Joins in DBMS and Types (Inner, Outer, Theta, Equi, Left, Right)

We can classify joins basically into two types

1. **INNER JOINS**: These joins are the one that has the tuples that satisfy some conditions and rest are discarded . Further they are classified as

- Theta join
- Equi join
- Natural join

2. **OUTER JOINS**: These have all the tuples from either or both the relations. Further they are classified as

- Left outer join
- Right outer join
- Full outer join

Let us now move on to the study the classified types with examples in detail.

INNER JOINS

1. **Theta join(θ)** – They have tuples from different relations if and only if they satisfy the theta condition, here the comparison operators (≤, ≥, <, >, =, ¬)come into picture. Let us consider simple example to understand in a much better way, suppose we want to buy a mobile and a laptop, based on our budget we have thought of buying both such that mobile price should be less than that of laptop. Look at the tables below,

MOBILE

MODEL	PRICE
Asus	10k
Samsung	20k
Iphone	50k

LAPTOP

MODEL	PRICE
Acer	20k
HP	35k
Apple	80k

Now, we have considered the condition as the cost of the mobile should be less than that of laptop so our resulting table will have only those tuples that satisfy this condition.

AFTER JOINS

Asus	Acer
Asus	HP
Asus	Apple
Samsung	HP
Samsung	Apple
Iphone	Apple

2. Equi join – As the name itself indicates, if suppose the join uses only the equality operator then it is called as equi join.

3. Natural join – It does not utilize any of the comparison operator. Here the condition is that the attributes should have same name and domain. There has to be at least one common attribute between between two relations. It forms the cartesian product of two arguments, performs selection forming equality on those attributes that appear in both relations and eliminates the duplicate attributes. Consider the example, where two tables namely employment table and department table have been shown. e

EMPLOYMENT

NAME	EMPID	DPT_NAME
A	11	Sales
B	12	Finance
C	13	Finance

DEPARTMENT

DPT_NAME	MANAGER
Finance	M1
Sales	M2

Looking at above tables we realize that they have a common attribute called DPT_NAME, thus after the natural join the table becomes
as

Name	EMPID	DPT_NAME	MANAGER
A	11	Sales	M2
B	12	Finance	M1
C	13	Finance	M1

Outer join

1. **Left outer join** – All the tuples of left table is displayed irrespective of whether it satisfies the matching conditions. Thus in the left all the tuples have been displayed but in the right only those are present that satisfy the matching conditions. For example consider below example of two tables – country table that has 3 records and state table that has 4 records. Country names are given the country_id that has to match with the country_id in the state table. India's state is Karnataka and Tamil Nadu, state of Pakistan is Islamabad but Nepal does not have state in the given table so right part will be null.

COUNTRY

COUNTRY_ID	COUNT_NME
1	India
2	Pakistan
4	Nepal

STATE

STATE_ID	COUNTRY_ID	STATE_NME
1	1	Karnataka
2	1	Tamil Nadu
3	2	Islamabad
4	NULL	Bangladesh

So for left join, left side table have all the values but right side only has those whose COUNTRY_ID has been matched.

COUNTRY_ID	COUNT_NME	STATE_ID	COUNTY_ID	STATE_NME
1	India	1	1	Karnataka
1	India	2	1	Tamil nadu
2	Pakistan	3	2	Islamabad
4	Nepal	NULL	NULL	NULL

LEFT OUTER JOIN

Bangladesh has not occurred since there is no match found.

2. Right outer join – All the tuples of right table are displayed irrespective of whether it satisfies the matching conditions or not.. Thus in the right, all the tuples have been displayed but in the left only those are present that satisfy the matching conditions. The previous example can be implemented here as well.

RIGHT OUTER JOIN

COUNTRY_ID	COUNT_NME	STATE_ID	COUNTY_ID	STATE_NME
1	India	1	1	Karnataka
1	India	2	1	Tamil nadu
2	Pakistan	3	2	Islamabad
NULL	NULL	4	NULL	Bangladesh

3. Full outer join– Tuples from both the relations takes part irrespective of whether it has the matching or non-matching conditions. Example is as shown

FULL OUTER JOIN

COUNTRY_ID	COUNT_NME	STATE_ID	COUNTY_ID	STATE_NME
1	India	1	1	Karnataka
1	India	2	1	Tamil nadu
2	Pakistan	3	2	Islamabad
4	Nepal	NULL	NULL	NULL
NULL	NULL	4	NULL	Bangladesh

Q: What are the difference Between Relational Algebra and Relational Calculus ?

BASIS FOR COMPARISON	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Claculus is Declarative language.
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.

Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Claculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.

BASIS FOR COMPARISON	**Sequential**	**Hash**
Method of storing	Stored as they come or sorted as they come	Stored at the hash address generated
Types	Pile file and sorted file Method	Static and dynamic hashing
Design	Simple Design	Medium
Storage Cost	Cheap (magnetic tapes)	Medium
Advantage	Fast and efficient when there is large volumes of data, Report generation, statistical calculations etc	Faster Access No Need to Sort Handles multiple transactions Suitable for Online transactions

Disadvantage	Sorting of data each time for insert/delete/ update takes time and makes system slow.	Accidental Deletion or updation of Data Use of Memory is inefficient Searching range of data, partial data, non-hash key column, searching single hash column when multiple hash keys present or frequently updated column as hash key are inefficient.

Q: Discuss different types of anomalies.

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.

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

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 –

ID Name Courses

1	A	c1, c2
2	E	c3
3	M	C2, c3

In the above table Course is a multi valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

ID	Name	Course
1	A	c1
1	A	c2
2	E	c3
3	M	c1
3	M	c2

2. Second Normal Form –To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. 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.

Example – 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

1. X is a super key.
2. 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 – Consider relation R(A, B, C, D, E)
A -> BC,
CD -> E,
B -> D,
E -> A
All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) –A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

Example –For example consider relation R(A, B, C)
A -> BC,
B ->
A and B both are super keys so above relation is in BCNF.

Q: What is self join ?

Self join is a kind of join in that each row combines with itself and each every other row in the table.

1. **SELECT column-names**
2. **FROM table-name T1 JOIN table-name T2**
3. **WHERE condition**

Q: Why "bcnf is stronger than 3nf" explain with the help of an example.

A relation R is in 3NF if and only if every dependency A->B satisfied by R meets at least ONE of the following criteria: 1. A->B is trivial (i.e. B is a subset of A) 2. A is a superkey 3. B is a subset of a candidate key BCNF doesn't permit the third of these options. Therefore BCNF is said to be stronger than 3NF because 3NF permits some dependencies which BCNF does not.

Why is SQL called a structured and a non-procedural language?

SQL is a declarative language in which the expected result or operation is given without the specific details about how to accomplish the task. The steps required to execute SQL statements are handled transparently by the SQL database. **Sometimes SQL is characterized as *non-procedural*** because procedural languages generally require the details of the operations to be specified, such as opening and closing tables, loading and searching indexes, or flushing buffers

and writing data to filesystems. Therefore, SQL is considered to be designed at a higher conceptual level of operation than procedural languages because the lower level logical and physical operations aren't specified and are determined by the SQL engine or server process that executes it.

Q: What are the advantages of hash file organization ?

- Records need not be sorted after any of the transaction. Hence the effort of sorting is reduced in this method.
- Since block address is known by hash function, accessing any record is very faster. Similarly updating or deleting a record is also very quick.
- This method can handle multiple transactions as each record is independent of other. i.e.; since there is no dependency on storage location for each record, multiple records can be accessed at the same time.
- It is suitable for online transaction systems like online banking, ticket booking system etc.

What is the difference between Tuple relational and Domain relational?

There is a very big conceptual difference between the two.In case of tuple relational calculus ,you operate on each tuples but in case of domain relational calculus ,you deal with each column or attribute.Both methods can be used to get any result

Q: Explain transaction with example.

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

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

Open_Account(A)

Old_Balance = A.balance

New_Balance = Old_Balance - 500

A.balance = New_Balance

Close_Account(A)

B’s Account

Open_Account(B)

Old_Balance = B.balance

New_Balance = Old_Balance + 500

B.balance = New_Balance

Close_Account(B)

Is SQL relationally complete?
Note: To prove that SQL is relationally complete, you need to show that for every expression of the relational algebra, there exists a semantically equivalent expression in SQL.

Proof by Contradiction:
Alternatively, to prove it is not, you need to show there exists at least one expression of the relational algebra for which no such SQL equivalent exists.

In order to show that SQL is relationally complete, it is sufficient to show that

a. there exist SQL expressions for each of the algebraic operators restrict, project, product, union, and difference (all of the other algebraic operators discussed can be defined in terms of these five), and
b. the operands to those SQL expressions can be arbitrarily complex SQL expressions in turn.

Attempt: First of all, as we know, SQL effectively does support the relational algebra RENAME operator, thanks to the availability of the optional AS specification on items in the SELECT clause. We can therefore ensure that tables do all have proper column names, and hence that the operands to product, union, and difference in particular satisfy the requirements of the algebra with respect to column naming.

SQL fails to support projection on no columns at all, because it does not support empty comma lists in the SELECT clause.
As a consequence, it does not support TABLE_DEE or TABLE_DUM, and therefore it is not relationally complete after all.
However, SQL is "nearly" relationally complete.

What is data model ? classify it.

- A data model is an idea which describes how the data can be represented and accessed from software system after its complete implementation.
- It is a simple abstraction of complex real world data gathering environment.
- It defines data elements and relationships among various data elements for a specified system.
- The main purpose of data model is to give an idea that how final system or software will look like after development is completed.

1. Hierarchical Model

- Hierarchical model was developed by IBM and North American Rockwell known as Information Management System.
- It represents the data in a hierarchical tree structure.
- This model is the first DBMS model.
- In this model, the data is sorted hierarchically.
- It uses pointer to navigate between the stored data.

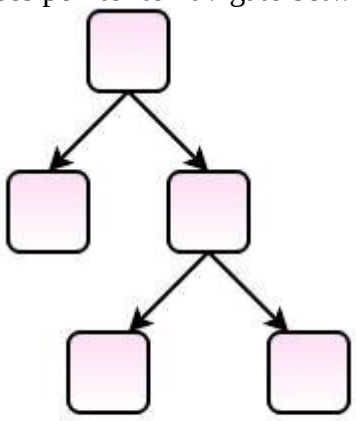


Fig. Hierarchical Model

2. Relational Model

- Relational model is based on first-order predicate logic.
- This model was first proposed by E. F. Codd.
- It represents data as relations or tables.
- Relational database simplifies the database structure by making use of tables and columns.

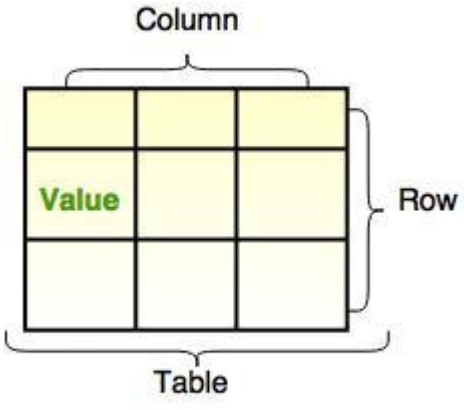


Fig. Relational Model

3. Network Database Model

- Network Database Model is same like Hierarchical Model, but the only difference is that it allows a record to have more than one parent.
- In this model, there is no need of parent to child association like the hierarchical model.
- It replaces the hierarchical tree with a graph.
- It represents the data as record types and one-to-many relationship.
- This model is easy to design and understand.

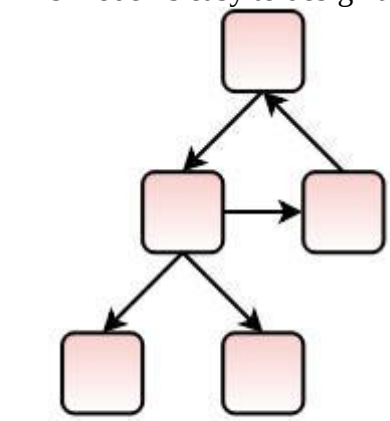


Fig. Network Model

4. Entity Relationship Model

- Entity Relationship Model is a high-level data model.
- It was developed by Chen in 1976.
- This model is useful in developing a conceptual design for the database.
- It is very simple and easy to design logical view of data.
- The developer can easily understand the system by looking at an ER model constructed.

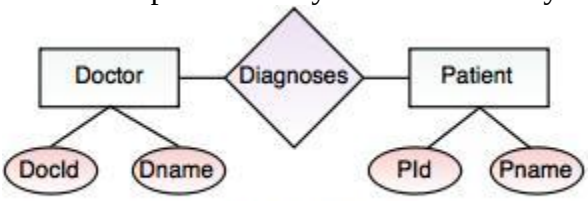
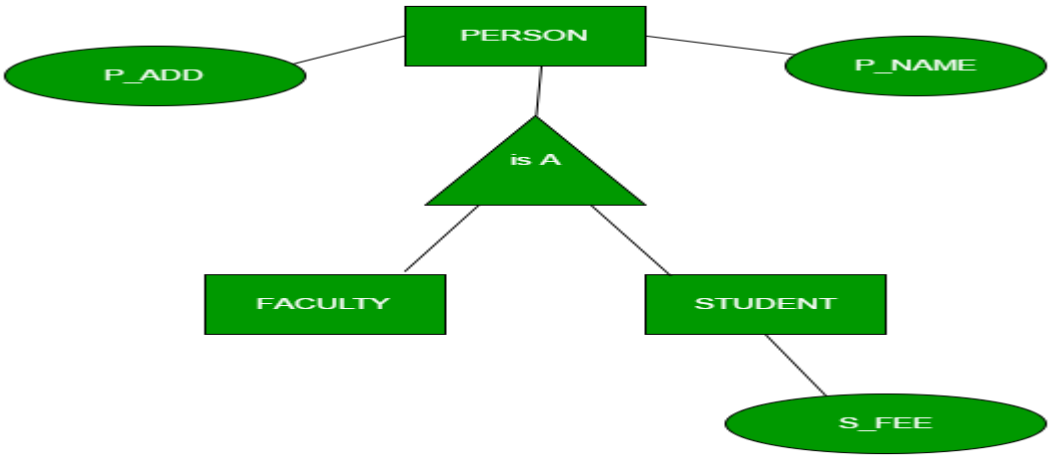


Fig. ER Model

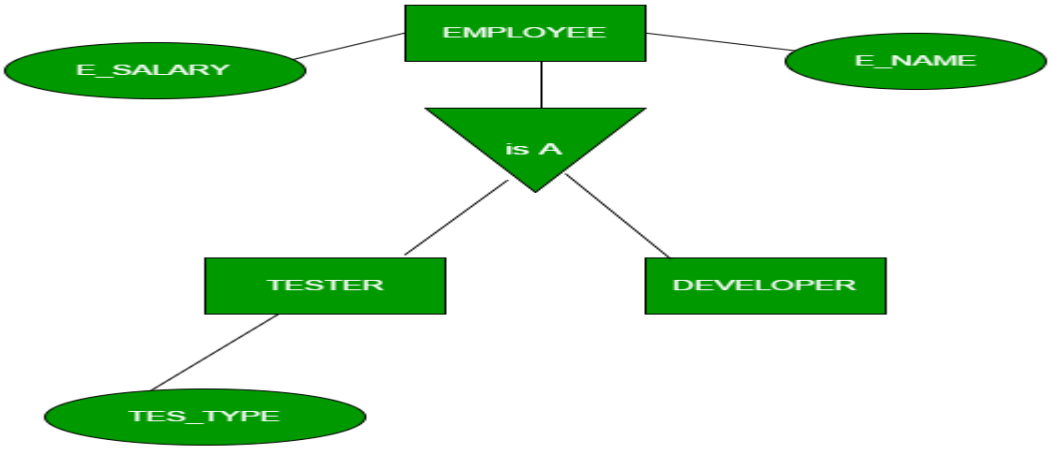
In this diagram,

- **Rectangle** represents the entities. Eg. Doctor and Patient.
- **Ellipse** represents the attributes. Eg. DocId, Dname, PId, Pname. Attribute describes each entity becomes a major part of the data stored in the database.
- **Diamond** represents the relationship in ER diagrams. Eg. Doctor diagnoses the Patient.

Generalization –
Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



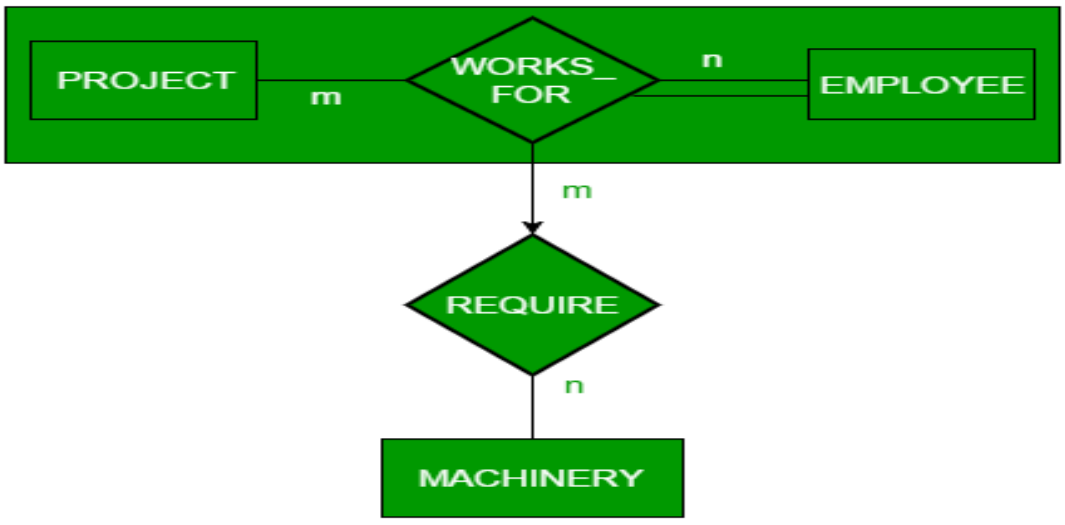
Specialization –
In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).



Specialization

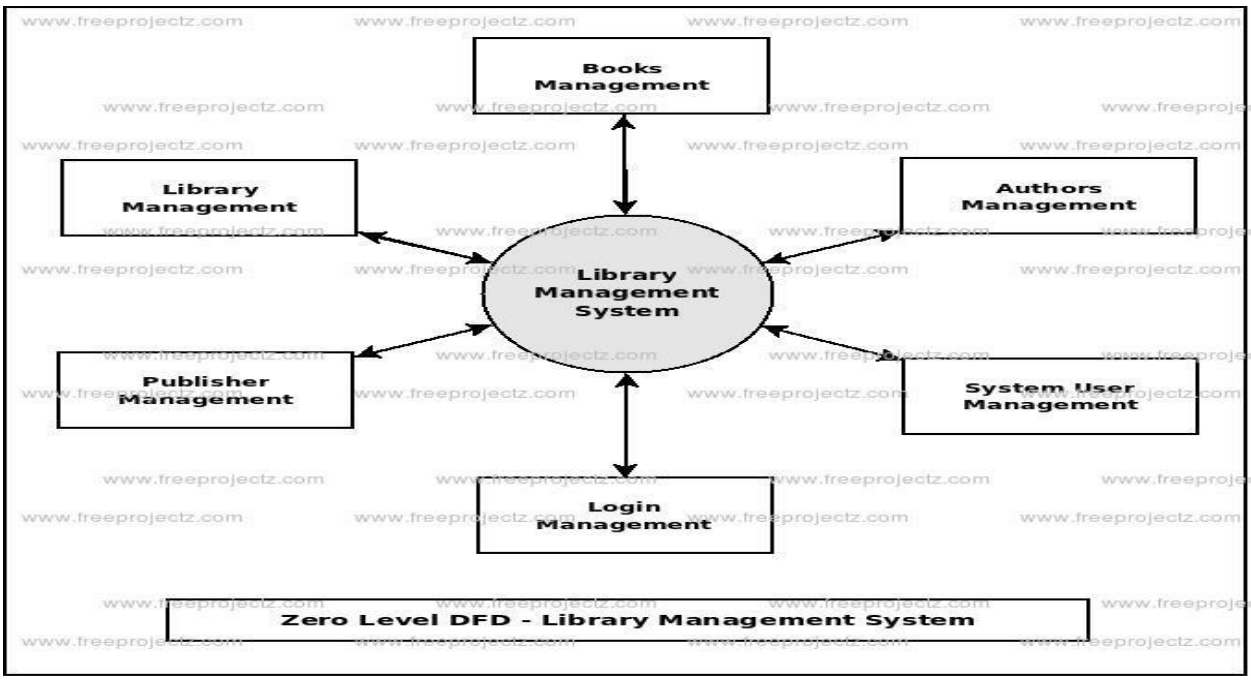
Aggregation –

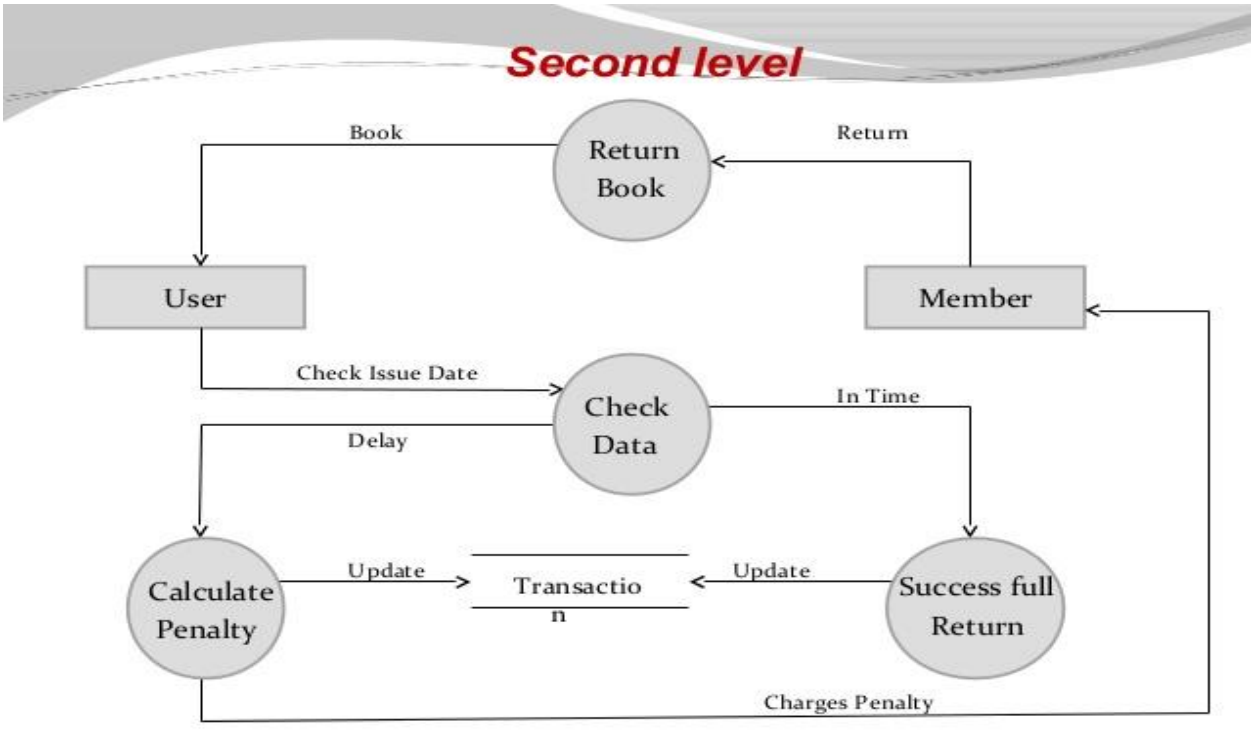
An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.



Aggregation

Q: Draw level 0draw level 0 and level1 dfd for library management .


Books Management
Library Management
Authors Management
Library Management System
Publisher Management
System User Management
Login Management
Zero Level DFD - Library Management System


Second level
Book
Return Book
Return
User
Member
Check Issue Date
Check Data
In Time
Delay
Calculate Penalty
Update
Transactio n
Update
Success full Return
Charges Penalty

Lossless Decomposition :

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example:

<EmpInfo>

Emp_ID	**Emp_Name**	**Emp_Age**	**Emp_Location**	**Dept_ID**	**Dept_Name**
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	**Emp_Name**	**Emp_Age**	**Emp_Location**
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	**Emp_ID**	**Dept_Name**
Dpt1	E001	Operations
Dpt2	E002	HR

Dpt3	E003	Finance