

UNIT-5

Triggers:

A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA). A database with a set of associated triggers is generally called an active database.

Parts of trigger

A triggers description contains three parts, which are as follows –

- **Event** – An event is a change to the database which activates the trigger.
- **Condition** – A query that is run when the trigger is activated is called as a condition.
- **Action** – A procedure which is executed when the trigger is activated and its condition is true.

Use of trigger

Triggers may be used for any of the following reasons –

- To implement any complex business rule, that cannot be implemented using integrity constraints.
- Triggers will be used to audit the process. For example, to keep track of changes made to a table.
- Trigger is used to perform automatic action when another concerned action takes place.

Types of triggers

The different types of triggers are explained below –

- **Statement level trigger** – It is fired only once for DML statement irrespective of number of rows affected by statement. Statement-level triggers are the default type of trigger.
- **Before-triggers** – At the time of defining a trigger we can specify whether the trigger is to be fired before a command like INSERT, DELETE, or UPDATE is executed or after the command is executed. Before triggers are automatically used to check the validity of data before the action is performed. For instance, we can use before trigger to prevent deletion of rows if deletion should not be allowed in a given case.

- **After-triggers** – It is used after the triggering action is completed. For example, if the trigger is associated with the INSERT command then it is fired after the row is inserted into the table.
- **Row-level triggers** – It is fired for each row that is affected by DML command. For example, if an UPDATE command updates 150 rows then a row-level trigger is fired 150 times whereas a statement-level trigger is fired only for once.

Create database trigger

To create a database trigger, we use the CREATE TRIGGER command. The details to be given at the time of creating a trigger are as follows –

- Name of the trigger.
- Table to be associated with.
- When trigger is to be fired: before or after.
- Command that invokes the trigger- UPDATE, DELETE, or INSERT.
- Whether row-level triggers or not.
- Condition to filter rows.
- PL/SQL block is to be executed when trigger is fired.

The syntax to create database trigger is as follows –

```
CREATE [OR REPLACE] TRIGGER trigger-name
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF COLUMNS] } ON table
[FOR EACH ROW {WHEN condition}]
[REFERENCE [OLD AS old] [NEW AS new]]
BEGIN
PL/SQL BLOCK
END
```

Example1:

```
CREATE TRIGGER initcount BEFORE INSERT ON Students          /* Event */
DECLARE count INTEGER;
BEGIN  /* Action */
count := 0;
```

END

Example2:

```
CREATE TRIGGER incrcount AFTER INSERT ON Students          /* Event */
FOR EACH ROW
WHEN (new.age < 18)                                         /* Condition; 'new' is just-inserted tuple */
BEGIN                                                       /* Action; a procedure in Oracle's PL/SQL syntax */
    count := count + 1;
END
```

Triggers are defined as stored programs which are automatically executed whenever some events such as CREATE, ALTER, UPDATE, INSERT, DELETE takes place. They can be defined on a database, table, view with which event is associated.

Triggers can be broadly classified into **Row Level** and **Statement Level** triggers. Broadly, these can be differentiated as:

Row Level Triggers	Statement Level Triggers
Row level triggers executes once for each and every row in the transaction.	Statement level triggers executes only once for each single transaction.
Specifically used for data auditing purpose.	Used for enforcing all additional security on the transactions performed on the table.
“FOR EACH ROW” clause is present in CREATE TRIGGER command.	“FOR EACH ROW” clause is omitted in CREATE TRIGGER command.
Example: If 1500 rows are to be inserted into a table, the row level trigger would execute 1500 times.	Example: If 1500 rows are to be inserted into a table, the statement level trigger would execute only once.

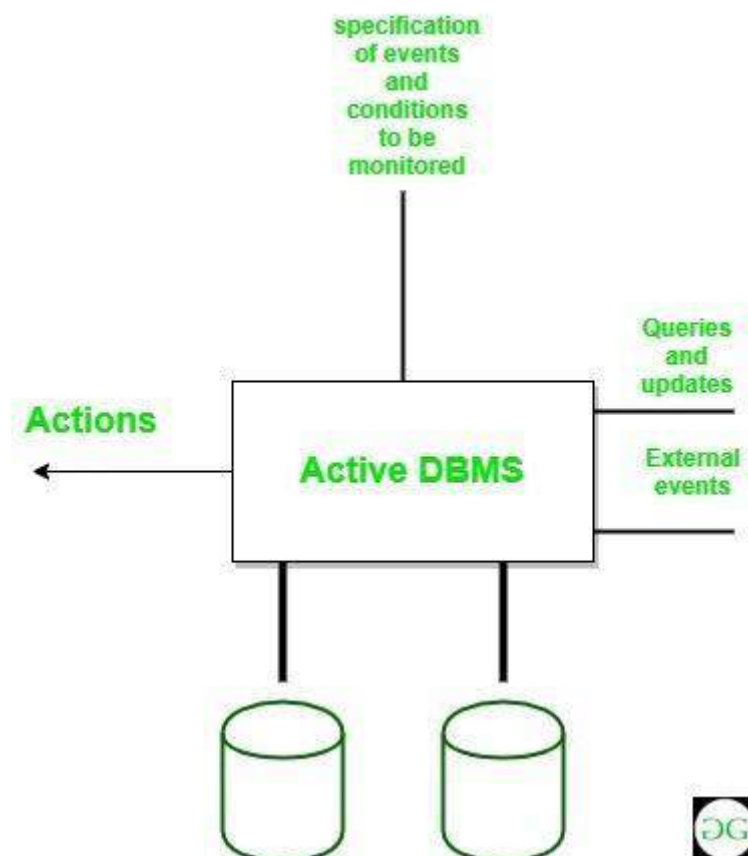
Active Databases:

Active Database is a database consisting of set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these

triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database) is activated or not, prior to executing the statement.

If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'. The DBMS executes such chains of trigger in some pre-defined manner but it affects the concept of understanding.



Features of Active Database:

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.
2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA (Event–Condition–Action) (ECA) rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution.

Advantages:

1. Enhances traditional database functionalities with powerful rule processing capabilities.
2. Enable a uniform and centralized description of the business rules relevant to the information system.

3. Avoids redundancy of checking and repair operations.
4. Suitable platform for building large and efficient knowledge base and expert systems.

Stored Procedures:

A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database. It is a subroutine or a subprogram in the regular computing language. **A procedure always contains a name, parameter lists, and SQL statements.** We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permission on the database tables.

Syntax:

DELIMITER &&

CREATE PROCEDURE procedure_name [[IN | OUT | INOUT] parameter_name datatype
[, parameter datatype]]

BEGIN

Declaration_section

Executable_section

END &&

DELIMITER;

Parameter Explanations

The procedure syntax has the following parameters:

Parameter Name	Descriptions
procedure_name	It represents the name of the stored procedure.
Parameter	It represents the number of parameters. It can be one or more than one.

Declaration_section	It represents the declarations of all variables.
Executable_section	It represents the code for the function execution.

MySQL procedure parameter has one of three modes:

IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

We can use the **CALL statement** to call a stored procedure. This statement returns the values to its caller through its parameters (IN, OUT, or INOUT).

The following syntax is used to call the stored procedure in MySQL:

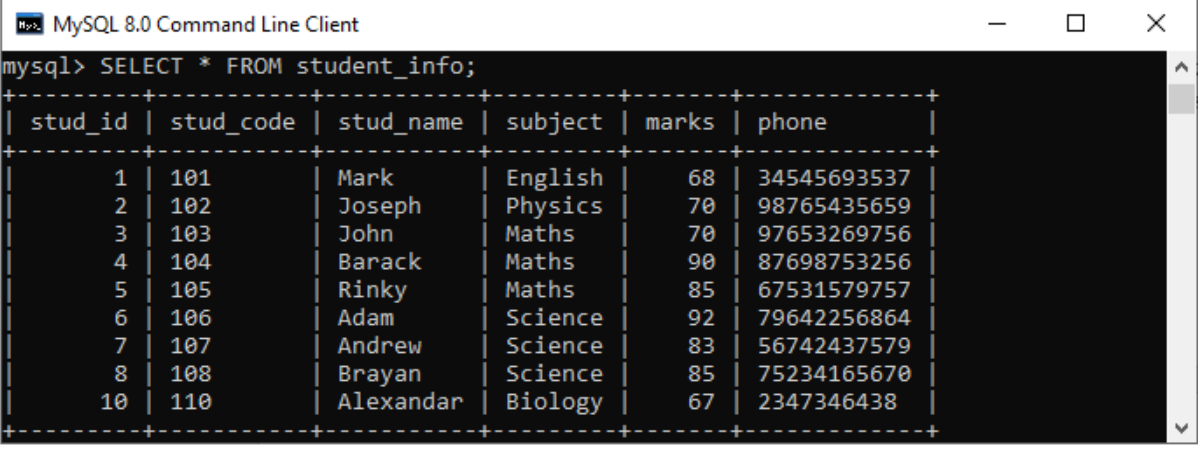
CALL procedure_name (parameter(s))

Example

First, we need to select a database that will store the newly created procedure. We can select the database using the below statement:

```
mysql> USE database_name;
```

Suppose this database has a table named **student_info** that contains the following data:



```
mysql> SELECT * FROM student_info;
```

stud_id	stud_code	stud_name	subject	marks	phone
1	101	Mark	English	68	34545693537
2	102	Joseph	Physics	70	98765435659
3	103	John	Maths	70	97653269756
4	104	Barack	Maths	90	87698753256
5	105	Rinky	Maths	85	67531579757
6	106	Adam	Science	92	79642256864
7	107	Andrew	Science	83	56742437579
8	108	Brayan	Science	85	75234165670
10	110	Alexandar	Biology	67	2347346438

Procedure without Parameter

Suppose we want to **display all records of this table whose marks are greater than 70** and count all the table rows. The following code creates a procedure named **get_merit_students**:

DELIMITER &&

CREATE PROCEDURE get_merit_student ()

BEGIN

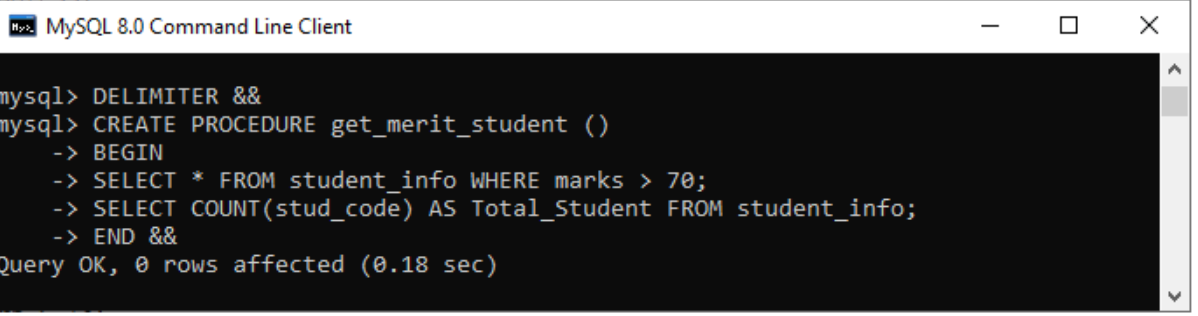
SELECT * FROM student_info **WHERE** marks > 70;

SELECT COUNT(stud_code) **AS** Total_Student **FROM** student_info;

END &&

DELIMITER ;

OUTPUT:



```
mysql> DELIMITER &&
mysql> CREATE PROCEDURE get_merit_student ()
-> BEGIN
-> SELECT * FROM student_info WHERE marks > 70;
-> SELECT COUNT(stud_code) AS Total_Student FROM student_info;
-> END &&
Query OK, 0 rows affected (0.18 sec)
```

Let us call the procedure to verify the output:

mysql> CALL get_merit_student();

It will give the output as follows:

```
MySQL 8.0 Command Line Client
mysql> CALL get_merit_student();
+----+-----+-----+-----+-----+-----+
| stud_id | stud_code | stud_name | subject | marks | phone |
+----+-----+-----+-----+-----+-----+
| 4 | 104 | Barack | Maths | 90 | 87698753256 |
| 5 | 105 | Rinky | Maths | 85 | 67531579757 |
| 6 | 106 | Adam | Science | 92 | 79642250804 |
| 7 | 107 | Andrew | Science | 83 | 50742437579 |
| 8 | 108 | Brayan | Science | 85 | 75234165670 |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

+-----+
| Total_Student |
+-----+
| 9 |
+-----+
1 row in set (0.03 sec)
```

Procedures with IN Parameter

In this procedure, we have used the IN parameter as '**var1**' of integer type to accept a number from users. Its body part fetches the records from the table using a **SELECT** statement and returns only those rows that will be supplied by the user. It also returns the total number of rows of the specified table. See the procedure code:

DELIMITER &&

CREATE PROCEDURE get_student (**IN** var1 INT)

BEGIN

SELECT * **FROM** student_info **LIMIT** var1;

SELECT COUNT(stud_code) **AS** Total_Student **FROM** student_info;

END &&

DELIMITER ;

After successful execution, we can call the procedure as follows:

```
mysql> CALL get_student(4);
```

We will get the below output:

```
MySQL 8.0 Command Line Client
mysql> CALL get_student(4);
+----+-----+-----+-----+-----+-----+
| stud_id | stud_code | stud_name | subject | marks | phone |
+----+-----+-----+-----+-----+-----+
| 1 | 101 | Mark | English | 68 | 34545693537 |
| 2 | 102 | Joseph | Physics | 70 | 98765435659 |
| 3 | 103 | John | Maths | 70 | 97653269756 |
| 4 | 104 | Barack | Maths | 88 | 87698753256 |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

+-----+
| Total_Student |
+-----+
| 9 |
+-----+
```


Procedures with OUT Parameter:

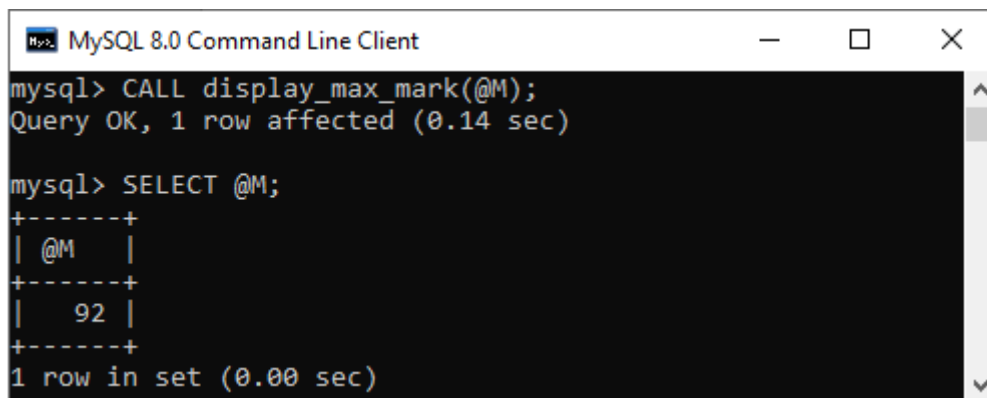
In this procedure, we have used the OUT parameter as the '**highestmark**' of integer type. Its body part fetches the maximum marks from the table using a **MAX()** function. See the procedure code:

```
DELIMITER &&
CREATE PROCEDURE display_max_mark (OUT highestmark INT)
BEGIN
    SELECT MAX(marks) INTO highestmark FROM student_info;
END &&
DELIMITER ;
```

This procedure's parameter will get the highest marks from the **student_info** table. When we call the procedure, the OUT parameter tells the database systems that its value goes out from the procedures. Now, we will pass its value to a session variable **@M** in the CALL statement as follows:

```
mysql> CALL display_max_mark(@M);
mysql> SELECT @M;
```

Here is the output:



```
MySQL 8.0 Command Line Client
mysql> CALL display_max_mark(@M);
Query OK, 1 row affected (0.14 sec)

mysql> SELECT @M;
+-----+
| @M    |
+-----+
| 92    |
+-----+
1 row in set (0.00 sec)
```

Execution of stored procedures from Java

The CallableStatement of JDBC API is used to call a stored procedure. A Callable statement can have output parameters, input parameters, or both. The prepareCall() method of connection interface will be used to create CallableStatement object.

Following are the steps to use Callable Statement in Java to call Stored Procedure:

1) Load MySQL driver and Create a database connection.

```

import java.sql.*;

public class JavaApplication1 {

    public static void main(String[] args) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");

        Connection
con=DriverManager.getConnection("jdbc:mysql://localhost/root","geek","geek");

    }
}

```

2) Create a SQL String

We need to store the SQL query in a String.

```
String sql_string="insert into student values(?,?,?)";
```

3) Create CallableStatement Object

The prepareCall() method of connection interface will be used to create CallableStatement object. The sql_string will be passed as an argument to the prepareCall() method.

```
CallableStatement cs = con.prepareCall(sql_string);
```

4) Set The Input Parameters

Depending upon the data type of query parameters we can set the input parameter by calling setInt() or setString() methods.

```
cs.setString(1,"geek1");
```

```
cs.setString(2,"python");
```

```
cs.setString(3,"beginner");
```

5) Call Stored Procedure

Execute stored procedure by calling execute() method of CallableStatement class.

Example of using Callable Statement in Java to call Stored Procedure

```

// Java program to use Callable Statement
// in Java to call Stored Procedure

```

```
package javaapplication1;
```

```
import java.sql.*;
```

```
public class JavaApplication1 {
```

```

    public static void main(String[] args) throws Exception
    {

```

```

Class.forName("com.mysql.jdbc.Driver");

// Getting the connection
Connection con = DriverManager.getConnection("jdbc:mysql://localhost/root",
"acm", "acm");

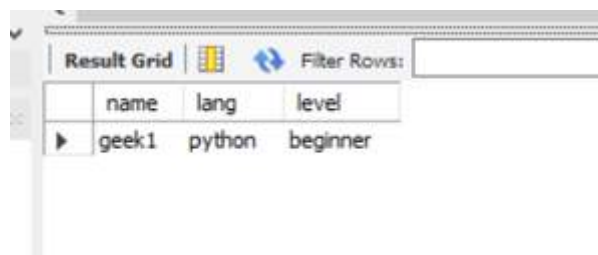
String sql_string = "insert into students values(?,?,?)";

// Preparing a CallableStatement
CallableStatement cs = con.prepareCall(sql_string);

cs.setString(1, "geek1");
cs.setString(2, "python");
cs.setString(3, "beginner");
cs.execute();
System.out.print("uploaded successfully\n");
}
}

```

Output:



The screenshot shows a window titled 'Result Grid' with a 'Filter Rows' input field. Below the header, there is a table with three columns: 'name', 'lang', and 'level'. The first row of data contains the values 'geek1', 'python', and 'beginner'.

	name	lang	level
▶	geek1	python	beginner

students table after running code

Database Administrator (DBA)

A **Database Administrator (DBA)** is individual or person responsible for controlling, maintenance, coordinating, and operation of database management system. Managing, securing, and taking care of database system is prime responsibility.

Importance of Database Administrator (DBA):

- Database Administrator manages and controls three levels of database like internal level, conceptual level, and external level of Database management system architecture and in discussion with comprehensive user community, gives definition of world view of database. It then provides external view of different users and applications.
- Database Administrator ensures / held responsible to maintain integrity and security of database from unauthorized users. DBA grants permission to users of database and contains profile of each and every user in database.

- Database Administrator also held accountable that database is protected and secured and that any chance of data loss keeps at minimum.

Role and Duties of Database Administrator (DBA):

- **Decides hardware –**
They decide economical hardware, based upon cost, performance and efficiency of hardware, and best suits organisation. It is hardware which is interface between end users and database.
- **Manages data integrity and security –**
Data integrity need to be checked and managed accurately as it protects and restricts data from unauthorized use. DBA eyes on relationship within data to maintain data integrity.
- **Database design –**
DBA is held responsible and accountable for logical, physical design, external model design, and integrity and security control.
- **Database implementation –**
DBA implements DBMS and checks database loading at time of its implementation.
- **Query processing performance –**
DBA enhances query processing by improving their speed, performance and accuracy.
- **Tuning Database Performance –**
If user is not able to get data speedily and accurately then it may loss organization business. So by tuning SQL commands DBA can enhance performance of database.

OR

Database Administrator Responsibilities

A DBA has many responsibilities. A good-performing database is in the hands of DBA.

- **Installing and upgrading the DBMS Servers:** – DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions that come into the market or requirement. If there is any failure in the up-gradation of the existing servers, he should be able to revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hotfixes/ patches to the DBMS servers.
- **Design and implementation:** – Designing the database and implementing is also DBA's responsibility. He should be able to decide on proper memory management, file organizations, error handling, log maintenance, etc for the database.
- **Performance tuning:** – Since the database is huge and it will have lots of tables, data, constraints, and indices, there will be variations in the performance from time to time. Also, because of some designing issues or data growth, the database will not work as expected. It is the responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs work in a fraction of seconds.
- **Migrate database servers:** – Sometimes, users using oracle would like to shift to SQL server or Netezza. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.

- **Backup and Recovery:** – Proper backup and recovery programs need to be developed by DBA and has to be maintained. This is one of the main responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.
- **Security:** – DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
- **Documentation:** – DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

Creating Users, Grant / Revoke Permissions on tables using DCL Commands:

Data Controlling Language:

Data Controlling Language (DCL) helps users to retrieve and modify the data stored in the database with some specified queries. Grant and Revoke belong to these types of commands of the Data controlling Language. DCL is a component of SQL commands.

Grant Privileges on Table

You can grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

Syntax

The syntax for granting privileges on a table in SQL Server is:

GRANT privileges ON object TO user;

privileges

The privileges to assign are listed below. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.

Privilege	Description
ALL	ALL does not grant all permissions for the table. Rather, it grants the ANSI-92 permissions which are SELECT, INSERT, UPDATE, DELETE, and REFERENCES.

object

The name of the database objects that you are granting permissions for. In the case of granting privileges on a table, this would be the table name.

user

The name of the user who will be granted these privileges.

Example

Let's look at some examples of how to grant privileges on tables in SQL Server.

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called *employees* to a user name *smithj*, you would run the following GRANT statement:

GRANT SELECT/INSERT/UPDATE/ DELETE ON employees TO smithj;

You can also use the ALL keyword to indicate that you wish to grant the ANSI-92 permissions

(ie: SELECT, INSERT, UPDATE, DELETE, and REFERENCES) to a user named *smithj*.

For example:

GRANT ALL ON employees TO smithj;

If you wanted to grant only SELECT access on the *employees* table to all users, you could grant the privileges to the public role.

For example:

GRANT SELECT ON employees TO public;

Revoke Privileges on Table:

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

Syntax

The syntax for revoking privileges on a table in SQL Server is:

REVOKE privileges ON object FROM user;

privileges

It is the privileges to assign. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
ALL	ALL does not revoke all permissions for the table. Rather, it revokes the ANSI-92 permissions which are SELECT, INSERT, UPDATE, DELETE, and REFERENCES.

object

The name of the database objects that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

user

The name of the user that will have these privileges revoked.

Example

Let's look at some examples of how to revoke privileges on tables in SQL Server.

For example, if you wanted to revoke DELETE privileges on a table called *employees* from a user named *anderson*, you would run the following REVOKE statement:

REVOKE DELETE ON employees FROM anderson;

If you wanted to revoke ALL ANSI-92 permissions (ie: SELECT, INSERT, UPDATE, DELETE, and REFERENCES) on a table for a user named *anderson*, you could use the ALL keyword as follows:

REVOKE ALL ON employees FROM anderson;

If you had granted SELECT privileges to the public role (ie: all users) on the *employees* table and you wanted to revoke these privileges, you could run the following REVOKE statement:

REVOKE SELECT ON employees FROM public;

Differences between Grant and Revoke commands:

S.NO	Grant	Revoke
1	This DCL command grants permissions to the user on the database objects.	This DCL command removes permissions if any granted to the users on database objects.
2	It assigns access rights to users.	It revokes the user access rights of users.
3	For each user you need to specify the permissions.	If access for one user is removed; all the particular permissions provided by that users to others will be removed.
4	When the access is decentralized granting permissions will be easy.	If decentralized access removing the granted permissions is difficult.