# **Relational Algebra in DBMS: Operations with Examples**

1. **SELECT (σ)**

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it.

**Notation:** **σ c(R)** where ‘c’ is selection condition

**Example 1:**

**σ topic = "Database" (tutorials)**

**Output** – Selects tuples from Tutorials where topic = ‘Database’.

**Example-2:**

Given a relation Student (Roll, Name, Class, Fees, Team) with the following tuples:

| **Roll** | **Name** | **Department** | **Fees** | **Team** |
| --- | --- | --- | --- | --- |
| 1 | Bikash | CSE | 22000 | A |
| 2 | Josh | CSE | 34000 | A |
| 3 | Kevin | ECE | 36000 | C |
| 4 | Ben | ECE | 56000 | D |

* Now, to display all the records of student table, we will use the following command −

**σ(student)**

* Select all the student of Team A:

**σ Team = 'A' (Student)**

| **Roll** | **Name** | **Department** | **Fees** | **Team** |
| --- | --- | --- | --- | --- |
| 1 | Bikash | CSE | 22000 | A |
| 2 | Josh | CSE | 34000 | A |

## 2. Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. (pi) symbol is used to choose attributes from a relation.

**Example of Projection:**

Consider the following table

|  |  |  |
| --- | --- | --- |
| **Customer ID** | **Customer Name** | **Status** |
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

Here, the projection of Customer Name and status will give

Π CustomerName, Status (Customers)

|  |  |
| --- | --- |
| **Customer Name** | **Status** |
| Google | Active |
| Amazon | Active |
| Apple | Inactive |
| Alibaba | Active |

1. **Rename (ρ)**

Rename is a unary operation used for renaming attributes of a relation.

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

## Union operation (υ)

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

The result <- A ∪ B

For a union operation to be valid, the following condition must hold –

* must have the same number of attributes.

Example: Consider the following tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table A** | |  | **Table B** | |
| **column 1** | **column 2** |  | **column 1** | **column 2** |
| 1 | 1 |  | 1 | 1 |
| 1 | 2 |  | 1 | 3 |

A ∪ B gives following table:

|  |  |
| --- | --- |
| **Table A ∪ B** | |
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

**Set Difference (-)**

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

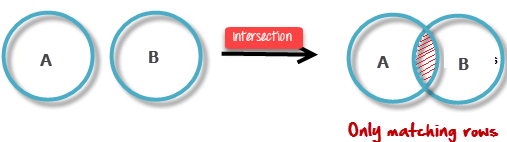
* The attribute name of A has to match with the attribute name in B.
* The two-operand relations A and B should be either compatible or Union compatible.
* It should be defined relation consisting of the tuples that are in relation A, but not in B.

**Example**: A-B

|  |  |
| --- | --- |
| **Table A – B** | |
| **column 1** | **column 2** |
| 1 | 2 |

**Intersection:** An intersection is defined by the symbol ∩

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

Visual Definition of Intersection

Example: A ∩ B

|  |  |
| --- | --- |
| **Table A ∩ B** | |
| **column 1** | **column 2** |
| 1 | 1 |

**Cartesian Product(X) in DBMS**

**Cartesian Product in DBMS** is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

**Example – Cartesian product**

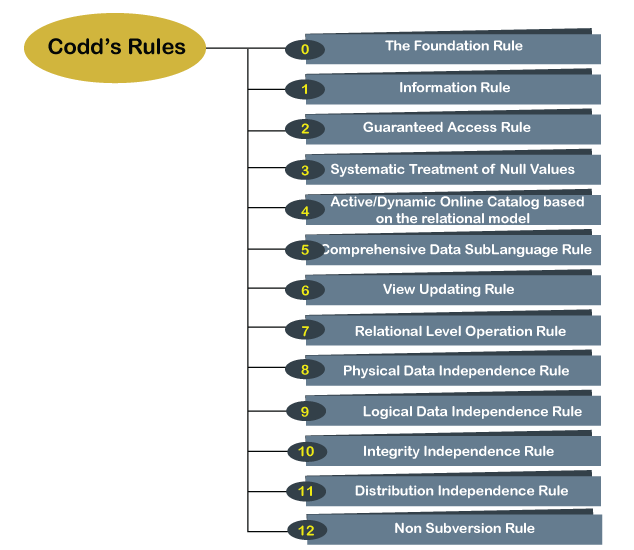
σ column 2 = ‘1’ (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

|  |  |
| --- | --- |
| **σ column 2 = ‘1’ (A X B)** | |
| **column 1** | **column 2** |
| 1 | 1 |
| 1 |  |

CODD'S RULES

**Codd's 12 rules** were developed by **Dr. Edgar F. Codd (E.F. Codd)** in **1985**, for a database to test the concept of [DBMS](https://www.javatpoint.com/dbms-tutorial) against his relational model, and if a database follows the rule, it is called a **true relational database (RDBMS).**



### **Rule 0: The Foundation Rule**

This is the foundational Rule. This rule states that any database system should have characteristics as relational, as a database and as a management system to be RDBMS.

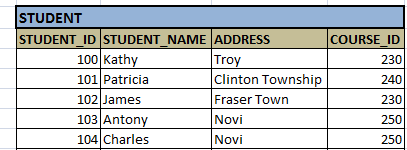
That means a database should be a relational by having the relation / mapping among the [tables](https://www.tutorialcup.com/dbms/tables.htm) in the database. They have to be related to one another by means of constraints/ relation. There should not be any independent tables hanging in the database.

RDBMS is a database i.e.; it stores the data in a well-organized form called tables. It should be able to handle large amount of information too. In short, it should meet the objectives of a database.

RDBMS is management system – that means it should be able to manage the data, relation, retrieval, update, delete, permission on the objects. It should be able handle all these administrative tasks without affecting the objectives of database. It should be performing all these tasks by using query languages.

### **Rule 1: Information Rule**

A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.



**For example:** Order of storing personal details about ‘James’ and ‘Antony’ in STUDENT table should not have any difference. There should be flexibility of storing them in any order in a row. Similarly, storing Person name first and then his address should be same as storing address and then his name. It does not make any difference on the meaning of table.

### **Rule 2: Guaranteed Access Rule**

### This rule refers to the primary key. It states that any data/column/attribute in the table should be able logically accessed by using the table in which it is stored, the primary key column of the table and the column which we want to access. When combination of these 3 is used, it should give the correct result.  Any column/ cell value should not be directly accessed without specifying the table and primary key.

**For Example:**

### Address of Kathy  STUDENT + STUDENT\_ID (Kathy) + ADDRESS is the right way of getting any cell value.

Address of Kathy  Troy should be able to access like this.

### **Rule 3: Systematic Treatment of Null Values**

This rule states about handling the NULLs in the database. As database consists of various types of data, each cell will have different datatypes. If any of the cell value is unknown, or not applicable or missing, it cannot be represented as zero or empty. It will be always represented as NULL.

This NULL should be acting irrespective of the datatype used for the cell. When used in logical or arithmetical operation, it should result the value correctly.

**For example:**

Adding NULL to numeric 5 should result NULL –

5+ unknown = unknown   5+ NULL = NULL  

5+ NULL! = 5 or 0   

It should not result in any zero or numeric value. DBMS should be strong enough to handle these NULLs according to the situation and the datatypes.

### **Rule 4: Active/Dynamic Online Catalog based on the relational model**

It represents the entire logical structure of the descriptive database that must be stored online and is known as a database dictionary. It authorizes users to access the database and implement a similar query language to access the database.

SELECT\* FROM STUDENT; # STUDENT is the table where user owns and has access.

### **Rule 5: Comprehensive Data Sublanguage Rule**

The relational database supports various languages, and if we want to access the database, the language must be the explicit, linear or well-defined syntax, character strings and supports the comprehensive: data definition, view definition, data manipulation, integrity constraints, and limit transaction management operations. If the database allows access to the data without any language, it is considered a violation of the database.

**For example:**

SQL is a structured query language which support creating tables / views/ constraints/indexes, accessing the records of tables/views (SELECT), manipulating the records by insert/delete/update, provides security by giving different level of access rights (GRANT and REVOKE) and integrity and consistency by using constraints.

### **Rule 6: View Updating Rule**

Views are the virtual tables created by using queries to show the partial view of the table. That is views are subset of table, it is only partial table with few rows and columns. This rule states that views are also be able to get updated as we do with its table.

**For example:**

Suppose we have created a view on Employee table, in which we have details of the employees who work for particular department, say ‘Testing’. Here STUDENT is the whole table and STUDENT\_TEST is the view with Testing Employees. According to this rule, we should be able to update the records in STUDENT\_VIEW.

But in real database systems, we cannot give this privilege on views. Basic intension of creating the view is to give the group of data to the user in the form of table. When lengthy queries have to be written to get some details from the database, view shortens the length of the query and gives more meaningful and shorter query. In such case, updating the view is not feasible. Although updating the view will update the table used for creating it, it is not recommended by most of the database. Hence this rule is not used in most of the database.

### **Rule 7: Relational Level Operation (High-Level Insert, Update and delete) Rule**

A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row. It also supports union, intersection and minus operation in the database system.

**For example:**

Suppose employees got 5% hike in a year. Then their salary has to be updated to reflect the new salary. Since this is the annual hike given to the employees, this increment is applicable for all the employees. Hence, the query should not be written for updating the salary one by one for thousands of employees. A single query should be strong enough to update the entire employee’s salary at a time.

### **Rule 8: Physical Data Independence Rule**

All stored data in a database or an application must be physically independent to access the database. Each data should not depend on other data or an application. If data is updated or the physical structure of the database is changed, it will not show any effect on external applications that are accessing the data from the database.

**For example:**

If the data stored in one disk is transferred to another disk, then the user viewing the data should not feel the difference or delay in access time. The user should be able to access the data as he was accessing before. Similarly, if the file name for the table is changed in the memory, it should not affect the table or the user viewing the table. This is known as physical independence and database should support this feature.

### **Rule 9: Logical Data Independence Rule**

It is similar to physical data independence. It means, if any changes occurred to the logical level (table structures), it should not affect the user's view (application). For example, suppose a table either split into two tables, or two table joins to create a single table, these changes should not be impacted on the user view application.

**For example:**

If we split the EMPLOYEE table according to his department into multiple employee tables, the user viewing the employee table should not feel that these records are coming from different tables. These split tables should be able to get joined and show the result. In our example we can use UNION and display the results to the user.

### **Rule 10: Integrity Independence Rule**

A database must maintain integrity independence when inserting data into table's cells using the SQL query language. All entered values should not be changed or rely on any external factor or application to maintain integrity. It is also helpful in making the database-independent for each front-end application.

**For example:**

Suppose we want to insert an employee for department 50 using an application. But department 50 does not exists in the system. In such case, the application should not perform the task of fetching if department 50 exists, if not, insert the department and then inserting the employee. It should all handled by the database.

### **Rule 11: Distribution Independence Rule**

The distribution independence rule represents a database that must work properly, even if it is stored in different locations and used by different end-users. Suppose a user accesses the database through an application; in that case, they should not be aware that another user uses particular data, and the data they always get is only located on one site. The end users can access the database, and these access data should be independent for every user to perform the SQL queries.

The database can be located at the user server or at any other network. The end user should not be able to know about the database servers. He should be able to get the records as if he is pulling the records locally. Even if the database is located in different servers, the accessibility time should be comparatively less.

### **Rule 12: Non-Subversion Rule**

The non-subversion rule defines RDBMS as a [SQL](https://www.javatpoint.com/sql-tutorial) language to store and manipulate the data in the database. If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data.

When a query is fired in the database, it will be converted into low level language so that it can be understood by the underlying systems to retrieve the data. In such case, when accessing or manipulating the records at low level language, there should not be any loopholes that alter the integrity of the database. In other words, even though the query written does not change the integrity of the tables, the converted low-level language should be same as the query written. It should not be converted into some other low-level language which changes the data integrity in the database or performs some unwanted actions in the database.

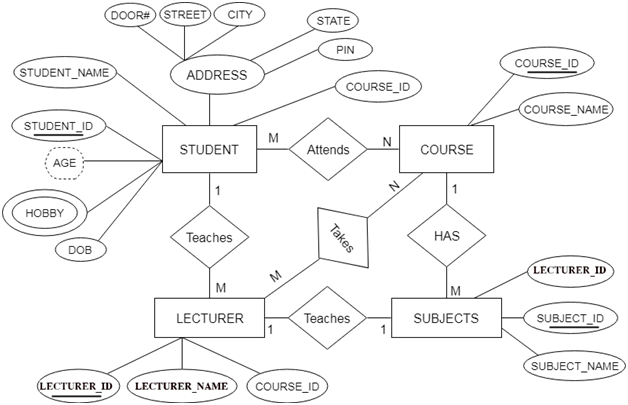
**For example:**

Update Student’s address query should always be converted into low level language which updates the address record in the student file in the memory. It should not be updating any other record in the file nor inserting some malicious record into the file/memory.

**Draw E-R Diagram and Convert entities and relationships to relation table for BANK and College. (Develop your own College database and Bank Database)**

Entity-Relationship diagram shows relationships between entities and their attributes. It is used to analyze to structure of the Database. An ER model provides a means of communication.

* **The ER diagram of COLLEGE Management System is given below:**



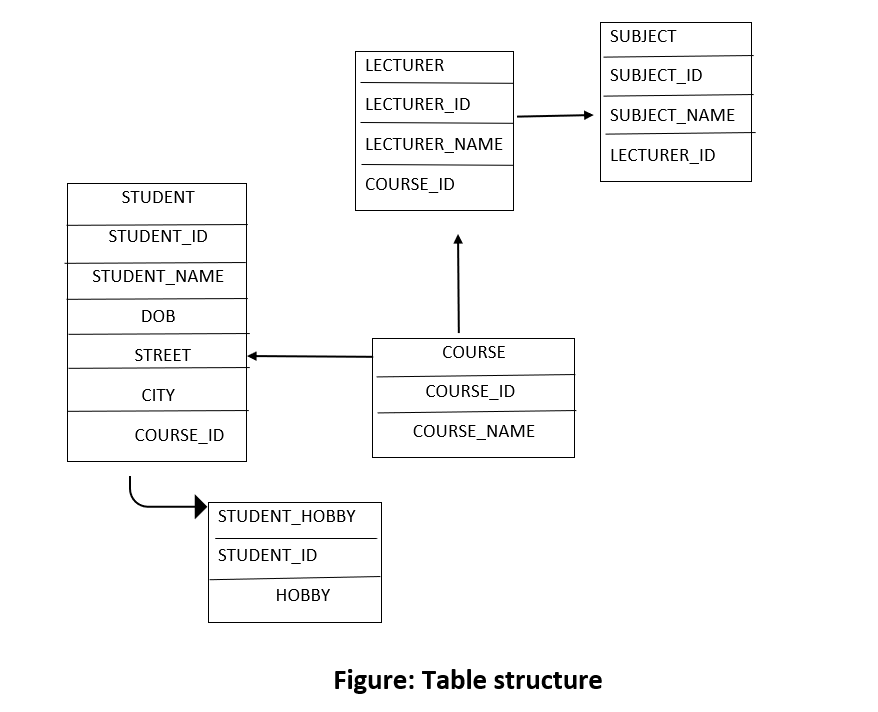
There are some points for converting the ER diagram to the table:

* **Entity type becomes a table:** In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.
* **All single-valued attribute becomes a column for the table:** In the STUDENT entity, STUDENT\_NAME and STUDENT\_ID form the column of STUDENT table. Similarly, COURSE\_NAME and COURSE\_ID form the column of COURSE table and so on.
* **A key attribute of the entity type represented by the primary key:** In the given ER diagram, COURSE\_ID, STUDENT\_ID, SUBJECT\_ID, and LECTURE\_ID are the key attribute of the entity.
* **The multivalued attribute is represented by a separate table:**

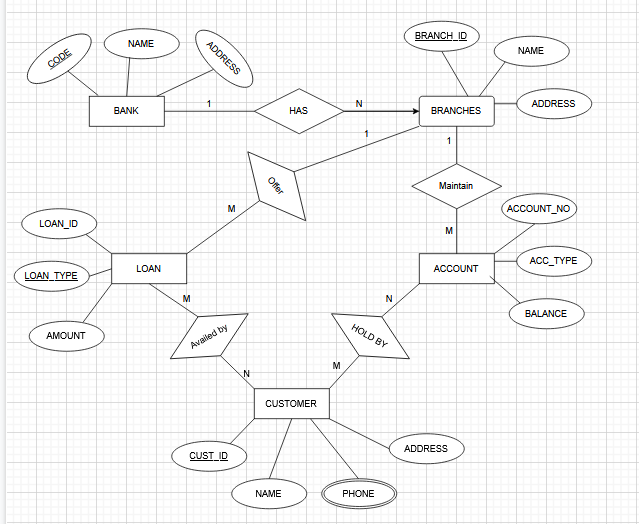
In the STUDENT table, HOBBY is a multivalued attribute. So, it is not possible to represent multiple values in a single column of STUDENT table. Hence, we create a table STUDENT\_HOBBY with column name STUDENT\_ID and HOBBY.

* **Derived attributes are not considered in the table.**

In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.



**The ER diagram of BANKING Management System is given below:**



**Entities** and their **Attributes** are:

* **Bank Entity:** Attributes of Bank Entity NAME, CODE, ADDRESS  
   CODE is Primary Key for Bank Entity.
* **Customer Entity:** Attributes of Customer Entity are CUST\_ID, NAME, PHONE, ADDRESS

CUST\_ID is Primary Key for Customer Entity.

* **Branch Entity:** Attributes of Branch Entity BRANCH\_ID, NAME, ADDRESS

BRANCH\_ID is Primary Key for Branch Entity.

* **Account Entity:** Attributes of Account Entity are ACCOUNT\_NO, ACC\_TYPE, BALANCE.

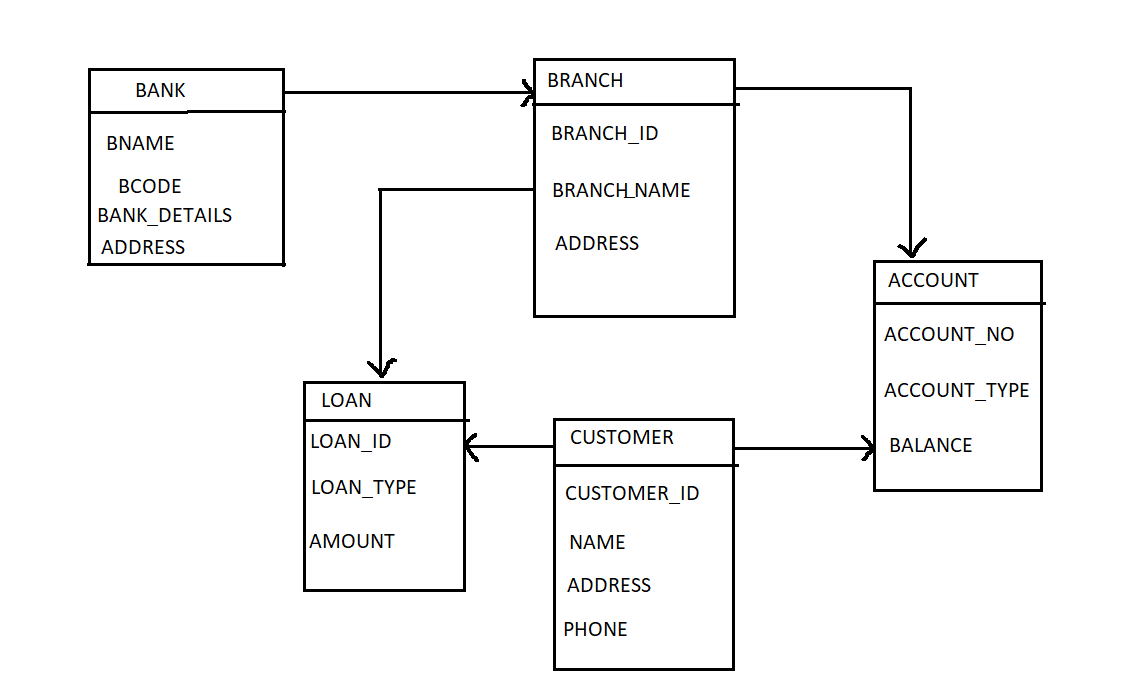
ACCOUNT\_NO is Primary Key for Account Entity.

* **Loan Entity:** Attributes of Loan Entity are LOAN\_ID, LOAN\_TYPE, AMOUNT LOAN\_ID is Primary Key for Loan Entity.

[**Relationships**](https://www.geeksforgeeks.org/attributes-to-relationships-in-er-model/) **are:**

* **Bank has Branches => 1: N**  
  One Bank can have many Branches but one Branch cannot belong to many Banks, so the relationship between Bank and Branch is one to many relationships.
* **Branch maintain Accounts => 1: M**  
  One Branch can have many Accounts but one Account cannot belong to many Branches, so the relationship between Branch and Account is one to many relationships.
* **Branch offer Loans => 1: M**  
  One Branch can have many Loans but one Loan cannot belong to many Branches, so the relationship between Branch and Loan is one to many relationships.
* **Account held by Customers => N:M**  
  One Customer can have more than one Accounts and also One Account can be held by one or more Customers, so the relationship between Account and Customers is many to many relationships.
* **Loan availed by Customer => M: N**  
  One Customer can have more than one Loans and also One Loan can be availed by one or more Customers, so the relationship between Loan and Customers is many to many relationship

**The Relational table of BANKING Management System is:**



SQL FEATURES

SQL COMMANDS

QUERY/SUBQUERY

SET OPERATIONS

RELATION BANAUNE

# **SQL Commands**

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

## Types of SQL Commands

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



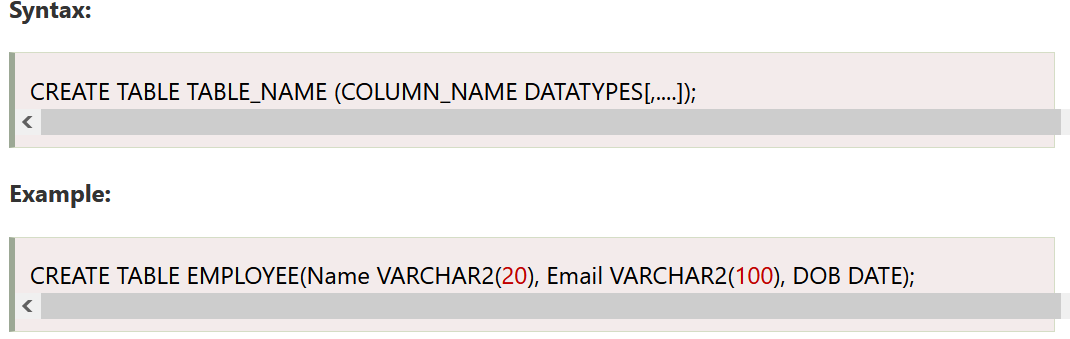
### . Data Definition Language (DDL)

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

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

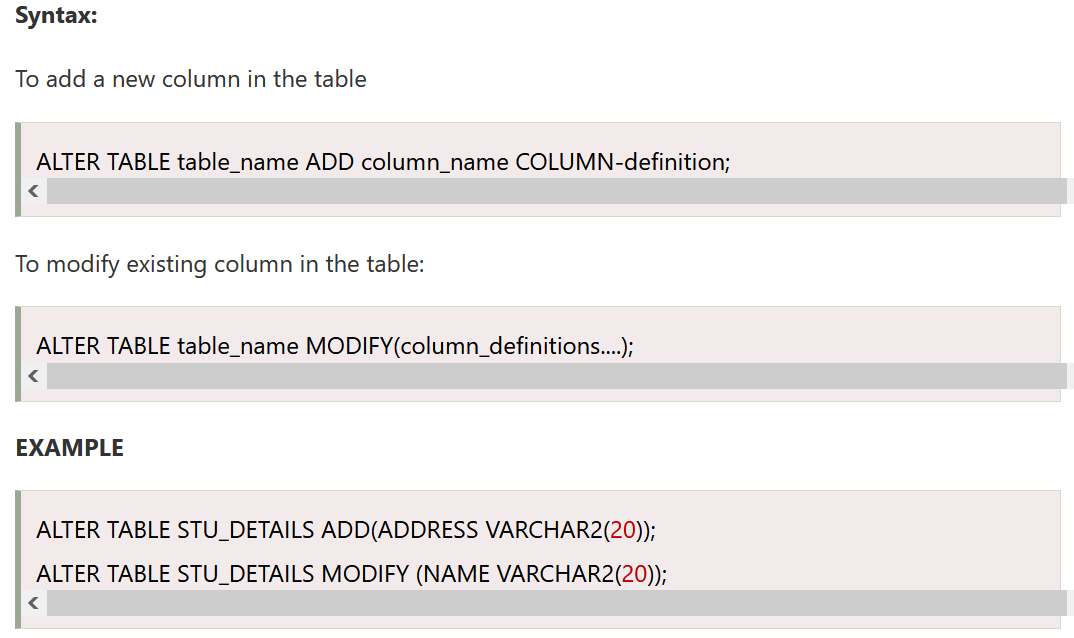
**a. CREATE** It is used to create a new table in the database.



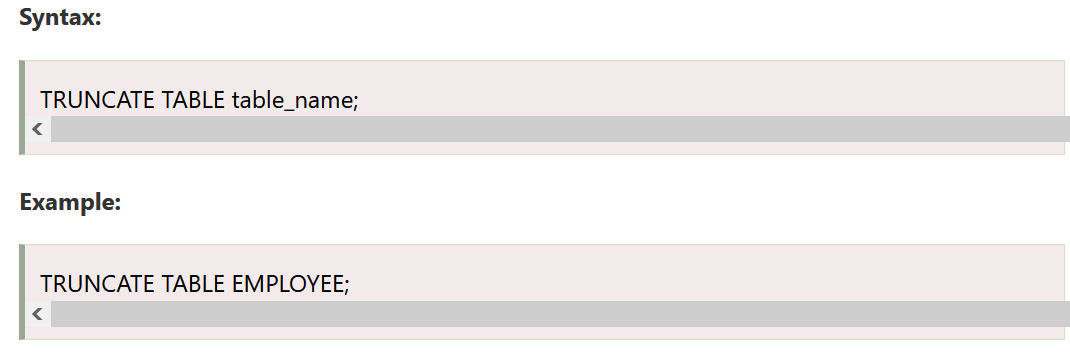
**b. DROP:** It is used to delete both the structure and record stored in the table.



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



**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.



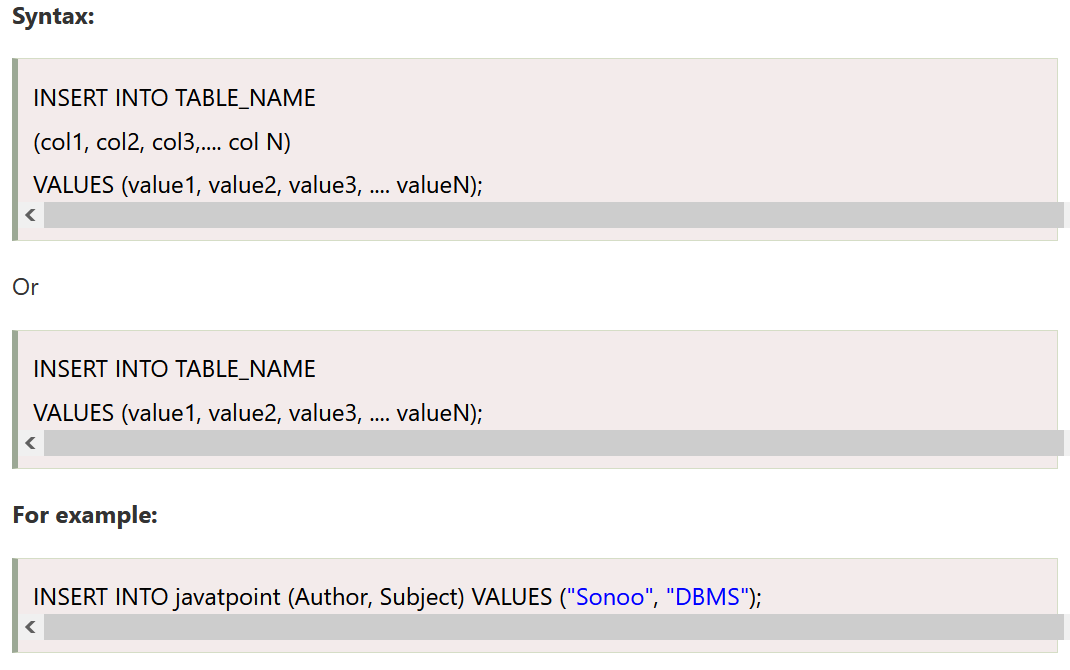
### **2. Data Manipulation Language**

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

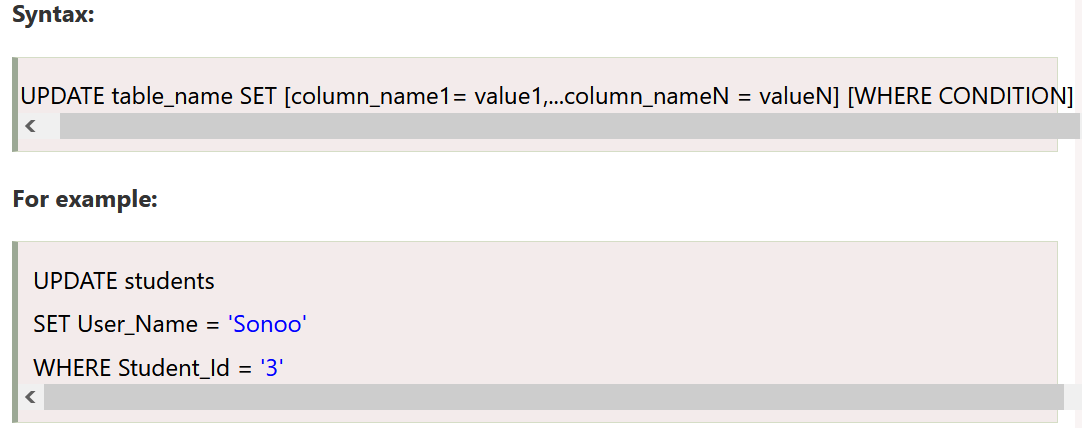
Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

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



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



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

### 

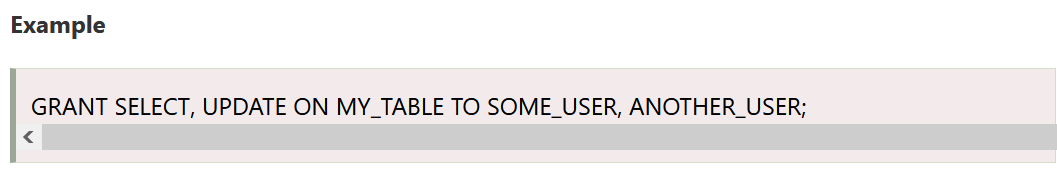
### **3. Data Control Language**

DCL commands are used to grant and take back authority from any database user.

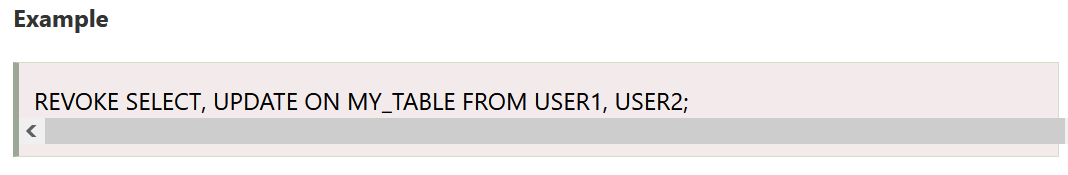
Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.



**b. Revoke:** It is used to take back permissions from the user.

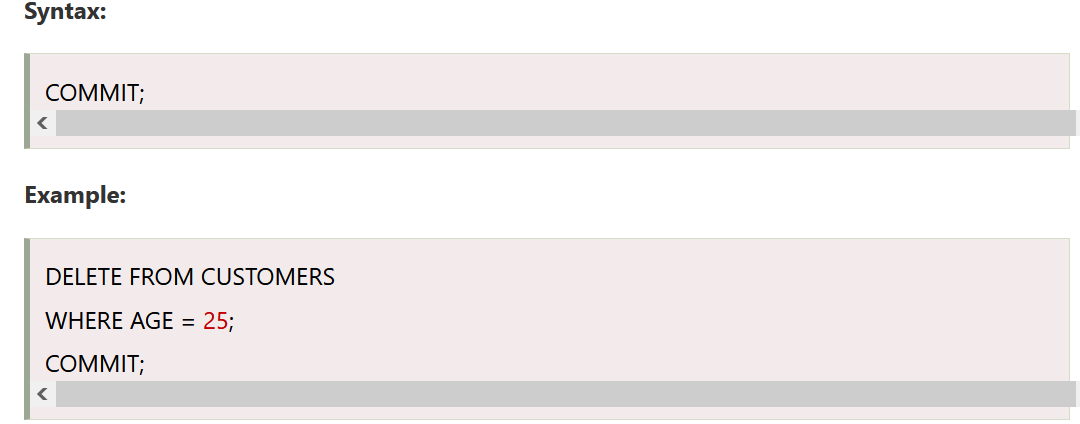


### **4. Transaction Control Language**

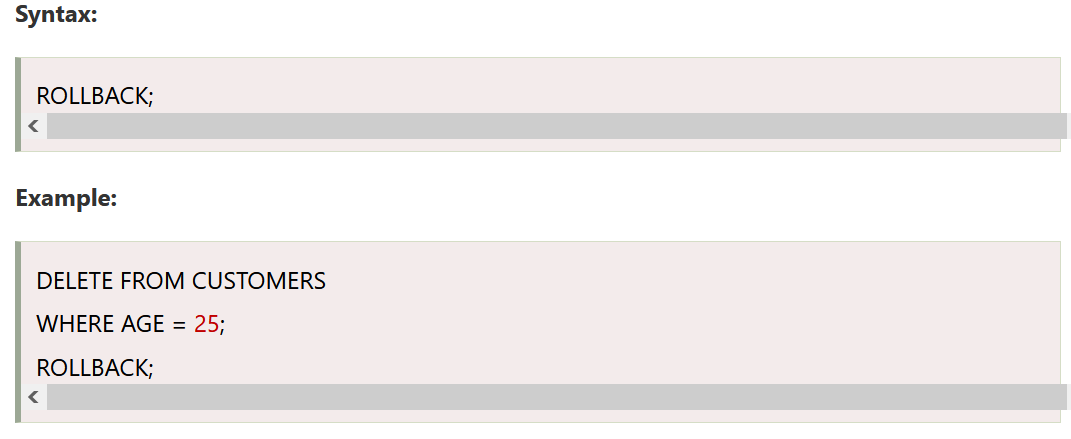
TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only. These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them. Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.



**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.



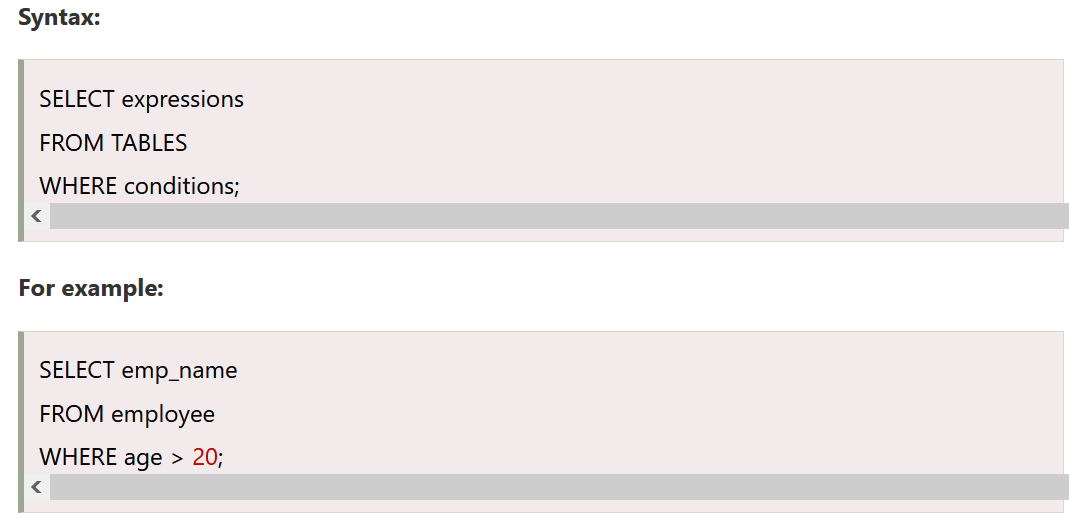
**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

### 

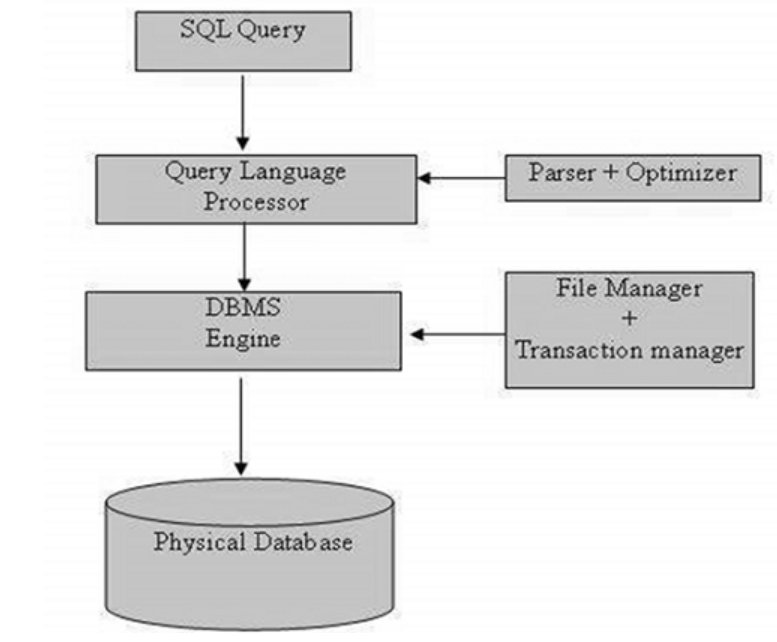
5. Data Query Language

DQL is used to fetch the data from the database. It uses only one command:

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



**SQL Architecture**



Following is the process which a SQL Query goes through.

1. The SQL query is fired at the application end. From there it is converted into TDS (Tabular Data Stream) Packets using ODBC (**Open Database Connectivity**) or OLEDB (**Object Linking and Embedding, Database**) or SNAC (**SQL Native Client**) or SNI (**SQL Server Network Interface**).

These are data access protocols which are used to access all types of data and grind them into TDS packets to be encapsulated within network packets to be travelled through network protocols from one endpoint to other.

1. Once TDS packets reach the server endpoint, the SNI decapsulates those packets into SQL commands.
2. The commands are passed through Query Parser or Command Parser and it checks the query for any Syntactical (Syntax) or Semantical (Logical) errors and if any error occurs it returns the error to the N/W protocol layer.
3. If it passes through the command parser, the next step is to generate a query plan. The query optimizer selects a cost-effective plan provided to it using Query Tree (It uses certain algorithms to generate different query plans and presents it to the Query Optimizer.
4. Query optimizer then selects a cost-effective plan and presents the query-to-Query Executor. To execute a plan Query Executor needs data as well. So, it passes the request to the Access Methods which is a collection of codes which provides an interface to retrieve data and present it to Query Executor after valid conversion using OLEDB. It itself does not do this work, rather it asks the buffer manager for the data.
5. If the data is there in the Buffer, Access methods fetch those data pages and pass them to the query executor to execute the query. If the query plan is already in the Plan cache the executor uses that plan.
6. The work of access method is to check if the query is select or non-select (DML). If the query is non-select the access methods contact Transaction Manager. Transaction Manager has two components,
   1. *Log Manager*: Logs the events that will be modifying the data into Log Buffer in Buffer Pool.
   2. *Lock Manager:* Assigns a lock on that transaction to provide data consistency and isolation.
7. Transaction Manager generates a Log Sequential Number (LSN) for that transaction and records the events that will be modifying the data in Log Buffer and the Transaction will make changes in the buffer cache using only locking mechanism to be isolated from any other transaction who wants to modify the same data. Changes are not directly made to the data pages on the disk.

These modified pages reside in the buffer cache and are known as Dirty Pages, as they are not written to the disk as of yet. Now there is a checkpoint process that is an automatic recurring event in SQL server and runs in the background.

When checkpoint happens, it flushes all the dirty pages (Modified Pages) to the disk and marks the pages as clean in the buffer cache, but does not deallocate those pages from cache. Before it does that, the Log records are pushed into a Virtual Log File (data page of log file) with the LSN in Transaction Log from Log Buffer. This process of writing to the Log File before writing to the disk is known as Write Ahead Logging.

Lazy Writer, again a background process, also flushes the pages out of the buffer pool to the disk. When the SQL server comes under the memory pressure, lazy writer deallocates the pages which are residing there unused, also the clean pages from the memory and writes the dirty pages to the disk to be able to make some memory space for other operations.