diagram illustrating the states for transaction execution.

A transaction goes into an **active state** immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends, it moves to the **partially committed state**. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log). Once this check is successful, the transaction is said to have reached its commit point and enters the **committed state**.

However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The **terminated state** corresponds to the transaction leaving the system.

Hence, the recovery manager keeps track of the following operations:

- BEGIN_TRANSACTION: This marks the beginning of transaction execution.

• READ or WRITE: These specify read or write operations on the database items that are executed as part of a transaction.

• END_TRANSACTION: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted.

• 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.

Concurrency:

Concurrency is the Process of managing simultaneous operations on the database without having them interfere with one another. It Prevents interference when two or more users are accessing database simultaneously and at least one is updating data. Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

Concurrency control and recovery mechanisms are mainly concerned with the database access commands in a transaction. Transactions submitted by the various users may execute concurrently and may access and update the same database items. If this concurrent execution is uncontrolled, it may lead to problems, such as an inconsistent database.

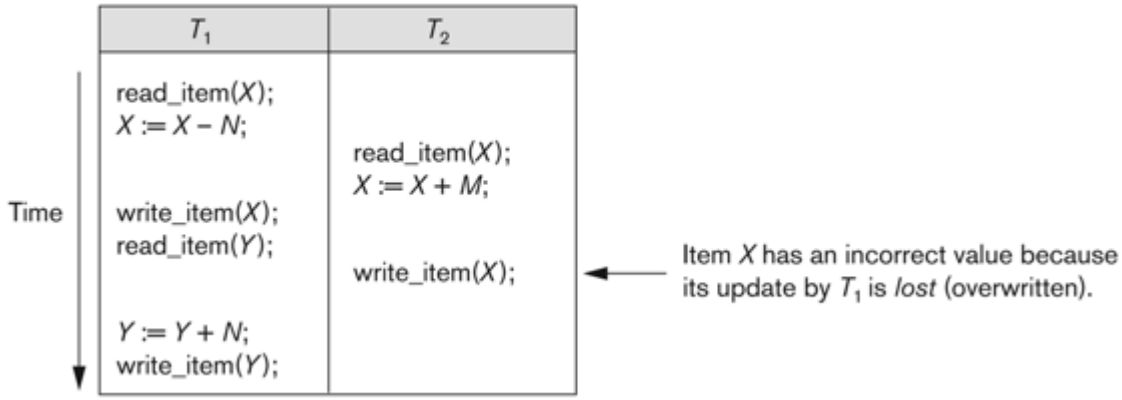
Problems that occur when Concurrent execution is uncontrolled:

(a) The lost update problem.
(b) The temporary update problem
(c) The incorrect summary problem

a) The Lost Update Problem:

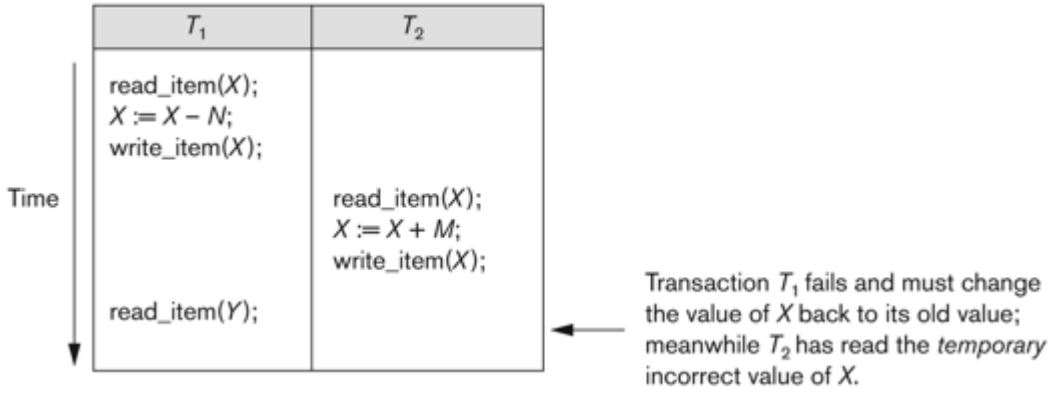
This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

For example, if X = 80 at the start (originally there were 80 reservations on the flight), N = 5 (transfers 5 seat reservations from the flight corresponding to X to the flight corresponding to Y), and M = 4 (reserves 4 seats on X), the final result should be X = 79; but in the interleaving of operations shown in Figure it is X = 84 because the update in that removed the five seats from X was lost.



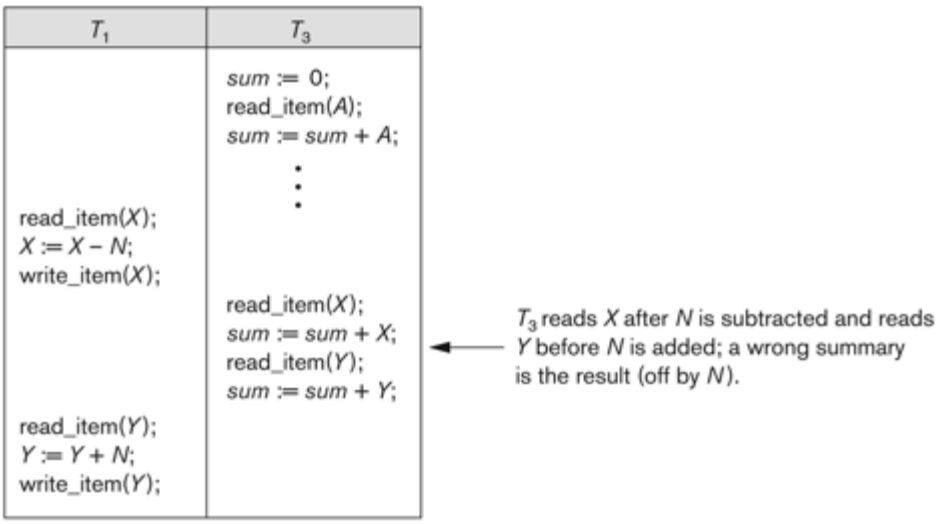
b) The temporary update problem

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. Example below shows an example where updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, however, transaction reads the "temporary" value of X, which will not be recorded permanently in the database because of the failure of. The value of item X that is read by is called dirty data, because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the dirty read problem.



c) 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.



Types of Failures

Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. *A computer failure (system crash):* A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.

2. *A transaction or system error:* Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

3. *Local errors or exception conditions detected by the transaction:* During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. Notice that an exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.

4. *Concurrency control enforcement:* The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5. *Disk failure:* Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. ***Physical problems and catastrophes:*** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Database performance tuning

Performance is the way a computer system behaves given a particular workload.
Performance is also affected by:

- The resources available in your system
- How well those resources are used and shared

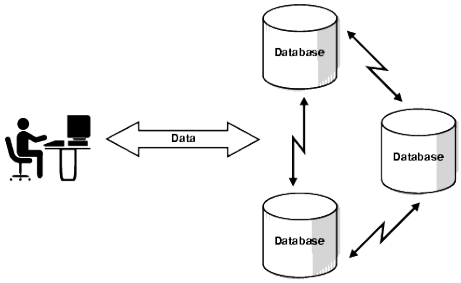
Database tuning describes a group of activities used to optimize and standardize the performance of a database. Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible.

How do you tune database performance?

- Tune database performance by designing tables efficiently
- Tune database performance by reducing disk writes and reads
- Creating indexes help greatly in tuning database performance
- Check for hardware as well as software problems
-

Distributed relational systems and Data Replication

A distributed database system allows applications to access data from local and remote databases.
A distributed database is a database in which storage devices are not all attached to a common processing unit such as the CPU and which is controlled by a distributed database management system. It may be stored in multiple computers, located in the same physical location; or may be distributed over a network of interconnected computers.



An example of a distributed database system is a mail order application with order entry clerks in several locations across the country.

The term **replication** refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment.

Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist. For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance.

Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible.
Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be complex and time-consuming depending on the size and number of the distributed databases. This process can also require a lot of time and computer resources.

Security considerations

Database security involves allowing or disallowing user actions on the database and the objects within it.

Types of Security

Database security is a very broad area that addresses many issues, including the following:

- Legal and ethical issues regarding the right to access certain information. Some information may be considered to be private and cannot be accessed legally by unauthorized persons.
- Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available—for example, credit ratings and personal medical records.
- System-related issues such as the system levels at which various security functions should be enforced—for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level.
- The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications—for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

Threads to databases:

Threads to databases results in the loss of degradation of some or all of the following commonly accepted security goals: integrity, availability, and confidentiality

Loss of Integrity: Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, deletion, modification, changing the status of data and deletion. Integrity is lost if unauthorized changes are made.

Loss of availability: Database availability refers to making objects available to a human user or a program to which they have a right.

Loss of confidentiality: Database confidentiality refers to the protection of data from unauthorized disclosure.

In a multiuser database system, the DBMS must provide techniques to enable certain users or user groups to access selected portions of a database without gaining access to the rest of the database. This is particularly important when a large integrated database is to be used by many different users within the same organization. For example, sensitive information such as employee salaries or performance reviews should be kept confidential from most of the database system's users. A DBMS typically includes a database

security and authorization subsystem that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

• Discretionary security mechanisms: These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

• Mandatory security mechanisms: These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification level to see only the data items classified at the user's own (or lower) classification level.