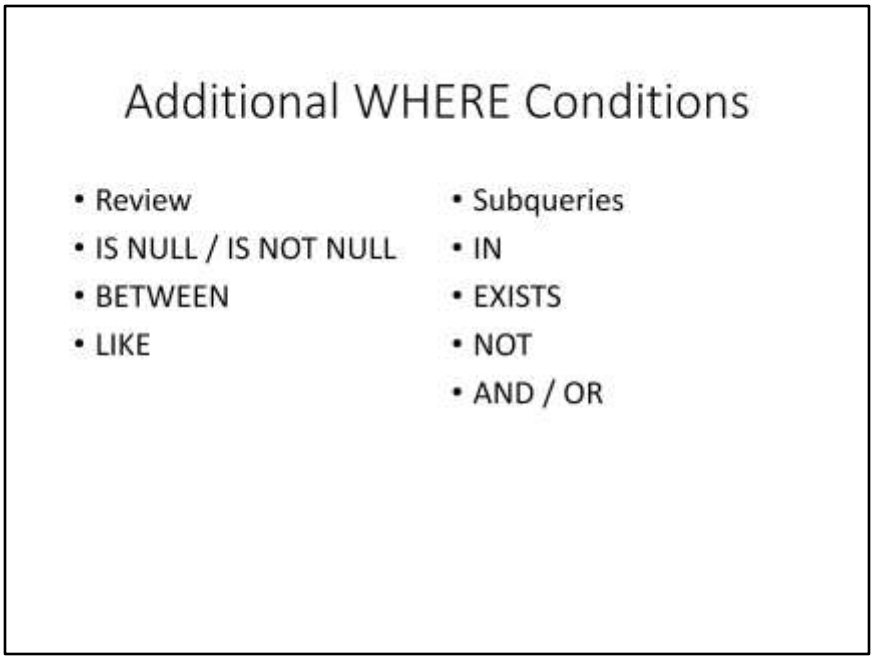
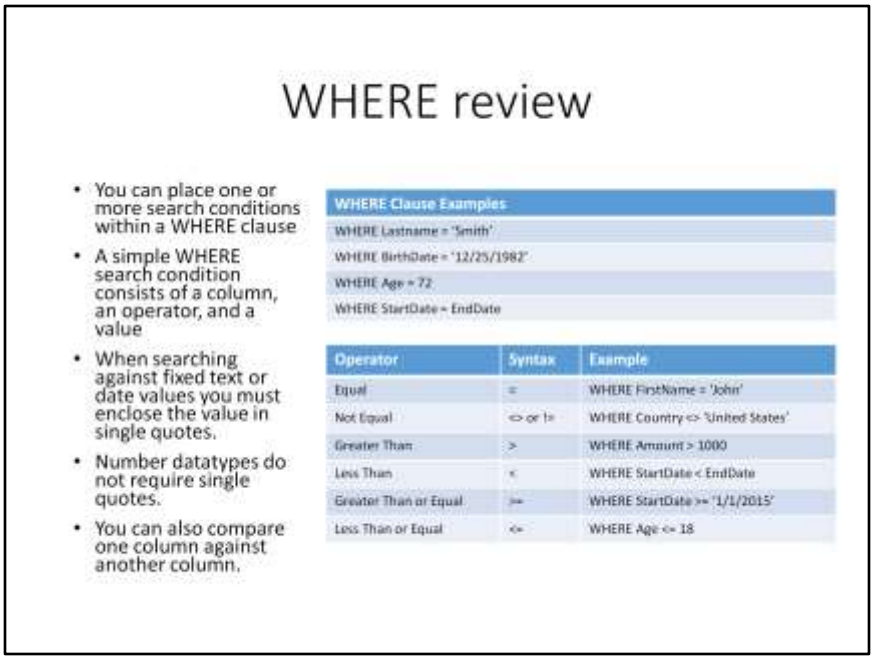


This presentation will be on additional WHERE clause conditions. I will talk about searching on null values; conditions like BETWEEN, IN, and EXISTS; subqueries; and joining multiple conditions in a single statement.



To review from a previous lesson, the where clause is used to filter the rows returned in a record set. The syntax of a where clause consists of the column on which you're searching, the operator such as an equal sign, and the value for which you are searching. When searching on text or date related columns, you must enclose the value within single quotes. Single quotes are not required when searching columns based on a numeric datatype. It is possible to compare two columns to one another within a where clause.

IS [NOT] NULL Keywords

- The IS NULL keywords are used to check if a column field has a Null value
- The IS NOT NULL keywords only return records that do not have a Null value
- Using an equal (=) or not equal (<>) sign is not valid syntax for a Null search in the WHERE clause

```
SELECT *
FROM Customer
WHERE Company IS NULL
```

	CustomerId	FirstName	LastName	Company
1	2	Leonie	Köhler	NULL
2	3	François	Tremblay	NULL
3	4	Bjørn	Hansen	NULL
4	5	Helena	Holý	NULL
5	7	Astrid	Gruber	NULL


```
SELECT *
FROM Customer
WHERE Company IS NOT NULL
```

	CustomerId	FirstName	LastName	Company
1	1	Luís	Gonçalves	Embraer - Empresa
2	5	František	Wichterlová	JetBrains s.r.o
3	10	Eduardo	Martins	Woodstock Discos
4	11	Alexandre	Rocha	Banco do Brasil S.A.
5	12	Roberto	Almeida	Riotur
6	14	Mark	Philips	Telus
7	15	Jennifer	Peterson	Rogers Canada

A null value represents undefined data and is treated differently values that may only contain zero or more spaces. The IS NULL condition is used to search for null values with a column, while the IS NOT NULL condition only returns records that are not null. You cannot use operators such as equal to or greater than to search for null values.

Searching Columns with Null Values

- Not equal, greater than, and less than operators will not include records with Null values in the result set
- If you wish to include Nulls you need to explicitly search for them

```
SELECT *
FROM Customer
WHERE Company != 'Telus'
```

	CustomerId	FirstName	LastName	Company
1	1	Luís	Gonçalves	Embraer - Empresa B
2	5	František	Wichterová	JetBrains s.r.o.
3	10	Eduardo	Martins	Woodstock Discos
4	11	Alexandre	Rocha	Banco do Brasil S.A.
5	12	Roberto	Almeida	Riotur
6	15	Jennifer	Peterson	Rogers Canada
7	16	Frank	Harris	Google Inc.

```
SELECT *
FROM Customer
WHERE Company != 'Telus'
OR Company IS NULL
```

	CustomerId	FirstName	LastName	Company
1	1	Luís	Gonçalves	Embraer - Empresa B
2	2	Leonie	Köhler	NULL
3	3	François	Tremblay	NULL
4	4	Bjørn	Hansen	NULL
5	5	František	Wichterová	JetBrains s.r.o.
6	6	Helena	Holý	NULL
7	7	Astrid	Gruber	NULL
8	8	Daan	Peeters	NULL
9	9	Kara	Nielsen	NULL
10	10	Eduardo	Martins	Woodstock Discos

Special care needs to be taken when filtering on columns with null values. Null values are not included in the result set of operators such as “not equal to”, “greater than”, or “less than”. You need to include the IS NULL condition in your WHERE clause to capture records with NULL values. In the example I am searching the Company column for names that are not equal to “Telus”. However null values are not included. To include null values I need to add the second line “OR Company IS NULL” to the WHERE clause.

The BETWEEN Keyword

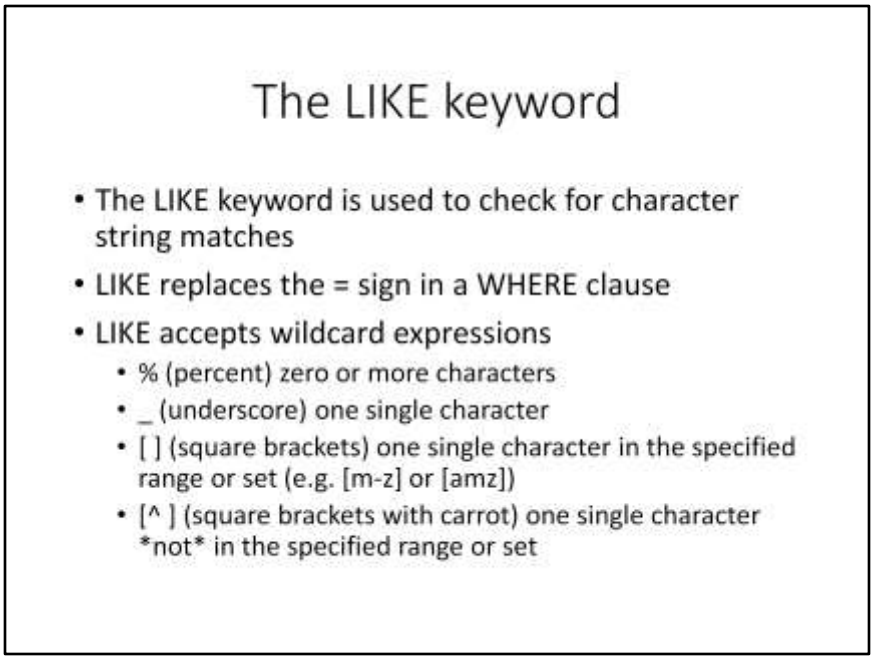
- Use the BETWEEN keyword to specify a range to search
- Place the AND keyword between the low and high values
- The low and high values will be included in the result set
- BETWEEN works on alphanumeric and date datatypes

```
SELECT
    CustomerId
    ,InvoiceDate
    ,BillingCity
    ,BillingCountry
FROM Invoice
WHERE InvoiceDate BETWEEN '2010-01-08' AND '2010-01-26'
```

Results | Messages

	CustomerId	InvoiceDate	BillingCity	BillingCountry
1	43	2010-01-08 00:00:00.000	Dijon	France
2	45	2010-01-08 00:00:00.000	Budapest	Hungary
3	47	2010-01-09 00:00:00.000	Rome	Italy
4	51	2010-01-10 00:00:00.000	Stockholm	Sweden
5	57	2010-01-13 00:00:00.000	Santiago	Chile
6	7	2010-01-18 00:00:00.000	Vienne	Austria
7	21	2010-01-26 00:00:00.000	Reno	USA

The BETWEEN condition is used to search for values within a specified range in a column. BETWEEN takes two values separated by the AND keyword. One value is the low range and the other is the high range. Both values will be included in the result set. The range datatypes can be character, numeric, or date-time. In the example I am searching for dates between January 8th and January 26th 2010. The 8th and the 26th are included in the result set.



The LIKE keyword is used to search for character string matches. It differs from the equal operator in that it can accept one or more wild card characters. The percent sign tells SQL to search for zero or more characters of any type. The underscore character searches for one and only one character of any type. Square brackets are used to define a set or range of characters for a single character slot. Ranges are represented by a low and high value separated by a dash, while sets are simple individual characters typed one after the other between the brackets. Only results that return a value in the range or set are returned. If you enter a carrot symbol after the first bracket, this has the effect of excluding the following characters from the result set.

LIKE Example Percent and Underscore

- The first example uses the % wildcard at front and end of the character string
- The second example has 2 underscore (_)wildcards at the front and the % wildcard at the end

In the first example the LIKE condition is using two percent signs, one before and one after the word “gmail”. This will return all records where “gmail” appears anywhere within the Email field. The second example has two underscores followed by the letter “A”, which is followed by a percent sign. This has the effect of returning only those fields in the Country column that have the letter “A in the 3rd character slot. Note that I surrounded both expressions with single quotes.

LIKE Example Character Range

- In example 1 the first character slot must be in the specified range of A through D
- In example 2 the first character slot can be anything other than the specified range of A through D

```
SELECT
    FirstName
    ,LastName
FROM Customer
WHERE LastName LIKE '[A-D]%'
ORDER BY LastName
```

	FirstName	LastName
1	Roberto	Almeida
2	Julie	Barnett
3	Camille	Bernard
4	Michelle	Brooks
5	Robert	Brown
6	Kathy	Chase
7	Richard	Cunningham
8	Marc	Dubois

```
SELECT
    FirstName
    ,LastName
FROM Customer
WHERE LastName LIKE '[^A-D]%'
ORDER BY LastName
```

	FirstName	LastName
1	João	Fernandes
2	Edward	Francis
3	Wyatt	Girard
4	Luís	Gonçalves
5	John	Gordon
6	Tim	Goyer
7	Patrick	Gray
8	Astrid	Gruber

In the first example I am using the brackets to represent a character range for the first character slot of LastName. Only those fields that begin with the letters A, B, C, or D will be returned in the result set. The second example is identical to the first except that I added the carrot symbol to the LIKE condition. This has the effect of returning only those fields that do *not* start with the letters A, B, C, or D.

LIKE Example Character Set

- In example 1 the first character slot must be in the specified set of A,B or Z
- IN example 2 the first character slot can be anything other than the specified set of A,B or Z

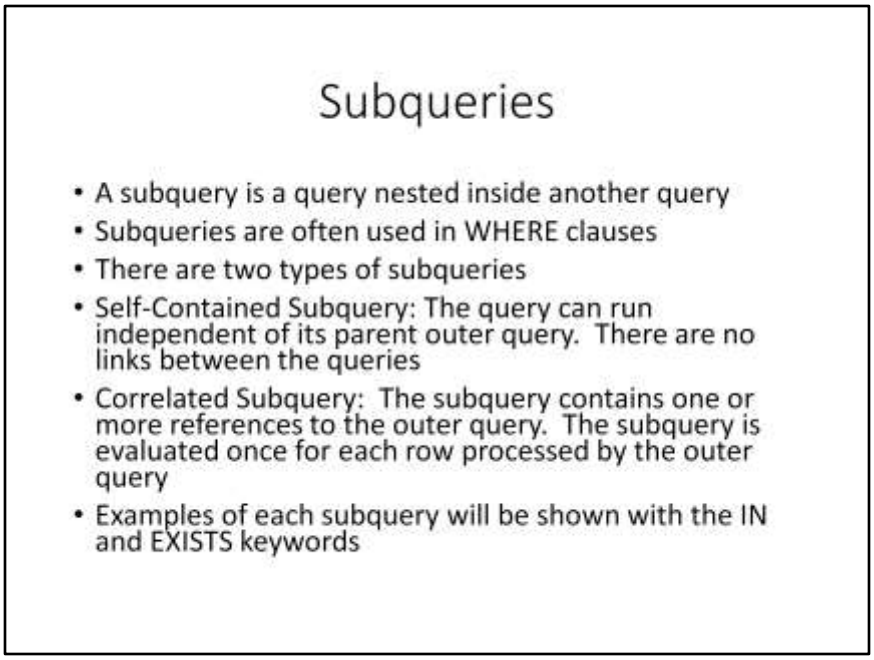
```
SELECT
    FirstName
    ,LastName
FROM Customer
WHERE LastName LIKE '[ABZ]%'
ORDER BY LastName
```

	FirstName	LastName
1	Roberto	Almeida
2	Julia	Barnett
3	Camille	Bernard
4	Michelle	Brooks
5	Robert	Brown
6	Fynn	Zimmermann

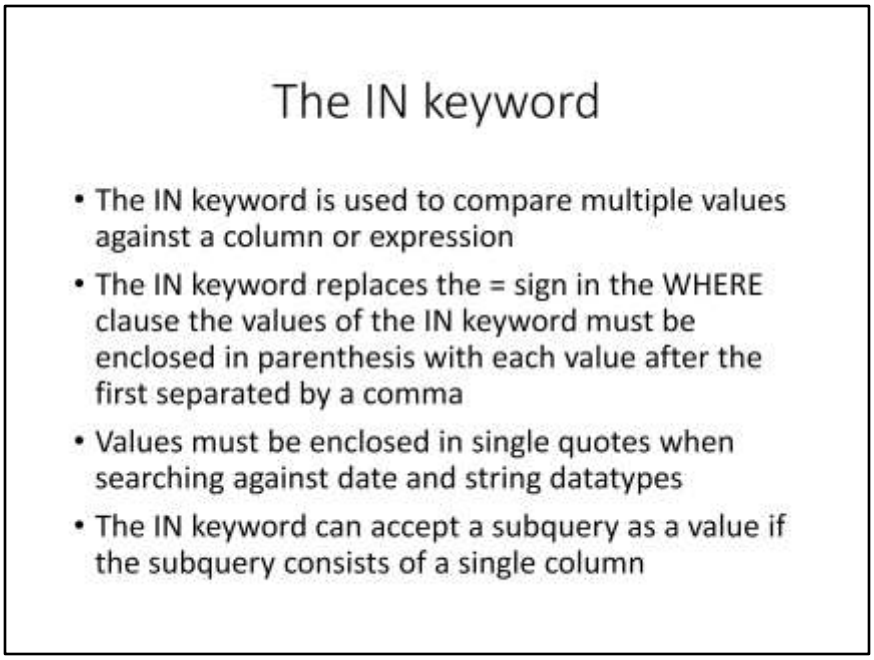
```
SELECT
    FirstName
    ,LastName
FROM Customer
WHERE LastName LIKE '[^ABZ]%'
ORDER BY LastName
```

	FirstName	LastName
1	Kathy	Chase
2	Richard	Cunningham
3	Marc	Dubois
4	Jodo	Fernandes
5	Edouard	Francois
6	Wyatt	Grant
7	Luis	Gonçalves
8	John	Gordon

In these examples I am using the brackets with a character set to filter data. The first example will only return records whose LastName field starts with the letters A, B or Z. The second example uses the carrot symbol so only records whose LastName field does *not* start with A, B or Z will be returned.



In order to make full use of the upcoming WHERE conditions it is necessary to understand the concept of a subquery. A subquery is essentially one query that is nested inside of another. There are two types of subqueries, a self-contained subquery, and a correlated subquery. A self-contained query is a query that does not reference the parent query in any way. You can run a self-contained subquery on its own without receiving an error. A correlated subquery on the other hand does have a reference to the outer query. In practice this usually means that there is a WHERE condition in the subquery that checks against the value of a column in the parent query. We will see examples of both types of subqueries later in this presentation.



The IN keyword is used in a WHERE clause when it is necessary to compare a column against more than one value. You place the values you wish to search against inside a set of parenthesis with each value separated by a comma. The values must be enclosed in single quotes when searching against a non-numeric datatype. The IN keyword can use a subquery instead of a value set provided that there is only one column listed in the SELECT clause.

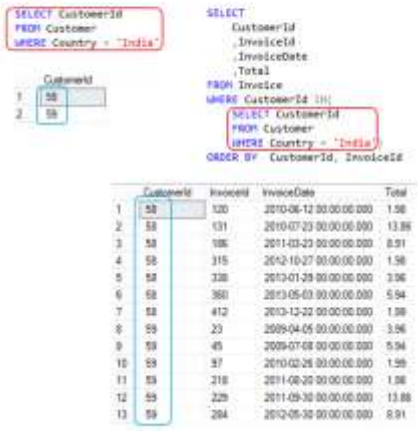
IN Example Character and Number Values

- In example one only the country values in the IN clause are returned in the record set
- In example two only the numbers are returned. Note that the numbers do not require single quotes

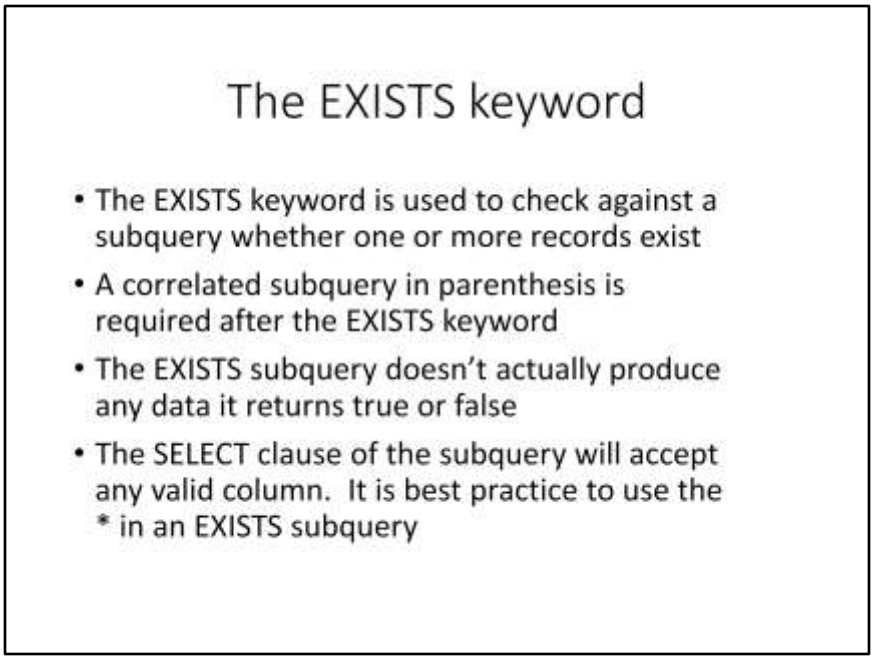
In the first example we only want to return records whose Country value equals Brazil, United Kingdom, or Sweden. Note the single quotes around each value. In the second example we are searching for a series of Album IDs. Because the AlbumId column is numeric, the values do not need to be enclosed in quotes. Keep in mind that it is not possible to use wildcards with an IN statement.

IN Example with a Subquery

- The subquery returns a single column with 2 values
- Those 2 values are used in the IN clause when the parent query is executed
- The parent query only returns records that contain the values in the subquery



In this example we are using an IN statement with a self-contained subquery. I have broken out and displayed the result set that is generated by the subquery. It returns values of 58 and 59. The IN statement uses the subquery to create the values on which it will filter. As you can see in the final result set, only records with a Customer ID of 58 or 59 were returned.



The EXISTS keyword is used to check the existence of a record against a correlated subquery. The subquery must be enclosed in parenthesis, and it must reference the column you wish to filter against in the outer query. An Exists subquery doesn't actually return any data it stops running after evaluating the WHERE clause of the query. Therefore it is common practice to represent the columns of an EXISTS condition with an asterisk. Exist statements are more efficient than IN statements with a subquery because the exist statement doesn't need to evaluate the SELECT clause.

EXISTS Example

- This example checks for the existence Album IDs in the Track table and returns all records in the Album table where a match is found
- Note that the subquery is checking for A.AlbumId which is in the outer query

In this example we are checking for the existence of an Album ID in the Track table. If the ID exists, then the query will return the entire record with the corresponding ID from the Album table. As you can see, the Album ID number 4 in the Album table does not have a corresponding Album ID in the Track table, therefore it is excluded from the final result set.

NOT EXISTS Example

- This example checks for Album IDs that don't exist in the Track table and only returns records in the Album table where a match is *not* found
- Other than the addition of the NOT keyword the query is identical to the EXISTS example

By adding the NOT keyword in front of the EXISTS statement we are flipping the logic criteria. Instead only Album records that don't have a corresponding Track record will be displayed in the result set. In this case only the record with an Album ID of 4 is returned.

The NOT Keyword

- The NOT keyword can be used to negate (i.e. return the opposite) the meaning of a keyword in a WHERE clause
- Place the NOT key word immediately before the keyword you wish to negate

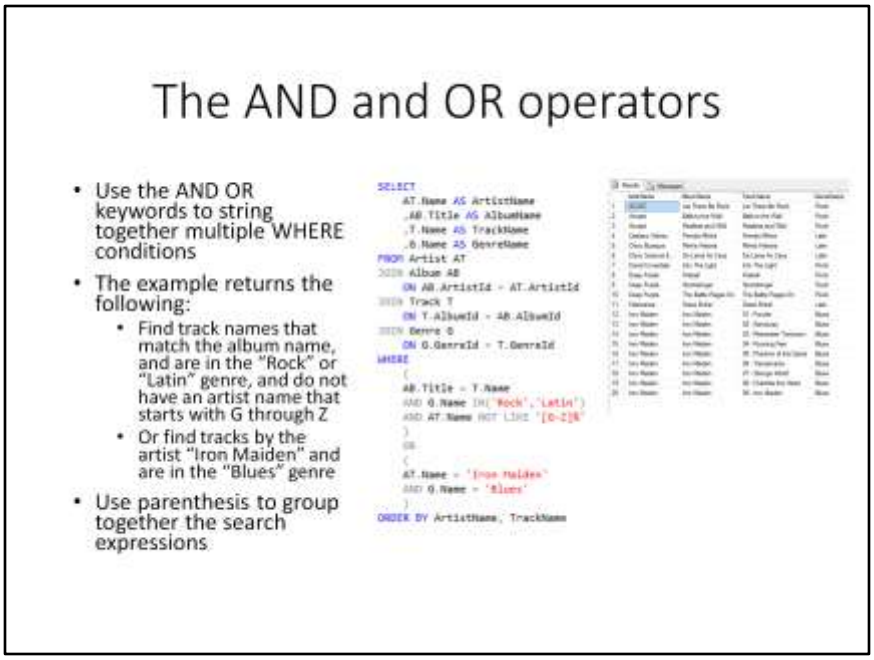
```
SELECT
    CustomerId
    ,InvoiceDate
    ,BillingCity
    ,BillingCountry
FROM Invoice
WHERE InvoiceDate NOT BETWEEN '2010-01-03' AND '2010-01-26'
```

```
SELECT *
FROM Album_temp A
WHERE NOT EXISTS (
    SELECT *
    FROM Track_temp T
    WHERE T.AlbumId = A.AlbumId)
```

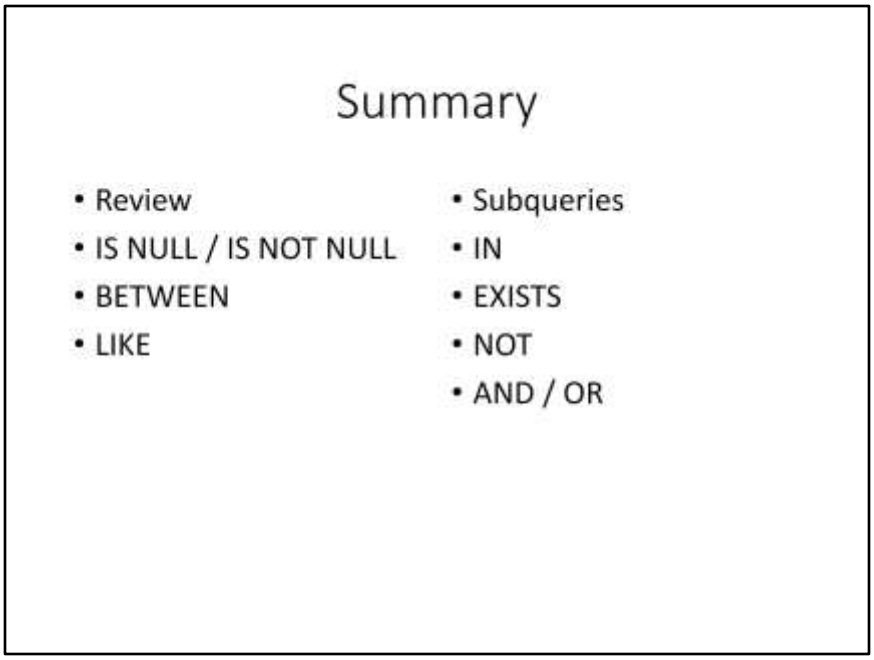
```
SELECT
    FirstName
    ,LastName
    ,Email
FROM Customer
WHERE Email NOT LIKE '%gmail%'
```

```
SELECT
    CustomerId
    ,LastName, Phone, Email, Country
FROM Customer
WHERE Country NOT IN('Brazil','United Kingdom','Sweden')
ORDER BY Country
```

The NOT keyword is used to reverse the logic used in a WHERE condition. The NOT keyword is inserted before the WHERE condition keyword. As you can see from the examples, the BETWEEN, EXISTS, LIKE and IN where conditions can all take the NOT keyword.



With knowledge of the WHERE conditions available in SQL Server, it is possible to string together multiple conditions using the AND and OR operators to create a complex search expression. In the example I use several equal operators as well as an IN and LIKE condition to return the result set I want. I start by searching for records that have the same album and track names. I further filter this result set by only including tracks in the Rock or Latin genre that don't have an Artist whose name begins with the letters G through Z. The OR operator creates a second search condition that looks for tracks by Iron Maiden that were labeled in the Blues genre. If the records meet either of these conditions then they are returned in the result set. In the example 20 records fit the criteria.



This completes the presentation on additional WHERE conditions.