Table Relationships

Database Concepts

- A ***relational database*** a collection of tables of data all of which are formally described and organized according to the relational model.

- The relationships between the tables in a database correspond to the relationships between the entities (tables) they represent.

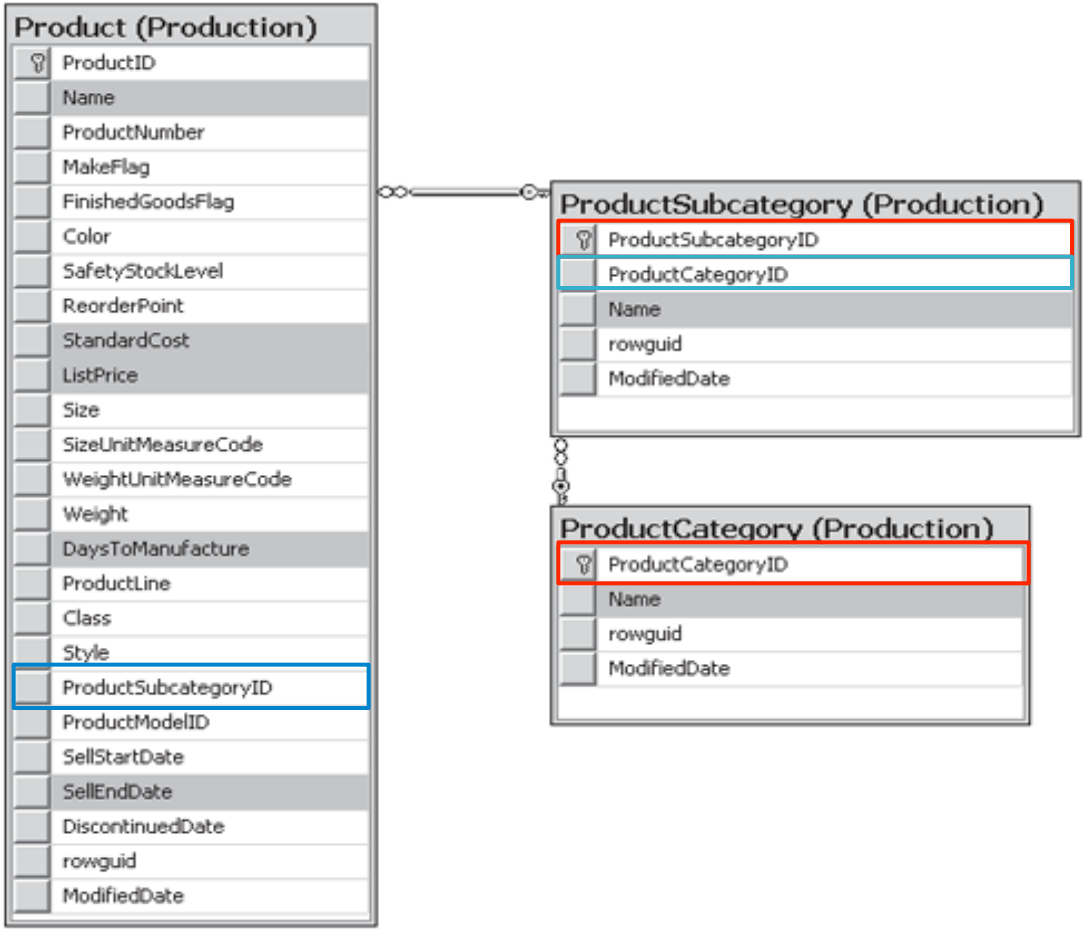
Database Concepts

- **How the tables in a relational database are related:**

- The tables in a relational database are related to each other through their key columns. A ***foreign key*** identifies a primary key in another table. A table may contain one or more foreign keys.

- When you define a **foreign key** for a table in SQL Server, you can't add rows to the table with the foreign key unless there's a matching **primary key** in the related table.

Sample relational structure



How the tables in a relational database are related:

The relationships between the tables in a database correspond to the relationships between the entities they represent.

There are three types of relationships between tables:

- ***one-to-many relationship*** (most common type)
- ***one-to-one relationship***
- ***many-to-many relationship***

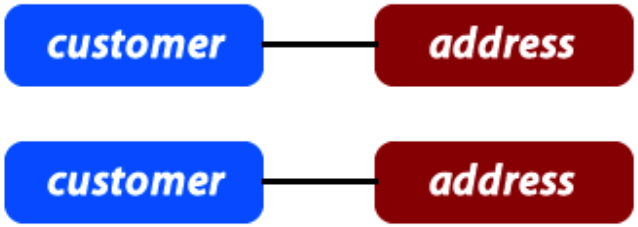
A *one-to-one relationship* with another table.

In a one-to-one relationship, a row in table A can have no more than one matching row in table B, and vice versa.

A one-to-one relationship is created if both of the related columns are primary keys or have unique constraints.

A *one-to-one relationship* with another table.

For example a Employee table would only have one address:



A *one-to-one relationship* with another table.

For example a Employee table would only have one address:

```
SELECT *
FROM
    Employee e INNER JOIN EmployeeAddress ea
    ON e.EmployeeID = ea.EmployeeID
```

A *one-to-one relationship* with another table.

This type of relationship is not common because most information related in this way would be all in one table. You might use a one-to-one relationship to:

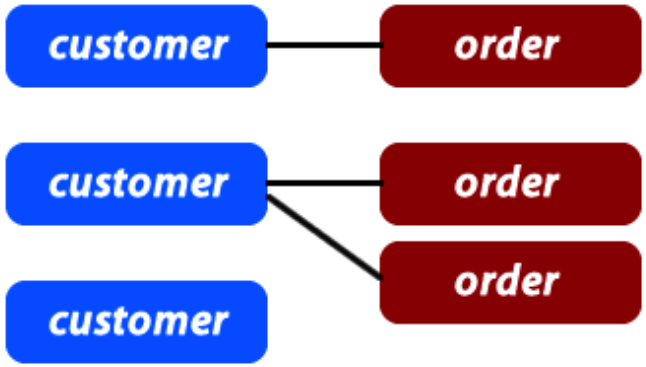
- Divide a table with many columns.
- Isolate part of a table for security reasons.
- Store data that is short-lived and could be easily deleted by simply deleting the table.
- Store information that applies only to a subset of the main table.

The primary key side of a one-to-one relationship is denoted by a key symbol. The foreign key side is also denoted by a key symbol.

A *one-to-many relationship* with another table.

A one-to-many relationship is the most common type of relationship.

In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A.



A *one-to-many relationship* with another table.

For example, a Customer can have many addresses.

The primary key side of a one-to-many relationship is denoted by a key symbol. The foreign key side of a relationship is denoted by an infinity symbol.

A *one-to-many relationship* with another table.

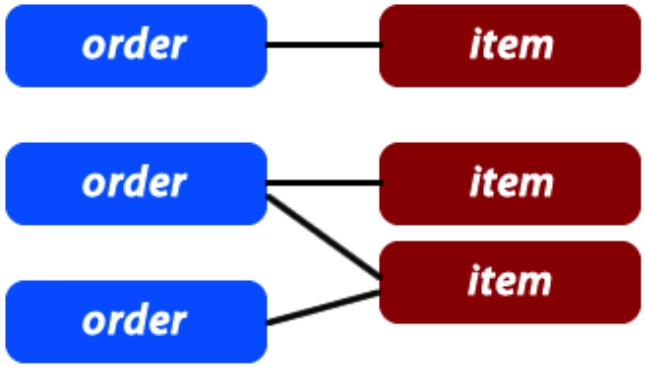
For example, a Customer can have many addresses.

```
SELECT
   c.CustomerID,
   ca.AddressID
FROM
   Customer c INNER JOIN CustomerAddress ca
   ON ca.CustomerID = c.CustomerID
   order by c.CustomerID, ca.AddressID
```

A *many-to-many relationship* with another table.

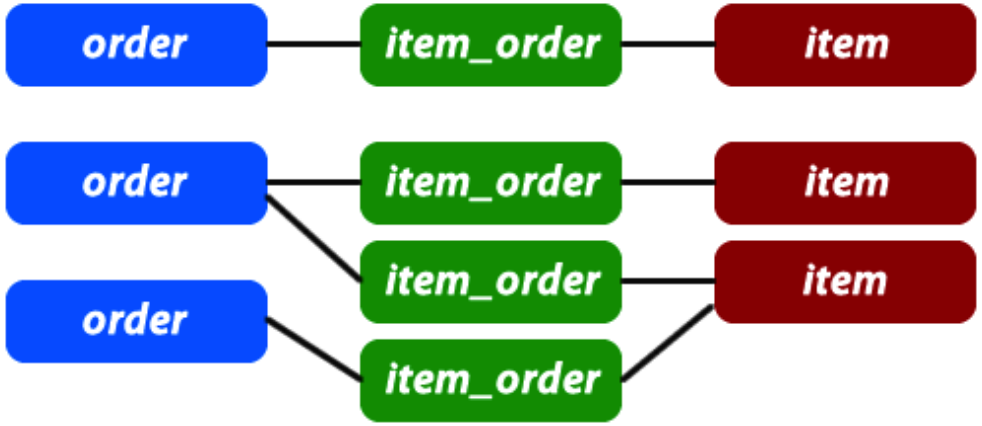
In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa.

You create such a relationship by defining a third table, called a junction table, whose primary key consists of the foreign keys from both table A and table B.



A *many-to-many relationship* with another table.

You create such a relationship by defining a third table, called a junction table, whose primary key consists of the foreign keys from both table A and table B.



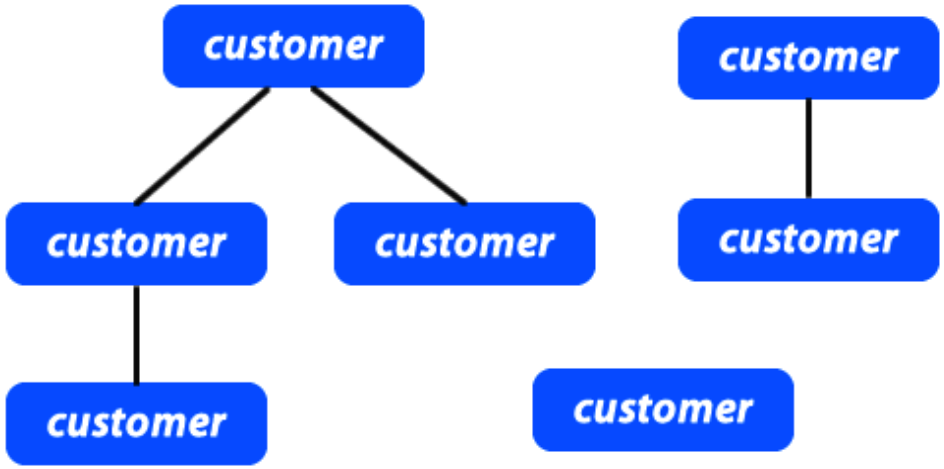
A *many-to-many relationship* with another table.

For example, the order table and the items table have a many-to-many relationship that is defined by a one-to-many relationship from each of these tables to the orderitems table.

The primary key of the orderitem table is the combination of the order_id column (the order table's primary key) and the item_id column (the item table's primary key).

A Self Referencing *relationship* .

This is used when a table needs to have a relationship with itself. For example, let's say you have a referral program. Customers can refer other customers to your shopping website. The table may look like this:



For example:

```
SELECT
   e1.EmployeeID EmployeeName,
   e2.EmployeeID AS ManagerName
FROM Employee e1
INNER JOIN Employee e2
ON e1.ManagerID = e2.EmployeeID
```

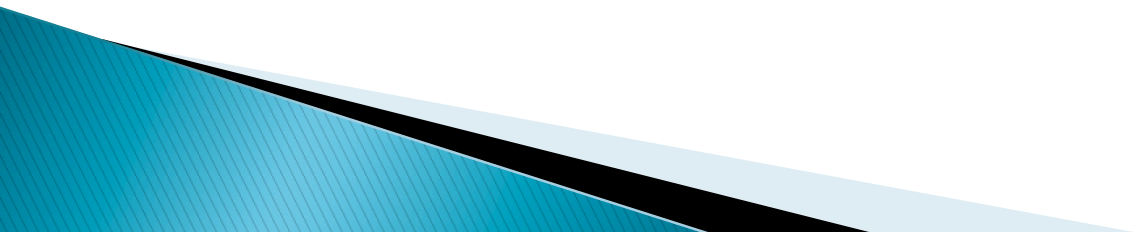
Database Diagram

STOP – Do Diagram Exercise

A database is just not tables. A database is a system of:

- Tables
- Views
- Stored Procedures
- Functions
- User Defined Data Types.

All of these objects combine together to make a database and support a relational database.



The benefits of a database:

- Data entry, updates and deletes will be efficient
- Data retrieval, summarization and reporting will be efficient
- It will behave predictably
- Since data stored in database is somewhat self-documenting
- Changes to database schema are easy to make

JOIN FUNDAMENTALS

- Joining Tables
- Combining Queries

Relational Database FUNDAMENTALS

- A **relational database** is a database that stores information about both the data and how it is related.
- Each database is a collection of related tables.
- Each table is a physical representation of an entity or object that is in a tabular format consisting of columns and rows.

Relational Database FUNDAMENTALS

- The primary keys within a database are used to define the relationships among the tables.
- When a PK migrates to another table, it becomes a foreign key in the other table.

JOIN FUNDAMENTALS

- By using joins, you can retrieve data from two or more tables based on logical relationships between the tables.
- SQL ***JOIN*** joins together two tables on a matching table column, ultimately forming one single temporary table. The key word here is *temporary*.
- The tables themselves remain intact, and running a ***JOIN*** query does not in any way change the data or table structure.

JOIN FUNDAMENTALS

- In order to perform a *JOIN* , we need a few pieces of information:
  - the name of the table and
  - column we want to join on and a condition to meet for the *JOIN* to happen.

JOIN FUNDAMENTALS

- A join condition defines the way two tables are related in a query by:
  - Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.
  - Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.

JOIN FUNDAMENTALS

- Implicit table JOINs

- SELECT *
  FROM employee, department
  WHERE employee.DepartmentID = department.DepartmentID

- Explicit table JOINs

- SELECT *
  FROM employee
  INNER JOIN department
  ON employee.DepartmentID = department.DepartmentID

JOIN FUNDAMENTALS

- Best Practice is to use Explicit JOINs. Here is basic SQL syntax:

- SELECT
  - Column_Name,
  - Column_Name
- FROM
  - Table_name1 t1 INNER JOIN Table_name2 t2
  - ON t2.column_name = t1.column_name

JOIN FUNDAMENTALS

- Example:

- SELECT
  - Column_Name,
  - Column_Name
- FROM
  - Table_name1 t1 INNER JOIN Table_name2 t2
  - ON t2.column_name = t1.column_name

JOIN FUNDAMENTALS

- Both inner and outer joins can be used to combine data from two or more tables; however, there are some key differences
- Though both inner and outer joins include rows from both tables when the match condition is successful, they differ in how they handle a false match condition
- Inner joins don't include non-matching rows; whereas, outer joins do include them.

JOIN FUNDAMENTALS

- Inner joins don't include non-matching rows; whereas, outer joins do include them.
- An inner join is used to return results by combining rows from two or more tables.
- In its simplest case, where there is no join condition, an inner join would combine all rows from one table with those from another.

JOIN FUNDAMENTALS

- Inner joins don't include non-matching rows; whereas, outer joins do include them.
- An inner join is used to return results by combining rows from two or more tables.
- In its simplest case, where there is no join condition, an inner join would combine all rows from one table with those from another.

JOIN FUNDAMENTALS

- If a customer has more than one address, then more than one match is made.
- From this you can see we may get more rows returned than we have for each customer

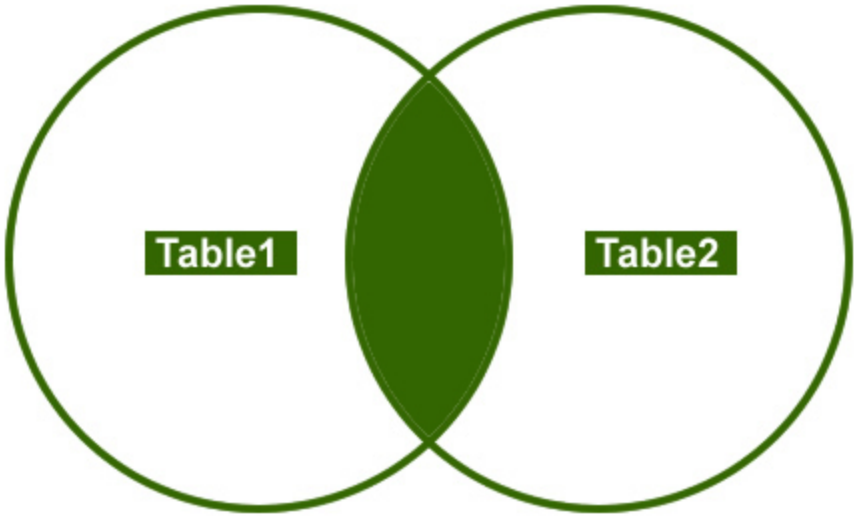
JOIN FUNDAMENTALS

For example:

```
SELECT
        c.CustomerID,
        ca.AddressID
FROM
        Customer c INNER JOIN CustomerAddress ca
        ON ca.CustomerID = c.CustomerID
        order by c.CustomerID, ca.AddressID
```

INNER JOIN

- This JOIN returns rows when there is at least one match in both the tables.
- INNER JOIN option is considered to be the most common join needed in applications and/or queries.



INNER JOIN

This query returns Name form Contact table and Title from Employee table.

```
SELECT  c.FirstName,
        c.LastName,
        e.Title
FROM Employee AS e
    INNER JOIN Contact AS c
    ON e.ContactID =  c.ContactID
ORDER BY c.LastName
```

JOIN FUNDAMENTALS

STOP – Do exercise e 14

JOIN FUNDAMENTALS

- An outer join is used to return results by combining rows from two or more tables.
- But unlike an inner join, the outer join will return every row from one specified table, even if the join condition fails

JOIN FUNDAMENTALS

- Take the Contact table example above. If we want to list every Contact regardless of whether they had a credit card , then using an outer join would make it so.

- So when a LEFT OUTER JOIN is used, all rows for the table in the FROM clause are included in the result, even if a match isn't found with the other table

JOIN FUNDAMENTALS

For example:

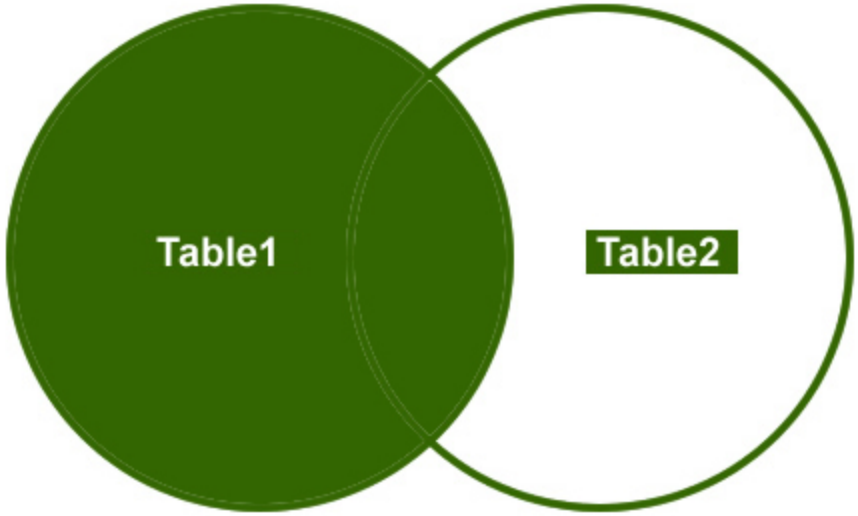
```
SELECT
   c.ContactID,
   cc.CreditCardID
FROM
        Contact c Left Outer JOIN ContactCreditCard cc
        ON cc.ContactID = c.ContactID
Order by cc.CreditCardID
```

JOIN FUNDAMENTALS

- Inner joins can be specified in either the FROM or WHERE clauses.
- Outer joins can be specified in the FROM clause only.
- The join conditions, combined with the WHERE and HAVING search conditions, control the rows that are selected from the base tables referenced in the FROM clause.

LEFT OUTER JOIN

- This join returns all the rows from the left table in conjunction with the matching rows from the right table. If there are no columns matching in the right table, it returns NULL values.
- Another item to keep in mind is that the LEFT and RIGHT OUTER JOIN logic is opposite of one another. So you can change either the order of the tables in the specific join statement or change the JOIN from left to right or vice versa and get the same results.



LEFT OUTER JOIN

To include all products, regardless of whether a review has been written for one, use an ISO left outer join. The following is the query:

```
SELECT
  p.Name,
  pr.ProductReviewID
FROM Product p LEFT OUTER JOIN ProductReview pr
  ON p.ProductID = pr.ProductID
```

JOIN FUNDAMENTALS

- STOP – Do exercise 15

SELF JOIN

- A table can be joined to itself in a self-join.
- Because this query involves a join of the Employee table with itself you must give the Employee table two different aliases (e1 and e2) in the FROM clause.
- We want to see which employee is being managed by which Manager.

SELF JOIN

- For example, you can use a self-join to find the products that are supplied by more than one vendor.

```
SELECT
        e1.ManagerID AS ManagerID,
        e2.EmployeeID EmployeeID
FROM Employee e1
        INNER JOIN Employee e2
        ON e2.ManagerID = e1.EmployeeID
 ORDER BY e1.ManagerID
```

SQL Server – Joining Tables

STOP – Do Exercise 16

JOIN FUNDAMENTALS

- Specifying the join conditions in the FROM clause helps separate them from any other search conditions that may be specified in a WHERE clause
- This is the recommended method for specifying JOINs.
- When multiple tables are referenced in a single query, all column references must be unambiguous.

JOIN FUNDAMENTALS

- Any column name that is duplicated between two or more tables referenced in the query must be qualified with the table name.
- When a column name is not duplicated in two or more tables used in the query, references to it do not have to be qualified with the table name.

SQL Server – Joining Tables

- The ability to join tables is one of the most powerful features of SQL
- Joins are the mechanism used to associate tables within SELECT statement
- Joins can be part of FROM or WHERE cluase

SQL Server – Joining Tables

- The key is to not have multiple occurrences of the same data in a table.
- Relational tables are designed to split data into multiple tables.
- You must have an understanding of relational database design

SQL Server – Joining Tables

- Each table has a unique identifier for each row know as *Primary Keys*.
- For example, a vendor table would have only vendor data and a Priamary key named vend_id

SQL Server – Joining Tables

- We would have another table called products which stores only data about products.
- The only vendor data it would store is the vendor_id column.
- This is called the *Foreign Key*. It relates the product table to the vendor table.

SQL Server – Joining Tables

- This *Foreign Key* (vend_id) in the products table allows you use the vendor table to retrieve vendor data .

SQL Server – Joining Tables

- STOP – Do exercise 17

SQL Server – Combining Queries

In SQL Server you have the ability to combine multiple datasets into one comprehensive dataset by using the:

- UNION operator or
- UNION ALL operator.

SQL Server – Combining Queries

There is a big difference in how these work as well as the final result set that is returned, but basically these commands join multiple datasets that have similar structures into one combined dataset.

SQL Server – Combining Queries

- UNION – this command will allow you to join multiple datasets into one dataset and will remove any duplicates that exist.

- Basically it is performing a DISTINCT operation across all columns in the result set.

SQL Server – Combining Queries

- UNION ALL – this command again allows you to join multiple datasets into one dataset, but it does not remove any duplicate rows.

- Because this does not remove duplicate rows this process is faster, but if you don't want duplicate records you will need to use the UNION operator instead.

SQL Server – Combining Queries

- In this first example we are using the UNION ALL operator against the Employee table from the AdventureWorks database.
- This is probably not something you would do, but this helps illustrate the differences of these two operators.
- There are 290 rows in table dbo.Employee.

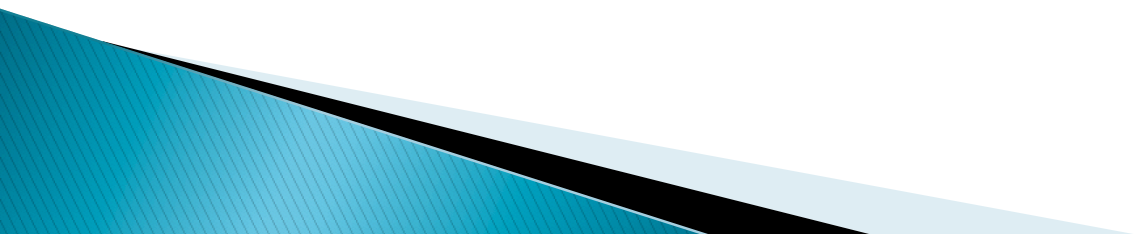
SQL Server – Combining Queries

```
SELECT * FROM dbo.Employee
UNION ALL
SELECT * FROM dbo.Employee
UNION ALL
SELECT * FROM dbo.Employee
```

SQL Server – Combining Queries

When this query is run the result set has 870 rows.

This is the 290 rows returned 3 times.



SQL Server – Combining Queries

In this next example we are using the UNION operator against the Employee table again from the AdventureWorks database.

SQL Server – Combining Queries

```
SELECT * FROM dbo.Employee
UNION
SELECT * FROM dbo.Employee
UNION
SELECT * FROM dbo.Employee
```

SQL Server – Combining Queries

When this query is run the result set has 290 rows.

Even though we combined the data three times the UNION operator removed the duplicate records and therefore returns just the 290 unique rows.

SQL Server – Joining Tables

- STOP – Do exercise 18