

## **Getting Data from Multiple Tables – The JOIN**

- In order to run a query that retrieves data from multiple tables, those tables can be **JOINED**.

## **Getting Data from Multiple Tables – The JOIN**

- In order to run a query that retrieves data from multiple tables, those tables can be **JOINED**.
- Joining two tables requires that the two tables have a common key (typically a foreign key relationship) that appears in both tables.

## **Getting Data from Multiple Tables – The JOIN**

- In order to run a query that retrieves data from multiple tables, those tables can be **JOINED**.
- Joining two tables requires that the two tables have a common key (typically a foreign key relationship) that appears in both tables.
- The common key columns need NOT have the same name, but must be of the same data type and length.

## Getting Data from Multiple Tables – The JOIN

- In order to run a query that retrieves data from multiple tables, those tables can be **JOINED**.
- Joining two tables requires that the two tables have a common key (typically a foreign key relationship) that appears in both tables.
- The common key columns need NOT have the same name, but must be of the same data type and length.
- A JOIN is one of the most **resource intensive** activities one can do in a relational database.

## **Basic Example**

Let's join nwOrders to nwEmployees

- nwOrders has 830 rows, each with an EmployeeID
- nwEmployees has 9 rows, each with an EmployeeID
- They have a common key: EmployeeID (primary key in nwEmployees; foreign key in nwOrders)
- We want SQL to join the rows in nwEmployees and nwOrders where the EmployeeID matches

Provide a listing showing Northwinds employees, sorted by LastName, and a count of each employee's orders from highest to lowest

```
Select LastName, Firstname, count(OrderID)as 'Orders'  
from nwEmployees, nwOrders  
where nwEmployees.EmployeeID =  
      nwOrders.EmployeeID  
GROUP BY LastName, FirstName  
Order By 1, 3 desc
```

## **Qualifying the Column Names**

- Since the column “EmployeeID” exists in BOTH tables in this query, when referring to EmployeeID, we need to tell SQL which one.
- Therefore, we suffix the table name in front of the column name separated by a “ . ”
- Failure to fully qualify the column name will result in an “ambiguous column” error

## Alternative

To save some typing, we can define an “alias” for each table. We can temporarily – only for the duration of this query -- rename the nwEmployees table “E”, and rename the nwOrders table “O”.

```
Select LastName, Firstname, count(OrderID) as 'Orders'
    from nwEmployees E, nwOrders O
   where E.EmployeeID = O.EmployeeID
  GROUP BY LastName, FirstName
 Order By 1
```



## **Beware the Cartesian Product**



- Product = one table multiplied by another table
- The JOIN often creates a product, then selects the product where the keys match
- For example, let's join nwOrders to nwEmployees
- nwOrders has 14 columns, 830 rows
- nwEmployees has 17 columns, 9 rows
- The Cartesian product has 31 (14+17) columns, and 7470 (830 \* 9) rows – most of which are meaningless

## Cartesian Product from an unqualified join:

customerid	companyname	contactname	country	OrderID	CustomerID	Orderdate	shipcountry
GREAL	Great Lakes Food Market	Howard Snyder	USA	10262	RATTC	2013-07-22	USA
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	USA	10262	RATTC	2013-07-22	USA
LAZYK	Lazy K Kountry Store	John Steel	USA	10262	RATTC	2013-07-22	USA
LETSS	Lets Stop N Shop	Jaime Yorres	USA	10262	RATTC	2013-07-22	USA
LONEP	Lonesome Pine Restaurant	Fran Wilson	USA	10262	RATTC	2013-07-22	USA
OLDWO	Old World Delicatessen	Rene Phillips	USA	10262	RATTC	2013-07-22	USA
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	USA	10262	RATTC	2013-07-22	USA
SAVEA	Save-a-lot Markets	Jose Pavarotti	USA	10262	RATTC	2013-07-22	USA
SPLIR	Split Rail Beer & Ale	Art Braunschweiger	USA	10262	RATTC	2013-07-22	USA
THEBI	The Big Cheese	Liz Nixon	USA	10262	RATTC	2013-07-22	USA
THECR	The Cracker Box	Liu Wong	USA	10262	RATTC	2013-07-22	USA
TRAIH	Trails Head Gourmet Provisioners	Helvetius Nagy	USA	10262	RATTC	2013-07-22	USA
WHITC	White Clover Markets	Karl Jablonski	USA	10262	RATTC	2013-07-22	USA
GREAL	Great Lakes Food Market	Howard Snyder	USA	10269	WHITC	2013-07-31	USA
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	USA	10269	WHITC	2013-07-31	USA
LAZYK	Lazy K Kountry Store	John Steel	USA	10269	WHITC	2013-07-31	USA
LETSS	Lets Stop N Shop	Jaime Yorres	USA	10269	WHITC	2013-07-31	USA
LONEP	Lonesome Pine Restaurant	Fran Wilson	USA	10269	WHITC	2013-07-31	USA
OLDWO	Old World Delicatessen	Rene Phillips	USA	10269	WHITC	2013-07-31	USA
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	USA	10269	WHITC	2013-07-31	USA
SAVEA	Save-a-lot Markets	Jose Pavarotti	USA	10269	WHITC	2013-07-31	USA
SPLIR	Split Rail Beer & Ale	Art Braunschweiger	USA	10269	WHITC	2013-07-31	USA

## **Cartesian Product**

- SQL must go through the Cartesian Product (which is an INTERIM answer set) row-by-row, and select only those rows where the EmployeeID from nwEmployees is equal to the EmployeeID from nwOrders
- Therefore, we must include the WHERE clause that describes this condition
- Failure to fully qualify a JOIN operation with a WHERE clause that matches all necessary keys will cause your answer set to include part or all of the Cartesian Product (which is mostly meaningless)
- The JOIN requires SQL to do a lot of work which consumes a lot of disk I/O and memory (= expensive)

## **Cartesian Product Example**

### **Create a table JoinOrder:**

```
CREATE TABLE JoinOrder (  
    OrderID    INT(11),  
    CustomerID CHAR(5),  
    OrderDate  DATE,  
    ShipCountry VARCHAR(15)  
);
```

## **Cartesian Product Example**

**Load JoinOrder with data from nwOrders where the order shipped to the USA**

```
INSERT INTO JoinOrder (OrderID, CustomerID,  
                        Orderdate, ShipCountry)  
    SELECT OrderID, CustomerID, Orderdate,  
           ShipCountry  
    FROM nwOrders where ShipCountry =  
'USA' ;
```

Loads 122 rows of data.

## **Cartesian Product Example**

### **Create a table JoinCustomer:**

```
CREATE TABLE JoinCustomer (  
    CustomerID  CHAR(5),  
    CompanyName VARCHAR(20),  
    ContactName VARCHAR(20),  
    Country    VARCHAR(20)  
);
```

## **Cartesian Product Example**

Load JoinCustomer with data from nwCustomers for customers located in the USA.

```
INSERT INTO JoinCustomer (CustomerID,  
                           CompanyName, ContactName, Country)  
SELECT CustomerID, CompanyName,  
       ContactName, Country  
FROM   nwCustomers  
WHERE  Country = 'USA';
```

Loads 13 rows of data.

## **Cartesian Product Example**

### **Create a Cartesian Product.**

```
SELECT c.customerid, companyname,  
       contactname, country, OrderID,  
       o.CustomerID, o.Orderdate, shipcountry  
FROM joincustomer c, joinorder o
```

Selects 1586 rows of data.



## Alternative join syntax

- SQL allows another syntax option for doing the JOIN.
- These queries are equivalent:

```
Select LastName, Firstname, count(OrderID) as 'Orders'
    from nwEmployees E, nwOrders O
    where E.EmployeeID = O.EmployeeID
    GROUP BY LastName, FirstName
    Order By 1
```

```
Select LastName, Firstname, count(OrderID) as 'Orders'
    from nwEmployees E JOIN nwOrders O
    on E.EmployeeID = O.EmployeeID
    GROUP BY LastName, FirstName
    Order By 1
```

## **Joining three or more tables**

- Every PAIR of tables being joined must have a common key
- Every PAIR of common keys must have a condition stated in a WHERE clause or in the “ON” clause of the JOIN
- Otherwise, your JOIN is not fully qualified and will result in a Cartesian Product (meaningless output)

## Examples – Joining three tables

Create a report showing each employee and the total value of their orders sorted from highest value to lowest. (Order Value = UnitPrice \* Quantity for each item on the order.)

```
Select LastName, Firstname,  
       sum(UnitPrice * Quantity) as 'OrderValue'  
from   nwEmployees E  
JOIN   nwOrders O on E.EmployeeID = O.EmployeeID  
JOIN   nwOrderDetails D on O.OrderID = D.OrderID  
GROUP BY LastName, FirstName  
Order By 3 desc
```

## Examples – Joining three tables

### Same Query, Different Syntax

```
Select LastName, Firstname,  
       sum(UnitPrice * Quantity) as 'OrderValue'  
from nwEmployees E, nwOrders O,  
nwOrderDetails D  
where E.EmployeeID = O.EmployeeID  
       and O.OrderID = D.OrderID  
GROUP BY LastName, FirstName  
Order By 3 desc
```

- **Join types**

## **Explicit inner join**

```
SELECT *  
FROM employee e  
      INNER JOIN department d  
      ON e.DepartmentID = d.DepartmentID
```

## **Implicit inner join:**

```
SELECT * FROM employee e, department d  
      WHERE e.DepartmentID = d.DepartmentID
```

## Left Outer Join

```
SELECT * FROM employees e  
    LEFT OUTER JOIN department d  
    ON e.DepartmentID = d.DepartmentID
```

**Returns ALL rows from LEFT table and only matching rows from RIGHT table.**

## Right Outer Join

```
SELECT * FROM employees e  
    RIGHT OUTER JOIN department d  
    ON e.DepartmentID = d.DepartmentID
```

**Returns ALL rows from RIGHT table and only matching rows from LEFT table.**

## **Analysis using Outer Joins**

1. Are there some customers who have no orders?
2. Are there orders in the nwOrders table that have an invalid reference to a Northwinds customer?
3. Are there orders in the nwOrders table that have an invalid reference to a Northwinds employee?



## **Analyzing Orders and Customers.**

```
SELECT COUNT(customerid)FROM nwcustomers
```

– there are 87 customers in nwcustomers

```
SELECT COUNT(distinct customerid)FROM nworders
```

– there are 89 distinct customers in nworders

**Is my data corrupt?**

**What's going on here...?**

- **Invalid CustomerID?**

**We want to find the orders in nworders whose customerID is NOT in nwcustomers**

**Method One: use a subquery**

```
SELECT DISTINCT customerID
FROM nworders
WHERE customerID NOT IN (
    SELECT customerID FROM nwcustomers);
```

**Method Two: use an outer join**

```
SELECT DISTINCT O.customerID
FROM nworders O LEFT OUTER JOIN nwcustomers C
ON O.customerID = C.customerID
WHERE C.customerID IS NULL
```

**This shows us FOUR customers who have orders in nwOrders that have no matching row in nwCustomers**

## **But FIRST !**

### **A note on UNION**

- **Assemble multiple queries connected by a UNION**
- **Combine multiple answer sets**
- **Each answer set must have the same number of columns**
- **Each column in all answer sets must have the same domain**

## EquiJoin:

```
SELECT C.customerID, CompanyName, COUNT(orderID)
      FROM nworders O
JOIN nwcustomers C ON O.customerID = C.customerID
      GROUP BY C.customerID, CompanyName
UNION
SELECT 'Total', 'Total', COUNT(orderID)
      FROM nworders O JOIN nwcustomers C
      ON O.customerID = C.customerID;
```

This shows us a grand total of **785** orders

## Outer Join:

```
SELECT DISTINCT O.customerID, CompanyName, COUNT(orderID)
  FROM nworders O
LEFT OUTER JOIN nwcustomers C ON O.customerID = C.customerID
  GROUP BY O.customerID, CompanyName
UNION
SELECT 'Total', 'Total', COUNT(orderID)
  FROM nworders O LEFT OUTER JOIN nwcustomers C
  ON O.customerID = C.customerID;
```

This shows us a grand total of **829** orders

- **So what is the impact of this (that several orders have bad customerID's)?**
- **This shows a discrepancy in the number of orders**
  - 785 orders versus 829 orders

**We could do a similar analysis comparing total orders dollar amounts (joining nwOrders, nwOrderDetails and nwCustomers)**

**Discrepancies in dollar amounts would fail an audit !!!**

- **We want to find the customers in nwcustomers who have no orders in nworders**

## **Method One: use a subquery**

```
SELECT customerID
FROM nwcustomers
WHERE customerID NOT IN (
    SELECT DISTINCT customerID FROM nworders);
```

## **Method Two: use an outer join**

```
SELECT DISTINCT C.customerID
FROM nworders O RIGHT OUTER JOIN nwcustomers C
ON O.customerID = C.customerID
WHERE O.customerID IS NULL
```

**This shows us TWO customers who have no orders in nworders**

- **We want to find the employees in nwEmployees who have no orders in nwOrders**

## **Method One: use a subquery**

```
SELECT employeeID
FROM nwEmployees
WHERE employeeID NOT IN (
    SELECT DISTINCT employeeID FROM nworders);
```

## **Method Two: use an outer join**

```
SELECT DISTINCT e.employeeID
FROM nwEmployees E RIGHT OUTER JOIN nwOrders O
ON E.employeeID = O.employeeID
WHERE O.employeeID IS NULL
```

**This shows us that all employees have some orders.**



- **We want to find any orders in nwOrders that have an invalid reference to nwEmployees**

## **Method One: use a subquery**

```
SELECT DISTINCT EmployeeID
FROM nworders
WHERE employeeID NOT IN (
    SELECT employeeID FROM nwEmployees));
```

## **Method Two: use an outer join**

```
SELECT DISTINCT O.EmployeeID
FROM nwOrders O LEFT OUTER JOIN nwEmployees E
ON O.EmployeeID = E.employeeID
WHERE E.customerID IS NULL
```

**This shows us that all orders have valid employeeIDs.**

## **Join Execution Plans**

- With any join, the database engine optimizer must calculate the most efficient query execution plan. There are three basic methods:

**Nested Loop Join** – when one join input table has a small number of rows and the other input table is large and indexed on the join key

**Merge Join** – when tables being joined are both sorted on the join key

**Hash Join** – when large, unsorted, non-indexed inputs are joined with an inner join with an "=" condition.

## **Join Execution Plans**

- The “explain plan” option will show you an analysis of the execution plan calculation by the query optimizer.