Integrity

Data integrity is a fundamental component of information security. In its broadcast use, "data integrity" refers to the accuracy and consistency of data stored in database, data warehouse, data marct or other construct. Data with "integrity is said to have a complete or whole structure. As a simple example, to maintain data integrity, numeric columns/cells should not accept alphabetic data.

Importance of DB Integrity

1. Enforcing business rules in the code of database application.
2. Using stored procedures to control access to data.
3. Enforcing business rules with triggered stored database procedure.

Different ways to ensure DB Integrity

There are 3 different ways that could be applied to ensure database integrity. They are:

1. Entity Integrity

This is concerned with the concept of primary keys. The rule states that every table must have its own primary key & that each has to be unique and not null.

2. Referential Integrity

This is concerned with the concept of foreign keys. The rule states that the foreign key value can be in two states. The first state is that the foreign key value would refer to a primary key value of another table or it can be null. Being null could simply mean that there is no relationship or the relationship is unknown.

Nayeem Mahmood / 4AM / C161026

3. Domain Integrity

This states that all columns in a relational database are in a defined domain. This concept ensures that all data in a database can be traced & connected to other data. This ensures that everything is recoverable and searchable.

The tests that must be made to preserve referential integrity for insert and delete operations

For Insert Operation

During an insert operation, if a row is to be ~~updated~~ added to a table and that row contains a value in a foreign key column that is not present in the table referenced by the foreign key, adding the row would violate referential integrity. So in this case, we must test the foreign key column value, whether it is present in the referenced table or not.
An alternative to discarding rows that violates referential integrity is to generate a row with new value and add it to the referenced table, thereby preserving referential integrity.

For Delete Operation

During a delete operation, if a row to be deleted contains a value that is referenced by a foreign key in another table, then the referential integrity can be avoided by either of the two ways:

1. Deleting the original row and also deleting the referencing row from the other table. This action is called Cascaded Deletion and can cascade through a series of referencing tables.

2. Deleting neither of the rows i.e. taking no action. This action is called Restricted Deletion.

Consider the following relational database:
Customer (customer_id, customer_name, street, city, mobile)
Sales (sales_id, customer_id, Product_id, quantity, date, time)
Product (Product_id, Product_name, Unit_Price)

Given an SQL DDL definition of this database. Identify referential integrity constraints that should hold, and include them in the DDL definition.

```
Create table customer
(
        customer_id varchar(50),
        customer_name varchar(100) not null,
        street varchar(100),
        city varchar(100),
        mobile varchar(11) unique,
        primary key (customer_id)
);

Create table Product
(
        Product_id  varchar(50),
        Product_name varchar(100) not null,
        Unit_price  decimal(7,2),
        primary key (Product_id),
        check (Unit_Price >= 500)
);
```

```
create table sales
(
        sales_id varchar(50),
        customer_id varchar(50),
        Product_id varchar(50),
        quantity decimal(3,1),
        date date,
        time time,
        check (quantity > 0),
        primary key (sales_id),
        foreign key (customer_id) references Customer,
        foreign key (Product_id) references Product

);
```

Domain

A domain is defined as the set of all unique values permitted for an attribute. For example, a domain of date is the set of all possible valid dates, a domain of integer is all possible whole numbers etc.

Domain Constraints

Domain constraints are user defined data types and we can define them like this :

Domain constraints = data type + Constraints, where Contraints are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT.

Example 1: To create a table "Student_Info" with "stu_id" field having value greater than 100, we can create a domain and table like this :

```
create domain id_value int
constraint id_test
check (value > 100);

create table Student_Info (
stu_id id_value PRIMARY KEY,
stu_name varchar (100),
stu_age int );
```

Example 2: To create a table "bank_acc" with "acc_type" field having value either "checking" or "saving", we can create a domain & table like this:

```
create domain account_type char (12)
constraint acc_type_test
check (value in ("checking", "saving"));

create table bank_acc (
acc_no int PRIMARY KEY,
acc_holder_name varchar (100),
acc_type account_type );
```

Create a domain constraint (1) min_salary for an employee table in a company db that will not allow entering monthly salary less than 5300 or (2) min_age which will not allow entering employees less than 18 years old.

(1) & (2) →

```
create domain min_salary float
constraint salary_test
check (value >= 5300);
```

```
create domain min_age int
constraint age_test
check (value >= 18);

create table employee (
emp_id int PRIMARY KEY,
emp_name varchar(100) NOT NULL,
emp_age min_age,
emp_salary min_salary );
```

Assertion

An assertion is a statement in SQL that ensures a certain condition will always exist in database. Assertions are like column and table constraints, except they are specified separately from table definitions.

Write an assertion for the bank database to ensure that the assets value for the chawkbazar branch is equal to the sum of all the amounts lent by the chawkbazar branch.

```
create assertion sum_test check
    (not exists (select * from branch
                 where branch_name = "chawkbazar" and
                 assets ≠ (select sum(amount) from loan
                           where branch_name = "chawkbazar")))
```

Triggers

Triggers are blocks of SQL code, which are executed as a result of any insert/update/delete on the table. There will not be any explicit trigger calls. It is always automatically executed. It might perform single query or set of queries to meet the goal. But it will be always written as an SQL block. The set of transactions inside the trigger can be performed on the same table or one or more other tables.

Necessity of using Triggers

1. If any change needs to be done in the application, it is done at the trigger, instead of changing each application that is accessing the trigger.
2. Triggers are easy to maintain and they enforce faster application development.
3. Triggers are useful for tasks such as enforcing business rules, validating input data and keeping an audit trail.

Write an SQL trigger to carry out the following action:

On delete of an account, for each owner of the account, check if the owner has any remaining accounts and if s/he doesn't, delete him/her from the depositor relation.

```
create trigger delete_all
after delete on account
referencing old row as old_row
for each row
delete from depositor
where depositor.customer_name not in
    (select customer_name from depositor
     where account_number <> old_row.account_number)
end;
```

Different factors that are liable for malicious access of Database

1. **SQL Injection**

This is the most common type of attack in a database system where malicious SQL statements are inserted in the system and are executed to retrieve crucial information from the system like username/password, credit card info etc.

2. **Privilege Elevation**

In this type of ~~attack~~ attack, a user tries to elevate his/her access to higher level so that s/he can perform some unauthorized activities in the system.

3. **DDoS**

A Distributed Denial of Service (DDoS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources.

Events that could happen as a lack of Database security and integrity

1. Unauthorized or unintended activity or misuse by authorized database user.
2. Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal data, alteration of data.
3. Physical damage to database server.
4. Data corruption or data loss.

Why do we need security in database?

Database security refers to the collective measures used to protect and secure a database or database management software from illegitimate use and malicious threats and attacks.

Databases often hold the backbone of an organization, its transactions, customers, employee info, financial data for both the company and its customers and much more. Database security is more than just important. It is an essential part of any company. It prevents the compromise or loss of data contained in the database.

Different security levels to Protect a database

1. Physical

The sites containing the computer system must be secured against armed or hidden entry by intruders.

2. Human

Users must be authorized carefully to reduce the chance of any such user giving access to an intruder in exchange for bribes or other favors.

Nayeem Mahmood / 4AM / C161026

3. OS

No matter how secure the database system is, weakness in operating system security might serve as a means of unauthorized access to the database.

4. Network

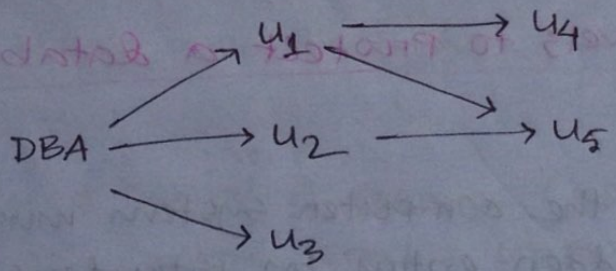
Since all database system allow remote access through terminals or networks, software level security within the network software is as important as physical security, both on the internet and in networks private to an enterprise.

5. DB System

some database system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries but may be forbidden to modify the data. It is responsibility of database system to ensure that these authorization restrictions are not violated.

security at all these levels must be maintained to ensure the complete security of database.

Authorization Grant Graph



Authorization graph prevents formation of grant privilege cycles with no path from the root.

For example, DBA grants authorization to user u_6 and user u_6 grants authorization to user u_7, also user u_7 grants authorization to u_6 forming a cycle which started with u_6 granting privilege and ended on u_6 with getting privilege. If DBA revokes authorization to u_6, then it must revoke grant u_6 to u_7 and from u_7 to u_6. Since there is no path from DBA to u_6 or u_7 anymore, this means that if a user has the same privilege as other users, then they keep it, otherwise all privileges dependent on the revoked one are also withdrawn.

Authentication

Database authentication is the process or act of confirming that a user who is attempting to log into database is authorized to do so and is only accorded the rights to perform activities that one has been authorized to do.

Authorization

Authorization is a security mechanism to determine user privileges or access levels related to system resources, including computer programs, files, services, data & application features. Authorization is normally proceeded by authentication for user identity verification.

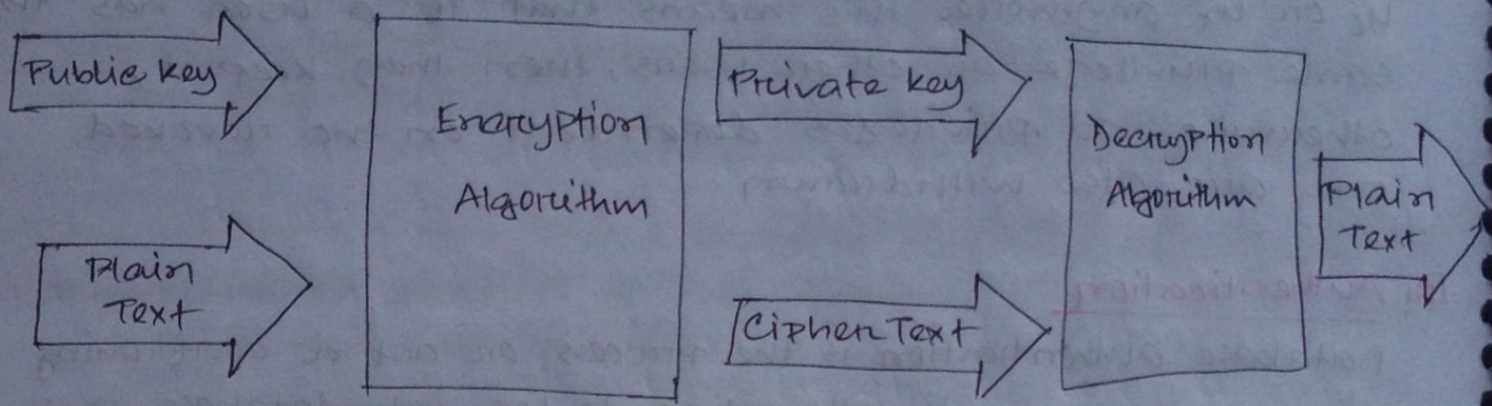
Encryption

Database encryption can generally be defined as a process that uses a certain algorithm to transform data stored in database into cipher text this is incomprehensible without being decrypted.

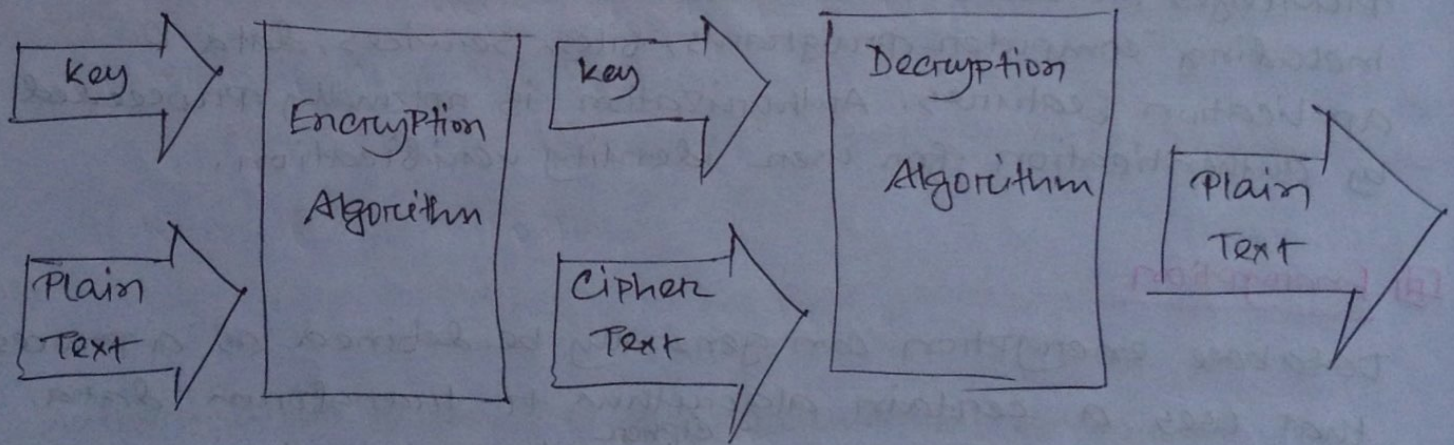
Decryption

It is the reverse process of encryption i.e. to gain back the original text/data from an encrypted cipher text.

Steps of Encryption-Decryption algorithm



Flow chart of Asymmetric Encryption & Decryption Process



Flow chart of Symmetric Encryption / Decryption Process

Functional Dependency

* Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as $X \rightarrow Y$, which specifies that Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated precisely with one Y value.

Inference rules of Functional Dependency

1. Reflexivity

   If B is a subset of A, then $A \rightarrow B$.

2. Augmentation

   If $A \rightarrow B$, then $AC \rightarrow BC$

3. Transitivity

   If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

4. Projectivity or Decomposition Rule

   If $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$

5. Union or Additive Rule

   If $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow BC$

6. Pseudo Transitive Rule

   If $A \rightarrow B$, $DB \rightarrow C$, then $DA \rightarrow C$

Nayeem Mahmood / 4AM / C161026

Transitive Dependency

Transitive dependency is a functional dependency which holds by virtue of transitivity. A transitive dependency can occur only in a relation that has 3 or more attributes. Let A, B, C designate 3 distinct attributes in the relation. Let us suppose all 3 of the following conditions hold:

1. $A \rightarrow B$
2. It is not the case that $B \rightarrow A$
3. $B \rightarrow C$

Then the functional dependency $A \rightarrow C$ is a transitive dependency.

Armstrong's Axioms

An important property of a functional dependency is Armstrong's ~~dependency~~ axioms, which are used in database normalization. In a relation R with 3 attributes A, B, C, Armstrong's axioms hold strong if the following conditions are satisfied:

1. Axiom of Reflexivity

   If B is a subset of A, then $A \rightarrow B$.

2. Axiom of Augmentation

   If $A \rightarrow B$, then $AC \rightarrow BC$.

3. Axiom of Transitivity

   If $A \rightarrow B$, and $B \rightarrow C$, then $A \rightarrow C$.

Armstrong's axioms are sound, because they don't generate any incorrect functional dependencies. They are complete, because for a given set F of functional dependencies, they allow us to generate all F^+.

Normalization

It is the process of reorganizing data in a database so that it meets two basic requirements –

1. There is no redundancy of data.
2. Data dependencies are logical.

Different Types of Normalization

1. First Normal Form (1NF)

For a table to be in 1NF, it should follow the following rules:

a) It should contain only single (atomic) valued attributes/columns.

b) Values stored in a column should be of the same domain.

c) All the columns in a table should have unique names.

d) The order in which data is stored, doesn't matter.

For example, lets consider a table which is not in 1NF;

Table: "student_info"

student_name	age	courses
Abdul	20	Physics
Karim	21	Math, Physics
Jabbar	20	Physics

Nayeem Mahmood / 4AM / C161026

To transform it into 1NF, we must separate data into multiple rows.

"student_info"

student_name	age	courses
Abdul	20	Physics
Karim	21	Math
Karim	21	Physics
Jabbar	20	Physics

2. Second Normal Form (2NF)

For a table to be in 2NF,

a) It should be in 1NF

b) It should not have partial dependencies.

If we transform the preceding table into 2NF, we'll get the followings:

"student_info"

student_name	age
Abdul	20
Karim	21
Jabbar	20

New "subject" table introduced for 2NF would be:

Student_name	courses
Abdul	Physics
Karim	Math
Karim	Physics
Jabbar	Physics

3. Third Normal Form (3NF)

A table is said to be in 3NF when

a) It is in 2NF
b) It doesn't have transitive dependency

For example, lets consider a table with the following fields :

student_info

stu_id	stu_name	DOB	street	city	state	zip

In this table, stu_id is primary key but street, city and state depend on zip. The dependency between zip and other fields is called transitive dependency. Hence to apply 3NF, we need to move the street, city and state to new table with zip as primary key.

New "student_info" table

stu_id	stu_name	DOB	zip

Newly introduced "address" table

zip	street	city	state

The advantages of removing transitive dependency:

1. Amount of data duplication is reduced.
2. Data integrity is achieved.

4. Boyce and Codd Normal Form (BCNF)

BCNF is a higher version of 3NF. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which doesn't have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

a) R must be in 3NF.

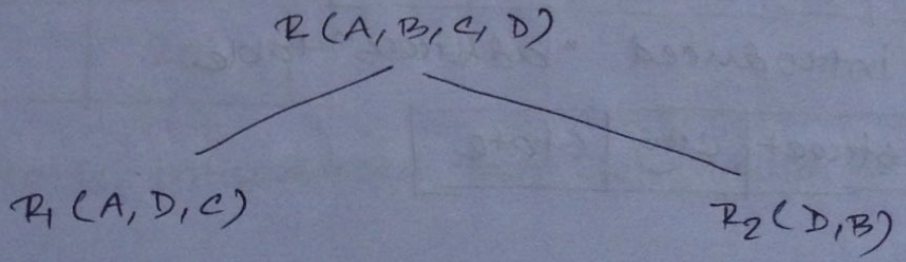
b) For each functional dependency $X \rightarrow Y$, X should be a super key.

Let us consider the following relationship: R(A, B, C, D) and following dependencies:

$A \rightarrow BCD$
$BC \rightarrow AD$
$D \rightarrow B$

Above relationship is already in 3NF. Keys are A & BC. Hence in functional dependency, $A \rightarrow BCD$, A is a super key. In $BC \rightarrow AD$, BC is also a key. But in $D \rightarrow B$, D is not a key.

Hence, we break our relationship R into two relationships R_1 & R_2.



Thus breaking into two tables, one with A, D, C and the other with D and B.

Lossless Join Decomposition

Let $r(R)$ be a relation schema and F be a set of functional dependencies on $r(R)$. Let R_1 and R_2 form a decomposition of R. We say that the decomposition is a lossless join decomposition if there is no loss of information by replacing $r(R)$ with two relation schemas $r_1(R_1)$ and $r_2(R_2)$. More precisely, we say the decomposition is lossless if, for all legal database instances, relation r contains the same set of tuples as the result of the following SQL query:

```
select *
from (select R1 from r)
     natural join
     (select R2 from r)
```

This is ~~stated~~ stated more compactly in the relational algebra as:

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

In other words, if we project r onto R_1 and R_2 and compute the natural join of the projection result, we get back exactly r.

A decomposition that is not a lossless decomposition is called a lossy decomposition.

Objectives of Normalization

1. To highlight constraints and dependency in the data.
2. To control data redundancy.
3. To reduce storage requirement.
4. To provide unique identification for records in a database.
5. To define efficient data structure.

Given the relation:

Book (Book_title, Author_name, Book_type, Listprice, Author_affil, Publisher)

The functional dependencies are

Book_title → Publisher, Book_type
Book_type → Listprice
Author_name → Author_affil

So, what normal form the relation in?

The key for this relation is (Book_title, Author_name). This relation is in 1NF and not in 2NF as no attributes are fully functional dependent on the key. It is also not in 3NF.

* 2NF decomposition

Book0 (Book_title, Author_name)
Book1 (Book_title, Publisher, Book_type, Listprice)
Book2 (Author_name, Author_affil)

This decomposition eliminates the partial dependencies.

*3NF decomposition

Book0 (Book_title, Author_name)

Book1-1 (Book_title, Publisher, Book_type)

Book1-2 (Book_type, Listprice)

Book2 (Author_name, Author_affil)

This decomposition eliminates the transitive dependency of Listprice.

How digital signature is used to verify authenticity of data?

An interesting application of public-key encryption is in digital signatures to verify authenticity of data; digital signatures play the electronic role of physical signatures on documents. The private-key is used to "sign", that is, encrypt, data, and the signed data can be made public. Anyone can verify the signature by decrypting the data using the public key, but no one could have generated the signed data without having the private key. Thus, we can authenticate the data.

Partial Dependency

Partial dependency means that a non-prime attribute is functionally dependent on part of a candidate key. A non-prime attribute is an attribute that's not part of any candidate key.

For example, lets start with R {ABCD} and the functional dependencies are AB → CD, A → C. The only candidate key of R is AB. C and D are non-prime attributes. C is functionally dependent on A. A is a part of candidate key. That's the partial dependency i.e. C is

partially dependent on AB.

Indexing

Indexing is a data structure technique to efficiently retrieve records from the database based on some attributes on which the indexing is to be done.

Ordered Index

Used to access data sorted by order of values.

Hash Index

Used to access data that is distributed uniformly across a range of buckets.

Which is better? Ordered or Hashed Indices?

Hashing is generally useful at retrieving records having a specified value of the key. So, in that case hashing is preferable.

But if range queries are common, ordered indices are preferable.

Difference between Primary and Secondary Index

Primary index is defined on an ordered data file. The data file is ordered on the basis of a key field. The key field is generally the primary key of the relation.

Secondary index may be generated from a field which is a candidate key and has a unique value in every record.

Nayeem Mahmood / 4AM / C161026

Necessity of Indexing

1. Indexing makes search queries faster.
2. Indices like primary key indices and unique indices help to avoid duplicate row data.
3. Full-text indices in MySQL, users have the opportunity to optimize searching against even large amounts of text located in any field indexed as such.

Transaction

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

ACID Properties

1. Atomicity

This property states that a transaction must be treated as an atomic unit, that is either all of its operations are executed or none. There must be no state in database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

2. Consistency

The database must remain in consistent state after any transaction is executed. No transaction should have any adverse effect on the data residing in the database.

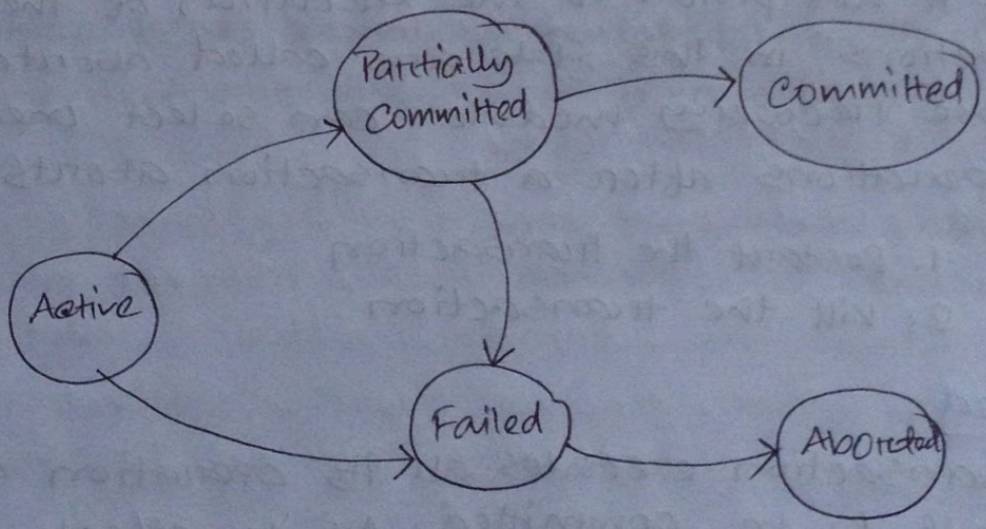
3. Isolation

In a database system where more than one transaction are being executed simultaneously and in parallel, the property of Isolation states that all the transaction will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

4. Durability

The database should be durable enough to hold all its updates even if the system fails or restarts. If a transaction updates a chunk of data and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written onto the disk, then that data will be updated once the system springs back into action.

State diagram of a transaction



1. Active

In this state, the transaction is being executed. This is the initial state of every transaction.

2. Partially Committed

When a transaction executes its final operation, it is said to be in partially committed state.

3. Failed

A transaction is said to be failed if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

4. Aborted

If any of the checks fails and transaction has reached the failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts:

1. Restart the transaction
2. Kill the transaction

5. Committed

If a transaction executes all its operation successfully, it is said to be committed. All its effects are now permanently established on the database system.

[A] How shadow copy technique helps ensuring "ACD" properties but not "I".

Shadow copy technique is a technique which is based on making copies of the database.

→ Let us assume that only one transaction is active at a time.

→ A pointer called db_pointer always points to the current consistent copy of the database.

→ All updates are made on a shadow copy of the database, and db_pointer is made to point to the updated shadow copy only after the transaction reaches partial commit and all updated pages have been flushed to disk.

→ In case transaction fails, old consistent copy pointed to by db_pointer can be used and the shadow copy can be deleted.

Those procedures ensure Atomicity, Consistency, Durability but not the Isolation property.

Causes of bucket overflow in hash file organization

1. Insufficient Buckets

$$n_B > n_r / f_r$$

where, $n_B \rightarrow$ number of buckets

$n_r \rightarrow$ number of records

$f_r \rightarrow$ number of records fit in a bucket.

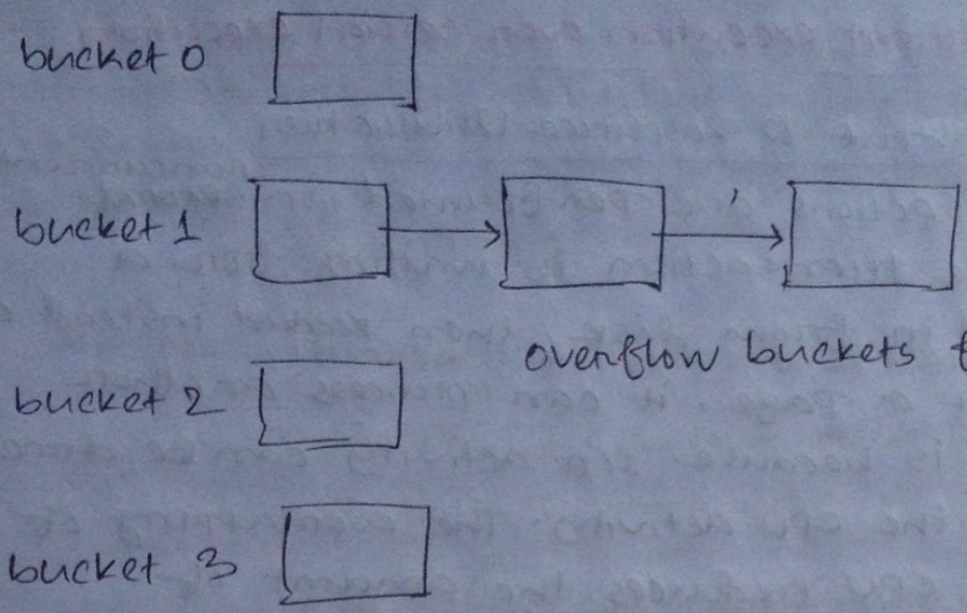
2. Skew

Some buckets may have more records than others, so a bucket may overflow while other buckets may have free space. This situation is called bucket skew. Skew can occur for two reasons.

1. Multiple records may have the same search key.
2. The chosen hash function may result in non-uniform distribution of search keys.

Reducing Bucket Overflow

We handle bucket overflow by using overflow buckets. If a record is inserted into a bucket while it is full, the system provides an overflow bucket. If it becomes full again, another overflow bucket is provided. All the overflow bucket are chained together in a linked list.



Distinction between closed and Open hashing

Open hashing may place keys with the same hash function value in different buckets. closed hashing always places such keys together in the same bucket. Thus in this case, different buckets can be of different sizes, though the implementation may be by linking together in fixed size buckets using overflow bucket chains. Deletion is difficult with open hashing as all the buckets may have to be inspected before we can ascertain that a key value has been deleted, whereas in closed hashing, only that bucket whose address is obtained by hashing the key value need to be inspected.

Deletions are more common in database and hence closed hashing is more appropriate for them. Static set of data lookups may be more efficient using open hashing. The symbol table of a compiler would be a good example.

Benefits of concurrent execution over serial execution

1. Improved Throughput & Resource Utilization

Assume that transactions are performed in concurrent ~~serial~~ order and active transaction is waiting for a page to be read in from disk, then ~~blocked~~ instead of CPU waiting for a page, it can process another transaction. This is because I/O activity can be done in parallel with the CPU activity. The overlapping of I/O activities of CPU reduces the amount of idle time of disks and processors and increase system's throughput. Correspondingly, the processor and disk utilization also increases.

2. Reduced Waiting Time

Interleaved execution of a short transaction with a long one usually allows the short one to complete quickly. In serial execution, a short transaction could get stuck behind a long one leading to unpredictable delay in response time or average time taken to complete a transaction.

Methoddogy of the timestamp based protocol

Timestamp ordering protocol works as follows:

1. If a transaction T_i issues a read(x) operation:

   a) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i needs to read a value of Q that was already overwritten. Hence the read operation is rejected and T_i is rolled back.

b) If $TS(T_i) \geq$ W-timestamp(Q), then the read operation is executed and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and $TS(T_i)$.

2. If a transaction T_i issues a write(x) operation:

a) If $TS(T_i) <$ R-timestamp(Q), then the value of Q that T_i is producing was needed previously.

b) If $TS(T_i) <$ W-timestamp(Q), then T_i is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls T_i back.

c) Otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$.

Replication

Replication is the process of copying and maintaining database objects in multiple databases that make up distributed database system.

Fragmentation

Fragmentation is the system of partitioning the relation into several fragments and storing each fragment at a different site.

Distributed Database

It is a database in which portions of the database are stored on multiple computers within a network. A centralized distributed dbms manages the database as if it were stored on the same computer. Domain Name Service (DNS) is a well known example of distributed database.

Pitfalls of lock based protocols

a. consider the partial schedule of figure 1 for transactions T_3 & T_4.

T_3	T_4
lock-X (B)	
read (B)	
B := B-50	
write (B)	
	lock-S(A)
	read (A)
	lock-S(B)
lock-X (A)	

Fig-1

b. Neither of the transactions can make progress.

c. Such a situation is called a deadlock. To resolve the deadlock, one of the transactions must be rolled back and its locks are released.

Nayeem Mahmood / 4AM / C161026

Working difference between two phase locking protocol and timestamp based protocol

Two phase locking Protocol

This protocol ensures conflicted serializable schedules.

Phase 1: Growing Phase

→ Transaction may obtain locks
→ Transaction may not release locks.

Phase 2: Shrinking Phase

→ Transaction may release locks.
→ Transaction may not obtain locks.

This protocol ensures serializability.

Timestamp based protocol

Each transaction is issued with a timestamp when it enters the system. If an old transaction T_i has time-stamp $TS(T_i)$, a new transaction T_j has timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$, the protocol manages concurrent execution such that the timestamps determine the serializability order. This protocol maintain for each data Q, two timestamp values:

1. W-timestamp (Q) is the largest timestamp of any transaction that executed write (Q) successfully.
2. R-timestamp (Q) is the largest timestamp of any transaction that executed read (Q) successfully.

Nayeem Mahmood / 4AM / C161026

Necessity of using checkpoints

1. A facility by which DBMS periodically refuses to ~~accept~~ accept any new transactions.
2. All transactions in progress are completed and the journal files are updated.
3. At this point, system is in a quiet state and the database & transaction logs are synchronized.
4. The dbms writes a special record (checkpoint record) to log file which is like a snapshot of the state of database.
5. The checkpoint record contains information necessary to restart the system.

Serializability

It is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operation.

Different types of prevention strategies of dead-lock

1. Wait-die scheme

This is a non-preemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if T_i has a smaller timestamp than that of T_j. Otherwise T_i is rolled back (dies). For example, suppose that transaction T_{22}, T_{23}, T_{24} have timestamps 5, 10, 15 respectively. If T_{22} requests a data item held by T_{23}, then T_{22} will wait. If T_{24} requests a data item held by T_{23}, T_{24} will be rolled back.

2. Wound-wait scheme

This is a preemptive technique. It is a counterpart to the wait-die scheme. When transaction T_i requests a data item held by T_j, T_i is allowed to wait only if T_i has longer timestamp than that of T_j. Otherwise T_j is rolled back. Returning to our example, if T_{22} requests a data item held by T_{23}, T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, T_{24} will wait.

Types of Serializability

1. View serializability

V.s. of a schedule is defined by a equivalence to a serial schedule with the same transaction such that respective transaction in the two schedules read and write the same data values. Every conflict serializable schedule is also view serializable. Below is a schedule which is view serializable but not conflict serializable:

T_1	T_2
read(Q)	
	write(Q)
write(Q)	

2. Conflict Serializabity

A schedule is said be conflict serializable when the schedule is conflict equivalent to one or more serial schedules. A schedule is conflict serializable iff there exists an acyclic precedence graph for the schedule.

T_1	T_2
read (A)	
write(A)	
	read(A)
	write(A)
read (B)	
write (B)	
	read (B)
	write(B)

Inconsistency in concurrent execution of transactions

T_1	T_2
read (A)	
A := A-50	
write (A)	
	read (A)
	temp := A * 0.1
	A := A - temp
read (B)	
B := B+50	
write (B)	
commit	
	read (B)
	B := B + temp
	write (B)
	commit

Fig - I

T_1	T_2
read (A)	
A := A-50	
	read (A)
	temp := A * 0.1
	A := A - temp
	write (A)
	read (B)
write (A)	
read (B)	
B := B+50	
write (B)	
commit	
	B := B + temp
	write (B)
	commit

Fig - II

Role of compatibility matrix in lock based protocol

1. A transaction may be granted a lock on item if the requested lock is compatible with the locks already held on the item by other transactions.

	S	X
S	true	false
X	false	false

2. Any number of transactions can hold shared locks on an item but if any transaction hold an exclusive lock on the item, no other transaction may hold any lock on them.

3. If a lock can't be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released.

4. A transaction can unlock a data item Q by the unlock(Q) instruction. Transaction Ti may unlock a data item that it held locked at some earlier point.