SQL Server

Database Development

(Based on MS SQL Server 2005 & 2008)

Contents

Contents...

- WHERE Clause
- AND, OR operations
- ORDER BY Keyword
- INSERT INTO Statement
- UPDATE Statement
- DELETE Statement
- TOP Clause
- SQL LIKE Operator
- SQL Wildcards
- The IN Operator
- SQL BETWEEN Operator
- Joins and Set Operations

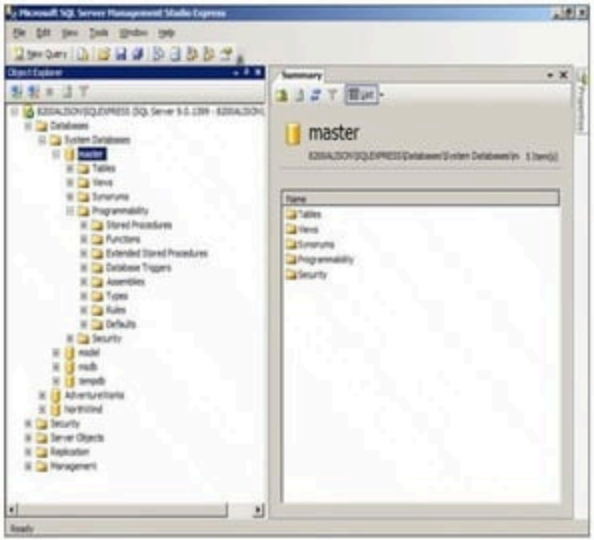
Contents...

Introduction

- SQL - Structured Query Language
  - The standard for relational database management systems (RDBMS)
- Microsoft SQL Server is a relational database management system.
- It is a software product whose primary function is to store and retrieve data

SQL Server Objects

- System Objects
  - System databases include Master, Model, MSDB, Resource, TempDB, and Distribution
  - SQL Server creates these databases during the installation process
  - There are also system



SQL Server Objects…

- User Objects
  - User objects include the databases, stored procedures, functions, and other database objects
  - You can add and drop user objects as necessary

SQL Server Database

- Database is a container for objects
  - Store data
  - Enable data storage and retrieval in a secure and safe manner
- A SQL Server 2008 database can hold the following
  - Table definitions
  - Columns within those tables, which make up rows of data
  - Programs (either stored procedures written using T-SQL or assemblies) used to access or manipulate the data
  - Indexes, which are used to speed up the retrieval of data
  - Views, which are specialized ways of looking at the actual data
  - Functions, which are repetitive tasks that can be applied to

Table

- Is a repository for data
- Items of data grouped in one or more columns
- Tables contain zero or more rows of information

app_id	app_cd	app_nm	app_description	create_user_id	update_user_id	create_dt	action_dt
1	APP1	Bluechip	Core Bluechip A...	1	1	2011-07-24 17:...	2011-07-24 17:06:00
2	APP2	Security	Security Module	1	1	2011-07-24 17:...	2011-07-24 17:06:00
3	APP3	SMS	SMS Module	1	1	2011-07-24 17:...	2011-07-24 17:07:00
4	APP4	NITA	NITA	1	1	2011-07-24 17:...	2011-07-24 17:07:00
5	APP5	Email	*NULL*	1	0	2011-07-25 17:...	2011-07-25 17:06:00
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Table...

- Create Table
  - Syntax/Example:

```
CREATE TABLE [dbo].[hrm_grade](
        [grade_cd] [tinyint] NOT NULL,
        [grade_nm] [varchar](50) NOT NULL,
        [usr_id] [smallint] NOT NULL,
        [action_tp] [bit] NOT NULL,
        [action_dt] [smalldatetime] NOT NULL,
        [rowguid] [uniqueidentifier] ROWGUIDCOL  NOT NULL,
    CONSTRAINT [PK_hrm_grade] PRIMARY KEY CLUSTERED
        (
            [grade_cd] ASC
```

SQL Server Data Types

Data Type Name	Class	Size in Bytes	Nature of the Data
`Bit`	Integer	1	The size is somewhat misleading. The first `bit` data type in a table takes up 1 byte; the next 7 make use of the same byte. Allowing nulls causes an additional byte to be used.
`Bigint`	Integer	8	This just deals with the fact that we use larger and larger numbers on a more frequent basis. This one allows you to use whole numbers from -2^{63} to $2^{63}-1$. That's plus or minus about 92 quintrillion.
`Int`	Integer	4	Whole numbers from –2,147,483,648 to 2,147,483,647.

SQL Server Data Types...

Data Type Name	Class	Size in Bytes	Nature of the Data
`SmallInt`	Integer	2	Whole numbers from –32,768 to 32,767.
`TinyInt`	Integer	1	Whole numbers from 0 to 255.
`Decimal` or `Numeric`	Decimal/ Numeric	Varies	Fixed precision and scale from $-10^{38}-1$ to $10^{38}-1$. The two names are synonymous.
`Money`	Money	8	Monetary units from -2^{63} to 2^{63} plus precision to four decimal places. Note that this could be any monetary unit, not just dollars.
`SmallMoney`	Money	4	Monetary units from –214,748.3648 to +214,748.3647.
`Float` (also a synonym for ANSI `Real`)	Approximate Numerics	Varies	Accepts an argument (for example, `Float(20)`) that determines size and precision. Note that the argument is in bits, not bytes. Ranges from –1.79E + 308 to 1.79E + 308.
`DateTime`	Date /Time	8	Date and time data from January 1, 1753, to December 31, 9999, with an accuracy of three hundredths of a second.
`DateTime2`	Date /Time	Varies (6–8)	Updated incarnation of the more venerable DateTime data type. Supports larger date ranges and large time-fraction precision (up to 100 nanoseconds). Like DateTime, it is not time zone aware, but does align with the .NET DateTime data type.
`SmallDateTime`	Date /Time	4	Date and time data from January 1, 1900, to June 6, 2079, with an accuracy of one minute.
`DateTimeOffset`	Date /Time	Varies (8–10)	Similar to the DateTime data type, but also expects an offset designation of –14:00 to +14:00 offset from UTC time. Time is stored internally as UTC time, and any comparisons, sorts, or indexing will be based on that unified time zone.

SQL Server Data Types...

Data Type Name	Class	Size in Bytes	Nature of the Data
Date	Date/Time	3	Stores only date data from January 1, 0001, to December 31, 9999, as defined by the Gregorian calendar. Assumes the ANSI standard date format (YYYY-MM-DD), but will implicitly convert from several other formats.
Time	Date/Time	Varies (3–5)	Stores only time data in user-selectable precisions as granular as 100 nanoseconds (which is the default).
Cursor	Special Numeric	1	Pointer to a cursor. While the pointer takes up only a byte, keep in mind that the result set that makes up the actual cursor also takes up memory. Exactly how much will vary depending on the result set.
Timestamp/ rowversion	Special Numeric (binary)	8	Special value that is unique within a given database. Value is set by the database itself automatically every time the record is either inserted or updated, even though the timestamp column wasn't referred to by the UPDATE statement (you're actually not allowed to update the timestamp field directly).
UniqueIdentifier	Special Numeric (binary)	16	Special Globally Unique Identifier (GUID) is guaranteed to be unique across space and time.
Char	Character	Varies	Fixed-length character data. Values shorter than the set length are padded with spaces to the set length. Data is non-Unicode. Maximum specified length is 8,000 characters.
VarChar	Character	Varies	Variable-length character data. Values are not padded with spaces. Data is non-Unicode. Maximum specified length is 8,000 characters, but you can use the "max" keyword to indicate it as essentially a very large character field (up to 2^31 bytes of data).
Text	Character	Varies	Legacy support as of SQL Server 2005. Use varchar(max) instead!

SQL Server Data Types...

Data Type Name	Class	Size in Bytes	Nature of the Data
NChar	Unicode	Varies	Fixed-length Unicode character data. Values shorter than the set length are padded with spaces. Maximum specified length is 4,000 characters.
NVarChar	Unicode	Varies	Variable-length Unicode character data. Values are not padded. Maximum specified length is 4,000 characters, but you can use the "max" keyword to indicate it as essentially a very large character field (up to 2^31 bytes of data).
NText	Unicode	Varies	Variable-length Unicode character data. Like the Text data type, this is legacy support only. In this case, use nvarchar(max).
Binary	Binary	Varies	Fixed-length binary data with a maximum length of 8,000 bytes
VarBinary	Binary	Varies	Variable-length binary data with a maximum specified length of 8,000 bytes, but you can use the "max" keyword to indicate it as essentially a LOB field (up to 2^31 bytes of data).
Image	Binary	Varies	Legacy support only as of SQL Server 2005. Use varbinary(max) instead!
Table	Other	Special	This is primarily for use in working with result sets, typically passing one out of a User-Defined Function or as a parameter for stored procedures. Not usable as a data type within a table definition (you can't nest tables).
HierarchyID	Other	Special	Special data type that maintains hierarchy-positioning information. Provides special functionality specific to hierarchy needs. Comparisons of depth, parent/child relationships, and indexing are allowed. Exact size varies with the number and average depth of nodes in the hierarchy.

Introduction to Relationships

- Three types of relationships can exist between tables in a database
  - One-to-many
  - One-to-one
  - Many-to-many
- The right type of relationship between two tables ensures
  - Data integrity
  - Optimal performance
  - Ease of use in designing system objects

Introduction to Relationships…

- One to Many
  - A record in one table can have many related records in another table
- One to One
  - Each record in the table on the one side of the relationship can have only one matching record in the table on the other side of the relationship
- Many to Many
  - Each record in the table on the one side of the relationship can have only one or many matching record in the table on the other side of the

Constraints

- Constraints limit or control the types of data that the user can enter into tables
- There are seven main categories of constraints.
  - Primary key constraints
  - Foreign key constraints
  - Default constraints
  - Not null constraints
  - Check constraints
  - Rules
  - Unique constraints.

Constraints...

- Primary Key Constraints
  - Primary key constraint is a column or a set of columns that uniquely identify a row in the table
  - Can be more than one field as the primary key
  - Syntax/Example:

```
ALTER TABLE Authors
ADD CONSTRAINT pk_authors PRIMARY KEY (AuthorID)
```

Constraints...

- Foreign Key Constraints
  - Foreign key constraint consists of a column or of a set of columns that participates in a relationship with a primary key table.
  - Primary key is on the one side of the relationship, whereas the foreign key is on the many side of the relationship.
  - Syntax/Example:

```
ALTER TABLE Orders

ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

Constraints...

- Default Constraints
  - Default constraint is a value that SQL Server automatically places in a particular field in a table.
  - Default value can be a constant, Null, or a function
  - Syntax/Example:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'SANDNES'
```

Constraints…

- Not null Constraints
  - Require the user to enter data into a field
  - Syntax/Example:

```
ALTER TABLE ConstraintTable
ALTER COLUMN ID INT NOT NULL
```

- **Check Constraints**
  - Check constraints limit the range of values that a user can enter into a column.
  - Can enter as many check constraints as you want for a particular column.
  - Syntax/Example:

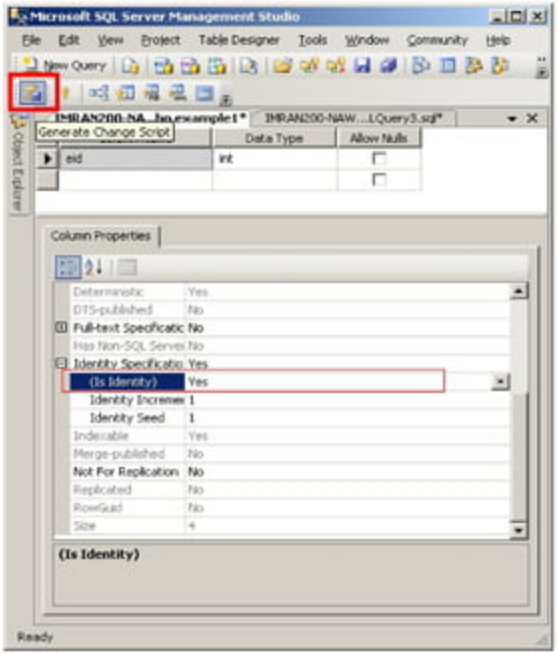
```
ALTER TABLE Person.Address
```

Constraints...

- Unique Constraints
  - Unique constraint requires that each entry in a particular column be unique
  - Syntax/Example:
    - ALTER TABLE Authors ADD CONSTRAINT IX_Authors_Name UNIQUE(Name)

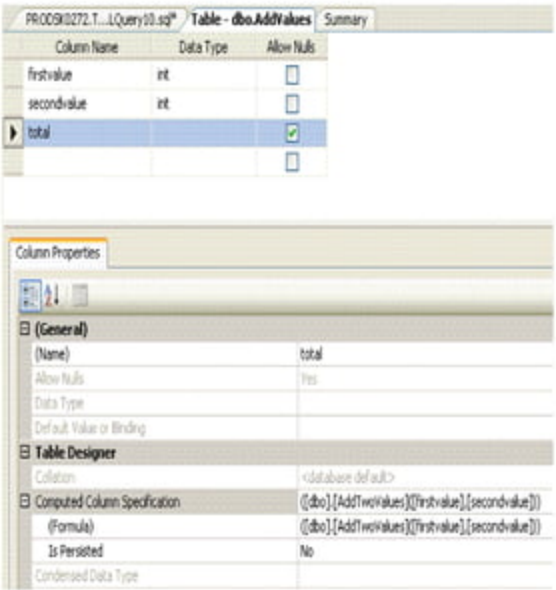
Identity Columns

- Provides an auto incrementing value for a table
- Should use an identity column as the primary key field for any table that has no natural primary key that is short, stable, and simple
- Identity columns are



Computed Columns

- With computed columns, you can create a column that is based on data in other columns.
- SQL Server automatically updates the computed column when the columns on which it depends are updated.



Indexes

- Indexes to improve performance when the user searches a field
- It's generally best to include too many indexes rather than too few
- Indexes speed up searching, sorting, and grouping data
- The downside is that they take up hard disk space and slow the process of editing, adding, and deleting data

Indexes...

- Syntax/Example:
  - **Single non clustered index**

```
CREATE UNIQUE NONCLUSTERED INDEX IX_NC_PresidentNumber
  ON dbo.Presidents (PresidentNumber)
```

  - **Multi-column (composite) clustered index**

```
CREATE UNIQUE CLUSTERED INDEX IX_C_PresidentNumber
  ON dbo.Presidents (PresidentNumber,PresidentName)
```

View Basics

- A view is a virtual Table
- Its contents are based on query
- Like a table, a view is composed of rows and columns
- Data in a view comes from one or more tables or views in the database
  - Syntax: {CREATE} {ALTER} VIEW view_name [(column [,... *n*])] [WITH < view_option > [,... *n*]] AS select_statement [WITH CHECK OPTION]
  - Example:

```
Create View dbo.vInventoryCost WITH SCHEMABINDING

as select ET.EqType, e.Make, e.Model, Sum(Cost) TotalCost, Count)*) [Count] from
dbo.Inventory I inner join dbo.Equipment e on i.EqId = e.EqId

inner join dbo.EqType ET on e.EqTypeId = ET.EqTypeId
```

SQL DML and DDL

- Data Manipulation Language (DML)
  - **SELECT** - extracts data from a database
  - **UPDATE** - updates data in a database
  - **DELETE** - deletes data from a database
  - **INSERT INTO** - inserts new data into a database

- Data Definition Language (DDL)
  - **CREATE DATABASE** - creates a new database
  - **ALTER DATABASE** - modifies a database
  - **CREATE TABLE** - creates a new table
  - **ALTER TABLE** - modifies a table
  - **DROP TABLE** - deletes a table

SELECT Statement

- The most basic select statement retrieve all rows and all columns from a table.

  SELECT * FROM Persons

  or

  SELECT LastName, FirstName FROM Persons

- Best Practices
  - Breaking line for readability
  - It is good idea to place comma at the beginning of the next line.

DISTINCT Statement

- The DISTINCT keyword can be used to return only distinct (different) values

```
SELECT DISTINCT column_name(s)
FROM table_name
```

```
SELECT DISTINCT City FROM Persons
```

WHERE Clause

- The WHERE clause is used to extract only those records that fulfill a specified criterion

```
SELECT * FROM dbo.hrm_employee
WHERE marital_stat = 'U'
```

AND & OR Operators

- The AND operator displays a record if both the first condition and the second condition are true.

```
SELECT * FROM dbo.hrm_employee
WHERE marital_stat = 'U' AND unit_cd = 100
```

- The OR operator displays a record if either the first condition or the second condition is true.

ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

INSERT INTO Statement

- The INSERT INTO statement is used to insert a new row in a table.
- It is possible to write the INSERT INTO statement in two forms.

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

```
INSERT INTO table_name (column1, column2,
column3,...) VALUES (value1, value2, value3,...)
```

UPDATE Statement

- The UPDATE statement is used to update existing records in a table.

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND
FirstName='Jakob'
```

DELETE Statement

- The DELETE statement is used to delete rows in a table

```
DELETE FROM table_name
WHERE some_column=some_value
```

```
DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob‘
```

```
DELETE FROM table_name
or
DELETE * FROM table_name
```

TOP Clause

- The TOP clause is used to specify the number of records to return

```
SELECT TOP 1 * FROM employee_info
```

SQL LIKE Operator

- LIKE operator is used to search for a specified pattern in a column.
  - SELECT * FROM Persons
    WHERE City LIKE 's%‘

- The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

SQL Wildcards

- SQL wildcards can substitute for one or more characters when searching for data in a database

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist]or [!charlist]	Any single character not in charlist

SQL Wildcards…

- SELECT * FROM Persons
  WHERE FirstName LIKE '_la‘
- SELECT * FROM Persons
  WHERE LastName LIKE 'S_end_on‘
- SELECT * FROM Persons
  WHERE LastName LIKE '[bsp]%‘
- SELECT * FROM Persons
  WHERE LastName LIKE '[!bsp]%'

The IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- SELECT column_name(s)
  FROM table_name
  WHERE column_name IN (value1,value2,...)
- SELECT * FROM Persons
  WHERE LastName IN ('Hansen','Pettersen')

SQL BETWEEN Operator

- The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

SQL Alias

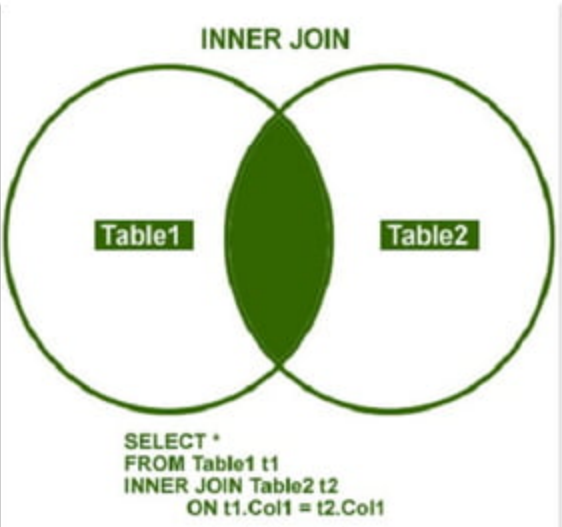
- With SQL, an alias name can be given to a table or to a column.
  - SELECT column_name(s)
    FROM table_name
    AS alias_name
  - SELECT column_name AS alias_name
    FROM table_name

Joins and Set Operations

- Joins
  - Joins are operations that allow you to match rows between tables.
  - Different types of Joins are
    - Inner Join
    - Outer Join
    - Right Outer Join
    - Full Outer Join
    - Cross Join

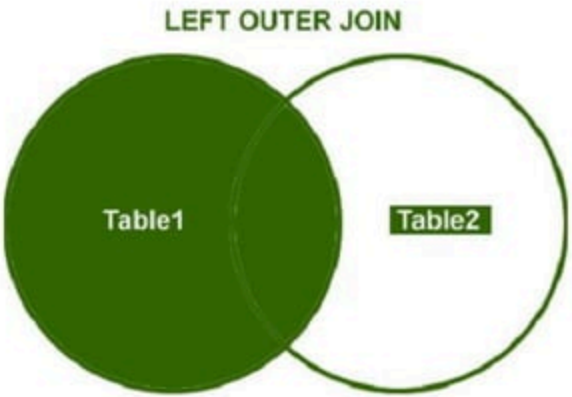
INNER JOIN

- INNER JOIN returns rows when there is at least one match in both the tables
- Returns matches data between table



OUTER JOIN

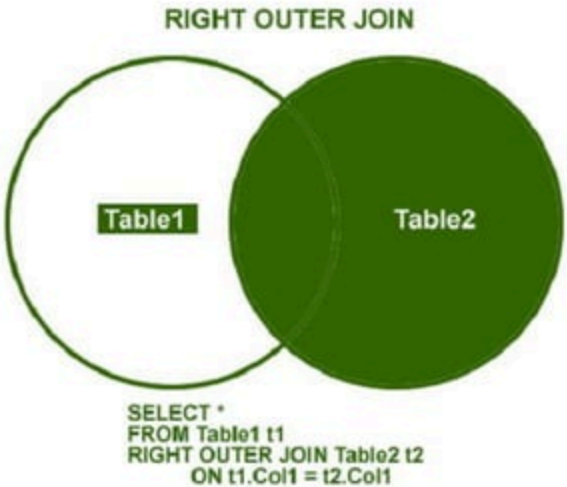
- **LEFT OUTER JOIN**
  - Returns all the rows from the left table in conjunction with the matching rows from the right table.
  - If there are no columns matching in the right table, it returns NULL values.




```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
    ON t1.Col1 = t2.Col1
```

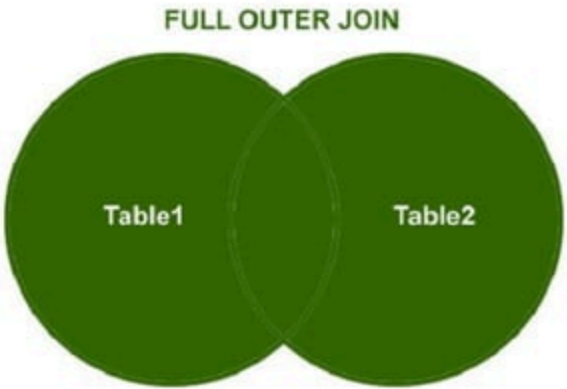
OUTER JOIN...

- **RIGHT OUTER JOIN**
  - Returns all the rows from the right table in conjunction with the matching rows from the left table.
  - If there are no columns matching in the left table, it returns NULL values.



OUTER JOIN...

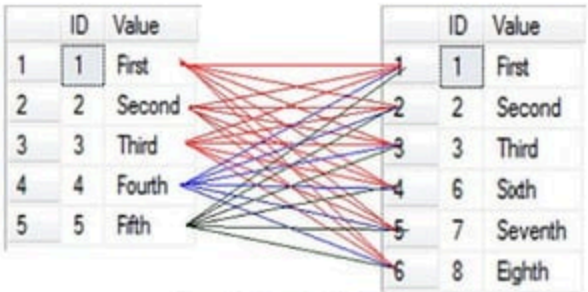
- **FULL OUTER JOIN**
  - Combines left outer join and right outer join.
  - It returns row from either table when the conditions are met and returns null value when there is no match




```
SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
    ON t1.Col1 = t2.Col1
```

CROSS JOIN

- This join is a Cartesian join that does not necessitate any condition to join.
- The result set contains records that are multiplication of record number from both the tables.

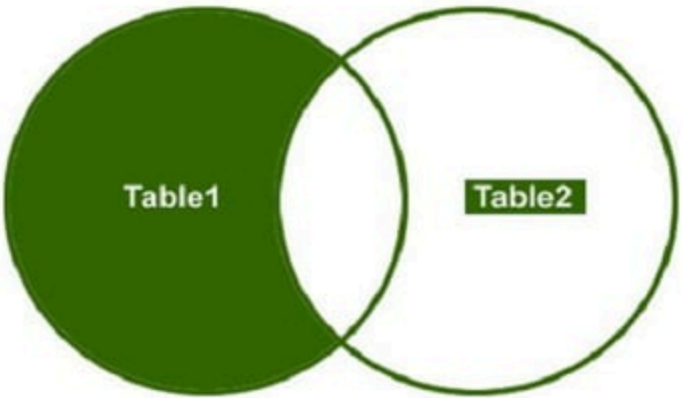


	ID	Value
1	1	First
2	2	Second
3	3	Third
4	4	Fourth
5	5	Fifth

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	6	Sixth
5	7	Seventh
6	8	Eighth

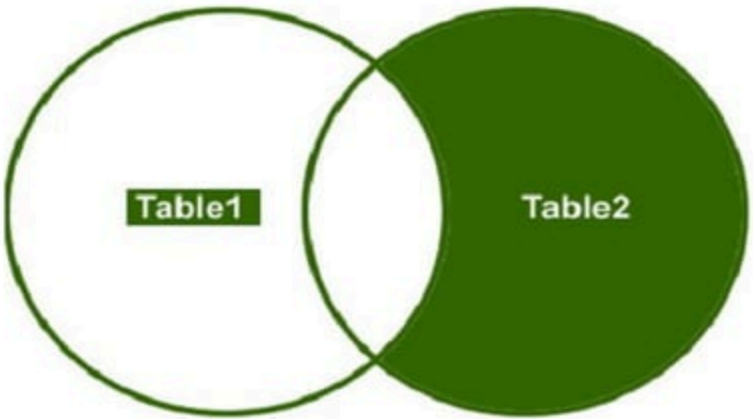
	ID	Value	ID	Value
1	1	First	1	First
2	1	First	2	Second
3	1	First	3	Third
4	1	First	6	Sixth
5	1	First	7	Seventh
6	1	First	8	Eighth
7	2	Second	1	First
8	2	Second	2	Second
9	2	Second	3	Third
10	2	Second	6	Sixth
11	2	Second	7	Seventh
12	2	Second	8	Eighth

LEFT OUTER JOIN WHERE NULL



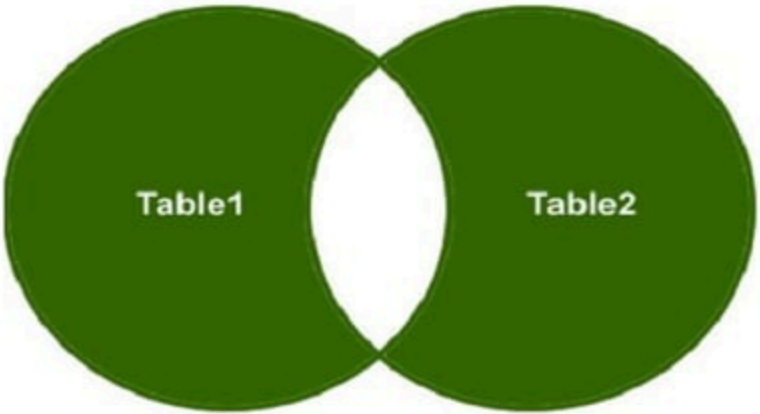

```
SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
        ON t1.Col1 = t2.Col1
WHERE t2.Col1 IS NULL
```

RIGHT OUTER JOIN WHERE NULL




```
SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
        ON t1.Col1 = t2.Col1
WHERE t1.Col1 IS NULL
```

OUTER JOIN WHERE NULL




```
SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
        ON t1.Col1 = t2.Col1
WHERE t1.Col1 IS NULL
        OR t2.Col1 IS NULL
```

UNION

- Used to select related information from two tables
- When using the UNION command all selected columns need to be of the same data type
- With UNION, only distinct values are selected.
- Example:

```
SELECT firstName, lastName, company FROM businessContacts
UNION ALL
SELECT firstName, lastName, NULL FROM nonBusinessContacts
```

UNION ALL

- Allows to join multiple datasets into one dataset When using the UNION command all selected columns need to be of the same data type
- Does not remove any duplicate rows
- Example:

```
SELECT firstName, lastName, company FROM businessContacts
UNION ALL
SELECT firstName, lastName, NULL FROM nonBusinessContacts
```

Subquery

- Subquery is a SELECT statement within another SQL statement
- Subqueries structure a complex query into isolated parts
- Subqueries allow you to use the results of another query in the outer query
- Example:

```
SELECT *
FROM HumanResources.Employee E
WHERE E.EmployeeID IN ( SELECT EA.EmployeeID
FROM HumanResources.EmployeeAddress EA
WHERE EA.EmployeeID = E.EmployeeID)
```

Aggregate Functions

- Aggregate functions that take a collection of values as input and returns a single value.
- Aggregate functions ignores NULL values except COUNT function
- HAVING clause is used, along with GROUP BY, for filtering query using aggregate values
- Some T-SQL Aggregate functions are: AVG, COUNT, MAX, MIN, SUM

Aggregate Functions...

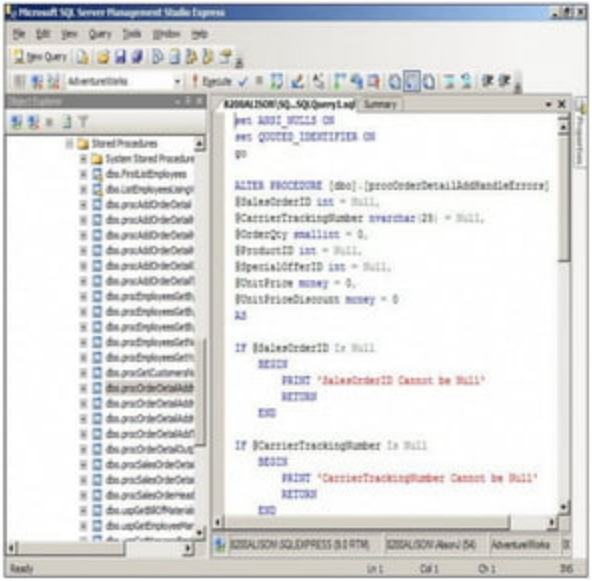
- AVG – To calculate average

```
Select branch_name, avg(balance) from account
Group by branch_name having avg(balance) > 1200
```

- COUNT - Returns the number of items in a group
  - SELECT COUNT(*) FROM HumanResources.Employee;
  - SELECT COUNT(DISTINCT Title) FROM HumanResources.Employee;

Stored Procedure

- Stored procedure is a piece of programming code
- Can accept input parameters and can return one or more output parameters
- Stored procedures generally perform operations on the database, including the process of calling



Anatomy of Stored Procedure

- A Stored procedure consists of
  - A ***header*** that defines the name of the stored procedure
  - Input and output parameters, and some miscellaneous processing options.
  - A ***body*** that contains one or more Transact-SQL statements to be executed at runtime.
  - Syntax:

CREATE PROC[EDURE] *procedure_name* [{*@parameter data_type*} [= default] [OUTPUT]] [,...*n*]

AS *sql_statement [...n]*

Anatomy of Stored Procedure…

- A sample stored procedure

```
ALTER PROCEDURE [dbo].[rsp_hrm_emp_bonus]
(
        @v_org_heading  VARCHAR(70),  --For Organization Name
    @v_org_address  VARCHAR(250), --For Organization Address
    @v_rpt_heading  VARCHAR(70),  --For Report Heading Name
        @year int, --For Year of Bonus
        @bonousType int  --For Bonus Type
)
AS
BEGIN
        select a.emp_id, a.emp_fullnm, dpt.look_nm as dept,
        des.look_nm as designation, a.basic,
        a.working_dt, a.bonous_amt
        from hrm_emp_bonous a, lookup des, lookup dpt
```

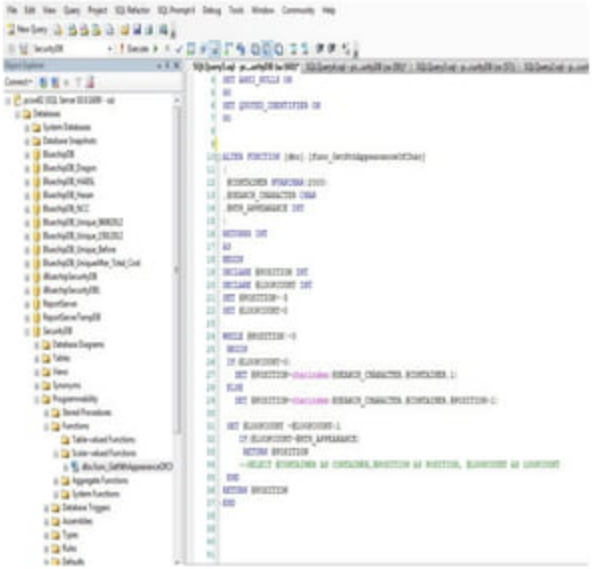
Store Procedure with output parameter

```
CREATE PROCEDURE getContactName
        @ContactID INT,
        @FirstName VARCHAR(50) OUTPUT,
        @LastName  VARCHAR(50) OUTPUT
AS
BEGIN
        SELECT @FirstName = FirstName, @LastName = LastName
        FROM Person.Contact
        WHERE ContactID = @ContactID
end
GO

DECLARE @CID INT, @FName VARCHAR(50), @LName VARCHAR(50)
SET @CID = 100
```

User defined function

- User-defined functions are procedures that accept parameters, perform some sort of action, and return the result of that action as a value
- The return value can be either a single scalar value or a result set



Scalar valued function

```
ALTER FUNCTION [dbo].[func_GetNthAppearanceOfChar]
(
 @CONTAINER NVARCHAR(2000)
,@SEARCH_CHARACTER CHAR
,@NTH_APPEARANCE INT
)
RETURNS INT
AS
BEGIN
DECLARE @POSITION INT
DECLARE @LOOPCOUNT INT
SET @POSITION=-5
SET @LOOPCOUNT=0

WHILE @POSITION!=0
 BEGIN
 IF(@LOOPCOUNT=0)
```

Table valued function

```
ALTER FUNCTION [dbo].[fn_hrm_total_leave_count]
(
	@emp_gid int,
	@frmdt smalldatetime,
	@todt smalldatetime
)
RETURNS @temp_Leave_count TABLE
(
	totalLeave int, totalLWP int, totalWeekEnd int,gov_holiday_ho int,other_holiday int,govt_holiday_mill int,govt_holiday_school int
)
AS
BEGIN
DECLARE
	@mindt int,
	@maxdt int,
	@betweendt int,
	@totalvdt int,
	@total_LWP int,
	@total_week_end int,
```

Trigger

- A trigger is a special type of stored procedure that executes when a language event executes.
- There are two kinds of triggers: DML triggers and DDL triggers
- DML triggers are invoked when a data manipulation language (DML) event takes place in the database
- DML Statement include INSERT, UPDATE, DELETE statement that occur when a data is modified in a table or view

DML Triggers...

- **After** Trigger
  - Syntax:
    - Create Trigger trigger_name On table {After {[Delete] [,] [Insert] [,] [Update]} As sql_statement [...n]
    - Example:

```
Create Trigger dbo.trMyEquipment_D
On dbo.MyEquipment After Delete -- For Delete
As
Print 'You have just deleted ' + Cast(@@rowcount as varchar) + ' record(s)!' Go
```

DML Triggers…

- Syntax:
  - Create Trigger trigger_name On table {After|For {[Delete] [,] [Insert] [,] [Update]} As sql_statement [...n]
  - Example:

```
Create Trigger dbo.trMyEquipment_D

On dbo.MyEquipment After Delete -- For Delete

As Print 'You have just deleted ' + Cast(@@rowcount as varchar) + ' record(s)!' Go
```

DML Triggers…

- Syntax:
  - Create Trigger trigger_name On table {After|For {[Delete] [,] [Insert] [,] [Update]} As sql_statement [...n]
  - Example:

```
Create Trigger dbo.trMyEquipment_D

On dbo.MyEquipment After Delete -- For Delete

As Print 'You have just deleted ' + Cast(@@rowcount as varchar) + ' record(s)!' Go
```

ACID

- **ACID** is a set of properties that guarantee that database transactions are processed reliably
- ACID
  - **Atomicity**
    - Requires that each transaction is "all or nothing".
    - If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged
  - **Consistency**
    - Ensures that any transaction will bring the database from one valid state to another.
    - Must be valid according to all defined rules, including but not limited to constraints, cascades, triggers
  - **Isolation**
    - Ensures that the concurrent execution of transactions results in a