# CS14 504: Database Management Systems

## Module IV Notes

## Transaction processing

What is a transaction?

- A transaction is the basic logical unit of execution in an information system.
- A transaction is a sequence of operations that must be executed as a whole, taking a consistent (& correct) database state into another consistent (& correct) database state;
- A collection of actions that make consistent transformations of system states while preserving system consistency
- An indivisible unit of processing
- 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.
- **Transactionstates**:
    - Active state
    - Partially committed state
    - Committed state
    - Failed state
    - Terminated State
- **Transaction 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 limit of transaction execution.
    - **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

## Desirable properties of transactions (ACID)

**A Atomicity**: a transaction is an atomic unit of processing and it is either performed entirely or not at all
**C Consistency Preservation**: a transaction's correct execution must take the database from one correct state to another

**I Isolation/Independence**: the updates of a transaction must not be made visible to other transactions until it is committed (solves the temporary update problem)

**D Durability (or Permanency)**: if a transaction changes the database and is committed, the changes must never be lost because of subsequent failure

The isolation property is enforced by the concurrency control subsystem of the DBMS

- If every p( p ) transaction does not make its updates (write operations) visible to other transactions until it is committed,
- One form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks problem and eliminates cascading rollbacks
- Level of isolation of a transaction
  - A transaction is said to have level 0 ( ) zero isolation if it does not overwrite the dirty reads of higher-level transactions.
  - Level 1 (one) isolation has no lost updates,
  - Level 2 isolation has no lost updates and no dirty reads. v Level 3 isolation (also called true isolation) has, in addition to level 2 properties, repeatable reads

## CHARACTERIZING SCHEDULES

## Characterizing Schedules Based on Recoverability

Schedules – sequences that indicate the chronological order in which instructions of concurrent transactions are executed

- schedule for a set of transactions must consist of all instructions of those transactions
- must preserve the order in which the instructions appear in each individual transaction.

<u>Schedules classified on recoverability</u>:

- Recoverable schedule:
  - One where no transaction needs to be rolled back
  - A schedule S is recoverable if no transaction T in S commits until all transactions T’ that

have written an item that T reads have committed

- Cascadeless schedule
    - One where every transaction reads only the items that are written by committed transactions

Schedules requiring cascaded rollback

- A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back

Strict Schedules

- A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed has committed

## Characterizing Schedules Based on Serializability

- Serial schedule: – A schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule. Otherwise, the schedule is called nonserial schedule.
- Serializable schedule: – A schedule S is serializable if it is equivalent to some serial schedule of the same n transactions.
- Result equivalent: – Two schedules are called result equivalent if they produce the same final state of the database.
- Conflict equivalent: – Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.
- Conflict serializable: – A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.
- Being serializable is not the same as being serial Being serializable implies that the schedule is a correct schedule. – It will leave the database in a consistent state. – The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.
- **View equivalence:**A less restrictive definition of equivalence of schedules
- **View serializability:**Definition of serializability based on view equivalence.
- A schedule is *viewserializable* if it is *viewequivalent* to a serial schedule.

## Concurrency control Techniques

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.

Locking is an operation which secures

(a) permission to Read or

(b) permission to Write a data item for a transaction.

Example: Lock (X). Data item X is locked on 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.

## Two-Phase Locking

Two phase locking is a process used to gain ownership of shared resources without creating the possibility for deadlock. It breaks up the modification of shared data into "two phases".

There are actually three activities that take place in the "two phase" update algorithm:

1. Lock Acquisition
2. Modification of Data
3. Release Locks

The modification of data, and the subesquent release of the locks that protected the data are generally grouped together and called the second phase.

Two phase locking prevents deadlock from occuring in distributed systems by releasing all the resources it has acquired, if it is not possible to obtain all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

The resource (or lock) acquisition phase of a "two phase" shared data access protocol is usually implemented as a loop within which all the locks required to access the shared data are acquired one by one. If any lock is not acquired on the first attempt the algorithm gives up all the locks it had previously been able to get, and starts to try to get all the locks again.

This "back-off and re-try" strategy can be a problem in distributed systems. It is not guaranteed to give access to the desired resources within a finite time. This can lead to process starvation, if a single process never acquires all the locks needed for it to continue exectuion. This is a problem for real-time systems. Consequently, two phase locking protocols cannot be used in hard realtime applications.

The Two Phase Locking Protocol assumes that a transaction can only be in one of two phases.
**Growing Phase:** In this phase the transaction can only acquire locks, but cannot release any lock. The transaction enters the growing phase as soon as it acquires the first lock it wants. From now on it has no option but to keep acquiring all the locks it would need. It cannot release any lock at this phase even if it has finished working with a locked data item. Ultimately the transaction reaches a point where all the lock it may need has been acquired. This point is called **Lock Point**.
**Shrinking Phase:** After Lock Point has been reached, the transaction enters the shrinking phase. In this phase the transaction can only release locks, but cannot acquire any new lock. The transaction enters the shrinking phase as soon as it releases the first lock after crossing the Lock Point. From now on it has no option but to keep releasing all the acquired locks.

## Time stamp ordering

A **timestamp** is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or data item had been activated in any way. We, who use computers, must all be familiar with the concepts of “Date Created” or “Last Modified” properties of files and folders. Well, timestamps are things like that.
A timestamp can be implemented in two ways. The simplest one is to directly assign the current value of the clock to the transaction or the data item. The other policy is to attach the value of a logical counter that keeps incrementing as new timestamps are required.
The timestamp of a transaction denotes the time when it was first activated. The timestamp of a data item can be of the following two types:
**W-timestamp (Q)**: This means the latest time when the data item Q has been written into.
**R-timestamp (Q)**: This means the latest time when the data item Q has been read from.
These two timestamps are updated each time a successful read/write operation is performed on the data item Q.

## How should timestamps be used?

The timestamp ordering protocol ensures that any pair of conflicting read/write operations will be executed in their respective timestamp order. This is an alternative solution to using locks.
**For Read operations**:

1. If TS (T) < W-timestamp (Q), then the transaction T is trying to read a value of data item Q which has already been overwritten by some other transaction. Hence the value which T wanted to read from Q does not exist there anymore, and T would be rolled back.
2. If TS (T) >= W-timestamp (Q), then the transaction T is trying to read a value of data item Q which has been written and committed by some other transaction earlier. Hence T will be allowed to read the value of Q, and the R-timestamp of Q should be updated to TS (T).

**For Write operations**:

1. If TS (T) < R-timestamp (Q), then it means that the system has waited too long for transaction T to write its value, and the delay has become so great that it has allowed another transaction to read the old value of data item Q. In such a case T has lost its relevance and will be rolled back.
2. Else if TS (T) < W-timestamp (Q), then transaction T has delayed so much that the system has allowed another transaction to write into the data item Q. in such a case too, T has lost its

relevance and will be rolled back.

3. Otherwise the system executes transaction T and updates the W-timestamp of Q to TS (T).

## Multi version concurrency control

Multiversion schemes keep old versions of data item to increase concurrency.

Multiversion Two-Phase Locking

Each successful write results in the creation of a new version of the data item written.

Use timestamps to label versions.

When a read(Q) operation is issued, select an appropriate version of Q based on the timestamp of the transaction, and return the value of the selected version.

reads never have to wait as an appropriate version is returned immediately.

## Validation (Optimistic) concurrency control

The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through two or three phases, referred to as read, validation, and write.

• During the read phase, the transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values. All update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions.

During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to the write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.

• During the write phase, the changes are permanently applied to the database.

• The optimistic approach is acceptable for most read or query database

systems that require few update transactions.

## Granularity of Data Items

It deals with the cost of implementing locks depending upon the space and time. Here, space refers to data structure in DBMS for each lock and time refers to handling of lock request and release.

The cost of implementing locks depends on the size of data items. There are two types of lock granularity:

- Fine granularity
- Coarse granularity

Fine granularity refers for small item sizes and coarse granularity refers for large item Sizes.

Here, Sizes decides on the basis:

- a database record
- a field value of a database record
- a disk block
- a whole file
- the whole database

If a typical transaction accesses a small number of records it is advantageous that the data item granularity is one record. If a transaction typically accesses many records of the same file it is better to have block or file granularity so that the transaction will consider all those records as one data item.

## Multiple Granularity Locking

There are circumstances, however, where it would be advantageous to group several data items, and to treat them as one individual synchronization unit. For example, if a transaction *Ti* needs to access the entire database, and a locking protocol is used, then *Ti* must lock each item in the database. Clearly, executing these locks is time consuming. It would be better if *Ti* could issue a *single* lock request to lock the entire database. On the other hand, if transaction *Tj* needs to access only a few data items, it should not be required to lock the entire database, since otherwise concurrency is lost.

What is needed is a mechanism to allow the system to define multiple levels of **granularity**. We can make one by allowing data items to be of various sizes and defining a hierarchy of

data granularities, where the small granularities are nested within larger ones. Such a hierarchy can be represented graphically as a tree. A nonleaf node of the multiple-granularity tree represents the data associated with its descendants. In the tree protocol, each node is an independent data item.

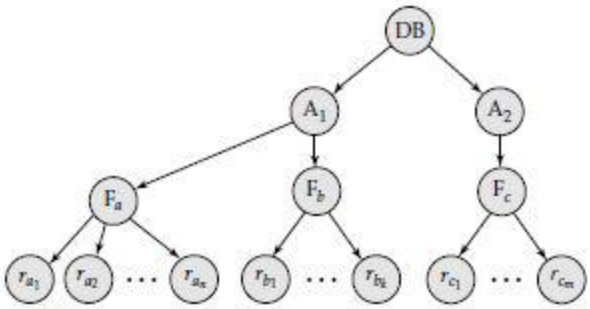



**Figure 16.16** Granularity hierarchy.

Consider the tree of Figure, which consists of four levels of nodes. The highest level represents the entire database. Below it are nodes of type *area*; the database consists of exactly these areas. Each area in turn has nodes of type *file* as its children. Each area contains exactly those files that are its child nodes. No file is in more than one area. Finally, each file has nodes of type *record*. As before, the file consists of exactly those records that are its child nodes, and no record can be present in more than one file.

Each node in the tree can be locked individually. As we did in the two-phase locking protocol, we shall use **shared** and **exclusive** lock modes. When a transaction locks a node, in either shared or exclusive mode, the transaction also has implicitly locked all the descendants of that node in the same lock mode. For example, if transaction *Ti* gets an **explicit lock** on file *Fc* of Figure 16.16, in exclusive mode, then it has an **implicit lock** in exclusive mode all the records belonging to that file. It does not need to lock the individual records of *Fc* explicitly.

If a node is locked in an **intention mode**, explicit locking is being done at a lower level of the tree (that is, at a finer granularity). Intention locks are put on all the ancestors of a node before that node is locked explicitly. Thus, a transaction does not need to search the entire tree to determine whether it can lock a node successfully.

There is an intention mode associated with shared mode, and there is one with exclusive mode. If a node is locked in **intention-shared (IS) mode**, explicit locking is being done at a lower level of the tree, but with only shared-mode locks. Similarly, if a node is locked in **intention-exclusive (IX) mode**, then explicit locking is being done at a lower level, with exclusive-mode or shared-mode locks. Finally, if a node is locked in **shared and intention-exclusive (SIX) mode**, the subtree rooted by that node is locked explicitly in shared mode, and that explicit locking is being done at a lower level with exclusive-mode locks.

The **multiple-granularity locking protocol**, which ensures serializability, is this:

Each transaction *Ti* can lock a node *Q* by following these rules:

**1.** It must observe the lock-compatibility function

**2.** It must lock the root of the tree first, and can lock it in any mode.

**3.** It can lock a node *Q* in S or IS mode only if it currently has the parent of *Q* locked in either IX or IS mode.

**4.** It can lock a node *Q* in X, SIX, or IX mode only if it currently has the parent of *Q* locked in either IX or SIX mode.

**5.** It can lock a node only if it has not previously unlocked any node (that is, *Ti* is two phase).

**6.** It can unlock a node *Q* only if it currently has none of the children of *Q* locked.

## Database recovery techniques -based on deferred update and immediate update

### We can recover the database in two methods:

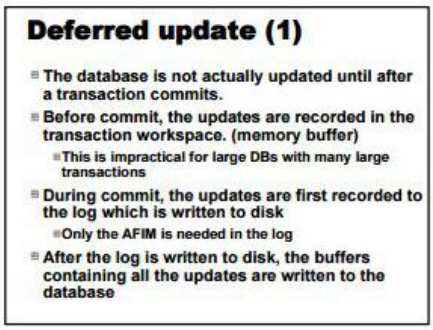
- **Log Based Recovery: -** In this method, log of each transaction is maintained in some stable storage, so that in case of any failure, it can be recovered from there to recover the database. But storing the logs should be done before applying the actual transaction on the database.

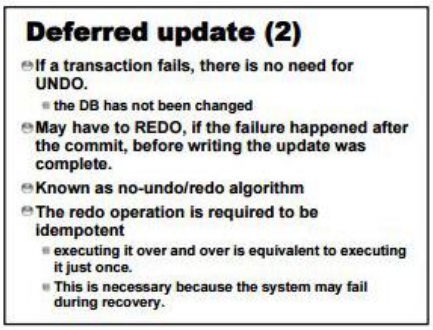
Every log in this case will have informations like what transaction is being executed, which values have been modified to which value, and state of the transaction. All these log information will be stored in the order of execution.

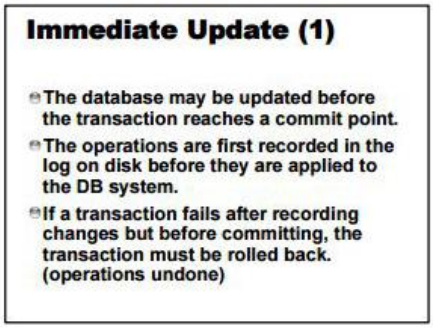
### There are two methods of creating this log files and updating the database

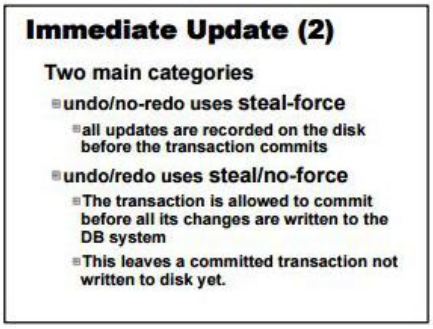
- **Deferred database modification: -** In this method, all the logs for the transaction is created and stored into stable storage system first. Once it is stored, the database is updated with changes. In the above example, after all the three log records are created and stored in some storage system, database will be updated with those steps.
- **Immediate database modification: -** After creating each log record, database is modified for each step of log entry immediately. In the above example, database is modified at each step of log entry i.e.; after first log entry, transaction will hit the database to fetch the record, then second log will be entered followed by updating the address, then the third log followed by committing the database changes.
- **Shadow paging: -** This is the method where all the transactions are executed in the primary memory. Once all the transactions completely executed, it will be updated to the database. Hence, if there is any failure in the middle of transaction, it will not be reflected

in the database. Database will be updated after all the transaction.




## Shadow paging

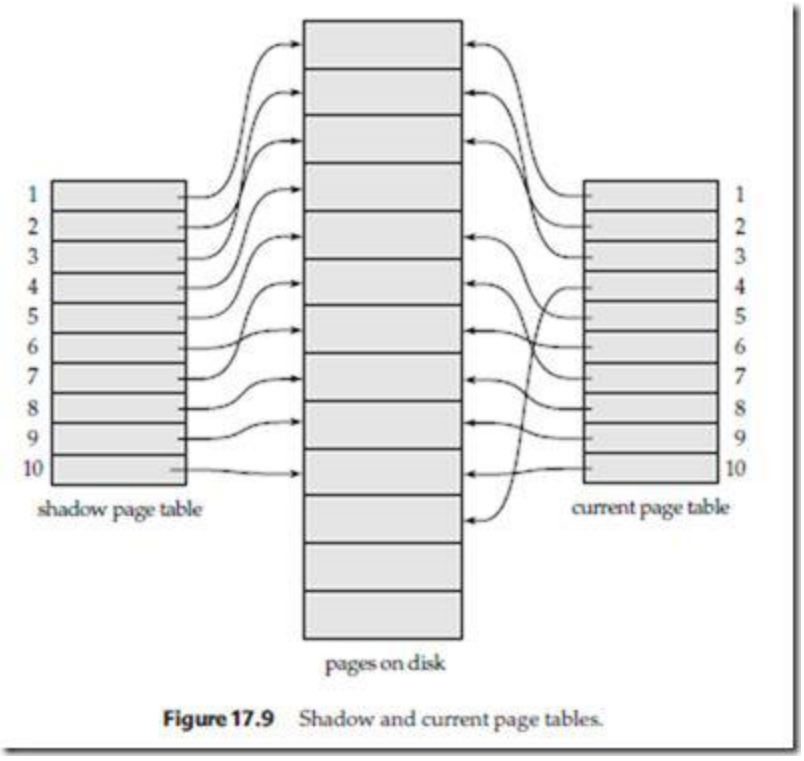



**Figure 17.9** Shadow and current page tables.

- This recovery scheme does not require the use of a log in a single-user environment.
- In a multiuser environment, a log may be needed for the concurrency control method.
- Shadow paging considers the database to be made up of a number of fixed-size disk pages. A **directory** with *n* entries is constructed, where the $i^{th}$ entry points to the $i^{th}$ database page on disk.
- The directory is kept in main memory if it is not too large, and all references—reads or writes—to database pages on disk go through it.
- When a transaction begins executing, the **current directory**—whose entries point to the most recent or current database pages on disk—is copied into a **shadow directory.** The shadow directory is then saved on disk while the current directory is used by the transaction.
- During transaction execution, the shadow directory is *never* modified.
- When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.* Instead, the new page is written elsewhere—on some previously unused disk block.
- The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block. For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory, and the new version by the current directory.
- To recover from a failure during transaction execution, it is sufficient to free the modified database pages and to discard the current directory. The state of the database before transaction execution is available through the shadow directory, and that state is recovered by reinstating the shadow directory. The database thus is

returned to its state prior to the transaction that was executing when the crash occurred, and any modified pages are discarded. Committing a transaction corresponds to discarding the previous shadow directory.

- One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies.
- Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant.
- A further complication is how to handle **garbage collection** when a transaction commits.

## ARIES recovery algorithm

ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) recovery is based on the Write Ahead Logging (WAL) protocol. Every update operation writes a log record which is one of

1. An undo-only log record: Only the before image is logged. Thus, an undo operation can be done to retrieve the old data.
2. A redo-only log record: Only the after image is logged. Thus, a redo operation can be attempted.
3. An undo-redo log record. Both before image and after images are logged.

Every log record is assigned a unique and monotonically increasing log sequence number (LSN). Every data page has a page LSN field that is set to the LSN of the log record corresponding to the last update on the page. WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk. For performance reasons, each log write is not immediately forced to disk. A log tail is maintained in main memory to buffer log writes. The log tail is flushed to disk when it gets full. A transaction cannot be declared committed until the commit log record makes it to disk.

Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table (which gives the list of active transactions) and the dirty page table (the list of data pages in the buffer pool that have not yet made it to disk). A master log record is maintained separately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk. On restart, the recovery subsystem reads the master log record to find the checkpoint's LSN, reads the checkpoint record, and starts recovery from there on.

The actual recovery process consists of three passes:

1. Analysis. The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.
2. Redo. Starting at the earliest LSN determined in pass (1) above, the log is read forward and each update redone.
3. Undo. The log is scanned backward and updates corresponding to loser transactions are undone.

## Introduction to Database security

Design of secure DBMS assumes identification of security risks and selection of right *security policies* (what is the security system supposed to do) and *mechanisms* (the way we are going to achieve that) for their neutralization.

Secure database system should satisfy three basic requirements on data protection: *security*, *integrity* and *availability*. What is the content of those words?

- ***Ensuring security*** - preventing, detecting and deterring improper disclosure of information. This is especially important in strongly protected environments (e.g. army).
- ***Ensuring integrity*** - preventing, detecting and deterring improper changes of information. The proper function of any organization depends on proper operations on proper data.
- ***Ensuring system availability*** - effort for prevention of improper denial of service that DBMS provides.

At this moment we have basic image of information system security and we can take a look at concrete aspects that should be covered with DBMS security mechanisms.

1. ***Protection from improper access***- only authorized users should be granted access to objects of DBMS. This control should be applied on smaller objects (record, attribute, value).
2. ***Protection from inference*** - inference of confidential information from available data should be avoided. This regards mainly statistical DBMSs.
3. ***Database integrity*** - partially is ensured with system controls of DBMS (atomic

transactions) and various back-up and recovery procedures and partially with security procedures.

4. ***Operational data integrity*** - logical consistence of data during concurrent transactions (concurrency manager), serializability and isolation of transactions (locking techniques).
5. ***Semantic data integrity*** - ensuring that attribute values are in allowed ranges. This is ensured with integrity constraints.
6. ***Accountability and auditing*** - there should be possibility to log all data accesses.
7. ***User authentication*** - there should be unambiguous identification of each DBMS user. This is basis for all authorization mechanisms.
8. ***Management and protection of sensitive data*** - access should be granted only to narrow round of users.
9. ***Multilevel security*** - data may be classified according to their sensitivity. Access granting should then depend on that classification.
10. ***Confinement (subject isolation)*** - there is necessity to isolate subjects to avoid uncontrolled data flow between programs (memory channels, covert channels).

At least five aspects from the previous list must be ensured with special techniques that do not exist in unsecure DBMSs. There are three basic ways to do it:

- ***flow control*** - we control information flows in frame of DBMS
- ***inference control*** - control of dependencies among data
- ***access control*** - access to the information in DBMS is restricted

## Database Security Issues

Issues depends on Security types and database threats. Security Types includes:

1. Legal and ethical issues regarding the right to access certain information. Some information may be deemed to be private and cannot be accessed by unauthorized persons.

2. 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.

3. System related issues such as system levels at which various security functions should be enforced for example whether a security function should be handled at the physical hardware level, operating system level and the DBMS level.

4. The need in some organization 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.

Database Threats includes:

1. Loss of integrity  Database integrity refers to the requirement that information be protected from improper modification includes creation, insertion, modification, changing the status of data, and deletion. Integrity lost if authorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data would result in inaccuracy, fraud or erroneous decisions.

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

3. Loss of confidentiality  Data confidentiality refers to the protection of data from unauthorized disclosure. The impact is of confidential information can range from violation of data privacy act. Unauthorized, unanticipated or unintentional disclosure could result in loss of public confidence, or legal action against the organization.

Database Security Control Measures There are four main control measures used to provide security of data in databases. They are :

1. Access control  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 the login process by the DBMS.

2. Inference control  Statistical databases are used to provide statistical information or summaries of values based on various criteria. Security for statistical databases must ensure that information about individuals cannot be accessed. It is possible to deduce or infer certain facts concerning individuals from queries that involve only summary statistics on groups, consequently this must not permitted either. This problem called statistical database security and corresponding control measures are called inference control measures.

3. Flow control  It prevents information from flowing in such a way that it reaches unauthorized users. Channels that are pathways for information to flow implicitly in ways that violate security policy of an organization are called covert channels.

4. Data encryption  It is used to protect sensitive data that is transmitted via some type of communication network. Encryption can be used to provide additional protection for sensitive portions of a database. The data is encoded using some coding algorithm. An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decryption algorithms to decipher data. Encrypting techniques are very difficult to decode without a key have been developed for military applications.

## Access control based on granting/revoking of privileges

A *privilege* is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to

- connect to the database (create a session)
- create a table
- select rows from another user's table
- execute another user's stored procedure

You grant privileges to users so these users can accomplish tasks required for their job. You should grant a privilege only to a user who absolutely requires the privilege to accomplish necessary work. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly. For example, you can explicitly grant the privilege to insert records into the EMP table to the user SCOTT.
- You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the EMP table to the role named CLERK, which in turn you can grant to the users SCOTT and BRIAN.

Because roles allow for easier and better management of privileges, you should normally grant privileges to roles and not to specific users.

There are two distinct categories of privileges:

- system privileges
- schema object privileges

## System Privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges. There are over 60 distinct system privileges.

### *Granting and Revoking System Privileges*

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, you can use the roles to manage system privileges (for example, roles permit privileges to be made selectively available).

## Granting and Revoking Roles

You grant or revoke roles from users or other roles using the following options:

- the Grant System Privileges/Roles dialog box and Revoke System Privileges/Roles dialog box of Oracle Enterprise Manager
- the SQL commands GRANT and REVOKE

MODULE - III

SQL uses the terms Table, row and column for the relational model ~~relation~~ terms relation, Tuple, and attribute.

## Schema and Catalogue

An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user account as well as descriptors for each element in the schema. A schema is created via the CREATE SCHEMA statement.

Eg:-

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Smith'
```

A named collection of schema in an SQL environment is called catalog.

## CREATE TABLE Command

The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and constraints.

Eg:-

```
CREATE TABLE EMPLOYEE
(Fname      VARCHAR(15) NOT NULL,
 Minit      char,
 Lname      VARCHAR(15) NOT NULL,
 Ssn        CHAR(9)
 Bdate      DATE
 Super_SSn  CHAR(9),
PRIMARY KEY (Ssn), FOREIGN KEY (Super_SSn)
REFERENCES EMPLOYEE (SSn));
```

## Attribute data types and Domains in SQL

### Numeric data types

INTEGER, INT, SMALLINT

FLOAT, REAL, DOUBLE PRECISION

DECIMAL, DEC, NUMERIC.

### Character-string

CHAR, CHARACTER

VARCHAR, CHAR VARYING.

### Bit string

BIT(n), BIT VARYING(n).

2

### Boolean

values as TRUE or FALSE.

### DATE

> Data type has 10 positions

> format YY Year, Month and Day.

### Timestamp.

Includes DATE and TIME fields. and an optional Time zone fields.

Eg:- '2008-09-27 09:12:47.648302'.

### Interval

A relative value that can be used to increment or decrement an absolute value of date, time or timestamp.

## Giving names to constraints.

A constraint name can be defined with the keyword CONSTRAINT. A constraint name is used to identify a particular constraint. The constraint must be dropped later and replaced with another constraint. Giving names to constraints are optional.

## Specifying constraints on Tuples Using CHECK.

The General constraints can be specified using the CHECK clause at the end of a CREATE TABLE statement.

eg:- Make sure that manager start date is later than the department creation date.

```
CHECK (Dept_Create_date <= Mgr_Start_date);
```

The CHECK clause can also be used to specify more general constraints.

## Basic Retrieval Queries in SQL:

SELECT statement is used to retrieve information from database. The basic form of the SELECT statement, sometimes called a mapping or select-from-where block,

Syntax

```
SELECT   <attribute list>
FROM     <table list>
WHERE    <condition>;
```

Where,

<attribute list> - list of attribute names whose values are to be retrieved by query.

<table list> - is a list of the relation names required to proces the query.

<condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

eg:-

① Retrieve the birth date and address of the employee whose name is 'John B. Smith'.

```
SELECT   Bdate, Address
FROM     EMPLOYEE
WHERE    Fname = 'John' AND Minit = 'B' AND
                           Lname = 'Smith';
```

② Retrive the name and address of all employees who work for the 'Research' department.

```
SELECT  Fname, Lname, Address
FROM    EMPLOYEE, DEPARTMENT
WHERE   Dname = 'Research' AND Dnumber = Dno;
```

3. For every project located in 'Stafford' list the project number, the controlling department number, and the department manager's last name, address and birthdate.

(iv)

```
SELECT   Pnumber, Dnum, Lname, Address, Bdate
FROM     PROJECT, DEPARTMENT, EMPLOYEE
WHERE    Dnum = Dnumber AND Mgr_ssn = Ssn AND
         plocation = 'Stafford';
```

A query that involves only selection and join conditions plus projection attributes is known as select-project-join query.

Ambiguous attribute names:

In SQL the same name can be used for two or more attributes as long as attributes are in different relations. i.e, multiple queries refers to two or more attributes with the same name. To avoid this ambiguity the attribute name qualify with Relation name.

Eg:-

```
SELECT   Fname, EMPLOYEE.Name, Address
FROM     EMPLOYEE, DEPARTMENT
WHERE    DEPARTMENT.Name = 'Research' AND
AND      DEPARTMENT.Dnumber = EMPLOYEE
                              Dnumber.
```

## Unspecified WHERE clause and Use of the Asterisk:

A missing WHERE clause indicates no condition on tuple selection.

① eg:- Select all EMPLOYEE SSNs.

```
SELECT  Ssn
FROM    EMPLOYEE;
```

② Retrieve all the attribute values of the selected tuples any employees who works in DEPARTMENT number 5.

```
SELECT  *
FROM    EMPLOYEE
WHERE   Dno = 5;
```

## DISTINCT

Duplicate tuples can be eliminated using the DISTINCT keyword in the SELECT clause.

eg: Retrieve the salary of every employee and all distinct salary values.

```
SELECT  DISTINCT Salary
FROM    EMPLOYEE.
```

## Substring Pattern Matching:

SQL allows comparison conditions only for character strings, using the LIKE comparison operator. Partial strings are specified using two reserved characters:

% → replaces arbitrary number of zeros.

_ → replaces single character.

eg:- Retrieve all employees whose address is in Houston, Texas.

```
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    Address LIKE '% Houston, TX %';
```

(4)

## Ordering of Query Results:

SQL allows the user to order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result, by using the ORDER BY clause.

eg:- Retrieve a list of employees and the projects they are working on, ordered by department and within each department, ordered alphabetically by last name then first name.

```
SELECT   D.Dname, E.Lname, E.Fname, P.Pname
FROM     DEPARTMENT D, EMPLOYEE E, WORKS_ON W,
         PROJECT P
WHERE    D.Dnumber = E.Dno AND E.Ssn =
         W.Essn AND W.Pno = P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

In SQL, the three commands that can be used to modify the database are INSERT, DELETE and UPDATE.

(5)

## INSERT Command

INSERT is used to add a single tuple to a relation. The relation name and a list of values for the tuple should be specified. The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

Eg:- To add new tuple in to EMPLOYEE table:

```
INSERT INTO   EMPLOYEE
VALUES        ('Richard', 'K', 'Marini', '653298653',
              '1962-12-30', 'TVM', 37000, '653298653');
```

> SQL support integrity constraints.
>

## DELETE COMMAND

The DELETE command removes tuples from a relation. It includes WHERE clause, to select the tuples to be deleted. Tuples are explicitely deleted from one table at a time.

Eg:-1) DELETE FROM EMPLOYEE
WHERE Lname = 'Brown';

2) DELETE FROM EMPLOYEE;

(8)

## The UPDATE Command:

The UPDATE command is used to modify attribute values of one or more selected tuples. WHERE clause selects the tuples to be modified from a single relation.

(b)

```
Eg:-  UPDATE   PROJECT
      SET      Plocation = 'Bellaire', Dnum = 5
      WHERE    Pnumber = 10.
```

## Additional features of SQL

- Various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins and recursive queries are allowed.
- SQL has various techniques for writing programs various programming languages.
- Additional SQL commands for design parameters, file structures are allowed.
- SQL has transaction control commands.
- SQL has language constructs for specifying the granting and revoking of privileges to users.
- SQL has language constructs for triggers.
- SQL has incorporated with object oriented models.
- SQL and relational databases can interact with new technologies such as XML.

6

## More Complex Retrieval Queries.

### Comparisons involving NULL values:

(7)

Unknown value – Exists but is not known.

Unavailable or withheld value – value purposefully withheld.

Not applicable – the attribute is undefined for the tuple.

Eg:- Retrieve the names of all employees who do not have supervisors.

```
SELECT  Fname, Lname
FROM    EMPLOYEE
WHERE   Super_ssn IS NULL;
```

### Nested Queries

Nested Queries, which are complete select-from-where clause blocks within the WHERE clause of another query. This query is called Outer Query.

Eg:- Retrieve the Essn of all employees who work the same combination on some project that employee 'John Smith' works on.

```
SELECT  DISTINCT Essn
FROM    WORKS_ON
WHERE   (Pno, Hours) IN (SELECT Pno, Hours
                         FROM   WORKS_ON
                         WHERE  Essn = '123456');
```

(10)

## Correlated Nested Queries

A condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.

(8)

## The EXISTS and UNIQUE functions in SQL.

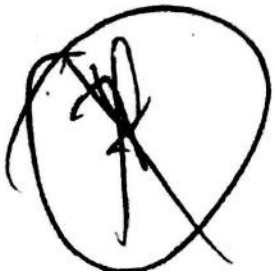
The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty or not. The result of EXISTS is a boolean value.

Eg:- Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
1):   SELECT   E.Fname , E.Lname
      FROM     EMPLOYEE AS E
      WHERE    EXISTS (SELECT *
                       FROM   DEPENDENT AS D
                       WHERE  E.Ssn = D.ESSN AND
                       E.Sex A = D.Sex AND
                       E.Fname = D.Dependent_name);
```

2) Retrieve the names of employees who have no dependents.

```
      SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    NOT EXISTS (SELECT * FROM DEPENDENT
                           WHERE Ssn = ESSN);
```



## Aggregate functions in SQL.

Aggregate functions are used to summarize information from multiple tuples into single-tuple summary.

Grouping is used to create subgroups of tuples before summarization.

### Aggregate functions.

(9)

COUNT
SUM
MAX
MIN
AVG.

eg:- find the sums of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
SELECT  SUM (salary), MAX (salary), MIN(salary),
        AVG (salary)
FROM    EMPLOYEE;
```

(2). Count the number of distinct salary values in the database.

```
SELECT  COUNT (DISTINCT Salary)
FROM    EMPLOYEE;
```

### GROUP BY and HAVING Clause.

GROUP BY Clause specifies the grouping attributes.

Eg:- For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT   Dno, COUNT (*), AVG(salary)
FROM     EMPLOYEE
GROUP BY Dno;
```

HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved.

Eg:- For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT   Pnumber, Pname, COUNT (*)
FROM     PROJECT, WORKS_ON
WHERE    Pnumber = Pno
GROUP BY Pnumber, Pname
HAVING   COUNT (*) > 2;
```

# VIEWS ASSERTIONS & TRIGGERS.



## Assertions:

Assertions can be used to specify additional types of constraints that are outside the scope of the built-in relational model constraints such as Primary key, entity integrity, and Referential integrity.

Users can specify general constraints via declarative assertions, using the CREATE ASSERTION statement.

Eg:- The salary of an employee must not be greater than the salary of the manager of the department.

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS ( SELECT *
                    FROM EMPLOYEE E, EMPLOYEE M,
                    DEPARTMENT D
                    WHERE E.Salary > M.Salary
                    AND E.Dno = D.Dnumber
                    AND D.Mgr_ssn = M.Ssn));
```

14

## Triggers:

Triggers can be used to specify automatic actions that the database systems will perform when certain events and conditions occur. This type of functionality is generally referred to as active databases.

The CREATE TRIGGER statement is used to implement triggers in database.

Eg:- To check whenever an employee's salary is greater than the salary of his or her direct supervisor in the company database.

```
CREATE TRIGGER Salary_violation
BEFORE INSERT OR UPDATE OF Salary, Super_ssn
ON EMPLOYEE
FOR EACH ROW
WHEN (New.Salary > (SELECT Salary FROM EMPLOYEE
                    WHERE SSN = new.Super_SSN))
INFORM_Super (new.Super_SSN,
              new.SSN);
```

The Trigger statement has three components:

1. The event(s):
   These are usually database update operations that are explicitely applied to the database.

2. The condition: That determines whether the rule action should be executed.

3. The action to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction that will be automatically executed.

## Views

A view in SQL is a single table that is derived from other tables. These other tables can be base tables or previously defined views. A view does not exist in physical form, it is considered to be a virtual table, in contrast to base table whose tuples are always physically stored in the database. This limits the pos...

update operations that can be applied to views, but it does not provide any limitations on queries.

The CREATE VIEW command is used to specify a view in database. The view is given a table name, a list of attribute names and a query to specify the contents of the view.

eg:- CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSn = Essn AND Pno = Pnumber;

(13)

The DROP VIEW command is used to dispose a view from database.

eg:- DROP VIEW WORKS_ON1;

View Implementation

Two main approaches to implement a view are:

1. Query Modification: Involves modifying or transforming the view query into a query on the underlying base tables.

eg:- SELECT Fname, Lname
FROM EMPLOYEE, PROJECT, WORKS_ON.
WHERE SSn = Essn AND Pno = Pnumber
AND Pname = 'ProductX';

The disadvantage of this approach is that it is inefficient for views defined via complex queries that are time-consuming to execute.

2. View Materialization, involves physically creating a temporary view table, when the view is first queried and keeping the table on the assumption that other queries on the view will follow. In this case, automatic updation of view table is needed.

Eg:-
```
UPDATE  WORKS-ON
SET     Pname = 'product Y'
WHERE   Lname = 'Smith' AND Fname = 'John'
        AND Pname = 'productX';
```

The disadvantages are:
Updating of views is complicated and can be ambiguous.

## Features of View

> A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as attributes with the NOT NULL constraint that do not have default values specified.

> Views defined on multiple tables using joins are generally not updatable.

> Views defined using grouping and aggregate functions are not updatable.