SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational or distributed database.

**The Structure –**  
**SQL** databases are **table-based**.

**NoSQL** databases are either **key-value pairs, document-based,** graph databases or wide-column stores.

**Property followed –**  
SQL databases follow ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers **CAP** theorem (Consistency, Availability and Partition tolerance).

Some examples of SQL databases include PostgreSQL, MySQL, Oracle and Microsoft SQL Server. NoSQL database examples include **Redis**, RavenDB Cassandra, **MongoDB**, BigTable, HBase, Neo4j and CouchDB.

* SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable.

**Upsert**

There are also two methods "db.collection.update()" method and "db.collection.save()" method used for the same purpose. These methods add new documents through an operation called upsert.

Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

A functional dependency A → B is said to be partial if removal of any subset of A still able to recognize B uniquely.    
**Partial Dependency :**  
**Partial Dependency i**s a form of Functional dependency that holds on a set of attributes. It is about the complete dependency of a right hand side attribute on one of the left hand side attributes.

For example, in a functional dependency PQ → R, if either P alone or Q alone can uniquely identify R, then this is said to be Partial Functional Dependency. We read this as R is partially functionally dependent on P or R is partially functionally dependent on Q.

OR

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

**Types of Dependecy in DBMS:**

**Trivial Functional dependency:**

**Symbolically**: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

**For example**: Consider a table with two columns Student\_id and Student\_Name.

{Student\_Id, Student\_Name} -> Student\_Id is a trivial functional dependency as Student\_Id is a subset of {Student\_Id, Student\_Name}.  That makes sense because if we know the values of Student\_Id and Student\_Name then the value of Student\_Id can be uniquely determined.

Also, Student\_Id -> Student\_Id & Student\_Name -> Student\_Name are trivial dependencies too.

**Non trivial Functional dependency:**

If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

**For example**:   
An employee table with three attributes: emp\_id, emp\_name, emp\_address.  
The following functional dependencies are non-trivial:  
emp\_id -> emp\_name (emp\_name is not a subset of emp\_id)  
emp\_id -> emp\_address (emp\_address is not a subset of emp\_id)

On the other hand, the following dependencies are trivial:  
{emp\_id, emp\_name} -> emp\_name [emp\_name is a subset of {emp\_id, emp\_name}]

**Multivalued dependency:**

Here columns manuf\_year and color are independent of each other and dependent on bike\_model. In this case these two columns are said to be multivalued dependent on bike\_model. These dependencies can be represented like this:

bike\_model ->manuf\_year

bike\_model ->color

**What is Transitive Dependency**

When an indirect relationship causes functional dependency it is called Transitive Dependency.

If  P -> Q and Q -> R is true, then P-> R is a transitive dependency.

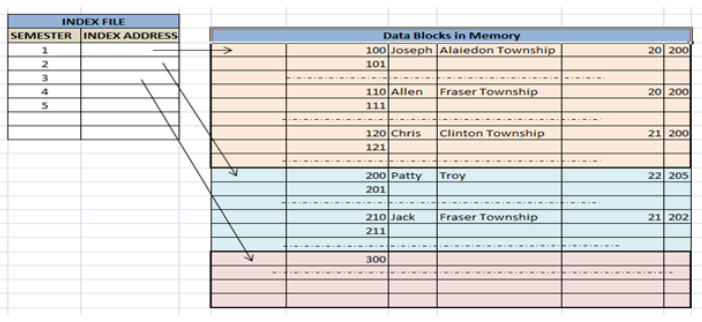
Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.

**Indexing:** Also see <https://www.guru99.com/indexing-in-database.html>

There are primarily three methods of indexing:

* **Clustered Indexing**
* **Non-Clustered or Secondary Indexing**
* **Multilevel Indexing**
* **Clustered Indexing**

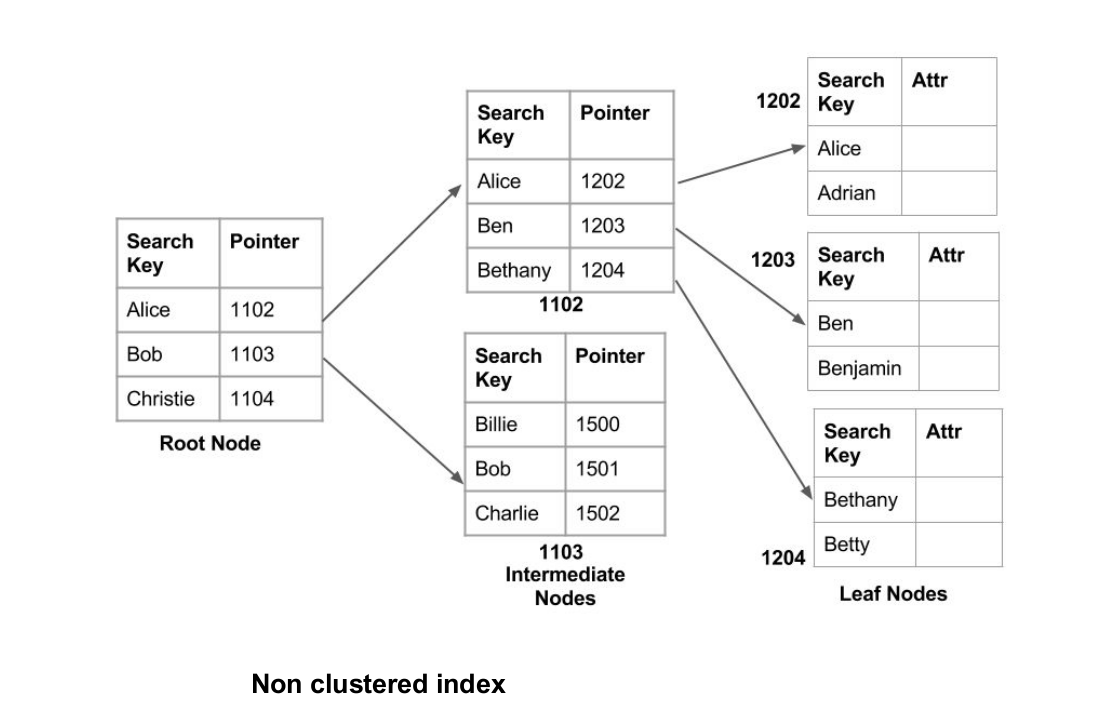
**Basically, records with similar characteristics are grouped together and indexes are created for these groups.**For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.



**Non-clustered or Secondary Indexing**

It gives us a **list of virtual pointers or references** to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes.

We can have only **dense** ordering in the non-clustered index as **sparse** ordering is not possible because data is not physically organized accordingly.  
It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer.



**Multilevel Indexing :**

The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.

