Data Modeling

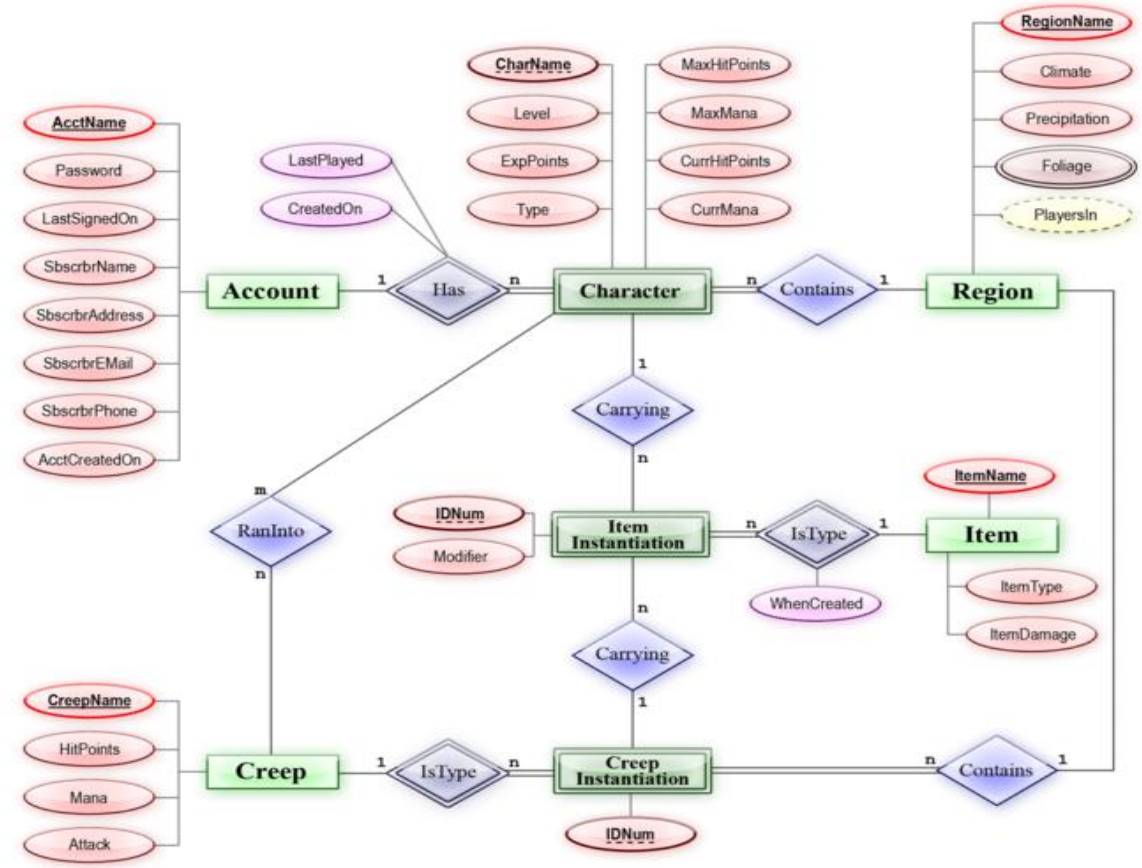
Data modeling in DBMS is the process of creating a data model for an information system by applying certain formal techniques . A data model in database management system (DBMS) is a tool that summarizes the description of the database and provides a transparent picture of data, which helps in creating an actual database . Data models are used to describe how the data is stored, accessed, and updated in a DBMS .

Entity-Relationship Diagram

An **entity–relationship model** (or **ER model**) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types).

In software engineering, an ER model is commonly formed to represent things a business needs to remember in order to perform business processes. Consequently, the ER model becomes an abstract data model,[1] that defines a data or information structure which can be implemented in a database, typically a relational database.

Entity–relationship modeling was developed for database and design by Peter Chen and published in a 1976 paper, with variants of the idea existing previously, but today it is commonly used for teaching students the basics of data base structure. Some ER models show super and subtype entities connected by generalization-specialization relationships, and an ER model can be used also in the specification of domain-specific ontologies.

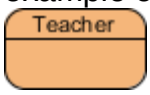


ERD notations guide

An ER Diagram contains entities, attributes, and relationships. In this section, we will go through the ERD symbols in detail.

Entity

An ERD entity is a **definable thing or concept within a system**, such as a person/role (e.g. Student), object (e.g. Invoice), concept (e.g. Profile) or event (e.g. Transaction) (note: In ERD, the term "entity" is often used instead of "table", but they are the same). When determining entities, think of them as nouns. In ER models, an entity is shown as a rounded rectangle, with its name on top and its attributes listed in the body of the entity shape. The ERD example below shows an example of an ER entity.

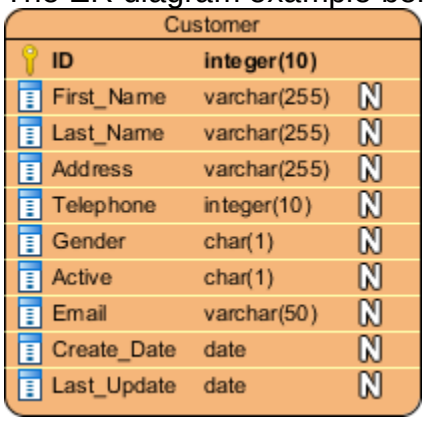


Entity Attributes

Also known as a column, an attribute is a **property or characteristic of the entity that holds it**.

An attribute has a name that describes the property and a type that describes the kind of attribute it is, such as varchar for a string, and int for integer. When an ERD is drawn for physical database development, it is important to ensure the use of types that are supported by the target RDBMS.

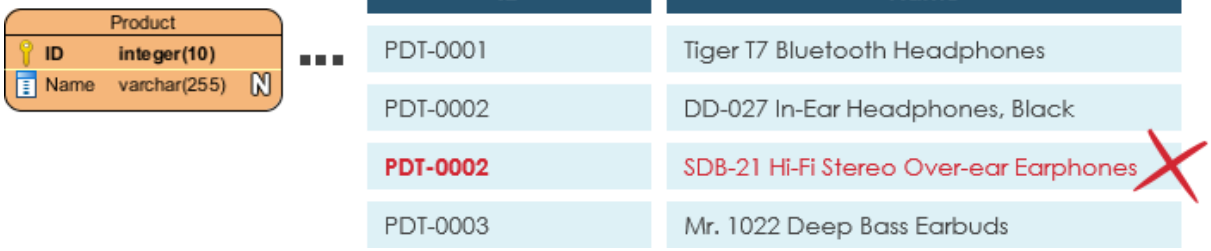
The ER diagram example below shows an entity with some attributes in it.



Primary Key

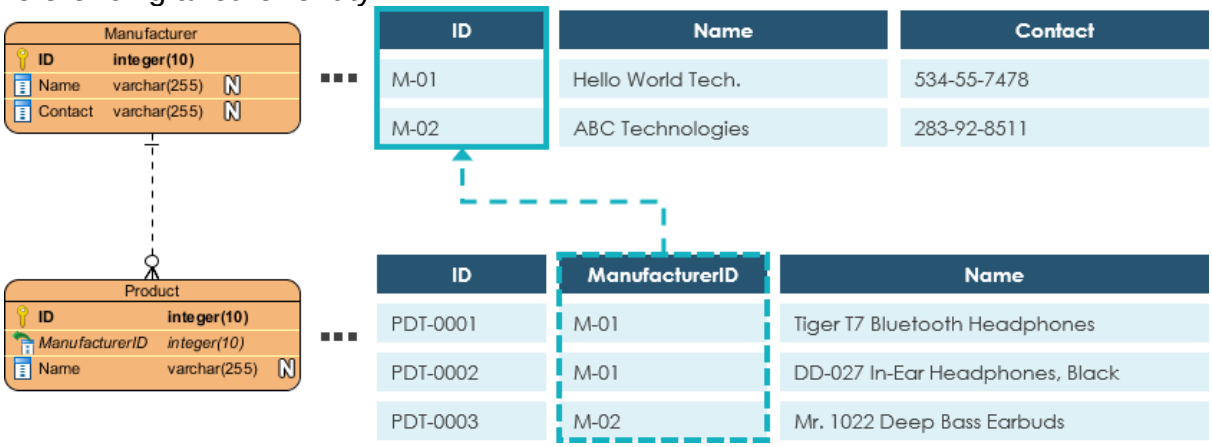
Also known as PK, a primary key is a special kind of entity attribute that **uniquely defines a record in a database table**. In other words, there must not be two (or

more) records that share the same value for the primary key attribute. The ERD example below shows an entity 'Product' with a primary key attribute 'ID', and a preview of table records in the database. The third record is invalid because the value of ID 'PDT-0002' is already used by another record.



Foreign Key

Also known as FK, a foreign key is a **reference to a primary key in a table**. It is used to identify the relationships between entities. Note that foreign keys need not be unique. Multiple records can share the same values. The ER Diagram example below shows an entity with some columns, among which a foreign key is used in referencing another entity.



Relationship

A relationship between two entities signifies that the **two entities are associated with each other somehow**. For example, a student might enroll in a course. The entity Student is therefore related to Course, and a relationship is presented as a connector connecting between them.

Cardinality

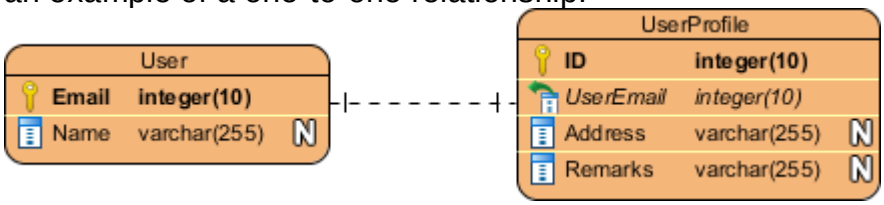
Cardinality defines the **possible number of occurrences in one entity which is associated with the number of occurrences in another**. For example, ONE

team has MANY players. When present in an ERD, the entity Team and Player are inter-connected with a one-to-many relationship.

In an ER diagram, cardinality is represented as a crow's foot at the connector's ends. The three common cardinal relationships are one-to-one, one-to-many, and many-to-many.

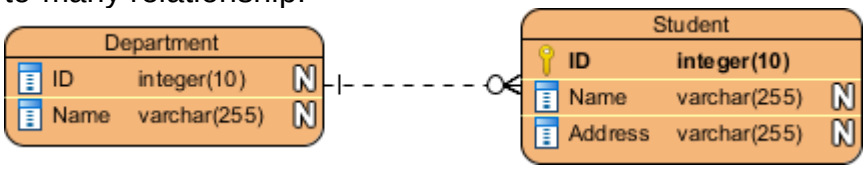
One-to-One cardinality example

A one-to-one relationship is mostly used to split an entity in two to provide information concisely and make it more understandable. The figure below shows an example of a one-to-one relationship.



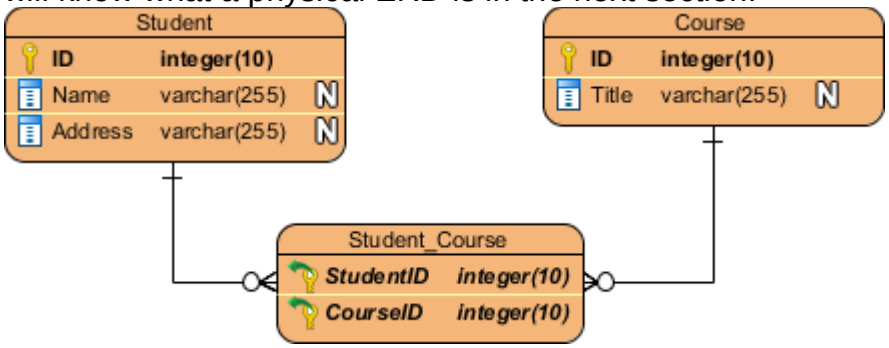
One-to-Many cardinality example

A one-to-many relationship refers to the relationship between two entities X and Y in which an instance of X may be linked to many instances of Y, but an instance of Y is linked to only one instance of X. The figure below shows an example of a one-to-many relationship.



Many-to-Many cardinality example

A many-to-many relationship refers to the relationship between two entities X and Y in which X may be linked to many instances of Y and vice versa. The figure below shows an example of a many-to-many relationship. Note that a many-to-many relationship is split into a pair of one-to-many relationships in a physical ERD. You will know what a physical ERD is in the next section.



Relational Model - Constraints

In modeling the design of the relational database we can put some restrictions like what values are allowed to be inserted in the relation, and what kind of modifications and deletions are allowed in the relation. These are the restrictions we impose on the relational database.

Constraints can be categorized into 3 main categories:

1. Constraints that are applied in the data model are called Implicit Constraints.
2. Constraints that are directly applied in the schemas of the data model, by specifying them in the DDL(Data Definition Language). These are called as Schema-Based Constraints or Explicit Constraints.
3. Constraints that cannot be directly applied in the schemas of the data model. We call these Application based or Semantic Constraints.

The relational model is a database model that represents the database as a collection of relations. Constraints are used to ensure the quality of the data in the database. There are four types of constraints: **Domain constraints**, **Key constraints**, **Entity Integrity Constraints** and **Referential integrity constraints** .

1. Domain Constraints

1. Every domain must contain atomic values (smallest indivisible units) which means composite and multi-valued attributes are not allowed.
2. We perform a datatype check here, which means when we assign a data type to a column, we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we can't give it values other than int datatype.

Example:

EID	Name	Phone
01	Lakshita Sharma	1200000019 7444444444

Explanation: In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

2. Key constraints ensure that every tuple in the relation is unique. A relation can have multiple keys or candidate keys (minimal superkey), out of which we choose one of the keys as the primary key. Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint .

Example:

EID	Name	Phone
01	Reyansh	7000000210
02	Deva	9400000001
01	Ravi	9123345454

Explanation: In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

3. Referential integrity constraints are specified between two tables. In these 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 .

Example:

EID	Name	DNO
01	Dilip	12
02	Dinda	22
04	Rohan	14

DNO	Place
12	Bangalore
13	Kolkata
14	Delhi

Explanation: In the above tables, the DNO of Table 1 is the foreign key, and DNO in Table 2 is the primary key. DNO = 22 in the foreign key of Table 1 is not allowed because DNO = 22 is not defined in the primary key of table 2. Therefore, Referential integrity constraints are violated here.

4. **Entity Integrity Constraints:**
    1. Entity Integrity constraints say that no primary key can take a NULL value, since using the primary key we identify each tuple uniquely in a relation.

Example:

EID	Name	Phone
01	Bithika	9001200000
02	Sinha	6000000009
NULL	Smita	9777777892

Explanation: In the above relation, EID is made the primary key, and the primary key can’t take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

Data modeling languages

SQL

While understanding SQL analytics functions clearly provides the user with fine-grained data manipulation capabilities, it also requires the highest level of technical skill and training. SQL usually requires external tools or pipelines for control flow and data integration.

LookML

Google Cloud's modeling language, LookML, is written as code. As such, it is designed to be used by technical users and developers with a strong understanding of SQL and database design principles. With LookML, developers create what Looker calls an 'explore.' An 'explore' is a projection of data from existing views and joins—analogous to a dbt model or ThoughtSpot worksheet.

dbt

This language also requires written code, but that is where the similarity with LookML ends. dbt data modeling focuses on a transformation-first approach, providing a templating language called Jinja—straightforward SQL statements, data testing, and DAGs for building pipelines and models. dbt is also developing an open semantic layer which is actively being integrated into multiple tools in the modern data stack including analytics, catalogs, and observability tools.

TML

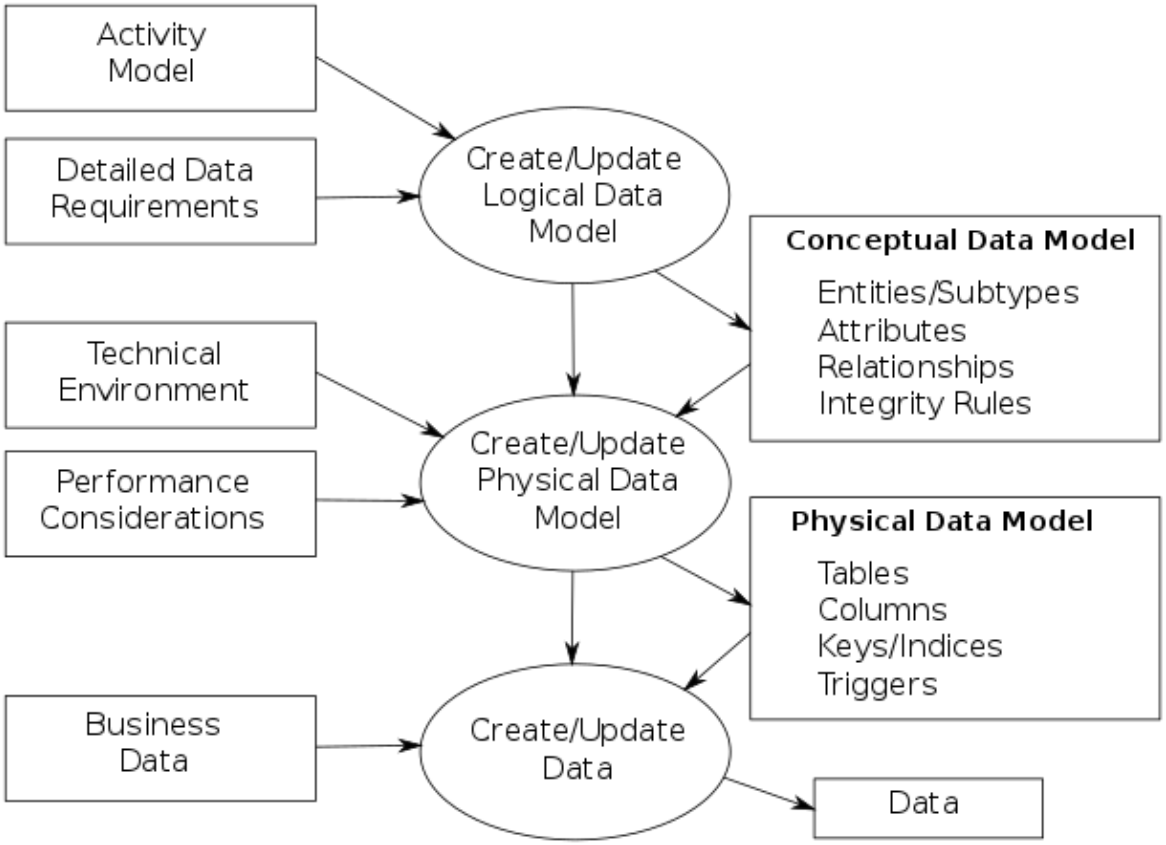
ThoughtSpot Modeling Language is a YAML base representation of all assets with ThoughtSpot including worksheets, views, SQL views, tables, answers, and Liveboards. TML approaches data modeling from a low-code and visual perspective. Designed to make data modeling and data visualization much more accessible for non-technical users, TML is created and edited with a low-code, AI-assisted web interface—reducing the technical demand on users and enabling self-service analytics.

Design of Data Modeling

Data modeling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures. It is used to illustrate the types of data used and stored within the system, the relationships among these data types, the ways the data can be grouped and organized, and its formats and attributes .
Data modeling employs standardized schemas and formal techniques, which provide a common, consistent, and predictable way of defining and managing data resources across an organization or even beyond . It is built around business needs, and rules and requirements are defined upfront through feedback from business stakeholders so they can be incorporated into the design of a new system or adapted in the iteration of an existing one .
Data modeling can be divided into three categories based on their degree of abstraction: **conceptual**, **logical**, and **physical** models. Each type of data model is discussed in more detail below :

- **Conceptual data models**: They offer a big-picture view of what the system will contain, how it will be organized, and which business rules are involved. They are usually created as part of the process of gathering initial project requirements. Typically, they include entity classes (defining the types of things that are important for the business to represent in the data model), their characteristics and constraints, the relationships between them, and relevant security and data integrity requirements. Any notation is typically simple.
- **Logical data models**: They provide greater detail about the concepts and relationships in the domain under consideration. They document structures of the data that can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models.
- **Physical data models**: They organize the data into tables, account for access, performance, and storage details.

In software engineering, **data modeling** is used to create a data model for an information system by applying certain formal techniques. It may be applied as part of broader Model-driven engineering (MDD) concept .



The data modeling process. The figure illustrates the way data models are developed and used today . A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.

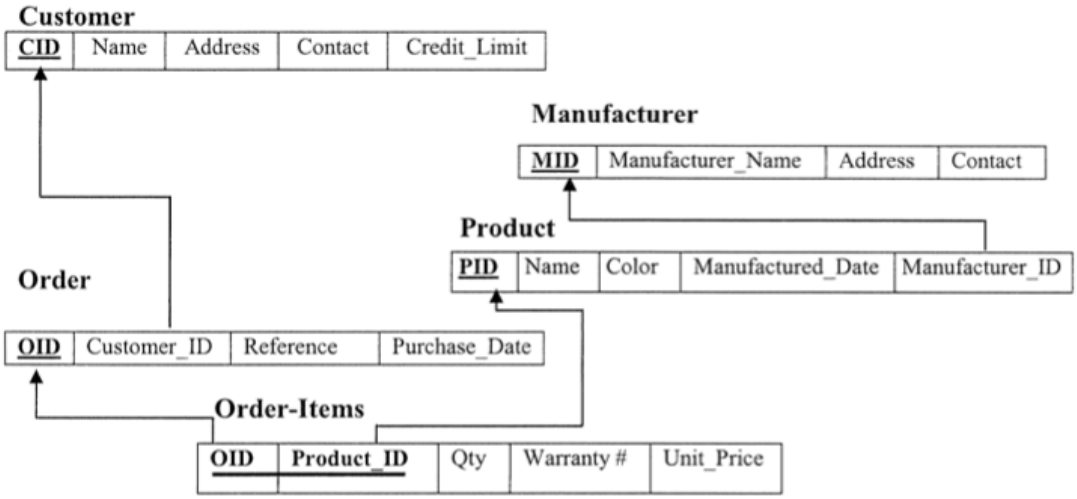
Relational Databases and Schemas

A database is a collection of interrelated data files or structures. It is designed to meet the various information needs of the organization. Also, it is integrated and shared. Thus, a relational database schema is an arrangement of relation states in such a manner that every relational database state fulfills the integrity constraints set on a relational database schema.

Relational Database and Schemas

As we know that a relational database schema is an arrangement of integrity constraints. Thus, in the context of relational database schema following points deserve a particular consideration:

1. A specific characteristic, that bears the same real-world concept, may appear in more than one relationship with the same or a different name. For example, in Employees relation, Employee Id (EmpId) is represented in Vouchers as AuthBy and PrepBy.
2. The specific real-world concept that appears more than once in a relationship should be represented by different names. For example, an employee is represented as subordinate or junior by using EmpId and as a superior or senior by using SuperId, in the employee's relation.
3. The integrity constraints that are specified on database schema shall apply to every database state of that schema.



Violation of constraints in relational database

There are mainly three operations that have the ability to change the state of relations, these modifications are given below:

1. **Insert –**
   To insert new tuples in a relation in the database.
2. **Delete –**
   To delete some of the existing relation on the database.
3. **Update (Modify) –**
   To make changes in the value of some existing tuples.

Whenever we apply the above modification to the relation in the database, the constraints on the relational database should not get violated.

Insert operation:
On inserting the tuples in the relation, it may cause violation of the constraints in the following way:

1. Domain constraint :
Domain constraint gets violated only when a given value to the attribute does not appear in the corresponding domain or in case it is not of the appropriate datatype.

Example:
Assume that the domain constraint says that all the values you insert in the relation should be greater than 10, and in case you insert a value less than 10 will cause you violation of the domain constraint, so gets rejected.

2. Entity Integrity constraint :
On inserting NULL values to any part of the primary key of a new tuple in the relation can cause violation of the Entity integrity constraint.

Example:
Insert (NULL, ‘Bikash, ‘M’, ‘Jaipur’, ‘123456’) into EMP

The above insertion violates the entity integrity constraint since there is NULL for the
primary key EID, it is not allowed, so it gets rejected.

3. Key Constraints :
On inserting a value in the new tuple of a relation which is already existing in another tuple of the same relation, can cause violation of Key Constraints.

Example:
Insert (’1200’, ‘Arjun’, ‘9976657777’, ‘Mumbai’) into EMPLOYEE

This insertion violates the key constraint if EID=1200 is already present in some tuple in the same relation, so it gets rejected.

Referential integrity :
On inserting a value in the foreign key of relation 1, for which there is no corresponding value in the Primary key which is referred to in relation 2, in such case Referential integrity is violated.

Example:
When we try to insert a value say 1200 in EID (foreign key) of table 1, for

which there is no corresponding EID (primary key) of table 2, then it causes violation, so gets rejected.

Solution that is possible to correct such violation is if any insertion violates any of the constraints, then the default action is to reject such operation.

Deletion operation:

On deleting the tuples in the relation, it may cause only violation of Referential integrity constraints.

Referential Integrity Constraints :

It causes violation only if the tuple in relation 1 is deleted which is referenced by foreign key from other tuples of table 2 in the database, if such deletion takes place then the values in the tuple of the foreign key in table 2 will become empty, which will eventually violate Referential Integrity constraint.

Solutions that are possible to correct the violation to the referential integrity due to deletion are listed below:

1. **Restrict –**
   Here we reject the deletion.
2. **Cascade –**
   Here if a record in the parent table(referencing relation) is deleted, then the corresponding records in the child table(referenced relation) will automatically be deleted.
3. **Set null or set default –**
   Here we modify the referencing attribute values that cause violation and we either set NULL or change to another valid value

Relational Algebra and Relational Calculus

Both Relational Algebra and Relational Calculus are formal query languages for a relational model. Both form the base for the SQL language which is used in most of the relational DBMSs.
Relational Algebra is a **procedural language**[1] that consists of a basic set of operations, which can be used for carrying out basic retrieval operations. The basic operation included in relational algebra are:

1. Select (σ)
2. Project (Π)
3. Union (U)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

1. Selection(σ): It is used to select required tuples of the relations.
Example:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

For the above relation, **σ(c>3)R** will select the tuples which have c more than 3.

A	B	C
1	2	4
4	3	4

Note: The selection operator only selects the required tuples but does not display them. For display, the data projection operator is used.
2. Projection(π): It is used to project required column data from a relation.
Example: Consider Table 1. Suppose we want columns B and C from Relation R. π(B,C)R will show following columns.

B	C
2	4

B	C
2	3
3	4

Note: By Default, projection removes duplicate data.

3. Union(U): Union operation in relational algebra is the same as union operation in set theory.

Example:

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Consider the following table of Students having different optional subjects in their course.

π(Student_Name)FRENCH U π(Student_Name)GERMAN

Student_Name
Ram
Mohan

Student_Name
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4. Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.

Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

π(Student_Name)FRENCH - π(Student_Name)GERMAN

Student_Name
Ram
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Set Intersection(∩): Set Intersection in relational algebra is the same set intersection operation in set theory.

Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

π(Student_Name)FRENCH ∩ π(Student_Name)GERMAN

Student_Name
Vivek
Geeta

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

6. Rename(ρ): Rename is a unary operation used for renaming attributes of a relation.

ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.

7. Cross Product(X): Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Example:

A

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

B

ID	Course
1	DS
2	DBMS

A X B

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: If A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘ n*m ‘ tuples.

Derived Operators

These are some of the derived operators, which are derived from the fundamental operators.

1. Natural Join(⋈)
2. Conditional Join

1. Natural Join(⋈): Natural join is a binary operator. Natural join between two or more relations will result in a set of all combinations of tuples where they have an equal common attribute.

Example:

EMP

Name	ID	Dept_Name
A	120	IT
B	125	HR
C	110	Sales
D	111	IT

DEPT

Dept_Name	Manager
Sales	Y
Production	Z
IT	A

Natural join between EMP and DEPT with condition :

EMP.Dept_Name = DEPT.Dept_Name

EMP ⋈ DEPT

Name	ID	Dept_Name	Manager
A	120	IT	A
C	110	Sales	Y
D	111	IT	A

2. Conditional Join: Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.

Example:

R

ID	Sex	Marks
1	F	45
2	F	55

ID	Sex	Marks
3	F	60

S

ID	Sex	Marks
10	M	20
11	M	22
12	M	59

Join between R and S with condition **R.marks >= S.marks**

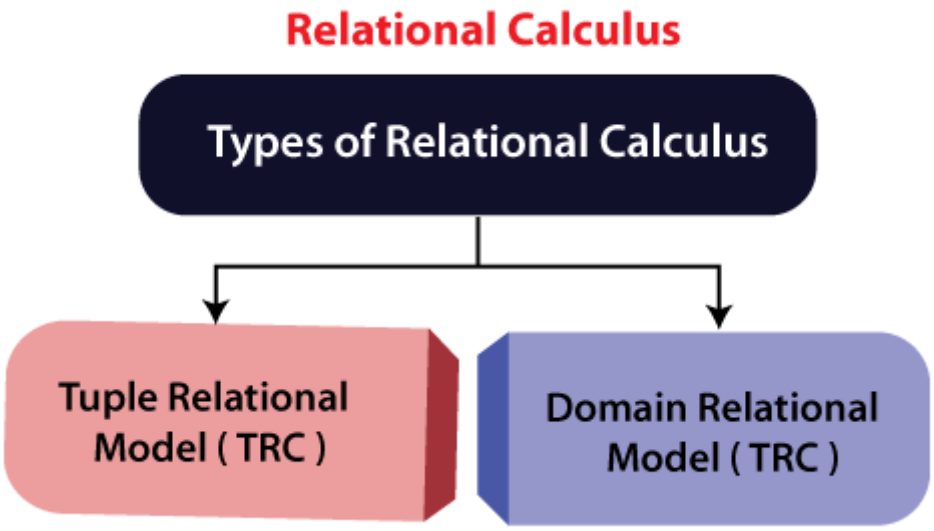
R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

In Relational Algebra, the order is specified in which the operations have to be performed[1]. Frameworks are created to implement the queries[1].
Relational Calculus is a **declarative language**[1] that provides declarative notations based on mathematical logic for specifying relational queries[3]. In Relational Calculus, the order is not specified in which the operation has to be performed[1]. Relational Calculus means what result we have to obtain[1]. Relational Calculus has two variations:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Relational Calculus is denoted as: { t | P (t) } Where, t: the set of tuples p: is the condition which is true for the given set of tuples[1].

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:

A Query in the tuple relational calculus is expressed as following notation

1. {T | P (T)} or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. { T.name | Author(T) AND T.article = 'database' }

Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1,a2 attributes
P stands for formula built by inner attributes

For example:

1. {< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

Codd's 12 Rules

Codd's 12 rules is a set of rules that a database management system (DBMS) must satisfy if it's to be considered *relational* (i.e. a relational DBMS).

The rules were proposed by Edgar F. Codd, who is considered a pioneer of the relational database model.

Codd's 12 rules is actually a set of thirteen rules, numbered from zero to twelve. The twelve rules are based on a single foundation rule — Rule Zero.

Codd's 12 rules are as follows.

Rule 0: The *foundation rule*:

> For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

Rule 1: The *information rule*:

> All information in a relational data base is represented explicitly at the logical level and in exactly one way — by values in tables.

Rule 2: The *guaranteed access rule*:

> Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

Rule 3: *Systematic treatment of null values*:

> Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Rule 4: *Dynamic online catalog based on the relational model*:

> The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Rule 5: The *comprehensive data sublanguage rule*:

> A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

- Data definition.
- View definition.
- Data manipulation (interactive and by program).
- Integrity constraints.
- Authorization.
- Transaction boundaries (begin, commit and rollback).

Rule 6: The *view updating rule*:

All views that are theoretically updatable are also updatable by the system.

Rule 7: *High-level insert, update, and delete*:

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

Rule 8: *Physical data independence*:

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

Rule 9: *Logical data independence*:

Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

Rule 10: *Integrity independence*:

Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

Rule 11: *Distribution independence*:

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

Rule 12: The *nonsubversion rule*:

If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).