

# Agenda

- Transaction
- ER Diagrams

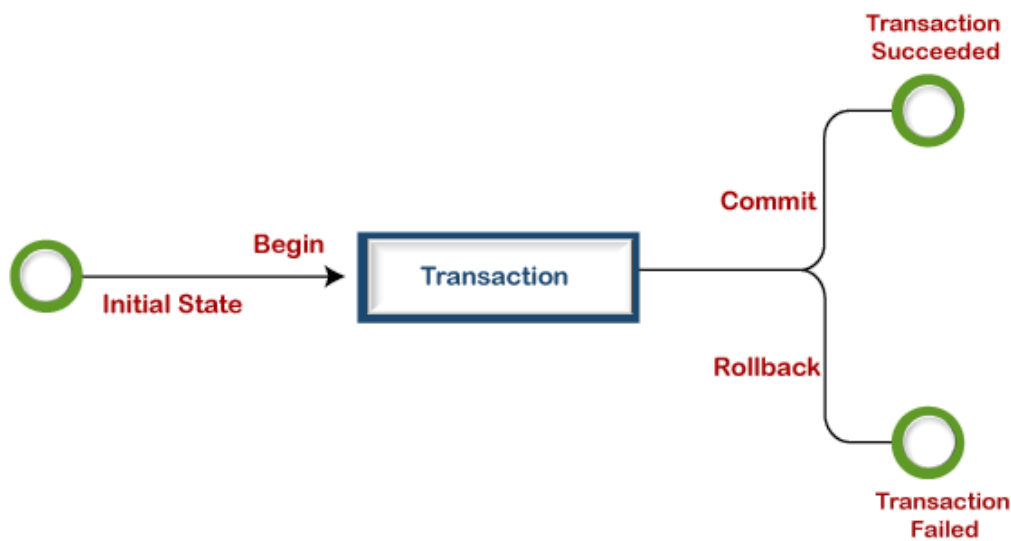
## Transactions

A transaction in SQL Server is a **sequential group of statements or queries** to perform single or multiple tasks in a database. Each transaction may have single read, write, update, or delete operations or a combination of all these operations. Each transaction must happen two things in SQL Server:

- Either all modification is successful when the transaction is committed.
- Or, all modifications are undone when the transaction is rollback.

A transaction cannot be successful until all of the operations in the set are completed. It means that if any argument fails, the transaction operation will fail. Each transaction begins with the first executable SQL statement and ends when it finds a commit or rollback, either explicitly or implicitly. It uses the **COMMIT** or **ROLLBACK** statements explicitly, as well as implicitly when a DDL statement is used.

The below pictorial representation explains the transaction process:



The following example will explain the concept of a transaction:

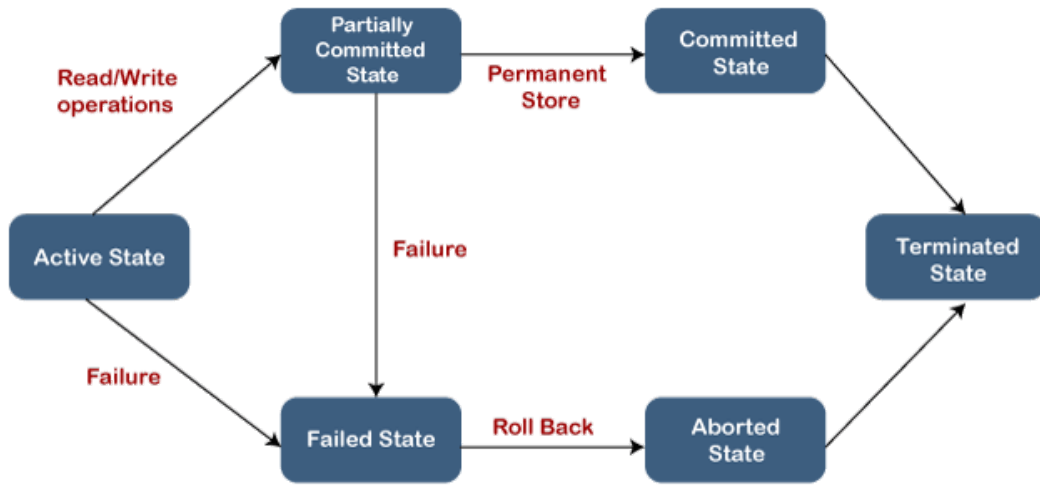
This example will use the banking database system to explain the concept of a transaction. Suppose a bank customer wants to withdraw money from their account by using ATM mode. The ATM can achieve this operation in the three steps:

1. The **first step** is to check the availability of the requested amount in the account.
2. The **second step** deducts the amount from the account if the amount is available and then updates the account balance.
3. The **third step** is to write the money withdrawing operation in the log file. This step writes about the transaction is either successful or failed. If successful, write the data modification in the database. Otherwise, the transaction will be rolled back into its previous state.

The basic principle behind transactions is that if one of the statements returns an error, the entire set of changes is rolled back to ensure data integrity. And if the transactions become successful, all changes will be permanent on the database. Hence, if there is a power outage or other issues while withdrawing money from an ATM, transactions guarantee that our balance remains consistent. A transaction statement best performs these operations because the transaction's four key properties make all operations more accurate and consistent. The transaction's four properties are referred to as ACID.

## Transaction State

It indicates how transactions go during their lifetime. It describes the current state of the transaction as well as how the transaction will be processed in the future. These states define the rules that determine whether a transaction commits or aborts.



Transaction States

Let us describe each transaction states in SQL Server:

**Active State:** The transaction is in an active state while the transaction's instructions are being executed. It changes to the "**partially committed state**" if all "read and write" operations are completed without errors. If any instruction fails, it changes to the "failed state."

**Partially Committed:** When all the read and write operations are completed, the change is made to the main memory or local buffer. The state would change to "**committed state**" if the changes are made permanent on the database. Otherwise, it goes to the "failed state".

**Failed State:** A transaction goes to the failed state when any transaction's instruction fails or a permanent modification on the database fails.

**Aborted State:** The transaction moves from a "**failed state**" to an "**aborted state**" when any kind of failure occurs. The changes are removed or rolled back because these changes are only made to the local buffer or main memory in previous states.

**Committed State:** A transaction is complete and goes into this state when the changes are made permanent on the database and terminated the "**terminated state**".

**Terminated State:** If there is no rollback and the transaction is in the "**committed state**," the system is consistent and ready for a new transaction while the old one is terminated.

## MySQL Transaction Statement

MySQL control transactions with the help of the following statement:

- MySQL provides a START TRANSACTION statement to begin the transaction. It also offers a "BEGIN" and "BEGIN WORK" as an alias of the START TRANSACTION.
- We will use a COMMIT statement to commit the current transaction. It allows the database to make changes permanently.
- We will use a ROLLBACK statement to roll back the current transaction. It allows the database to cancel all changes and goes into their previous state.
- We will use a SET auto-commit statement to disable/enable the auto-commit mode for the current transaction. By default, the COMMIT statement executed automatically. So if we do not want to commit changes automatically, use the below statement:

```

SET autocommit = 0;
OR,
SET autocommit = OFF;
  
```

Again, use the below statement to enable auto-commit mode:

```

SET autocommit = 1;
OR,
SET autocommit = ON;
  
```

## MySQL Transaction Example

Suppose we have two tables named "**employees**" and "**Orders**" that contains the following data:

**Table: employees**

emp_id	emp_name	emp_age	city	income
101	Peter	32	Newyork	200000
102	Mark	32	California	300000
103	Donald	40	Arizona	1000000
104	Obama	35	Florida	5000000
105	Linklon	32	Georgia	250000
106	Kane	45	Alaska	450000
107	Adam	35	California	5000000
108	Macculam	40	Florida	350000
109	Brayan	32	Alaska	400000
110	Stephen	40	Arizona	600000

Table: orders

Order_ID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
3	Desktop	2135	2020-01-05
4	Mobile	3432	2020-02-22
5	Anti-Virus	5648	2020-03-10

## COMMIT Example

If we want to use a transaction, it is required to break the [SQL](#) statements into logical portions. After that, we can define whether the data should be committed or rollback.

The following steps illustrate to create a transaction:

1. Begin the transaction using the START TRANSACTION statement.
2. Then, select maximum income among the employee.
3. Add a new record to the employee table.
4. Add a new record into the order table.
5. Use the COMMIT statement to complete the transaction.

Below are the commands that perform the above operations:

```
-- 1. Start a new transaction

START TRANSACTION;

-- 2. Get the highest income

SELECT @income:= MAX(income) FROM employees;

-- 3. Insert a new record into the employee table

INSERT INTO employees(emp_id, emp_name, emp_age, city, income)
```



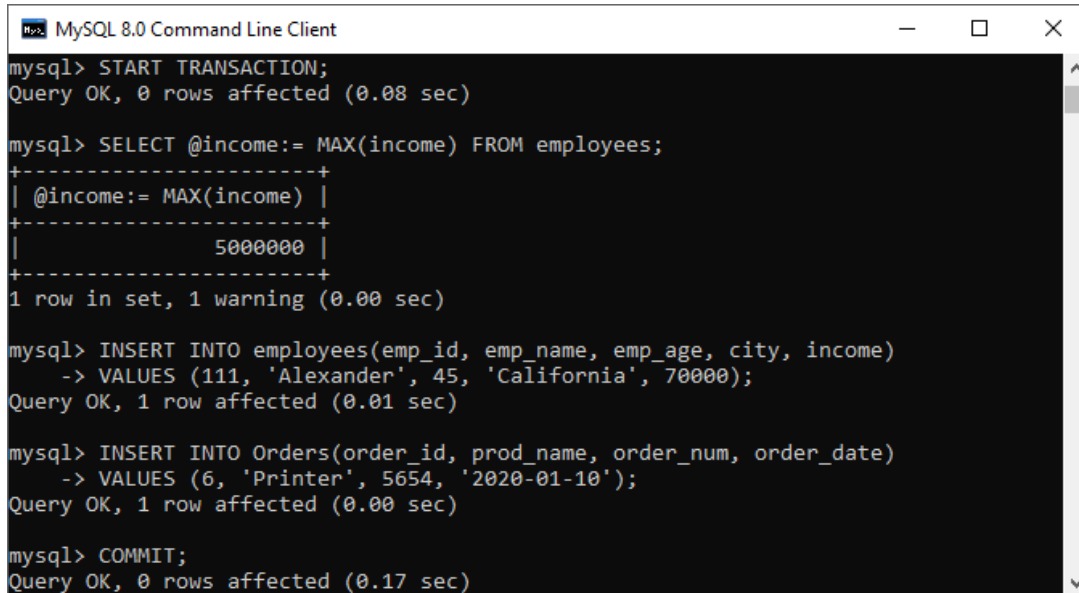
```
VALUES (111, 'Alexander', 45, 'California', 70000);

-- 4. Insert a new record into the order table

INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');

-- 5. Commit changes
COMMIT;
```

The below image explains it more clearly:



```
MySQL 8.0 Command Line Client
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT @income:= MAX(income) FROM employees;
+-----+
| @income:= MAX(income) |
+-----+
|          5000000      |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> INSERT INTO employees(emp_id, emp_name, emp_age, city, income)
-> VALUES (111, 'Alexander', 45, 'California', 70000);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Orders(order_id, prod_name, order_num, order_date)
-> VALUES (6, 'Printer', 5654, '2020-01-10');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.17 sec)
```

## ROLLBACK Example

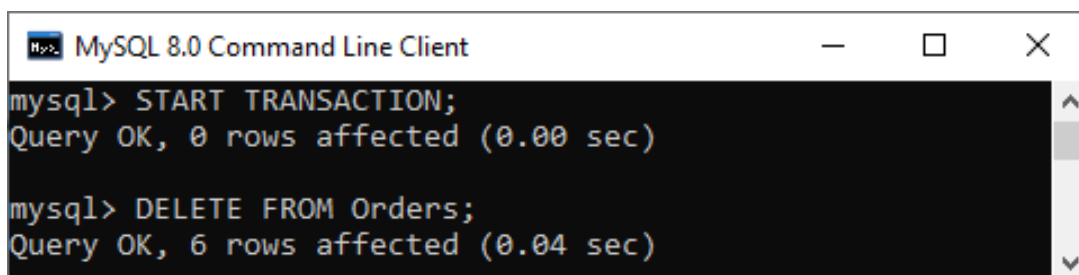
We can understand the rollback transaction with the help of the following illustration. First, open the MySQL command prompt and log into the database server using the password. Next, we have to select a database.

Suppose our database contains the "Orders" table. Now, the following are the scripts that perform the rollback operations:

```
-- 1. Start a new transaction
START TRANSACTION;

-- 2. Delete data from the order table
DELETE FROM Orders;
```

After the execution of the above statement, we will get the output as below that shows all the records from the table Orders were successfully deleted.



```
MySQL 8.0 Command Line Client
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM Orders;
Query OK, 6 rows affected (0.04 sec)
```

Now, we need to open a separate session of MySQL database server and execute the below statement to verify the data in Orders table:

```
SELECT * FROM Orders;
```

It will give the output as below

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM Orders;
+-----+-----+-----+-----+
| order_id | prod_name | order_num | order_date |
+-----+-----+-----+-----+
| 1 | Laptop | 5544 | 2020-02-01 |
| 2 | Mouse | 3322 | 2020-02-11 |
| 3 | Desktop | 2135 | 2020-01-05 |
| 4 | Mobile | 3432 | 2020-02-22 |
| 5 | Antivirus | 5648 | 2020-03-10 |
| 6 | Printer | 5654 | 2020-01-10 |
+-----+-----+-----+-----+
```

Although we have made changes in the first session, we still can see the records are available in the table. It is because the changes are not permanent until we have not executed the **COMMIT** or **ROLLBACK** statement in the first session.

Therefore if we want to make changes permanent, use the **COMMIT** statement. Otherwise, execute the **ROLLBACK** statement to roll back the change in the first session.

```
-- 3. Rollback changes
ROLLBACK;
```

```
-- 4. Verify the records in the first session
SELECT * FROM Orders;
```

After the successful execution, it will produce the following result where we can see that the change has been rolled back.

```
MySQL 8.0 Command Line Client
mysql> ROLLBACK;
Query OK, 0 rows affected (1.68 sec)

mysql> SELECT * FROM Orders;
+-----+-----+-----+-----+
| order_id | prod_name | order_num | order_date |
+-----+-----+-----+-----+
| 1 | Laptop | 5544 | 2020-02-01 |
| 2 | Mouse | 3322 | 2020-02-11 |
| 3 | Desktop | 2135 | 2020-01-05 |
| 4 | Mobile | 3432 | 2020-02-22 |
| 5 | Antivirus | 5648 | 2020-03-10 |
| 6 | Printer | 5654 | 2020-01-10 |
+-----+-----+-----+-----+
```

**Statements that cannot be a rollback in using MySQL Transaction.**

MySQL Transaction cannot be able to roll back all statements. For example, these statements include DDL (Data Definition Language) commands such as **CREATE**, **ALTER**, or **DROP** database as well as **CREATE**, **UPDATE**, or **DROP** tables or stored routines. We have to make sure that when we design our transaction, these statements do not include.

**SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT**

The **SAVEPOINT** statement creates a special mark with the name of the **identifier** inside a transaction. It allows all statements that are executed after the savepoint would be rolled back. So that the transaction restores to the previous state it was in at the point of the savepoint. If we have set multiple savepoints in the current transaction with the same name, the newly savepoint is responsible for rollback.

The **ROLLBACK TO SAVEPOINT** statement allows us to roll back all transactions to the given savepoint was established without aborting the transaction.

The **RELEASE SAVEPOINT** statement destroys the named savepoint from the current transaction without undoing the effects of queries executed after the savepoint was established. After these statements, no rollback command occurs. If the savepoint does not exist in the transaction, it gives an error.

The following are the **syntax** of the above statements in MySQL Transaction:

```
SAVEPOINT savepoint_name
ROLLBACK TO [SAVEPOINT] savepoint_name
RELEASE SAVEPOINT savepoint_name
```

## Example

Let us understand how to use these statements through the example. In the below example, we are going to use SAVEPOINT and ROLLBACK TO SAVEPOINT statements that explain how a savepoint determines which records of the current transaction can be rolled back.

```
START TRANSACTION;

SELECT * FROM Orders;

INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');

SAVEPOINT my_savepoint;

INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');

ROLLBACK TO SAVEPOINT my_savepoint;

INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (8, 'Speaker', 6065, '2020-02-18');

COMMIT;
```

In the above,

- We have to first begin the transaction and then show the records available in the Orders table.
- Next, we have inserted one record into the table and then creates a savepoint mark.
- Again, we have inserted one record into the table and then use a ROLLBACK TO SAVEPOINT statement to remove changes where the savepoint established.
- Again, we have inserted one record into the table.
- Finally, execute the COMMIT statement to make changes permanently.

The output below explains the above steps in a sequential order that helps to understand it very easily.

```
MySQL 8.0 Command Line Client
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Orders;
+-----+-----+-----+-----+
| order_id | prod_name | order_num | order_date |
+-----+-----+-----+-----+
| 1 | Laptop | 5544 | 2020-02-01 |
| 2 | Mouse | 3322 | 2020-02-11 |
| 3 | Desktop | 2135 | 2020-01-05 |
| 4 | Mobile | 3432 | 2020-02-22 |
| 5 | Antivirus | 5648 | 2020-03-10 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> INSERT INTO Orders(order_id, prod_name, order_num, order_date)
-> VALUES (6, 'Printer', 5654, '2020-01-10');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT my_savepoint;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Orders(order_id, prod_name, order_num, order_date)
-> VALUES (7, 'Ink', 5894, '2020-03-10');
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT my_savepoint;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Orders(order_id, prod_name, order_num, order_date)
-> VALUES (8, 'Speaker', 6065, '2020-02-18');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.21 sec)
```

Now, we will use a SELECT statement to verify the above operation. In the output, we can see that the order\_id=6 and order\_id=8 is added successfully but order\_id=7 is not inserted into the table. It rolls back the values entered after the savepoint was established:

```
MySQL 8.0 Command Line Client
mysql> SELECT * FROM Orders;
+-----+-----+-----+-----+
| order_id | prod_name | order_num | order_date |
+-----+-----+-----+-----+
| 1 | Laptop | 5544 | 2020-02-01 |
| 2 | Mouse | 3322 | 2020-02-11 |
| 3 | Desktop | 2135 | 2020-01-05 |
| 4 | Mobile | 3432 | 2020-02-22 |
| 5 | Antivirus | 5648 | 2020-03-10 |
| 6 | Printer | 5654 | 2020-01-10 |
| 8 | Speaker | 6065 | 2020-02-18 |
+-----+-----+-----+-----+
```

Now we are going to take another example RELEASE SAVEPOINT that establishes the my\_savepoint and then removes a savepoint.

```
START TRANSACTION;

INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');

SAVEPOINT my_savepoint;

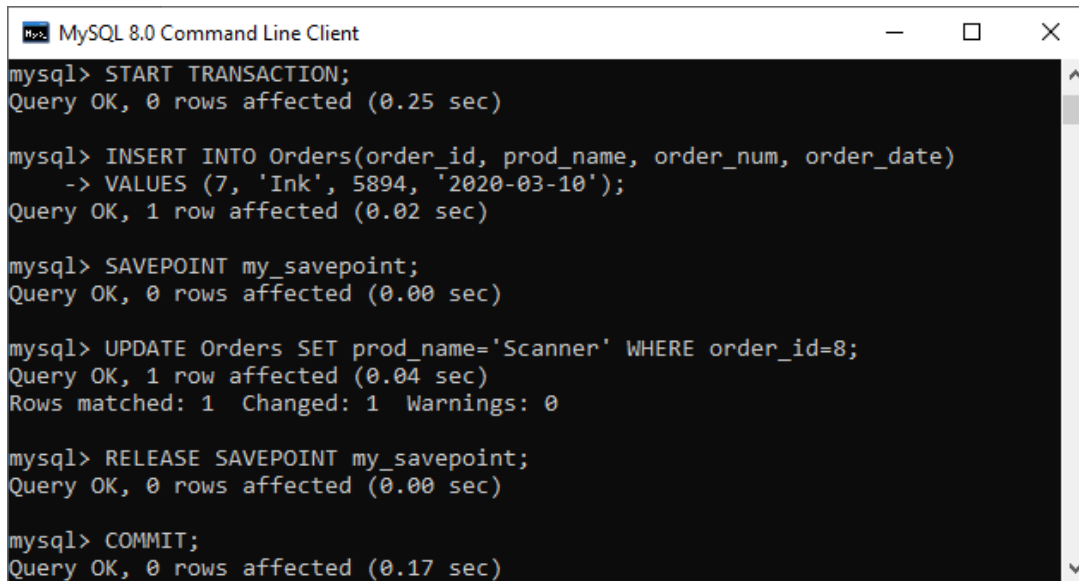
UPDATE Orders SET prod_name='Scanner' WHERE order_id=8;

RELEASE SAVEPOINT my_savepoint;
```



```
COMMIT;
```

In the output, we can see that all statements in the transaction executed successfully. Here, both INSERT and UPDATE statements modify the table COMMIT.

A screenshot of the MySQL 8.0 Command Line Client window. The window has a title bar with the MySQL logo and the text "MySQL 8.0 Command Line Client". The main area is a black terminal with white text showing the execution of a transaction. The commands and their outputs are: 1. "mysql> START TRANSACTION;" followed by "Query OK, 0 rows affected (0.25 sec)". 2. "mysql> INSERT INTO Orders(order\_id, prod\_name, order\_num, order\_date) -> VALUES (7, 'Ink', 5894, '2020-03-10');" followed by "Query OK, 1 row affected (0.02 sec)". 3. "mysql> SAVEPOINT my\_savepoint;" followed by "Query OK, 0 rows affected (0.00 sec)". 4. "mysql> UPDATE Orders SET prod\_name='Scanner' WHERE order\_id=8;" followed by "Query OK, 1 row affected (0.04 sec)" and "Rows matched: 1 Changed: 1 Warnings: 0". 5. "mysql> RELEASE SAVEPOINT my\_savepoint;" followed by "Query OK, 0 rows affected (0.00 sec)". 6. "mysql> COMMIT;" followed by "Query OK, 0 rows affected (0.17 sec)".

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.25 sec)

mysql> INSERT INTO Orders(order_id, prod_name, order_num, order_date)
-> VALUES (7, 'Ink', 5894, '2020-03-10');
Query OK, 1 row affected (0.02 sec)

mysql> SAVEPOINT my_savepoint;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Orders SET prod_name='Scanner' WHERE order_id=8;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> RELEASE SAVEPOINT my_savepoint;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.17 sec)
```

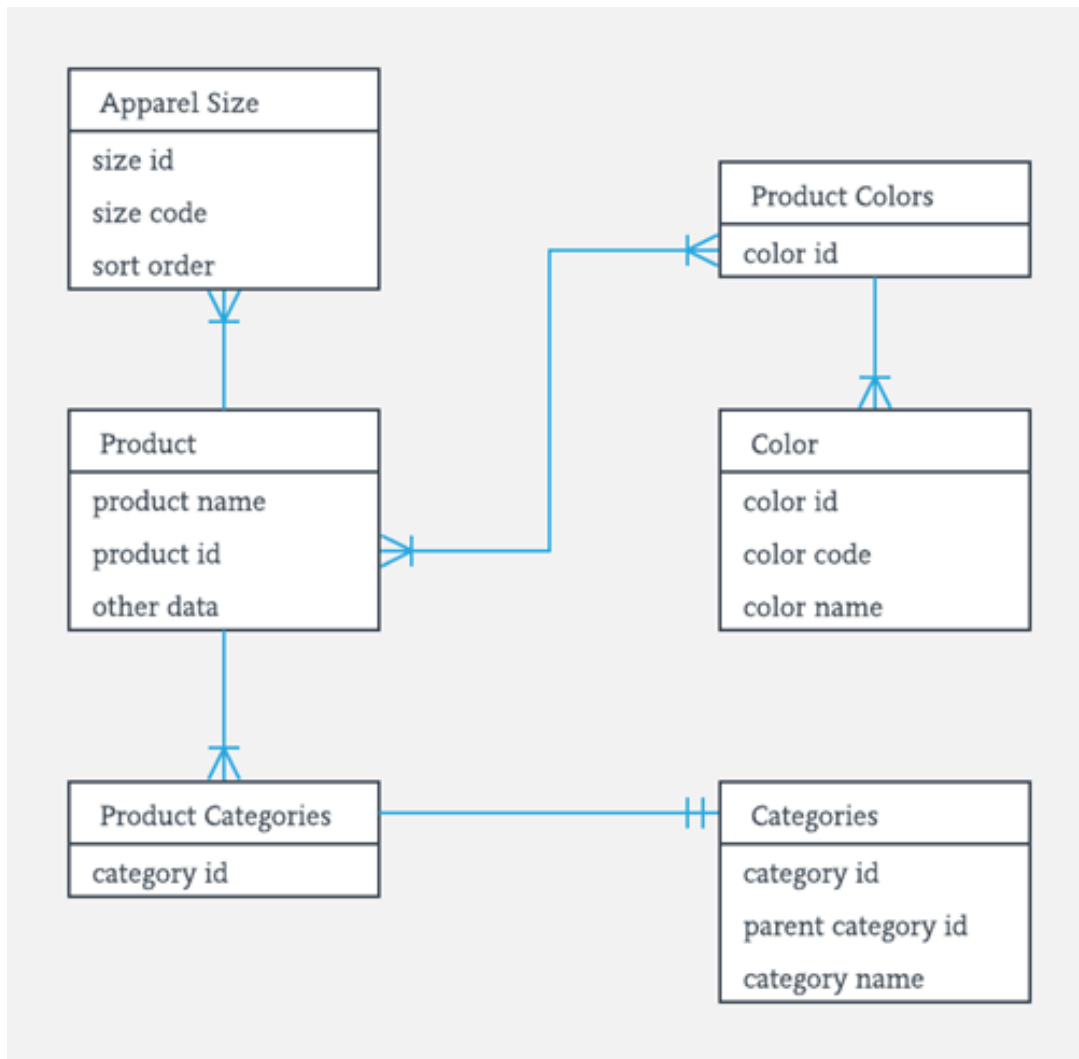
## Entity Relationship (ER) Diagrams

**ER Diagram** stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attribute and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represe relationships.

At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make th model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.





## What is ER Model?

**ER Model** stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze da requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them. Creating an E Model in DBMS is considered as a best practice before implementing your database.

## Facts about ER Diagram Model

Now in this ERD Diagram Tutorial, let's check out some interesting facts about ER Diagram Model:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

## ER Diagrams Symbols & Notations

**Entity Relationship Diagram Symbols & Notations** mainly contains three basic symbols which are rectangle, oval and diamond to represe relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations.

Following are the main components and its symbols in ER Diagrams:

- **Rectangles:** This Entity Relationship Diagram symbol represents entity types
- **Ellipses :** Symbol represent attributes
- **Diamonds:** This symbol represents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types

- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes

#### ER Diagram Symbols



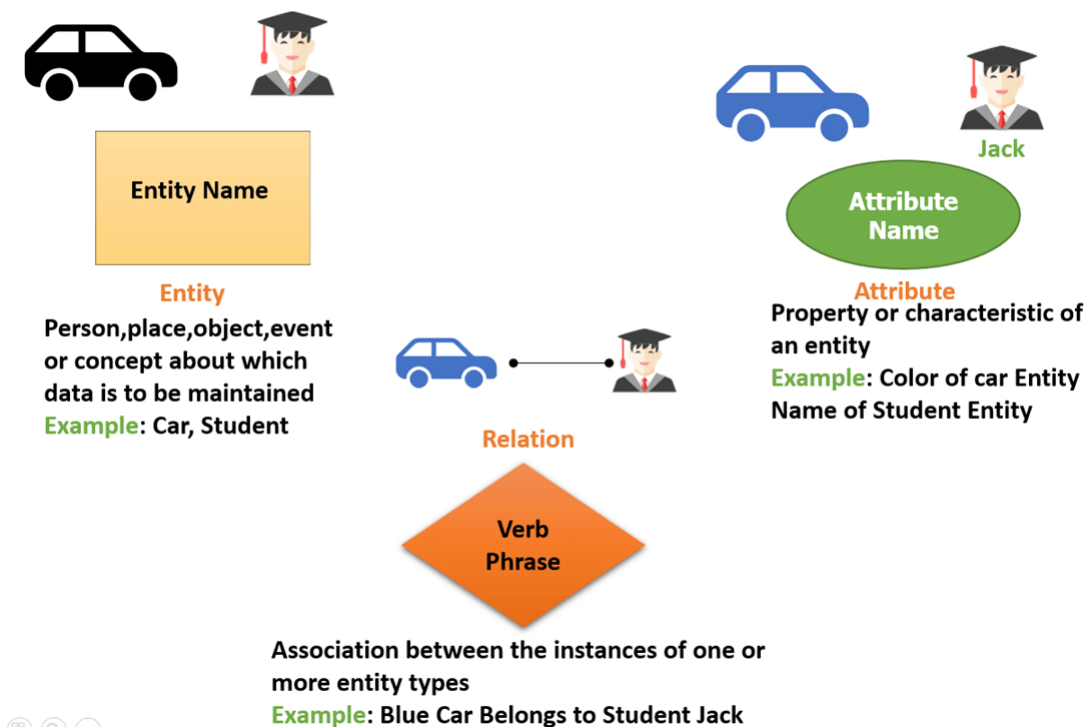
#### Components of the ER Diagram

This model is based on three basic concepts:

- Entities
- Attributes
- Relationships

#### ER Diagram Examples

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Roll Number, and DeptID. They might have relationships with Courses and Lecturers.



#### WHAT IS ENTITY?

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in a database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities must have an attribute and a unique key. Every entity is made up of some 'attributes' which represent that entity.

#### Examples of entities:

- **Person:** Employee, Student, Patient

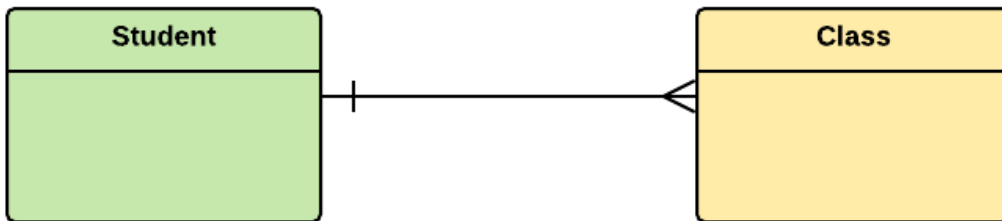
- **Place:** Store, Building
- **Object:** Machine, product, and Car
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course

Notation of an Entity

#### Entity set:

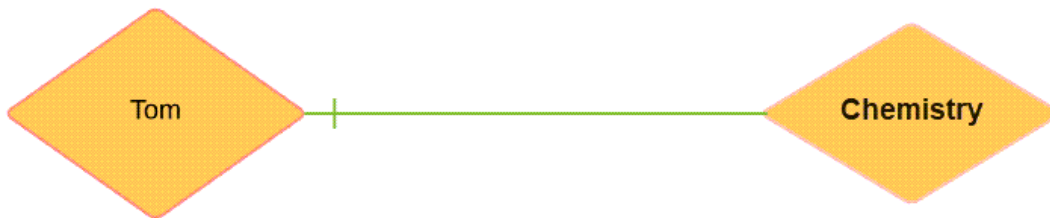
Student

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties which are also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.



#### Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.



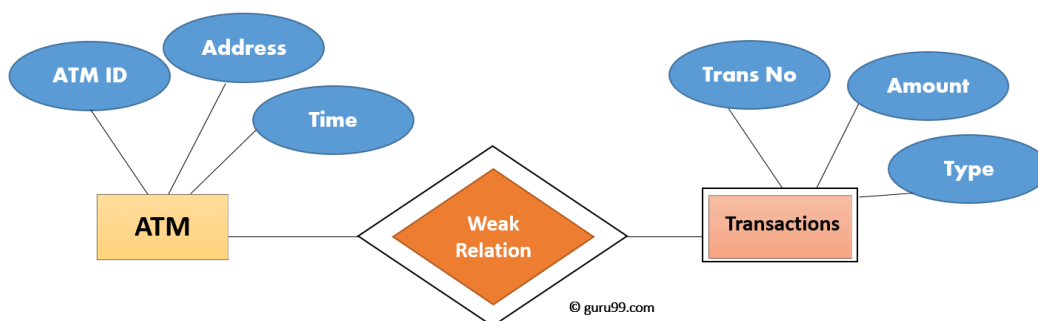
Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

#### For example:

- You are attending this lecture
- I am giving the lecture
- Just like entities, we can classify relationships according to relationship-types:
- A student attends a lecture
- A lecturer is giving a lecture.

#### Weak Entities

A weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For the weak entity sets need to have participation.



In above ER Diagram examples, "Trans No" is a discriminator within a group of transactions in an ATM.

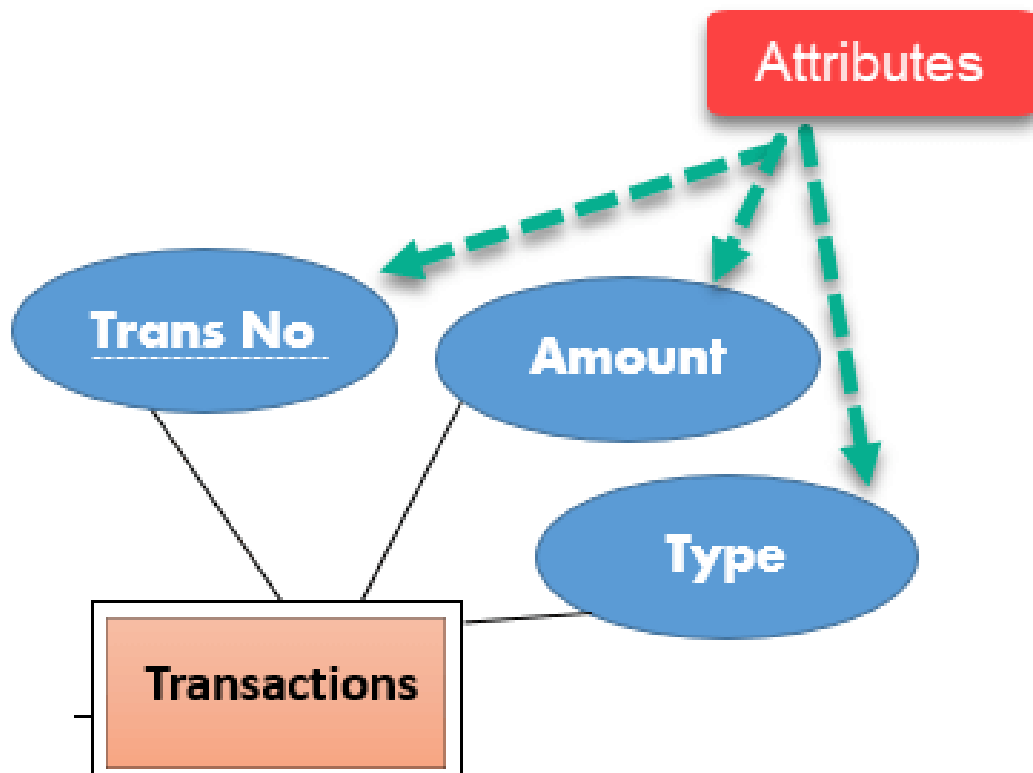
Let's learn more about a weak entity by comparing it with a Strong Entity

## Attributes

It is a single-valued property of either an entity-type or a relationship-type.

For example, a lecture might have attributes: time, date, duration, place, etc.

An attribute in ER Diagram examples, is represented by an Ellipse

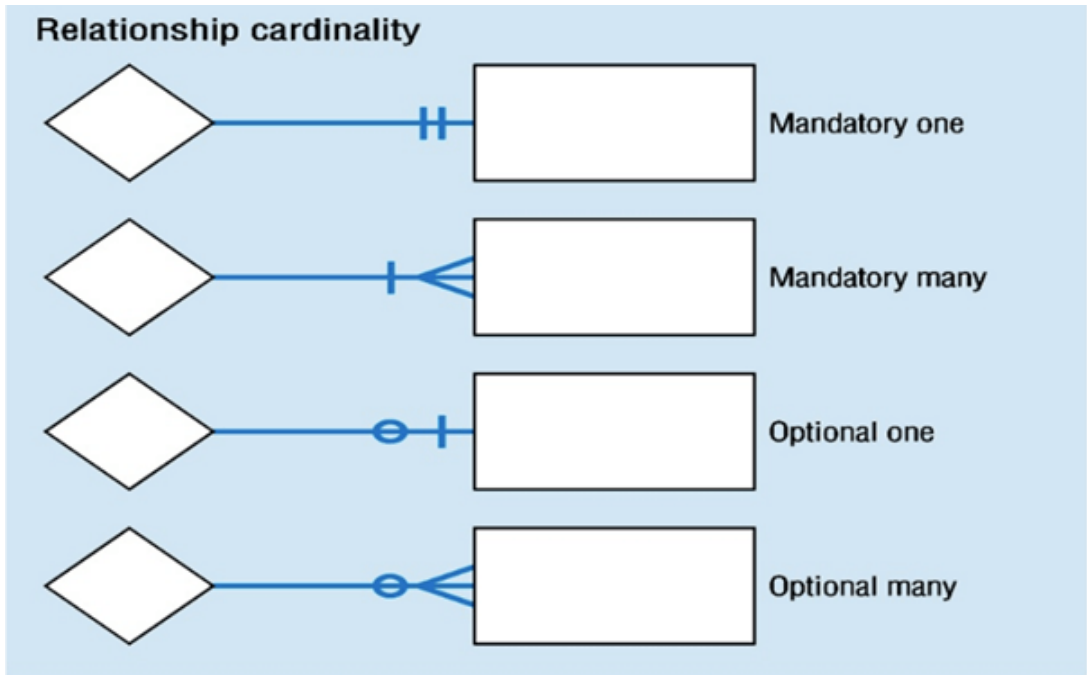


## Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

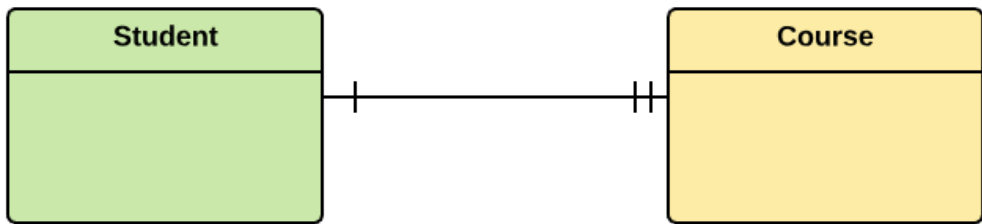
- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships
- Many-to-Many Relationships



#### 1. One-to-one:

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

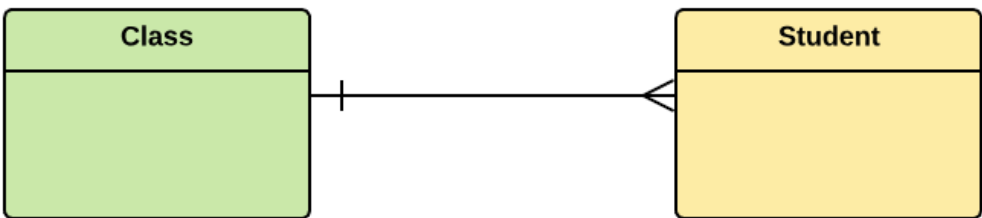
Example: One student can register for numerous courses. However, all those courses have a single line back to that one student



#### 2. One-to-many:

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

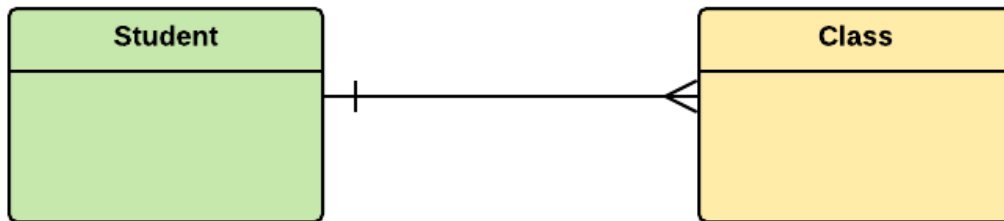
For example, one class is consisting of multiple students.



#### 3. Many to One

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

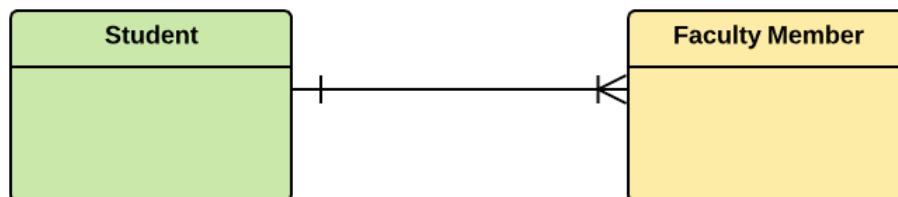
For example, many students belong to the same class.



#### 4. Many to Many:

One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.



#### How to Create an Entity Relationship Diagram (ERD)

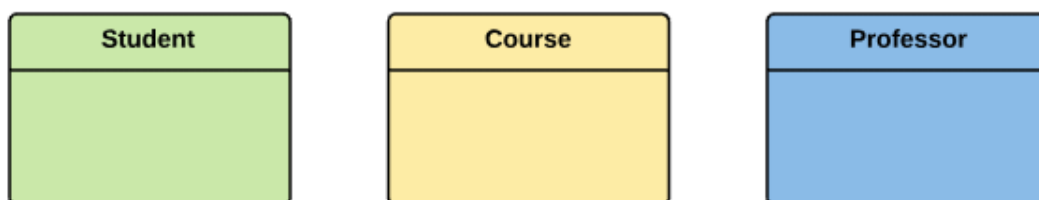
Let's study them with an Entity Relationship Diagram Example:

In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

##### Step 1) Entity Identification

We have three entities

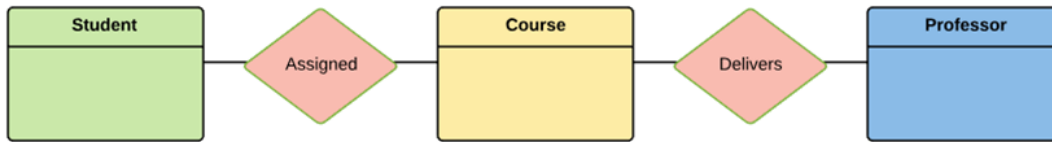
- Student
- Course
- Professor



##### Step 2) Relationship Identification

We have the following two relationships

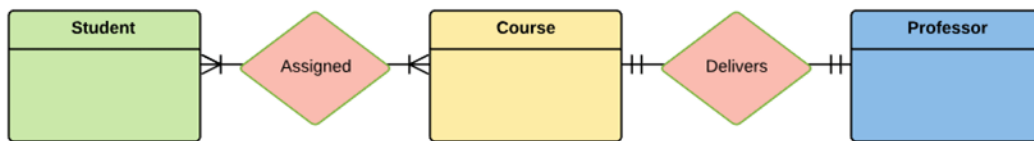
- The student is **assigned** a course
- Professor **delivers** a course



### Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course

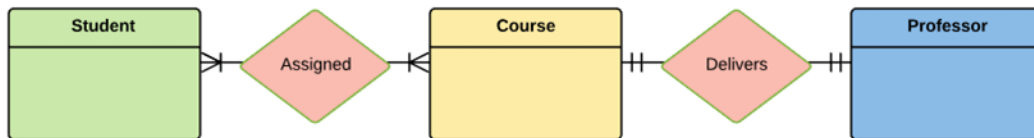


### Step 4) Identify Attributes

You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

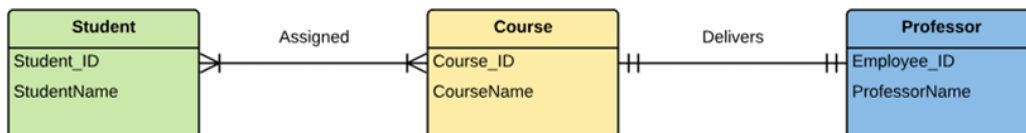
Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

### Step 5) Create the ERD Diagram

A more modern representation of Entity Relationship Diagram Example



## Conclusion

In this session we've learned about :

- Transactions in SQL
- ER Diagrams

Thank You !