**Md. Mahmudul Hassan #MCE 07905539**

**SQL TABLE**

|  |  |
| --- | --- |
| **sourcecredential** | |
| credential\_id | INTEGER |
| source\_id | INTEGER(PK) |
| credential\_info | INTEGER |

|  |  |
| --- | --- |
| **sourceinfo** | |
| source\_id | INTEGER(PK) |
| source\_name | varchar(100) |
| source\_contact | varchar(11) |
| source\_address | varchar(100) |
| username | varchar(30) |
| password | varchar(30) |

|  |  |
| --- | --- |
| **crimeinfo** | |
| crime\_id | INTEGER(PK) |
| crime\_description | varchar(200) |
| date\_reported | varchar(30) |
| time\_reported | timestamp |
| Location | varchar(100) |
| source\_id | INTEGER(PK) |
| crime\_category\_id | INTEGER |
| case\_status | varchar(150) |

|  |  |
| --- | --- |
| **crimecategory** | |
| crime\_category\_id | INTEGER(PK) |
| category\_description | varchar(150) |

|  |  |
| --- | --- |
| **responseinfo** | |
| response\_info\_id | INTEGER |
| crime\_id | INTEGER(PK) |
| response\_date | varchar(30) |
| response\_time | varchar(30) |
| police\_id | INTEGER |
| case\_status | varchar(150) |

|  |  |
| --- | --- |
| **policeinfo** | |
| police\_id | INTEGER(PK) |
| police\_name | varchar(100) |
| designation | varchar(100) |
| mobile\_no | varchar(11) |
| police\_station | varchar(100) |
| username | varchar(30)(FK) |
| password | varchar(30) |

|  |  |
| --- | --- |
| **admin** | |
| user\_id | INTEGER(PK) |
| fullname | varchar(100) |
| address | varchar(100) |
| contact | varchar(11) |
| username | varchar(30)(FK) |
| password | varchar(30) |

**SQL VIEW**

To create a  [view](https://www.sqlservertutorial.net/sql-server-views/) in SQL Server

* In SQL, a view is a virtual table based on the result-set of an SQL statement.
* A view contains rows and columns, just like a main table.
* **Syntax for creating a view:**

**--SQL VIEW-1--**

CREATE VIEW Source\_view AS

SELECT source\_id, source\_name, source\_contact

FROM Sourceinfo;

SELECT \* FROM Source\_view;

**--SQL VIEW-2--**

CREATE VIEW police\_view AS

SELECT police\_id, police\_name, police\_station

FROM policeinfo;

SELECT \* FROM police\_view;

**--SQL VIEW-3--**

CREATE VIEW crime\_view AS

SELECT crime\_id, crime\_description, location, case\_status

FROM crimeinfo;

SELECT \* FROM crime\_view;

* **Modifying a view:**

UPDATE Source\_view

SET source\_name=’Mahmud’

WHERE source\_id=103;

UPDATE police\_view

SET police\_station=’Romna Model Thana’

WHERE police\_id=903;

UPDATE crime\_view

SET case\_status=’Five year jail’

WHERE crime\_id=500;

**In this syntax:**

* First, specify the name of the view after the CREATE VIEW keywords. The schema\_name is the name of the schema to which the view belongs.
* Second, specify a SELECT statement (select\_statement) that defines the view after the AS keyword. The [SELECT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) statement can refer to one or more tables.

If we don’t explicitly specify a list of columns for the view, SQL Server will use the column list derived from the SELECT statement.

In case we want to redefine the view e.g., adding more columns to it or removing some columns from it, you can use the OR ALTER keywords after the CREATE VIEW keywords.

**SQL Trigger**

* **CREATE TRIGGER statement**

The CREATE TRIGGER statement allows we to create a new trigger that is fired automatically whenever an event such as [INSERT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/), [DELETE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-delete/), or [UPDATE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-update/) occurs against a table.

The following illustrates the syntax of the CREATE TRIGGER statement:

--SQL TRIGGER

SELECT \* FROM Sourceinfo

GO

CREATE TABLE ActionStatus

(

crime\_data varchar(50),

ActionStatus varchar(100)

)

GO

CREATE TRIGGER trgcrimeInsert

ON sourceinfo

FOR INSERT

AS

BEGIN

DECLARE @source\_id INTEGER

SELECT @source\_id = source\_id FROM inserted

INSERT INTO ActionStatus

VALUES (@source\_id, 'Data has been Inserted at – ' + CAST(GETDATE() AS varchar(50)))

END

INSERT INTO ActionStatus VALUES('0001', 'Implementation')

SELECT \* FROM ActionStatus

**In this syntax:**

* The schema\_name is the name of the schema to which the new trigger belongs. The schema name is optional.
* The trigger\_name is the user-defined name for the new trigger.
* The table\_name is the table to which the trigger applies.
* The event is listed in the AFTER clause. The event could be INSERT, UPDATE, or DELETE. A single trigger can fire in response to one or more actions against the table.
* The NOT FOR REPLICATION option instructs SQL Server not to fire the trigger when data modification is made as part of a replication process.
* The sql\_statements is one or more Transact-SQL used to carry out actions once an event occurs.

**SQL Stored Procedure**

* **Creating a stored procedure**

To create a stored procedure that wraps this query, we use the CREATE PROCEDURE statement as follows:

--SQL Stored Procedure

CREATE PROCEDURE SpQueryresponseinfo

AS

BEGIN

SELECT \* FROM responseinfo

END;

ALTER PROCEDURE SpQueryresponseinfo

AS

BEGIN

SELECT E.Name.s.crime\_id FROM responseinfo E

LEFT JOIN crimeinfo S ON E.crime\_id=S.crime\_description

END;

In this syntax:

* The **responseinfo** is the name of the stored procedure.
* The AS keyword separates the heading and the body of the stored procedure.
* If the stored procedure has one statement, the BEGIN and END keywords surrounding the statement are optional. However, it is a good practice to include them to make the code clear.

Note that in addition to the CREATE PROCEDURE keywords, we can use the CREATE PROC keywords to make the statement shorter.

* **Executing a stored procedure**

## EXEC SpQueryresponseinfo