

*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 }

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 collect data from different resource (like Weather data, Geographical data, Finance data, Scientific data, Transport data, Cultural data [Cultural means: the ideas, customs, and social behaviour of a particular people or society..], 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" refers to a **table** within the database that follows the principles of the relational model **OR** an **entity** in ERD) than contain attributes.

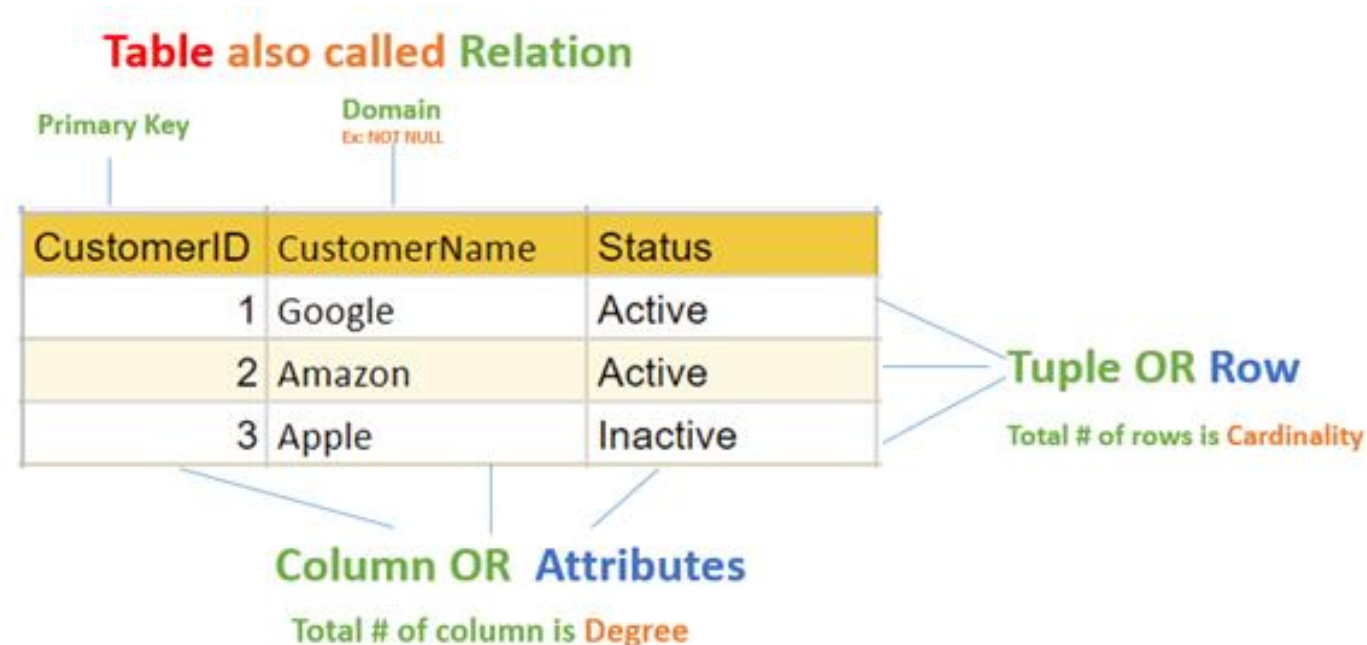
Relationship: In database, relationship is that how the two entities are **connected** to each other, i.e. what kind of relationship type they hold between them.

Primary/Foreign key is used to specify this relationship.

Remember:

Foreign Key is also know as

- referential constraint
- referential integrity constraint. (Referential integrity constraint is the state of a database in which all values of all foreign keys are valid.)



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

File System VS DBMS

```
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];
```

file-oriented system

File Anomalies

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
.		
.		
500	sam	3500
.		
3	rajan	4500
.		
.		
500	sam	3500
.		
1000	amit	2300

c:\employee.txt

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

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

file-oriented system

File Anomalies

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.
- **Flexibility:** File systems provide flexibility in storing various types of data, including text documents, images, audio, video, and more
- **Cost-Effectiveness:** File systems often do not incur licensing costs, making them cost-effective for basic data storage needs.
- **Editing:** It is easy to edit any information stored in computers in form of files.
- **Remote access:** It is possible to access data from 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.

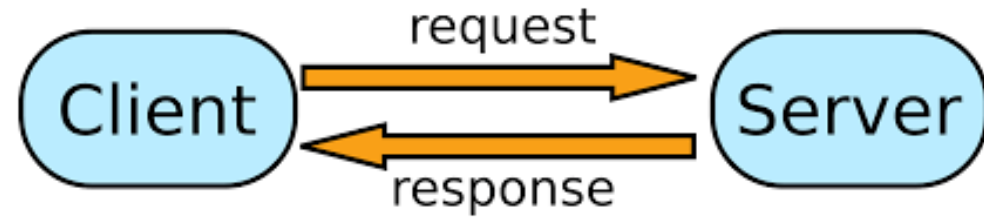


relational database management system?

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)*



- A server is a computer program or a machine that provides service to another computer program, known as the client.

object relational database management system?

An object database is a database management system in which information is represented in the form of objects.

PostgreSQL is the most popular pure ORDBMS. Some popular databases including Microsoft SQL Server, Oracle, and IBM DB2 also support objects and can be considered as ORDBMS.

Advantage of ORDBMS

- Function/Procedure overloading.
- Extending server functionality with external functions written in C or Java.
- User defined data types.
- Inheritance of tables under other tables.

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 (column) 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, 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, 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

Remember:

- A **Binary Large Object (BLOB)** is a MySQL data type that can store binary data such as multimedia, and PDF files.
- A **Character Large Object(CLOB)** is aa MySQL data type which is used to store large amount of textual data. Using this datatype, you can store data up to 2,147,483,647 characters.
- A number is a mathematical value used to count, measure, and label.



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?

An entity in DBMS is a real-world object that has certain properties called attributes that define the nature of the 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**.

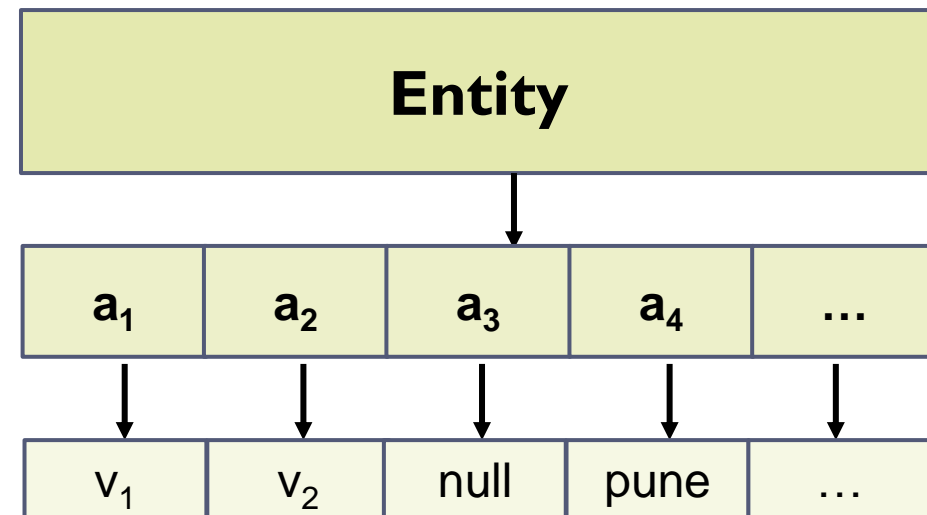
Every entity has its own characteristics.

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)



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.

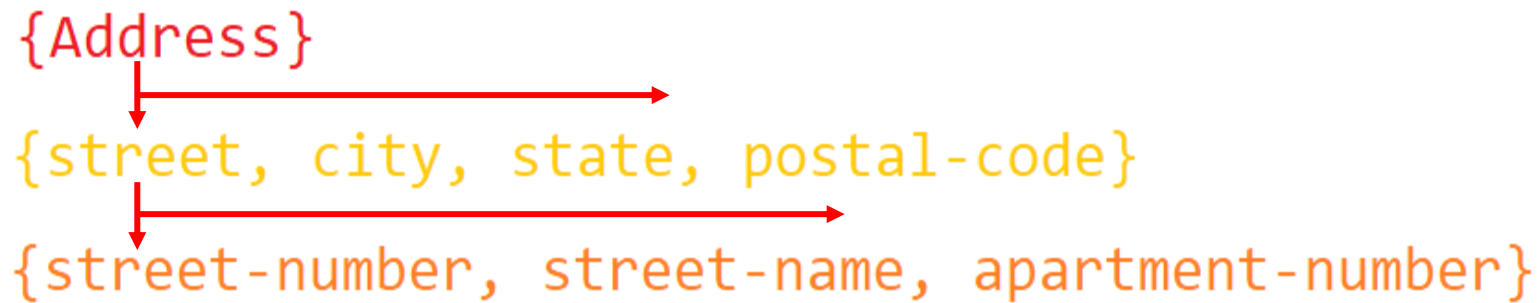
- **Atomic Attribute:** An attribute that cannot be divided into smaller independent attribute is known as atomic attribute.
e.g. ID's, PRN, age, gender, zip, marital status cannot further divide.
- **Single Value Attribute:** An attribute that has only single value is known as single valued attribute.
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 Attribute

Person Entity

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



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@yahoo.com, ...]

What is an Prime, Non-Prime Attribute?

Prime attribute (*Entity integrity*)

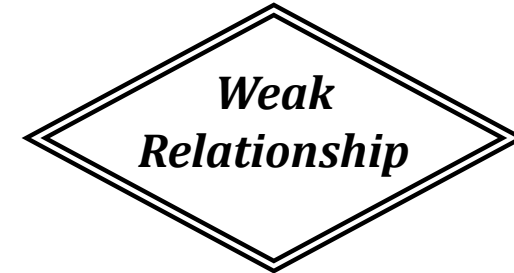
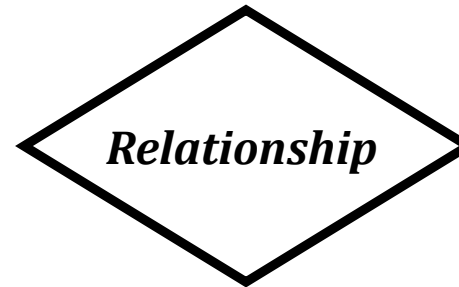
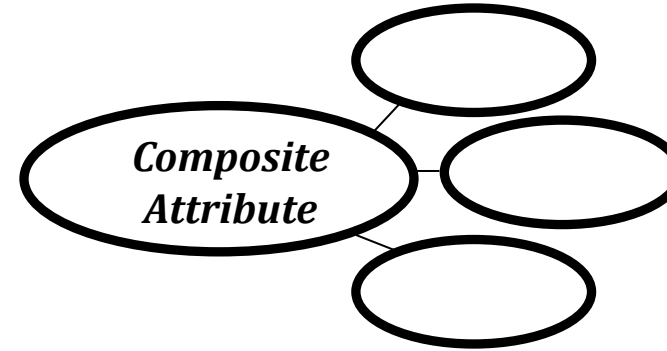
An attribute, which is a **part of the prime-key** (candidate key), is known as a prime attribute.

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

entity relationship diagram symbols



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. This relationship is also known as identifying relationship.

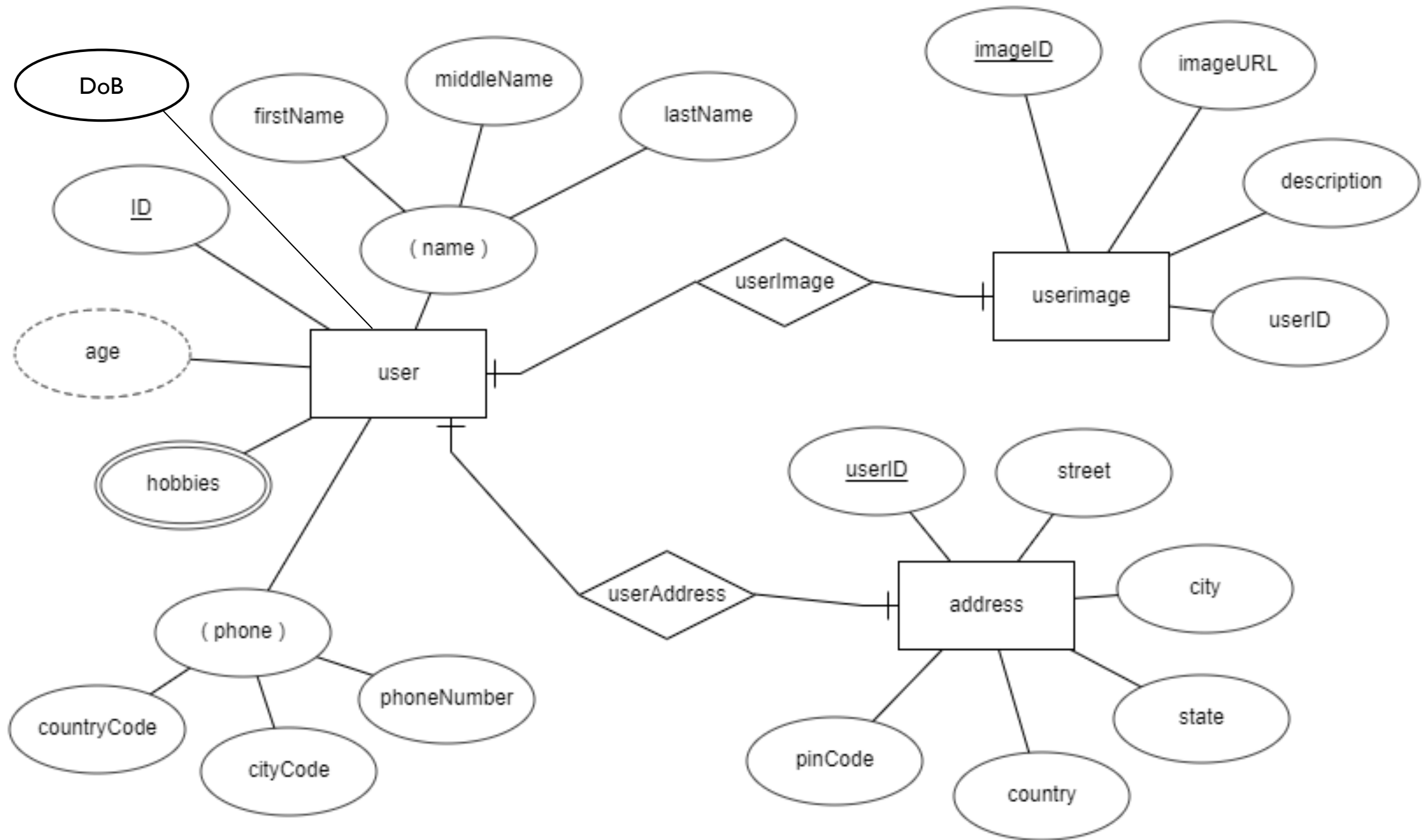
Example 1 – A loan entity can not be created for a customer if the customer doesn't exist

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

Example 3 – A customer address entity can not be created for the customer if the customer 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 **$|R|$** .

Cardinality is the numerical relationship between rows of one table and rows in another. Common cardinalities include *one-to-one*, *one-to-many*, and *many-to-many*.

- **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..

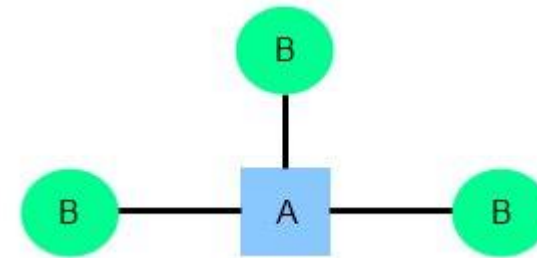
Common relationships

Common relationship

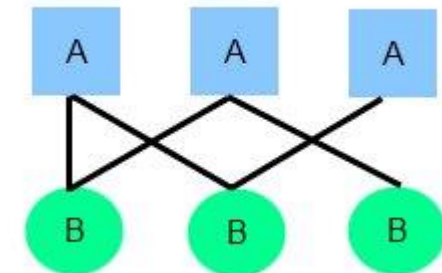
1. one-to-one (1:1)



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



3. many-to-many (M:N)

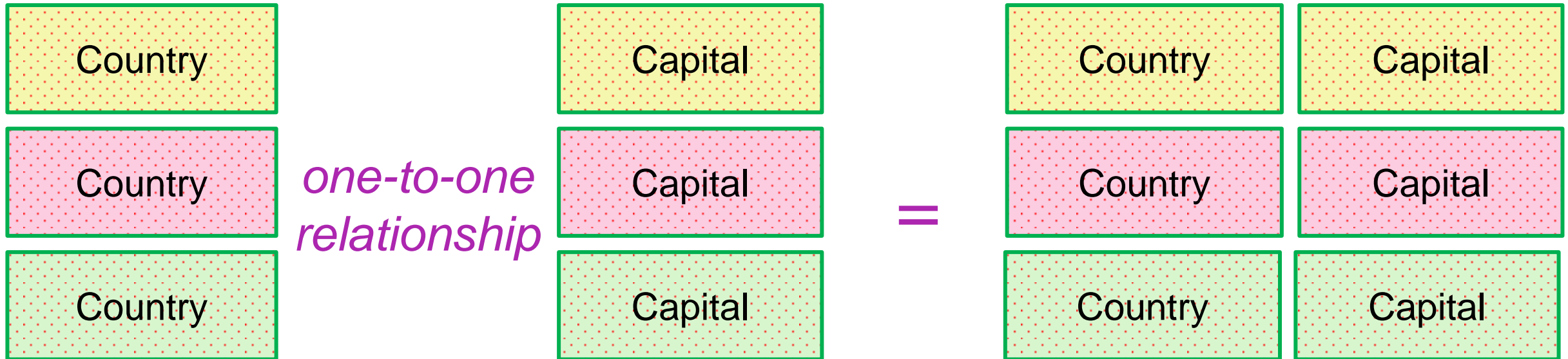


one-to-one relationship

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.

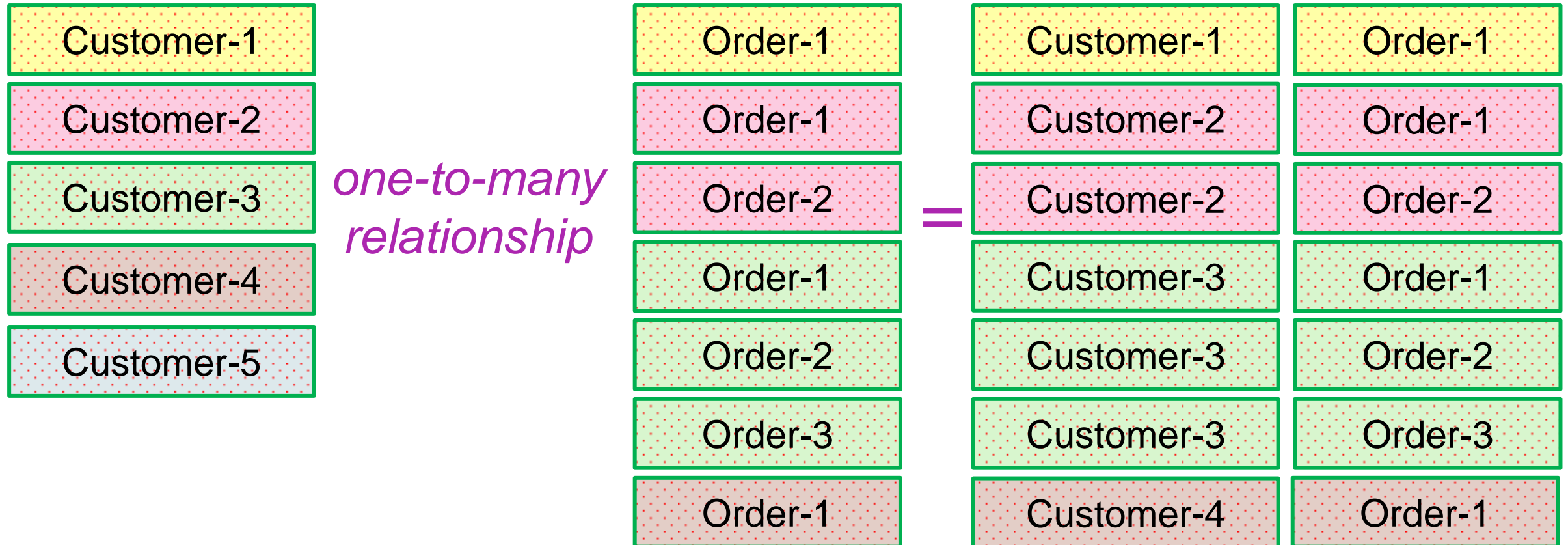


one-to-many relationship

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.

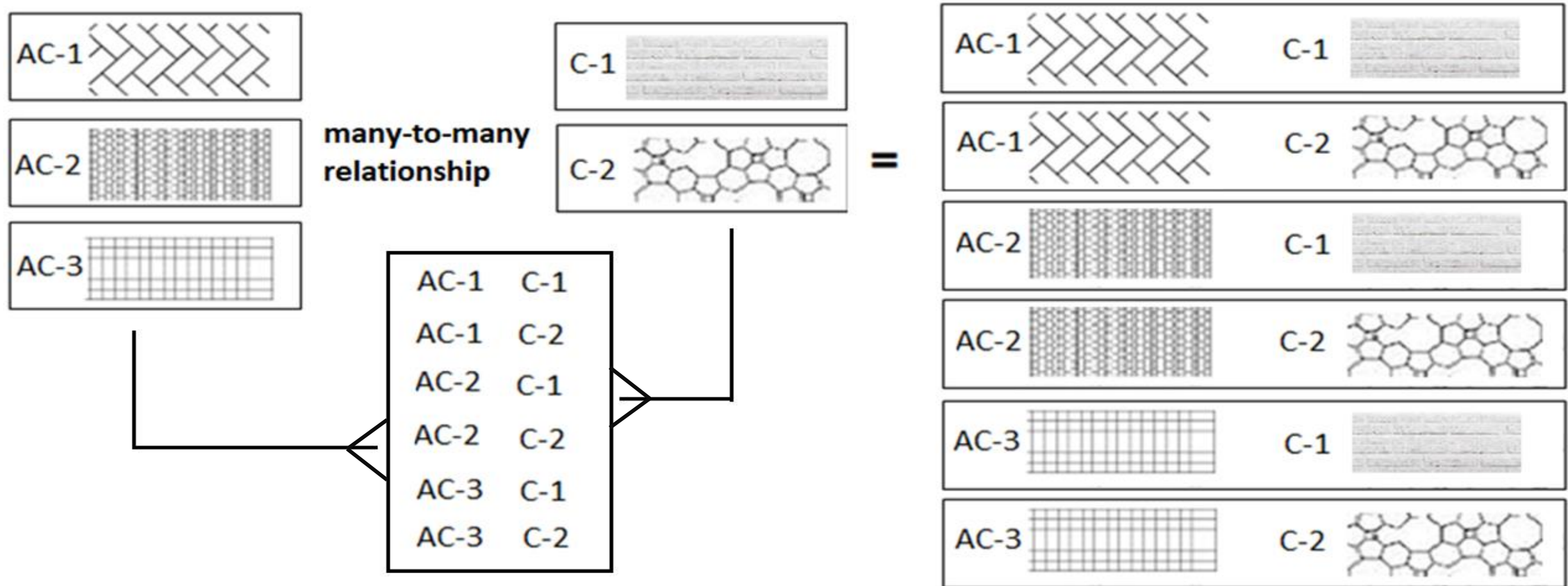
a *one-to-many* relationship is a type of cardinality that refers to the relationship between two entities R and S in which an element of R may be linked to many elements of S , but a member of S is linked to only one element of R .



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.



Schema: A schema is a collection of database objects (like table, columns , primary key, foreign key, views, etc.) associated with one particular database username. This username is called the schema owner. You may have one or multiple schemas in a database.

What is schema and instance

Instance

- The data stored in database at a particular moment of time is called instance of database.

For example, lets say we have a single table student in the database, today the table has 100 records, so today the instance of the database has 100 records. Lets say we are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table.

An instance of a relation is a set of tuples, also called records

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?

Remember:

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

what is sql?

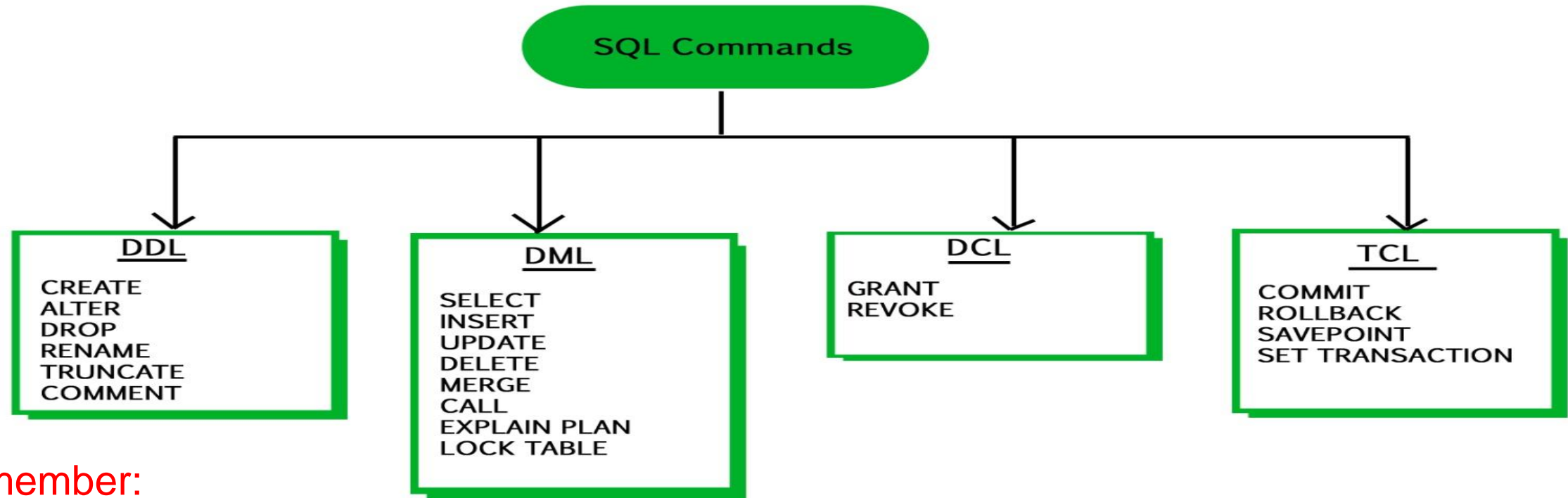
SQL (Structured Query Language) is a database language designed and developed for managing data in relational database management systems (RDBMS). SQL is common language for all Relational Databases.



Remember:

what is sql?

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



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.

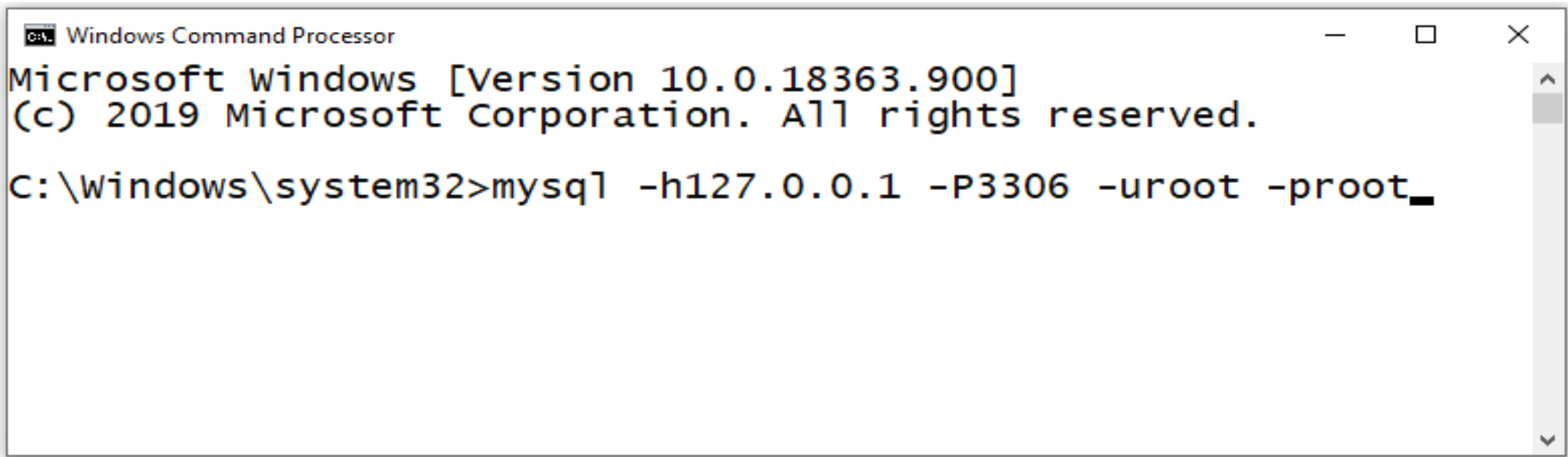
Reconnect to the server	\r
Execute a system shell command	\!
Exit mysql	\q
Change your mysql prompt.	prompt str or \R str

Login to MySQL

Default port for MySQL Server: 3306

login

- 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]

A screenshot of a Windows Command Processor window. The title bar reads "C:\ Windows Command Processor". The window content shows the following text: "Microsoft Windows [Version 10.0.18363.900]", "(c) 2019 Microsoft Corporation. All rights reserved.", and "C:\Windows\system32>mysql -h127.0.0.1 -P3306 -uroot -proot_". The cursor is at the end of the command line.

```
C:\ Windows Command Processor
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_
```

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

datatypes

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

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)		

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

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

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

Remember:

- Max 4096 columns per table provided the row size \leq 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';

syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
    (create_definition, . . . )  
    [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
```

create table

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

col_name data_type DEFAULT value

default value

The DEFAULT specifies a default value for the column.

- BLOB, TEXT, GEOMETRY or JSON column can't have a default value.
e.g. CREATE TABLE temp(c1 TEXT DEFAULT('PUNE'));

```
INSERT [IGNORE] [INTO] tbl_name [PARTITION (partition_name [, partition_name] ...)] [  
(col_name, . . .) ] { VALUES | VALUE } ( { expr | DEFAULT }, . . . ), ( . . . ), . . .
```

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

Do not use the ***** operator in your SELECT statements. Instead, use column names. Reason is that in MySQL Server scans for all column names and replaces the ***** with all the column names of the table(s) in the SELECT statement. Providing column names avoids this search-and-replace, and enhances performance.

SELECT statement...

```
SELECT what_to_select  
FROM which_table  
WHERE conditions_to_satisfy;
```

SELECT CLAUSE

The **SELECT** statement retrieves or extracts data from tables in the database.

- You can use one or more tables separated by comma to extract data.
- You can fetch one or more fields/columns in a single **SELECT** command.
- You can specify star (*) in place of fields. In this case, **SELECT** will return all the fields.
- **SELECT** can also be used to retrieve rows computed without reference to any table e.g. **SELECT 1 + 2;**



Capabilities of SELECT Statement

1. SELECTION
2. PROJECTION
3. JOINING



Capabilities of *SELECT* Statement

➤ *SELECTION*

Selection capability in SQL is to choose the record's/row's/tuple's in a table that you want to return by a query.

R

EMPNO	ENAME	JOB	HIREDATE	DEPTNO
1	Saleel	Manager	1995-01-01	10
2	Janhavi	Sales	1994-12-20	20
3	Snehal	Manager	1997-05-21	10
4	Rahul	Account	1997-07-30	10
5	Ketan	Sales	1994-01-01	30



Capabilities of *SELECT* Statement

➤ *PROJECTION*

Projection capability in SQL to choose the column's/attribute's/field's in a table that you want to return by your query.

R

EMPNO	ENAME	JOB	HIREDATE	DEPTNO
1	Saleel	Manager	1995-01-01	10
2	Janhavi	Sales	1994-12-20	20
3	Snehal	Manager	1997-05-21	10
4	Rahul	Account	1997-07-30	10
5	Ketan	Sales	1994-01-01	30



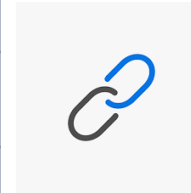
Capabilities of *SELECT* Statement

➤ JOINING

Join capability in SQL to bring together data that is stored in different tables by creating a link between them.

R

EMPNO	ENAME	JOB	HIREDATE	DEPTNO
1	Saleel	Manager	1995-01-01	20
2	Janhavi	Sales	1994-12-20	10
3	Snehal	Manager	1997-05-21	10
4	Rahul	Account	1997-07-30	20
5	Ketan	Sales	1994-01-01	30




S

DEPTNO	DNAME	LOC
10	HRD	PUNE
20	SALES	BARODA
40	PURCHASE	SURAT



SELECTION Process

SELECT * FROM <table_references>



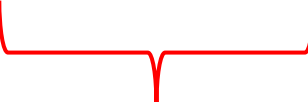
selection-list | field-list | column-list

Remember:

- Here, " * " is known as metacharacter (all columns)

PROJECTION Process

SELECT column-list FROM <table_references>



selection-list | field-list | column-list

Remember:

- Position of columns in SELECT statement will determine the position of columns in the output (as per user requirements)

```
UPDATE tbl_name SET col_name1 = { expr1 | DEFAULT } [, col_name2 = { expr2 | DEFAULT } ] ...  
[WHERE where_condition]
```

In a **SET** statement, **=** is treated identically to **:=**

single-table update

UPDATE is used to change/modify the values of some attributes of one or more selected tuples.

```
DELETE FROM tbl_name  
[WHERE where_condition]
```

single-table delete

DELETE is used to delete tuples from a relation.

constraints

CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns. The purpose of inducing constraints is to enforce the integrity of a database.

CONSTRAINTS can be classified into two types –

- *Column Level*
- *Table Level*

Remember:

- **PRI** => primary key
- **UNI** => unique key
- **MUL** => is basically an index that is neither a **primary key** nor a **unique key**. The name comes from "multiple" because multiple occurrences of the same value are allowed.

constraints

To limit or to restrict or to check or to control.

Note:

- a table with a foreign key that references another table's primary key is **MUL**.
- If more than one of the Key values applies to a given column of a table, Key displays the one with the highest priority, in the order **PRI**, **UNI**, and **MUL**.
- If a table has a PRIMARY KEY or UNIQUE NOT NULL index that consists of a single column that has an integer type, you can use **_rowid** to refer to the indexed column in SELECT statements.

Remember:

- A primary key cannot be NULL.
- A primary key value must be unique.
- A table has only one primary key.
- The primary key values cannot be changed, if it is referred by some other column.
- **An index can consist of 16 columns, at maximum. Since a PRIMARY KEY constraint automatically adds an index, it can't have more than 16 columns.**

col_name data_type PRIMARY KEY

PRIMARY KEY constraint

A primary key is a special column (or set of combined columns) in a relational database table, that is used to uniquely identify each record.

Note:

- Primary key in a relation is always associated with an **INDEX** object.
- If, we give on a column a combination of **NOT NULL & UNIQUE** key then it behaves like a PRIMARY key.
- If, we give on a column a combination of **UNIQUE key & AUTO_INCREMENT** then also it behaves like a PRIMARY key.


```
ALTER TABLE table_name  
  ADD [ CONSTRAINT constraint_name ]  
    PRIMARY KEY (column1, column2, . . .  
column_n)
```

Add / Drop Primary Key

```
ALTER TABLE table_name  
  DROP PRIMARY KEY
```

Remember:

- A unique key can be NULL.
- A unique key value must be unique.
- A table can have multiple unique key.
- A column can have unique key as well as a primary key.

col_name data_type **UNIQUE KEY**

UNIQUE KEY constraint

A **UNIQUE key** constraint is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

Note:

- Unique key in a relation is always associated with an ***INDEX*** object.

```
ALTER TABLE table_name  
  ADD [ CONSTRAINT constraint_name ]  
    UNIQUE (column1, column2, . . . column_n)
```

Add / Drop Unique Key

```
ALTER TABLE table_name  
  DROP INDEX constraint_name;
```

```
[CONSTRAINT [symbol] FOREIGN KEY (col_name, ...) REFERENCES tbl_name  
(col_name,...)  
[ON DELETE reference_option]  
[ON UPDATE reference_option]
```

reference_option: CASCADE | SET NULL

FOREIGN KEY constraint

- A **FOREIGN KEY** is a **key** used to link two tables together.
- A **FOREIGN KEY** is a field (or collection of fields) in one table that refers to the **PRIMARY KEY** in another table.
- The table containing the **foreign key** is called the child table, and the table containing the candidate **key** is called the referenced or parent table.

constraints – foreign key

Remember:

- A foreign key can have a different column name from its primary key.
- DataType of primary key and foreign key column must be same.
- It ensures rows in one table have corresponding rows in another.
- Unlike the Primary key, they do not have to be unique.
- Foreign keys can be null even though primary keys can not.

Note:

- The table containing the FOREIGN KEY is referred to as the child table, and the table containing the PRIMARY KEY (referenced key) is the parent table.
- PARENT and CHILD tables must use the same storage engine,
- and they cannot be defined as temporary tables.

```
ALTER TABLE table_name  
  ADD [ CONSTRAINT constraint_name ]  
    FOREIGN KEY (child_col1, child_col2, . . . child_col_n)  
    REFERENCES parent_table (parent_col1, parent_col2, . . . parent_col_n);
```

Add / Drop Foreign Key

```
ALTER TABLE table_name  
  DROP FOREIGN KEY constraint_name
```

col_name data_type CHECK (expr)

Check Constraint

CHECK condition expressions must follow some rules.

- Literals, deterministic built-in functions, and operators are permitted.
 - Non-generated and generated columns are permitted, except columns with the `AUTO_INCREMENT` attribute.
 - Sub-queries are not permitted.
 - Environmental variables (such as `CURRENT_USER`, `CURRENT_DATE`, ...) are not permitted.
 - Non-Deterministic built-in functions (such as `AVG`, `COUNT`, `RAND`, `LAST_INSERT_ID`, `FIRST_VALUE`, `LAST_VALUE`, ...) are not permitted.
 - Variables (system variables, user-defined variables, and stored program local variables) are not permitted.
 - Stored functions and user-defined functions are not permitted.
-

Note:

Prior to MySQL 8.0.16, `CREATE TABLE` permits only the following limited version of table `CHECK` constraint syntax, which is parsed and ignored.

Remember:

If you omit the constraint name, MySQL automatically generates a name with the following convention:

- `table_name_chk_n`


```
ALTER TABLE table_name  
  ADD [ CONSTRAINT constraint_name ]  
    CHECK (condition)
```

Add / Drop Check Constraint

```
ALTER TABLE table_name  
  DROP { CHECK | CONSTRAINT }  
constraint_name
```

alter table

ALTER TABLE changes the structure of a table.

Note:

- you can add or delete columns,
- create or destroy indexes,
- change the type of existing columns, or
- rename columns or the table itself.
- You cannot change the position of columns in table structure. If not, then what? create a new table with **SELECT statement**.

ALTER TABLE *tbl_name*

[*alter_specification* [, *alter_specification*] . . .

- | **ADD** [**COLUMN**] *col_name column_definition* [**FIRST** | **AFTER** *col_name*]
- | **ADD** [**COLUMN**] (*col_name column_definition*, . . .)
- | **MODIFY** [**COLUMN**] *col_name column_definition* [**FIRST** | **AFTER** *col_name*]
- | **DROP** [**COLUMN**] *col_name*
- | **RENAME** [**TO**|**AS**] *new_tbl_name*
- | **RENAME COLUMN** *old_col_name* **TO** *new_col_name*
- | **ALTER** [**COLUMN**] *col_name* { **SET DEFAULT** {*literal* | (*expr*)} | **DROP DEFAULT** | **SET** { **VISIBLE** | **INVISIBLE**}}

`DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name] ...`

drop table

Remember:

- DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
- DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back (DDL statements are auto committed).
- Dropping a TABLE also drops any TRIGGERS for the table.
- Dropping a TABLE also drops any INDEX for the table.
- Dropping a TABLE will not drop any VIEW for the table.
- If you try to drop a PARENT/MASTER TABLE, it will not get dropped.

Do not use the * operator in your SELECT statements. Instead, use column names. Reason is that in MySQL Server scans for all column names and replaces the * with all the column names of the table(s) in the SELECT statement. Providing column names avoids this search-and-replace, and enhances performance.

continue with SELECT statement...

```
SELECT what_to_select  
FROM which_table  
WHERE conditions_to_satisfy;
```

The asterisk symbol “ * ” can be used in the SELECT clause to denote “all attributes.”

column - alias

A programmer can use an alias to temporarily assign another name to a **column** or **table** for the duration of a *SELECT* query.

In the selection-list, a quoted column alias can be specified using identifier (`) or string quote (' or ") characters.

Note:

- Assigning an alias_name does not actually rename the column or table.
- You cannot use alias in an expression.

select statement - alias

`SELECT A_1 [[AS] alias_name], A_2 [[AS] alias_name], . . . , A_N FROM r [[AS] alias_name]`


column-name as new-name


table-name as new-name

Remember:

- A select_expr can be given an alias using **AS alias_name**. The alias is used as the expression's column name and can be used in **GROUP BY**, **HAVING**, or **ORDER BY** clauses.
- The **AS** keyword is optional when aliasing a select_expr with an identifier.
- Standard SQL **disallows** references to column aliases in a **WHERE** clause.
- A table reference can be aliased using **tbl_name alias_name** or **tbl_name AS alias_name**
- If the column alias contains spaces, **put it in quotes**.
- Alias name is **max 256 characters**.

assignment_operator

= (assignment), :=

comparison functions and operator

Comparison operations result in a value of 1 (**TRUE**), 0 (**FALSE**), or **NULL**.

comparison functions and operator

1. *arithmetic_operators:*

* | / | DIV | % | MOD | - | +

2. *comparison_operator:*

= | <=> | >= | > | <= | < | <> | !=

3. *boolean_predicate:*

IS [NOT] NULL
| expr <=> null

4. *predicate:*

expr [NOT] LIKE expr [ESCAPE char]
| expr [NOT] IN (expr1, expr2, ...)
| expr [NOT] IN (subquery)
| expr [NOT] BETWEEN expr1 AND expr2

5. *logical_operators*

{ AND | && } | { OR | || }

6. *assignment_operator*

= (assignment), :=

- SELECT 23 DIV 6 ; #3
- SELECT 23 / 6 ; #3 .8333

Note:

- AND has higher precedence than OR.

column - expressions

select statement - expressions

Column EXPRESSIONS

SELECT A_1, A_2, A_3, A_4 , expressions, . . . FROM r

- SELECT 1001 + 1;
- SELECT 1001 + '1';
- SELECT '1' + '1';
- SELECT '1' + 'a1';
- SELECT '1' + '1a';
- SELECT 'a1' + 1;
- SELECT '1a' + 1;
- SELECT 1 + -1;
- SELECT 1 + -2;
- SELECT -1 + -1;
- SELECT -1 - 1;
- SELECT -1 - -1;
- SELECT 123 * 1;
- SELECT -123 * 1;
- SELECT 123 * -1 ;
- SELECT -123 * -1;
- SELECT 2 * 0;
- SELECT 2435 / 1;
- SELECT 2 / 0;
- SELECT '2435Saleel' / 1;
- SELECT sal, sal + 1000 AS 'New Salary' FROM emp;
- SELECT sal, comm, sal + comm FROM emp;
- SELECT sal, comm, sal + IFNULL(comm, 0) FROM emp;
- SELECT ename, ename = ename FROM emp;
- SELECT ename, ename = 'smith' FROM emp;
- SELECT c1, c1 / 1 R1 FROM numberString;

Note:

If any expression evaluated with NULL, returns NULL.

- SELECT 2 + NULL ;
- SELECT 2 * NULL ;
- SELECT 2 - NULL ;
- SELECT 2 / NULL ;

identifiers

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, stored functions, triggers, partition, tablespace, and other object names are known as **identifiers**.

identifiers

The maximum length for each type of identifiers like (Database, Table, Column, Index, Constraint, View, Stored Program, Compound Statement Label, User-Defined Variable, Tablespace) is **64 characters**, whereas for Alias is **256 characters**.

- You can refer to a table within the default database as
 1. `tbl_name`
 2. `db_name.tbl_name`.
- You can refer to a column as
 1. `col_name`
 2. `tbl_name.col_name`
 3. `db_name.tbl_name.col_name`.

control flow functions

control flow functions - ifnull

IFNULL function

MySQL IFNULL() takes two expressions, if the first expression is not NULL, it returns the first expression. Otherwise, it returns the second expression, **it returns either numeric or string value.**

IFNULL(expression1, expression2)

IF function

If **expr1 is TRUE or expr1 <> 0 or expr1 <> NULL**, then IF() returns expr2, otherwise it returns expr3, **it returns either numeric or string value.**

IF(expr1, expr2 , expr3)