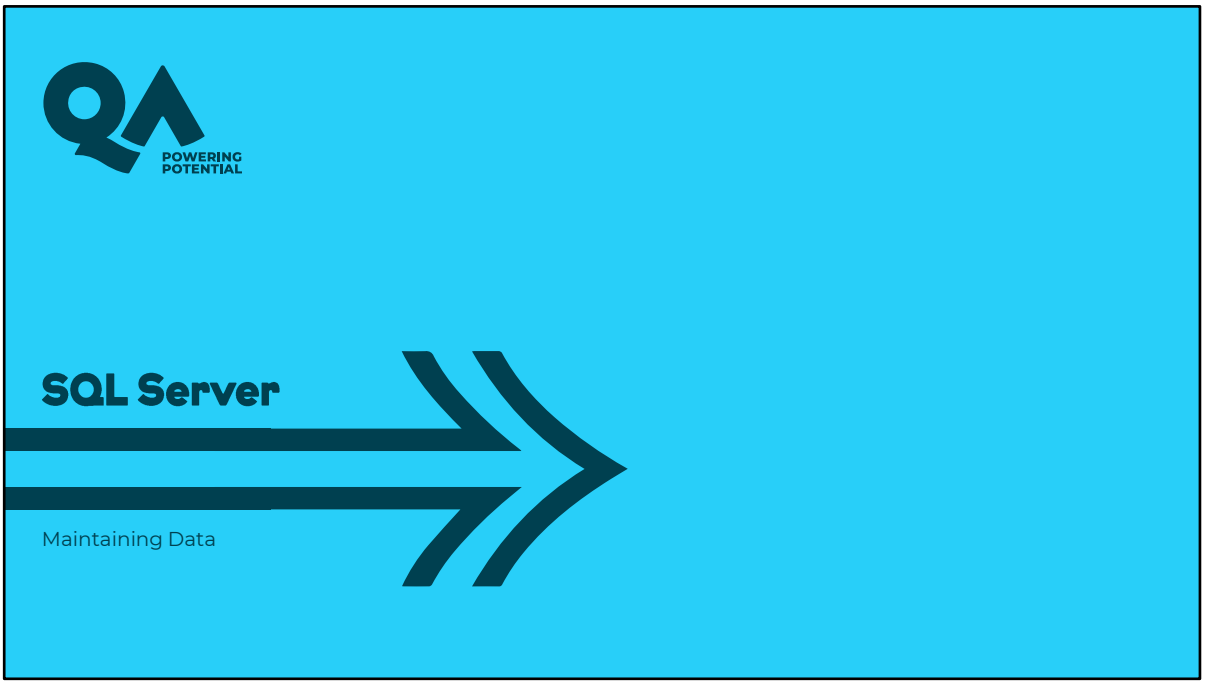


This module will enable the learner to write SQL to maintain data on a table. Specifically the learner will be able to:

- Use INSERT to add a new record to a table including NULL, date / time, default and function generated values.
- Use INSERT to copy records from one table to another table.
- Use UPDATE to amend records on a table.
- Use UPDATE to amend records based on another table.
- Use DELETE to remove records from a table.
- Use DELETE to remove records based on another table.

Code examples will be used to explain and demonstrate the content and for learners to imitate. Supporting exercises will provide practice opportunities. At the end of the module there is a short quiz to review the topic.

This module provides an introduction to maintaining data on a table both directly and using another table. The topics covered are:

- Objectives – What are the Learning Outcomes and Assessment Criteria for the module.
- Add a Record – Using the INSERT statement.
- Amend a Record – Using the UPDATE statement.
- Remove a Record – Using the DELETE statement.
- Review – A short recap of the module including a quiz.

INSERT Single Record – CREATE TABLE

```
-- CREATE TABLE new_dept
CREATE TABLE new_dept
    (
    dept_no INT NOT NULL,
    dept_name VARCHAR(20) UNIQUE NOT NULL,
    manager VARCHAR(20) DEFAULT 'Manager TBC',
    sales_target DECIMAL(12, 4) NULL,
    update_date_time DATETIME DEFAULT GETDATE(),
    agreed CHAR(1) NULL,
    CONSTRAINT PK_new_dept_no PRIMARY KEY (dept_no)
    );
```

To demonstrate the various options to maintain data a new table will be used which needs to be created first. Creating tables will be covered in more detail in the next module '08– Maintain Database Objects'.
The SQL creates a table called new_dept with six columns or fields:

- dept_no – A mandatory or not null integer field.
- dept_name – A mandatory or not null text field with a maximum length of 20 characters; each dept_name must also be unique.
- manager – An optional text field with a maximum length of 20 characters; if a value is not entered it is populated with a default of 'Manager TBC'.
- sales_target – An optional number field of up to 12 digits of which 4 are decimal places; if a value is not entered it is populated with a value of 'NULL'.
- update_date_time – An optional date and time field; if a value is not entered it is populated with the current date and time using the GETDATE() function.
- agreed – An optional text field of that is a fixed single character; if a value is not entered it is populated with a value of 'NULL'.

A primary key of dept_no is also specified; each record on the table must have a unique dept_no.

INSERT Single Record – Using Columns

```
INSERT INTO new_dept
    (dept_no,dept_name,manager,sales_target,
     update_date_time,agreed)
VALUES
    (6,'Factory Fittings','Francis Forsyth',60.0000,
     '2016-08-10 15:12:48','Y');
```

- Default to automatically commit transactions
  - **SET IMPLICIT_TRANSACTIONS ON** or **OFF**
  - Menu bar: Tools / Options

INSERT will add a new record to a table.
First, it is required to specify the table that new record will be added to.
It is recommended that you then specify the columns or fields that data will be added for. As we will see soon this is optional. This could be a subset of the fields on the table but must include any mandatory or not-null fields. If not the SQL will fail when it is executed. The fields do not need to be in the order they are specified on the table.
Next we specify the actual values that will be added for each field. These must be in the same order as the fields have been specified and the format and size of data appropriate to the field definition on the table. For example numbers as integers or decimals without quotes and text in quotes. The default colours in the 'SQL Query Panel' indicate the type of data. Again the SQL will fail when it is executed if the data is not specified correctly.
Values and intended fields can be accidentally swopped if they are of the same type. For example dept_name and manager are both VARCHAR(20) fields. It is down to the developer to make sure the order of fields and corresponding values align as the SQL will execute successfully in such circumstances and put the wrong values in the corresponding fields.
The record will be added automatically when the SQL is executed if the 'Set Implicit Transactions' option is off. This is the default option and can be set using SQL or by ticking the appropriate box in the Options Window. If the option is on additional SQL will be required to commit the change. This will be covered in more

detail later in the module.

'Manually' Commit Transactions

```
-- INSERT single record using columns
SET IMPLICIT_TRANSACTIONS ON;
BEGIN TRANSACTION;

INSERT INTO new_dept
    (dept_no,dept_name,manager,sales_target,
     update_date_time,agreed)
VALUES
    (6,'Factory Fittings','Francis Forsyth',60.0000,
     '2016-08-10 15:12:48','Y');

COMMIT TRANSACTION;
```

- ROLLBACK if error – Reverse changes
- Why needed?

The default mode for SQL Server is to automatically commit any changes to data or transactions. For simple transactions this is OK but for more complex or multiple transactions it is necessary to confirm that all the transactions will be successful before making or committing the change. For example if transferring money between bank account A and bank account B there are at least three transactions taking place:

- An update to the account A record on the account table to reduce its balance by the amount being transferred.
- An update to the account B record on the account table to increase its balance by the amount being transferred.
- Create or insert a new record on the History table with the details of the transfer – Account from / to, value, date / time.

If any of these three transactions were not possible then the transfer should be cancelled. The SQL to do this is:

- SET IMPLICIT_TRANSACTIONS ON = 0 – Set Implicit Transactions on if not already switched on.
- BEGIN TRANSACTION – Start the data changes.
- Data changes – Typically several INSERTs, UPDATEs and / or DELETEs.
- Some logic to determine if the complete transaction is possible.

- COMMIT TRANSACTION – If the complete transaction is possible all the data changes are actioned to their new state.
- ROLLBACK TRANSACTION – If the complete transaction is not possible all the data changes are reversed to their original state.

INSERT Single Record – Not using Columns

```
INSERT INTO new_dept
VALUES
    (7,'Garden Gnomes','Geoff Gordan',70.0000,
    '2016-08-10 15:27:23','Y');
```

- **Will the statement work if the column values are not in the same column order as in the table?**

A record can be added without specifying the fields on the table. The values will be added to the fields in the order they are specified on the table. Care is required to ensure the correct values are added to their corresponding fields. If insufficient values are specified the remaining fields will be populated with a 'NULL', default or calculated value as appropriate. If any remaining field is mandatory or non-null then the SQL will fail.

INSERT Single Record using Non-Null Columns

```
INSERT INTO new_dept
    (dept_no,dept_name)
VALUES
    (8,'Hospital Hotplants');
```

- **What does the SQL do?**

In this example only the mandatory or non-null fields on the table are specified. The corresponding values will be added for those fields. The optional or null fields will be set to their specified default values, such as manager to ‘Manager TBC’, or calculated using the specified function, such as GETDATE() for update_date_time. The other fields are set to ‘NULL’.
If a non-null column is not set to a valid value the SQL will fail when executed.

INSERT Single Record using a Function

```
INSERT INTO new_dept
    (dept_no,dept_name,manager,sales_target,
     update_date_time,agreed)
VALUES
    (9,'Industrial Iceboxes','Ian Irving',90.0000,
     SYSDATETIME(),'Y');
```

- **What does the SQL do?**

The value for a field can set using a function. In this case update_date_time is set to the date and time calculated by the SYSDATETIME function when the SQL is executed.

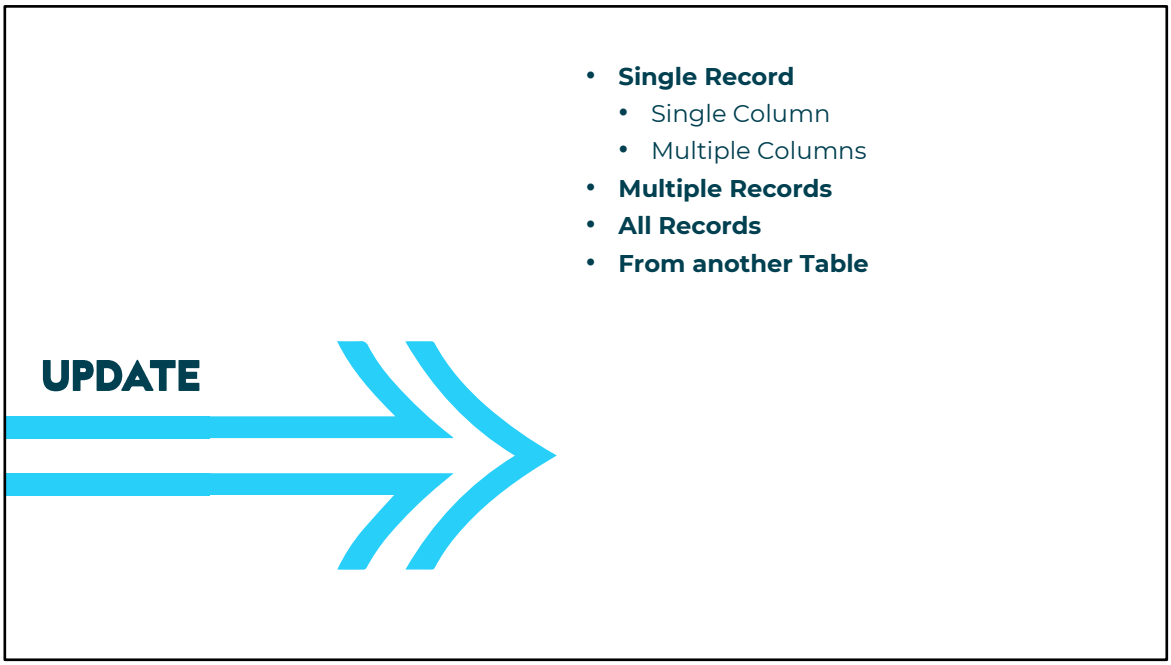
INSERT Multiple Records from another Table

```
INSERT INTO dept
      (dept_no,dept_name,manager,sales_target)

SELECT dept_no,dept_name,manager,sales_target
FROM  new_dept
WHERE agreed = 'Y';
```

- **What does the SQL do?**

Multiple records can be added from one table to another by using the SELECT statement.
The fields specified from the 'Select' table must correspond in number, order, format and size with the fields to be added on the 'Insert' table. They do not need to be the same name. An addition WHERE clause can be added to only add records that met certain criteria on the 'Select' table.
Again any fields not specified will be set to their default, calculated or a 'NULL' value.



One, multiple or all records can amended on a table directly or my using another table. The options covered are:

- Single Record
  - Single Column – Amending just one column on just one record on a table..
  - Multiple Columns – Amending multiple columns on just one record on a table.
- Multiple Records – Amending many records on a table.
- All Records – Amending all records on a table.
- From another Table – Amending records based on data in another table.

UPDATE Single Record Single Column

```
UPDATE new_dept
SET sales_target = 65
WHERE dept_no = 6;
```

- **What does the SQL do?**

UPDATE will amend an existing record.
First, specify the table that the record will be amended on.
Next the field that will be amended and its new value. As with the INSERT the new value needs to be correct format and size for the amended field.
A WHERE clause can be used to specify which records will be amended. Quite often this will be just be a single record as determined by its primary key value; in this case the record with a dept_no equal to 6.

UPDATE Single Record multiple Columns

```
UPDATE new_dept
SET dept_name = 'Factory Fixtures',
    sales_target = 70
WHERE dept_no = 6;
```

- **What does the SQL do?**

Several fields can be amended in one UPDATE by listing the required fields and their new values separated by a comma.
As in the previous example the WHERE clause specifies which records will be changed; in this case just the single record with a dept_no equal to 6.

UPDATE Multiple Records

```
UPDATE new_dept
SET    sales_target = sales_target * 1.25
WHERE sales_target IS NOT NULL;
```

- **What does the SQL do?**

A field can be amended by using a function. This could be independent of the field, or as in this case, based on the current value of the field.
Dependent on the WHERE clause multiple records can be updated. In this case all records with a sales_target that is an actual, non-null, value will be amended.

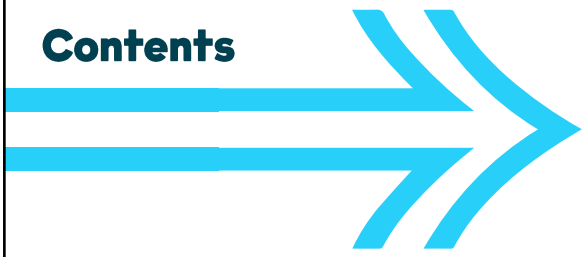
UPDATE All Records

```
UPDATE new_dept
SET sales_target = 100;
```

- **What does the SQL do?**

If the WHERE clause is omitted all records on the table will be amended.

Contents



The next section covers removing a record using the DELETE statement.

DELETE Single Record

```
DELETE FROM new_dept
WHERE dept_no = 6;
```

- **What does the SQL do?**

DELETE will remove an existing record.
First, specify the table that the record will be removed from. Note there is no need to specify any fields as the whole record will be removed not just individual fields.
A WHERE clause can be used to specify which records will be removed. As in the previous example the WHERE clause restricts the removal to just the single record with a dept_no equal to 6.
As before the record will be amended automatically when the SQL is run if the 'Auto-Commit Transactions' option is ticked on the Query Tab.

DELETE Multiple Records

```
DELETE FROM new_dept
WHERE agreed = 'Y';
```

- **What does the SQL do?**

Dependent on the WHERE clause multiple records can be removed. In this case all records that have an agreed value of ‘Y’ are removed.

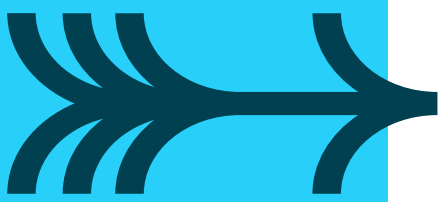
DELETE All Records

```
DELETE FROM new_dept;
```

- **What does the SQL do?**
  - **Will the table still exist?**

If the WHERE clause is omitted all records on the table will be removed.

REVIEW OBJECTIVES, QUESTIONS AND FEEDBACK



In this chapter you learned to:

- Use INSERT to add a new record to a table including NULL, date / time, default and function generated values
- Use INSERT to copy records from one table to another table
- Use UPDATE to amend records on a table
- Use UPDATE to amend records based on another table
- Use DELETE to remove records from a table

Have we achieved the Learning Outcomes for this module? As a learner can you:

- 7. Write SQL to maintain data on a table.

Also the Assessment Criteria? As a learner can you:

- 7.1 Use INSERT to add a new record to a table including NULL, date / time, default and function generated values.
- 7.2 Use INSERT to copy records from one table to another table.
- 7.3 Use UPDATE to amend records on a table.
- 7.4 Use UPDATE to amend records based on another table.
- 7.5 Use DELETE to remove records from a table.
- 7.6 Use DELETE to remove records based on another table.

POWERING POTENTIAL

Thank you