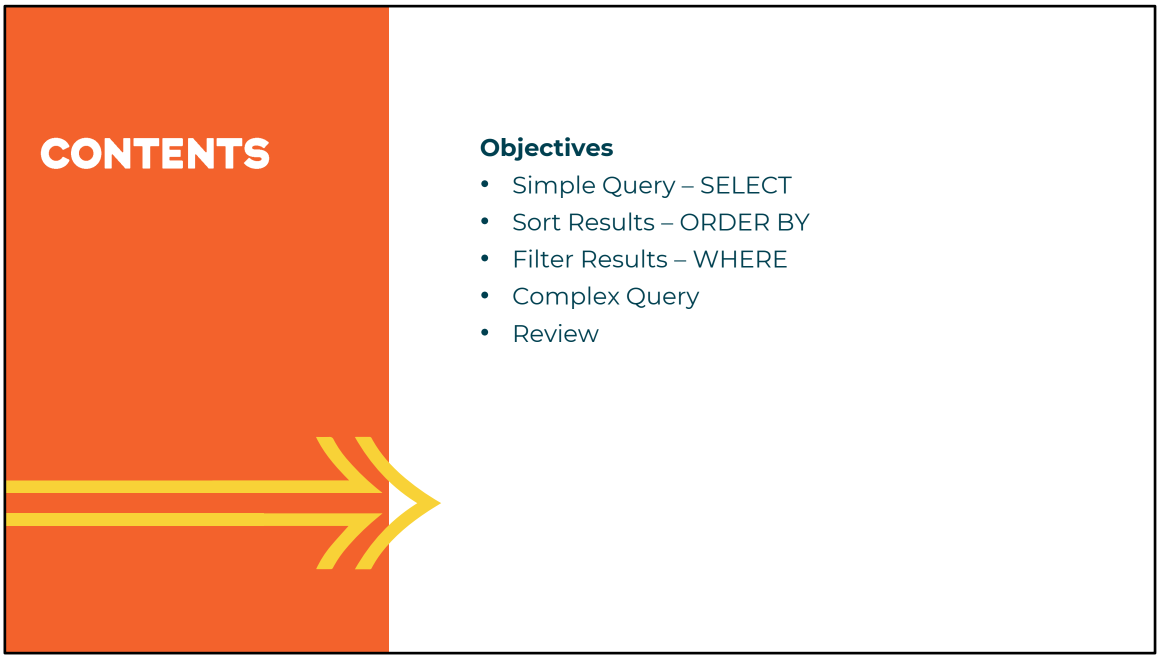
SQL Server

Displaying Data

This module will enable the learner to write SQL to display data from a single table. Specifically the learner will be able to:

- Create a query using the SELECT statement.
- Filter the results of a query using the WHERE clause.
- Sort the results of a query using the ORDER BY clause.
- Create complex queries using advanced SELECT options: LIMIT / TOP, UNION, CASE and NULL.

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.



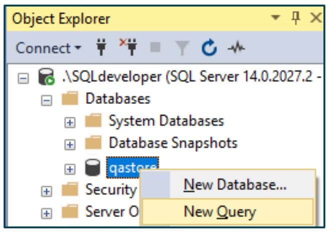
CONTENTS

This module provides an introduction to the SELECT command to fetch data from a database table and display a result set of records consisting of rows and columns. The topics covered are:

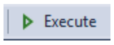
- Objectives – What are the Learning Outcomes and Assessment Criteria for the module.
- Simple Query – The basics of the SELECT statement to fetch data and display the results.
- Sort Records– The ORDER BY clause to sort the results of the query.
- Filter Records– The WHERE clause to filter the results of the query.
- Complex Query – Various options for more complex queries.
- Review – A short recap of the module including a quiz.

Your First SQL program – SQL Server Management Studio

- Create a query



```
-- SELECT all columns
SELECT * FROM company;
```

- Execute file by pressing F5 or 

```
company_no    company_name       tel            county   post_code
1000          Happy Heaters PLC  (01306)345672  London   SE3 89L
2000          Icicle Igloos Inc  0181-987-1265  London   N1 4LH
3000          Judo Jeans PLC     0171-478-2990  London   N9 2FG
4000          Kipper Kickers Inc 01254-987766   Devon    PL4 9RT
```

The simplest form of the SELECT statement is:

```
SELECT * FROM table
```

A simple SELECT like this will query and display data from every column in the table mentioned in the FROM clause. The '*' signifies that you want to display data from all of the columns in the table. Notice how powerful this is compared with a procedural programming language, where you would have to use a looping construct to achieve the same result. A danger however is the amount of data you can easily extract, with the resulting cost in performance if the table has many columns and records.

The columns are displayed horizontally in the order they are defined on the table. The SQL standard does not specify the order in which the records are displayed. This is implementation dependent and is typically driven by the order the data is physically saved on the file. This order will become more random as the data is maintained.

The GUI colours the SQL depending on its function. The defaults here are:

- Grey: Comments
- Blue: SQL keywords
- Black: User defined data such as tables and column names

Other you will see are:

- Green: Text
- Orange: Values
- Purple: Some functions

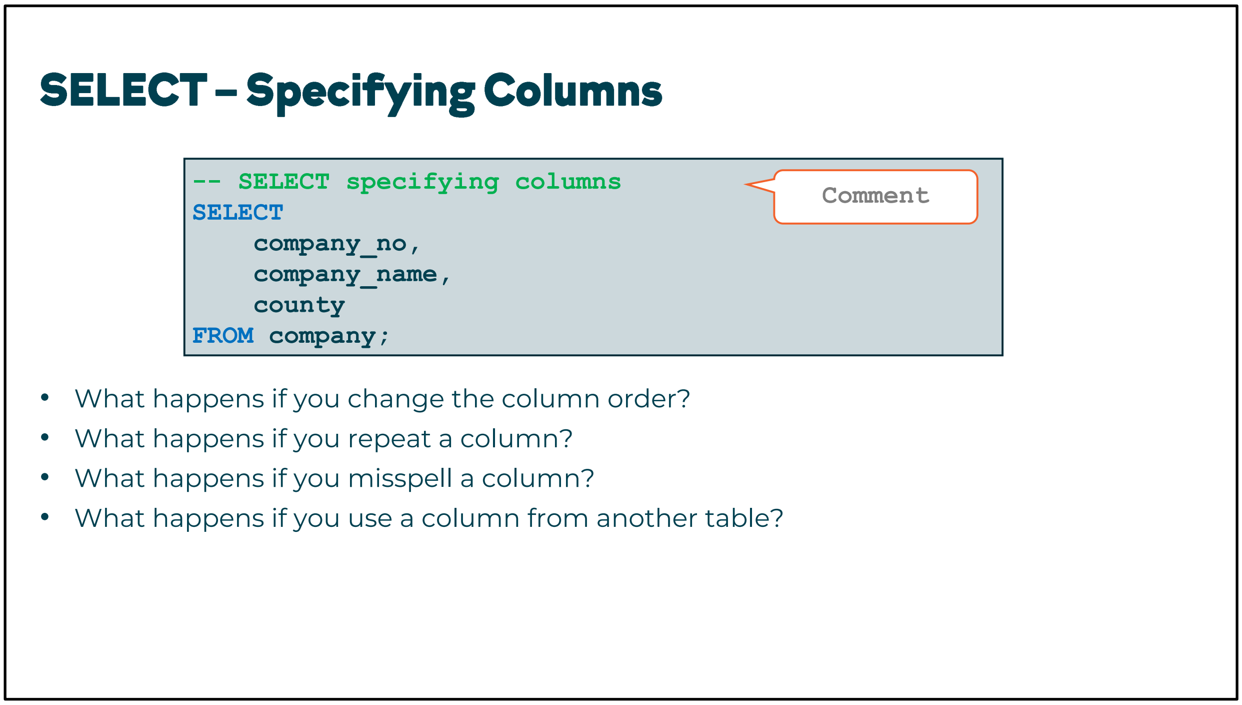
Comments:

There are three types of comments that can be used in SQL:

- A line comment starts with a ‘--’
- An in-line comment follows the SQL again starting from a ‘--’
- A block command starts with a ‘/*’, then the comments, and ends with a ‘*/’

All comments are ignored when the SQL is executed; as are blank lines and white space. SQL is a free format but syntactically fussy language. Good practice is to:

- Code SQL keywords in upper case and user defined data in lower case.
- Break the SQL into its constituent parts and split them over several lines.
- Use new lines, tabs and indents consistently to make it more readable.



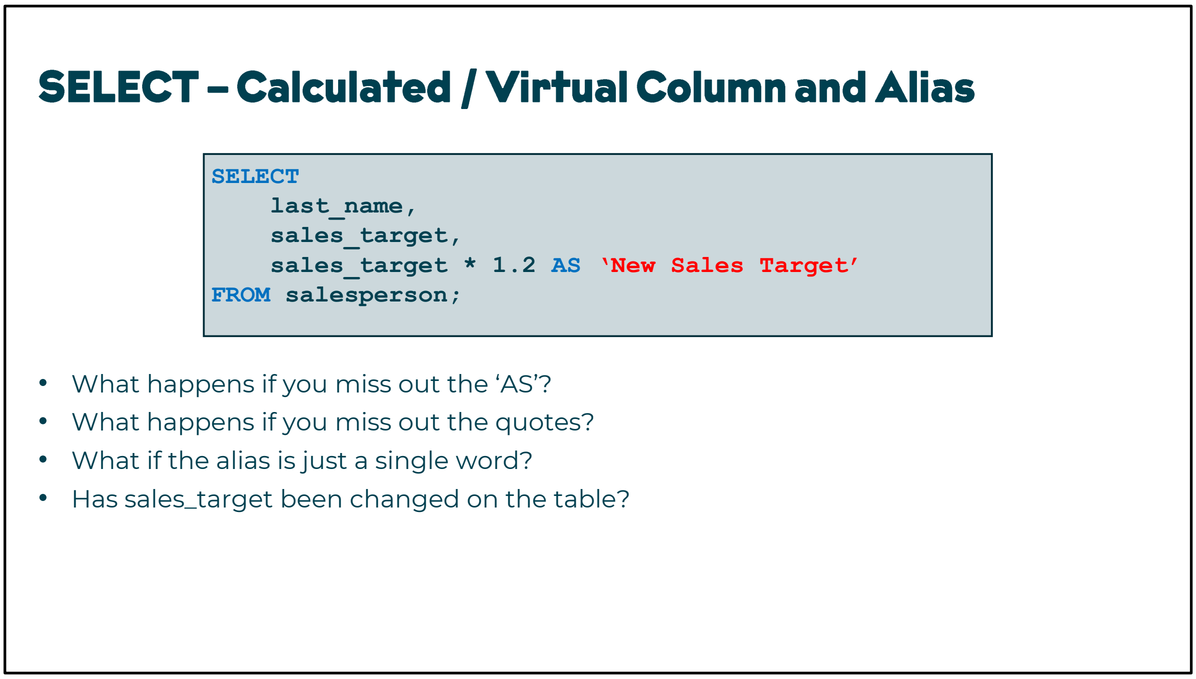
SELECT – Specifying Columns

```sql
-- SELECT specifying columns
SELECT
    company_no,
    company_name,
    county
FROM company;
```

Comment

- What happens if you change the column order?
- What happens if you repeat a column?
- What happens if you misspell a column?
- What happens if you use a column from another table?

The SELECT statement can be extended by replacing the * with a comma-separated list of column names. Only these columns are then displayed horizontally in the order specified.
One problem with this is that you need to know the names of the columns in the table. The * syntax avoids this, but for large tables it can result in vast amounts of data being displayed.
It is possible to repeat a column in a query. The benefit of which will become more apparent later in the module.
If you misspell a column name or include a column name from another table an unknown column error message will be generated when you execute the SQL.



SELECT – Calculated / Virtual Column and Alias

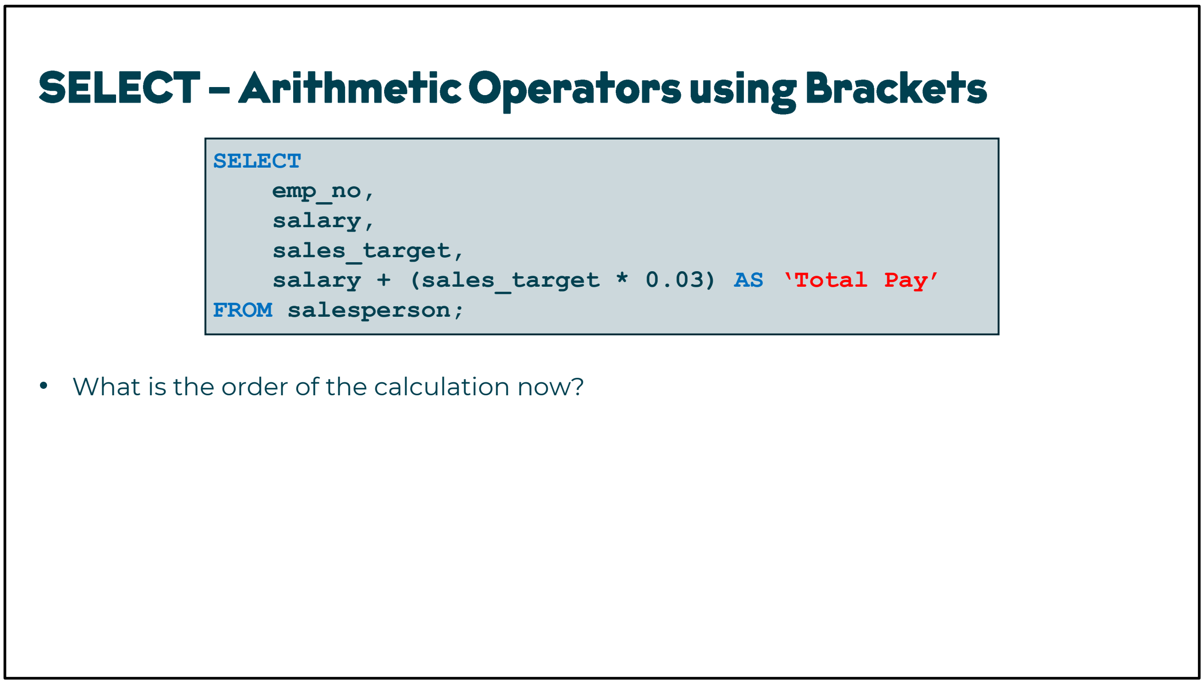
```
SELECT
    last_name,
    sales_target,
    sales_target * 1.2 AS 'New Sales Target'
FROM salesperson;
```

- What happens if you miss out the 'AS'?
- What happens if you miss out the quotes?
- What if the alias is just a single word?
- Has sales_target been changed on the table?

A 'virtual' column can be created by manipulating the existing columns using an expression.
In the above example we have created a 'virtual' column by multiplying the current sales target by 1.2. The result is displayed like a column from the table. Note that this column only exists during the life of the query; as soon as the query is complete the column disappears. This means that you can display 'virtual' columns but not insert or update values within them.
Note it is possible to create expressions using other arithmetic operations such as +, -, *, / and many more.
This column header or alias for the 'virtual' column is specified using the AS syntax. Most database manufacturers support column aliasing, but nearly all omit the use of the AS keyword as it is optional in SQL. This can lead to subtle bugs occurring in statements if a comma is accidentally omitted. For example, the statement below is syntactically correct but probably not what the author intended; it will display only the first name of each salesperson, but with the column heading 'lname':

```
SELECT fname lname FROM salesperson
```

Note that the default column header will be the column name for non-expression columns. This header may be also be renamed using the AS syntax shown above.

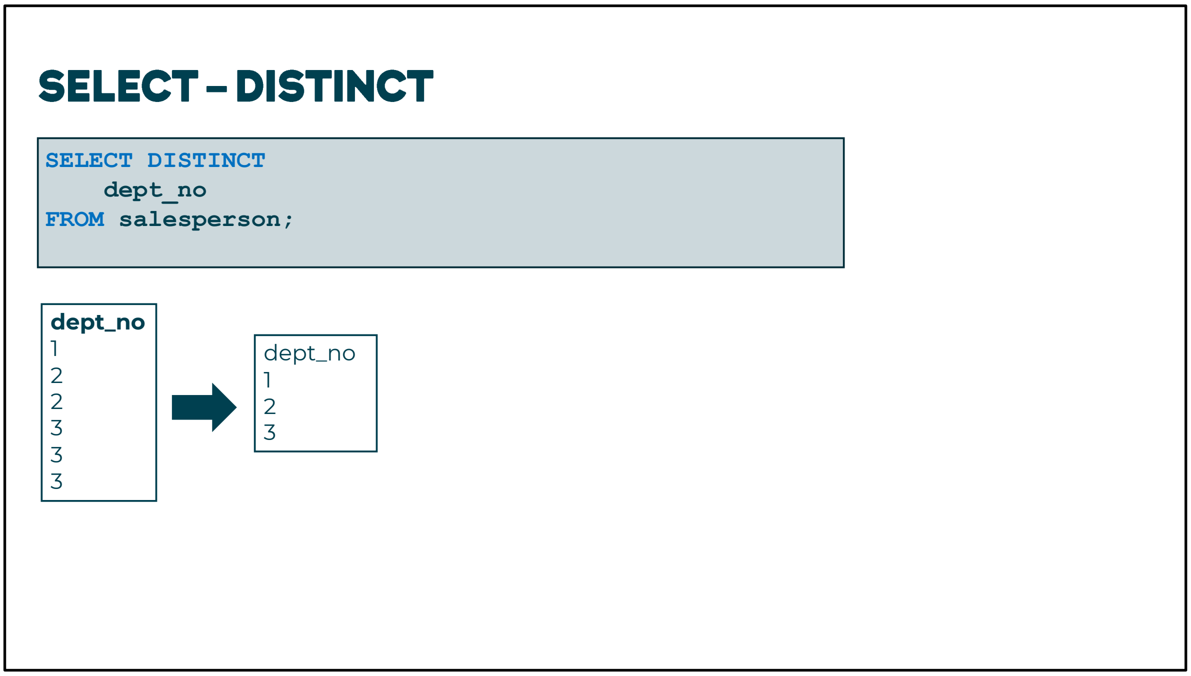


SELECT – Arithmetic Operators using Brackets

```
SELECT
    emp_no,
    salary,
    sales_target,
    salary + (sales_target * 0.03) AS ‘Total Pay’
FROM salesperson;
```

- What is the order of the calculation now?

An suggested in the previous slide the use of brackets makes it clear that ‘Total Pay’ is calculated as salary plus 0.03 (3%) of sales_target.



SELECT – DISTINCT

```
SELECT DISTINCT
    dept_no
FROM salesperson;
```

dept_no
1
2
2
3
3
3

dept_no
1
2
3

The default for SELECT is to display all the rows specified.
The modifying clause, DISTINCT, can be used the eliminate duplicate rows and return one copy of each.
Typically to eliminate duplicates the database engine must do a sort or use other techniques of removing duplicates that do not require a sort.
The elimination occurs across all the data. As above the SQL to just display a list of unique departments that employ a salesperson is:

```
            SELECT DISTINCT dept_no FROM salesperson
```

DISTINCT also works on multiple columns. To display unique combinations of departments and counties the SQL is:

```
            SELECT DISTINCT dept_no, county FROM
salesperson
```

The DISTINCT keyword appears only once and applies to the entire set of columns in the list.

SELECT with ORDER BY Ascending

```
SELECT *
FROM contact
ORDER BY company_no ASC,contact_code ASC;
```

- What happens if you omit the ASC?
- What happens is you replace company_no by 1?
- What happens is you replace contact_code by 2?
- What happens if you swop the ORDER BY columns?

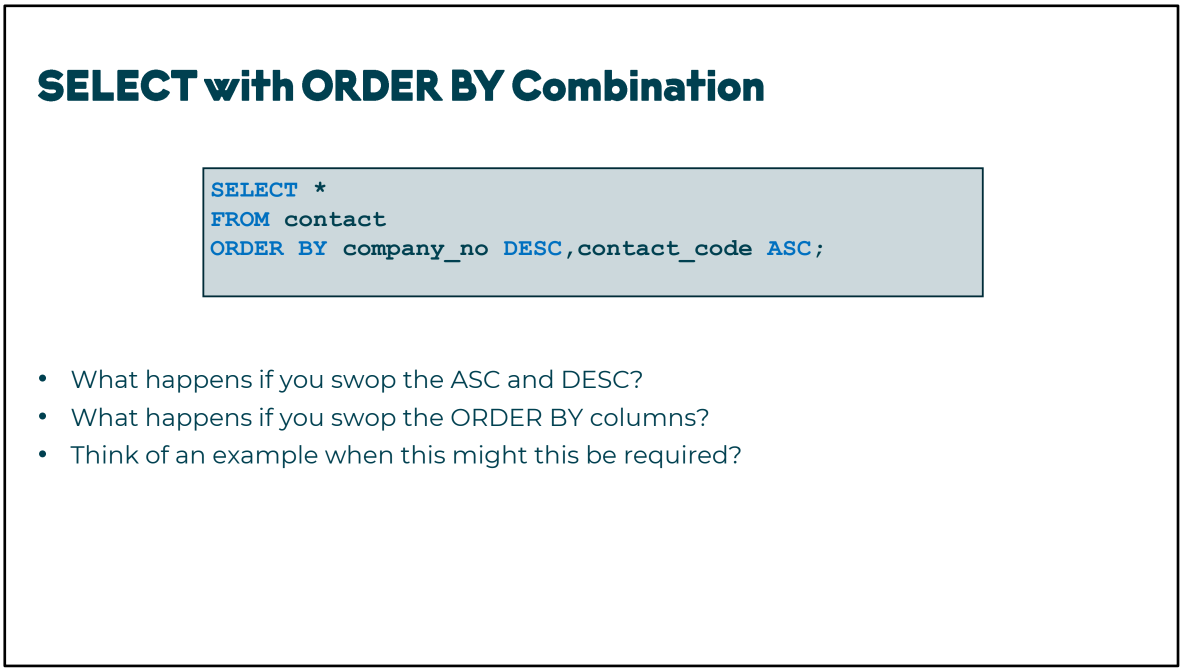
As noted previously the SQL standard does not specify the order in which results are displayed. This is implementation dependent and is typically driven by the order the data is physically saved on the file.
You can specify a sort order by using the ORDER BY clause. The column can be specified by its name or ordinal position in the column list. You may specify several columns and have the data sorted in ascending or descending order. An ascending sort can be specified by the option ASC after the sort column. However ascending is the default option and does not need to be specified.
If more than one column is specified in the ORDER BY clause the first column is the major sort and subsequent columns the minor sorts in the specified order.
In our example the data would be sorted by company_no first and then by contact_code for records with the same company_no. Swopping the columns the sort would be by contact_code first and then company_no.

SELECT with ORDER BY Descending

```
SELECT *
FROM contact
ORDER BY company_no DESC;
```

- What happens if you omit the DESC?
- What happens if you swop the ORDER BY columns?

The DESC option is used to sort records in descending order. This is placed after the corresponding sort column. As noted previous if this is omitted the default sort order is ascending.
As with the previous example swopping the sort columns will order the data by contact_code first and company_no second.



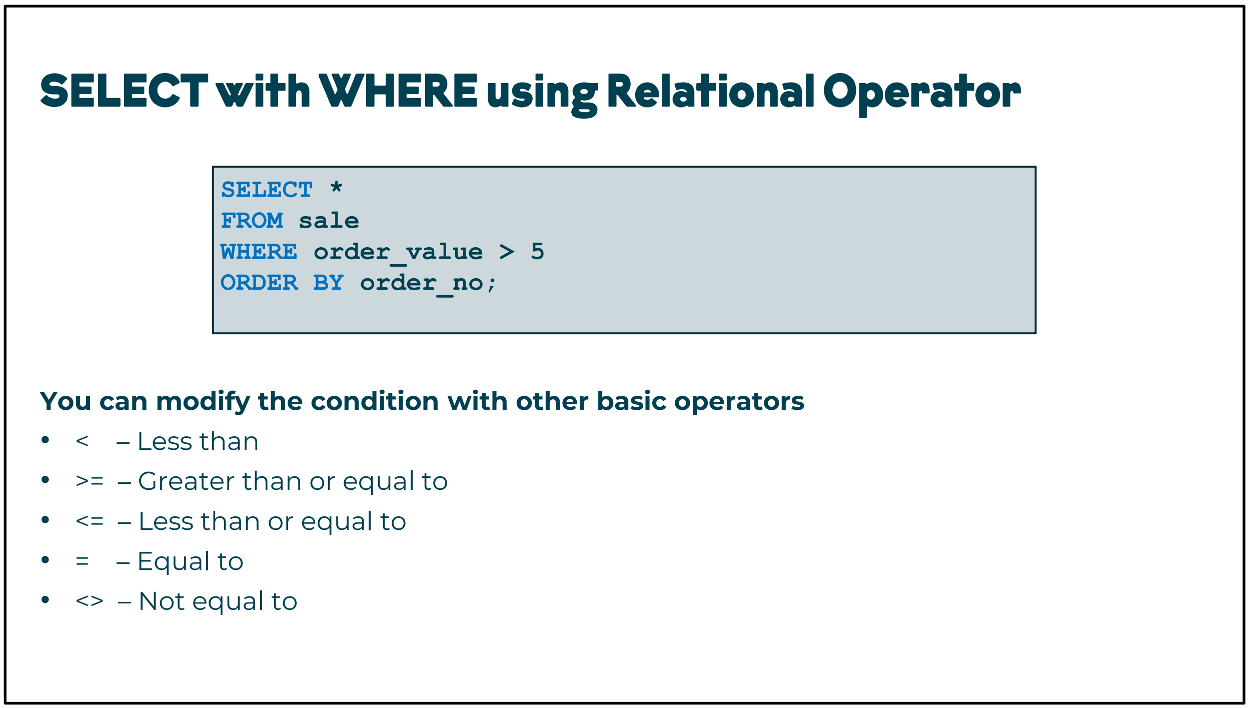
SELECT with ORDER BY Combination

```
SELECT *
FROM contact
ORDER BY company_no DESC,contact_code ASC;
```

- What happens if you swop the ASC and DESC?
- What happens if you swop the ORDER BY columns?
- Think of an example when this might this be required?

It is possible to sort by combinations of ascending and descending columns if required.
For example to display salespeople firstly by ascending department_no and then in descending order of salary from biggest to smallest the SQL is.

```
SELECT * FROM salesperson
ORDER BY department_no ASC, salary DESC
```



SELECT with WHERE using Relational Operator

```
SELECT *
FROM sale
WHERE order_value > 5
ORDER BY order_no;
```

You can modify the condition with other basic operators

- <   – Less than
- >= – Greater than or equal to
- <= – Less than or equal to
- =   – Equal to
- <> – Not equal to

A SELECT will display all the records on a table unless a WHERE clause is used to filter the selection. The WHERE clause is similar to an ‘IF’ statement in other programming languages. For example “Select this row IF the boolean_expression or condition is true”. The WHERE clause is used to limit the rows to just those that meet the required criteria.
There are six basic operators that may be used in a WHERE clause to compare values and to select that rows that meet the specified condition:

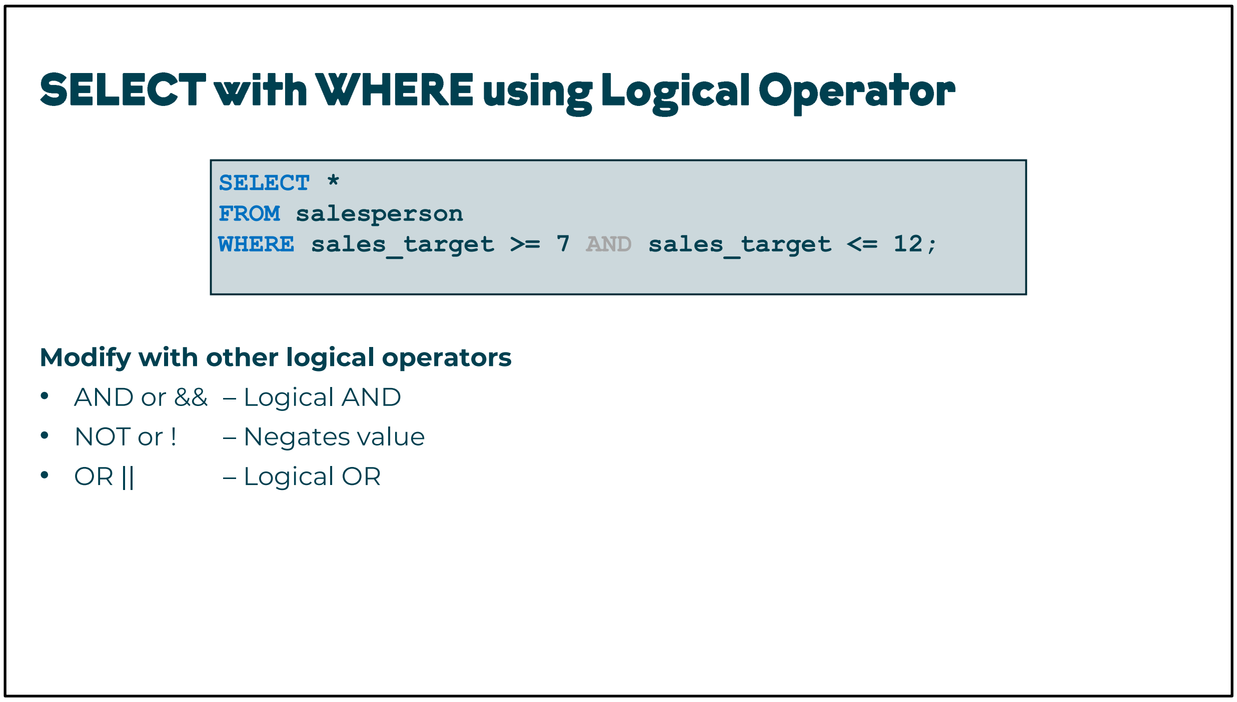
- =	Equal
- <>	Not Equal
- <	Less Than
- >	Greater Than
- <=	Less Than or Equal To
- >=	Greater Than or Equal To

The values tested may be columns, constants or expressions. In addition to these basic tests we can perform more complex checks as we will see in subsequent examples.

The boolean_expression or condition may contain a function for example:

```
SELECT *
FROM salesperson
WHERE SUBSTRING(lname,1,1) = ‘P’
```

This will select all salespersons whose surname starts with a “P”.



SELECT with WHERE using Logical Operator

```
SELECT *
FROM salesperson
WHERE sales_target >= 7 AND sales_target <= 12;
```

Modify with other logical operators

- AND or && – Logical AND
- NOT or ! – Negates value
- OR || – Logical OR

Multiple conditions can be specified in the WHERE clause. These are linked using the logical operators: AND, OR and NOT.
Generally when you join conditions with an AND this will result in less rows qualifying for selection. You are saying “This must be true and also that must be true” for the record to be selected.
Alternatively when you join conditions using an OR this will result in more rows qualifying for selection. You are saying “Either this may be true or that may be true or both” for the record to be selected.
The NOT operator will select the records that do not meet the specified condition.
You can link as many conditions as you like. Care must be taken when combining conditions that they select what is actually required. Similar to the BODMAS rule there is a precedence of operators: ANDs will be evaluated first before ORs. Good practice is to use brackets to specify and clearly illustrate the logic required.

SELECT with WHERE using BETWEEN

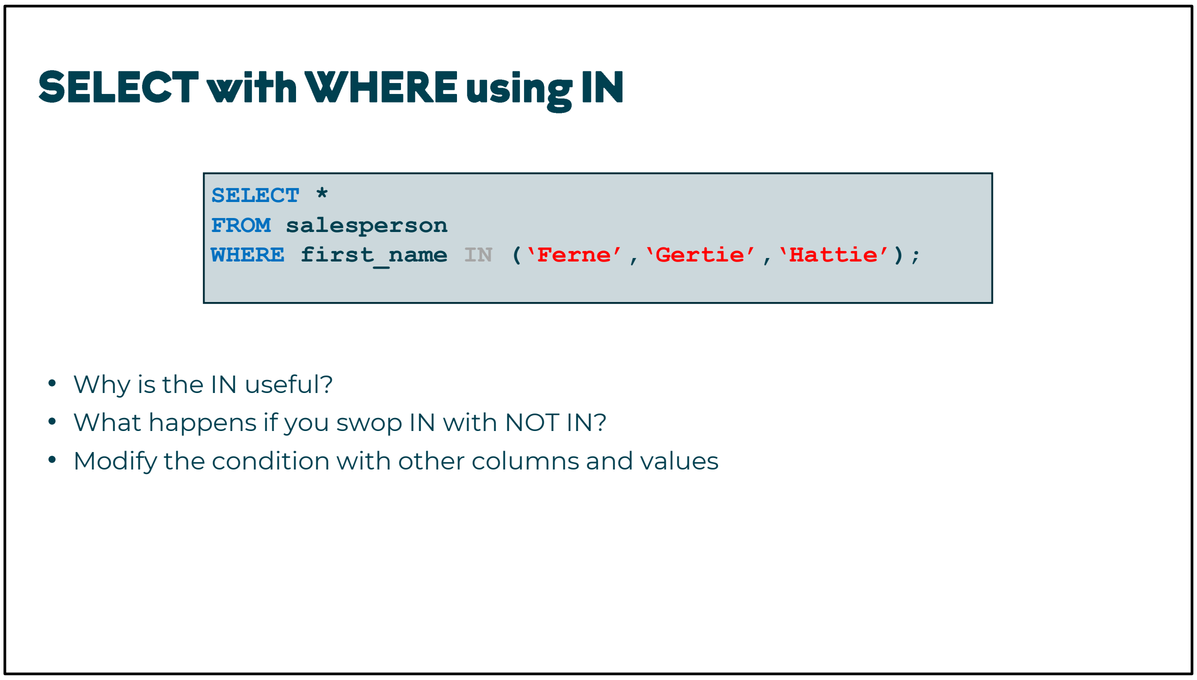
```
SELECT *
FROM salesperson
WHERE sales_target BETWEEN 7 AND 12;
```

- Are the values inclusive?
- Modify the condition with other columns and values
- What happens if you specify a higher starting value?
- What happens if you write: NOT BETWEEN 7 AND 12

The BETWEEN option is used to specify a range of values between, and including, two limits.
The first value sets the start of the range; the second value sets the end. Note that the AND keyword between the two values is mandatory.
Also that the SQL Standard expects the lower end of the range to be specified first, otherwise no rows will be returned.
The SQL could be coded as follows:

```
SELECT *
FROM salesperson
WHERE salestarget >= 7 AND salestarget <=12;
```

BETWEEN is a shorter and neater way of coding the logic and ensure that that range values are included in the selection if that is what is required.



SELECT with WHERE using IN

```
SELECT *
FROM salesperson
WHERE first_name IN ('Ferne','Gertie','Hattie');
```

- Why is the IN useful?
- What happens if you swop IN with NOT IN?
- Modify the condition with other columns and values

The IN option is used to specify a list, or a set of values to test against. This can simplify tests where the result may take one of several values. Using IN is easier to read and maintain. For example compare the above with:

```
SELECT *
FROM salesperson
WHERE fname = 'Ferne'
OR fname = 'Gertie'
OR fname = 'Hattie';
```

Note that the following syntax, which is a common mistake, is invalid:

```
WHERE fname = 'Ferne' OR 'Gertie' OR
'Hattie';
```

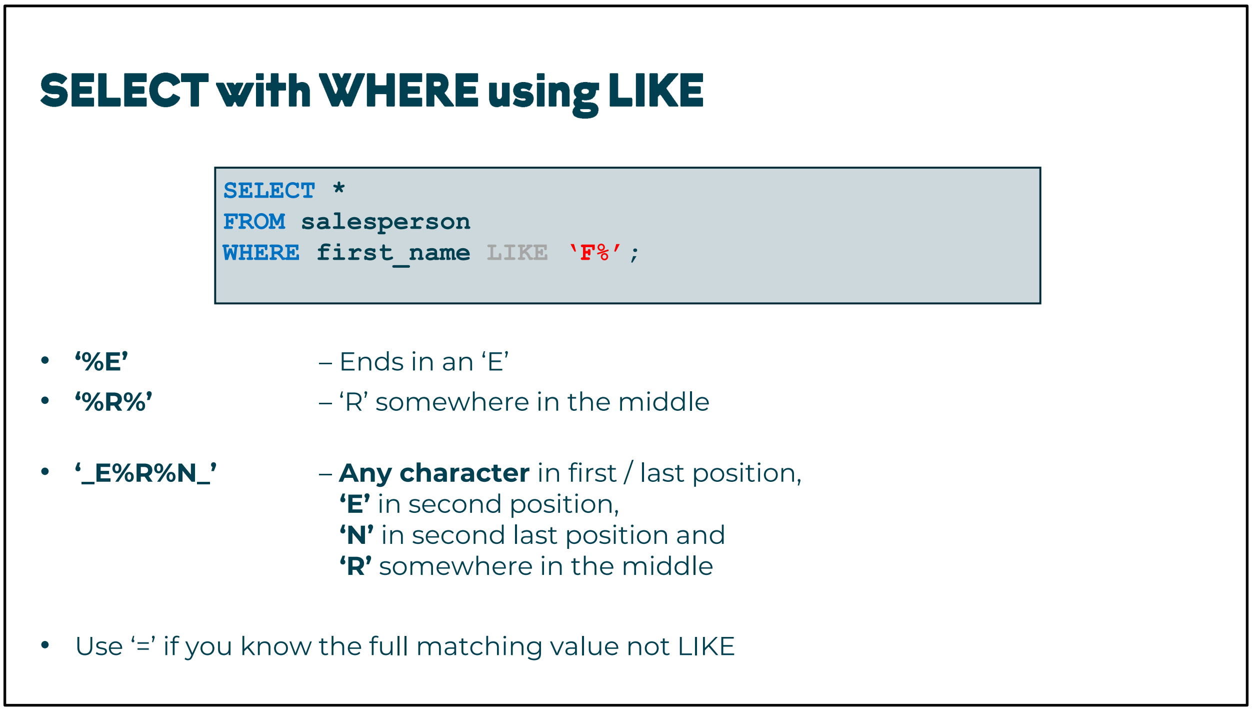
It is most unusual for the database engine to be 'case sensitive', Oracle being the notable exception.
If it is, use this syntax to find any combination of upper and lower case characters such as 'gertie', 'Gertie', 'GERTIE' or even 'GeRtIe':

```
WHERE UPPER(fname) = 'GERTIE';
```

or

```
WHERE LOWER(fname) = 'gertie';
```

The NOT option is a modifier that may be used to reverse the boolean outcome of a test. It selects the rows that do NOT meet the test criteria. Again this may be the quicker, clearer and only way to specify the actual condition required.



SELECT with WHERE using LIKE

```
SELECT *
FROM salesperson
WHERE first_name LIKE 'F%';
```

• **'%E'**	– Ends in an 'E'
• **'%R%'**	– 'R' somewhere in the middle
• **'_E%R%N_'**	– **Any character** in first / last position, **'E'** in second position, **'N'** in second last position and **'R'** somewhere in the middle

- Use '=' if you know the full matching value not LIKE

Normal comparison tests, =, <, >, <>, etc. on strings must match the tested string exactly. The LIKE option allows us to use wildcard searches or partial matches on the strings. Note you should always use = if you know the full matching value not LIKE.
There are two command characters, underscore (_) and percent (%).

- Underscore is used to match any single character.
- Percentage is used to match any number of characters, including no characters at all.

Some systems, for example MS SQL Server, allow further comparisons with the LIKE option for example:

- [ ]		Any single character within the specified range [a-f] or set [abcdef].
- [^] Any single character not within the specified range [^a-f] or set [^abcdef].

In some instances we wish to find a string that actually contains one of the wildcard characters. For example we want to find a company whose name includes the string '100% Computers'. This is when the escape character comes in handy. We can use the escape character, "\", to enable us to find the occurrence of the command character within the string.

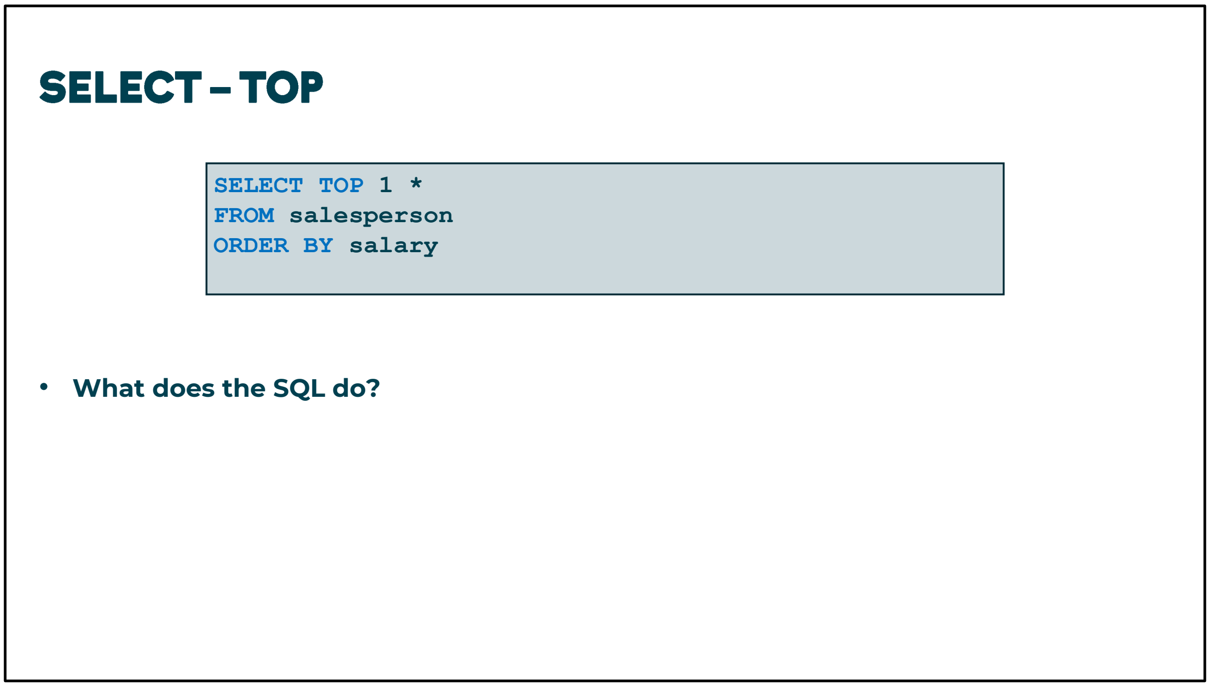
```
SELECT   *
FROM     company
WHERE name LIKE '%100\% Computers%'
ESCAPE   '\'
```

SELECT with WHERE Multiple AND / OR

```
SELECT *
FROM salesperson
WHERE county = 'Hampshire'
OR dept_no = 3
AND first_name = 'Karena';
```

- Does the AND or OR take precedence?
- Modify with brackets to alter the logic and results

As noted earlier multiple conditions can be specified in a WHERE clause. Care must be taken when combining these conditions to ensure they select what is actually required. Similar to the BODMAS rule, ANDs take precedence over ORs. Good practice is to use brackets to specify and clearly illustrate the logic required.

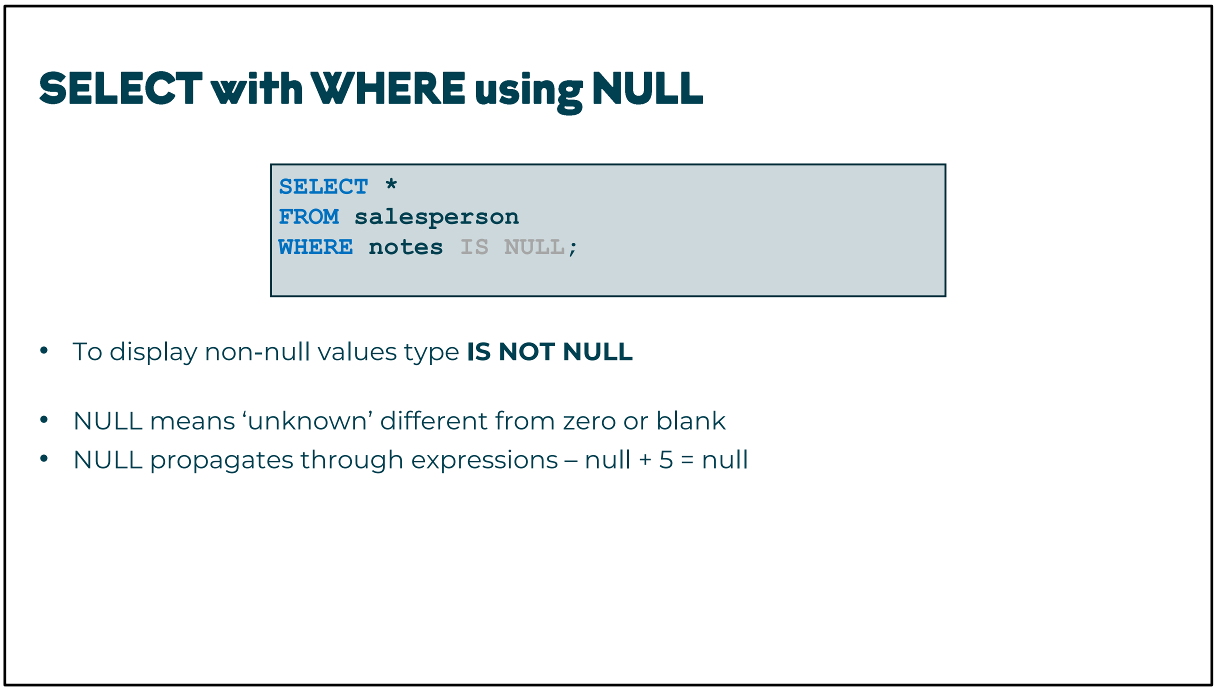


SELECT – TOP

```
SELECT TOP 1 *
FROM salesperson
ORDER BY salary
```

- **What does the SQL do?**

There may be a requirement to only display a few of the possible rows returned, especially with a sorted query.
The TOP option enables us to answer questions like “what are our 10 most expensive products?” We can sort our results by price in a descending order and then use TOP to select to just the top 10 results.
If there are several records that match the last value selected by the TOP option they can all be displayed by adding the WITH TIES. For example if we have several product that match the tenth most expensive price then all will be displayed.
Also there is the TOP n PERCENT option that will display the n% of records for the given selection and sorted order.
Again the WITH TIES option will display any duplicates of the final value.



SELECT with WHERE using NULL

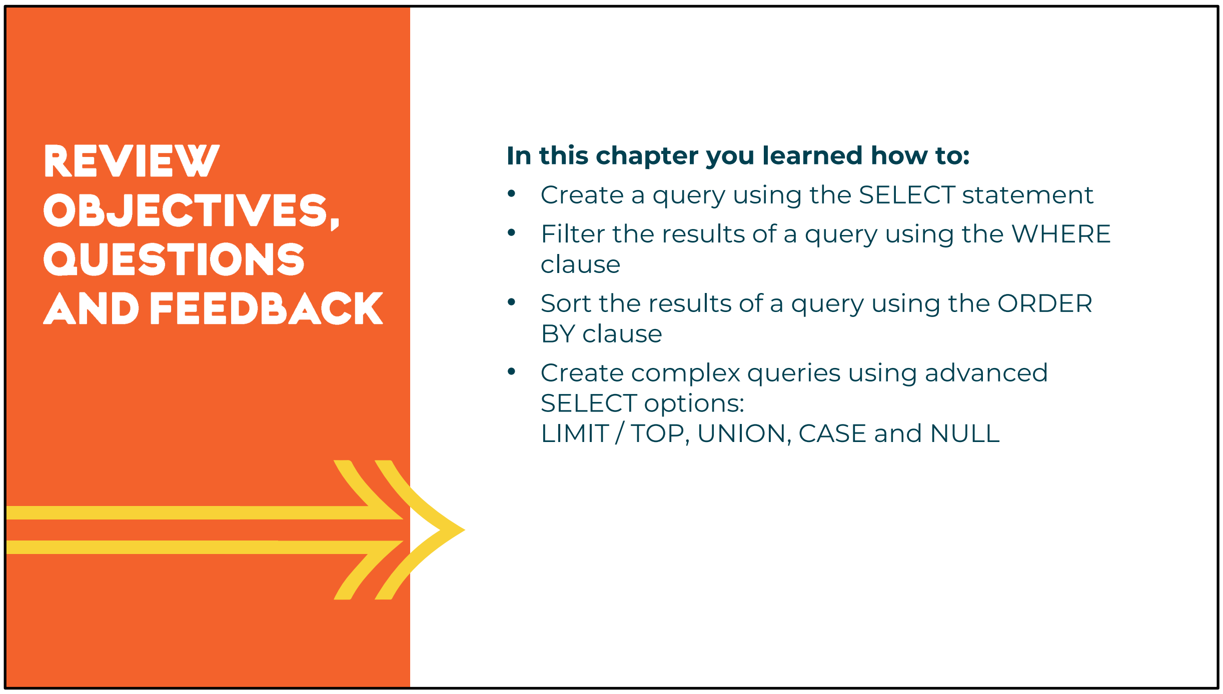
```
SELECT *
FROM salesperson
WHERE notes IS NULL;
```

- To display non-null values type **IS NOT NULL**
- NULL means ‘unknown’ different from zero or blank
- NULL propagates through expressions – null + 5 = null

A NULL value is stored where no data has been entered for given column on a record. It represents an unknown value and is different from a zero in a numeric field and a blank in a text field.
A NULL will propagate through an expression. If you add a value to a NULL value the answer will still be NULL.
To select a NULL value the IS NULL option is used in the WHERE clause. Likewise IS NOT NULL is used to select values that are not-NULL.
If the WHERE clause specifies a condition to select records that are not equal to a specified value it will only display the non-NULL values that are not equal. To also select the NULL values will require an extra condition as follows:

```
SELECT *
FROM salesperson
WHERE notes <> ‘Joined in Sept 1996’ OR notes IS
NULL;
```

This is often referred to as 3-way logic.



REVIEW OBJECTIVES, QUESTIONS AND FEEDBACK

In this chapter you learned how to:

- Create a query using the SELECT statement
- Filter the results of a query using the WHERE clause
- Sort the results of a query using the ORDER BY clause
- Create complex queries using advanced SELECT options:
  LIMIT / TOP, UNION, CASE and NULL

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

- 2. Write SQL to display data from a single table.

Also the Assessment Criteria? As a learner can you?

- 2.1 Create a query using the SELECT statement.
- 2.2 Filter the results of a query using the WHERE clause.
- 2.3 Sort the results of a query using the ORDER BY clause.
- 2.4 Create complex queries using advanced SELECT options: LIMIT / TOP, UNION, CASE and NULL.

Do you have any further questions on the content covered or feedback on the module?

SELECT – CASE using Value

```
-- SELECT using CASE value
SELECT
    last_name,
    dept_no,
    CASE dept_no
        WHEN 1 THEN ‘Dept 1’
        WHEN 2 THEN ‘Dept 2’
        ELSE 'Dept unknown‘
    END AS ‘Department’
FROM salesperson
ORDER BY last_name;
```

- **What does the SQL do?**

The CASE option can be used to replace values that are found in the result set with other values. In the example above the dept_no in the table salesperson will be output as its corresponding department name. If a dept_no does not equate to one of the specified values the ELSE clause acts as a catch all and is output as ‘Dept unknown’.

SELECT – CASE using Condition

```
-- SELECT using CASE condition
SELECT
    last_name,
    sales_target,
    CASE
        WHEN sales_target <= 9  THEN 'Low'
        WHEN sales_target <= 12 THEN 'Medium'
        ELSE 'High'
    END AS 'Rating'
FROM salesperson
ORDER BY last_name,sales_target;
```

- **What does the SQL do?**
  - Modify the CASE syntax with other columns and values

It is also possible to specify the CASE statement using conditions. The first condition in the order specified that is true will be the one selected. Take care to specify your conditions in the required order. Again if no specified condition is true a default option is output using the ELSE clause.

POWERING POTENTIAL

Thank you