

# Database Query: Advanced

IT 4153 Advanced Database

J.G. Zheng  
Spring 2012



# Overview

## ◆ Advanced query features

- Set operation: union, except, intersect
- Self join
- Derived tables and common table expression
- Functions

# Set Operations

- ◆ Set operations combine results from two or more queries into a single result set.
- ◆ Operands
  - UNION
  - EXCEPT
  - INTERSECT
- ◆ Common basic rules
  - The number and the order of the columns must be the same in all queries.
  - The data types must be compatible.
  - Ordering of the final result (ORDER BY) should be placed at the end of the whole statement

# UNION

## ◆ Combing query results (records) using "UNION"

- The number and the order of the columns must be the same in all queries.
- The data types must be compatible.
- Use "UNION ALL" to include duplidates

## ◆ Example

- The company is changing its business process; the customers and suppliers need to be notified. The director needs a combined list of their contacts.

```
SELECT CompanyName, ContactName, Phone, 'Customer' AS Type  
FROM Customers
```

```
UNION
```

```
SELECT CompanyName, ContactName, Phone, 'Supplier' AS Type FROM  
Suppliers
```

```
ORDER BY CompanyName;
```

The number of expressions, and their data types, from the two sets have to be exactly the same.

# EXCEPT and INTERSECT

## ◆ EXCEPT

- EXCEPT returns any distinct records from the query on the left side of the EXCEPT operator that are not returned by the query on the right side.

## ◆ INTERSECT

- INTERSECT returns any distinct records that are returned by both the query on the left and right sides of the INTERSECT operator.

# Self Join

- ◆ A self join is joining a table to itself
  - Use alias to differentiate tables

- ◆ Example

- List the direct manager name for each employee

```
SELECT e1.FirstName, e1.LastName, e2.FirstName, e2.LastName
FROM Employees AS e1 LEFT JOIN Employees AS e2
ON e1.ReportsTo = e2.EmployeeID
```

# Subquery: More

- ◆ Use the output of a “SELECT” query (sub-query, or inner query) as an input for another “SELECT” query
- ◆ A subquery can appear
  - anywhere an expression can be used, or if it returns a single value.
    - ◆ WHERE (=, IN, EXISTS), HAVING
    - ◆ As an expression in place where columns are usually placed.
  - As derived tables
    - ◆ FROM clause
- ◆ Rules
  - May only include an ORDER BY clause when a TOP clause is also specified.
  - <http://msdn.microsoft.com/en-us/library/ms189543.aspx>

# Sub-Query in WHERE

```
SELECT * FROM Products
WHERE CategoryId =
    (SELECT CategoryId FROM Categories
     WHERE CategoryName = 'Seafood');
```

The sub query returns a single value (scalar value); use "="

```
SELECT * FROM Products
WHERE CategoryId IN
    (SELECT CategoryId FROM Categories
     WHERE CategoryName IN 'Seafood','Beverages','Produce');
```

The sub query returns a list of values; use IN

# Sub-Query and Table Join

- ◆ In the previous cases these statements can also be re-written as table joins

```
SELECT * FROM Products, Categories  
WHERE Products.CategoryId = Categories.CategoryId  
AND CategoryName = 'Seafood';
```

```
SELECT * FROM Products, Categories  
WHERE Products.CategoryId = Categories.CategoryId  
AND CategoryName IN ('Seafood','Beverages','Produce');
```

# Sub-Queries for Comparison

- ◆ Sub-queries can be used with other comparison operators  $>$ ,  $<$ ,  $>=$ ,  $<=$ , etc.

```
SELECT * FROM Products
```

```
WHERE UnitPrice >
```

```
(SELECT AVG(UnitPrice) FROM Products);
```

The sub query returns a single value (scalar value)

- ◆ In these cases, there is no equivalent table join format

# Subquery for Derived Table

- ◆ Who are some of biggest customers in terms of sales?

```
select top 10 CompanyName, Total from Customers,  
    (select CustomerID, SUM(Quantity*UnitPrice) as Total  
     from [Order Details] inner join Orders on  
     orders.OrderID = [Order Details].OrderID  
     group by CustomerID) as T2  
where Customers.CustomerID = T2.CustomerID  
order by Total desc;
```

# Common Table Expression

- ◆ A common table expression (CTE) can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.
- ◆ A CTE is similar to a derived table in that it is not stored as an object and lasts only for the duration of the query. Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query.
- ◆ A CTE can be used to:
  - Substitute for a view when the general use of a view is not required; that is, you do not have to store the definition in metadata.
  - Reference the resulting table multiple times in the same statement.

# CTE Syntax

- ◆ The basic syntax structure for a CTE

```
WITH CTE_name [ ( column_name [,...n] ) ]  
AS  
( CTE_query_definition )  
SELECT <column_list>  
FROM CTE_name;
```

//The list of column names is optional only if distinct names for all resulting columns are supplied in the query definition.

# CTE Example

◆ Who are some of biggest customers in terms of sales?

with T2

as (select CustomerID, SUM(Quantity\*UnitPrice) as Total  
from [Order Details] inner join Orders on  
orders.OrderID = [Order Details].OrderID  
group by CustomerID)

select top 3 CompanyName, Total from Customers,T2  
where Customers.CustomerID = T2.CustomerID  
order by Total desc;

# Functions

## ◆ Date

- <http://msdn.microsoft.com/en-us/library/ms186724.aspx>

## ◆ Math

- <http://msdn.microsoft.com/en-us/library/ms177516.aspx>

## ◆ String functions

- <http://msdn.microsoft.com/en-us/library/ms181984.aspx>

# More SQL Resources

## ◆ Set operations

- <http://msdn.microsoft.com/en-us/library/ms191523.aspx>

## ◆ CTE

- <http://msdn.microsoft.com/en-us/library/ms190766.aspx>

## ◆ T-SQL functions

- <http://msdn.microsoft.com/en-us/library/ms174318.aspx>