Data administration: refers to a function that concern's the organization data assets.

Database administration: a person or an office specific to a single database and it's applications.

Database administration tasks:

Summary of Database Administration Tasks
• Manage database structure
• Control concurrent processing
• Manage processing rights and responsibilities
• Develop database security
• Provide for database recovery
• Manage the DBMS
• Maintain the data repository

Database administration responsibilities:

Participate in Database and Application Development
• Assist in the requirements analysis stage and data model creation
• Play an active role in database design and creation
Facilitate Changes to Database Structure
• Seek communitywide solutions
• Assess impact on all users
• Provide configuration control forum
• Be prepared for problems after changes are made
• Maintain documentation

Concurrency control: to make sure that the user's wok doesn't influence the other user's work.

- There is no single concurrency technique that is ideal for all the problems of concurrency.

---

atomic transaction:

It means that the transaction will return a consistent version because the user is trying to do the action while he isn't able to do it, so the atomic transaction will prevent the user to make the action.

in a simple word:

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

Concurrent transactions: when two users try to do their action at the same time, this problem comes because the CPU can only execute one instruction at a time.

Concurrency problems are lost update (it means the value won't be the same ex. Bank credit) and inconsistent reads (after the lost update the values in the database won't be the same ex. Bank credit can't be read because of the inconsistent read)

Resource locking: is one of the solutions for concurrencies it prevents the applications from obtaining copies of the same record when the record is about to be changed ex. Using the ATM at the same time to take money will make a concurrent transaction so resource locking will prevent the action to be completed.

Lock terminology:

There are three locks terminology:

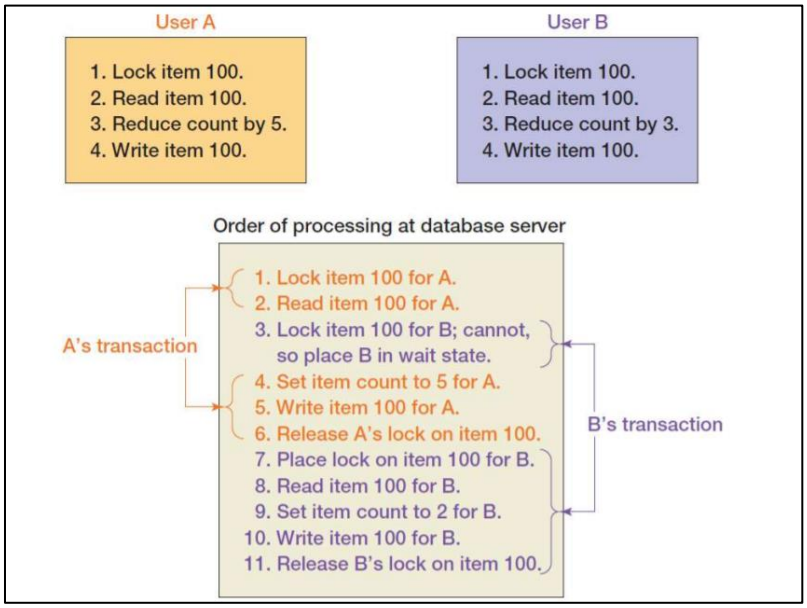
1- Implicit lock: locks placed by database management systems.
2- Explicit lock: are issued by the application programs (created outside the database management systems).
3- Lock granularity: it refers to the size of a locked resource ex. Rows, page, tables and database, levels.

Lock granularity is easy to manage but frequently causes conflicts.

Types of lock:

- Exclusive lock: prohibits (prevents) other users from reading the locked resource.
- Shared lock: allows other users to read the locked resource, but they can't update it.

Example of explicit lock:



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

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

How the two-phased locking works:

- The transactions can have locks as necessary (it's called growing phase).
- Once the first lock is released (it's called shrinking phase), no other lock can be obtained, and the process will be done.

No lock is released until the COMMIT or ROLLBACK command is issued.

Dead lock: it happens when two transaction are each waiting on a resource that the other transaction holds.

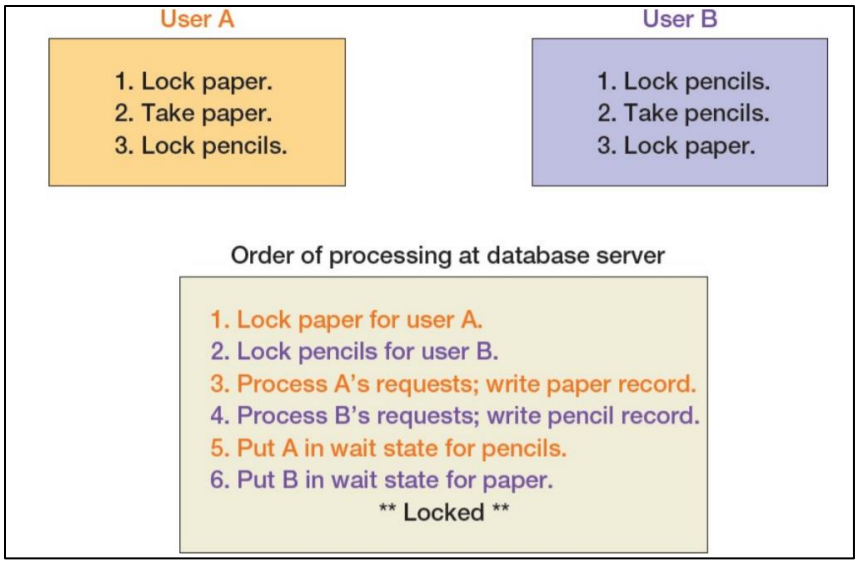
How to prevent deadlock:

- Allow the users to issue all lock requests at one time.
- Require all application programs to lock resources in the same order.

How to break the deadlock:

- Almost every database management system has algorithms to detect deadlocks.
- When the database management system detects a deadlock, it aborts one of the transactions and rolls back partially completed work so it can complete the tasks without a deadlock.

Deadlock ex.



Optimistic versus Pessimistic Locking:

Optimistic locking assumes that no transaction conflict will occur while Pessimistic locking assumes that conflict will occur.

How will the optimistic locking works?

DBMS processes a transaction by checking whether conflict occurred or not:

- If not, the transaction is finished.
- If so, the transaction is repeated until there is no conflict

In pessimistic locking Locks are issued before transaction is processed, and then the locks are released.

Optimistic locking is preferred for the Internet and for many intranet applications.

NOTES: optimistic locking read then locks, pessimistic locking locks then starts reading.

Most applications programs do not don't declare locks because of its complication, instead they mark transaction boundaries:

Transaction boundaries markers:

- SQL begin transaction.
- SQL commit transaction.
- SQL rollback transaction.

If the locking behavior needs to be changed, you will only need to change the lock declaration, not the hole application program, this is its advantage.

ACID transactions: atomic, consistent, isolated, and durable transactions

Atomic: either all or none of the database actions occur.

Durable: database committed changes are permanent.

Consistency: either statement level or transaction level consistency.

- Statement level consistency: each statement independently processes rows consistently.
- Transaction level consistency: all rows impacted by either of the SQL statements are protected from changes during the entire transaction.

Isolation: application programmers can declare the type of isolation level and to have the DBMS manage locks to achieve that level of isolation.

There are four transaction isolation levels:

- Read uncommitted.
- Read committed.
- Repeatable read.
- Serializable.

Data Read Problems Level

Data Read Problem Type	Definition
Dirty Read	The transaction reads a row that has been changed, but the change has *not* been committed. If the change is rolled back, the transaction has incorrect data.
Nonrepeatable Read	The transaction rereads data that has been changed, and finds updates or deletions due to committed transactions.
Phantom Read	The transaction rereads data and finds new rows inserted by a committed transaction.

Transaction Isolation Level

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

Cursor Summary

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

Database security: makes sure that only authorized users can perform authorized activities at authorized time.

To develop database security:

- Determine user's processing rights and responsibilities.
- Enforce security requirements using security features from both DBMS and application programs.

To see DBMS security guidelines, open page 9-38.

You can give permissions by using SQL data control language (DCL):

- SQL grant statement: it gives permissions to users and groups so that they can perform some operations on the data in the database.
- SQL revoke statement: it takes the existing permissions away from user and groups.

You can use your own security code in the database, and it can be written in the application program, but still you should use the DBMS security features first.

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

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

The SQL injection attack: it occurs when data from a user was entered to modify SQL statement.

To fix this problem you can ensure that the user can only add valid input with no additional SQL statements.

Example:

- Example: users are asked to enter their names into a Web form textbox:
  - User input:

    'Benjamin Franklin' OR TRUE

  - Resulting effective user input:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH09-09 *** */
SELECT      *
FROM        EMPLOYEE
WHERE       EMPLOYEE.Name = 'Benjamin Franklin' OR TRUE;
```

Database recovery: when system fails by an event, the database must restore to a usable state as soon as possible.

There is two recovery technique:

- Recovery via reprocessing.
- Recovery via rollback/rollforward.

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

Unfeasible strategy because:

- The recovered system may never catch up if the computer is heavily scheduled.
- Asynchronous events, although concurrent transactions, may cause different results.

Recovery via rollback/rollforward: it saves periodically the database logs, so that you have a list of history for your database.

Logs will have your records changing in your database.

When there is a failure, either rollback or rollforward is applied:

- Rollback: undo the erroneous changes made to the database and reprocess valid transactions.
- Rollforward: restore database using saved data and valid transactions since the last save.

Before image is used in rollback recovery because you will take the history of your database.

After image is used in the rollforward recovery because you will redo what you have restored.

In a simple word:

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

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

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

How it makes a checkpoint:

- DBMS refuses new requests, finishes processing outstanding requests, and writes its buffers to disk.
- The DBMS waits until the writing is successfully completed ◊ the log and the database are synchronized.

Checkpoints speed up database recovery process, by recovering the database using after-image since the last checkpoint.

Checkpoint can be done several times per hour.

NOTE: Most DBMS products automatically checkpoint themselves.