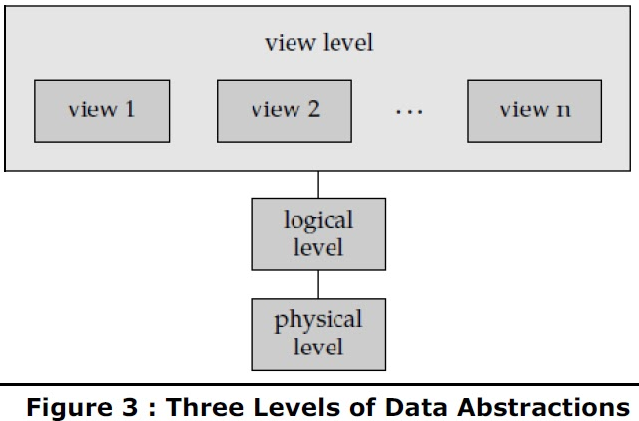
**Three levels of abstraction :-**

**Physical level:** This is the lowest level of data abstraction. It describes how data is actually stored in database.

**Logical level:** It describes what data is stored in database.

**View level:** Highest level of data abstraction. This level describes the user interaction with database system.

****

**Association:-**

Association is a simple relationship between two classes.

**Aggregation:-**

\*) Aggregation is a special type of Association. It is known as “Has-A” relationship.

\*) Aggregation is the process of compiling information on an object,

thereby abstracting a higher level object. So,an entity Person is derived by aggregating

the characteristics name, house\_no., city and social security number(SSN).

\*)A relationship represents a connection between two entity types that are conceptually at the same level. Sometimes you may want to model a ‘has-a’, ‘is-a’ or ‘is-part-of’ relationship, in which one entity represents a larger entity (the ‘whole’) that will consist of smaller entities (the ‘parts’). This special kind of relationship is termed as an aggregation.

**Composition**

Composition is a special type of Aggregation. It is known as “Is-A” relationship.

**Generalization**

Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass. Shared characteristics can be attributes, associations, or methods.

Generalization is a relationship between a Parent and its Derived class. It is nothing but inheritance.

**Specialization**

Specialization means creating new subclasses from an existing class.

**Specialization Hierarchy** - has the constraint that every subclass participates as a subclass in only one class/subclass relationship, i.e. that each subclass has only one parent. This results in a tree structure.

Eg: Tree Topology

**Specialization Lattice** - has the constraint that a subclass can be a subclass of more than one class/subclass relationship.

Eg : DAG topology

**CARDINALITY :-**

In tables, the number of rows (or tuples) is called the cardinality.

**ENTITY INTEGRITY:-**

Entity Integrity is the mechanism the system provides to maintain primary keys. The primary key serves as a unique identifier for rows in the table. Entity Integrity ensures two properties for primary keys: The primary key for a row is unique; it does not match the primary key of any other row in the table.

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

**External Schema** :-

Defines view of the database for particular Users

**CLOSURE OF FUNCTIONAL DEPENDENCY**

Ref: https://minigranth.in/dbms-tutorial/closure-of-functional-dependency

The Closure Of Functional Dependency means the complete set of all possible attributes that can be functionally derived from given functional dependency using the inference rules known as Armstrong’s Rules.

**Example-1 :** Consider the table student\_details having (Roll\_No, Name,Marks, Location) as the attributes and having two functional dependencies.

FD1 : Roll\_No -> Name, Marks

FD2 : Name -> Marks, Location

{Roll\_no}+ = {Roll\_No, Marks, Name, Location}

{Name}+ = {Name, Marks, Location}

**Example-2 :** Consider a relation R(A,B,C,D,E) having below mentioned functional dependencies.

FD1 : A -> BC

FD2 : C -> B

FD3 : D -> E

FD4 : E -> D

Now, we need to calculate the closure of attributes of the relation R. The closures will be:

{A}+ = {A, B, C}

{B}+ = {B}

{C}+ = {B, C}

{D}+ = {D, E}

{E}+ = {E}

**Calculating Candidate Key:-**

**Example-1:** Consider the relation R(A,B,C) with given functional dependencies :

FD1 : A -> B

FD2 : B -> C

Now, calculating the closure of the attributes as :

{A}+ = {A, B, C}

{B}+ = {B, C}

{C}+ = {C}

Clearly, “A” is the candidate key as, its closure contains all the attributes present in the relation “R”.

**Example-2 :** Consider another relation R(A, B, C, D, E) having the Functional dependencies :

FD1 : A -> BC

FD2 : C -> B

FD3 : D -> E

FD4 : E -> D

Now, calculating the closure of the attributes as :

{A}+ = {A, B, C}

{B}+ = {B}

{C}+ = {C, B}

{D}+ = {E, D}

{E}+ = {E, D}

In this case, a single attribute is unable to determine all the attribute on its own like in previous example. Here, we need to combine two or more attributes to determine the candidate keys.

{A, D}+ = {A, B, C, D, E}

{A, E}+ = {A, B, C, D, E}

Hence, "AD" and "AE" are the two possible keys of the given relation “R”. Any other combination other than these two would have acted as extraneous attributes.

**NOTE :** Any relation “R” can have either single or multiple candidate keys

**Key Definitions:-**

**Prime Attributes :** Attributes which are indispensable part of candidate keys. For example : “A, D, E” attributes are prime attributes in above example-2.

**Non-Prime Attributes :** Attributes other than prime attributes which does not take part in formation of candidate keys. For example.

**Extraneous Attributes :** Attributes which does not make any effect on removal from candidate key.

For example : Consider the relation R(A, B, C, D) with functional dependencies :

FD1 : A -> BC

FD2 : B -> C

FD3 : D -> C

Here, Candidate key can be “AD” only. Hence,

**Prime Attributes** : A, D.

**Non-Prime Attributes** : B, C

**Extraneous Attributes :** B, C(As if we add any of the to the candidate key, it will remain unaffected). Those attributes, which if removed does not affect closure of that set.

**CANONICAL COVER**

Ref <https://minigranth.in/dbms-tutorial/canonical-cover-of-functional-dependency>

**Canonical Cover Of Functional Dependency:**

In any relational model, there exists a set of functional dependencies. These functional dependencies when closely observed might contain redundant attributes. The ability of removing these redundant attributes without affecting the capabilities of the functional dependency is known as “Canonical Cover of Functional Dependency”.

Canonical cover of functional dependency is sometimes also referred to as “Minimal Cover”. Mc

**Canonical Cover Of Functional Dependency :** Example

Consider a relation R(A,B,C,D) having some attributes and below are mentioned functional dependencies.

FD1 : B -> A

FD2 : AD -> C

FD3 : C -> ABD

**Step-1 :** Decompose the functional dependencies using Decomposition rule(Armstrong’s Axiom) i.e. single attribute on right hand side.

FD1 : B -> A

FD2 : AD -> C

FD3 : C -> A

FD4 : C -> B

FD5 : C -> D

**Step-2 :** Remove extraneous attributes from LHS of functional dependencies by calculating the closure of FD’s having two or more attributes on LHS.

Here, only one FD has two or more attributes of LHS i.e. AD C.

{A}+ = {A}

{D}+ = {D}

In this case, attribute “A” can only determine “A” and “D” can only determine “D”. Hence, no extraneous attributes are present and the FD will remain the same and will not be removed.

**Step-3 : Remove FD’s having transitivity.**

FD1 : B -> A

FD2 : C -> A

FD3 : C -> B

FD4 : AD -> C

FD5 : C -> D

Above FD1, FD2 and FD3 are forming transitive pair. Hence, using Armstrong’s law of transitivity i.e. if X Y, Y X then X Z should be removed. Therefore we will have the following FD’s left :

FD1 : B -> A

FD2 : C -> B

FD3 : AD -> C

FD4 : C -> D

Also, FD2 & FD4 can be clubbed together now. Hence, the canonical cover of the relation R(A,B,C,D) will be:

Mc {R(ABCD)} = {B A , C BD, AD C}

**NORMAL FORM**

* A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is fully functionally dependent on the primary key
* It does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation.
* A relation schema R is in **third normal form** (**3NF**) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key.

Non key attributes are non-transitively dependent on the primary key.

* A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an FD X 🡪 A holds in R, then X is a superkey of R

A relation R is in Boyce-Codd normal form (BCNF) if and only if every determinant is a candidate key.

* A table is in 4NF if it is in BCNF and it it has no multi-valued dependencies.
* A table is in 5NF if it is in 4NF and it has no join dependency.

It is not always possible to satisfy all three design goals:

* BCNF.
* Lossless join.
* Dependency preservation.

**LOSSLESS JOIN :-**

For lossless join decomposition, these three conditions must hold true:

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

Att(R1) U Att(R2) = Att(R)

2. Intersection of Attributes of R1 and R2 must not be NULL.

Att(R1) ∩ Att(R2) ≠ Φ

3. Common attribute must be a key for at least one relation (R1 or R2)

Att(R1) ∩ Att(R2) ->Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)

**These 3 rules are called ARMSTRONGS AXIOMS**

**1. Reflexivity:**

If Y is a subset of X, then X->Y

Examples: AB->A, ABC->AB, etc

**2. Augmentation:**

If X ->Y, then XZ -> YZ

Examples: If A->B, then AC->BC

**3 Transitivity:**

If X ->Y, and Y -> Z, then X -> Z

**4. Union Rule:**

ifX->Y, X->Z, then X->YZ

**5. Pseudo-Transitivity Rule:**

If X->Y, WY->Z, then XW->Z

**6.Decomposition**

If{A → BC} and {A → B}, then {A → C}

**.Composition**

If X → Y and Z → W then {X, Z} → {Y, W}

**Trivial Functional Dependency:-**

If A holds B {A → B}, where A is a subset of B, then it is called a Trivial Functional Dependency. Trivial always holds Functional Dependency.

**Non-Trivial Functional Dependency:-**

If A holds B {A → B}, where B is not a subset A, then it is called as a Non-Trivial Functional Dependency.

**DEPENDENCY PRESERVATION**

**EXERCISE1:-**

Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.

A relation R is decomposed into -

R1 = (A, B, C) with FDs F1 = {A -> B, A -> C}, and

R2 = (C, D) with FDs F2 = {C -> D}.

F' = F1 ∪ F2 = {A -> B, A -> C, C -> D}

so, F' = F.

And so, F'+ = F+

Thus, the decomposition is dependency preserving decomposition.

**EXERCISE2:-**

If we decompose R(A, B, C, D) under F = {A → B, B → C}. into

R1(AB), R2(AC), and R3(AD), then:

S = {R1, R2, R3}.

F' = F1 ∪ F2 = {A → B, A → C, (omitting trivial FDs)}

F'+ ≠ F+

Therefore, S is not dependency preserving.

**EXERCISE3:-**

If we decompose R(A, B, C, D) under F = {A → B, B → C}. into

R1(AB), R2(BC), and R3(AD), then:

S = {R1, R2, R3}.

F = F1 ∪ F2 = {A → B, B → C, (omitting trivial FDs)}

F'+ = F+

Therefore, S is dependency preserving.

**DBMS**

**Dead Lock Prevention :-**

**Wait-Die Scheme:-**

When a transaction T1 requests data item X held by transaction T2, deadlock prevention protocol decides to allow T1 to wait or to roll-back based on the following conditions;

Condition 1: If timestamp of T1 is smaller than the timestamp of T2, ie, T1 started before T2, then allow T1 to wait for T2 to release lock on X.

Condition 2: If timestamp of T1 is larger than the timestamp of T2, i.e, T1 started after T2, then roll-back T1. [Also, let T1 starts again with the same timestamp and request X in a random amount of time.]

**Wound-Wait Scheme:-**

When a transaction T1 requests data item X held by transaction T2, deadlock prevention protocol decides to allow T1 to wait or to roll-back based on the following conditions;

Condition 1: If timestamp of T1 is larger than the timestamp of T2, ie, T1 started after T2, then allow T1 to wait for T2 to release lock on X.

Condition 2: If timestamp of T1 is smaller than the timestamp of T2, i.e, T1 started before T2, then roll-back T2. That is, the data item requested by T1 will be preempted from T2 and T2 is rolled-back.

**Dead Lock Avoidance :-**

1. Wait For Graph

This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction Ti requests for a lock on an item, say X, which is held by some other transaction Tj, a directed edge is created from Ti to Tj. If Tj releases item X, the edge between them is dropped and Ti locks the data item.

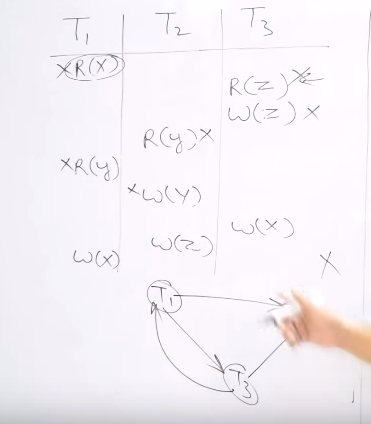
<https://exploredatabase.blogspot.in/search?q=thomas>

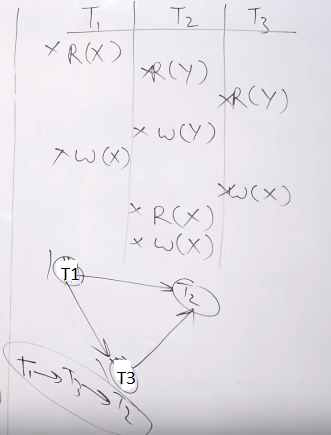
**CONCURRENCY CONTROL**

**CONFLICT SERIALIZABILITY**

Operations are conflict, if they satisfy all three of the following conditions :

1. They belong to different transactions
2. They access the same data item
3. At least one of the operation is a write operation.





Refer Video Tutorial and https://www.geeksforgeeks.org/precedence-graph-testing-conflict-serializability/

**RAID SERVER:-**

http://www.ecs.umass.edu/ece/koren/architecture/Raid/basicRAID.html

There are many types of RAID and some of the important ones are introduced below:

-Non-Redundant (RAID Level 0)

-Mirrored (RAID Level 1)

-Memory-Style(RAID Level 2)

-Bit-Interleaved Parity (RAID Level 3)

-Block-Interleaved Parity (RAID Level 4)

-Block-Interleaved Distributed-Parity (RAID Level 5)

-P+Q redundancy (RAID Level 6)

-Striped Mirrors (RAID Level 10)

**JOINS:-**



**DISTRIBUTED DATABASE MANAGEMENT SYSTEM**

**The three dimensions of distribution transparency are −**

**Location transparency:-**

Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user’s site.

**Fragmentation transparency:-**

Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites.

**Replication transparency:-**

Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists.

Whenever a user updates a data item, the update is reflected in all the copies of the table. However, this operation should not be known to the user.

**DATA WARE HOUSING**

In the OLAP world, there are mainly two different types: Multidimensional OLAP (MOLAP) and Relational OLAP (ROLAP). Hybrid OLAP (HOLAP) refers to technologies that combine MOLAP and ROLAP.

**MOLAP**

This is the more traditional way of OLAP analysis. In MOLAP, data is stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats.

**ROLAP**

This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement.

**HOLAP**

HOLAP technologies attempt to combine the advantages of MOLAP and ROLAP. For summary-type information, HOLAP leverages cube technology for faster performance. When detail information is needed,HOLAP can "drill through" from the cube into the underlying relational data.

**Different views regarding the design of a data warehouse:-**

**The top-down view** - This view allows the selection of relevant information needed for a data warehouse.

**The data source view** - This view presents the information being captured, stored, and managed by the operational system.

**The data warehouse view** - This view includes the fact tables and dimension tables. It represents the information stored inside the data warehouse.

**The business query view** - It is the view of the data from the viewpoint of the end-user.