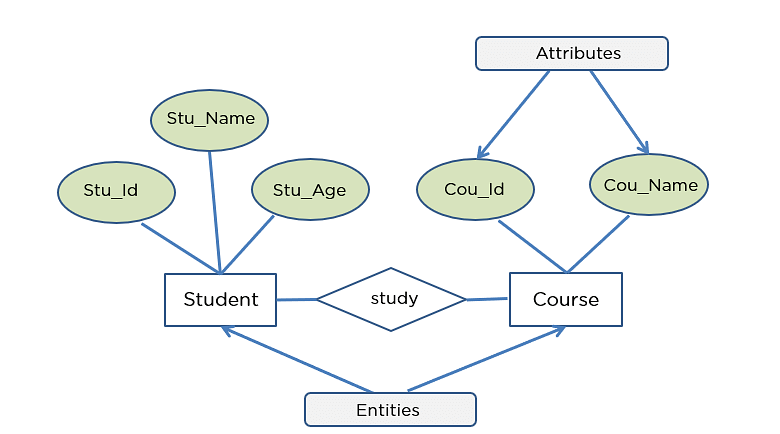
ER Diagrams in DBMS: Entity Relationship Diagram Model

An Entity Relationship Diagram is a diagram that represents relationships among entities in a database. It is commonly known as an ER Diagram. An ER Diagram in [DBMS](https://www.simplilearn.com/what-is-database-management-article) plays a crucial role in designing the database. Today’s business world previews all the requirements demanded by the users in the form of an ER Diagram. Later, it's forwarded to the database administrators to design the database.



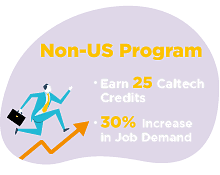
What is an ER Diagram?

An Entity Relationship Diagram (ER Diagram) pictorially explains the relationship between entities to be stored in a database. Fundamentally, the ER Diagram is a structural design of the database. It acts as a framework created with specialized symbols for the purpose of defining the relationship between the database entities. ER diagram is created based on three principal components: entities, attributes, and relationships.

The following diagram showcases two entities - Student and Course, and their relationship. The relationship described between student and course is many-to-many, as a course can be opted by several students, and a student can opt for more than one course. Student entity possesses attributes - Stu\_Id, Stu\_Name & Stu\_Age. The course entity has attributes such as Cou\_ID & Cou\_Name.

Become a Skilled Web Developer in Just 9 Months!

Caltech PGP Full Stack Development[EXPLORE PROGRAM](https://www.simplilearn.com/pgp-full-stack-web-development-certification-training-course?source=GhPreviewCTABanner)



What is an ER Model?

An Entity-Relationship Model represents the structure of the [database](https://www.simplilearn.com/tutorials/sql-tutorial/create-mysql-database) with the help of a diagram. ER Modelling is a systematic process to design a database as it would require you to analyze all data requirements before implementing your database.

History of ER models

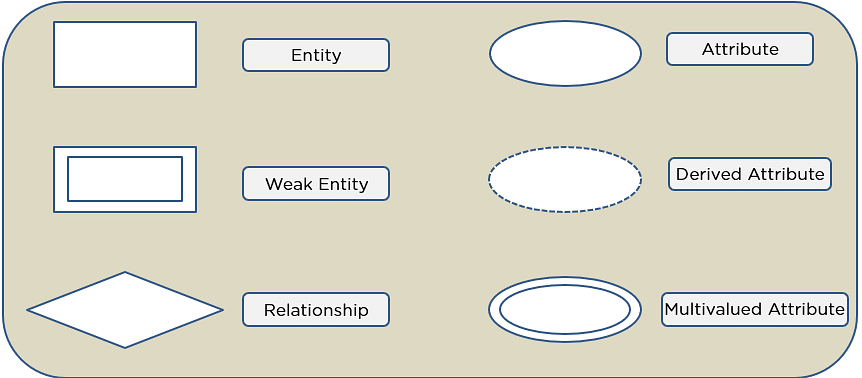
Peter Chen proposed ER Diagrams in 1971 to create a uniform convention that can be used as a conceptual modeling tool. Many models were presented and discussed, but none were suitable. The data structure diagrams offered by Charles Bachman also inspired his model.

Why Use ER Diagrams in DBMS?

* ER Diagram helps you conceptualize the database and lets you know which fields need to be embedded for a particular entity
* ER Diagram gives a better understanding of the information to be stored in a database
* It reduces complexity and allows database designers to build databases quickly
* It helps to describe elements using Entity-Relationship models
* It allows users to get a preview of the logical structure of the database

Symbols Used in ER Diagrams

* Rectangles: This Entity Relationship Diagram symbol represents entity types
* Ellipses: This symbol represents attributes
* Diamonds: This symbol represents relationship types
* Lines: It links attributes to entity types and entity types with other relationship types
* Primary key: Here, it underlines the attributes
* Double Ellipses: Represents multi-valued attributes



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.

**SQL(Structured Query Language)** commands are used to create and maintain Databases. **Data Definition Language(DDL)** is a type of SQL command used to define the [components of databases](https://www.scaler.com/topics/components-of-dbms/) using Database Management Systems.

DDL in DBMS is used to create or modify the database objects like tables, [views](https://www.scaler.com/topics/views-in-dbms/) etc. These commands deal with defining a schema and adding a table description to it.

DDL commands define the table structure and its column properties. While **DML(Data Manipulation Language)** commands in SQL are used to manipulate the data in the tables. DML commands add, delete and update data in the table using commands like **INSERT**, **UPDATE**, and **DELETE** in SQL.

## What is a DDL ?

**Structured Query Language**(SQL) is a standard language used to create, access, and manipulate databases, being an integral part of Database Management Systems(DBMS). SQL uses various commands to manipulate the database and these commands are divided into 5 parts as follows :

* **DDL :**  
  Data Definition Language(DDL) is used to define [database schema in DBMS](https://www.scaler.com/topics/schema-in-dbms/).
* **DML :**  
  [Data Manipulation Language(DML)](https://www.scaler.com/topics/dml-in-dbms/) is used to manipulate data present in the database.
* **DCL :**  
  Data Control Language(DCL) deals with access rights and data control on the data present in the database.
* **TCL :**  
  Transaction Control Language(TCL) deals with the transactions happening in the database.
* **DQL :**  
  Data Query Language(DQL) is used to retrieve data from the database using SQL queries.

Types of SQL Commands

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



Data Definition Language(DDL) is a subset of SQL and a part of DBMS(Database Management System). DDL consist of Commands to commands like CREATE, ALTER, TRUNCATE and DROP. These commands are used to create or modify the tables in SQL.

**DDL Commands**

* DDL in DBMS is a language that allows user to define the database components and their relationship with each other.
* These commands work with the structure of tables like creating a table, deleting a table, and altering a table.
* DDL commands allow us to define and change the data types of table columns, and integrity constraints while defining the table structure. [**Integrity constraints**](https://www.scaler.com/topics/dbms/integrity-constraints-in-dbms/) are rules set for the data being stored in the table. This is explained in the next section of this article.
* All DDL commands are auto-committed which means the changes done using them are permanently saved in the Database but they can be rolled back.

**The DDL commands in DBMS are as follows :**

1. CREATE
2. ALTER
3. TRUNCATE
4. DROP

Command-1 :

CREATE :

This command is used to create a new table in SQL. The user has to give information like table name, column names, and their datatypes.

Syntax –

**CREATE TABLE table\_name**

**(**

**column\_1 datatype,**

**column\_2 datatype,**

**column\_3 datatype,**

**....**

**);**

Example –

We need to create a table for storing Student information of a particular College. Create syntax would be as below.

**CREATE TABLE Student\_info**

**(**

**College\_Id number(2),**

**College\_name varchar(30),**

**Branch varchar(10)**

**);**

Command-2 :

ALTER :

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.

Syntax –

Syntax to add a column to an existing table.

**ALTER TABLE table\_name**

**ADD column\_name datatype;**

Example –

In our Student\_info table, we want to add a new column for CGPA. The syntax would be as below as follows.

**ALTER TABLE Student\_info**

**ADD CGPA number;**

Command-3 :

TRUNCATE :

This command is used to remove all rows from the table, but the structure of the table still exists.

Syntax –

Syntax to remove an existing table.

**TRUNCATE TABLE table\_name;**

Example –

The College Authority wants to remove the details of all students for new batches but wants to keep the table structure. The command they can use is as follows.

**TRUNCATE TABLE Student\_info;**

Command-4 :

DROP :

This command is used to remove an existing table along with its structure from the Database.

Syntax –

Syntax to drop an existing table.

**DROP TABLE table\_name;**

Example –

If the College Authority wants to change their Database by deleting the Student\_info Table.

**DROP TABLE Student\_info;**

Command -5

RENAME:

It is possible to change name of table with or without data in it using simple RENAME command.

We can rename any table object at any point of time.

Syntax –

**RENAME TABLE <Table Name> To <New\_Table\_Name>;**

Example:

If you want to change the name of the table from Employee to Emp we can use rename command as

**RENAME TABLE Employee To EMP;**

**Notes :**

1. **Some of the essential datatypes in SQL are as follows :**
   * **int/number :**  
     Stores integer numbers
   * **varchar :**  
     Stores string
   * **date :**  
     Stores date
2. Integrity constraints are the rules that a table's data columns must follow.

**Some of the integrity constraints in SQL are as follows :**

* + **PRIMARY KEY :**  
    Primary key is a column that is unique in the table and is used to uniquely identify each record in the table.
  + **FOREIGN KEY :**  
    Foreign key is a column used to define a relationship between two tables by referencing to Primary key of another table.
  + **NOT NULL :**  
    Not null integrity constraint is used to say that this column cannot be null. It has to have some data.

1. SQL commands are **, not case-sensitive**.

**Conclusion**

* **Data Definition Language** is a subset of SQL used to define the structure of the database.
* All DDL commands are **auto-committed** i.e. all the changes are saved permanently in the database.
* The structure of the database is defined using these commands hence they are known as **Data Definition Language** in DBMS.
* CREATE, ALTER, TRUNCATE, and DROP are some of the common DDL commands in DBMS.

# SQL | Join (Inner, Left, Right and Full Joins)

**SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN
* NATURAL JOIN

Consider the two tables below as follows:

**Student**



**StudentCourse**



The simplest Join is INNER JOIN.

### **A. INNER JOIN**

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

**table1**: First table.

**table2**: Second table

**matching\_column**: Column common to both the tables.

***Note****: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*



**Example Queries(INNER JOIN)**

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse

ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:



### **B. LEFT JOIN**

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

***Note****: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.*



**Example Queries(LEFT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:



### **C. RIGHT JOIN**

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

***Note****: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.*



**Example Queries(RIGHT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**



### **D. FULL JOIN**

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values.



**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

**Example Queries(FULL JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**

| **NAME** | **COURSE\_ID** |
| --- | --- |
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | NULL |
| ROHIT | NULL |
| NIRAJ | NULL |
| NULL | 4 |
| NULL | 5 |
| NULL | 4 |

### E. Natural join (⋈)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

Both table must have at list one common column with same column name and same data type.

The two table are joined using Cross join.

DBMS will look for a common column with same name and data type Tuples having exactly same values in common columns are kept in result.

Example:

| **Employee** | | |
| --- | --- | --- |
| **Emp\_id** | **Emp\_name** | **Dept\_id** |
| **1** | **Ram** | **10** |
| **2** | **Jon** | **30** |
| **3** | **Bob** | **50** |

| **Department** | |
| --- | --- |
| **Dept\_id** | **Dept\_name** |
| **10** | **IT** |
| **30** | **HR** |
| **40** | **TIS** |

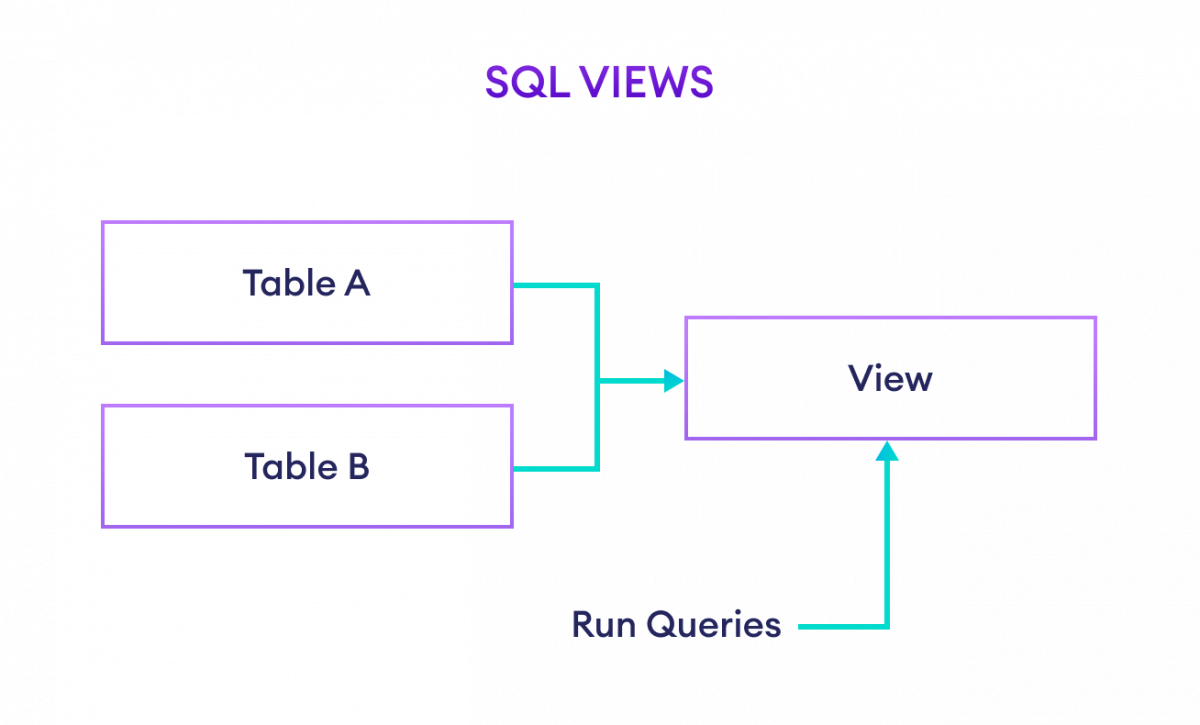
Query: Find all Employees and their respective departments.

Solution: (Employee) ⋈ (Department)

| **Emp\_id** | **Emp\_name** | **Dept\_id** | **Dept\_id** | **Dept\_name** |
| --- | --- | --- | --- | --- |
| **1** | **Ram** | **10** | **10** | **IT** |
| **2** | **Jon** | **30** | **30** | **HR** |
| **Employee data** | | | **Department data** | |

# SQL | Views

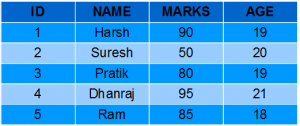
Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

  
**EXAMPLE**:

StudentDetails

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-57.png)

StudentMarks

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-58.png)

**CREATING VIEWS**

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

**Syntax**:

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE condition;

**view\_name**: Name for the View

**table\_name**: Name of the table

**condition**: Condition to select rows

**Examples**:

**Creating View from a single table:**

* In this example we will create a View named DetailsView from the table StudentDetails.
* Query:

**CREATE VIEW DetailsView AS**

**SELECT NAME, ADDRESS**

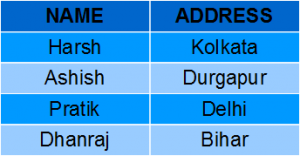
**FROM StudentDetails**

**WHERE S\_ID < 5;**

* To see the data in the View, we can query the view in the same manner as we query a table.

SELECT \* FROM DetailsView;

Output:

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-571.png)

* In this example, we will create a view named StudentNames from the table StudentDetails.
* Query:

CREATE VIEW StudentNames AS

SELECT S\_ID, NAME

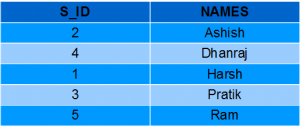
FROM StudentDetails

ORDER BY NAME;

* If we now query the view as,

SELECT \* FROM StudentNames;

Output:

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-64.png)

* **Creating View from multiple tables**: In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement.

Query:

CREATE VIEW MarksView AS

SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS

FROM StudentDetails, StudentMarks

WHERE StudentDetails.NAME = StudentMarks.NAME;

* To display data of View MarksView:

SELECT \* FROM MarksView;

* Output:

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-591.png)

# SQL indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly.

An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and is then referred to one or more specific page numbers.

**Creating an Index:**

**Syntax:**

CREATE INDEX index

ON TABLE column;

where the **index** is the name given to that index and **TABLE** is the name of the table on which that index is created and **column** is the name of that column for which it is applied.

**For multiple columns:**

**Syntax:**

CREATE INDEX index

ON TABLE (column1, column2,.....);

**Unique Indexes:**Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for fast performance, it does not allow multiple values to enter into the table.

**Syntax:**

CREATE UNIQUE INDEX index

ON TABLE column;

### **When should indexes be created:**

* A column contains a wide range of values.
* A column does not contain a large number of null values.
* One or more columns are frequently used together in a where clause or a join condition.

### **When should indexes be avoided:**

* The table is small
* The columns are not often used as a condition in the query
* The column is updated frequently

**Removing an Index:**Remove an index from the data dictionary by using the **DROP INDEX** command.

**Syntax:**

DROP INDEX index;

To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege.

**Altering an Index:**To modify an existing table’s index by rebuilding, or reorganizing the index.

ALTER INDEX IndexName

ON TableName REBUILD;

**Confirming Indexes:**You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

**Syntax:**

select \* from USER\_INDEXES;

It will show you all the indexes present in the server, in which you can locate your own tables too.

**Renaming an index:**You can use the system-stored procedure sp\_rename to rename any index in the database.

**Syntax:**

EXEC sp\_rename

index\_name,

new\_index\_name,

N'INDEX';

**Here are some scenarios along with their explanation related to Indexing:**

1. When executing a query on a table having huge data ( > 100000 rows ), MySQL performs a full table scan which takes much time and the server usually gets timed out. To avoid this always check the explain option for the query within MySQL which tells us about the state of execution. It shows which columns are being used and whether it will be a threat to huge data. On basis of the columns repeated in a similar order in conditions, we can create an index for them in the same order to maximize the speed of the query.
2. The order of the index is of huge importance as we can use the same index in many scenarios. Using only one index we can utilize it in more than one query which different conditions. like for example, in a query, we make a join with a table based on customer\_id wards we also join another join based on customer\_id and order\_date. Then we can simply create a single index by the order of customer\_id, order\_date which would be used in both cases. This also saves storage.
3. We should also be careful to not make an index for each query as creating indexes also take storage and when the amount of data is huge it will create a problem. Therefore, it’s important to carefully consider which columns to index based on the needs of your application. In general, it’s a good practice to only create indexes on columns that are frequently used in queries and to avoid creating indexes on columns that are rarely used. It’s also a good idea to periodically review the indexes in your database and remove any that are no longer needed.
4. Indexes can also improve performance when used in conjunction with sorting and grouping operations. For example, if you frequently sort or group data based on a particular column, creating an index on that column can greatly improve performance. The index allows MySQL to quickly access and sort or group the data, rather than having to perform a full table scan.

## What are CRUD operations?

Any organization that tracks data (such as user accounts, payment information, or other records) needs systems that provide persistent storage, which is usually organized as a database.

A relational database consists of data organized in tables where each table comprises rows and columns. A table can be connected to other tables using primary and [foreign keys](https://www.educative.io/blog/what-is-foreign-key-database).

CRUD refers to the four operations we use to implement persistent storage applications like relational databases. Examples of relational databases include Oracle, Microsoft SQL Server, and [MySQL](https://www.educative.io/blog/mysql-tutorial).

Operations such as security control, transaction control, access, and permission, and performance optimization are all based on CRUD.

| **Letter** | **Operation** | **Function** |
| --- | --- | --- |
| C | Create | Create, Insert |
| R | Read | Select |
| U | Update | Edit |
| D | Delete | Delete |

## Why are CRUD operations so important?

You can’t interact with databases without using CRUD. CRUD is essential for anything requiring persistent storage. Without CRUD, web developers wouldn’t be able to use REST (Representational State Transfer), which is a superset of CRUD used to access HTTP resources. REST APIs are some of the most commonly used in web development.

Basically, software developers wouldn’t be able to complete any project requiring persistent storage without CRUD operations.

On the other end, CRUD is just as crucial for end-users as it is for application developers. The end-users access CRUD operations through an application’s interface. Without CRUD, actions like registering for websites, creating blogs, or using bookmarks would be impossible.

**Benefits of CRUD**

* Facilitates security control by satisfying the various access requirements
* Simplifies application design and makes it more scalable
* Better performance compared to ad-hoc SQL statements

1. Create:

In CRUD operations, 'C' is an acronym for **create,** which *means to add or insert data into the SQL table.* So, firstly we will create a table using CREATE command and then we will use the INSERT INTO command to insert rows in the created table.

**Syntax for table creation:**

1. **CREATE** **TABLE** Table\_Name (ColumnName1 Datatype, ColumnName2 Datatype,..., ColumnNameN Datatype);

**Syntax for insertion of data in table:**

1. **INSERT** **INTO** Table\_Name (ColumnName1,...., ColumnNameN) **VALU**

## 2. Read:

In CRUD operations, 'R' is an acronym for **read,** which means **retrieving or fetching the data from the SQL table.** So, we will use the SELECT command to fetch the inserted records from the SQL table. We can retrieve all the records from a table using an asterisk (\*) in a SELECT query. There is also an option of retrieving only those records which satisfy a particular condition by using the WHERE clause in a SELECT query.

**Syntax to fetch all the records:**

1. **SELECT** \***FROM** TableName;

**Syntax to fetch records according to the condition:**

1. **SELECT** \***FROM** TableName **WHERE** CONDITION;

## 3. Update:

In CRUD operations, 'U' is an acronym for the **update,** which **means making updates to the records present in the SQL tables.** So, we will use the UPDATE command to make changes in the data present in tables.

**Syntax:**

1. **UPDATE** Table\_Name **SET** ColumnName = Value **WHERE** CONDITION;

**Example 1:**

Write a query to update an employee's last name as 'Bose', whose employee id is 6.

**Query:**

1. mysql> **UPDATE** employee **SET** Last\_Name = "Bose" **WHERE** ID =



**Example 1:**

Write a query to update an employee's last name as 'Bose', whose employee id is 6.

**Example 2:**

Write a query to update the salary and email id of an employee as '35000' and 'shwetawagh03@gmail.com', respectively, whose employee id is 10.

**Example 1:**

Write a query to delete the employee record from the employee table whose salary is above 34000.

Write a query to fetch only those records from the employee table whose salary is above 35000.