**Database:** Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc.

**Database Management System:** The software which is used to manage database is called Database Management System (DBMS).

**Data Definition:** It helps in creation, modification and removal of definitions that define the organization of data in database.

**DDL COMMANDS:**

* [**CREATE**](https://www.geeksforgeeks.org/sql-create/) **– is used to create the database or its objects (like table, index, function, views, store procedure and triggers).**
* [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) **– is used to delete objects from the database.**
* [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)**-is used to alter the structure of the database.**
* [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)**–is used to remove all records from a table, including all spaces allocated for the records are removed.**
* [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) **–is used to add comments to the data dictionary.**
* [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/) **–is used to rename an object existing in the database.**

**Drop vs Delete vs Truncate:**

**TRUNCATE(no rollback)**

* TRUNCATE is a DDL command
* TRUNCATE is executed using a table lock and whole table is locked for remove all records.
* We cannot use Where clause with TRUNCATE.
* TRUNCATE removes all rows from a table.
* Minimal logging in transaction log, so it is performance wise faster.
* Truncate uses the less transaction space than Delete statement.
* Truncate cannot be used with indexed views/ foreign key referencing it.

**DELETE**

* DELETE is a DML command.
* DELETE is executed using a row lock, each row in the table is locked for deletion.
* We can use where clause with DELETE to filter & delete specific records.
* The DELETE command is used to remove rows from a table based on WHERE condition.
* It maintain the log, so it slower than TRUNCATE.
* The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row.
* Delete uses the more transaction space than Truncate statement.
* Delete can be used with indexed views.

**DROP**

* The DROP command removes a table from the database.
* All the tables' rows, indexes and privileges will also be removed.
* No DML triggers will be fired.
* The operation cannot be rolled back.
* DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
* DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.

**DML COMMANDS:**

[**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/) **– is used to retrieve data from the a database.**

* [**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/) **– is used to insert data into a table.**
* [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) **– is used to update existing data within a table.**
* [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) **– is used to delete records from a database table.**

**DCL(Data Control Language) :**

* GRANT-gives user’s access privileges to database.
* REVOKE-withdraw user’s access privileges given by using the GRANT command.

**GRANT privileges\_names ON object TO user;**

GRANT SELECT, INSERT, DELETE, UPDATE ON Users TO 'Amit'@'localhost;

GRANT EXECUTE ON FUNCTION Calculatesalary TO 'Amit'@localhost';

**TCL(transaction Control Language) :** deals with the [transaction within the database](https://www.geeksforgeeks.org/sql-transactions/).

1. **COMMIT**– commits a Transaction.
2. [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)– rollbacks a transaction in case of any error occurs.
3. **SAVEPOINT**–sets a savepoint within a transaction.
4. **SET TRANSACTION**–specify characteristics for the transaction.

**Phases of database design**

**1)** Conceptual Design: ER model is used for conceptual design of database.

**2)** Logical Design:. ER diagram produced is used to convert the data into Relational Model.

3) Physical Design:data in relational model is implemented using DBMS like Oracle, DB2.

**RELATIONAL MODEL:**

**1)Super Key:** The set of attributes which can uniquely identify a tuple.

**2)Candidate Key:** The minimal set of attribute which can uniquely identify a tuple

**3)Primary Key:** There can be more than one candidate key in a relation out of which one can be a P\_key

**4)** **Alternate Key:** The candidate key other than primary key is called as alternate key

**5) Foreign Key:** If an attribute can only take the values which are present as values of some other attribute, it will be foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and corresponding attribute is called referenced attribute and corresponding attribute is called referencing attribute.

Referenced attribute of referencing attribute should be unique key

**Anomalies in Relational Model**

**Insertion anomaly:** If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation. For Example, If we try to insert a record in STUDENT\_COURSE with STUD\_NO =7, it will not allow.

**Deletion and Updation anomaly:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will not allow deleting the tuple from referenced relation. For Example, If we try to delete a record from STUDENT with STUD\_NO =1, it will not allow. To avoid this, following can be used in query:

* **ON DELETE/UPDATE SET NULL:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and set the value of referenced attribute to NULL.
* **ON DELETE/UPDATE CASCADE:** If a tuple is deleted or updated from referenced relation and referenced attribute value is used by referencing attribute in referencing relation, it will delete/update the tuple from referenced relation and referencing relation as well.

# Functional Dependency:

A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have same value for attribute B. For Example, in relation STUDENT shown in table 1, Functional Dependencies

STUD\_NO->STUD\_NAME, STUD\_NO->STUD\_ADDR **hold**

**PRIME/ NON-PRIME ATTRIBUTES**

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

# Database Normalization

the process of organizing the attributes of database to reduce or eliminate **data redundancy.**

**Problems because of data redundancy**

Data redundancy unnecessarily **increases size of database**.

**Inconsistency problems** also arise during insert, delete and update operations.(insertion, deletion and updation anomalies)

**Functional Dependency**

Functional Dependency is a constraint between two sets of attributes in a relation from a database.

A function dependency A → B mean for all instances of a particular value of A, there is same value of B.

**Trivial Functional Dependency**

X –> Y is trivial only when Y is subset of X.

Examples

ABC --> AB  
ABC --> A  
ABC --> ABC

**Non Trivial Functional Dependencies**

X –> Y is a non trivial functional dependencies when Y is not a subset of X.

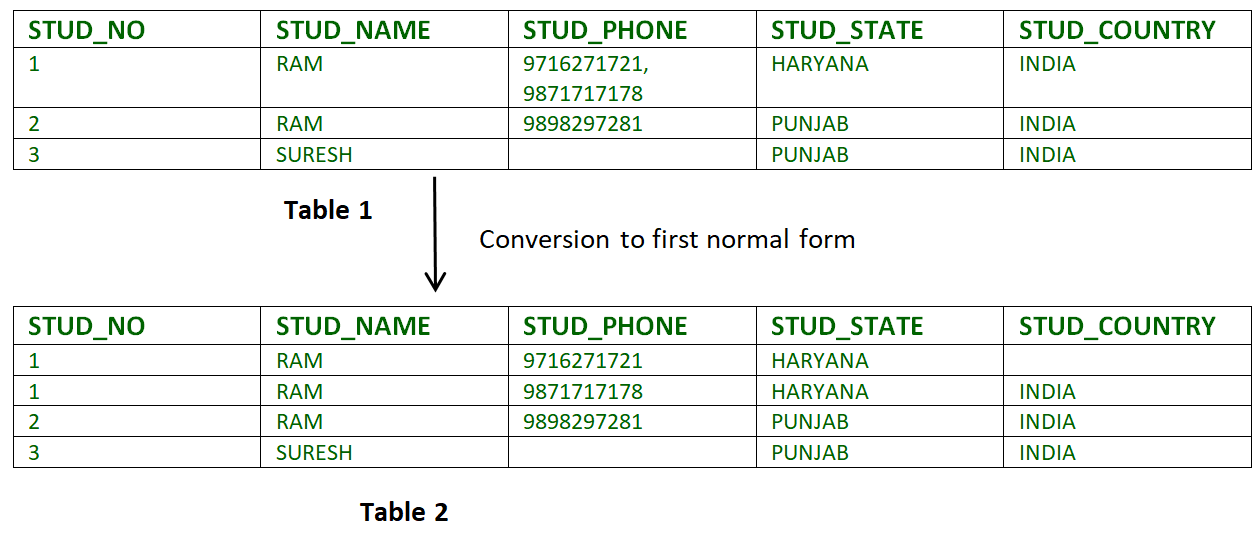
X –> Y is called completely non-trivial when X intersect Y is NULL.

Examples:

Id --> Name,   
Name --> DOB

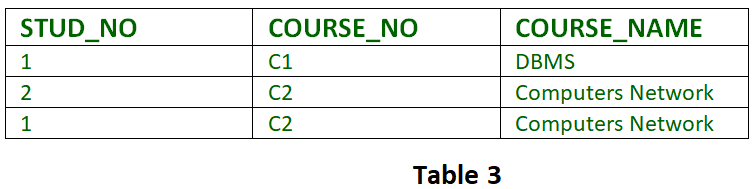
### 1. First Normal Form –

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

* **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD\_PHONE. Its decomposition into 1NF has been shown in table 2.
* 
* **Example 2 –**  
  ID   Name  Courses  
  ------------------  
  1    A  c1, c2  
  2    E  c3  
  3    M  C2, c2
* ID   Name  Course  
  ------------------  
  1    A   c1  
  1    A   c2  
  2    E   c3  
  3    M   c1  
  3    M   c2

### 2. Second Normal Form –

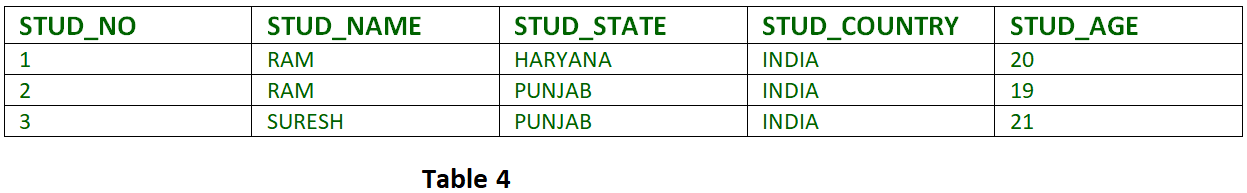
A relation is in 2NF iff it has **No Partial Dependency,** i.e.**, no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.**



**Partial Dependency –** If proper subset of candidate key determines non-prime attribute, it is called partial dependency.

* **Example 1 –** In relation STUDENT\_COURSE given in Table 3,
* FD set: {COURSE\_NO->COURSE\_NAME}  
  Candidate Key: {STUD\_NO, COURSE\_NO}
* In FD COURSE\_NO->COURSE\_NAME, COURSE\_NO (proper subset of candidate key) is determining COURSE\_NAME (non-prime attribute). Hence, it is partial dependency and relation is not in second normal form.
* To convert it to second normal form, we will decompose the relation STUDENT\_COURSE (STUD\_NO, COURSE\_NO, COURSE\_NAME) as :
* STUDENT\_COURSE (STUD\_NO, COURSE\_NO)  
  COURSE (COURSE\_NO, COURSE\_NAME)
* **Note – This decomposition will be lossless join decomposition as well as dependency preserving.**
* **Example 2 –** Consider following functional dependencies in relation  R (A, B , C, D )
* AB -> C  [A and B together determine C]  
  BC -> D  [B and C together determine D]
* In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn’t determine any non-prime attribute.

### 3. Third Normal Form –

* A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes is it is in second normal form.
* A relation is in 3NF iff **at least one of the following condition holds** in every non-trivial function dependency X –> Y
  + X is a super key.
  + Y is a prime attribute (each element of Y is part of some candidate key).
* 
* **Transitive dependency –** If A->B and B->C are two FDs then A->C is called transitive dependency.
  + **Example 1 –** In relation STUDENT given in Table 4,
  + FD set: {STUD\_NO -> STUD\_NAME, STUD\_NO -> STUD\_STATE, STUD\_NO -> STUD\_COUNTRY, STUD\_NO -> STUD\_AGE, STUD\_STATE -> STUD\_COUNTRY}
  + Candidate Key: {STUD\_NO}
  + For this relation in table 4, STUD\_NO -> STUD\_STATE and STUD\_STATE -> STUD\_COUNTRY are true. So STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY\_STUD\_AGE) as:
  + STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE)
  + STATE\_COUNTRY (STATE, COUNTRY)

BCNF?

**Dependency Preservation**

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

**(F1 ∪ F2 ∪ … ∪ Fm)+ = F+.**  
Consider a relation R  
R ---> F{...with some functional dependency(FD)....}  
  
R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then  
there can be three cases:  
  
**f1 U f2 = F** -----> Decomposition is dependency preserving.   
**f1 U f2** is a subset of F -----> Not Dependency preserving.  
**f1 U f2** is a super set of F -----> This case is not possible.

1. [Dependency Preserving Decomposition](https://www.geeksforgeeks.org/data-base-dependency-preserving-decomposition/)
2. [Lossless Join Decomposition](https://www.geeksforgeeks.org/database-management-system-lossless-decomposition/)
3. [LossLess Join and Dependency Preserving Decomposition](https://www.geeksforgeeks.org/lossless-join-and-dependency-preserving-decomposition/)

# DBMS | Data Replication

**Data Replication** is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Data replication encompasses duplication of transactions on an ongoing basis, so that the **replicate is in a consistently updated state** and synchronized with the source.However in data replication data is available at different locations, but a particular relation has to reside at only one location.

There can be full replication, in which the whole database is stored at every site. There can also be partial replication, in which some frequently used fragment of the database are replicated and others are not replicated.

**Types of Data Replication –**

1. **Transactional Replication –** In Transactional replication users receive full initial copies of the database and then receive updates as data changes. Data is copied in real time from the publisher to the receiving database(subscriber) in the same order as they occur with the publisher therefore in this type of replication, **transactional consistency is guaranteed.** Transactional replication is typically used in server-to-server environments. It does not simply copy the data changes, but rather consistently and accurately replicates each change.
2. **Snapshot Replication –** Snapshot replication distributes data exactly as it appears at a specific moment in time does not monitor for updates to the data. The entire snapshot is generated and sent to Users. **Snapshot replication is generally used when data changes are infrequent**. It is bit slower than transactional because on each attempt it moves multiple records from one end to the other end. Snapshot replication is a good way to perform initial synchronization between the publisher and the subscriber.
3. **Merge Replication –** Data from two or more databases is combined into a single database. Merge replication is the most complex type of replication because it allows both publisher and subscriber to independently make changes to the database. Merge replication is typically used in server-to-client environments. It allows changes to be sent from one publisher to multiple subscribers.

**ACID PROPERTIES:**

Atomicity:

i)  Each transaction is considered as one unit and either runs to completion or is not executed at all.

ii)  There is no midway i.e. transactions do not occur partially.

Consistency:

i)  correctness of a database

ii)  integrity constraints must be maintained so that the database is consistent before and after the transaction.

Isolation:

Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

**Durability:**

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even is system failure occurs.

These updates now become permanent and are stored in a non-volatile memory

**Transaction:**

logical sequence of database operations (reads/writes)

### Properties of a transaction: **ACID**

Can be achieved by serial schedule

Concurrency/ Interleaving transaction?

how can we improve resource utilization to efficiently process transactions while maintaining correct results?

Can lead to dirty read, lost update, unrepeatable read

A schedule, S, of n transactions T1, T2, … Tn is an ordering of the operations of the transactions

Defining Recoverability

• To satisfy durability, once a transaction is committed, it should never have to be rolled back

• A schedule that satisfies this criterion is recoverable

some recoverable schedules lead to cascading rollbacks:

where T1 needs to rollback because T2 did – This is expensive!

A schedule is cascadeless if every transaction reads only items that were written by committed transactions

The most restrictive type is strict: transactions can neither read nor write X until the last transaction that wrote X has committed/rolled back

• Serial scheduling is typically too slow for real-world use

• A schedule is serializable if it is “equivalent” to some serial schedule

• We will focus on one definition of how to compare two schedules, conflict serializability, which involves the idea of conflicting operations

Conflicting Operations

Two operations in a schedule are said to conflict if they satisfy all three of the following conditions… 1. They belong to different transactions 2. They access the same item (e.g. X) 3. At least one is a write operation

Read-Write

• r1(X), w2(X)

• r2(X), w1(X)

Write-Write

W1(x),W2(x)

**A schedule is conflict serializable if it is conflict equivalent to a serial schedule**

precedence/serialization graph without cycles

**INDEXING/PHYSICAL DESIGN:**

a data structure which is used to quickly locate and access the data in a database table.

a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed.

CREATE [UNIQUE] INDEX index\_name ON table\_name (c\_name1, …) [OPTIONS];

Ordering of columns is VERY important

An index answers certain kinds of questions very efficiently (depends upon type of index) – Equality:

fieldname=value

Range/ordering: fieldname>value

• Only index that maintains ordering (e.g. **tree-based**) Can be used for **WHERE clause, as well as JOIN and ORDER BY**

PROS:

Can make the difference between full table scan and log/constant lookup

CONS:

• Extra space – Linear with # rows

• Extra time – Creation (moderate) – Maintenance (can offset savings)

Choosing the Index(es) to Create

• Table size – Many rows = larger cost to table scan

• Data distribution (selectivity) – Fewer distinct values = higher likelihood needing to touch many rows, independent of index usage

• Index can lead to lots of IO/cache misses vs. sequential scan via clustered index

• Query vs. update load – Many updates = higher relative index maintenance cost – Analysis of frequent queries leads to choosing key attributes that get you the most bang for your buck

Index Types

• Clustered vs. Non-clustered

• Covering (w.r.t. a query)

• Balanced Trees (B+-Trees)

• Hash Tables

Clustered vs. Non-clustered

• Clustered: affects physical order on disk

– At most one per table (for some RDBMSs, PK)

– Fast when data accessed in order/reverse

• Non-clustered: induces logical ordering

– Arbitrary number per table

– Depending on the query/data, can lead to significant slowdown due to cache misses and frequent disk access

­­­

**Join:**

