

Advanced SQL SELECT

CIS 3730

Designing and Managing Data

J.G. Zheng
Fall 2010



Overview

◆ More on SQL SELECT queries

1. SQL expressions
2. Aggregate functions with grouping
3. More table join types and options
4. Sub-queries
5. Alias
6. Top, distinct, union

1. Expression

- ◆ Expression is a combination of symbols and operators that returns a single value

- ◆ An expression can be
 - a single constant, variable, column, or scalar function
 - ◆ 10
 - ◆ 3.14
 - ◆ 'John Doe'
 - ◆ '10/10/2010'
 - ◆ CompanyName
 - ◆ GetDate()
 - columns, numbers, literals, functions connected by operators
 - ◆ $10 * 3 + 3$
 - ◆ 'John' + 'Doe'
 - ◆ Address + ', ' + City + ', ' + ZipCode
 - ◆ Quantity * SalePrice
 - ◆ $1.2 * ListPrice$

1.1 Expressions used as Columns

◆ Expressions can be used as derived columns

◆ Examples

A text constant which will
the same for every record

SELECT CompanyName, 'Supplier' as Type FROM Suppliers;

SELECT ProductName, UnitPrice*1.2 AS 'New Price'
FROM Products;

String concatenation

SELECT FirstName+' '+LastName FROM Employees;

SELECT UPPER(ProductName) FROM Products;

SELECT GETDATE();

A system function not
related to any table.


1.2 Expressions used for Comparison

◆ Expressions can be used in the WHERE clause

◆ Examples

```
SELECT * from Products  
WHERE UnitsInStock-ReorderLevel<0;
```

Comparison
between columns



```
SELECT * FROM Products  
WHERE UnitsInStock + UnitsOnOrder < ReorderLevel;
```

```
SELECT ProductName, UnitPrice * 1.1 AS Discount  
FROM Products  
WHERE UnitPrice * 1.1 >= 20
```

2. Aggregate Functions

- ◆ Using aggregate functions for row calculation
 - MIN (minimum of all or selected values)
 - MAX (maximum of all or selected values)
 - COUNT (number of all or selected rows)
 - AVG (average of all or selected values)
 - SUM (sum of all or selected values)

◆ 2.1 Calculation for all or selected rows

```
SELECT COUNT(ProductID) AS NumberOfProducts FROM Products;
```

```
SELECT AVG(UnitPrice) AS 'Average Price for Category 1'  
FROM Products  
WHERE CategoryId = 1;
```

This criterion limits the records to be averaged.

Expression in Aggregation

- ◆ Expressions can be used with aggregate functions

- ◆ Example

- What is the total payment for each line item?

```
SELECT SUM(UnitPrice * Quantity) AS LineItemTotal  
FROM [Order Details];
```

2.2 Grouping

- ◆ **GROUP BY: aggregation with groups**
 - To get aggregation results for different groups of records
- ◆ **Example**
 - What is the average unit price of products in each category?

```
SELECT CategoryID, AVG(UnitPrice)
FROM Products
GROUP BY CategoryID;
```

Limitations of GROUP BY

- ◆ Columns or expressions (except the aggregate function) can be in the SELECT clause only if they are in the GROUP BY clause.

Good! Country is in the GROUP BY clause

```
SELECT Country, Region, COUNT(CustomerId)
FROM Customers
GROUP BY Country
```

WRONG! Region is not in the GROUP BY clause

2.3 Sorting Aggregation Result

- ◆ You can sort by the aggregation results
 - Example: what is the average unit price of products in each category? Sort by the average unit price

```
SELECT CategoryID, AVG(UnitPrice) FROM Products  
GROUP BY CategoryID  
ORDER BY AVG(UnitPrice)
```

Using aggregate function

```
SELECT CategoryID, AVG(UnitPrice) AS Price FROM Products  
GROUP BY CategoryID  
ORDER BY Price
```

Using alias.

```
SELECT CategoryID, AVG(UnitPrice) AS Price FROM Products  
GROUP BY CategoryID  
ORDER BY 2
```

Using column index

2.4 Filtering Aggregation Result

- ◆ Use "HAVING" clause to filter aggregation result (after aggregation)
 - What is the average unit price of products in each category? Only return those with an average price greater than 10.
 - **Important: aggregate functions cannot be used in WHERE clause!**

```
SELECT CategoryID, AVG(UnitPrice) FROM Products
GROUP BY CategoryID
Having AVG(UnitPrice) > 10;
```

This is **wrong**:
SELECT CategoryID, AVG(UnitPrice)
FROM Products
WHERE AVG(UnitPrice) > 10
GROUP BY CategoryID

- ◆ Use "WHERE" clause to filter records to be aggregated (before aggregation)
 - What is the average unit price of products whose unit price is greater than 10?

```
SELECT CategoryID, AVG(UnitPrice) FROM Products
WHERE UnitPrice > 10
GROUP BY CategoryID;
```

3. More Table Join Types

◆ Inner join (equal join)

- Only include records that have matching records (based on PK/FK pair) from two tables (either direction)
- Records that do not have matching ones in the other table are not included in the results.

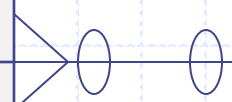
◆ Outer join (outer joins are usually needed when minimum cardinality is optional on a table)

- Left join: include all qualified records from the left table in the join condition even if they do not have matching records in the right table.
- Right join: include all qualified records from the right table in the join condition even if they do not have matching records in the left table.
- Full join: include all qualified records from both tables in the join condition

Table Join Effect

Product	CategoryId
A	1
B	2
C	

CategoryId	Name
1	Canned
2	Drink
3	Fresh



Product	Category
A	Canned
B	Drink

Inner join

Product	Category
A	Canned
B	Drink
C	(Null)

Left Join

Product	Category
A	Canned
B	Drink
C	(Null)
(Null)	Fresh

Full Join

Product	Category
A	Canned
B	Drink
(Null)	Fresh

Right Join

Outer Join Example

- ◆ Get customers and their orders; also include customers who have never placed an order

```
SELECT CompanyName, OrderID
FROM
Customers LEFT JOIN Orders ON
    Customers.CustomerID = Orders.CustomerID
ORDER BY OrderID
```

- ◆ Execution result

	CompanyName	OrderID
1	Paris spécialités	NULL
2	FISSA Fabrica Inter. Salchichas S.A.	NULL
3	Wilman Kala	10248
4	Tradição Hipermercados	10249
5	Henri Carnes	10250

The first two rows will not be included for an inner join.

4. Sub-Query

- ◆ Use the output of a "SELECT" query (sub-query, or inner query) as an input for another "SELECT" query

```
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

4.1 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');
```

4.2 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

5. Alias

- ◆ Column alias: representing derived and constant columns

```
SELECT CategoryID, AVG(UnitPrice) Price
FROM Products
GROUP BY CategoryID
ORDER BY Price;
```

Column alias can be used in ORDER BY

```
SELECT ProductName, UnitPrice * 0.9 Discount
FROM Products
WHERE UnitPrice * 0.9 > 20;
```

Column alias can NOT be used in WHERE or HAVING clause (SQL Server)

- ◆ Table alias: commonly used in table joins

```
SELECT ProductName, CategoryName
FROM Products AS p, Categories c
Where p.CategoryID = c.CategoryID
```

"AS" is optional.

If an alias is assigned, it must be used instead of the original table name

Alias Variations

◆ Column alias: use [] or ' '

```
SELECT ProductName, UnitPrice * 0.9 AS 'Discount Price'  
FROM Products  
ORDER BY 'Discount Price';
```

```
SELECT ProductName, UnitPrice * 0.9 AS [Discount Price]  
FROM Products  
ORDER BY [Discount Price];
```

◆ Table alias: use []

```
SELECT ProductName, CategoryName  
FROM Products AS [table p], Categories c  
Where [table p].CategoryID = c.CategoryID
```

6.1 Uniqueness

- ◆ Use the keyword "DISTINCT" to eliminate duplicate rows in the results

- ◆ Example

Without DISTINCT, it returns 91 rows; with DISTINCT, it returns only 21 rows.

```
SELECT DISTINCT Country from Customers  
ORDER BY country;
```

```
SELECT Count(DISTINCT Country)  
FROM Customers
```

DISTINCT can be used with aggregate functions. Without DISTINCT, the result is 91; with DISTINCT, the result is 21.

6.2 TOP

◆ Use the keyword "TOP" to limit the number of rows returned

◆ Example

Only returns 10 records.

SELECT TOP 10 * FROM Customers;

SELECT TOP 5 PERCENT * FROM Customers;

Only returns 5% of the total records in the original results.

6.3 UNION

◆ Combing query results (records) using “UNION”

◆ 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;
```

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

Summary

- ◆ Key concepts
 - Expression
 - Aggregate function
 - Outer join: left join, right join, full join
 - Sub-query
 - Alias

- ◆ Key skills: write SQL SELECT statement to retrieve desired data, and know the result of a given SQL SELECT statement involving
 - Expressions
 - Aggregate functions with groups
 - Outer joins
 - Sub-queries
 - Alias
 - Top, distinct, union

More SQL Resources

- ◆ W3Schools SQL Tutorial
 - <http://www.w3schools.com/sql/>
- ◆ SQL Course
 - <http://sqlcourse2.com/>
- ◆ A gentle introduction to SQL
 - <http://sqlzoo.net/>
- ◆ Other
 - <http://www.1keydata.com/sql>