# DBMS Architecture

In the previous tutorials, we learned basics of DBMS. In this guide, we will see the **DBMS architecture**. Database management systems architecture will help us understand the components of database system and the relation among them.

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine. We will understand this communication with the help of diagrams.

## Types of DBMS Architecture

There are three types of DBMS architecture:

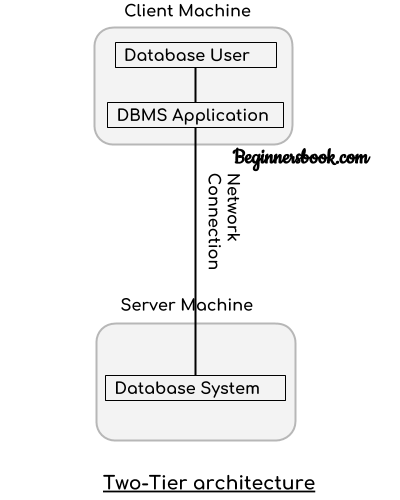
1. Single tier architecture  
2. Two tier architecture  
3. Three tier architecture

### 1. Single tier architecture

In this type of architecture, the database is readily available on the client machine, any request made by client doesn’t require a network connection to perform the action on the database.

For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

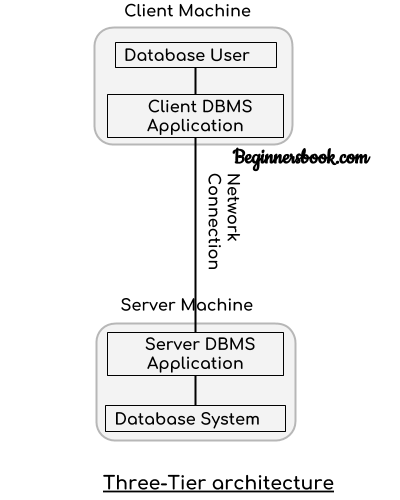
### 2. Two tier architecture



In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

### 3. Three tier architecture



In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn’t communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server.

**Candidate Key in DBMS**

A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.

**Definition of Candidate Key in DBMS**: A [super key](https://beginnersbook.com/2015/04/super-key-in-dbms/) with no redundant attribute is known as candidate key. Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is that the candidate key should not have any redundant attributes. That’s the reason they are also termed as minimal super key.

**Candidate Key Example**

Lets take an example of table “Employee”. This table has three attributes: Emp\_Id, Emp\_Number & Emp\_Name. Here Emp\_Id & Emp\_Number will be having unique values and Emp\_Name can have duplicate values as more than one employees can have same name.

Emp\_Id Emp\_Number Emp\_Name

------ ---------- --------

E01 2264 Steve

E22 2278 Ajeet

E23 2288 Chaitanya

E45 2290 Robert

How many super keys the above table can have?  
1. {Emp\_Id}  
2. {Emp\_Number}  
3. {Emp\_Id, Emp\_Number}  
4. {Emp\_Id, Emp\_Name}  
5. {Emp\_Id, Emp\_Number, Emp\_Name}  
6. {Emp\_Number, Emp\_Name}

Lets select the candidate keys from the above set of super keys.

1. {Emp\_Id} – No redundant attributes  
2. {Emp\_Number} – No redundant attributes  
3. {Emp\_Id, Emp\_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.  
4. {Emp\_Id, Emp\_Name} – Redundant attribute Emp\_Name.  
5. {Emp\_Id, Emp\_Number, Emp\_Name} – Redundant attributes. Emp\_Id or Emp\_Number alone are sufficient enough to uniquely identify a row of Employee table.  
6. {Emp\_Number, Emp\_Name} – Redundant attribute Emp\_Name.

The **candidate keys** we have selected are:  
{Emp\_Id}  
{Emp\_Number}

**Note**: A [primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/) is selected from the set of candidate keys. That means we can either have Emp\_Id or Emp\_Number as primary key. The decision is made by DBA (Database administrator)

# ACID properties in DBMS

To ensure the integrity of data during a transaction (**A transaction is a unit of program that updates various data items, read more about it here**), the database system maintains the following properties. These properties are widely known as ACID properties:

* **Atomicity**: This property ensures that either all the operations of a transaction reflect in database or none.
* **Consistency**: To preserve the consistency of database, the execution of transaction should take place in isolation (that means no other transaction should run concurrently when there is a transaction already running).
* **Isolation**: For every pair of transactions, one transaction should start execution only when the other finished execution. I have already discussed the example of Isolation in the Consistency property above.
* **Durability**: Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let’s discuss about anomalies first then we will discuss normal forms with examples.

**Anomalies in DBMS**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

**Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp\_id for storing employee’s id, emp\_name for storing employee’s name, emp\_address for storing employee’s address and emp\_dept for storing the department details in which the employee works. At some point of time the table looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_dept |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

The above table is not normalized. We will see the problems that we face when a table is not normalized.

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

**Normalization**

Here are the most commonly used normal forms:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)

**First normal form (1NF)**

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

**Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212  9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123  8123450987 |

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF**as the rule says “each attribute of a table must have atomic (single) values”, the emp\_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_address | emp\_mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

**Second normal form (2NF)**

A table is said to be in 2NF if both the following conditions hold:

* Table is in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

|  |  |  |
| --- | --- | --- |
| teacher\_id | subject | teacher\_age |
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys**: {teacher\_id, subject}  
**Non prime attribute**: teacher\_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher\_age is dependent on teacher\_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.

To make the table complies with 2NF we can break it in two tables like this:  
**teacher\_details table:**

|  |  |
| --- | --- |
| teacher\_id | teacher\_age |
| 111 | 38 |
| 222 | 38 |
| 333 | 40 |

**teacher\_subject table:**

|  |  |
| --- | --- |
| teacher\_id | subject |
| 111 | Maths |
| 111 | Physics |
| 222 | Biology |
| 333 | Physics |
| 333 | Chemistry |

Now the tables comply with Second normal form (2NF).

**Third Normal form (3NF)**

A table design is said to be in 3NF if both the following conditions hold:

* Table must be in 2NF
* [Transitive functional dependency](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [candidate key](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

* X is a [super key](https://beginnersbook.com/2015/04/super-key-in-dbms/) of table
* Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

**Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee\_details that looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_zip | emp\_state | emp\_city | emp\_district |
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

**Super keys**: {emp\_id}, {emp\_id, emp\_name}, {emp\_id, emp\_name, emp\_zip}…so on  
**Candidate Keys**: {emp\_id}  
**Non-prime attributes**: all attributes except emp\_id are non-prime as they are not part of any candidate keys.

Here, emp\_state, emp\_city & emp\_district dependent on emp\_zip. And, emp\_zip is dependent on emp\_id that makes non-prime attributes (emp\_state, emp\_city & emp\_district) transitively dependent on super key (emp\_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

**employee table:**

|  |  |  |
| --- | --- | --- |
| emp\_id | emp\_name | emp\_zip |
| 1001 | John | 282005 |
| 1002 | Ajeet | 222008 |
| 1006 | Lora | 282007 |
| 1101 | Lilly | 292008 |
| 1201 | Steve | 222999 |

**employee\_zip table:**

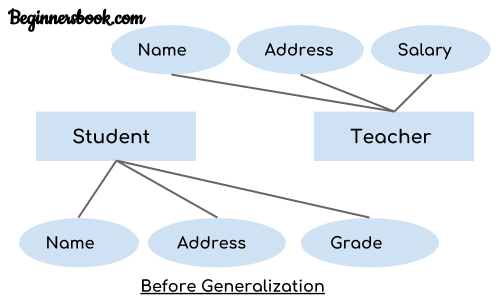
|  |  |  |  |
| --- | --- | --- | --- |
| emp\_zip | emp\_state | emp\_city | emp\_district |
| 282005 | UP | Agra | Dayal Bagh |
| 222008 | TN | Chennai | M-City |
| 282007 | TN | Chennai | Urrapakkam |
| 292008 | UK | Pauri | Bhagwan |
| 222999 | MP | Gwalior | Ratan |

# DBMS Generalization

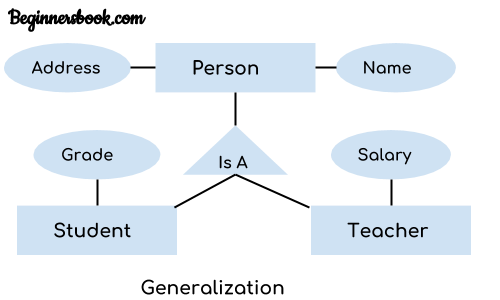
**Generalization** is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.

## Generalization Example

Lets say we have two entities Student and Teacher.  
Attributes of Entity Student are: Name, Address & Grade  
Attributes of Entity Teacher are: Name, Address & Salary

**The ER diagram before generalization looks like this:**  


These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Lets have a look at the ER model after generalization.

**The ER diagram after generalization:**  
We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following [ER diagram](https://beginnersbook.com/2015/04/e-r-model-in-dbms/) that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).  


**Note:**  
1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.  
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.