A day without new knowledge is a lost day.

Database Technologies - MySQL

If A and a, B and b, C and c etc. are treated in the same way then it is case-insensitive. **MySQL is case-insensitive**

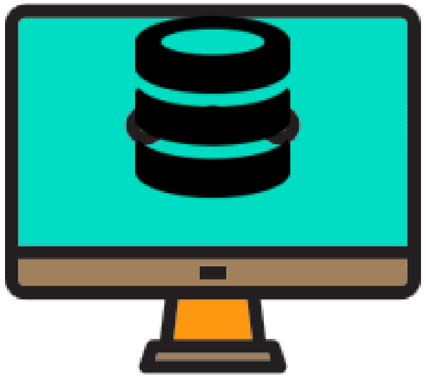
In this module we are going to learn **SQL**, **PL/SQL** and **NoSQL(MongoDB)**

Introduction

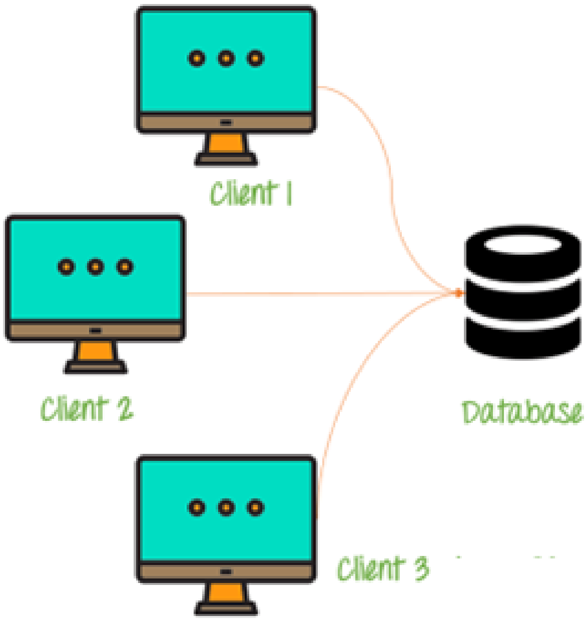
- If anyone who wants to develop a good application
  then he should have the knowledge three major components.

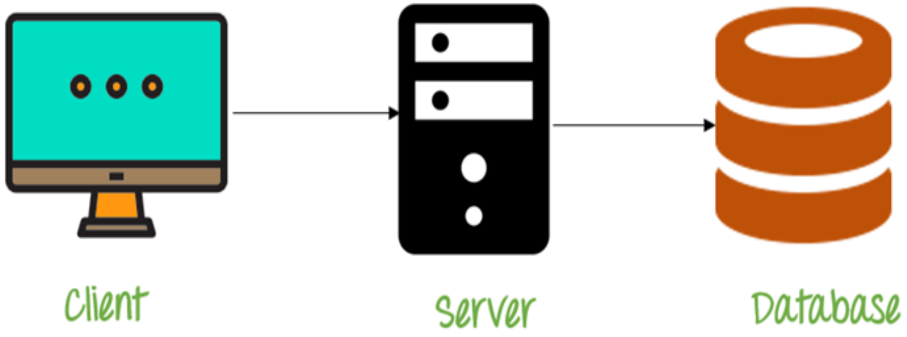
They are . . . . . .

- Presentation Layer [UI]
- Application Layer [Server Application and Client Application]
- Data Layer [Data Access Object (DAO) / Data Access Layer (DAL)] { Flat Files | RDBMS | NoSQL }



Single Tier Architecture





Three Tier Architecture

Introduction

Why do we need databases (Use Case)?

We **need databases** because they organize data in a manner which allows us to store, query, sort, and manipulate data in various ways. Databases allow us to do all this things.

Many companies collects data from different resource (like Weather data, Geographical data, Finance data, Scientific data, Transport data, Cultural data, etc.)

What is Relation and Relationship?

Remember:

- A ***reference*** is a relationship between two tables where the values in one table refer to the values in another table.
- A ***referential key*** is a column or set of columns in a table that refers to the primary key of another table. It establishes a relationship between two tables, where one table is called the parent table, and the other is called the child table.

relation and relationship?

Relation ***(in Relational Algebra "R" stands for relation)*:** In Database, a relation represents a **table** or an **entity** than contain attributes.

Relationship: In database, relationship is that how the two entities are **connected** to each other, i.e. what kind of relationship type they hold between them.

Primary/Foreign key is used to specify this relationship.

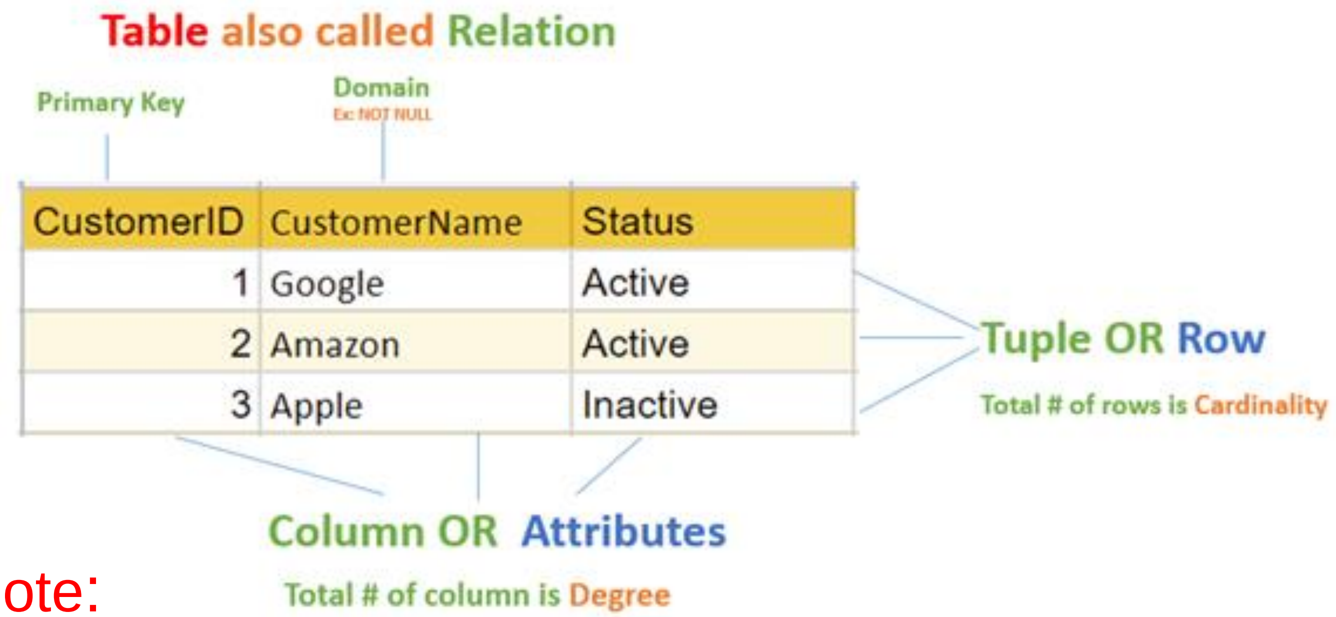


Table also called Relation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Remember:

Foreign Key is also know as
- referential constraint
- referential integrity constraint.

Note:

- **Table -** The physical instantiation of a relation in the database schema.
- **Relation -** A logical construct that organizes data into rows and columns.

File Systems is the traditional way to keep your data organized.

File System
vs
DBMS

file-oriented system

File Anomalies

```
struct Employee {
   int emp_no;
   char emp_name[50];
   int salary;
} emp[1000];
```

```
struct Employee {
   int emp_no;
   char emp_name[50];
   int salary;
};
struct Employee emp[1000];
```

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.       |
.       |
.       |
500 sam 3500
.       |
.       |
.       |
1000 amit 2300
```

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.       |
.       |
.       |
500 sam 3500
.       |
.       |
1000 amit 2300
.       |
.       |
.       |
2000 jerry 4500
.       |
.       |
```

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.       |
.       |
500  sam 3500
.       |
3   rajan 4500
.       |
500  sam 3500
.       |
.       |
1000 amit 2300
```

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.       |
.       |
sam 500 3500
.       |
ram  550 5000
.       |
1000 amit 2300
```

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.       |
500 sam 3500
.       |
600 neel 4500
```

file-oriented system

File Anomalies

- Create/Open an existing file
- Reading from file
- Writing to a file
- Closing a file

c:\employee.txt

```
1   suraj 4000
2   ramesh 6000
3   rajan 4500
.
.
.
500 sam 3500
.
.
.
1000 amit 2300
```

file attributes

- File Name
- Type
- Location

file permissions

- File permissions
- Share permissions

search empl ID=1

```
1 suraj 4000
2 ramesh 6000
3 rajan 4500
.
.
.
500 sam 3500
.
.
.
1000 amit 2300
```

search emp_name

```
1 suraj 4000
2 ramesh 6000
3 rajan 4500
.
.
.
500 sam 3500
.
.
.
1000 amit 2300
```

advantages of file-oriented system

The biggest advantage of file-based storage is that anyone can understand the system.

Advantage of File-oriented system

- **Backup**: It is possible to take faster and automatic back-up of database stored in files of computer-based systems.
- **Data retrieval:** It is possible to retrieve data stored in files in easy and efficient way.
- **Editing***:* It is easy to edit any information stored in computers in form of files.
- **Remote access***:* It is possible to access data remote location.
- **Sharing***:* The files stored in systems can be shared among multiple users at a same time.

disadvantage of file-oriented system

The biggest disadvantage of file-based storage is as follows.

Disadvantage of File-oriented system

- **Data redundancy***:* It is possible that the same information may be duplicated in different files. This leads to data redundancy results in memory wastage.

(Suppose a customer having both kind of accounts- saving and current account. In such a situation a customers detail are stored in both the file, saving.txt- file and current.txt- file , which leads to Data Redundancy.)

- **Data inconsistency***:* Because of data redundancy, it is possible that data may not be in consistent state.

(Suppose customer changed his/her address. There might be a possibility that address is changed in only one file (saving.txt) and other (current.txt) remain unchanged.)

- **Limited data sharing***:* Data are scattered in various files and also different files may have different formats (for example: .txt, .csv, .tsv and .xml) and these files may be stored in different folders so, due to this it is difficult to share data among different applications.
- **Data Isolation:** Because data are scattered in various files, and files may be in different formats (for example: .txt, .csv, .tsv and .xml), writing new application programs to retrieve the appropriate data is difficult.
- **Data security:** Data should be secured from unauthorized access, for example a account holder in a bank should not be able to see the account details of another account holder, such kind of security constraints are difficult to apply in file processing systems.

***Relation Schema*:** A relation schema represents name of the relation with its attributes.

- e.g. student (roll_no int, name varchar, address varchar, phone varchar and age int) is relation schema for STUDENT

DBMS

- **database:** Is the collection of **related data** which is **organized,** database can store and retrieve large amount of data easily, which is stored in one or more data files by one or more users, it is called as **structured data.**
- **management system**: it is a software, designed to **define**, **manipulate**, **retrieve** and **manage** data in a database.

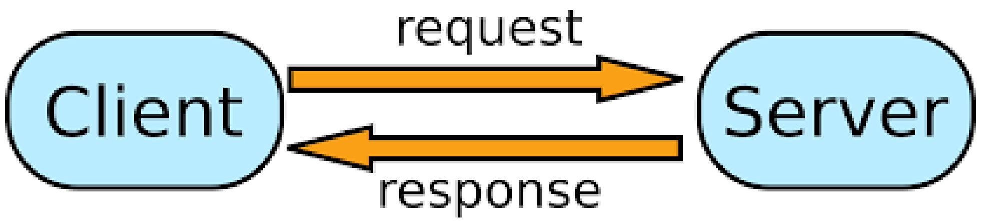
Microsoft SQL Server
ORACLE
PostgreSQL
MySQL
SYBASE
IBM DB2.

relational database management system?

A RDBMS is a database management system (DBMS) that is based on the **relational model** introduced by Edgar Frank Codd at IBM in 1970.

RDBMS supports

- *client/server Technology*
- *Highly Secured*
- *Relationship (PK/FK)*



relational model concepts

and

properties of relational table

relational model concepts

Relational model organizes data into one or more tables (or "relations") of columns and rows. Rows are also called records or tuples. Columns are also called attributes.

- **Tables** – In relational model, relations are saved in the form of Tables. A table has rows and columns.
- **Attribute** – Attributes are the properties that define a relation. e.g. (**roll_no, name, address, phone and age)**
- **Tuple** – A single row of a table, which contains a single record for that relation is called a tuple.
- **Relation schema** – A relation schema describes the relation name (table name) with its attribute (columns) names.

  e.g. **student(prn, name, address, phone, DoB, age, hobby, email, status)** is relation schema for student relation.
- **Attribute domain** – An attribute domain specifies the data type, format, and constraints of a column, and defines the range of values that are valid for that column.

Remember:

- In database management systems, null is used to represent missing or unknown data in a table column.

properties of relational table

ID	job	firstName	DoB	salary
1	manager	Saleel Bagde	yyyy-mm-dd	••••••
3	salesman	Sharmin	yyyy-mm-dd	••••••
4	accountant	Vrushali	yyyy-mm-dd	••••••
2	salesman	Ruhan	yyyy-mm-dd	••••••
5	9500	manager	yyyy-mm-dd	••••••
5	Salesman	Rahul Patil	yyyy-mm-dd	••••••

Relational tables have six properties:

- Values are atomic.
- Column values are of the same kind. (***Attribute Domain*: ** Every attribute has some pre-defined datatypes, format, and constraints of a column, and defines the range of values that are valid for that column known as attribute domain.)
- Each row is unique.
- The sequence of columns is insignificant – (unimportant).
- The sequence of rows is insignificant – (unimportant).
- Each attribute/column must have a unique name.

What is data?

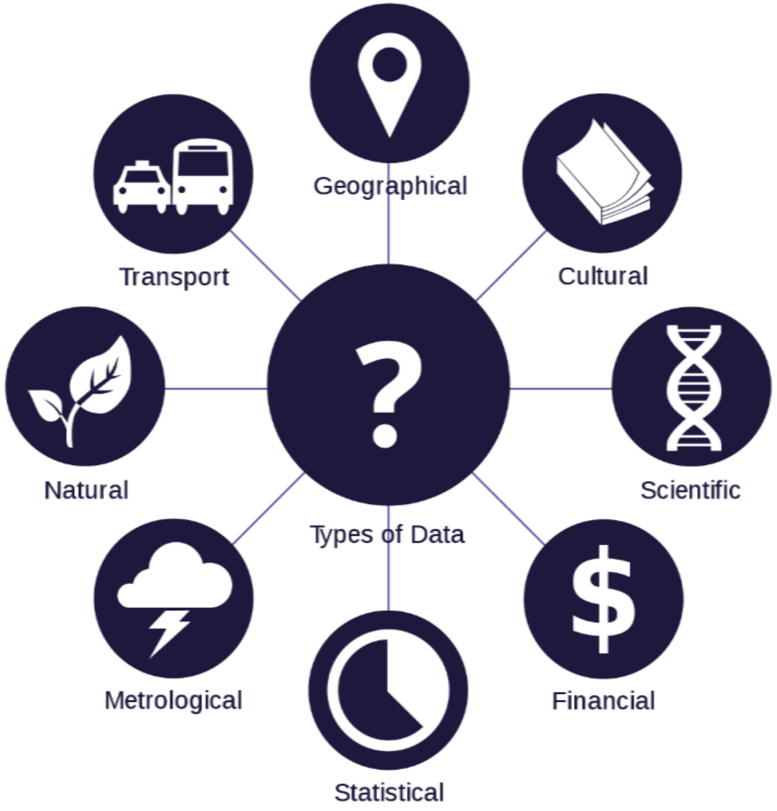
what is data?

Data is any facts that can be stored and that can be processed by a computer.

Data can be in the form of Text or Multimedia

e.g.

- number, characters, or symbol
- images, audio, video, or signal



What is Entity Relationship Diagram?

Entity Relationship Diagram (ER Diagram)

Use E-R model to get a high-level graphical view to describe the **"ENTITIES"** and their **"RELATIONSHIP"**

The basic constructs/components of ER Model are **Entity**, **Attributes** and **Relationships**.

An entity can be a **real-world object.**

What is Entity?

entity

In relation to a database , an entity is a

- Person(student, teacher, employee, department, …)
- Place(classroom, building, …) --a particular position or area
- Thing(computer, lab equipment, …) --an object that is not named
- Concept(course, batch, student's attendance, …) -- an idea,

about which data can be stored. All these entities have some **attributes** or **properties** that give them their **identity**.

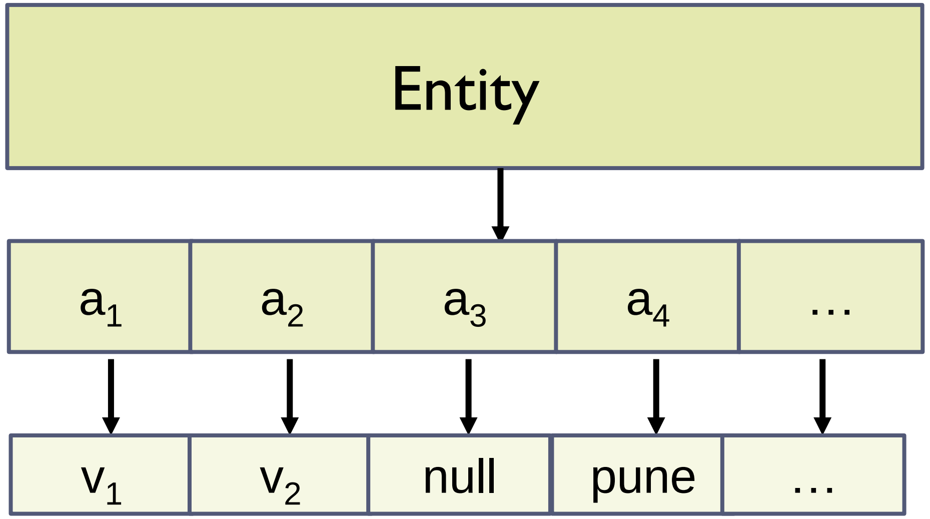
Every entity has its own characteristics.

In database management systems, **null** is used to represent missing or unknown data in a table column.

What is an Attribute?

Attributes are the properties that define a relation.

e.g. student(ID, firstName, middleName, lastName, city)



attributes

In Entity Relationship(ER) Model attributes can be classified into the following types.

- Simple/Atomic and Composite Attribute
- Single Valued and Multi Valued attribute
- Stored and Derived Attributes
- Complex Attribute

Remember:

In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different relations.

attributes

Simple / Atomic Attribute (Can’t be divided further)	**--VS--**	**Composite Attribute** (Can be divided further)
Single Value Attribute (Only One value)	**--VS--**	**Multi Valued Attribute** (Multiple values)
Stored Attribute (Only One value)	**--VS--**	**Derived Attribute** (Virtual)
Complex Attribute (Composite & Multivalued)		

Employee ID: An employee ID can be a composite attribute, which is composed of sub-attributes such as department code, job code, and employee number.

- **Atomic Attribute:** An attribute that cannot be divided into smaller independent attribute is known as atomic attribute.
  e.g. ID’s, PRN, age, gender, zip, marital status cannot further divide.

- **Single Value Attribute:** An attribute that has only single value is known as single valued attribute.
  e.g. manufactured part can have only one serial number, voter card, blood group, price, quantity, branch can have only one value.

- **Stored Attribute:** The stored attribute are such attributes which are already stored in the database and from which the value of another attribute is derived.
  e.g. (HRA, DA…) can be derive from salary, age can be derived from DoB, total marks or average marks of a student can be derived from marks.

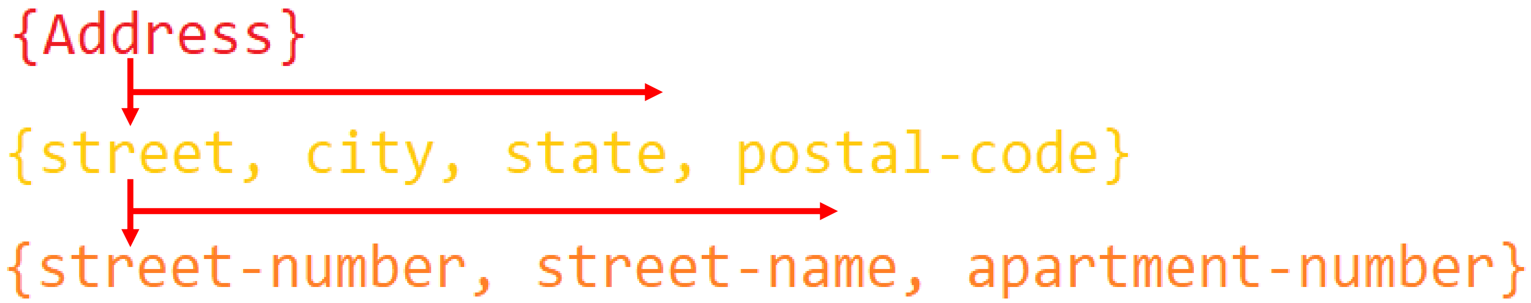
Composite **VS** Multi Valued Attribute

composite / multi valued attributes

Composite Attribute

Person Entity

- *Name* attribute: (firstName, middleName, and lastName)
- *PhoneNumber* attribute: (countryCode, cityCode, and phoneNumber)



{Address}

{street, city, state, postal-code}

{street-number, street-name, apartment-number}

Multi Valued Attribute

Person Entity

- *Hobbies* attribute: [reading, hiking, hockey, skiing, photography, . . .]
- *SpokenLanguages* attribute: [Hindi, Marathi, Gujarati, English, . . .]
- *Degrees* attribute: [10th , 12th, BE, ME, PhD, . . .]
- *emailID* attribute: [saleel@gmail.com, salil@yahoomail.com, . . .]

What is an Prime, Non-Prime Attribute?

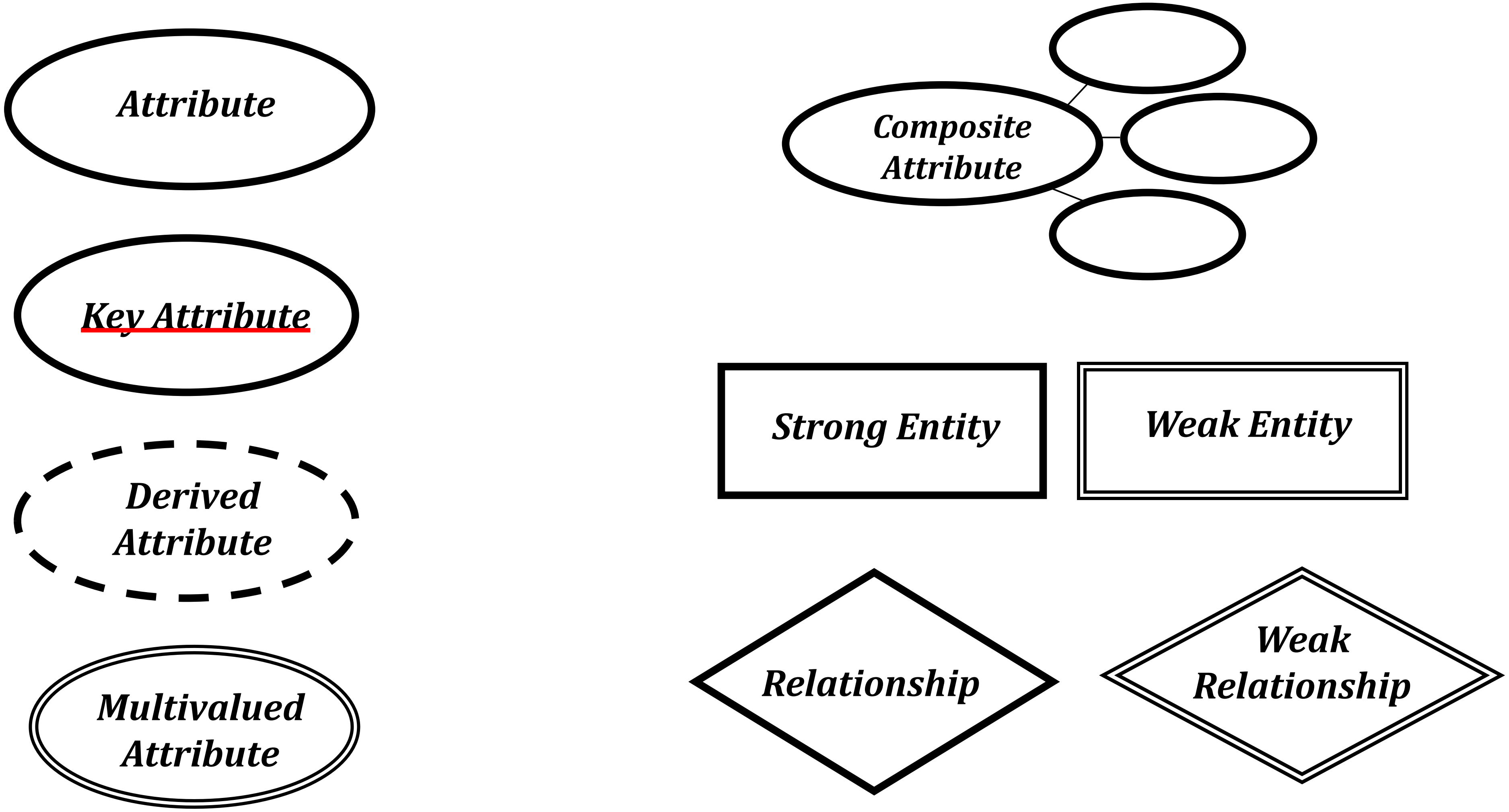
Prime attribute (*Entity integrity*)

An attribute, which is a **part of the prime-key** **(**candidate key**)**, is known as a prime attribute.

Non-prime attribute

An attribute, which is **not a part of the prime-key** **(**candidate key**)**, is said to be a non-prime attribute.

entity relationship diagram symbols



Attribute

Key Attribute

Derived Attribute

Multivalued Attribute

Composite Attribute

Strong Entity

Weak Entity

Relationship

Weak Relationship

strong and weak entity

An entity may participate in a relation either totally or partially.

Strong Entity: A strong entity is not dependent on any other entity in the schema. A strong entity will always have a **primary key**. Strong entities are represented by a single rectangle.

Weak Entity: A weak entity is dependent on a strong entity to ensure its existence. Unlike a strong entity, a weak entity does not have any primary key. A weak entity is represented by a double rectangle. The relation between one strong and one weak entity is represented by a double diamond.

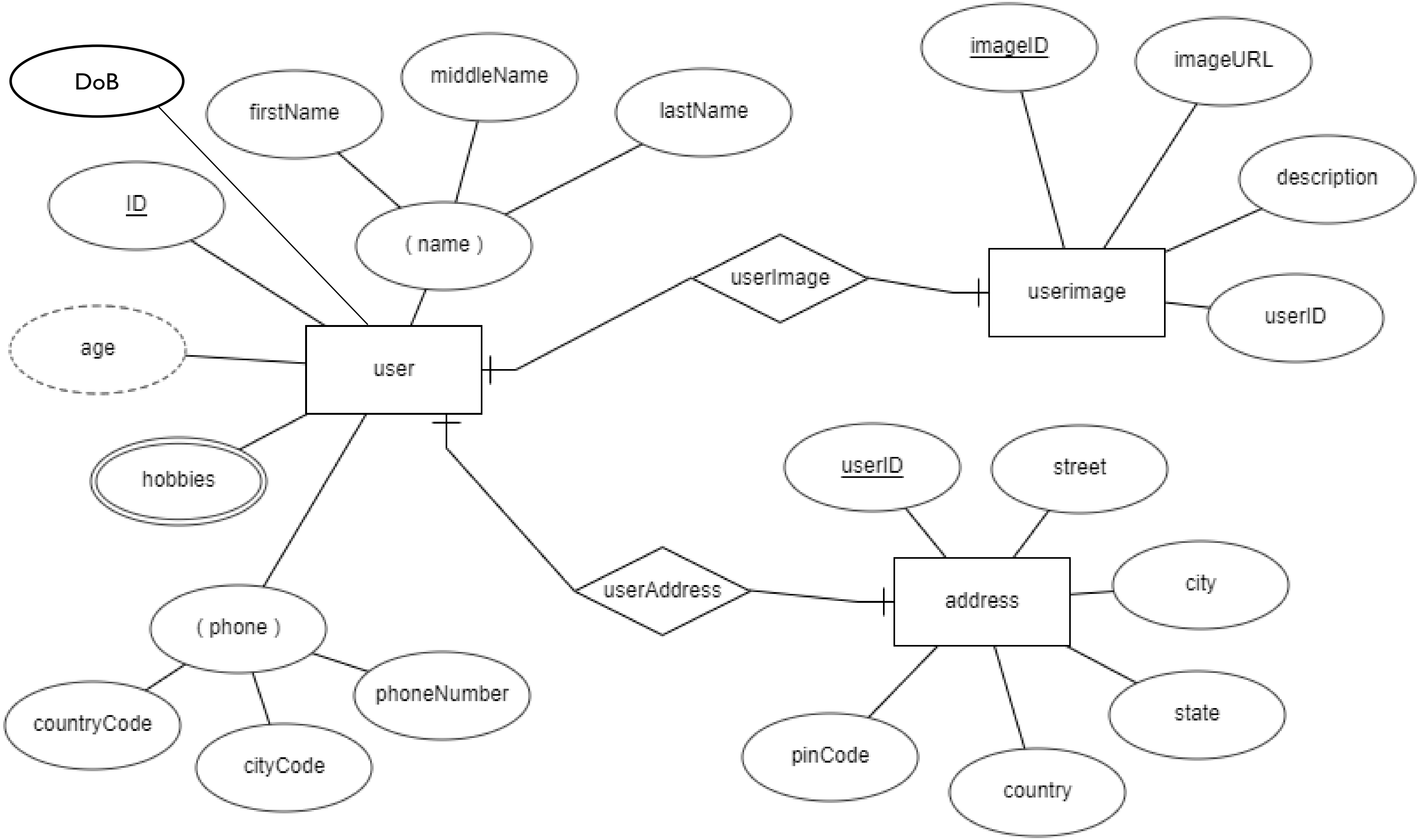
Example 1 – A loan entity can not be created for a customer if the customer doesn’t exist

Example 2 – A payment entity can not be created for a loan if the loan doesn’t exist

Example 3 – A dependents list entity can not be created if the employee doesn’t exist

Example 4 – A prescription entity can not be created for a patient if the patient doesn’t exist

entity relationship diagram



What is a degree, cardinality, domain and union in database?

What is a degree, cardinality, domain and union in database?

- **Degree d(R) / Arity**: Total number of **attributes/columns** present in a relation/table is called **degree of the relation** and is denoted by **d(*R*)**.

- **Cardinality |R|**: Total number of **tuples/rows** present in a relation/table, **is called cardinality of a relation** and is denoted by **|*R*|**.

- **Domain**: Total range of accepted values for an attribute of the relation **is called the domain of the attribute**. (**Data Type(size)**)

- **Union Compatibility**: Two relations *R* and *S* are set to be Union Compatible to each other if and only if:
  1. They have the **same degree d(R)**.
  2. Domains of the respective attributes should also be same.

What is domain constraint and types of data integrity constraints?

Data integrity refers to the correctness and completeness of data.

A domain constraint and types of data integrity constraints

- **Domain Constraint** = data type + Constraints (**not null/unique/primary key/foreign key/check/default**)

  e.g. custID INT, constraint pk_custid PRIMARY KEY(custID)

Three types of integrity constraints: **entity integrity, referential integrity** and **domain integrity**:

- **Entity integrity:** Entity Integrity Constraint is used to ensure the uniqueness of each record the table. There are primarily two types of integrity constraints that help us in ensuring the uniqueness of each row, namely, UNIQUE constraint and PRIMARY KEY constraint.

- **Referential integrity:** Referential Integrity Constraint ensures that there always exists a valid relationship between two tables. This makes sure that if a foreign key exists in a table relationship then it should always reference a corresponding value in the second table **t_1[FK] = t_2[PK]** or it should be null.

- **Domain integrity:** A domain is a set of values of the same type. For example, we can specify if a particular column can hold null values or not, if the values have to be unique or not, the data type or size of values that can be entered in the column, the default values for the column, etc..

types of Keys?

Keys are used to establish relationships between tables and also to uniquely identify any record in the table.

types of Keys?

r = Employee(EmployeeID, FullName, job, salary, PAN, DateOfBirth, emailID, deptno)

- **Candidate Key:** are individual columns in a table that qualifies for uniqueness of all the rows. Here in Employee table EmployeeID, PAN or emailID are Candidate keys.

- **Primary Key**: is the columns you choose to maintain uniqueness in a table. Here in Employee table you can choose either EmployeeID, PAN or emailID columns, EmployeeID is preferable choice.

- **Alternate Key**: Candidate column other the primary key column, like if EmployeeID is primary key then , PAN or emailID columns would be the Alternate key.

- **Super Key**: If you add any other column to a primary key then it become a super key, like EmployeeID + FullName is a Super Key.

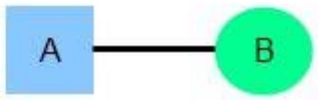
- **Composite Key**: If a table do not have any single column that qualifies for a Candidate key, then you have to select 2 or more columns to make a row unique. Like if there is no EmployeeID, PAN or emailID columns, then you can make FullName + DateOfBirth as Composite key. But still there can be a narrow chance of duplicate row.

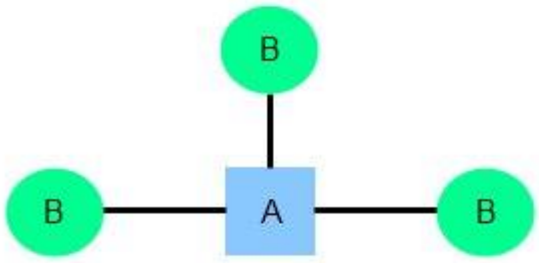
Common relationships

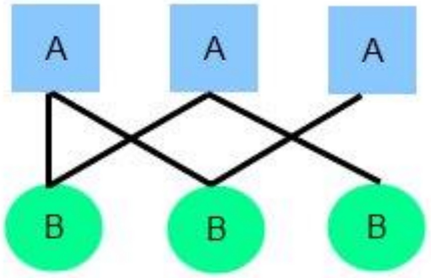
relationships

Common relationship

1. one-to-one (1:1)
2. one-to-many (1:M)
3. many-to-many (M:N)





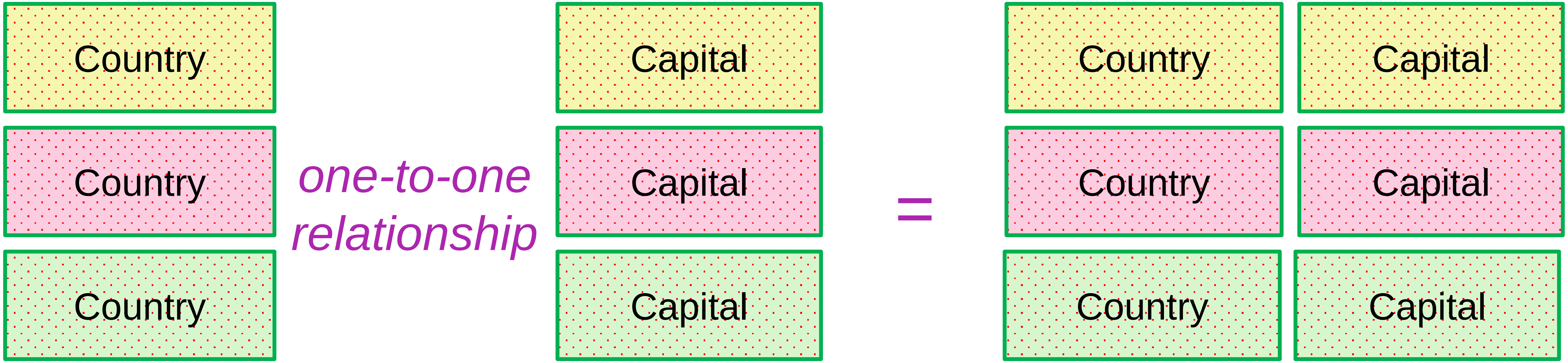


one-to-one relationship

one-to-one relationship

A *one-to-one* relationship between two tables means that a row in one table can only relate to zero/one row in the table on the other side of their relationship. This is the least common database relationship.

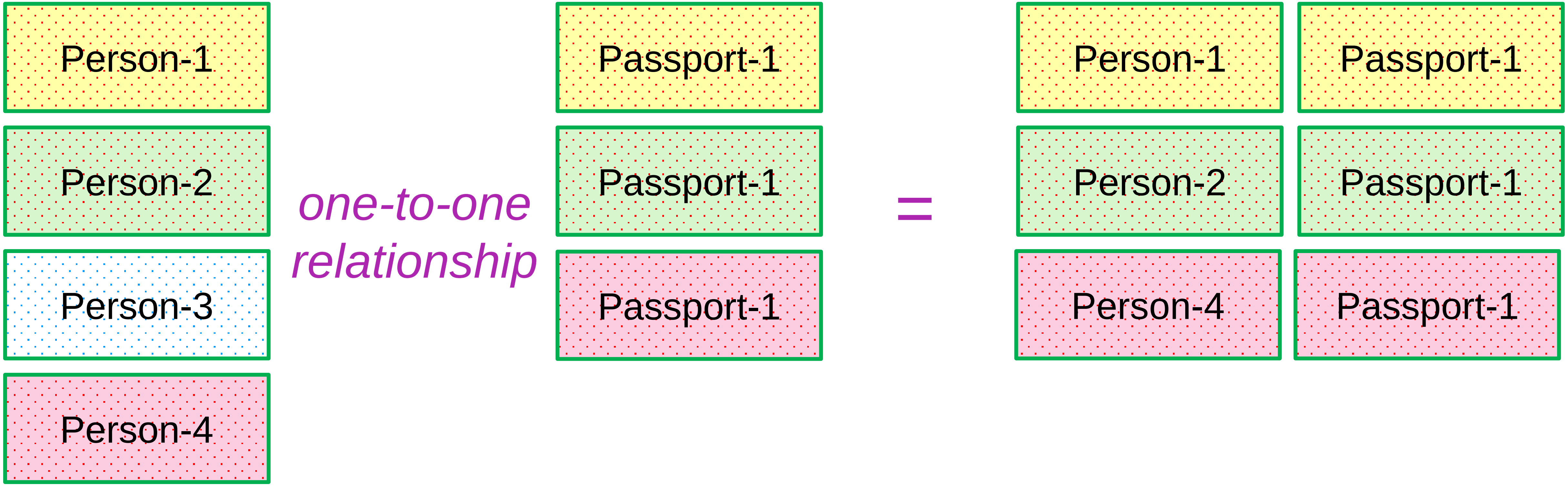
A *one-to-one* relationship is a type of cardinality that refers to the relationship between two entities ***R*** and ***S*** in which one element of entity ***R*** may only be linked to zero/one element of entity ***S***, and vice versa.



one-to-one relationship

A *one-to-one* relationship between two tables means that a row in one table can only relate to zero/one row in the table on the other side of their relationship. This is the least common database relationship.

A *one-to-one* relationship is a type of cardinality that refers to the relationship between two entities ***R*** and ***S*** in which one element of entity ***R*** may only be linked to zero/one element of entity ***S***, and vice versa.



Person-1

Person-2

Person-3

Person-4

one-to-one relationship

Passport-1

Passport-1

Passport-1

=

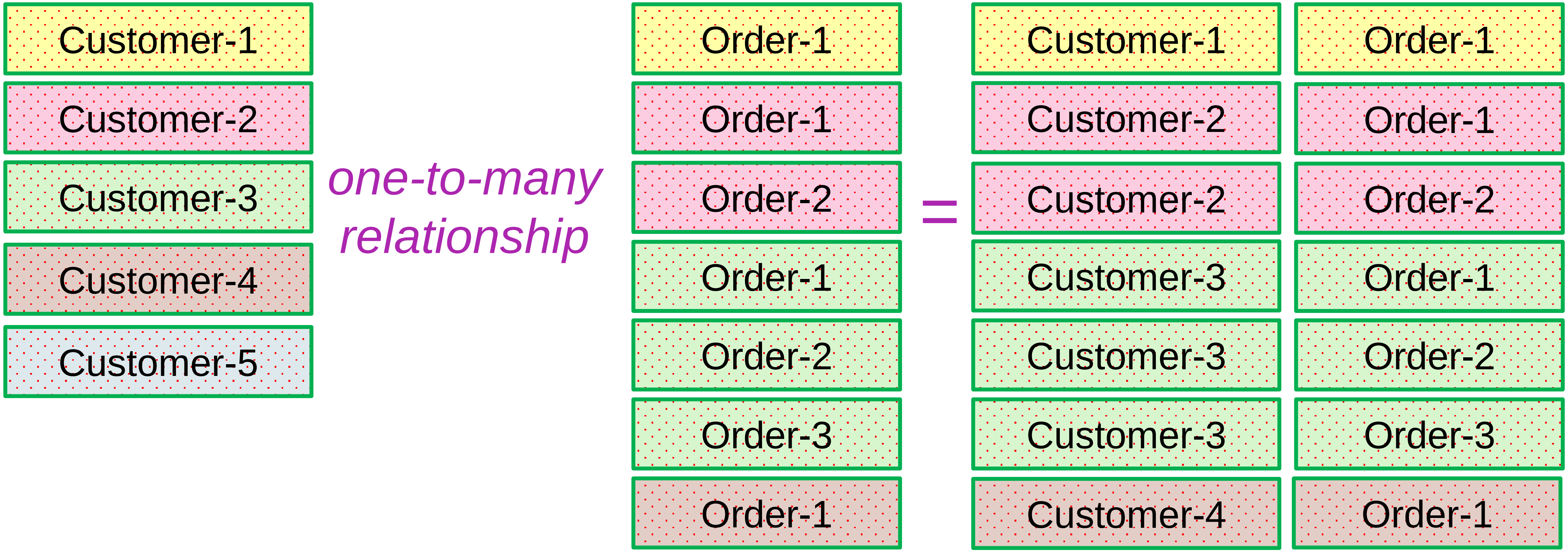
Person-1	Passport-1
Person-2	Passport-1
Person-4	Passport-1

one-to-many relationship

one-to-many relationship

A *one-to-many* relationship between two tables means that a row in one table can have zero or more row in the table on the other side of their relationship.

a *one-to-many* relationship is a type of cardinality that refers to the relationship between two entities ***R*** and ***S*** in which an element of ***R*** may be linked to many elements of ***S***, but a member of ***S*** is linked to only one element of ***R***.



Customer-1
Customer-2
Customer-3
Customer-4
Customer-5

one-to-many relationship

Order-1
Order-1
Order-2
Order-1
Order-2
Order-3
Order-1

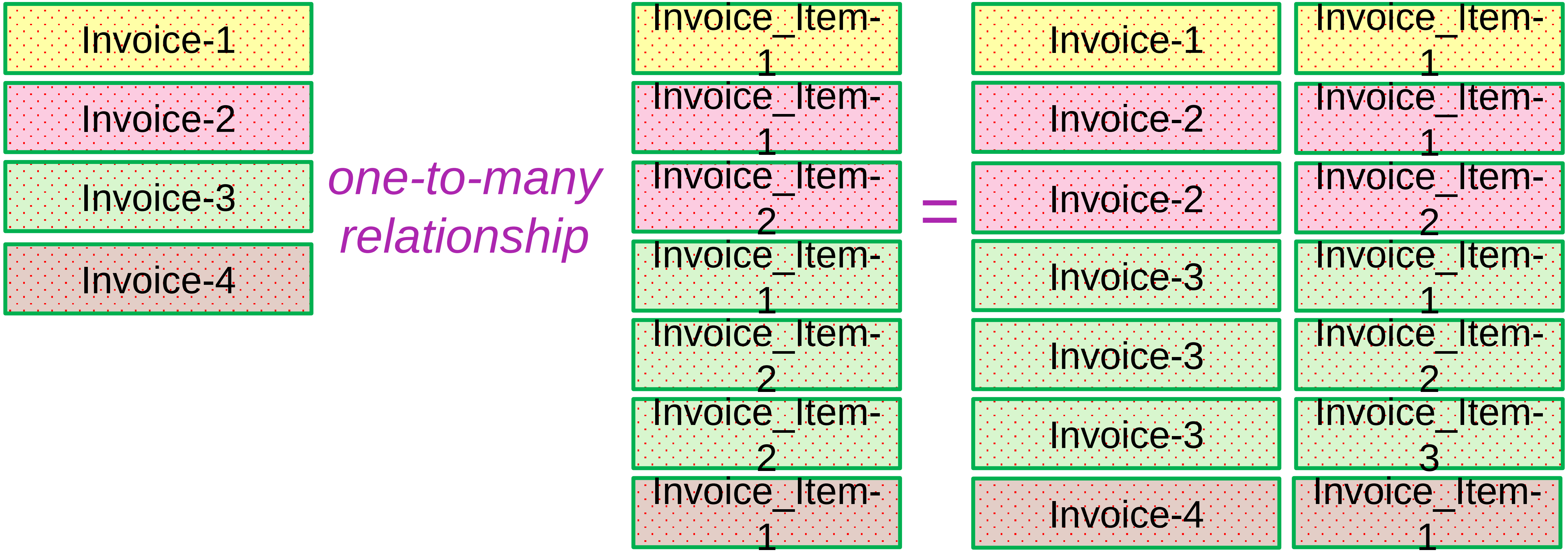
=

Customer-1	Order-1
Customer-2	Order-1
Customer-2	Order-2
Customer-3	Order-1
Customer-3	Order-2
Customer-3	Order-3
Customer-4	Order-1

one-to-many relationship

A *one-to-many* relationship between two tables means that a row in one table can have one or more row in the table on the other side of their relationship.

a *one-to-many* relationship is a type of cardinality that refers to the relationship between two entities ***R*** and ***S*** in which an element of ***R*** may be linked to many elements of ***S***, but a member of ***S*** is linked to only one element of ***R***.



Invoice-1
Invoice-2
Invoice-3
Invoice-4

one-to-many relationship

Invoice_Item-1
Invoice_Item-1
Invoice_Item-2
Invoice_Item-1
Invoice_Item-2
Invoice_Item-2
Invoice_Item-1

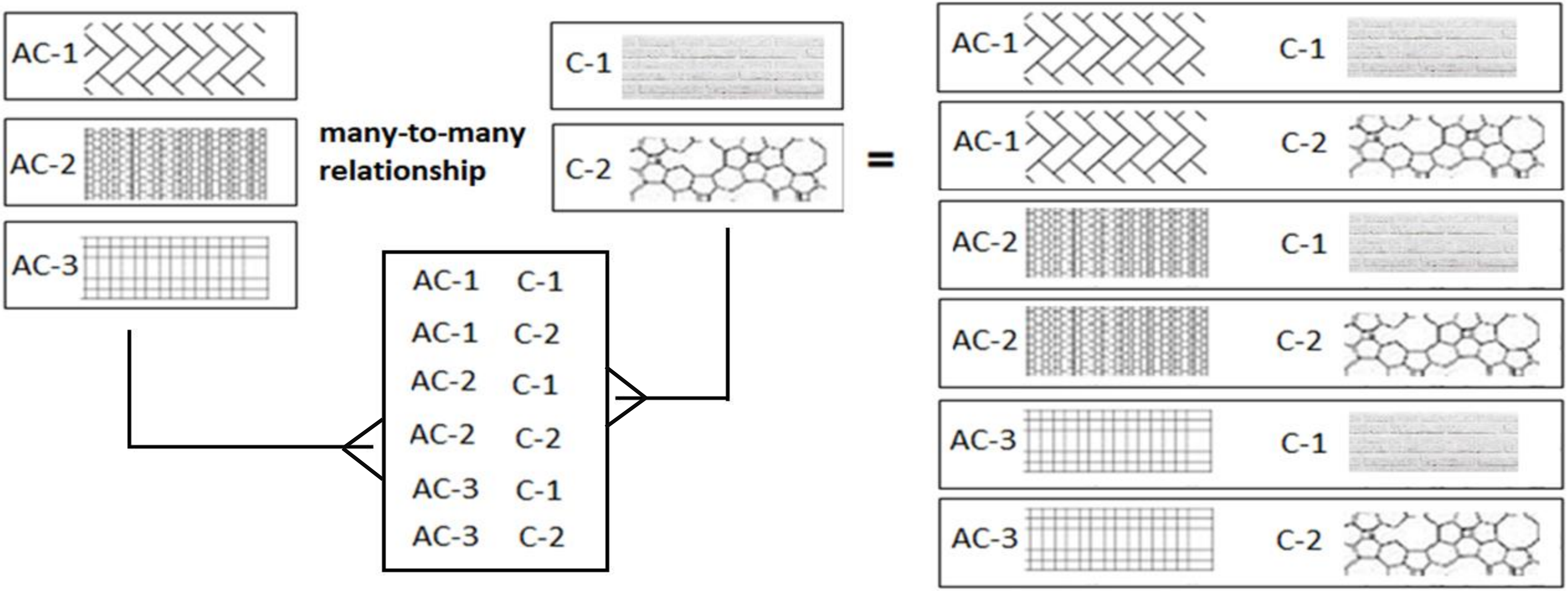
=

Invoice-1	Invoice_Item-1
Invoice-2	Invoice_Item-1
Invoice-2	Invoice_Item-2
Invoice-3	Invoice_Item-1
Invoice-3	Invoice_Item-2
Invoice-3	Invoice_Item-3
Invoice-4	Invoice_Item-1

many-to-many relationship

many-to-many relationship

A *many-to-many* relationship is a type of cardinality that refers to the relationship between two entities ***R*** and ***S*** in which ***R*** may contain a parent instance for which there are many children in ***S*** and vice versa.



MySQL

MySQL is the most popular **Open Source** Relational Database Management System.

MySQL was created by a Swedish company - MySQL AB that was founded in 1995. It was acquired by Sun Microsystems in 2008; Sun was in turn acquired by Oracle Corporation in 2010.

When you use MySQL, you're actually using at least two programmes. One program is the MySQL server (*mysqld.exe*) and other program is MySQL client program (*mysql.exe*) that connects to the database server.

What is SQL?

what is sql?

Remember:

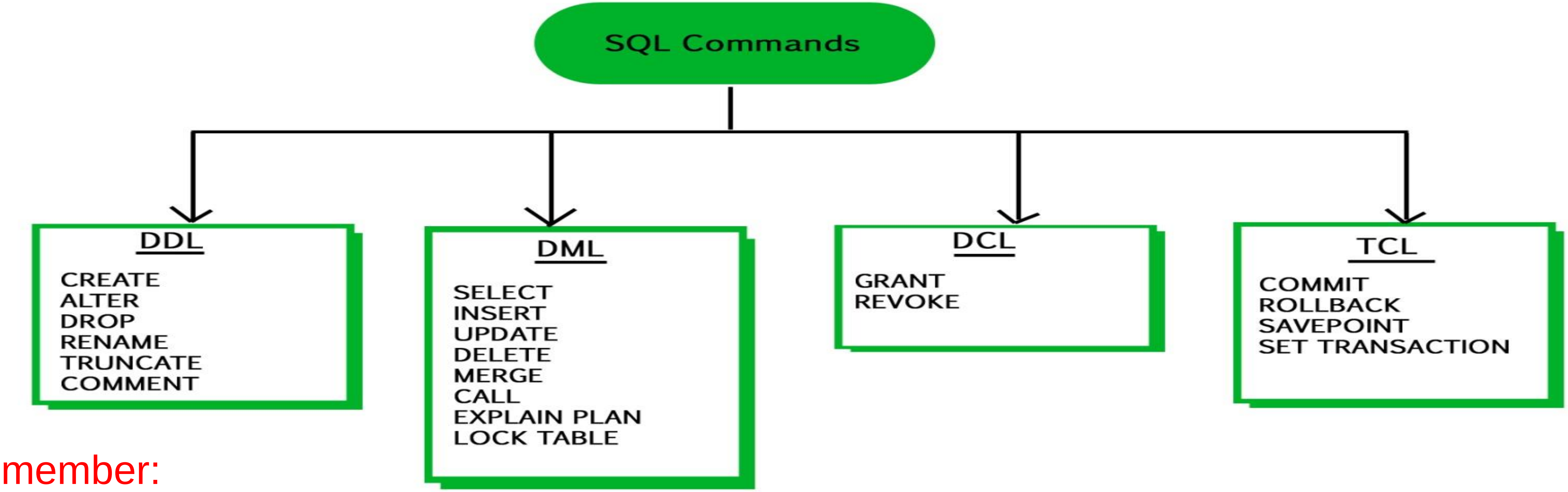
- **EXPLICIT or IMPLICIT commit will commit the data.**

SQL (Structured Query Language) is a database language designed and developed for managing data in relational database management systems (RDBMS). SQL is common language for all Relational Databases.

Remember:

what is sql?

- **An EXPLICIT commit happens when we execute an SQL "COMMIT" command.**
- **An IMPLICIT commits occur without running a "COMMIT" command.**



SQL Commands

DDL
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT

DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE

DCL
GRANT
REVOKE

TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

Remember:

- A **NULL** value is not treated as a **blank** or **0**. Null or NULL is a special marker used in Structured Query Language to indicate that a data value does not exist or missing or unknown in the database.
- **Degree d(R)**: Total no. of attributes/columns present in a relation/table is called degree of the relation and is denoted by **d(R).**
- **Cardinality |R|**: Total no. of tuples present in a relation or Rows present in a table, is called cardinality of a relation and is denoted by **|R|.**

comments in mysql

- From a # character to the end of the line.
- From a **--** sequence to the end of the line.
- From a /* sequence to the following */ sequence.

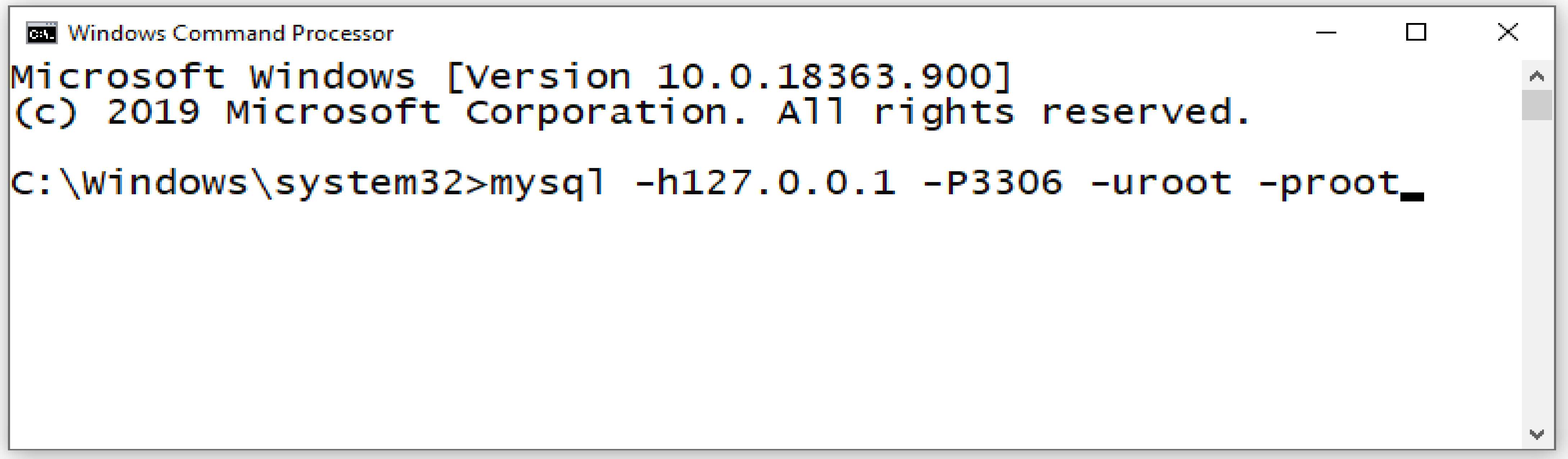
Reconnect to the server	**\r**
Execute a system shell command	**\!**
Exit mysql	**\q**
Change your mysql prompt.	**prompt str or \R str**

Login to MySQL

login

Default port for MySQL Server: 3306

- C:\> mysql -hlocalhost -P3307 -uroot -p
- C:\> mysql -h127.0.0.1 -P3307 -uroot -p [*database_name*]
- C:\> mysql -h192.168.100.14 -P3307 -uroot -psaleel [*database_name*]
- C:\> mysql --host localhost --port 3306 --user root --password=ROOT [*database_name*]
- C:\> mysql --host=localhost --port=3306 --user=root --password=ROOT [*database_name*]



```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>mysql -h127.0.0.1 -P3306 -uroot -proot_
```

SHOW DATABASES

SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern' | WHERE
expr]
```

SHOW SCHEMAS is a synonym for SHOW DATABASES.

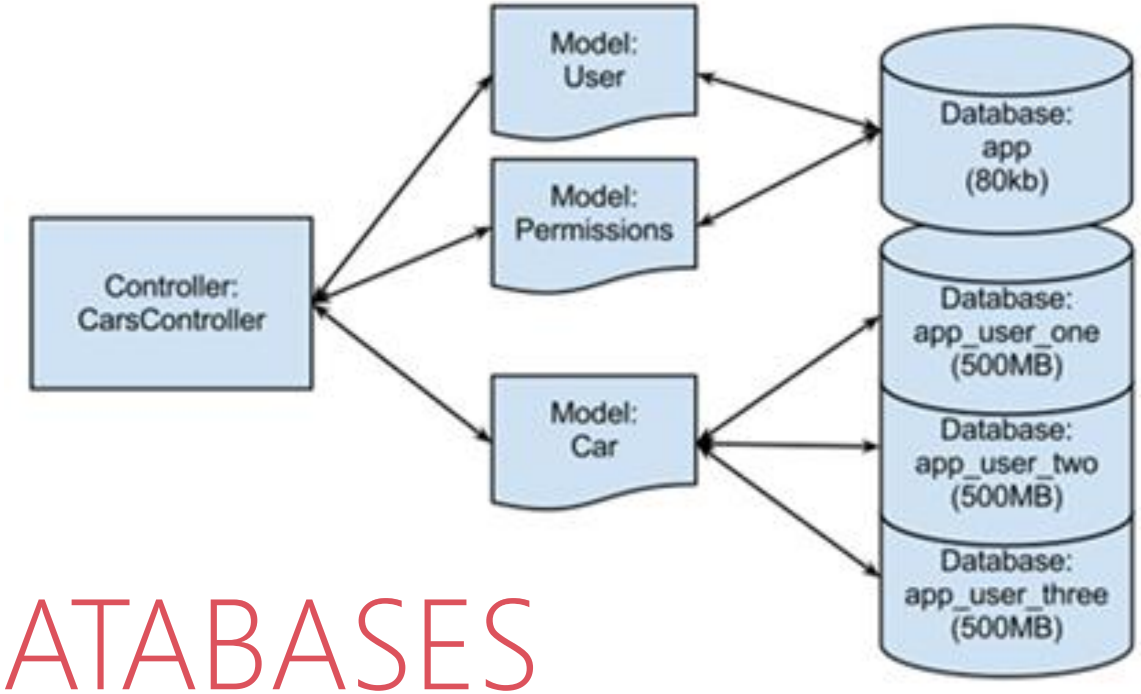
```
SHOW DATABASES;

SHOW SCHEMAS;

SHOW DATABASES LIKE 'U%';

SHOW SCHEMAS LIKE 'U%';
```

NULL means “no database is selected”. Issue the ***USE dbName*** command to select the database.



USE DATABASES

The **USE db_name** statement tells MySQL to use the db_name database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another **USE** statement is issued.

USE DATABASES Syntax

```
USE db_name
\U db_name
```

Note:

- USE, does not require a semicolon.
- USE must be followed by a database name.

```
USE db1
```

```
\U db1
```

The **char** is a fixed-length character data type,
The **varchar** is a variable-length character data type.

```
CREATE TABLE temp (c1 CHAR(10), c2
VARCHAR(10));
INSERT INTO temp VALUES('SALEEL', 'SALEEL');
SELECT * FROM temp WHERE c1 LIKE 'SALEEL';
```

datatypes

ENAME CHAR (10)	S	A	L	E	E	L					LENGTH -> 10
ENAME VARCHAR2(10)	S	A	L	E	E	L					LENGTH -> 6

In MySQL

When CHAR values are retrieved, the trailing spaces are removed (unless the ***PAD_CHAR_TO_FULL_LENGTH*** SQL mode is enabled)

ENAME CHAR (10)	S	A	L	E	E	L					LENGTH -> 6
ENAME VARCHAR(10)	S	A	L	E	E	L					LENGTH -> 6

Note:

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they store binary strings rather than nonbinary strings. That is, they store byte strings rather than character strings.

datatype - string

Datatypes	Size	Description
CHAR [(length)]	0-255	
VARCHAR (length)	0 to 65,535	The maximum row size (65,535 bytes, which is shared among all columns.
TINYTEXT [(length)]	$(2^8 - 1)$ bytes	
TEXT [(length)]	$(2^{16} -1)$ bytes	65,535 bytes ~ 64kb
MEDIUMTEXT [(length)]	$(2^{24} -1)$ bytes	16,777,215 bytes ~16MB
LONGTEXT [(length)]	$(2^{32} -1)$ bytes	4,294,967,295 bytes ~4GB
ENUM('value1', 'value2',...)	65,535 members	
SET('value1', 'value2',...)	64 members	
BINARY[(length)]	255	
VARBINARY(length)		

By default, trailing spaces are trimmed from CHAR column values on retrieval. If ***PAD_CHAR_TO_FULL_LENGTH*** is enabled, trimming does not occur and retrieved CHAR values are padded to their full length.

- *SET sql_mode = '';*
- *SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';*

datatype - numeric

Datatypes	Size	Description
TINYINT	1 byte	-128 to +127 **(The unsigned range is 0 to 255).**
SMALLINT [(length)]	2 bytes	-32768 to 32767. **(The unsigned range is 0 to 65535).**
MEDIUMINT [(length)]	3 bytes	-8388608 to 8388607. **(The unsigned range is 0 to 16777215).**
INT, INTEGER [(length)]	4 bytes	-2147483648 to 2147483647. **(The unsigned range is 0 to 4294967295).**
BIGINT [(length)]	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
FLOAT [(length[,decimals])]	4 bytes	**FLOAT(255,30)**
DOUBLE [PRECISION] [(length[,decimals])], REAL [(length[,decimals])]	8 bytes	**REAL(255,30) / DOUBLE(255,30)** REAL will get converted to DOUBLE
DECIMAL [(length[,decimals])], NUMERIC [(length[,decimals])]		**DECIMAL(65,30) / NUMERIC(65,30)** NUMERIC will get converted in DECIMAL

For: float(M,D), double(M,D) or decimal(M,D), M must be >= D

Here, **(M,D)** means than values can be stored with up to ***M*** digits in total, of which ***D*** digits may be after the decimal point.

UNSIGNED prohibits negative values.

datatype – date and time

Datatypes	Size	Description
YEAR	1 byte	YYYY
DATE	3 bytes	YYYY-MM-DD
TIME	3 bytes	HH:MM:SS
DATETIME	8 bytes	YYYY-MM-DD hh:mm:ss

datatype – boolean

```
CREATE TABLE temp (col1 INT ,col2 BOOL,  col3 BOOLEAN);
```

Note:

- BOOL and BOOLEAN are **synonym of TINYINT(1)**

Use a CREATE TABLE statement to specify the layout of your table.

```
CREATE TABLE `123` (c1 INT, c2 VARCHAR(10));
```

Remember:

- Max 4096 columns per table provided the row size <= 65,535 Bytes

create table

Use a **CREATE TABLE** statement to specify the layout of your table.

Note:

- **USER TABLES**: This is a collection of tables created and maintained by the user. Contain USER information.
- **DATA DICTIONARY**: This is a collection of tables created and maintained by the MySQL Server. It contains database information. All data dictionary tables are owned by the SYS user.

create table

Use a **CREATE TABLE** statement to specify the layout of your table.

Remember:

- by default, tables are created in the default database, using the InnoDB storage engine.
- table name should not begin with a number or special symbols.
- table name can start with _table_name (underscore) or $table_name (dollar sign)
- table name and column name can have max 64 char.
- multiple words as table_name is invalid, if you want to give multiple words as table_name then give it in `table_name` (backtick)
- error occurs if the table exists.
- error occurs if there is no default database.
- error occurs if the database does not exist.

Note:

- Table names are stored in lowercase on disk. MySQL converts all table names to lowercase on storage. This behavior also applies to database names and table aliases.
  e.g. show variables like 'lower_case_table_names';

create table

syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_defineation, . . .)
    [table_options]
    [partition_options]
```

create_definition:

```
    col_name column_definition
```

column_definition:

```
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [reference_definition]
  | data_type [GENERATED ALWAYS] AS (expression)  [VIRTUAL]
     [VISIBLE | INVISIBLE]
```

table_options:

```
ENGINE [=] engine_name
```

e.g.

-
  ```
  CREATE TABLE student (
     ID INT,
     firstName VARCHAR(45),
     lastName VARCHAR(45),
     DoB DATE,
     emailID VARCHAR(128)
  );
  ```

```
show engines;
set default_storage_engine = memory
```

default value

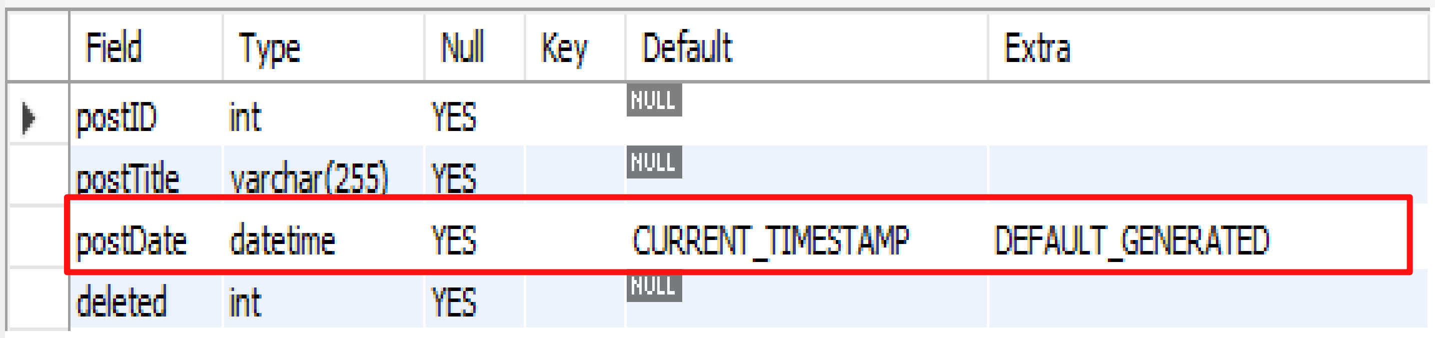
The DEFAULT specifies a default value for the column.

default value

col_name data_type DEFAULT value

The **DEFAULT** specifies a **default** value for the column.

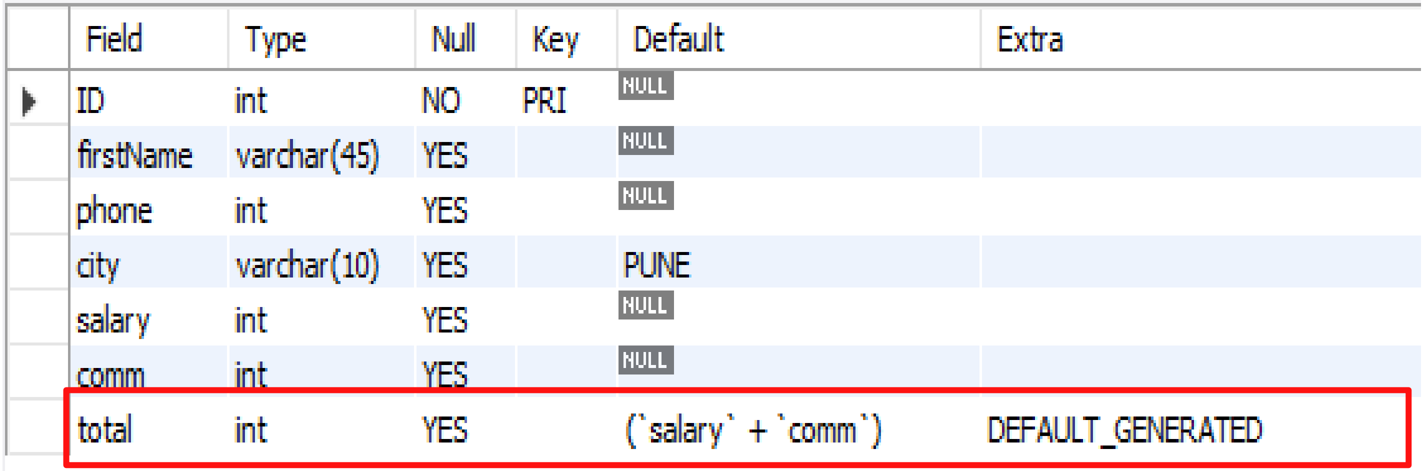
-
  ```
  CREATE TABLE posts (
     postID INT,
     postTitle VARCHAR(255),
     postDate DATETIME DEFAULT NOW(),
     deleted INT
  );
  ```



Field	Type	Null	Key	Default	Extra
postID	int	YES		NULL	
postTitle	varchar(255)	YES		NULL	
postDate	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
deleted	int	YES		NULL	

# version 8.0 and above.

-
  ```
  CREATE TABLE empl (
     ID INT PRIMARY KEY,
     firstName VARCHAR(45),
     phone INT,
     city VARCHAR(10) DEFAULT 'PUNE',
     salary INT,
     comm INT,
     total INT DEFAULT(salary + comm)
  );
  ```



Field	Type	Null	Key	Default	Extra
ID	int	NO	PRI	NULL	
firstName	varchar(45)	YES		NULL	
phone	int	YES		NULL	
city	varchar(10)	YES		PUNE	
salary	int	YES		NULL	
comm	int	YES		NULL	
total	int	YES		(`salary` + `comm`)	DEFAULT_GENERATED

insert rows

INSERT is used to add a single or multiple tuple to a relation. We must specify the relation name and a list of values for the tuple. **The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.**

You can insert data using following methods:

- INSERT ... VALUES
- INSERT ... SET
- INSERT ... SELECT

INSERT can violate for any of the four types of constraints.

Important:

- If an attribute value is not of the appropriate data type.
- Entity integrity can be violated if a key value in the new tuple *t* already exists in another tuple in the relation r(R).
- Entity integrity can be violated if any part of the primary key of the new tuple *t* is NULL.
- Referential integrity can be violated if the value of any foreign key in *t* refers to a tuple that does not exist in the referenced relation.

INSERT will also fail in following cases.

Important :

- Your database table has **X** columns, Where as the **VALUES** you are passing are for (**X-1**) or (**X+1**). This mismatch of column-values will giving you the error.
- Inserting a string into a string column (CHAR, VARCHAR, TEXT, or BLOB) that exceeds the column maximum length. The value is truncated to the column maximum length.

- **INSERT** is used to add a single or multiple tuple to a relation. We must specify the relation name and a list of values for the tuple. **The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.**
- A second form of the **INSERT** statement allows the user to specify explicit attribute names that correspond to the values provided in the **INSERT** command. This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple. However, the values must include all attributes with **NOT NULL** specification and no default value. Attributes with **NULL** allowed or **DEFAULT** values are the ones that can be left out.

insert rows using values

dml- insert … values

INSERT inserts new row(s) into an existing table. The INSERT ... VALUES

INSERT [IGNORE] [INTO] tbl_name [(col_name, . . .)] { VALUES | VALU E } ({ expr | DEFAULT }, . . .), (. . .), . . .