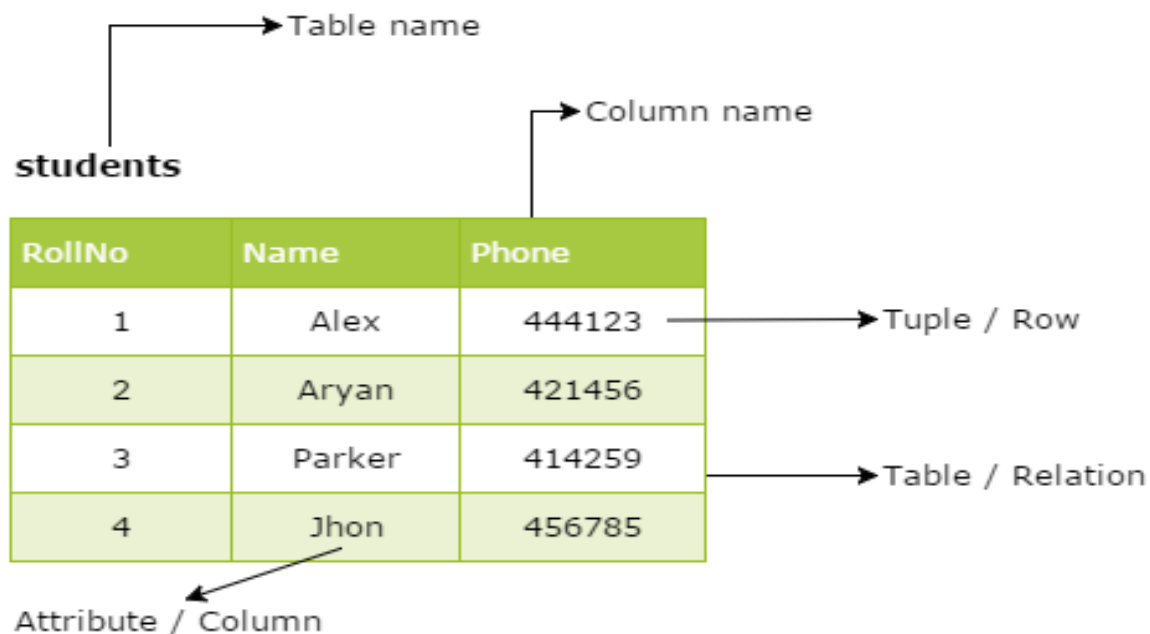


Relational Data Model

Basic Concepts

- Relational Data Model in DBMS: Concepts, Constraints, Example
- What is Relational Model?
- Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.
- The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.



Relational Model Terms

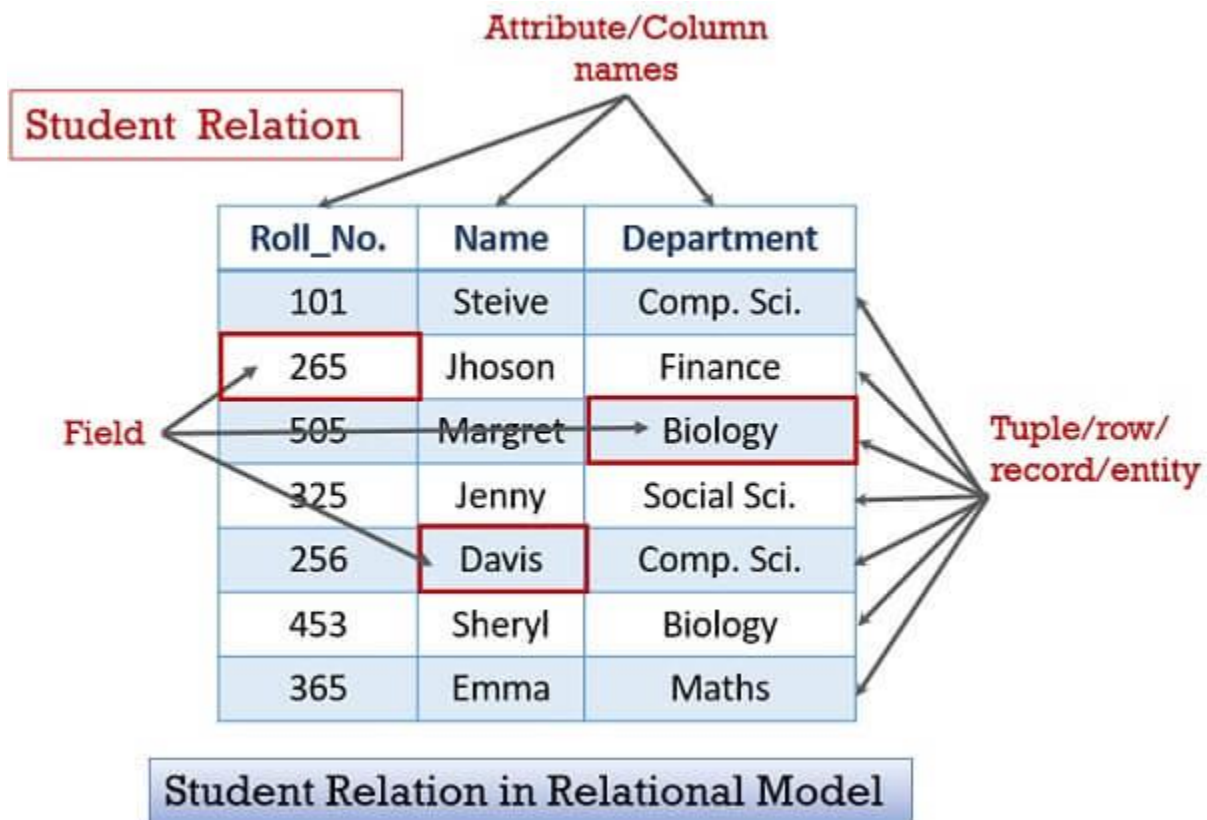
Relational Model Diagram

The figure below indicates a relation in a relational model.

Roll_No.	Name	Department
101	Steive	Comp. Sci.
265	Jhoson	Finance
505	Margret	Biology
325	Jenny	Social Sci.
256	Davis	Comp. Sci.
453	Sheryl	Biology
365	Emma	Maths

Student Relation in Relational Model

- It is a Student relation and it is having entries of 5 students (tuples) in it. The figure below will help you identify the relation, attributes, tuples and field in a relational model.



Roll_No.	Name	Department
101	Steive	Comp. Sci.
265	Jhoson	Finance
505	Margret	Biology
505	Margret	Biology
256	Davis	Comp. Sci.
453	Sheryl	Biology
365	Emma	Maths



Student Relation in Relational Model

Roll_No.	Name	Department	Roll_No.	Name	Department
101	Steive	Comp. Sci.	101	Steive	Comp. Sci.
265	Jhoson	Finance	265	Jhoson	Finance
	Charles		346	Charles	Finance
505	Margret	Biology	505	Margret	Biology
325	Jenny	Social Sci.	325	Jenny	Social Sci.
256	Davis	Comp. Sci.	256	Davis	Comp. Sci.
453	Sheryl	Biology	453	Sheryl	Biology
365	Emma	Maths	365	Emma	Maths



- **Advantages:**
- It is the simplest and easy to use, data model.
- It hides the physical storage details from the database developers and database users.
- It is scalable as you can keep adding records and attributes to records in a database.

Let us count points to understand the relational model in short:

- The database is a **set** of related **relations** (table of values).

- Each **relation** has a **name** which indicates what **type of tuples** the relation has. For example,

a relation named 'Student' indicates that it has student entities in it.

- Each relation has a **set of attributes** (column names) which represents, what type of information,

the entities or tuples have? For example, **Student** relation has a set of attributes **Roll_No., Name,**

Department. It indicates that the Student relation has student entities/tuples that have

information about their roll_no., name and department.

- A tuple (row) in a relation, is a real-world **entity**, it has a set of values for corresponding attributes.
- Each data value in a row or tuple is called **field**.

Relational Constraints

- [NOT NULL Constraint](#) – Ensures that a column cannot have NULL value.
- [DEFAULT Constraint](#) – Provides a default value for a column when none is specified.
- [UNIQUE Constraint](#) – Ensures that all values in a column are different.

- [PRIMARY Key](#) – Uniquely identifies each row/record in a database table.
- [FOREIGN Key](#) – Uniquely identifies a row/record in any of the given database table.
- [CHECK Constraint](#) – The CHECK constraint ensures that all the values in a column satisfies certain conditions

Relational Model Constraints

Relational model constraints are **restrictions** specified to the data values in the relational database. Initially, we will describe the **constraints on the database**, they are categorized as follows:

- **Inherent Model-Based Constraints:** The constraints that are **implicit** in a data model are inherent model-based constraints. For example, a relation in a database must not have duplicate tuples, there is no constraint in the ordering of the tuples and attributes.
- **Schema-Based Constraints:** The constraints that are specified while **defining the schema** of a database using DDL are schema-based constraints. They are further categorized as domain constraints, key constraints, entity integrity constraints, referential integrity constraints and constraints on Null Value.
- **Application-based Constraints:** The constraints that **cannot be applied while defining the database schema** are expressed in application programs. For example, the salary of an employee cannot be more than his supervisor.

Now let us explore the **Schema-based** constraints in detail:

- **Domain Constraints:**
Each attribute in a tuple is declared to be of a particular domain (for example,

integer, character, Boolean, String, etc.) which specifies a constraint on the values that an attribute can take.

- **Key Constraint and Constraint on Null Values:**

In relation, a key can either be a **single attribute** or a **subgroup of attributes** that can recognize a particular tuple in a relation. Now, the key constraint specifies that a key (attribute/subset of attribute) must not have the same set of values for the tuples in a relation.

The constraint on NULL values defines whether an attribute is allowed to carry Null value or not. For example, in a student tuple, its name attribute must be NOT NULL.

- **Entity Integrity Constraint:**

Entity integrity constraint specifies that a **primary key** of a tuple can never be **NULL**. As primary key used to identify individual tuple in a relation.

- **Referential Integrity Constraint:**

The referential integrity constraint holds if the foreign key of relation R1 that **refers to** the relation R2 satisfies following two conditions:

1. The set of attributes that form **foreign key** of relation R1 should have the **same domain** as the **primary key** of the referenced relation R2.
2. In the current state, the **set of values** of the **foreign key** in tuple t1 of relation R1 must **match** a **primary key** value in referenced relation R2 or it could be **NULL**.

Relational Algebra

RELATIONAL ALGEBRA is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ) Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is propositional logic

Example 1

```
 $\sigma_{\text{topic} = \text{"Database"}}(\text{Tutorials})$ 
```

Output - Selects tuples from Tutorials where topic = 'Database'.

Example 2

```
 $\sigma_{\text{topic} = \text{"Database"} \text{ and } \text{author} = \text{"guru99"}}(\text{Tutorials})$ 
```

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.

Example 3

```
 $\sigma_{\text{sales} > 50000}(\text{Customers})$ 
```

Output - Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (π) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
------------	--------------	--------

1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}(\text{Customers})$	
CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Union operation (\cup)

UNION is symbolized by \cup symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result $\leftarrow A \cup B$

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A			Table B	
column 1	column 2		column 1	column 2
1	1		1	1
1	2		1	3

$A \cup B$ gives

Table $A \cup B$	
column 1	column 2
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A-B

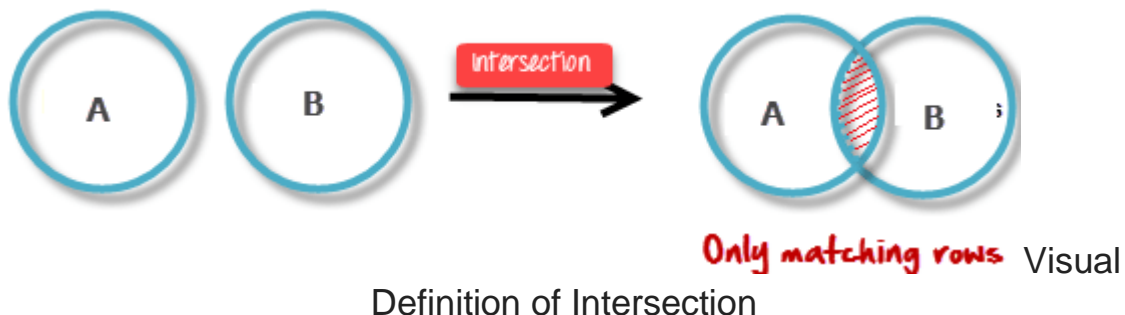
Table A - B	
column 1	column 2
1	2

Intersection

An intersection is defined by the symbol \cap

$A \cap B$

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Example:

A \cap B

Table A \cap B

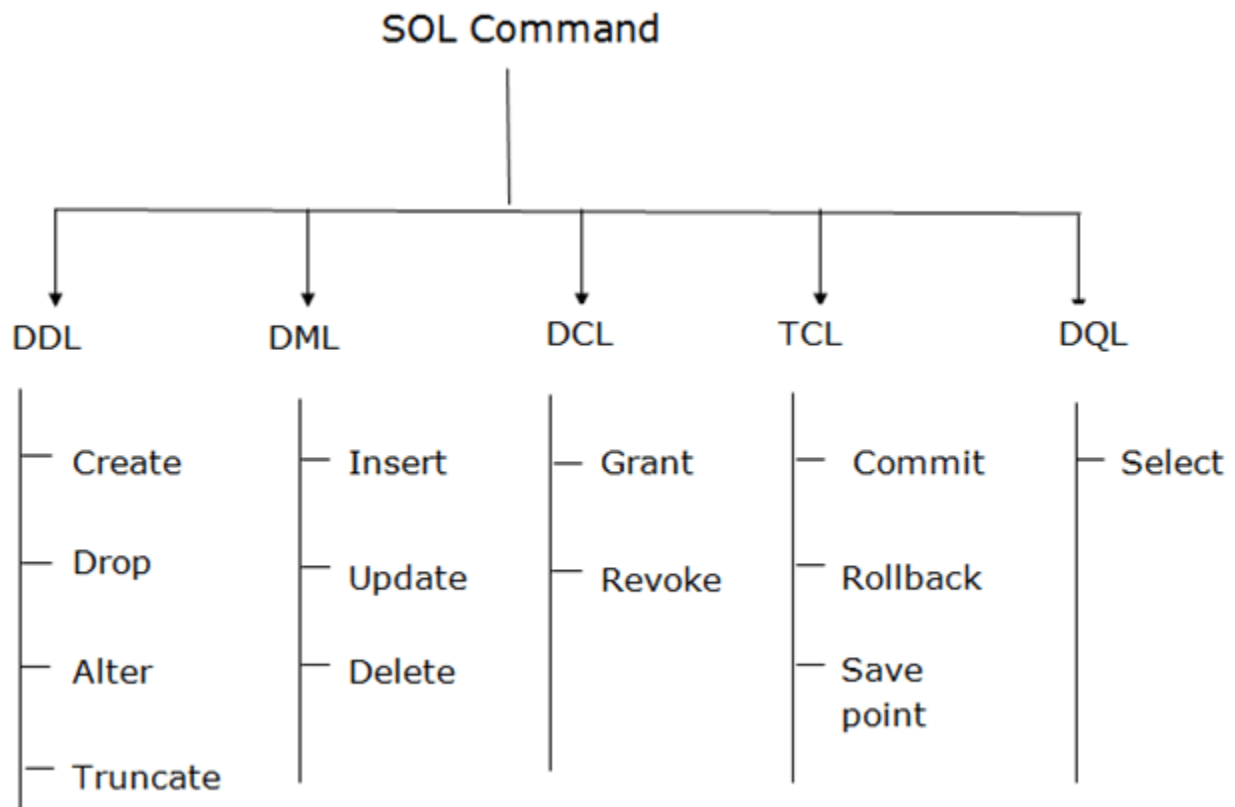
column 1	column 2
1	1

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP

a. CREATE It is used to create a new table in the database.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

1. DROP TABLE ;

Example

1. DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

1. ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE MODIFY(COLUMN DEFINITION....);

EXAMPLE

1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

1. INSERT INTO TABLE_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, valueN);

Or

1. INSERT INTO TABLE_NAME
2. VALUES (value1, value2, value3, valueN);

For example:

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:

1. UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

For example:

1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'

c. DELETE: It is used to remove one or more row from a table.

Syntax:

1. DELETE FROM table_name [WHERE condition];

For example:

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

SQL Commands: DML, DDL, DCL, TCL, DQL with Query Example

What is SQL?

SQL is a database language designed for the retrieval and management of data in a relational database.

SQL is the standard language for database management. All the RDBMS systems like MySQL, MS Access, Oracle, Sybase, Postgres, and SQL Server use SQL as their standard database language. SQL programming language uses various commands for different operations. We will learn about the like DCL, TCL, DQL, DDL and DML commands in SQL with examples.

In this SQL commands in DBMS tutorial, you will learn:

- [What is SQL?](#)
- [Why Use SQL?](#)
- [Brief History of SQL](#)
- [Types of SQL](#)
- [What is DDL?](#)
- [What is Data Manipulation Language?](#)
- [What is DCL?](#)
- [What is TCL?](#)
- [What is DQL?](#)

Why Use SQL?

Here, are important reasons for using SQL

- It helps users to access data in the RDBMS system.
- It helps you to describe the data.
- It allows you to define the data in a database and manipulate that specific data.
- With the help of SQL commands in DBMS, you can create and drop databases and tables.
- SQL offers you to use the function in a database, create a view, and stored procedure.
- You can set permissions on tables, procedures, and views.

Brief History of SQL

Here, are important landmarks from the history of SQL:

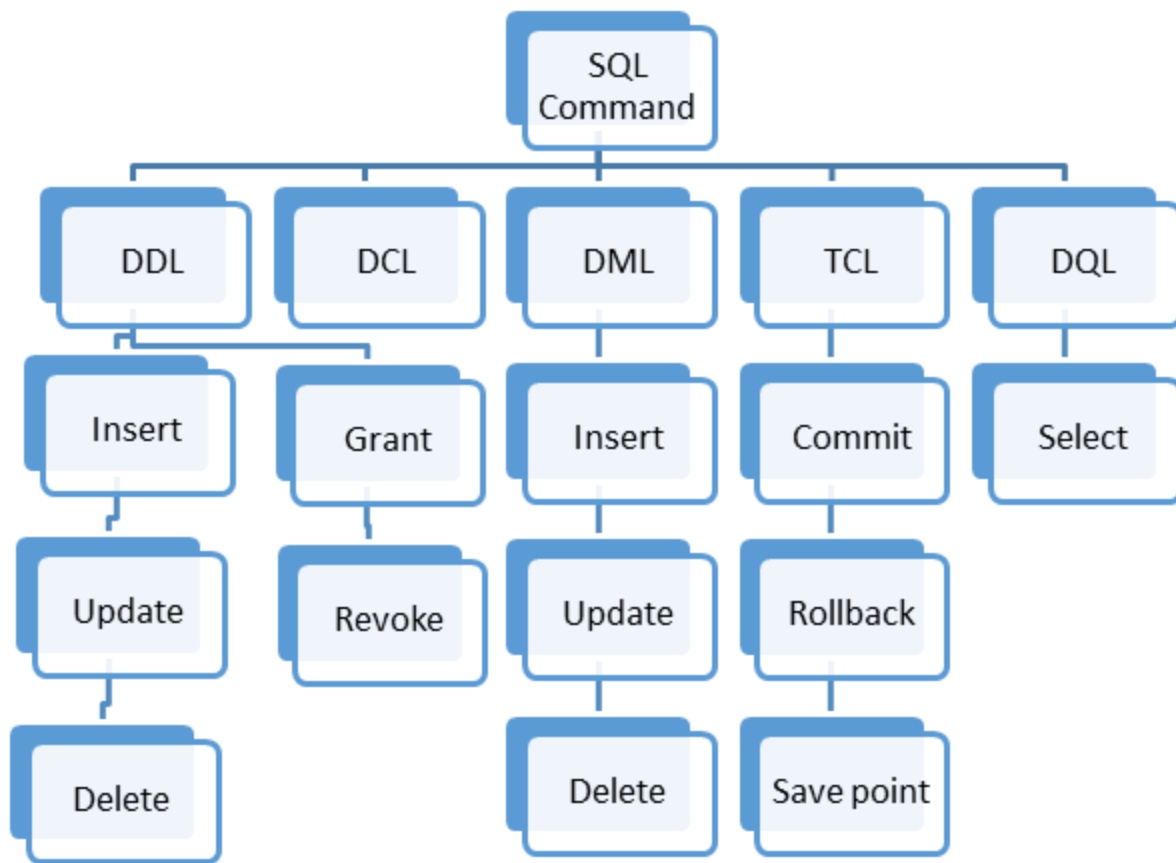
- 1970 - Dr. Edgar F. "Ted" Codd described a relational model for databases.
- 1974 - Structured Query Language appeared.

- 1978 - IBM released a product called System/R.
- 1986 - IBM developed the prototype of a relational database, which is standardized by ANSI.
- 1989- First ever version launched of SQL
- 1999 - SQL 3 launched with features like triggers, object-orientation, etc.
- SQL2003- window functions, XML-related features, etc.
- SQL2006- Support for XML Query Language
- SQL2011-improved support for temporal databases

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)
- Data Query Language (DQL)



Types of SQL

Let see each of them in detail:

What is DDL?

Data Definition Language helps you to define the database structure or schema. Let's learn about DDL commands with syntax.

Five types of DDL commands in SQL are:

CREATE

CREATE statements is used to define the database structure schema:

Syntax:

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

For example:

```
Create database university;  
Create table students;  
Create view for_students;
```

DROP

Drops commands remove tables and databases from RDBMS.

Syntax

```
DROP TABLE ;
```

For example:

```
Drop object_type object_name;  
Drop database university;  
Drop table student;
```

ALTER

Alters command allows you to alter the structure of the database.

Syntax:

To add a new column in the table

```
ALTER TABLE table_name ADD column_name COLUMN-definition;
```

To modify an existing column in the table:

```
ALTER TABLE MODIFY(COLUMN DEFINITION....);
```

For example:

```
Alter table guru99 add subject varchar;
```

TRUNCATE:

This command used to delete all the rows from the table and free the space containing the table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE table students;
```

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

- INSERT
- UPDATE
- DELETE

INSERT:

This is a statement is a SQL query. This command is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME (col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');
```

UPDATE:

This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
UPDATE students  
SET FirstName = 'Jhon', LastName= 'Wick'  
WHERE StudID = 3;
```

DELETE:

This command is used to remove one or more rows from a table.

Syntax:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM students  
WHERE FirstName = 'Jhon';
```

What is DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions." Other permission controls parameters of the database system.

Examples of DCL commands:

Commands that come under DCL:

- Grant
- Revoke

Grant:

This command is use to give user access privileges to a database.

Syntax:

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

For example:

```
GRANT SELECT ON Users TO 'Tom'@'localhost';
```

Revoke:

It is useful to back permissions from the user.

Syntax:

```
REVOKE privilege_name ON object_name FROM {user_name | PUBLIC | role_name}
```

For example:

```
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;
```

What is TCL?

Transaction control language or TCL commands deal with the transaction within the database.

Commit

This command is used to save all the transactions to the database.

Syntax:

```
Commit;
```

For example:

```
DELETE FROM Students  
WHERE RollNo =25;  
COMMIT;
```

Rollback

Rollback command allows you to undo transactions that have not already been saved to the database.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM Students  
WHERE RollNo =25;
```

SAVEPOINT

This command helps you to sets a savepoint within a transaction.

Syntax:

```
SAVEPOINT SAVEPOINT_NAME;
```

Example:

```
SAVEPOINT RollNo;
```

What is DQL?

Data Query Language (DQL) is used to fetch the data from the database. It uses only one command:

SELECT:

This command helps you to select the attribute based on the condition described by the WHERE clause.

Syntax:

```
SELECT expressions  
FROM TABLES  
WHERE conditions;
```

For example:

```
SELECT FirstName  
FROM Student  
WHERE RollNo > 15;
```

Summary:

- SQL is a database language designed for the retrieval and management of data in a relational database.
- It helps users to access data in the RDBMS system
- In the year 1974, the term Structured Query Language appeared
- Five types of SQL queries are 1) Data Definition Language (DDL) 2) Data Manipulation Language (DML) 3) Data Control Language(DCL) 4) Transaction Control Language(TCL) and, 5) Data Query Language (DQL)
- Data Definition Language(DDL) helps you to define the database structure or schema.
- Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data.
- DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions."
- Transaction control language or TCL commands deal with the transaction within the database.
- Data Query Language (DQL) is used to fetch the data from the [database](#)

SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Show Examples

Sr.No.	Operator & Description
1	ALL The ALL operator is used to compare a value to all values in another value set.
2	AND

	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	BETWEEN The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	IN The IN operator is used to compare a value to a list of literal values that have been specified.
7	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
8	NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
9	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	IS NULL The NULL operator is used to compare a value with a NULL value.

11	UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).
----	--

SQL Logical Operators

Here is a list of all the logical operators available in SQL.

[Show Examples](#)

Sr.No.	Operator & Description
1	ALL The ALL operator is used to compare a value to all values in another value set.
2	AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	BETWEEN The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	IN

	The IN operator is used to compare a value to a list of literal values that have been specified.
7	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
8	NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
9	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	IS NULL The NULL operator is used to compare a value with a NULL value.
11	UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Introduction to SQL Clauses

MySQL queries are SQL functions that help us to access a particular set of records from a database table. We can request any information or data from the database using the clauses or let's say SQL statements. SQL Clauses receives a conditional expression that can be a column name or valid term involving columns where this supports the MySQL functions to calculate the result values for a table in the database.

1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

1. SELECT column
2. FROM table_name
3. WHERE conditions
4. GROUP BY column
5. ORDER BY column

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150

Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example:

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

```
Com1    5
Com2    3
Com3    2
```

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

1. SELECT column1, column2
2. FROM table_name
3. WHERE conditions
4. GROUP BY column1, column2
5. HAVING conditions
6. ORDER BY column1, column2;

Example:

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST

3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

```
Com1    5
Com2    3
```

3. ORDER BY

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

Syntax:

1. SELECT column1, column2
2. FROM table_name
3. WHERE condition
4. ORDER BY column1, column2... ASC|DESC;

Where

ASC: It is used to sort the result set in ascending order by expression.

DESC: It sorts the result set in descending order by expression.

Example: Sorting Results in Ascending Order

Table:

CUSTOMER

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
23	David	Bangkok
34	Alina	Dubai

45	John	UK
56	Harry	US

Enter the following SQL statement:

1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME;

Output:

CUSTOMER_ID	NAME	ADDRESS
34	Alina	Dubai
23	David	Bangkok
56	Harry	US
45	John	UK
12	Kathrin	US

Example: Sorting Results in Descending Order

Using the above CUSTOMER table

1. SELECT *
2. FROM CUSTOMER
3. ORDER BY NAME DESC;

Output:

CUSTOMER_ID	NAME	ADDRESS
12	Kathrin	US
45	John	UK
56	Harry	US
23	David	Bangkok
34	Alina	Dubai

2.5. WHERE Clause

The WHERE clause defines the criteria to limit the records affected by SELECT, UPDATE, and DELETE statements.

The general form of the WHERE is:

- WHERE [criteria](#)

2.6. GROUP BY Clause

The GROUP BY clause denotes that rows should be grouped according to the specified expression values. One row will be returned for each group, after optionally filtering those aggregate rows based on a HAVING clause.

The general form of the GROUP BY is:

- GROUP BY expression (,expression)*

Syntax Rules:

- Column references in the group by clause must be to unaliased output columns.

- Expressions used in the group by must appear in the select clause.
- Column references and expressions in the select clause that are not used in the group by clause must appear in aggregate functions.
- If an aggregate function is used in the SELECT clause and no GROUP BY is specified, an implicit GROUP BY will be performed with the entire result set as a single group. In this case, every column in the SELECT must be an aggregate function as no other column value will be fixed across the entire group.
- The group by columns must be of a comparable type.

2.7. HAVING Clause

The HAVING clause operates exactly as a WHERE clause although it operates on the output of a GROUP BY. It supports the same syntax as the WHERE clause.

Syntax Rules:

- Expressions used in the group by clause must either contain an aggregate function: COUNT, AVG, SUM, MIN, MAX. or be one of the grouping expressions.

2.8. ORDER BY Clause

The ORDER BY clause specifies how records should be sorted. The options are ASC (ascending) and DESC (descending).

Usage:

```
ORDER BY expression [ASC|DESC] [NULLS (FIRST|LAST)], ...
```

Syntax Rules:

- Sort columns may be specified positionally by a 1-based positional integer, by SELECT clause alias name, by SELECT clause expression, or by an unrelated expression.
- Column references may appear in the SELECT clause as the expression for an aliased column or may reference columns from tables in the FROM clause. If the column reference is not in the SELECT clause the query must not be a set operation, specify SELECT DISTINCT, or contain a GROUP BY clause.
- Unrelated expressions, expressions not appearing as an aliased expression in the select clause, are allowed in the order by clause of a non-set QUERY. The columns referenced in the expression must come

from the from clause table references. The column references cannot be to alias names or positional.

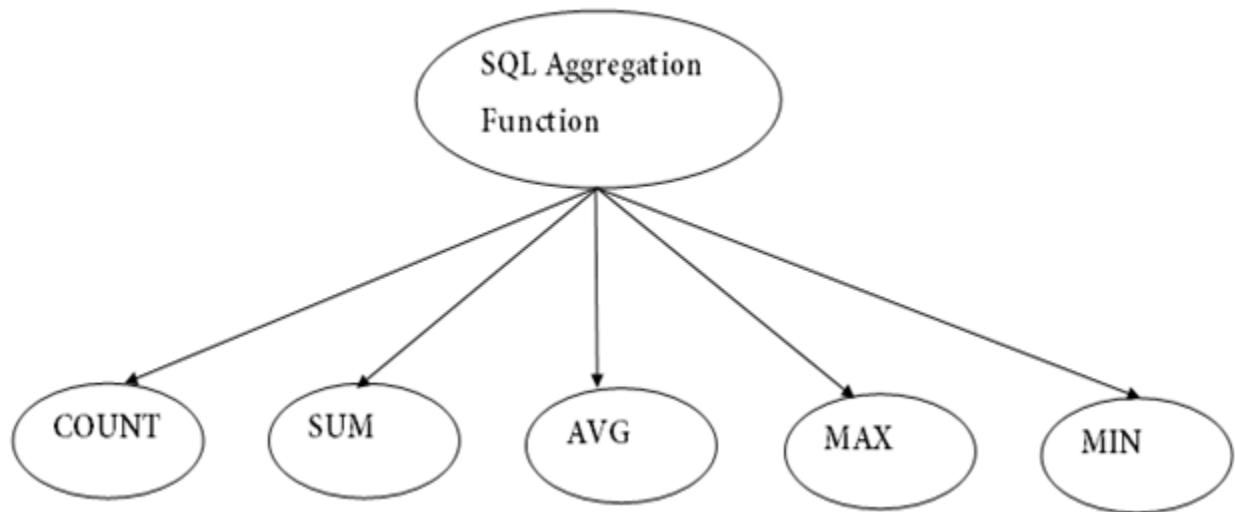
- The ORDER BY columns must be of a comparable type.
- If an ORDER BY is used in an inline view or view definition without a limit clause, it will be removed by the Teiid optimizer.
- If NULLS FIRST/LAST is specified, then nulls are guaranteed to be sorted either first or last. If the null ordering is not specified, then results will typically be sorted with nulls as low values, which is Teiid's internal default sorting behavior. However not all sources return results with nulls sorted as low values by default, and Teiid may return results with different null orderings.

The use of positional ordering is no longer supported by the ANSI SQL standard and is a deprecated feature in Teiid. It is preferable to use alias names in the order by clause.

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

1. COUNT(*)
2. or
3. COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75

Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;

Output:

```
10
```

Example: COUNT with WHERE

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE >= 20;

Output:

```
7
```

Example: COUNT() with DISTINCT

1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;

Output:

Example: COUNT() with GROUP BY

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

Com1	5
Com2	3
Com3	2

Example: COUNT() with HAVING

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

Com1	5
Com2	3

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

1. SUM()
2. or
3. SUM([ALL|DISTINCT] expression)

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

Output:

```
670
```

Example: SUM() with WHERE

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>**3**;

Output:

```
320
```

Example: SUM() with GROUP BY

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>**3**
4. GROUP BY COMPANY;

Output:

```
Com1    150  
Com2    170
```

Example: SUM() with HAVING

1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=**170**;

Output:

```
Com1    335  
Com3    170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

1. AVG()
2. or
3. AVG([ALL|DISTINCT] expression)

Example:

1. SELECT AVG(COST)
2. FROM PRODUCT_MAST;

Output:

```
67.00
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
2. FROM PRODUCT_MAST;

```
30
```

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

1. MIN()

2. or
3. MIN([ALL|DISTINCT] expression)

Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT_MAST;

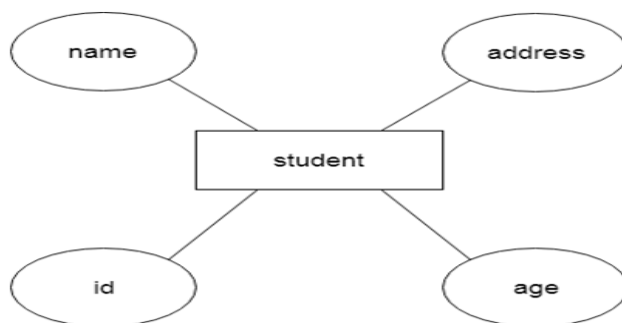
Output:

10

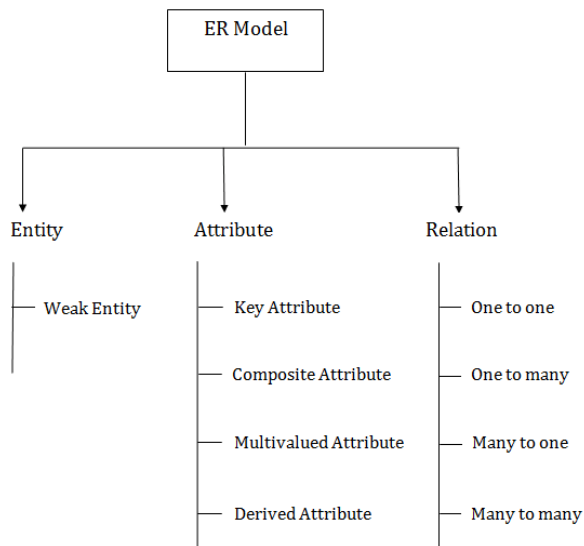
ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



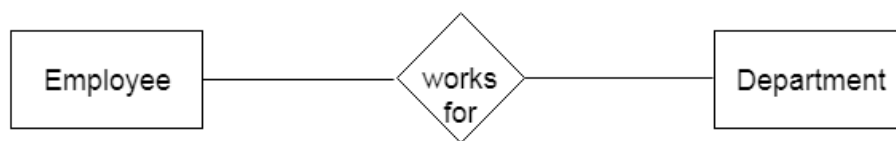
Component of ER Diagram



1. Entity:

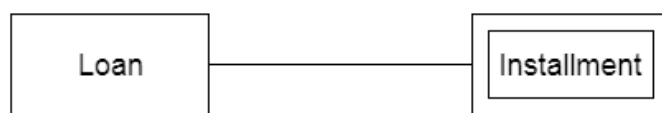
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

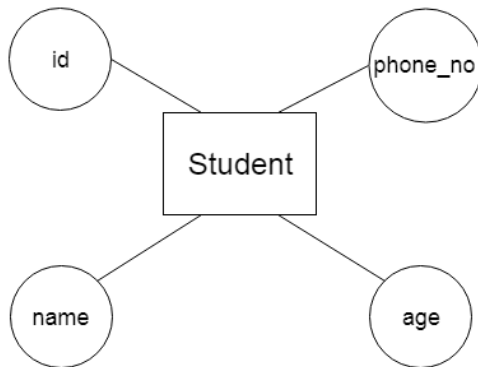
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

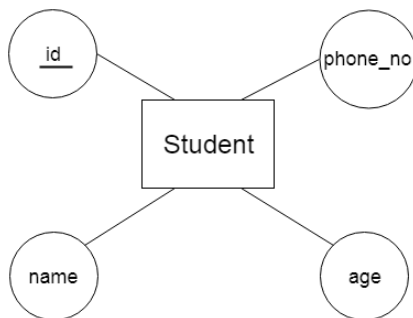
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



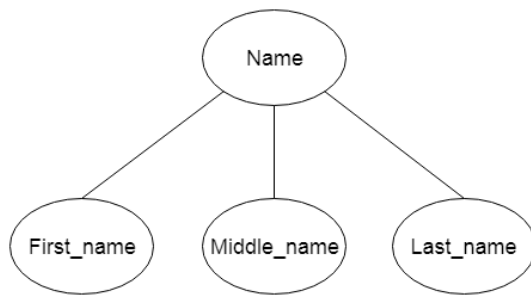
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

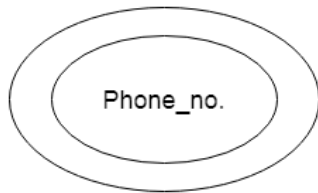
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

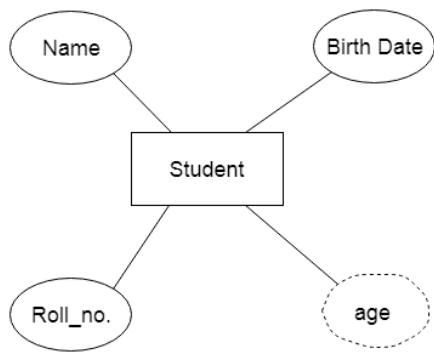
For example, a student can have more than one phone number.



d. Derived Attribute

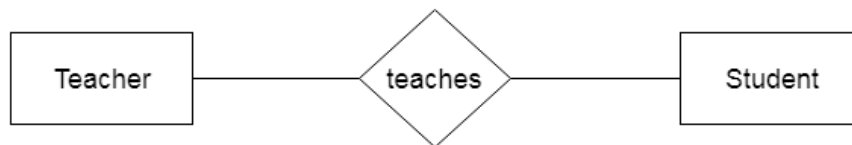
An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

a. One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

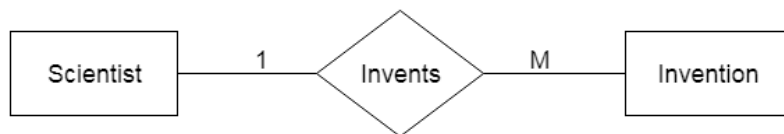
For example, A female can marry to one male, and a male can marry to one female.



b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

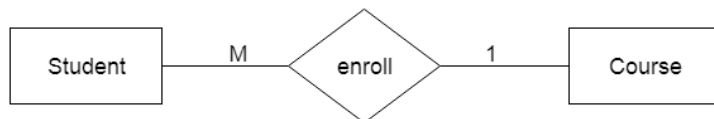
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

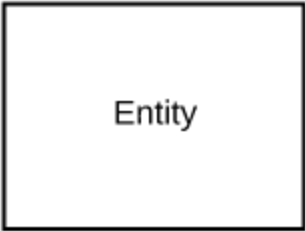
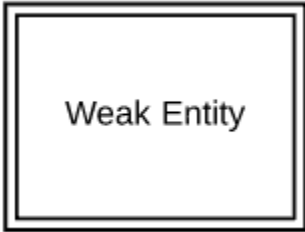
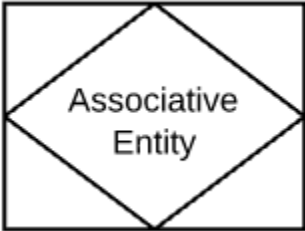
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.



Conceptual ER diagram symbols

Conceptual Data Models establish a broad view of what should be included in the model set. Conceptual ERDs can be used as the

Entity Symbol	Name	Description
	Strong entity	These shapes are independent from other entities, and are often called parent entities, since they will often have weak entities that depend on them. They will also have a primary key, distinguishing each occurrence of the entity.
	Weak entity	Weak entities depend on some other entity type. They don't have primary keys, and have no meaning in the diagram without their parent entity.
	Associative entity	Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

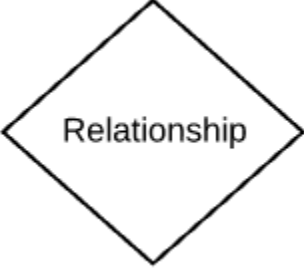

foundation for logical data models. They may also be used to form commonality relationships between ER models as a basis for data model integration. All of the symbols shown below are found in the UML Entity Relationship and Entity Relationship shape library of Lucidchart.

ERD entity symbols

Entities are objects or concepts that represent important data. Entities are typically nouns such as product, customer, location, or promotion. There are three types of entities commonly used in entity relationship diagrams.

ERD relationship symbols

Within entity-relationship diagrams, relationships are used to document the interaction between two entities. Relationships are usually verbs such as assign, associate, or track and provide useful information that could not be discerned with just the entity types.

Relationship Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.
	Weak relationship	Weak Relationships are connections between a weak entity and its owner.

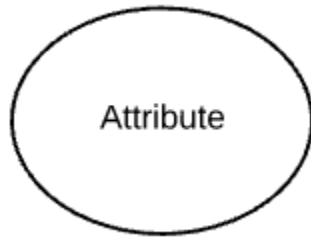
ERD attribute symbols

ERD attributes are characteristics of the entity that help users to better understand the database. Attributes are included to include details of the various entities that are highlighted in a conceptual ER diagram.

Attribute Symbol

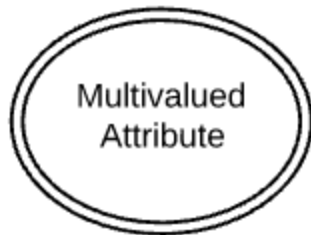
Name

Description



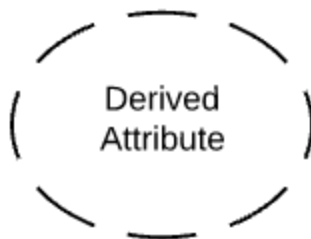
Attribute

Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship.



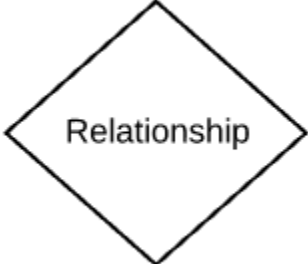
Multivalued
attribute

Multivalued attributes are those that are can take on more than one value.



Derived
attribute

Derived attributes are attributes whose value can be calculated from related attribute values.

Attribute Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.

Diagramming is quick and easy with Lucidchart. Start a free trial today to start creating and collaborating.

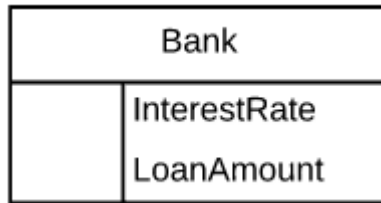
Physical ER diagram symbols

The physical data model is the most granular level of entity-relationship diagrams, and represents the process of adding information to the database. Physical ER models show all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

As shown below, tables are another way of representing entities. The key parts of Entity-relationship Tables are:

Fields

Fields represent the portion of a table that establish the attributes of the entity. Attributes are typically thought of as columns in the database that the ERD models.



In the image above, InterestRate and LoanAmount are both attributes of the entity that are contained as fields.

Keys

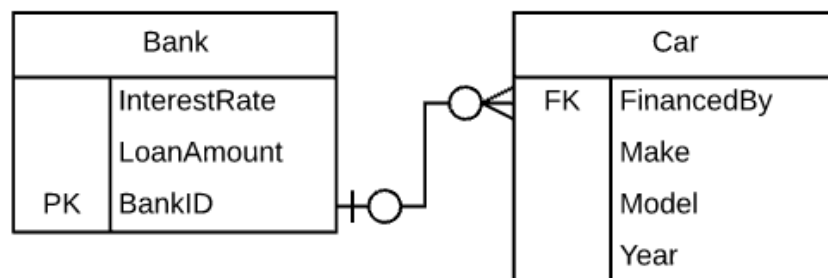
Keys are one way to categorize attributes. ER diagrams help users to model their databases by using various tables that ensure that the database is organized, efficient, and fast. Keys are used to link various tables in a database to each other in the most efficient way possible.

Primary Keys

Primary keys are an attribute or combination of attributes that uniquely identifies one and only one instance of an entity.

Foreign Keys

Foreign keys are created any time an attribute relates to another entity in a one-to-one or one-to-many relationship.

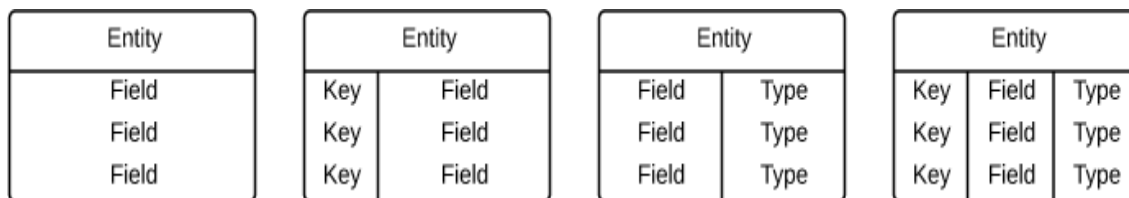


Each car can only be financed by one bank, therefore the primary key BankId from the Bank table is used as the foreign key FinancedBy in

the Car table. This BankID is able to be used as the foreign key for multiple cars.

Types

Types refer to the type of data in the corresponding field in a table. Types can also refer to entity types, which describe the composition of an entity; e.g., a book's entity types are author, title, and published date.



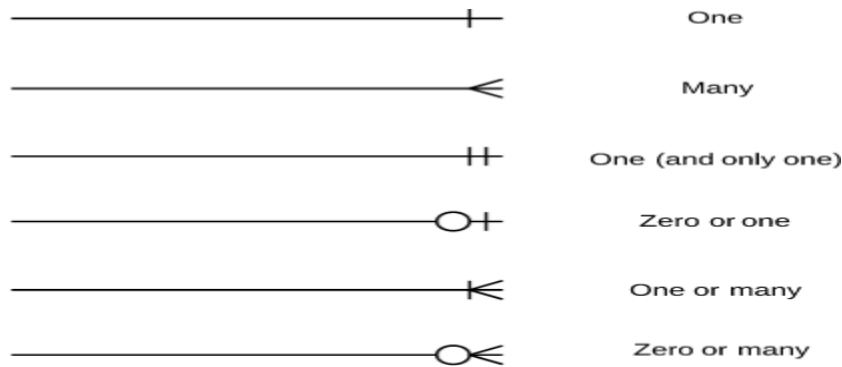
ER diagram notation

While crow's foot notation is often recognized as the most intuitive style, some use OMT, IDEF, Bachman, or UML notation, according to their preferences. Crow's foot notation, however, has an intuitive graphic format, making it the preferred ERD notation for Lucidchart. Consider using one of our [Crow Foot ER Diagram templates](#).

Cardinality and ordinality

Cardinality refers to the maximum number of times an instance in one entity can relate to instances of another entity. Ordinality, on the other hand, is the minimum number of times an instance in one entity can be associated with an instance in the related entity.

Cardinality and ordinality are shown by the styling of a line and its endpoint, according to the chosen notation style.



ER diagram of Library Management System

- Difficulty Level : [Basic](#)
- Last Updated : 28 Sep, 2020

[ER Diagram](#) is known as Entity-Relationship Diagram, it is used to analyze to the structure of the Database. It shows relationships between entities and their attributes. An ER Model provides a means of communication.

The Library Management System database keeps track of readers with the following considerations –

- The system keeps track of the staff with a single point authentication system comprising login Id and password.
- Staff maintains the book catalog with its ISBN, Book title, price(in INR), category(novel, general, story), edition, author Number and details.
- A publisher has publisher Id, Year when the book was published, and name of the book.
- Readers are registered with their user_id, email, name (first name, last name), Phone no (multiple entries allowed), communication address. The staff keeps track of readers.
- Readers can return/reserve books that stamps with issue date and return date. If not returned within the prescribed time period, it may have a due date too.
- Staff also generate reports that has readers id, registration no of report, book no and return/issue info.

- **Authentication System Entity** : It has LoginId and password with LoginID as Primary Key.
- **Reports Entity** : It has UserId, Reg_no, Book_no, Issue/Return date. Reg_no is the Primary Key of reports entity.
- **Staff Entity** : It has name and staff_id with staff_id as Primary Key.
- **Reserve/Return Relationship Set** : It has three attributes: Reserve date, Due date, Return date.

Relationships between Entities –

- A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.
- A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.
- Staff keeps track of readers. The relationship is M:N.
- Staff maintains multiple reports. The relationship 1:N.
- Staff maintains multiple Books. The relationship 1:N.
- Authentication system provides login to multiple staffs. The relation is 1:N.

Extended Entity-Relationship (EE-R) Model

RDBMSMCADatabase

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

In addition to ER model concepts EE-R includes –

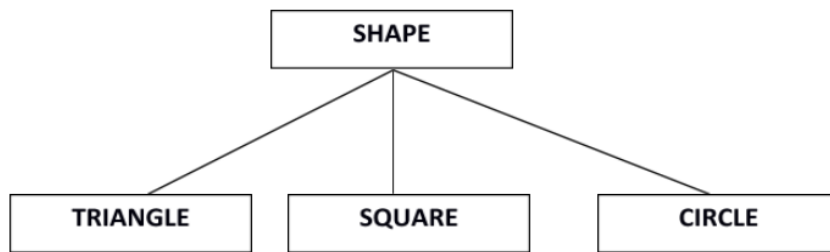
- Subclasses and Super classes.
- Specialization and Generalization.
- Category or union type.
- Aggregation.

These concepts are used to create EE-R diagrams.

Subclasses and Super class

Super class is an entity that can be divided into further subtype.

For **example** – consider Shape super class.

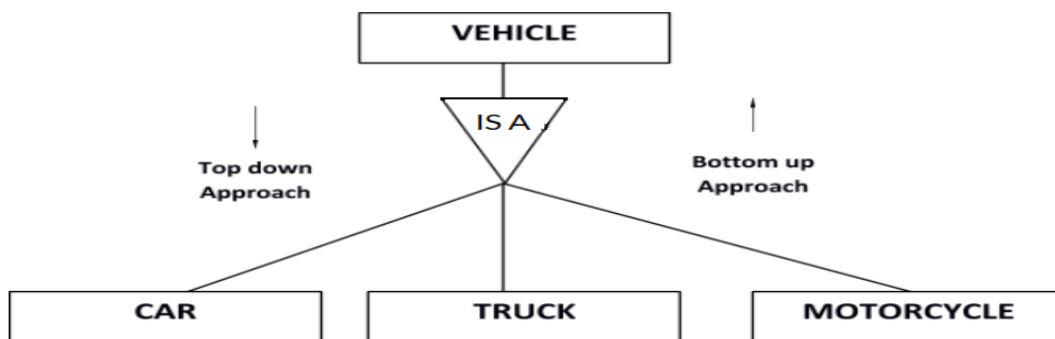


Super class shape has sub groups: Triangle, Square and Circle.

Sub classes are the group of entities with some unique attributes. Sub class inherits the properties and attributes from super class.

Specialization and Generalization

Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities.



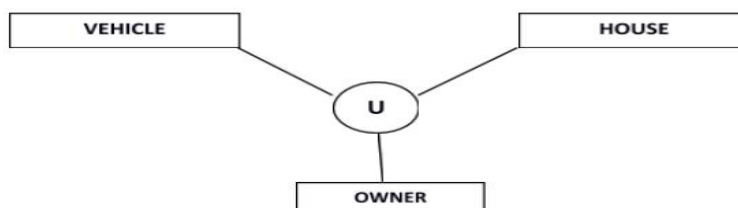
It is a Bottom up process i.e. consider we have 3 sub entities Car, Truck and Motorcycle. Now these three entities can be generalized into one super class named as Vehicle.

Specialization is a process of identifying subsets of an entity that share some different characteristic. It is a top down approach in which one entity is broken down into low level entity.

In above example Vehicle entity can be a Car, Truck or Motorcycle.

Category or Union

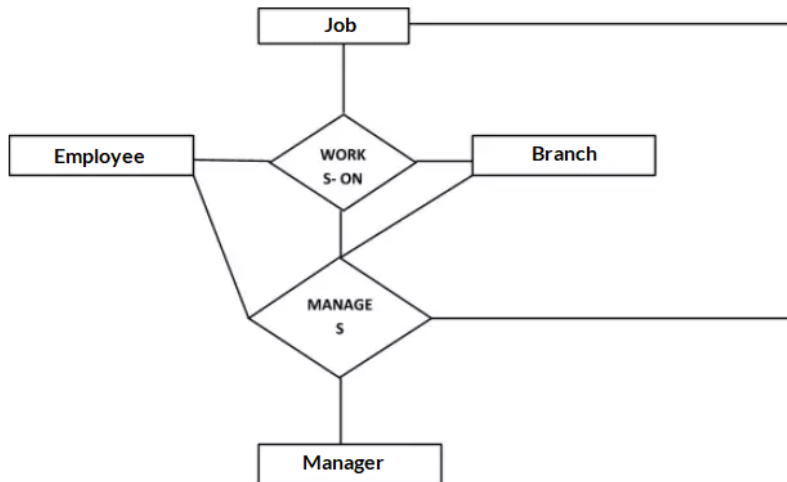
Relationship of one super or sub class with more than one super class.



Owner is the subset of two super class: Vehicle and House.

Aggregation

Represents relationship between a whole object and its component.



Consider a ternary relationship Works_On between Employee, Branch and Manager. Now the best way to model this situation is to use aggregation, So, the relationship-set, Works_On is a higher level entity-set. Such an entity-set is treated in the same manner as any other entity-set. We can create a binary relationship, Manager, between Works_On and Manager to represent who manages what tasks.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. $X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Stay

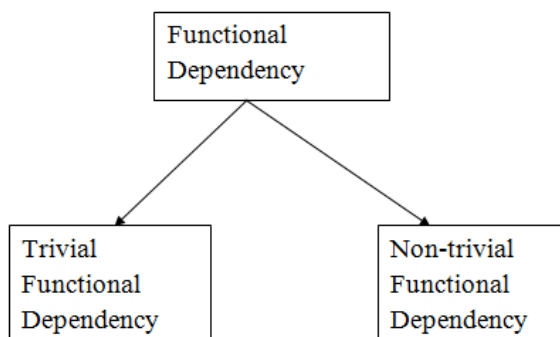
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. $\text{Emp_Id} \rightarrow \text{Emp_Name}$

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. $\{\text{Employee_id}, \text{Employee_Name}\} \rightarrow \text{Employee_Id}$ is a trivial functional dependency as
3. Employee_Id is a subset of $\{\text{Employee_Id}, \text{Employee_Name}\}$.
4. Also, $\text{Employee_Id} \rightarrow \text{Employee_Id}$ and $\text{Employee_Name} \rightarrow \text{Employee_Name}$ are trivial dependencies too.

2. Non-trivial functional dependency

- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
- When $A \cap B$ is NULL, then $A \rightarrow B$ is called as complete non-trivial.

Example:

1. $ID \rightarrow Name$,
2. $Name \rightarrow DOB$

Types of Functional dependencies in DBMS

- Difficulty Level : [Easy](#)
- Last Updated : 21 Aug, 2021

Prerequisite: [Functional dependency and attribute closure](#)

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as $X \rightarrow Y$, where X is a set of attributes that is capable of determining the value of Y . The attribute set on the left side of the arrow, X is called **Determinant**, while on the right side, Y is called the **Dependent**. Functional dependencies are used to mathematically express relations among database entities and are very important to understand advanced concepts in Relational Database System and understanding problems in competitive exams like Gate.

Example:

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

From the above table we can conclude some valid functional dependencies:

- $roll_no \rightarrow \{ name, dept_name, dept_building \}$, \rightarrow Here, $roll_no$ can determine values of fields $name$, $dept_name$ and $dept_building$, hence a valid Functional dependency
- $roll_no \rightarrow dept_name$, Since, $roll_no$ can determine whole set of $\{name, dept_name, dept_building\}$, it can determine its subset $dept_name$ also.

- $\text{dept_name} \rightarrow \text{dept_building}$, Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building
- More valid functional dependencies: $\text{roll_no} \rightarrow \text{name}$, $\{\text{roll_no}, \text{name}\} \twoheadrightarrow \{\text{dept_name}, \text{dept_building}\}$, etc.

Here are some invalid functional dependencies:

- $\text{name} \rightarrow \text{dept_name}$ Students with the same name can have different dept_name, hence this is not a valid functional dependency.
- $\text{dept_building} \rightarrow \text{dept_name}$ There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2, hence $\text{dept_building} \rightarrow \text{dept_name}$ is an invalid functional dependency.
- More invalid functional dependencies: $\text{name} \rightarrow \text{roll_no}$, $\{\text{name}, \text{dept_name}\} \rightarrow \text{roll_no}$, $\text{dept_building} \rightarrow \text{roll_no}$, etc.

Armstrong's axioms/properties of functional dependencies:

1. **Reflexivity:** If Y is a subset of X, then $X \rightarrow Y$ holds by reflexivity rule
For example, $\{\text{roll_no}, \text{name}\} \rightarrow \text{name}$ is valid.
2. **Augmentation:** If $X \rightarrow Y$ is a valid dependency, then $XZ \rightarrow YZ$ is also valid by the augmentation rule.
For example, If $\{\text{roll_no}, \text{name}\} \rightarrow \text{dept_building}$ is valid, hence $\{\text{roll_no}, \text{name}, \text{dept_name}\} \rightarrow \{\text{dept_building}, \text{dept_name}\}$ is also valid.→
3. **Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$ are both valid dependencies, then $X \rightarrow Z$ is also valid by the Transitivity rule.
For example, $\text{roll_no} \rightarrow \text{dept_name}$ & $\text{dept_name} \rightarrow \text{dept_building}$, then $\text{roll_no} \rightarrow \text{dept_building}$ is also valid.

Types of Functional dependencies in DBMS:

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.
i.e. If $X \rightarrow Y$ and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18

roll_no	name	age
---------	------	-----

44	xyz	18
----	-----	----

Here, $\{\text{roll_no}, \text{name}\} \rightarrow \text{name}$ is a trivial functional dependency, since the dependent **name** is a subset of determinant set $\{\text{roll_no}, \text{name}\}$

Similarly, $\text{roll_no} \rightarrow \text{roll_no}$ is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If $X \rightarrow Y$ and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

For example,

roll_no	name	age
---------	------	-----

42	abc	17
----	-----	----

43	pqr	18
----	-----	----

44	xyz	18
----	-----	----

Here, $\text{roll_no} \rightarrow \text{name}$ is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**

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

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other**.

i.e. If $a \rightarrow \{b, c\}$ and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency**.

For example,

roll_no	name	age
---------	------	-----

42	abc	17
----	-----	----

roll_no	name	age
---------	------	-----

43	pqr	18
----	-----	----

44	xyz	18
----	-----	----

45	abc	19
----	-----	----

Here, **roll_no** \rightarrow {**name**, **age**} is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name** \rightarrow **age** or **age** \rightarrow **name** doesn't exist !)

4. Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on determinant.
i.e. If **a** \rightarrow **b** & **b** \rightarrow **c**, then according to axiom of transitivity, **a** \rightarrow **c**. This is a **transitive functional dependency**

For 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** \rightarrow **dept** and **dept** \rightarrow **building_no**,

Hence, according to the axiom of transitivity, **enrol_no** \rightarrow **building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Types of Keys in Relational Model (Candidate, Super, Primary, Alternate and Foreign)

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNT RY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key: The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

Play Video

Advertisement

- The value of the Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO is the candidate key for relation STUDENT.
- The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.
- No. of candidate keys in a Relation are $nC(\text{floor}(n/2))$, for example if a Relation have 5 attributes i.e. R(A,B,C,D,E) then total no of candidate keys are $5C(\text{floor}(5/2))=10$.

Note – In SQL Server a unique constraint that has a nullable column, **allows** the value ‘null’ in that column **only once**. That’s why the STUD_PHONE attribute is a candidate here, but can not be ‘null’ values in the primary key attribute.

Super Key: The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.

- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.

Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

Alternate Key: The candidate key other than the primary key is called an alternate key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be an alternate key (only one out of many candidate keys).

Foreign Key: If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key to it. For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

It may be worth noting that unlike Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For Example, STUD_NO in STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuples. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique, and it cannot be null.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

What is Functional Decomposition?

Functional decomposition corresponds to the various functional relationships as how the original complex business function was developed. It mainly focusses on how the overall functionality is developed and its interaction between various components.

Large or complex functionalities are more easily understood when broken down into pieces using functional decomposition.

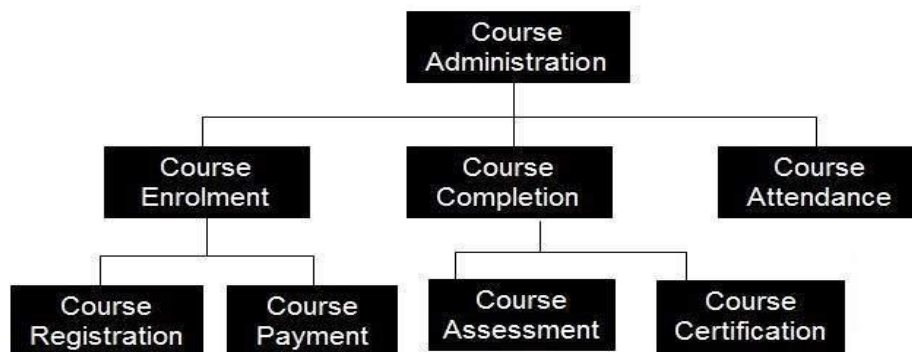
When and How?

- Functional decomposition is mostly used during the project analysis phase in order to produce functional decomposition diagrams as part of the functional requirements document.
- Functional Decomposition is done after meeting with business analysts and subject matter expertise.
- Decompose the first level components with their functions and continue to decompose to lower levels until sufficient level of detail is achieved

- Perform an end-to-end walk-through of the business operation and check each function to confirm that it is correct.

Example:

The below example, best describes the Functional Decomposition:



Normal Forms in DBMS

- Difficulty Level : [Medium](#)
- Last Updated : 12 Nov, 2021

Prerequisite – [Database normalization and functional dependency concept.](#)

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update 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 1** – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

- **Example 2 –**

-

- 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 c2

3 M c3

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 if 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.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1** – Consider table-3 as following below.

- STUD_NO COURSE_NO COURSE_FEE

- 1 C1 1000

- 2 C2 1500

- 1 C4 2000

- 4 C3 1000

- 4 C1 1000

{Note that, there are many courses having the same course fee. }

Here,

COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;

COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;

COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,

COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

But, COURSE_NO \rightarrow COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,

we need to split the table into two tables such as :

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	
COURSE_FEE			
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000
2	C5		

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

- **Example 2** – Consider following functional dependencies in relation R (A, B, C, D)
- AB \rightarrow C [A and B together determine C]
- BC \rightarrow 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 as well as it is in second normal form.

A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency $X \rightarrow Y$

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency – If $A \rightarrow B$ and $B \rightarrow C$ are two FDs then $A \rightarrow C$ is called transitive dependency.

3. **Example 1** – In relation STUDENT given in Table 4,
FD set: { $STUD_NO \rightarrow STUD_NAME$, $STUD_NO \rightarrow STUD_STATE$,
 $STUD_STATE \rightarrow STUD_COUNTRY$, $STUD_NO \rightarrow STUD_AGE$ }
Candidate Key: { $STUD_NO$ }

For this relation in table 4, $STUD_NO \rightarrow STUD_STATE$ and $STUD_STATE \rightarrow STUD_COUNTRY$ are true. So $STUD_COUNTRY$ is transitively dependent on $STUD_NO$. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT ($STUD_NO$, $STUD_NAME$, $STUD_PHONE$, $STUD_STATE$, $STUD_COUNTRY$, $STUD_AGE$) as:
 $STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)$
 $STATE_COUNTRY (STATE, COUNTRY)$

4. **Example 2** – Consider relation $R(A, B, C, D, E)$
 $A \rightarrow BC$,
 $CD \rightarrow E$,
 $B \rightarrow D$,
 $E \rightarrow A$

All possible candidate keys in above relation are { A, E, CD, BC } All attributes 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 $X \rightarrow Y$, X is a super key.

- **Example 1** – Find the highest normal form of a relation R(A,B,C,D,E) with FD set as $\{BC \rightarrow D, AC \rightarrow BE, B \rightarrow E\}$
Step 1. As we can see, $(AC)^+ = \{A, C, B, E, D\}$ but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key $\{AC\}$.
Step 2. Prime attributes are those attributes that are part of candidate key $\{A, C\}$ in this example and others will be non-prime $\{B, D, E\}$ in this example.
Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute. The relation is in 2nd normal form because $BC \rightarrow D$ is in 2nd normal form (BC is not a proper subset of candidate key AC) and $AC \rightarrow BE$ is in 2nd normal form (AC is candidate key) and $B \rightarrow E$ is in 2nd normal form (B is not a proper subset of candidate key AC).
The relation is not in 3rd normal form because in $BC \rightarrow D$ (neither BC is a super key nor D is a prime attribute) and in $B \rightarrow E$ (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal form, either LHS of an FD should be super key or RHS should be prime attribute.
So the highest normal form of relation will be 2nd Normal form.
- **Example 2** – For example consider relation R(A, B, C)
 $A \rightarrow BC$,
 $B \rightarrow C$
A and B both are super keys so above relation is in BCNF.

Key Points –

- BCNF is free from redundancy.
- If a relation is in BCNF, then 3NF is also satisfied.
- If all attributes of relation are prime attribute, then the relation is always in 3NF.
- A relation in a Relational Database is always and at least in 1NF form.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.

- If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
- Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
- There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC \twoheadrightarrow D

CD \twoheadrightarrow AE

Important Points for solving above type of question.

1) It is always a good idea to start checking from BCNF, then 3 NF, and so on.

2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, $ABC \rightarrow D$ is in BCNF (Note that ABC is a superkey), so no need to check this dependency for lower normal forms.

Candidate keys in the given relation are {ABC, BCD}

BCNF: $ABC \rightarrow D$ is in BCNF. Let us check $CD \rightarrow AE$, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: $ABC \rightarrow D$ we don't need to check for this dependency as it already satisfied BCNF. Let us consider $CD \rightarrow AE$. Since E is not a prime attribute, so the relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD is a proper subset of a candidate key and it determines E, which is non-prime attribute. So, given relation is also not in 2 NF. So, the highest normal form is 1 NF.

GATE CS Corner Questions

Practicing the following questions will help you test your knowledge. All questions have been asked in GATE in previous years or in GATE Mock Tests. It is highly recommended that you practice them.