Departement Informatika
Department of Informatics

MS SQL 2008

Lecture 4 | Data retrieval
Chapter 8 (Part 1) | Hotek, 2008

Learning objectives

- Admin
- Enforcing data integrity
- Retrieving data from single tables
- Filtering and sorting query results

Admin

- Consultation can be done for issues not understood
- SQL like any other programming language requires that you consult the textbook, Help and other tutorial materials over and above class notes
- Check out for some extra guides:
  - http://www.w3schools.com/sql/default.asp

Enforcing data integrity – Primary Key

- Using the primary key constraint
  - Single column or concatenated
  - Primary keys defined as constraints are automatically made unique:

```
CREATE TABLE Blu_Ray_Category
(Bluray_ID       INT,
Bluray_Initials     CHAR(3),
Bluray_Name      varchar(50),
Bluray_Genre     varchar(50),
CONSTRAINT pk_blurraycategory PRIMARY KEY (Bluray_ID, Bluray_Initials))
```

Enforcing data integrity – Unique Key

- Using the unique key constraint
  - Ensures unique values in the column for example, passport numbers and ID numbers

```
CREATE TABLE Customer
(Customer_ID    INT PRIMARY KEY,
Customer_FirstName     varchar(50),
Customer_LastName     varchar(50),
Customer_Address     varchar(50),
Customer_RDAIDNumber     varchar(50)    UNIQUE)
```

Enforcing data integrity – Foreign Key

- Using the foreign keys – referential integrity
  - ON DELETE or ON UPDATE clause controls what actions are taken when you try to delete a row to which existing foreign keys point
    - NO ACTION specifies that the deletion fails with an error.
    - CASCADE specifies that all the rows with foreign keys pointing to the deleted row are also deleted.
    - SET NULL specifies that all rows with foreign keys pointing to the deleted row are set to NULL.
    - SET DEFAULT specifies that all rows with foreign keys pointing to the deleted row are set to their default value. For more information, see Defaults.

Enforcing data integrity – Foreign Key

- Using the foreign keys – referential integrity

```
CREATE TABLE Video_orders
(Video_order_ID      int    PRIMARY KEY,
Serial_number    INT FOREIGN KEY REFERENCES
VIDEO(Serial_number) ON DELETE NO ACTION,
Qty_ordered      int);
GO
```

Ensuring data integrity – CHECK constraint

- CHECK constraints enforce domain integrity by limiting the values that can be put in a column

```
CREATE TABLE cust_sample
(cust_id int PRIMARY KEY,
cust_name char(50),
cust_address char(50),
cust_credit_limit money,
CONSTRAINT chk_id CHECK (cust_id BETWEEN 0 and 10000 ) )
```

Ensuring data integrity–NOT NULL constraint

- NOT NULL specifies that the column does not accept NULL values

The SELECT statement

- Retrieves rows from the database and enables the selection of one or many rows or columns from one or many tables.
- The full syntax of the SELECT statement is complex, but the main clauses can be summarized as…

SELECT Statement main syntax

- [WITH <common_table_expression>]
- SELECT *select_list* [INTO *new_table*]
- [FROM *table_source*] [WHERE *search_condition*]
- [GROUP BY *group_by_expression*]
- [HAVING *search_condition*]
- [ORDER BY *order_expression* [ASC | DESC]]

SELECT clause

For example in a basic form:

```
SELECT 'This is my name', 'and then I want', 'some milk',
'10 March 2010'
go
```

To retrieve data from a table…

- SELECT <column_names>
- FROM <table_name>

```
SELECT Serial_number, Category_ID, Video_Title
FROM VIDEO
```

```
SELECT *
FROM VIDEO
```

Renaming a Column Name for display

- These are referred to as aliases using 3 methods
  - *table_name* AS *table_alias*
  - *table_name table_alias*
  - *Table_alias* = <*column*>

SELECT MyNumber=Serial_number, Category_ID AS 'Category ID', Video_Title 'Video Title'
FROM VIDEO

Data manipulation

- When expressions need to be converted from one data type to another.
  - CAST

  ```
  CAST ( expression AS data_type [ (length ) ])
  ```

  - CONVERT

  ```
  CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
  ```

CAST and CONVERT

```
USE AdventureWorks
go

SELECT
   GETDATE() AS UnconvertedDateTime,
   CAST(GETDATE() AS nvarchar(30)) AS UsingCast,
   CONVERT(nvarchar(30), GETDATE(), 126) AS
UsingConvertTo_ISO8601  ;
GO
```

ISNULL and COALESCE

```
SELECT AddressID, AddressLine1, AddressLine2, AddressLine1
+ ' ' + AddressLine2 AS Address, City, StateProvinceID,
PostalCode
FROM Person.Address
GO
```

Adding NULL values to anything produces NULL values

```
SELECT AddressID, AddressLine1 + ' ' +
ISNULL(AddressLine2,"), City, StateProvinceID
FROM Person.Address
```

COUNT and MAX

- COUNT returns the number of items in a group

```
SELECT COUNT(*)
FROM Person.Address
GO
```

- MAX returns the maximum value in the expression

```
SELECT MAX(TaxRate)
FROM Sales.SalesTaxRate
GO
```

CASE function

- The CASE expression evaluates a list of conditions and returns one of multiple possible result expressions
  - The simple CASE expression compares an expression to a set of simple expressions to determine the result.
  - The searched CASE expression evaluates a set of Boolean expressions to determine the result.

- Both formats support an optional ELSE argument.

Simple CASE expression

CASE *input_expression*

  WHEN *when_expression* THEN *result_expression* [...*n*]

  [ELSE *else_result_expression*]

END

For example compare these two:

```
SELECT ProductNumber, ProductLine, Name
FROM Production.Product
```

```
USE AdventureWorks;
GO
SELECT   ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
GO
```

Searched CASE expression

```
CASE
    WHEN Boolean_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

For example:

```
USE AdventureWorks;
GO
SELECT ProductNumber, Name, 'Price Range' =
CASE
WHEN ListPrice = 0 THEN 'Mfg item - not for resale'
WHEN ListPrice < 50 THEN 'Under $50'
WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under $1000'
ELSE 'Over $1000'
END
FROM Production.Product
ORDER BY ProductNumber
GO
```

SORTING Results – ORDER BY

- ORDER BY clause specifies the sort order used on columns returned in a SELECT statement
- Simple syntax is:

```
[ ORDER BY
    {
    order_by_expression
   [ ASC | DESC ]
    } [ ,...n ]
]
```

For example:

```
USE AdventureWorks
GO
SELECT ProductID, Name
FROM Production.Product
WHERE Name LIKE 'Lock Washer%'
ORDER BY ProductID, Name DESC;
```

FILTERING data - WHERE

- Specifies the search condition for the rows returned by the query
- Syntax:
  - [WHERE <search_condition>]
- Equality condition can be replaced with:
- >, <, >=, <=, <> OR !=

```
USE AdventureWorks
GO
SELECT ProductID, Name
FROM Production.Product
WHERE Name = 'Blade' ;
GO
```

Using compound search conditions with WHERE

- The search condition is a combination of one or more predicates that use the logical operators AND, OR, and NOT
- Others include:
  - !> used to test the condition of one expression not being greater than the other expression
  - [NOT] LIKE indicates that the subsequent character string is to be used with pattern matching, Used with % and _
  - [NOT] BETWEEN specifies an inclusive range of values
  - EXISTS is used with a subquery to test for the existence of rows returned by the subquery

Example

```
USE AdventureWorks ;
GO
SELECT *
FROM Production.ProductPhoto
WHERE LargePhotoFileName LIKE '%greena_%'
ESCAPE 'a' ;
```

Exercise

- Run an SQL script which will create the MusicologyWarehouse database and populate it with data
- Create 7 queries to retrieve different sorts of data