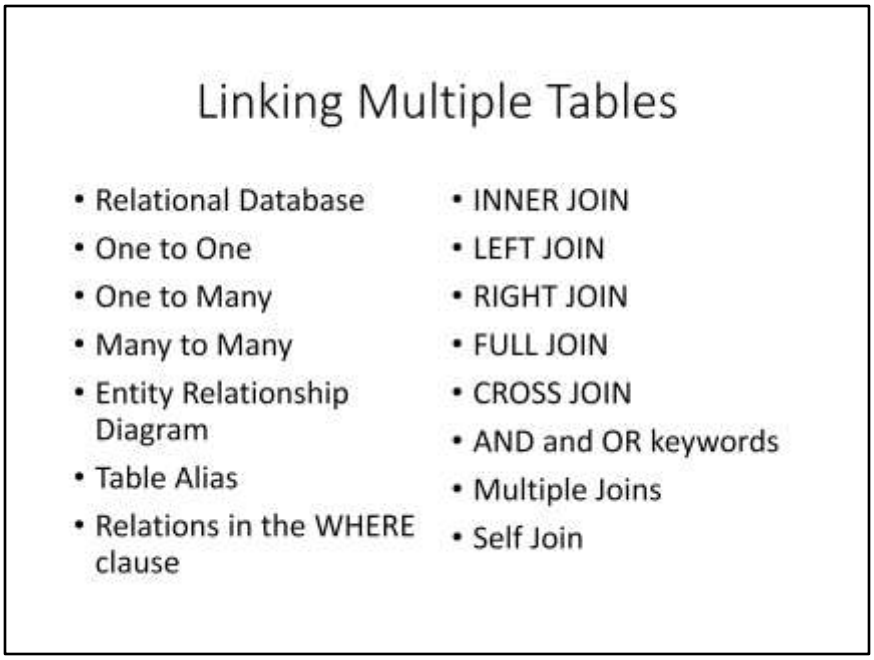
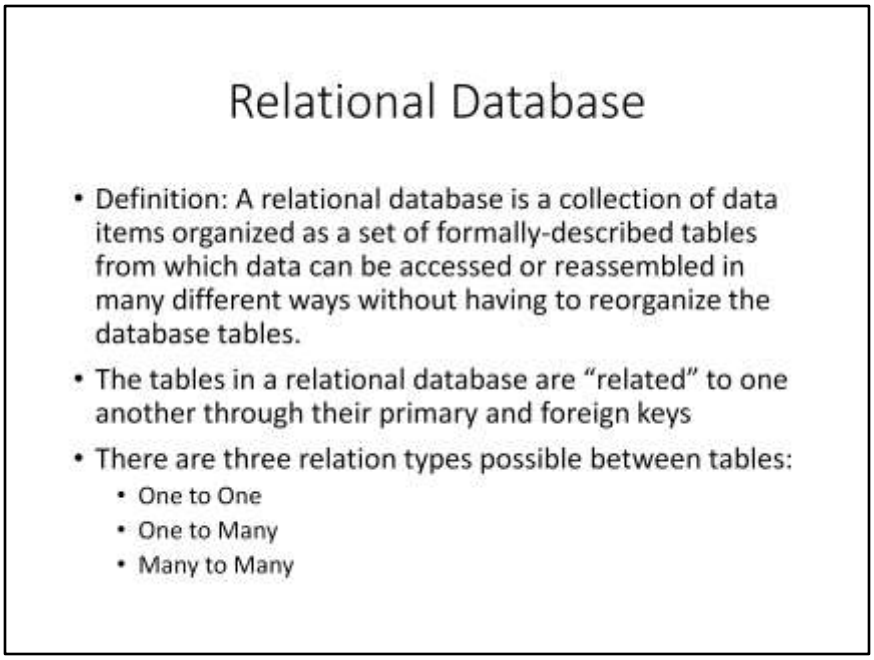


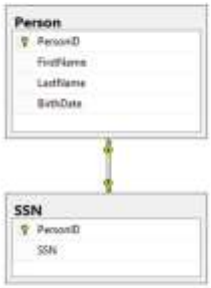
This presentation will cover the basics of linking together multiple tables. I will talk about table relationships, join types, and their components.



To understand how tables link to one another you need to understand the definition of a relational database. As defined by the website "TechTarget": A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. Not having to reorganize tables is the important point to take away. You can use SQL to display your data in the format needed. Tables in a database are related to each other through their primary and foreign keys. The three relation types are one to one, one to many, and many to many.

One to One Relationship

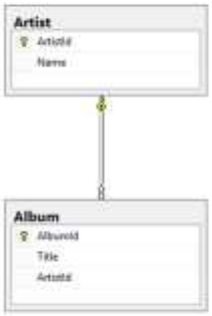
- A single record in the parent table can only relate to a single record in the child table
- For example a person can only have one Social Security Number



In a one to one relationship a record in the parent table can only connect to a single record in the child table. Oftentimes the foreign key in the child table is also the table primary key. This type of relation is common when a large table with hundreds of columns is broken down into two or more smaller tables.

One to Many Relationship

- A single record in the parent table can relate to multiple records in the child table, but a child record can only have one parent record
- For example a single artist can have multiple albums



In a one to many relationship, a record in the parent table can link to several records in the child table. However the child table records can only link to a single record in the parent table. In the example on this slide the records in the Artist table can have several Album records, but an album can only have one artist.

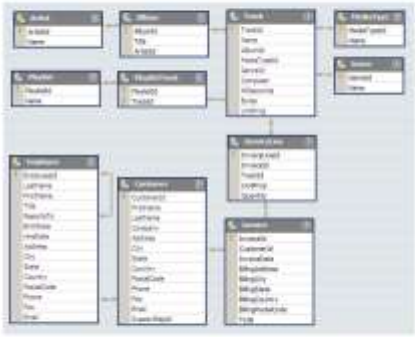
Many to Many Relationship

- Multiple records in the parent table can relate to multiple records in the child table
- To represent a many to many relationship, a linking table needs to be created that references the primary keys of both tables
- For example a playlist can have many tracks, and a track can be in many playlists
- The PlayListTrack table is a linking table. It contains the primary keys for both the Playlist and Track tables

In a many to many relationship the parent table can have several children, and the children can have several parents. It is not possible to directly link two tables in a many to many relationship. Therefore a linking table is required. The linking table will include the primary keys of both the child and parent tables. In the example a playlist can have multiple tracks, and a track can be on multiple playlists. Therefore the PlaylistTrack table is needed as a go-between to store playlist and track matches.

Entity Relationship Diagrams

- An ERD shows how tables relate to one another
- The tables are connected via their primary and foreign keys
- There are applications that allow you to create ERDs like Microsoft Visio (I prefer 2010 to 2013)
- SQL Server Management Studio can also display relationships explicitly defined in the database



When working with tables in a database, it is ideal to have an entity relationship diagram of the database so you can see how the tables relate to one another. Defining relationships and building an ERD are good database documentation practices. Unfortunately in the real world not all database documentation will include an ERD. If you want or need to create an entity relationship diagram, there are serveral 3rd party tools on the market. My personal favorite is Microsoft Visio 2010. SQL Server Management Studio has a built-in data diagram tool which is useful if you have explicitly defined your primary and foreign keys in the database.

Multiple Tables in the FROM Clause

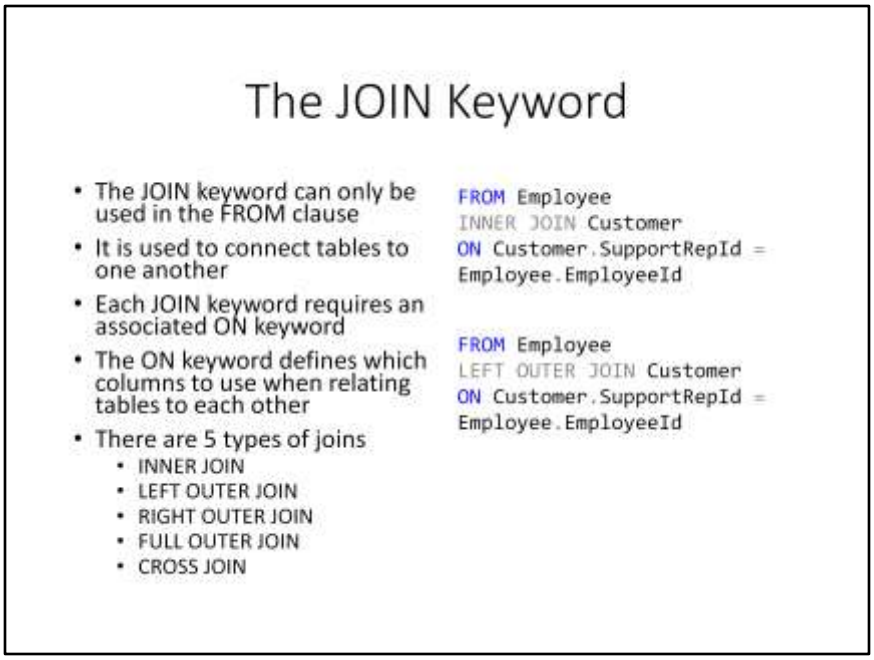
- If you want to pull data from multiple tables then they must all be included in the FROM clause
- The JOIN keyword is used to define relationships between tables
- When working with data from multiple tables it is good practice to prefix the columns with their table name or alias

```
SELECT
    Customer.FirstName
    ,Customer.LastName
    ,Invoice.Total
    ,Invoice.InvoiceDate
FROM Customer
JOIN Invoice
    ON Invoice.CustomerId = Customer.CustomerId
WHERE Customer.LastName = 'Philips'
```

Results | Messages

	FirstName	LastName	Total	InvoiceDate
1	Mark	Philips	8.91	2009-01-06 00:00:00.000
2	Mark	Philips	1.98	2010-08-13 00:00:00.000
3	Mark	Philips	3.96	2010-11-15 00:00:00.000
4	Mark	Philips	5.94	2011-02-17 00:00:00.000
5	Mark	Philips	0.99	2011-10-08 00:00:00.000
6	Mark	Philips	1.98	2013-03-31 00:00:00.000
7	Mark	Philips	13.86	2013-05-11 00:00:00.000

If you need to extract data from more than one table in a query, then you need to include those tables in the FROM clause. Within a FROM clause you use the JOIN keyword to define table relationships. When you are working with multiple tables it is a good practice to prefix column names with their corresponding table name or table alias.



The JOIN keyword is use to define how tables relate to one another within a query. The JOIN keyword is used as a part of the FROM clause. Each JOIN keyword requires an ON keyword. The ON keyword is where the relating columns are defined. There are five types of joins. Inner joins, left joins, right joins, full joins, and cross joins.

JOIN Sample Tables

- The tables Album_temp and Track_temp are used in the upcoming JOIN slides
- The tables are related on the AlbumId column
- There is a missing matching record on each table

```
SELECT
    AlbumId
    ,Title
FROM Album_temp
```

AlbumId	Title
1	For Those About To Rock We Salute You
3	Restless and Wild
4	Let There Be Rock

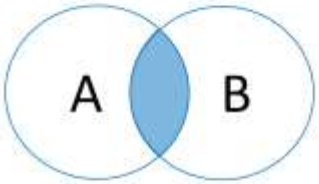
```
SELECT
    TrackId
    ,Name
    ,AlbumId
FROM Track_temp
ORDER BY AlbumId
```

TrackId	Name	AlbumId
8	Inject The Venom	1
2	Balls to the Wall	2
3	Fast As a Shark	3

For the following set of slides I created two sample tables called album temp and track temp. Each table has a total of 3 records. Two of three records are related while the 3rd record on each table doesn't have corresponding record in the other table. The tables are related on the column named albumID.

INNER JOIN

- An INNER JOIN only returns values that have a record in both table A and table B
- The "INNER" key word is optional when using an inner join



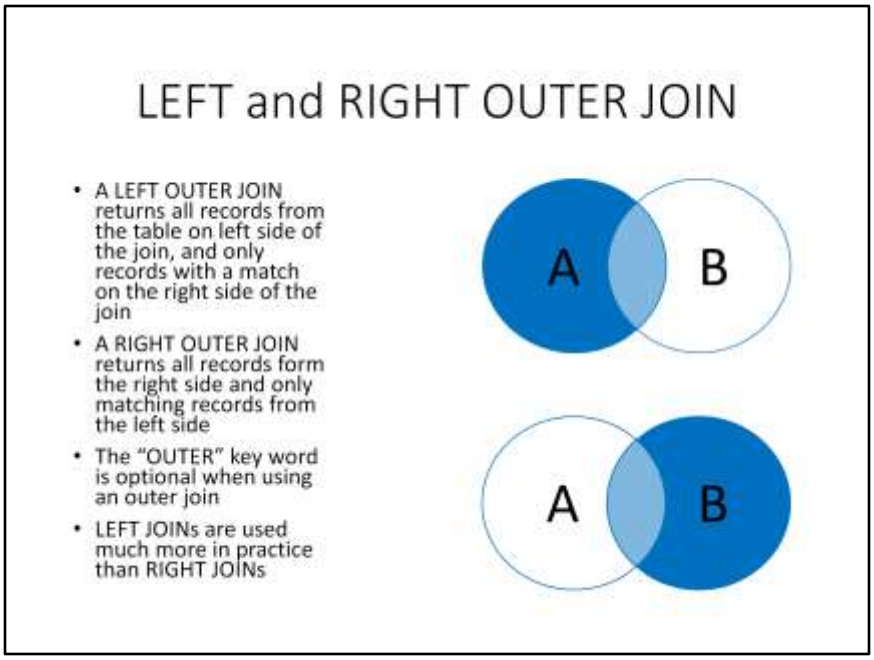
An INNER JOIN will only return records that exist in both tables. In the diagram on this slide only the area where the tables overlap is returned. The INNER keyword is optional when writing an inner join and does not need to be included.

INNER JOIN Example

- Joining the Album_temp and Track_temp tables
- Only rows returned are those where the AlbumID exists in both tables

```
SELECT
    Album_temp.AlbumId
    ,Album_temp.Title
    ,Track_temp.TrackId
    ,Track_temp.Name
FROM Album_temp
JOIN Track_temp
    ON Album_temp.AlbumId = Track_temp.AlbumId
ORDER BY Album_temp.AlbumId
```

In the example on this slide I am running an inner join against the album and track temp tables on the column AlbumID. This means only records that have the same AlbumID number in both tables will be returned. Only 2 records meet that criteria so they are the ones displayed in the result set. Also note that I am pulling column information from both tables.



A LEFT JOIN returns all records from the table on the left side of the join, and only those records on the right side that have a match on the left. Similarly, a RIGHT JOIN returns all records from the right side and only matching records from the left. The OUTER keyword is optional when using a LEFT or RIGHT JOIN. In practice LEFT JOINs are used about 99% more than RIGHT JOINs.

LEFT JOIN Example

- All rows from the left table (Album_temp) are returned
- Only rows from the right table (Track_temp) with a matching AlbumId are returned

```
SELECT
    Album_temp.AlbumId
    ,Album_temp.Title
    ,Track_temp.TrackId
    ,Track_temp.Name
FROM Album_temp
LEFT JOIN Track_temp
    ON Album_temp.AlbumId = Track_temp.AlbumId
ORDER BY Album_temp.AlbumId
```

In this example I executed a LEFT JOIN on the Album and Track temp tables. Because Album is in the left position all of its records were returned. Data from the Track table was returned where there was a match. However there is no AlbumID with the number 4 in the Track table so null values were returned instead for the Track columns.

RIGHT JOIN Example

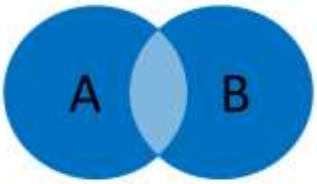
- All rows from the right table (Track_temp) are returned
- Only rows from the left table (Album_temp) with a matching AlbumId are returned

```
SELECT
    Album_temp.AlbumId
    ,Album_temp.Title
    ,Track_temp.TrackId
    ,Track_temp.Name
FROM Album_temp
RIGHT JOIN Track_temp
    ON Album_temp.AlbumId = Track_temp.AlbumId
ORDER BY Album_temp.AlbumId
```

In this example I switched the LEFT JOIN to a RIGHT JOIN. This means all data from the Track table will be returned while only matching Album data will be returned. As you can see in the result set, we have values for all the Track columns, but TrackID 2 has no Album column data because there wasn't a match.

FULL OUTER JOIN

- A FULL OUTER JOIN returns all records from the table on left side of the join, and all records from the right side of the join
- The "OUTER" key word is optional when using a full outer join



A FULL JOIN combines the aspects of a LEFT and RIGHT JOIN into one. All data from the left table and all data from the right table will be displayed. As with RIGHT and LEFT JOINs, the OUTER keyword is optional.

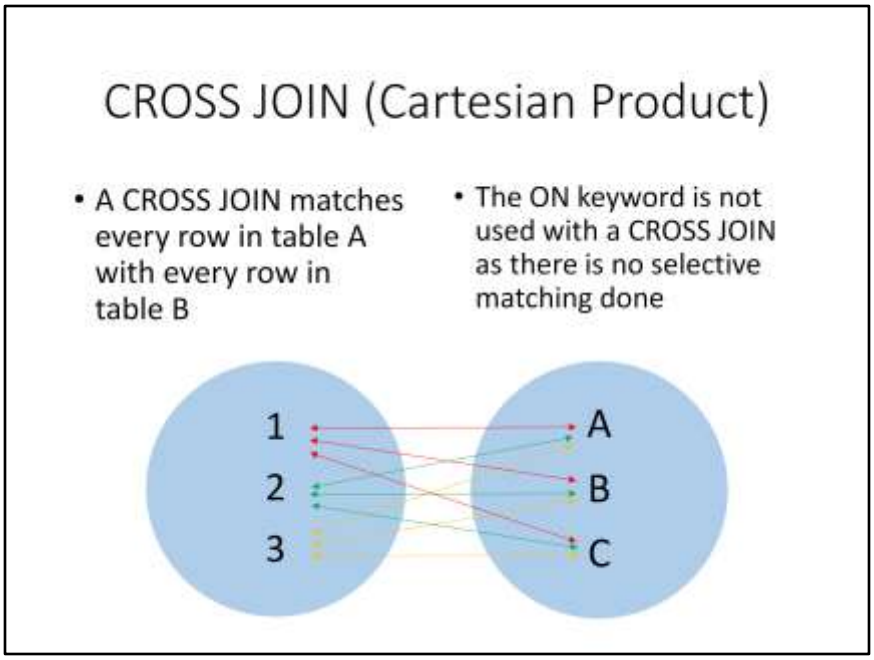
FULL JOIN Example

- All rows from the left table (Album_temp) are returned
- All rows from the right table (Track_temp) are returned

```
SELECT
    Album_temp.AlbumId
    ,Album_temp.Title
    ,Track_temp.TrackId
    ,Track_temp.Name
FROM Album_temp
FULL JOIN Track_temp
    ON Album_temp.AlbumId = Track_temp.AlbumId
ORDER BY Album_temp.AlbumId
```

	AlbumId	Title	TrackId	Name
1	NULL	NULL	2	Balls to the Wall
2	1	For Those About To Rock We Salute You	8	Inject The Venom
3	3	Restless and Wild	3	Fast As a Shark
4	4	Let There Be Rock	NULL	NULL

In the example you can see all records from the Album and Track tables are displayed. When a match is found then the records are merged together. When no match is found then only the data from the respective table is displayed.

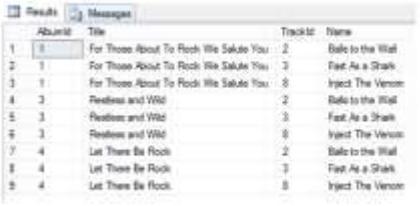


A CROSS JOIN is the equivalent to a Cartesian Product. A Cartesian Product is when a record in one table is matched to the every other record in the second table. The total rows returned will be the product of the count of each table. If there are 3 rows in each table then 3 times 3 equals 9. Four rows in each table will return 16, and so on and so on. You do not use the ON keyword with a CROSS JOIN.

CROSS JOIN Example

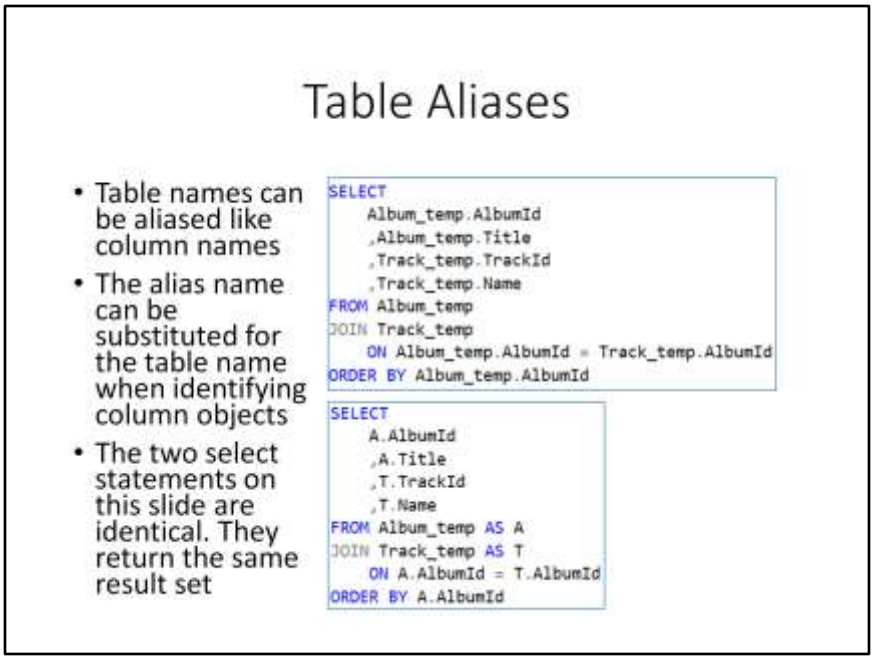
- Every row in the Album_temp table are matched with every row in the Track_temp table
- Each table has 3 rows so 3x3 returns 9 rows in the result set

```
SELECT
    Album_temp.AlbumId
    ,Album_temp.Title
    ,Track_temp.TrackId
    ,Track_temp.Name
FROM Album_temp
CROSS JOIN Track_temp
ORDER BY Album_temp.AlbumId
```



	AlbumId	Title	TrackId	Name
1	1	For Those About To Rock We Salute You	2	Balls to the Wall
2	1	For Those About To Rock We Salute You	3	Fast As a Shark
3	1	For Those About To Rock We Salute You	8	Inject The Venom
4	3	Restless and Wild	2	Balls to the Wall
5	3	Restless and Wild	3	Fast As a Shark
6	3	Restless and Wild	8	Inject The Venom
7	4	Let There Be Rock	2	Balls to the Wall
8	4	Let There Be Rock	3	Fast As a Shark
9	4	Let There Be Rock	8	Inject The Venom

In our example you can see that the each of the 3 rows in the Album table were matched to each of the 3 rows in the Track table. This returned a result set of 9 rows.



It is a common practice to alias your table names when working with multiple tables. This allows you to identify which table a column belongs to without having to write out the full name of the table. The syntax for creating a table alias is the same as that for a column alias. In the example both queries will return the same results, but the query with the table aliases, is cleaner and easier to read.

Joining more than two tables

- Additional tables can be added to the query with additional JOIN clauses

```
SELECT
    A.Name AS ArtistName
    ,AB.Title AS AlbumName
    ,T.Name AS TrackName
FROM Artist A
JOIN Album AB
    ON AB.ArtistId = A.ArtistId
JOIN Track T
    ON T.AlbumId = AB.AlbumId
    AND T.Name LIKE 'The%'
WHERE A.Name = 'Iron Maiden'
ORDER BY
    AlbumName
    ,TrackName
```

It is possible to join more than two tables together in a query. You only need to add additional JOIN keywords for each additional table. In the example I joined the Artist table to the Album table, and the Album table to the Track table. Notice that I used column aliases to prevent duplicate column names from appearing in the result set.

AND and OR keywords

- Use the AND and OR keywords after the ON keyword to further specify the relation between the tables in the JOIN clause
- AND and OR have the same syntax as in a WHERE clause

```
SELECT
    A.Name AS ArtistName
    ,AB.Title AS AlbumName
    ,T.Name AS TrackName
FROM Artist A
JOIN Album AB
    ON AB.ArtistId = A.ArtistId
LEFT JOIN Track T
    ON T.AlbumId = AB.AlbumId
    AND T.Name = 'The Longest Day'
WHERE A.Name = 'Iron Maiden'
```

Results | Messages

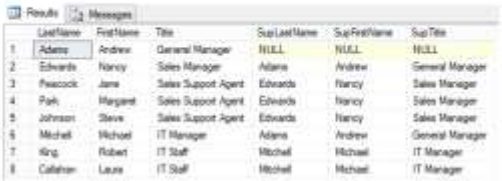
	ArtistName	AlbumName	TrackName
1	Iron Maiden	A Matter of Life and Death	The Longest Day
2	Iron Maiden	A Real Dead One	NULL
3	Iron Maiden	A Real Live One	NULL
4	Iron Maiden	Brave New World	NULL
5	Iron Maiden	Dance Of Death	NULL
6	Iron Maiden	Fear Of The Dark	NULL
7	Iron Maiden	Iron Maiden	NULL
8	Iron Maiden	Killers	NULL
9	Iron Maiden	Live After Death	NULL

Sometimes you will need to specify more than one criteria in a JOIN clause to get the data you need. You can tack on these additional criteria by using the AND or OR keywords. In the example I wanted to return all Albums by Iron Maiden but only display Track names that are equal to “The Longest Day”. You can see in the result set that only the Longest Day track is displayed, but all album titles are displayed because we used a LEFT JOIN.

Self Joins

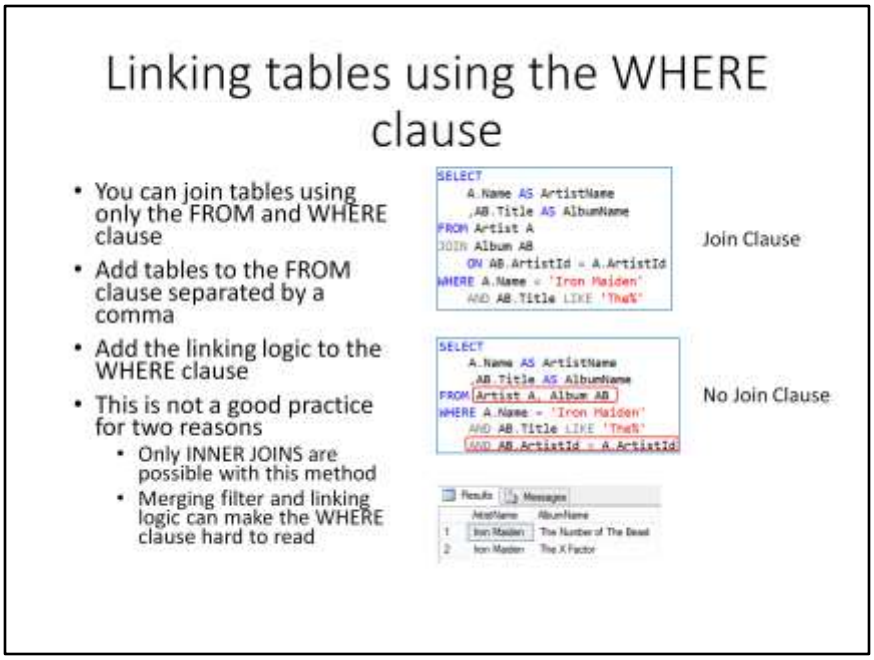
- It is possible to join a table against itself
- The columns you join on should be related in some way
- In the example the ReportsTo column is the EmployeeId of an employee's manager
- It is mandatory to use table aliases when implementing self joins

```
SELECT
    E.LastName
    ,E.FirstName
    ,E.Title
    ,RT.LastName AS SupLastName
    ,RT.FirstName AS SupFirstName
    ,RT.Title AS SupTitle
FROM Employee E
LEFT JOIN Employee RT
    ON RT.EmployeeId = E.ReportsTo
```

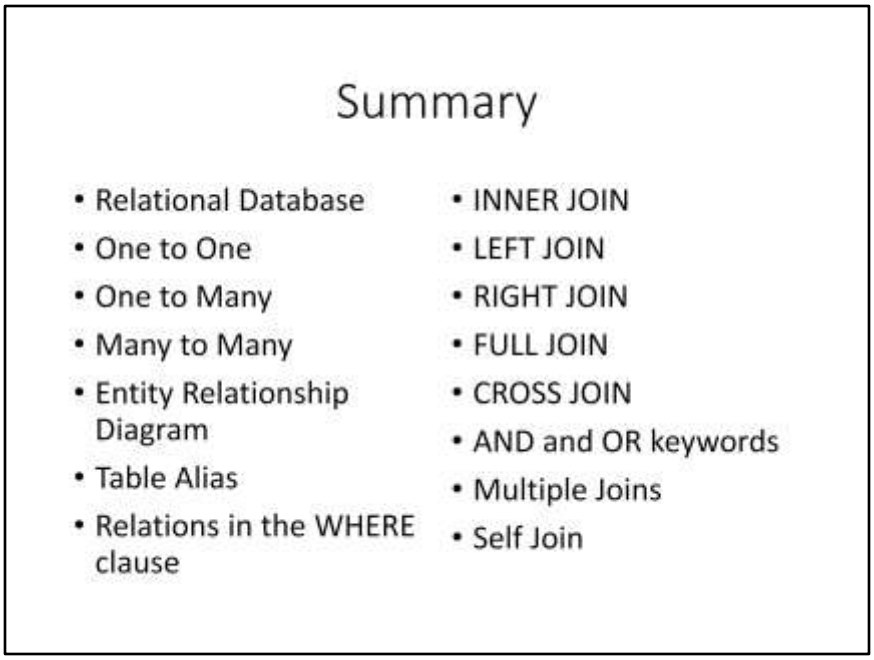


	LastName	FirstName	Title	SupLastName	SupFirstName	SupTitle
1	Adams	Andrew	General Manager	NULL	NULL	NULL
2	Edwards	Nancy	Sales Manager	Adams	Andrew	General Manager
3	Peacock	Jane	Sales Support Agent	Edwards	Nancy	Sales Manager
4	Park	Margaret	Sales Support Agent	Edwards	Nancy	Sales Manager
5	Johnson	Steve	Sales Support Agent	Edwards	Nancy	Sales Manager
6	Mitchell	Michael	IT Manager	Adams	Andrew	General Manager
7	King	Robert	IT Staff	Mitchell	Michael	IT Manager
8	Callahan	Laura	IT Staff	Mitchell	Michael	IT Manager

It is possible for a table to be joined against itself. This usually occurs when the table columns contain hierarchical data that is related to one another. In the Employee table we have the EmployeeID column and the ReportsTo column. The ReportsTo column is the employee ID of that person's supervisor. By joining on these two columns, we can display the name and title of an employee's supervisor on the same row. An important point to remember is it is mandatory to use table aliases, otherwise SQL Server will not be able to differentiate between two tables with the same name.



Although it is not commonly done, it is possible to join tables by using the FROM and WHERE clause only. This is done by entering the table names on the FROM clause separated by commas. You then enter the join criteria as part of the WHERE clause statement. The example shows two queries that return the same results. One uses a JOIN and the other one doesn't. It is not considered good practice to use the WHERE clause for your joins for two reasons. The first reason is that only INNER JOINS can be represented in a WHERE clause. OUTER JOINs are not possible. Second merging your filter and linking logic can clutter the WHERE clause and make your statement difficult to read.



This concludes the presentation on linking multiple tables.