DBMS(Database Management System)

Other Notes Links:

- DBMS
- CN
- OS

Index

Database

Index

- It is an organized collection of interrelated information that show some aspects of the real world.
- Database System is designed to make the information easily accessible and manageable.
- Database is a collection of data in some organised manner,such that the user can easily access,manage,upload the data.

DBMS

Index

- Database Management System.

- It is a software which is used to manage the data easily. It provides an interface to perform various operations like creating database , creating table in it , insertion, updation , deletion of data and lot more.
- It provides security to the data and maintain consistency.
- it is the collection of programs or a software through which the user can insert,modify and delete the data into the database.

> *DBMS allow users to maintain the following tasks.*

1. **Data Defination** : It is used to create update delete the defination that defines the organization of data in databse.
2. **Data Updation** : It is used to insert , delete , update the actual data in the database.
3. **Data Retrival** : It is used to retrive the data from the databse to use in various applications.
4. **User Administration** : It is used to register and monitoring users and data. It enforces the data security and monitoring recovery information curroupted by unexpected failures.

> *Characteristics :*

- Provide clear and logical view of process that manipulates the data
- Contain Automatic backup and recovery procedures
- Provide Security of the data
- Reduce complex relationship between the data
- It can view the data in various viewpoints according to the requirement of the users.
- It contains ACID properties which maintain the data in healty state in case of failure.
- ACID - Automacity,Consistency,Isolation, Durability

> *Advantages :*

- **Data Sharing** : Autorised users of an organization can share tge data among various users.
- **Easily Maintenance** : It can be easily maintained due to centralized nature of the database.
- **Backup and Recovery** : It provides backup and recovery features which is used during the hardware and software failures.
- It Provides multiple user interface like GUI , API (graphical user interface and application program interface)
- **In short (Reduces Redundancy,Unautharised acess is restricted,Provide multiple user interface,Provide backup and recovery)**

> *Disadvantages :*

- **Cost of Hardware and Software** : Reuqire high speed of data processor and large memory size to run DBMS efficiently.
- **Size** : It occupies Large space of disks and large memory to run them efficiently.
- **Higher Impact of Failure** : Failure impact the database highly because in most of the organization, all the data is placed in a single darabase and if database is corruped then data may lost forever.

ER-Diagram

Index *ER-Diagram* : It stands for Entity Relationship Diagram.

- It is a conceptual model that gives graphical representation of the logical structure of the database.
- It shows all the constraints and relationships among the entities.
- An ER-Diagram has three main components : 1) Entity 2) Relationships 3) Attributes

> *Entity: An object which has a physical existence is entity. For E.g. : Student is an entity*

> *Attribute : It is used to describe the property of an entity, for e.g. Student has attributes like rollnumber , name , dob , address etc.*

> *Relationship : It is used to describe the connection between entities.*

`Enrolled in` is a relationship that exists between entities Student and Course.

Entity Sets and Relationship Set

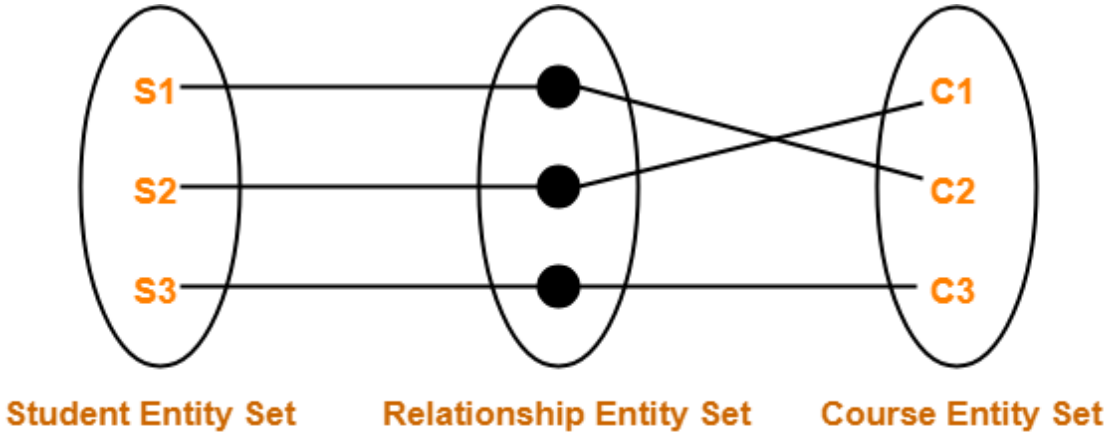
Index

Entity Set - set of same type of entities

- **Strong entity set** : 1) it is an entity set that contains sufficient attributes that can uniquely identify all its entities. 2) In this , primary key exists for a strong entity set. 3) it is represented by underline
- **Weak entity set** : 1) an entity set that does not contain sufficient attribute that can uniquely identify all its entities. 2) a primary key doesnot exist for a weak entity set 3) it contains a partial key called discriminator 4) discriminator can identify a group of entites from the entity sets. 5) discriminator is represented by underlining with a dashed line.

Relationship Set : A relationship set is a set of relationships of same type.

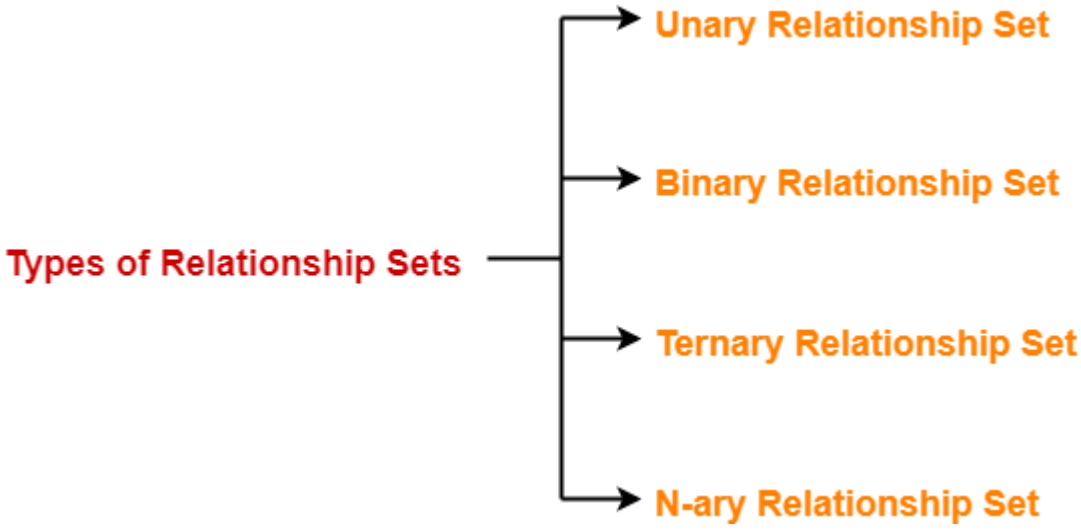
-Example: Set representation of above ER diagram is-



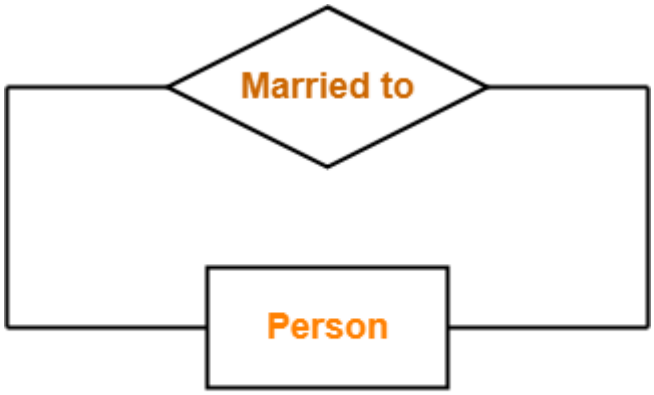
Set Representation of ER Diagram

> *Degree of a Relationship Set- # Degree of a relationship set = Number of entity sets participating in a relationship set*

[Types of Relationship Sets] -



1. **Unary Relationship Set** : It is a relationship where only one entity set participate in a relationship set. Example: One person is married to only one



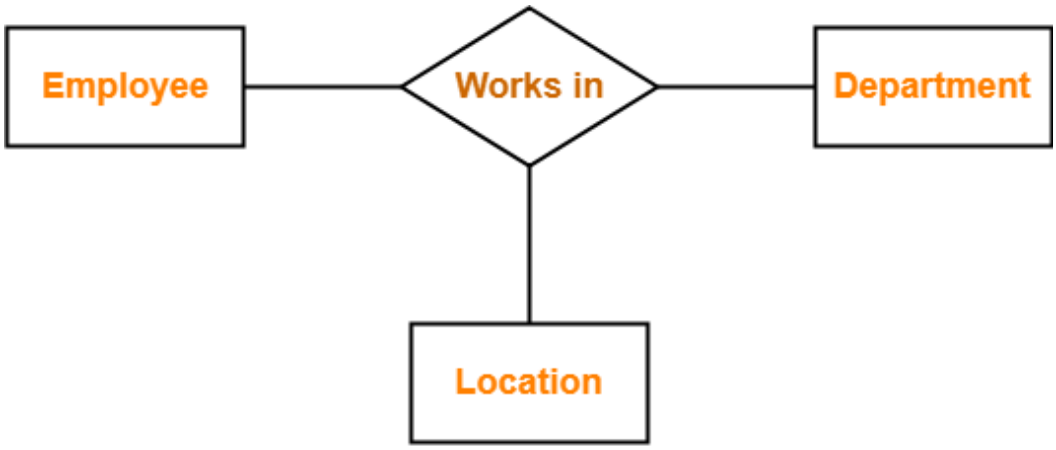
Unary Relationship Set

person

2. **Binary Relationship Set** : It is a relationship where only two entity sets participates in a relationship set. Example: Student is enrolled in a Course.

Binary Relationship Set

3. **Ternary Relationship Set** : It is a relationship set where only three entity set participates in a relationship set.



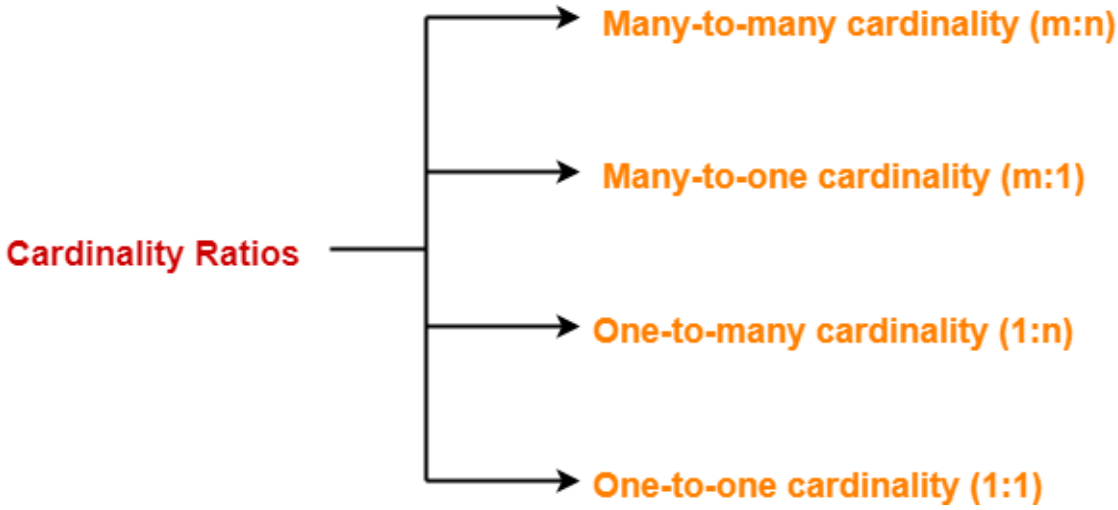
Ternary Relationship Set

4. **N-ary Relationship Set** : where `n` entity sets particitaes in relationship set.

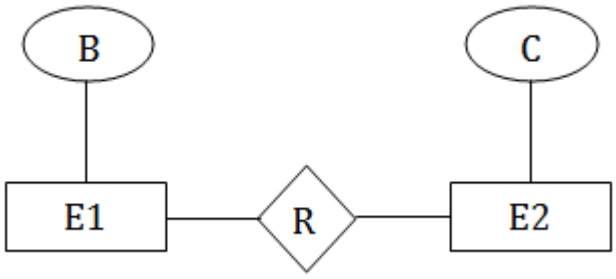
Cardinality Constraint

Index

- The number of entities to which another entity can be associated through a relationship.
- We have four types of cardinality constraints.



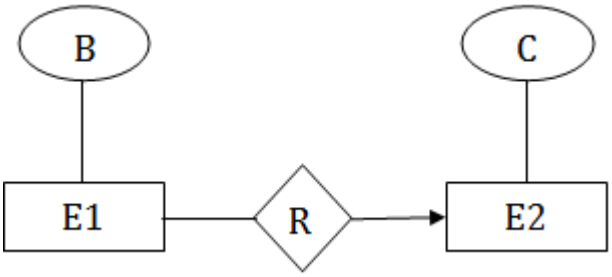
- **Many-to-Many Cardinality**- In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1. (Association - to make a connection between people or things in your mind)



Example: Here, 1. One student can enroll in any number (zero or more) of courses. 2. One course can be enrolled by any number (zero or more) of students.

Many to Many Relationship

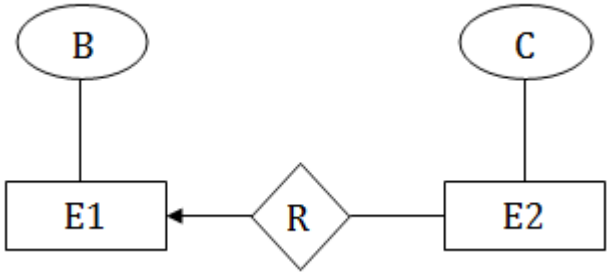
- **Many-to-One Cardinality**- In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



Example: Here, 1. One student can enroll in at most one course. 2. One course can be enrolled by any number (zero or more) of students.

Many to One Relationship

- **One-to-Many Cardinality**- In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.

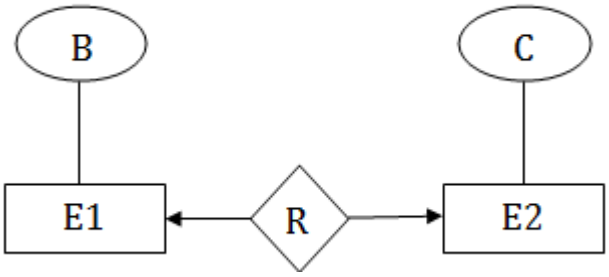


Example: Here,

1. One student can enroll in any number (zero or more) of courses.
2. One course can be enrolled by at most one student.

One to Many Relationship

- **One-to-One Cardinality**- In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



Example: Here, 1. One student can enroll in at most one course. 2. One course can be enrolled by at most one student.

One to One Relationship

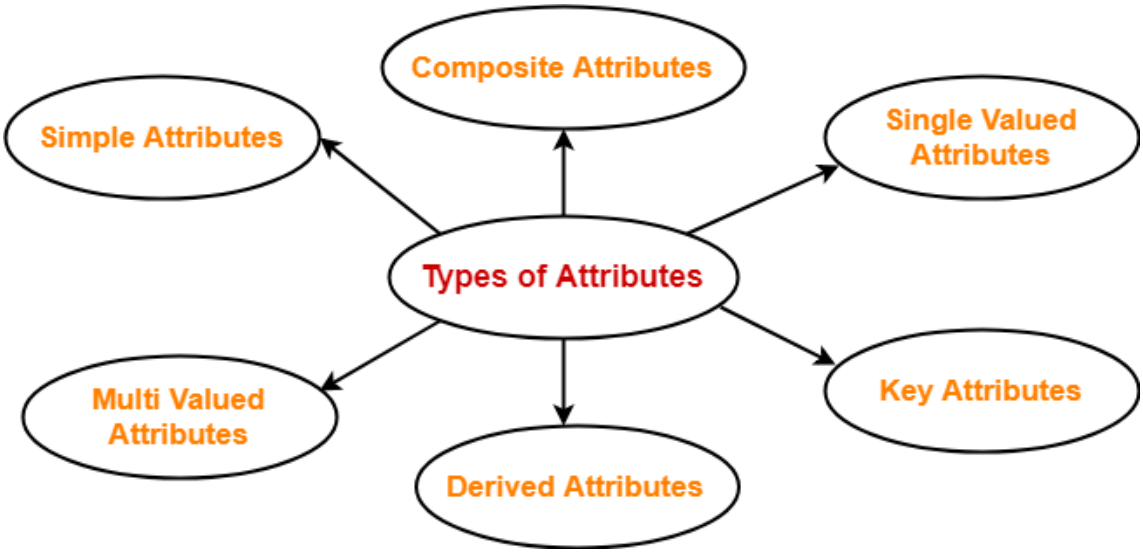
Attributes

Index

> *It is used to describe the property of an entity*

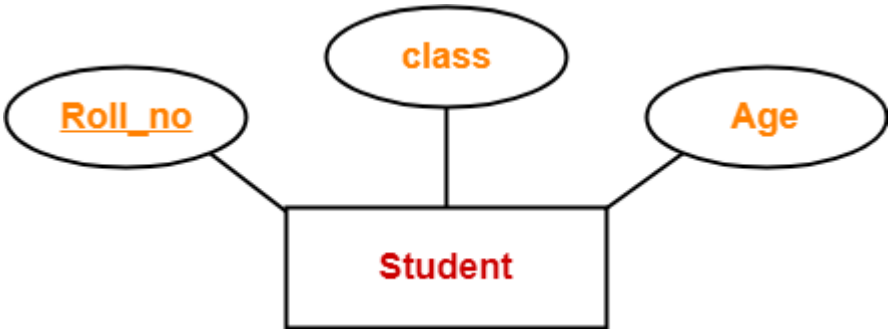
> *There exist a specific domain or set of values for each attribute from where that attribute can take its values.*

Types of Attributes-

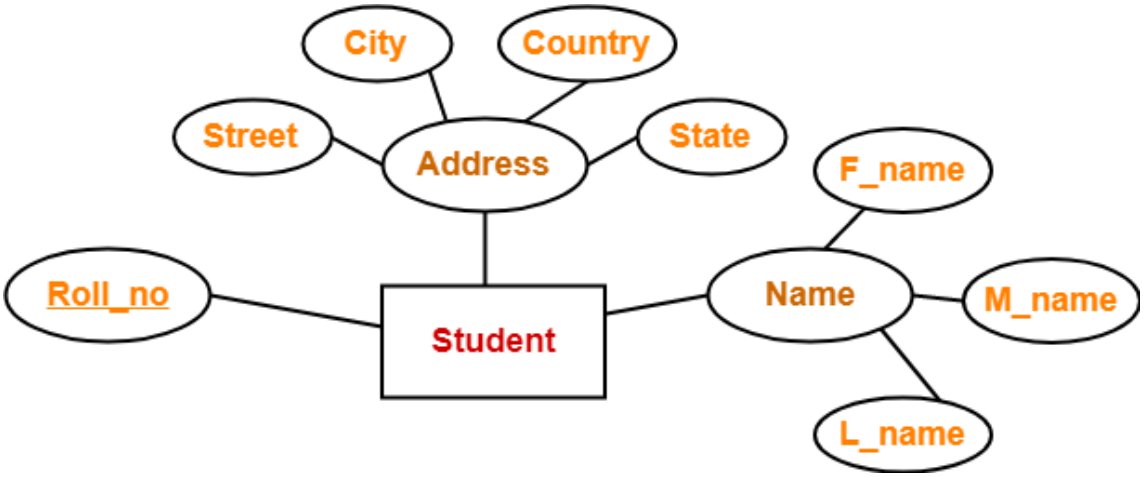


- Simple attributes
- Composite attributes
- Single valued attributes
- Multi valued attributes
- Derived attributes
- Key attributes

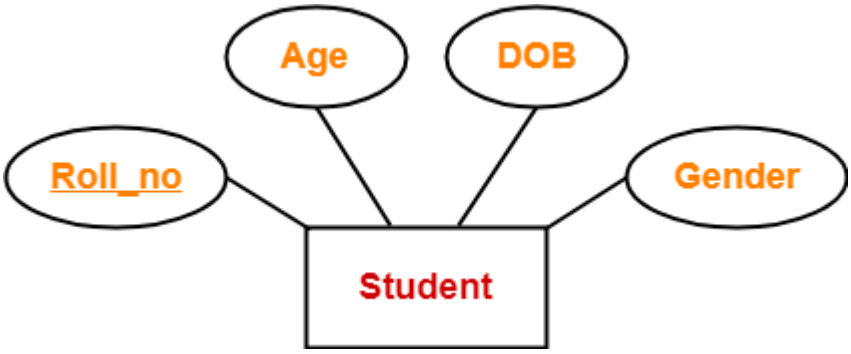
`Simple attributes -` : Simple attributes are those attributes which can not be divided further. Example: Here, all the attributes are simple attributes as they can not be divided further.



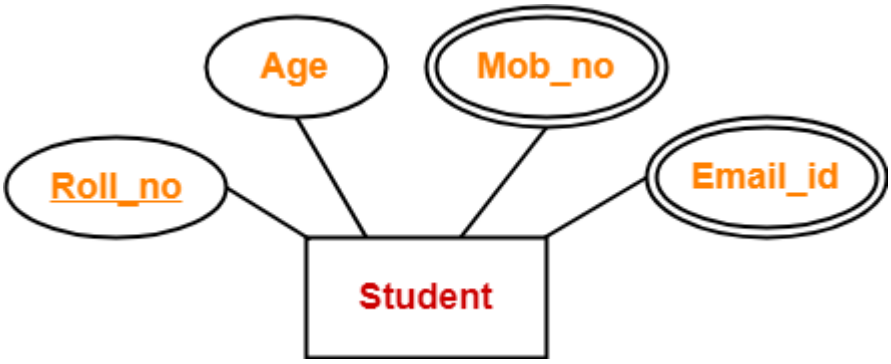
`Composite Attributes -` : Composite attributes are those attributes which are composed of many other simple attributes. (Which can be further divided). Example: Here, the attributes “Name” and “Address” are composite attributes as they are composed of many other simple attributes.



`Single Valued Attributes -` : Single valued attributes are those attributes which can take only one value for a given entity from an entity set. Example: Here, all the attributes are single valued attributes as they can take only one specific value for each entity.

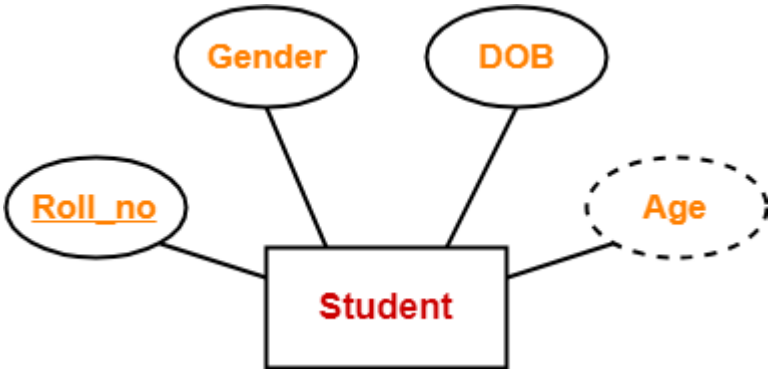


`Multi Valued Attributes -` : Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set. Example: Here, the attributes “Mob_no” and “Email_id” are multi valued attributes as they can take more than one values for a given entity.

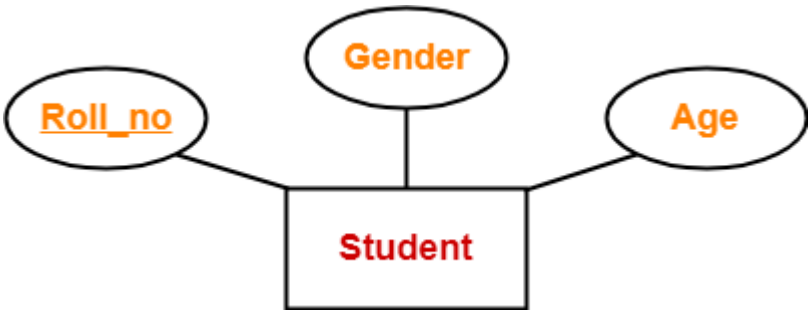


`Derived Attributes -` : Derived attributes are those attributes which can be derived from other attribute(s). Example: Here, the attribute “Age” is a derived attribute as

it can be derived from the attribute “DOB”.



`Key Attributes -` : Key attributes are those attributes which can identify an entity uniquely in an entity set. Example : Here, the attribute “Roll_no” is a key attribute as it can identify any student uniquely.



Closure of An Attribute -

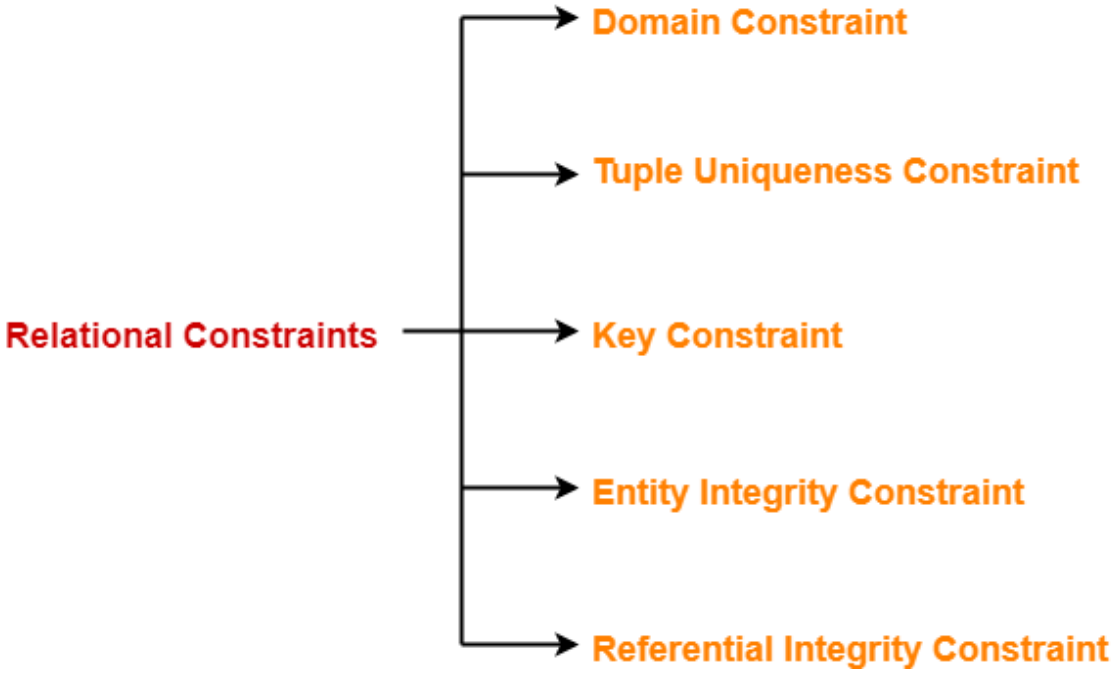
Youtube Index Not Important just go through it.

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.
- Closure of attribute set {X} is denoted as {X}+.

Constraints

Index

- Constraints are the rules enforced (that are applied) on the data columns of a table
- These are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the database.



- Domain constraint
- Tuple Uniqueness constraint
- Key constraint
- Entity Integrity constraint
- Referential Integrity constraint

1. Domain Constraint-

- Domain constraint defines the domain or set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc.
- It specifies that the value taken by the attribute must be the atomic value from its domain.

Example: Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	A

Here, value `A` is not allowed since only integer values can be taken by the age attribute.

2. Tuple Uniqueness Constraint- Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation. Example 1 : Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20
This relation satisfies the tuple uniqueness constraint since here all the tuples are unique.		

Example 2: Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Akshay	21
S003	Shashank	20
S004	Rahul	20
This relation does not satisfy the tuple uniqueness constraint since here all the tuples are not unique.		

3. Key Constraint- Key constraint specifies that in any relation-

- All the values of primary key must be unique.
- The value of primary key must not be null. Example: Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20
This relation does not satisfy the key constraint as here all the values of primary key are not unique.		

4. Entity Integrity Constraint-

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

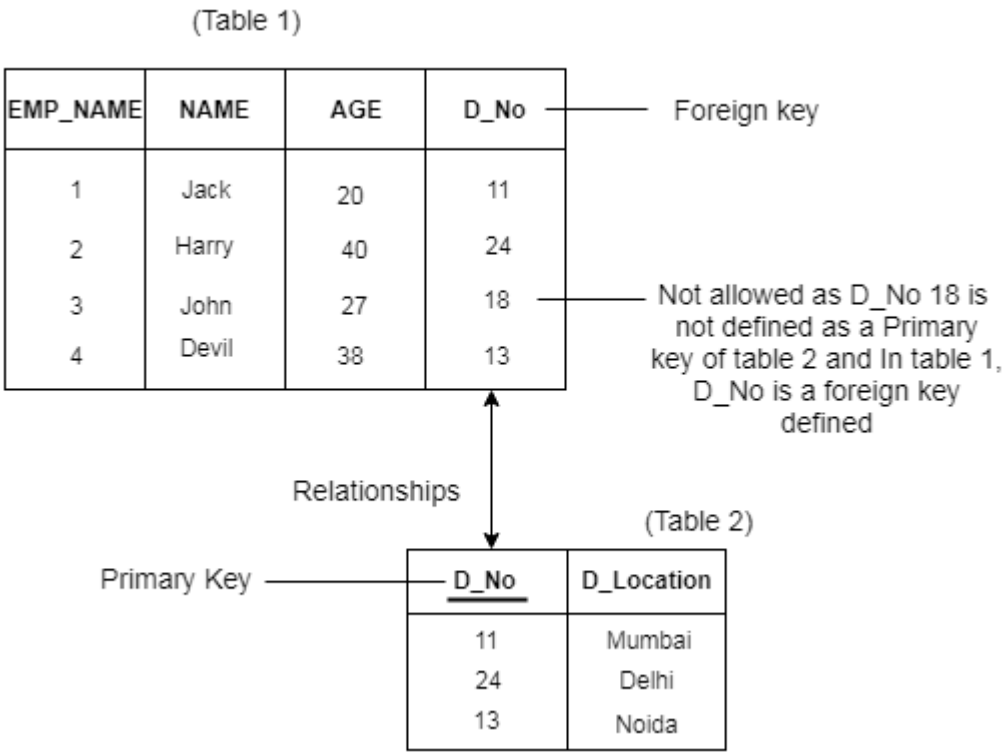
EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

5. Referential Integrity Constraint-

- A referential integrity constraint is specified between two tables.
- This constraint is enforced(applied) when a foreign key references the primary key of a relation.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

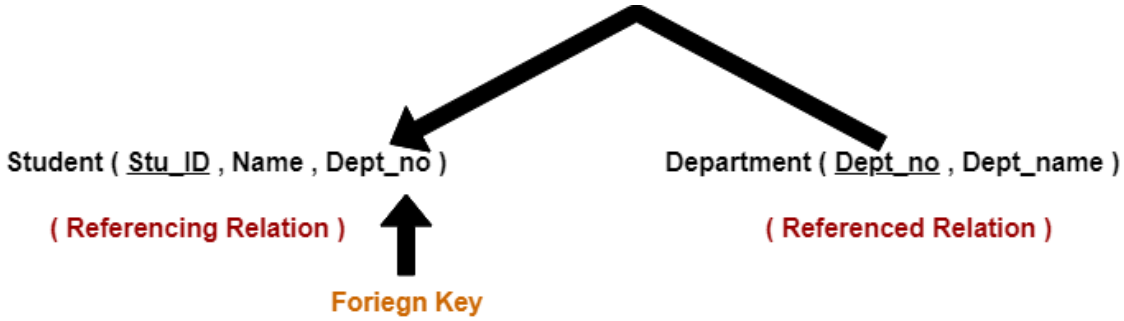


Important Results-

The following two important results emerges out due to referential integrity constraint-

- We can not insert a record into a referencing relation if the corresponding record does not exist in the referenced relation.
- We can not delete or update a record of the referenced relation if the corresponding record exists in the referencing relation.

Example- Consider the following two relations- `Student` and `Department`. Here, relation `Student` references the relation `Department`.



Student

STU_ID	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	D14

Department

Dept_no	Dept_name
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation ‘Student’ does not satisfy the referential integrity constraint.
- This is because in relation ‘Department’, no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

KEYS

Index (Do any one defination)

- It is set of attribute that uniquely identifies any record from the table.

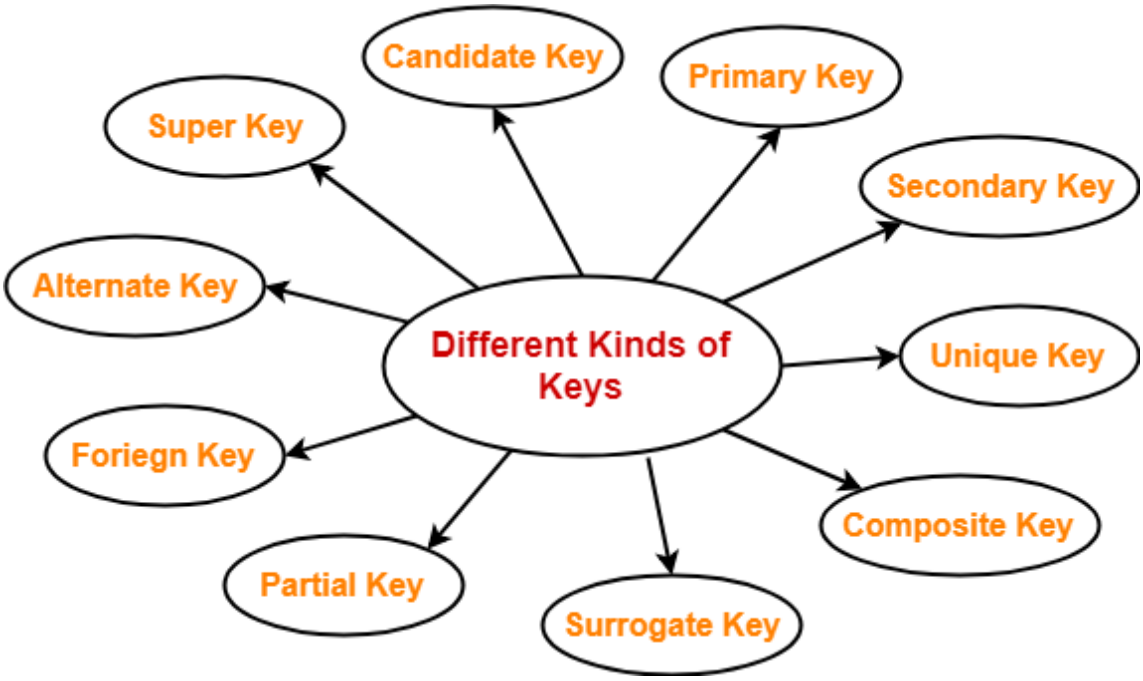
- A key is a set of attributes that can identify each tuple uniquely in the given relation.

> *Different Types Of Keys in DBMS -*

Video - [Youtube Link]

There are following 10 important keys in DBMS -

- Super key
- Candidate key
- Primary key
- Alternate key
- Foreign key
- Partial key
- Composite key
- Unique key



- `Super Key` :

  1. It the combination of all possible attributes that can be uniquely identify the rows in the given relation.

  2. A super key may consist of any number of attributes.

     Example- Consider the following Student schema-

     - Student (roll , name , sex , age , address , class , section) Given below are the examples of super keys since each set can uniquely identify each student in the Student table
     - (roll , name , sex , age , address , class , section)
     - (class , section , roll)
     - (class , section , roll , sex)

- (name , address)

NOTE - All the attributes in a super key are definitely sufficient to identify each tuple uniquely in the given relation but all of them may not be necessary.

- `Candidate Key` :
  1. A minimal super key is called as a candidate key.
  2. It is an attribute with uniquely identify a tuple.

     Example- Consider the following Student schema-

     - Student (roll , name , sex , age , address , class , section)

     Given below are the examples of candidate keys since each set consists of minimal attributes required to identify each student uniquely in the Student table-

     - (class , section , roll)
     - (name , address)

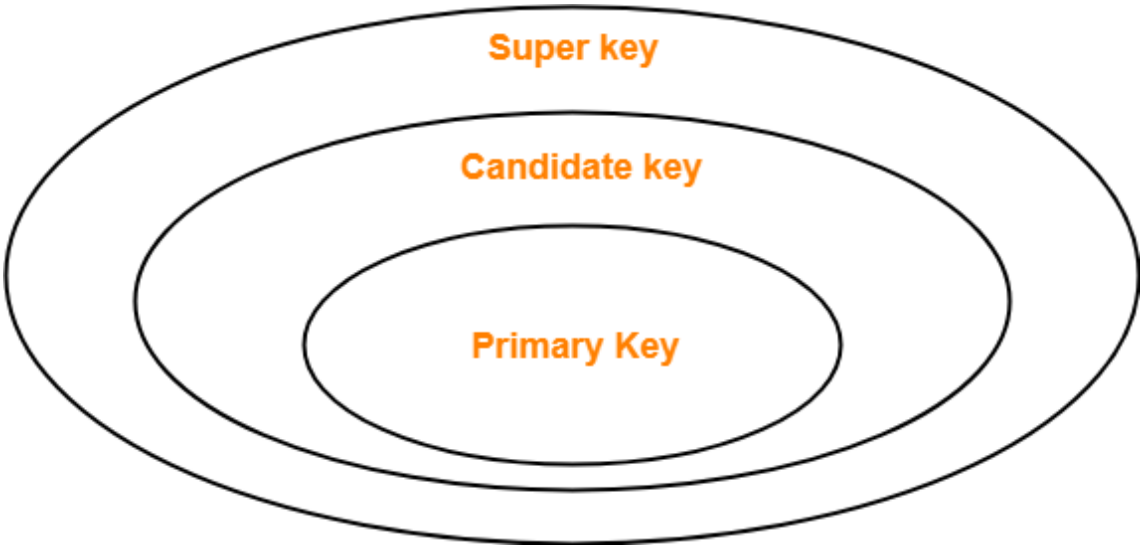
NOTES -

1. All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.
2. Removing any attribute from the candidate key fails in identifying each tuple uniquely.
3. The value of candidate key must always be unique.
4. The value of candidate key can never be NULL.
5. It is possible to have multiple candidate keys in a relation.
6. hose attributes which appears in some candidate key are called as prime attributes.

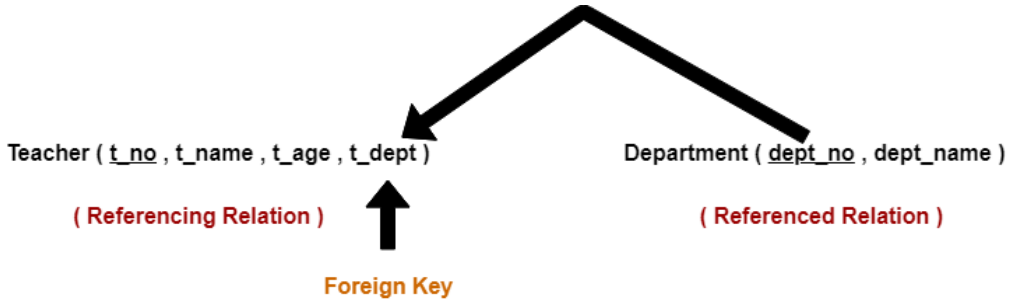
- `Primary Key` :
  1. It is one of the candidate key which uniquely identify the tuple in a table.
  2. A primary key is a candidate key that the database designer selects while designing the database.

NOTES-

1. The value of primary key can never be NULL.
2. The value of primary key must always be unique.
3. The values of primary key can never be changed i.e. no updation is possible.
4. The value of primary key must be assigned when inserting a record.
5. A relation is allowed to have only one primary key.



- `Alternate Key` :
    1. Alternate keys out of all the candidate is only one get selected as primary key but the remaining ki are alternate keys.
    2. Unimplemented candidate keys are called as alternate keys.
- `Foreign Key` : It is used to link the two tables together it references the primary key of another table Example : Here, t_dept can take only those values which are present in dept_no in Department table since only those departments actually exist.



NOTES-

1. Foreign key references the primary key of the table.
2. Foreign key can take only those values which are present in the primary key of the referenced relation.
3. Foreign key may have a name other than that of a primary key.
4. Foreign key can take the NULL value.
5. There is no restriction on a foreign key to be unique.
6. In fact, foreign key is not unique most of the time.
7. Referenced relation may also be called as the master table or primary table.
8. Referencing relation may also be called as the foreign table.

- `Composite Key` :

1. It having more than one attribute to identify the tuple is called the composite key.
2. A primary key comprising of multiple attributes and not just a single attribute is called as a composite key.

- `Unique Key` : Unique key is a key with the following properties-

  1. It is unique for all the records of the table.
  2. It may have a NULL value.
  3. Once assigned, its value can not be changed i.e. it is non-updatable.

     Example: The best example of unique key is Adhaar Card Numbers.

     - The Adhaar Card Number is unique for all the citizens (tuples) of India (table).
     - If it gets lost and another duplicate copy is issued, then the duplicate copy always has the same number as before.
     - Thus, it is non-updatable.
     - Few citizens may not have got their Adhaar cards, so for them its value is NULL.

Functional Dependancy

Index

- It is a **relationship between set of attributes** of a table, **dependent on each other**.
- It typically **exists between the primary key and non-key attribute** within a table.
- Functional Dependency is represented by → (arrow sign)
- Let us consider P is a relation with attributes A and B where A is primary key then

  > *A → B*
  >
  > *Here A is **Determinant** and B is **Dependent***

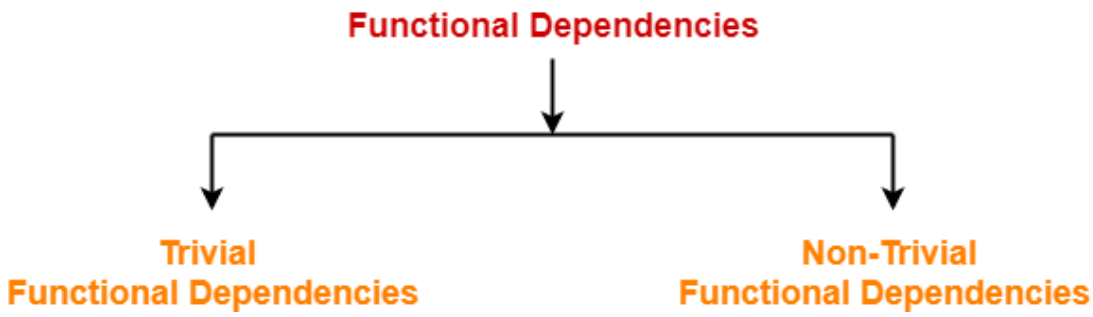
Example: We have a *Department* table with two attributes − *DeptId* and *DeptName*. The *DeptId* is our primary key. Here, *DeptId* uniquely identifies the *DeptName* attribute. This is because if you want to know the department name, then at first you need to have the *DeptId*.

DeptId	DeptName
001	Finance
002	Marketing
003	HR

Function Dependency : *DeptId → DeptName*

There are two types of functional dependencies-

1. Trivial Functional Dependencies
2. Non-trivial Functional Dependencies
3. Multivalued Functional Dependency
4. Transitive Functional Dependency



1. Trivial Function Dependencies :

- A functional dependency X → Y is said to be t**rivial if and only if Y X i.e Y is the subset of X**

Example: The examples of trivial functional dependencies are-

- AB → A
- AB → B
- AB → AB

We are considering the same *Department* table with two attributes to understand the concept of trivial dependency.

The following is a trivial functional dependency since *DeptId* is a subset of *DeptId* and *DeptName*

> *{ DeptId, DeptName } → Dept Id*

2. Non-Trivial Functional Dependencies :

- A functional dependency X → Y is said to be n**on-trivial if and only if Y X i.e if Y is not the subset of X.**

Example: The examples of non-trivial functional dependencies are-

- AB → BC
- AB → CD

Example:

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, *roll_no → name* is a non-trivial functional dependency, since the dependent *name* is not a subset of determinant *roll_no*.

Similarly, *{roll_no, name} → age* is also a non-trivial functional dependency, since age is not a subset of *{roll_no, name}*

3. Multivalued Functional Dependency :

- Entities of the dependent set are not dependent on each other.
- If *a → {b, c}* and there exists no functional dependency between b and c, then it is called a multivalued functional dependency.

Example:

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, *roll_no → {name, age} i*s a multivalued functional dependency, since the dependents name & age are not dependent on each other(i.e. *name → age *or a*ge → name d*oesn’t exist !)

4. Transitive Functional Dependency :

- Dependent is indirectly dependent on determinant.
- If *a → b & b → c,* then according to axiom of transitivity, *a → c.* This is a transitive functional dependency.

Example:

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, *enrol_no → dept *and *dept → building_no,* Hence, according to the axiom of transitivity,* enrol_no → building_no* is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Inference Rules-

1. **Reflexivity**- If B is a subset of A, then A → B always holds.
2. **Transitivity**- If A → B and B → C, then A → C always holds.
3. **Augmentation**- If A → B, then AC → BC always holds.
4. **Decomposition**- If A → BC, then A → B and A → C always holds.
5. **Composition**- If A → B and C → D, then AC → BD always holds.
6. **Additive**- If A → B and A → C, then A → BC always holds.

Decomposition of a Relation-

Index

> *The process of **breaking** up a **single relation** into **two or more sub relations** is called as decomposition of a relation.*

Properties of Decomposition-

1. Lossless decomposition- It ensures -

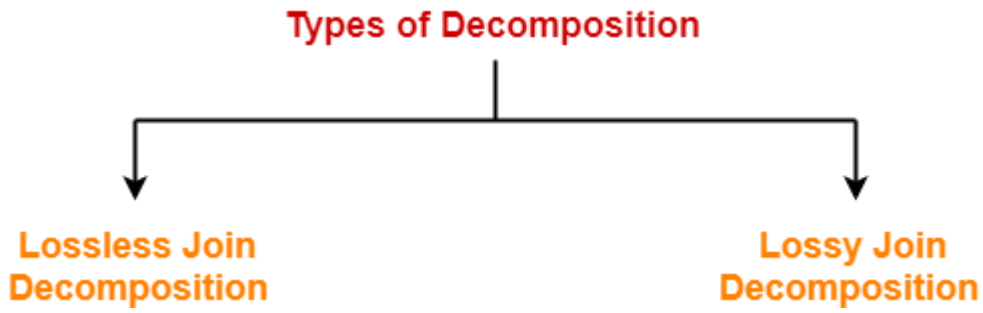
- No information is lost from the original relation during decomposition.
- When the sub relations are joined back, the same relation is obtained that was decomposed.
- Every decomposition must always be lossless.

2. Dependency Preservation- It ensures -

- None of the functional dependencies that holds on the original relation are lost.
- The sub relations still hold or satisfy the functional dependencies of the original relation.

Types of Decomposition-

1. Lossless Join Decomposition
2. Lossy Join Decomposition-



1. Lossless Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R1 , R2 , …. , Rn.
- This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed
- Also Known as **non-additive join decomposition.**
- For lossless join decomposition, we always have-

> *R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn = R where ⋈ is a natural join operator*

Example

2. Lossy Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R1 , R2 , …. , Rn.

- This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
- The natural join of the sub relations is always found to have some extraneous tuples.
- Also Known as **careless decomposition.**
- For lossy join decomposition, we always have-

  > *R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn ⊃ R where ⋈ is a natural join operator*

Example

Normalization

Index

- Database Normalization is a technique of organizing the data in the database
- Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition).
- and undesirable characteristics like Insertion, Update and Deletion Anomalies.
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes:

1. Eliminating redundant(useless) data.
2. Ensuring data dependencies make sense i.e data is logically stored.

Video - Youtube Link

Problems Without Normalization

1. If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss.
2. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized.

Example: To understand these anomalies let us take an example of a Student table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is Data Redundancy.

- Insertion Anomaly :

  1. Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to

set the branch information as NULL.
2. Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students. These scenarios are nothing but Insertion anomalies.

- Updation Anomaly : What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.
- Deletion Anomaly : In our Student table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

> *Normalization of a Database is achieved by following a set of rules called 'forms' in creating the database.*

Normalization Rule

(Must See the explanation part in starting of revision for some days.)

> *Short Note :*

Need for Normalization :

1. Improve database design.
2. Ensures minimum redundancy of data
3. Reduces need to reorganize data when design is modified.
4. Reduce anomalies for database activities.

Normalization rules are divided into the following normal forms:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. BCNF
5. Fourth Normal Form (4NF)

> *First Normal Form (1 NF)*

`Explanation` : Blog | Video

For a table to be in the First Normal Form, it should follow the following 4 rules:

- It should only have single(atomic) valued attributes/columns.
- Values stored in a column should be of the same domain
- All the columns in a table should have unique names.
- And the order in which data is stored, does not matter.

> *Second Normal Form (2NF)*

`Explanation` : Blog | Video

For a table to be in the Second Normal Form,

- It should be in the First Normal form.
- And, it should not have Partial Dependency.

Partial Dependency :

- Where an attribute in a table depends on only a part of the primary key and not on the whole key(Composite Key).
- Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key.
- Example

> *Third Normal Form (3NF)*

`Explanation` : Blog | Video

A table is said to be in the Third Normal Form when,

- It is in the Second Normal form.
- And, it doesn't have Transitive Dependency.

Transitive Dependency :

- When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.
- 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.
- Example

> *Boyce and Codd Normal Form (BCNF)*

`Explanation` : Blog | Video

A table is said to be in the BCNF when,

- It should be in the Third Normal Form.
- And, for any dependency A → B, A should be a super key. The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a non-prime attribute, if B is a prime attribute.

> *Fourth Normal Form (4NF)*

`Explanation` : Blog | Video

A table is said to be in the Fourth Normal Form when,

- It is in the Boyce-Codd Normal Form.
- And, it doesn't have Multi-Valued Dependency.

What is Multi-valued Dependency :

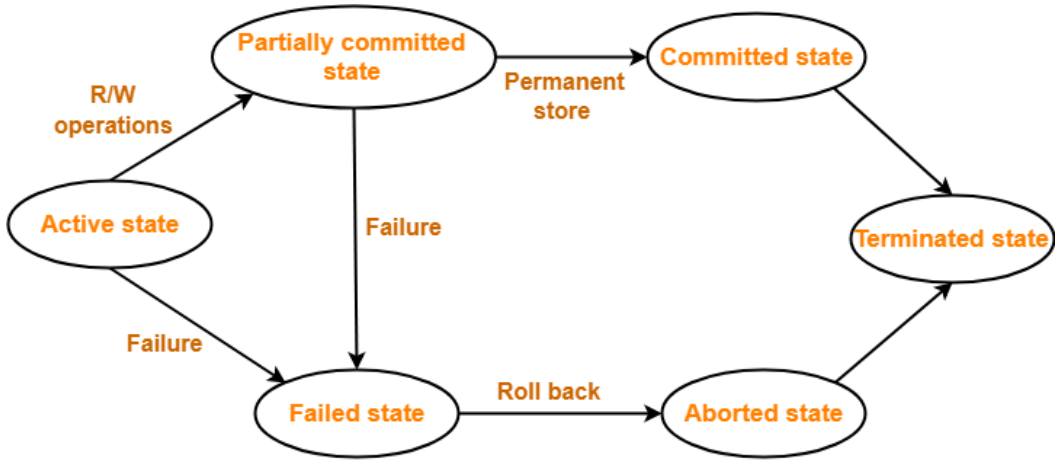
- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.
- Example

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Transaction

Index

- A transaction is a set of logically related operations.
- A transaction goes through different states throughout its life cycle.
- The life cycle of a transaction is-



Transaction States in DBMS

Transaction State - Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state

1. Active State-

- This is the first state in the life cycle of a transaction.
- A transaction is called in an active state as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

2. Partially Committed State-

- After the last instruction of transaction has executed, it enters into a partially committed state.
- After entering this state, the transaction is considered to be partially committed.
- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

3. Committed State-

- After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
- Now, the transaction is considered to be fully committed.

> *Note :*

- After a transaction has entered the committed state, it is not possible to roll back the transaction.
- In other words, it is not possible to undo the changes that has been made by the transaction.
- This is because the system is updated into a new consistent state.
- The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operations.

4. Failed State-

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

5. Aborted State-

- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state.

6. Terminated State-

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

Operations in Transaction-

- **Read Operation** *R(A)/Read(A)* : Read operation reads the data from the database and then stores it in the buffer in main memory.
- **Write Operation** *W(A)/Write(A)* : Write operation writes the updated data value back to the database from the buffer.

ACID Properties of Transaction

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as ACID Properties of a transaction.

A = Atomicity

C = Consistency

I = Isolation

D = Durability

1. Atomicity -

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as “All or nothing rule“.
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

2. Consistency -

- This property ensures that integrity constraints are maintained. [Integrity Constraints]
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

3. Isolation -

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

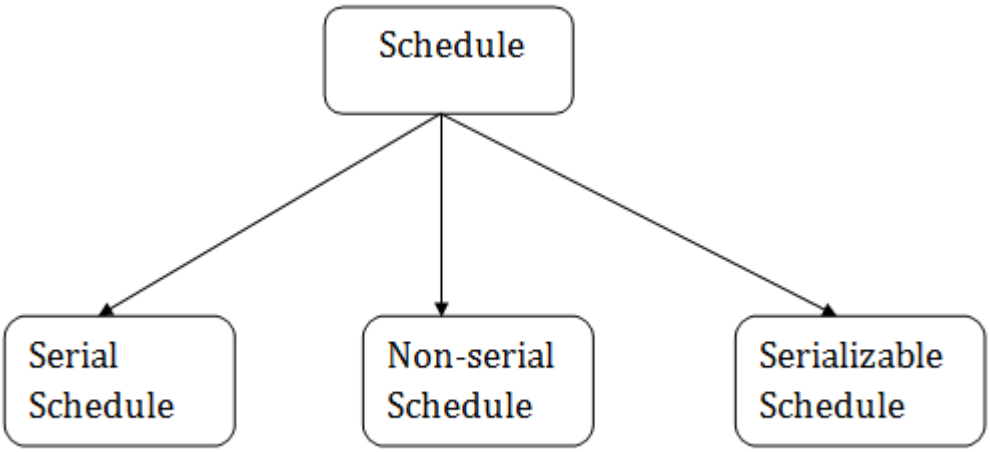
4. Durability -

- This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
- It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

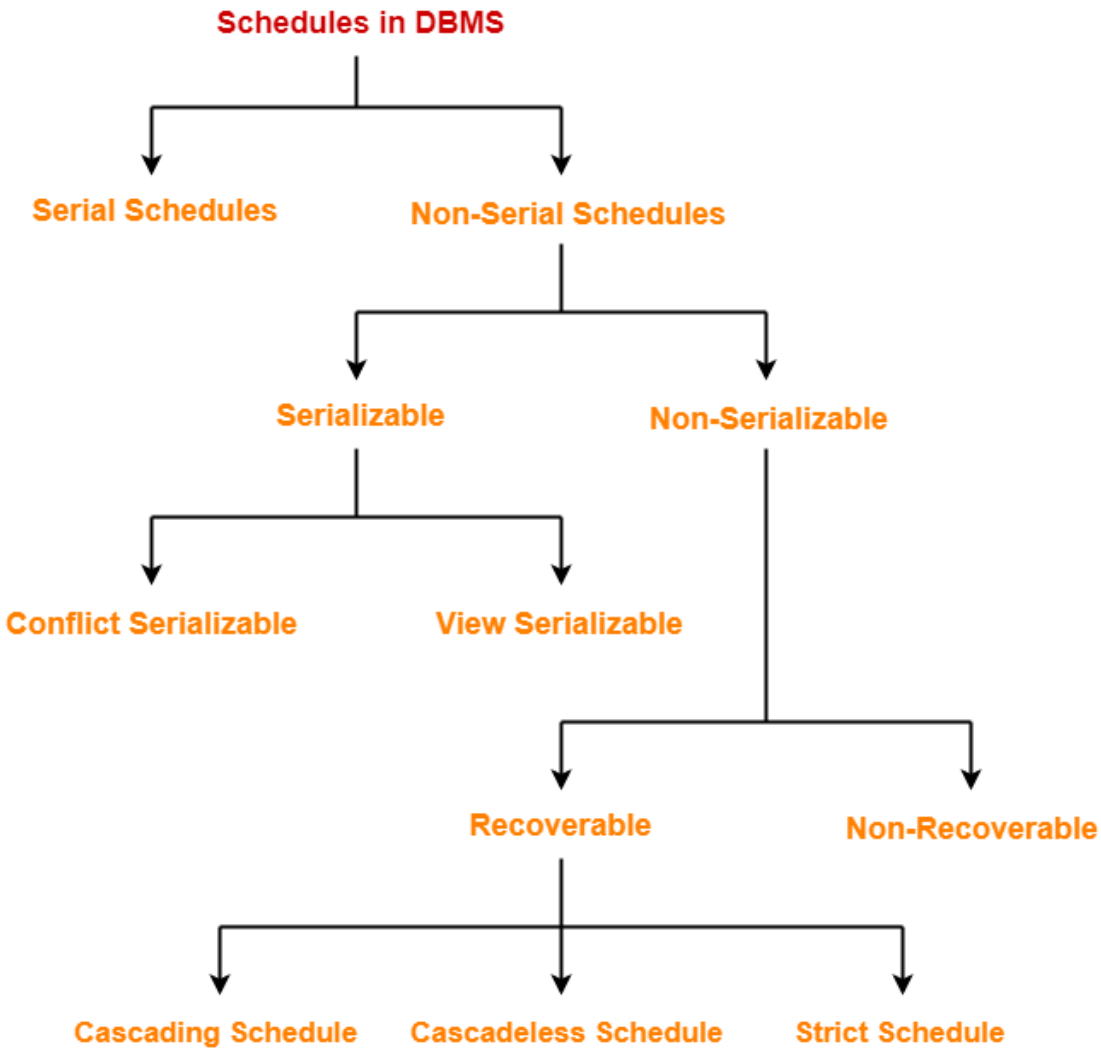
Schedules

Index

- A series of operation from one transaction to another transaction is known as schedule



Types of Schedules-



1. Serial Schedules-

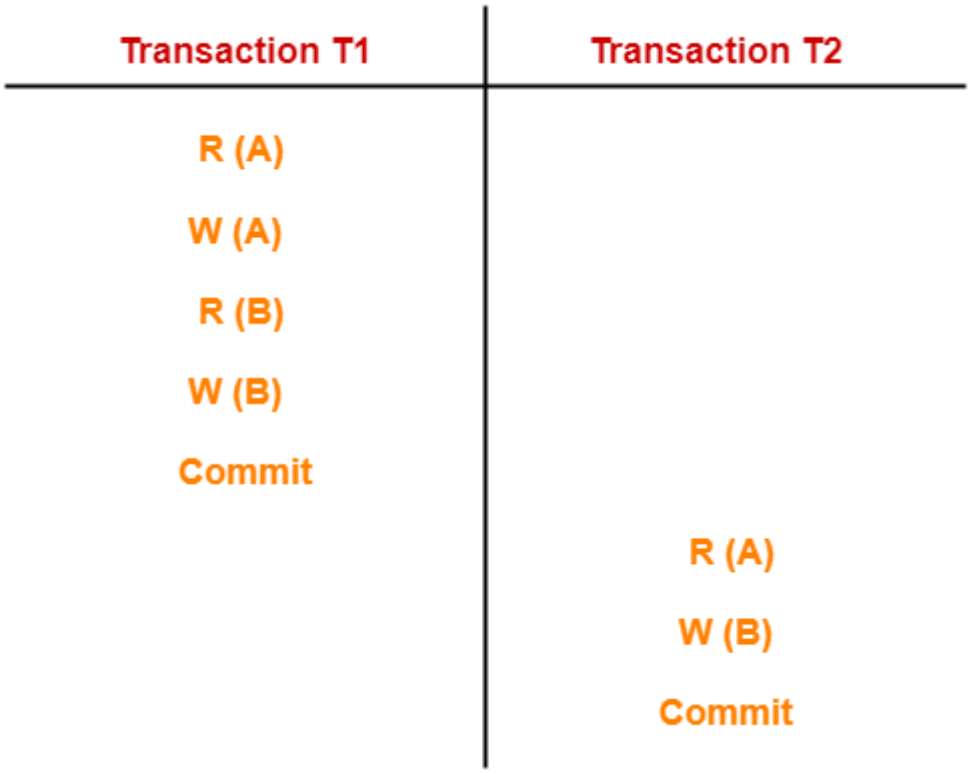
- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

> *Characteristics-*

Serial schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

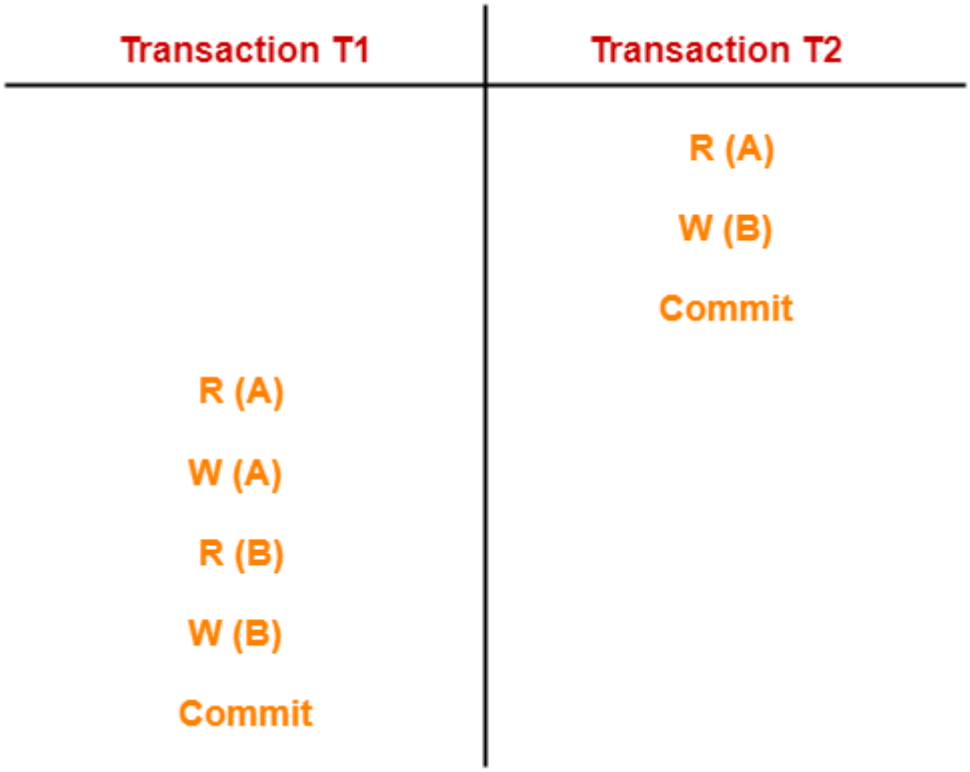
Example:



In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a Serial Schedule.

Example:



In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T2 executes first.
- After T2 completes its execution, transaction T1 executes.
- So, this schedule is an example of a Serial Schedule.

2. Non-Serial Schedules-

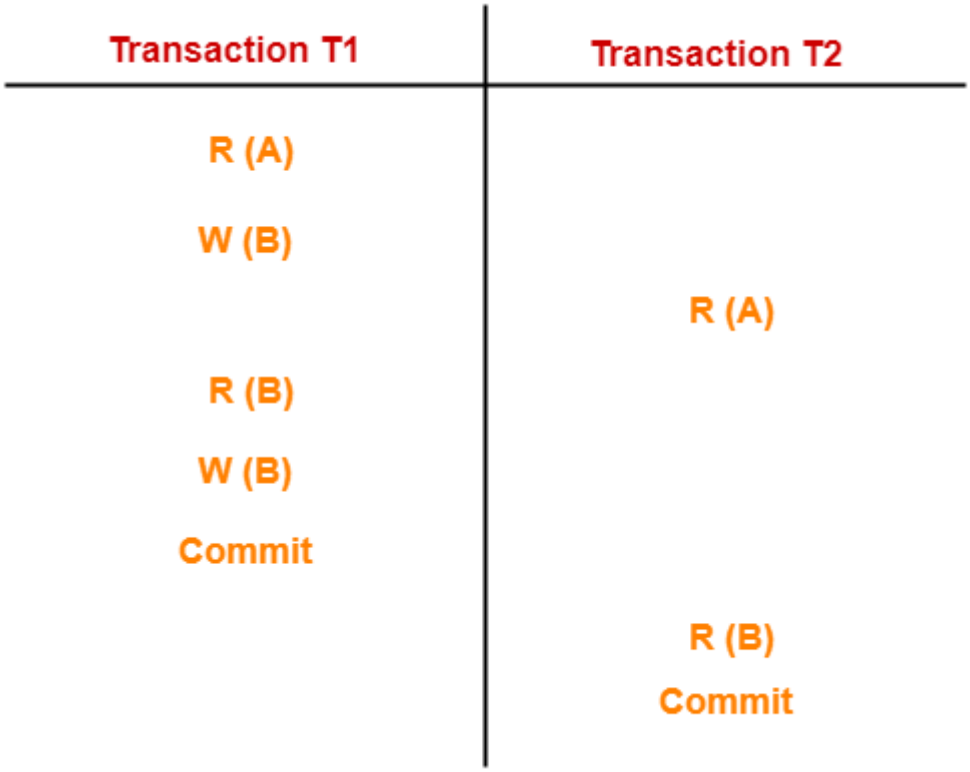
- Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.

> *Characteristics-*

Non-serial schedules are **NOT** always-

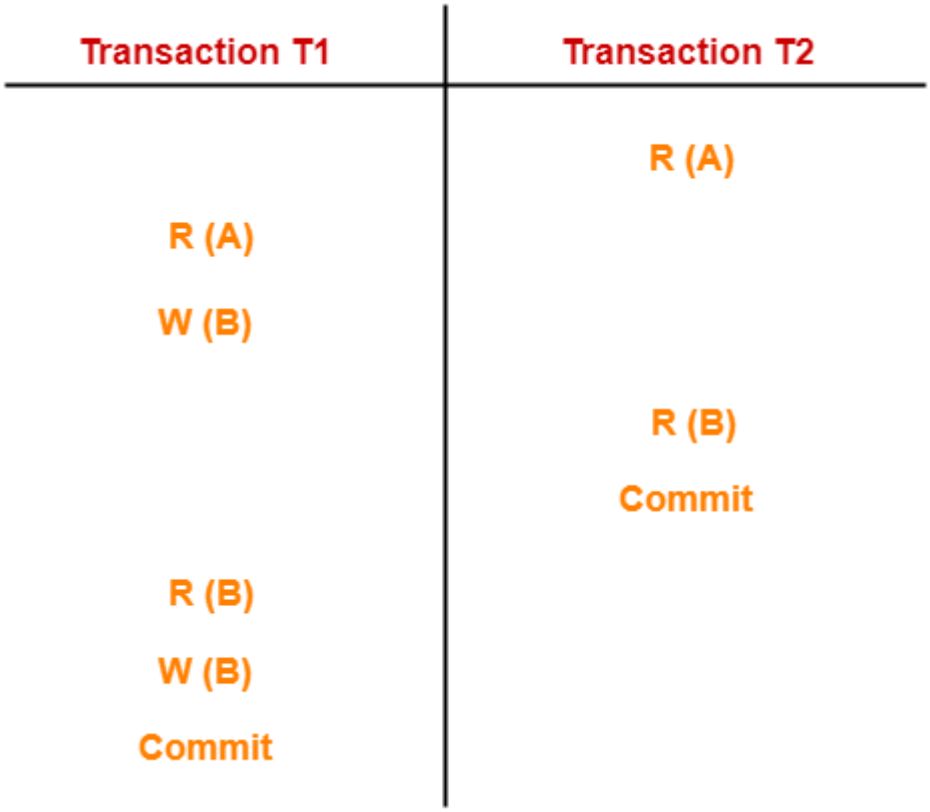
- Consistent
- Recoverable
- Cascadeless
- Strict

Example:



In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.



In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a Non-Serial Schedule.

Note-

- Serial schedules are always consistent.
- Non-serial schedules are not always consistent.

---

3. Serializable schedule [Youtube]

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

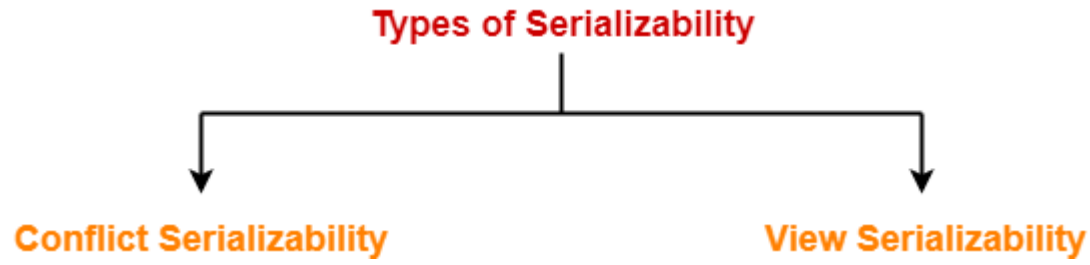
> *If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a serializable schedule.*

Characteristics- Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Casacadeless
- Strict

Types of Serializability-



Serializability is mainly of two types-

- Conflict Serializability
- View Serializability

Conflict Serializability- Youtube If a given **non-serial schedule** can be **converted into** a **serial schedule** by **swapping its non-conflicting operations**, then it is called as a **conflict serializable schedule**.

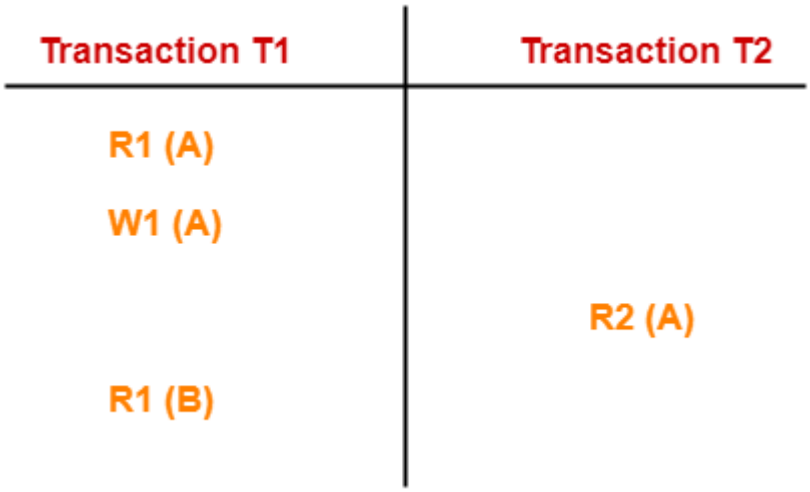
> *Conflicting Operations-*

Two operations are called as conflicting operations if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

> *Conflictig Pairs: (Important)*

- Write - Write (W - W)
- Write - Read (W - R)
- Read - Write (R - W)



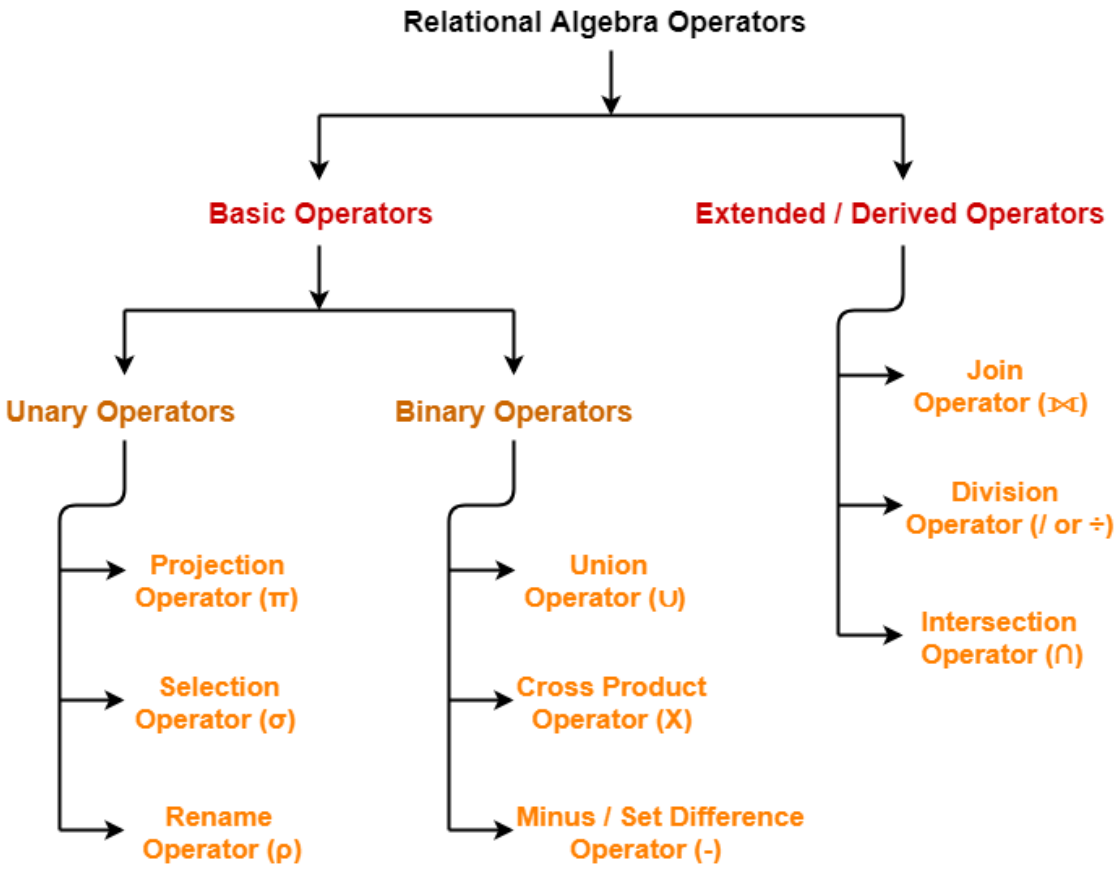
In this schedule, W1 (A) and R2 (A) are called as conflicting operations.

View Serializability- Youtube

Relational Algebra

Index

- Relational Algebra is a procedural query language which takes a relation as an input and generates a relation as an output.



Types of Relational operation

- Selection Operation
- Projection Operation
- Union Operation
- Set Intersection
- Set Difference
- Cartesian product
- Rename Operation

> ***Selection Operation***:- *BLOG*

- Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.
- It selects those rows or tuples from the relation that satisfies the selection condition.

Note-

- Selection operator only selects the required tuples according to the selection condition.
- It does not display the selected tuples.
- To display the selected tuples, projection operator is used.
- Selection operator always selects the entire tuple. It can not select a section or part of a tuple.
- We may use logical operators like , , ! and relational operators like = , ≠ , > , < , <= , >= with the selection condition.

> ***Projection Operation***:- *BLOG*

- Relational Operators always work on one or more relational tables.
- Relational Operators always produce another relational table. The table produced by a relational operator has all the properties of a relational model.

Note-

- Projection operator automatically removes all the duplicates while projecting the output relation.
- So, cardinality of the original relation and output relation may or may not be same.
- If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.
- If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation.
- This is because then there will be no duplicates to filter.
- There is only one difference between projection operator of relational algebra and SELECT operation of SQL.
- Projection operator does not allow duplicates while SELECT operation allows duplicates.
- To avoid duplicates in SQL, we use “distinct” keyword and write SELECT distinct
- Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

Note-

- Selection Operator performs horizontal partitioning of the relation.
- Projection operator performs vertical partitioning of the relation.

> ***Union Operation***:- *BLOG*

Let R and S be two relations. Then-

- R S is the set of all tuples belonging to either R or S or both.
- In R S, duplicates are automatically removed.
- Union operation is both commutative and associative.

> ***Intersection Operator (∩)***:- *BLOG*

Let R and S be two relations. Then-

- R ∩ S is the set of all tuples belonging to both R and S.
- In R ∩ S, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

> ***Difference Operator (-)***:- *BLOG*

Let R and S be two relations. Then-

- R – S is the set of all tuples belonging to R and not to S.
- In R – S, duplicates are automatically removed.
- Difference operation is associative but not commutative.

> ***Cartesian product***:- *BLOG*

- The Cartesian product is used to combine each row in one table with each row in the other table.
- It is also known as a cross product.
- It is denoted by X.
- Example in Blog

> ***Rename Operation***:- *BLOG*

- The rename operation is used to rename the output relation. - It is denoted by rho (ρ).
- Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

> ***Division Operator***:- *Video*

Joins

Video Index

> *Summary:*

- There are mainly **two types** of **joins** in DBMS
  1. **Inner Join**
  2. **Outer Join**
- An **inner join** is the widely used join operation and can be considered as a **default join-type**.
- **Inner Join** is further divided into **three subtypes**:
  1. **Theta join**
  2. **Natural join**
  3. **EQUI join**
- **Theta Join** allows you to **merge two tables** based on the **condition** represented by **theta**
- When a **theta join use**s only **equivalence condition**, it becomes an **equi join**.
- **Natural join** does not utilize any of the comparison operators.
- An **outer join** doesn’t require each record in the two join tables to have a matching record.
- Outer Join is further divided into three subtypes are:
  1. **Left Outer Join**
  2. **Right Outer Join**
  3. **Full Outer Join**
- The **LEFT Outer Join** returns all the rows from the table on the left, even if no matching rows have been found in the table on the right.
- The **RIGHT Outer Join** returns all the columns from the table on the right, even if no matching rows have been found in the table on the left.
- In a **full outer join**, all tuples from both relations are included in the result, irrespective of the matching condition.

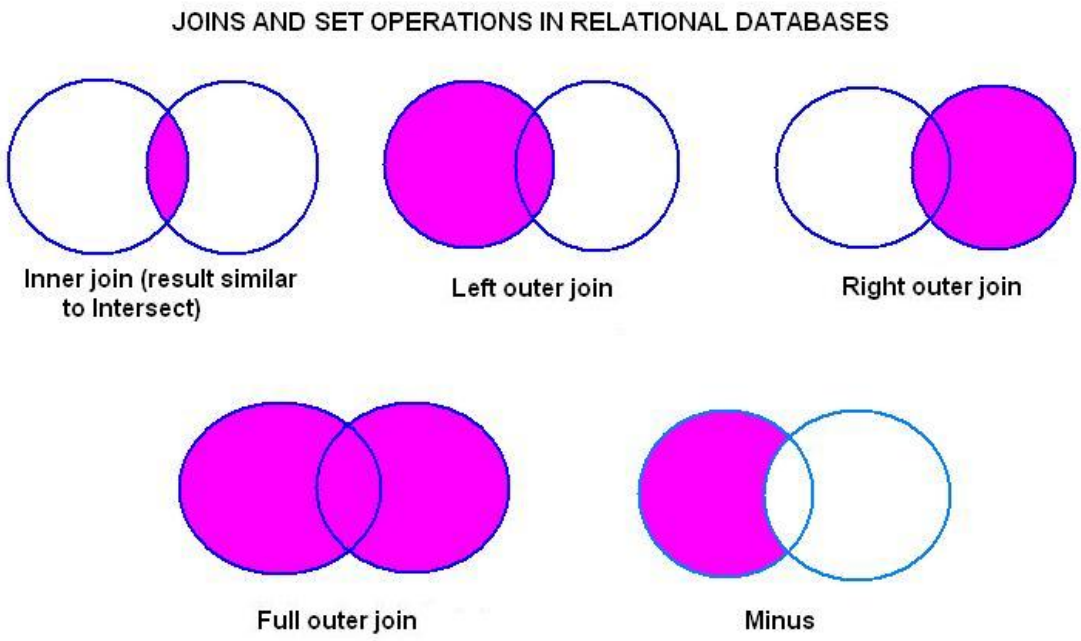
> *Defination and Explanation :*

- **Join** is a **combination** of a **Cartesian product** followed by a **selection process**.

  > *Join = Cartesian Product + Selection*

- A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.
- Symbol :

  > *A B = σ(A X B)*



Difference between Cartesian Product and Joins

Joins	Catesian Product
Combination of Tuples that satisfy matching conditions	All possible combinations of tuples/rows from the realtion/table
Fewer tuples then cross product,might be able to compute efficiently	Huge Number of tuples and costly to manage

There are two types of Joins

1. Inner Join
2. Outer Join

Inner Join	Outer Join
Theta Join	Left Join
Equi Join	Right Join
Natural Join	Full Join

Inner Join-

- Contains only those tuples that satisfy the matching conditions.

Outer Join-

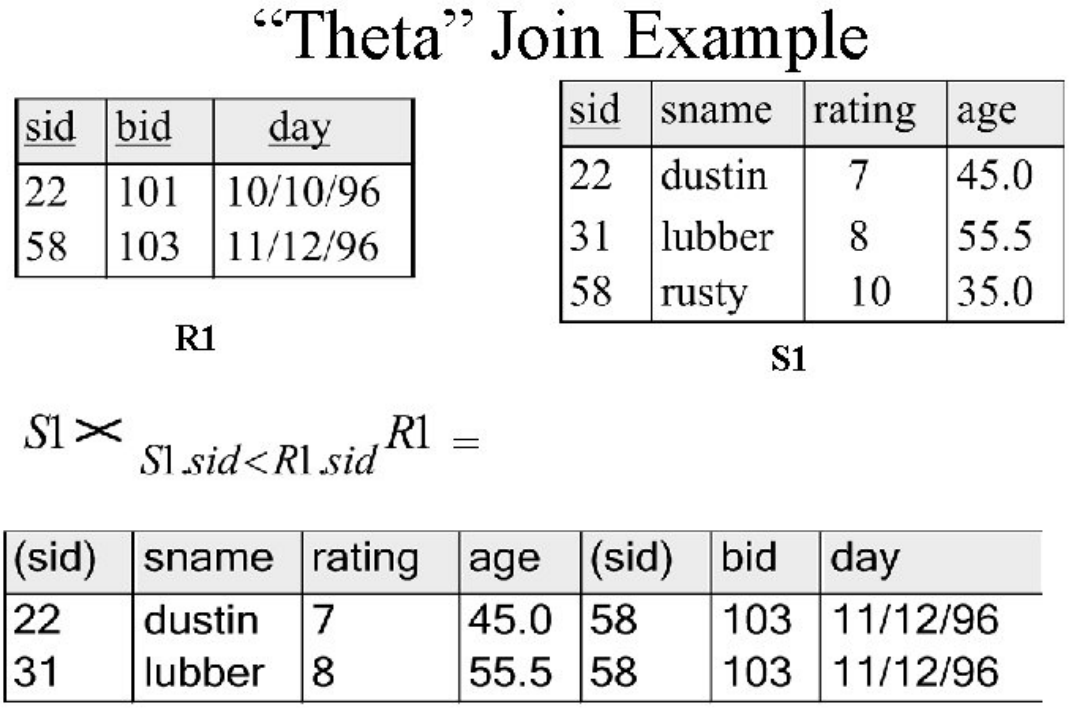
- Extension of Join
- Contains those tuples that satisfy the matching conditions, along with some or all tuples that do not satisfy the matching conditions
- Contain all rows from either one or both relations.

Inner Join -

> *Theta Join:*

- THETA JOIN allows you to merge two tables based on the condition represented by theta.
- Theta joins work for all comparison operators.
- It is denoted by symbol θ.
- The general case of JOIN operation is called a Theta join.

Example:



> *Equi Join*

- EQUI JOIN is done when a Theta join uses only the equivalence condition (=).
- EQUI join is the most difficult operation to implement efficiently in an RDBMS, and one reason why RDBMS have essential performance problems.

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

age	P.zip	disease	name	V.zip
54	98125	heart	p1	98125
20	98120	flu	p2	98120

> *Natural Join*

- NATURAL JOIN does not utilize any of the comparison operators.
- In this type of join, the attributes should have the same name and domain.
- In Natural Join, there should be at least one common attribute between two relations.
- It performs selection forming equality on those attributes which appear in both relations and eliminates the duplicate attributes.

Example:

Employee

Name	EmpId	GrpName
Jens	1234	A
Marcos	1235	B
Shant	1236	A
Lukas	1237	B

Group

GrpName	Manager
A	Jens
B	Lukas
C	Hans

Employee ⋈ Group

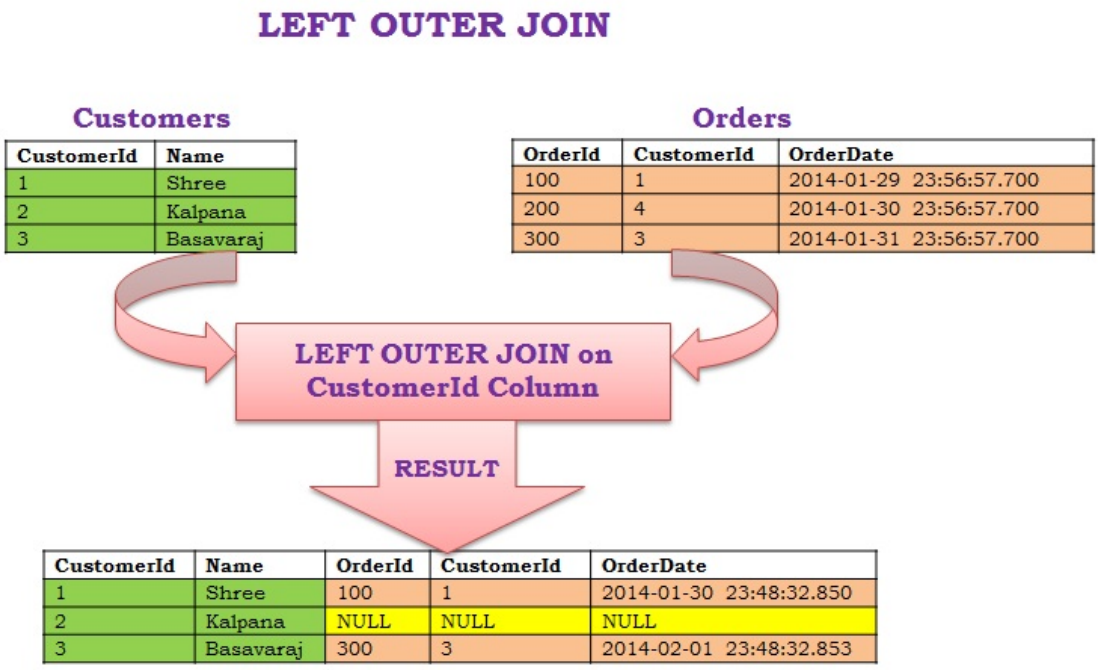
Name	EmpId	GrpName	Manager
Jens	1234	A	Jens
Marcos	1235	B	Lukas
Shant	1236	A	Jens
Lukas	1237	B	Lukas

Outer Join

> *Left Outer Join:*

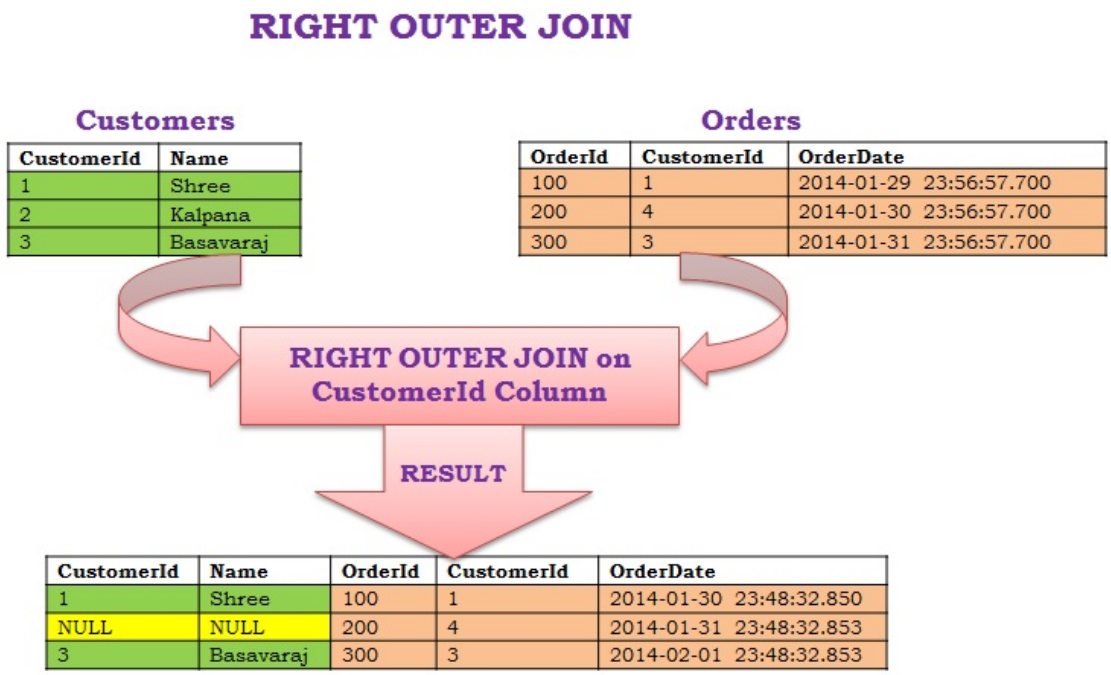
(Check Symbol on Google)

- LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right.
- When no matching record found in the table on the right, NULL is returned.



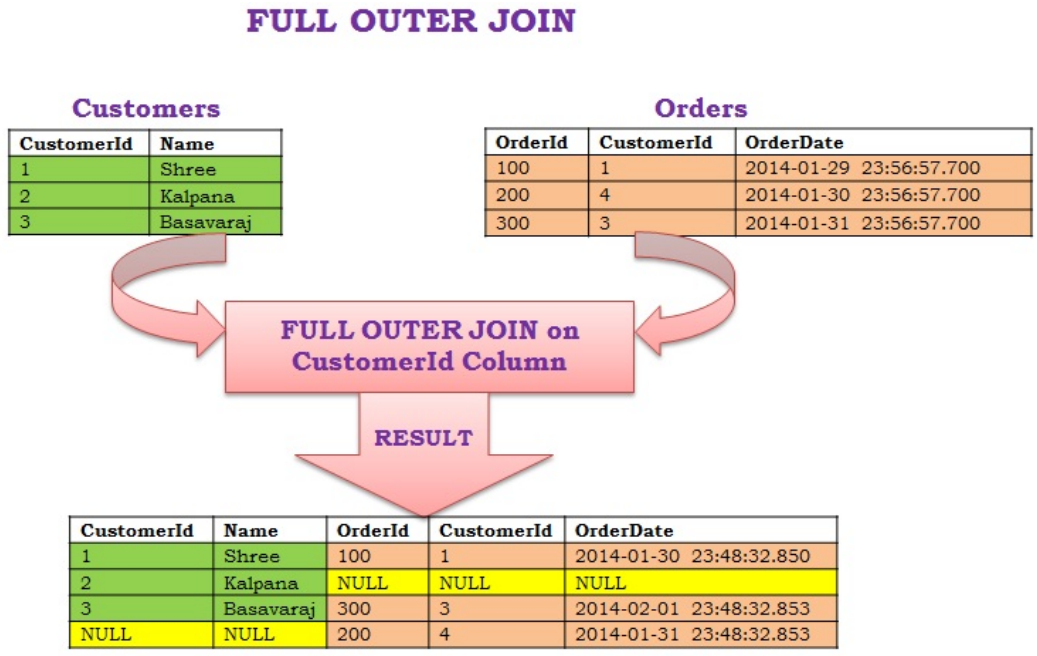
> *Right Outer Join:*

- RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left.
- When no matches have been found in the table on the left, NULL is returned.
- RIGHT outer JOIN is the opposite of LEFT JOIN.



> *Full Outer Join:*

- In a FULL OUTER JOIN , all tuples from both relations are included in the result, irrespective of the matching condition.



-]

Created by Sanheen Sethi and Riya Tripathi (2021)