

# Introduction to SQL Server (For Fresher)

As the title indicates this is only for fresher who has just started the carrier or who is waiting to open up the carrier in the application programming side. But that does not mean this article is a tutorial for a fresher who does not know anything about SQL. This article is meant for who already have a little knowledge in SQL and want to improve it.

The prerequisite for learning SQL is knowledge in Discrete Mathematics (Set Theory, Relations and Functions). Although it is not necessary to learn all the theorems and proof for the theorems in the Discrete Mathematics, you should have learned the basic concepts of the Sets, Relations and Functions. This will help you to learn SQL queries and fundamentals easily. If you want to explore a RDBMS more deeply you should learn Graph Theory too.

Although I tried to avoid SQL Server specific topics in this article, I am sure that some topics are pure to SQL server such as SQL Enterprise manager.

**By Samuel**

## Data to DBMS

Data is something that should be stored for future manipulations (In terms of Database). The system which provides such a facility is called Database Management System or DBMS.

The simplest form to store a data for latter retrieval is using a text file. For example you may want to store your friends name and phone numbers to use it latter. (In this case you may use notepad or word to do so.) This kind of storage is called flat file storage or unstructured storage. In this case the text editor uses the File and Directory services provided by the Operating System to accomplish the task of storing and retrieving data.

But these unstructured flat files are not suitable to large data such as storing stock details. Since the stock data is large in volume and added and updated frequently it is not scale up well if we use a simple flat file. To overcome this we need some system which should perform the storing, retrieving, manipulating and querying operations on the data and give output to us. This kind of system is called Database Management System. So a DBMS is a system which organizes and stores data in a structural way for fast retrieval. A DBMS uses one or more files to store the given data.

Since the DBMS is meant mainly for developers every DBMS has it is own language to write the commands. The languages are standardized under the name SQL (Structured Query Language). The following definition for SQL is from Books Online "A language used to insert, retrieve, modify, and delete data in a relational database. SQL also contains statements for defining and administering the objects in a database. SQL is the language supported by most relational databases, and is the subject of standards published by the International Standards Organization (ISO) and the American National Standards Institute (ANSI). SQL Server 2000 uses a version of the SQL language called Transact-SQL."

## Representation of Data by DBMS

DBMS should represent the data stored by them in some form and it is most common to represent them as Column, Row, Tables and Databases. As you know SQL you know what are columns, rows, tables and database. However I want to give one line explanation for those things here.

Column – A finite unit of data. It is represented by one of the data type.

Row – A collection of Columns.

Tables – Collection of rows.

Database – Collection of tables and other objects.

For example in case of Employee records – the name, date of join, salary etc will be called as Column or Field. The value for a particular employee for the above field is called as Row or Record.

## Primary Key

A column or set of columns used to uniquely identify the records in a table. Primary Keys don't allow NULL values. You can relate a table with other table if you have defined primary on it. (So defining UNIQUE and NOT NULL constraint is not equivalent to primary key). SQL server will create a Clustered index while you create a primary key.

Primary keys should be carefully defined since based on it other relations are defined. Poorly designed primary keys will affect insert, update and delete operations. Primary keys are different from which are used for paperwork. For example – you can have unique EmployeeNo(varchar) field. It may be of the format XXXX##XXX##. Although it is unique, making it primary key is not a good design because it is of character type and it may be clustered. Instead you should use it as alternate key and create primary key EmployeeId (int).

Since primary keys needs to be unique, SQL server provides two methods to generate unique keys – IDENTITY and UNIQUEIDENTIFIER.

## Server - Client

In Client Server technology, a server is a program which does all the jobs that is requested by the client and Client is a program which depends on the Server to accomplish its task. Client is mainly used to interact with Users and most Server software doesn't have direct interaction with Users.

A DBMS should have the following functionality – Interaction with users (developers), Managing Data. Since these two functionalities are entirely different modern DBMS is divided into two or more parts. Almost all current DBMS has the following module – Server and one or more clients. A server module is program which manages the data and handles requests from the client applications. It interprets the SQL queries and returns results back to the clients. A client program is a program which sends SQL commands to server and gets the result. Client program does not know the detailed underground structure of the server.

SQL Server 2000 runs as a service in NT based machines. You have to establish a connection to the server first if you want to do some SQL operation. SQL Server uses Local or Network IPC (Inter process communication) to communicate with the clients. See Books Online for more details.

The most basic misunderstanding of many junior developers is considering SQL Enterprise Manager as the SQL server. SQL Enterprise Manager is a tool used to access the SQL server. SQL Query analyzer is also similar tool used to access the SQL server. The difference between SQL query analyzer and Enterprise Manager is Query Analyzer is a light weight process and Enterprise manager is heavy weight processes due to its Graphical User Interface.

Enterprise Manager is a handy tool in the initial stage of a project. It will save lot more time while designing the database. You can easily create Tables, Triggers, Relationships, Constraints etc using Enterprise manger easily. I have seen many of developers don't have experience in using Enterprise Manager. Never become one of them, use Enterprise Manager whenever you start creating a database. Query Analyzer is a tool used to Create, Analyze, Modify and Delete T-SQL queries.

You can also access SQL server using osql.exe or isql.exe which is a very light weight command line utility. You can execute T-SQL commands in osql/isql. The difference between osql and isql is osql uses ODBC library whereas isql uses DB library to communicate with the SQL server. SQL Query analyzer is Windows version of isql.exe.

You can directly access SQL server by accessing the TCP Port/Named Pipes and issuing certain commands. However it is very tedious, so SQL Server has provided some library to access it easily. You can find the source code for these libraries in the installation directory of the SQL Server. For database communication standards SQL Server fully supports ODBC. Further it has own database driver for communication. SQL Server also offers SQL-DMO (SQL Distributed Management Objects) a COM component which offers easy programming interface. I have created one application called Whiz using the SQL-DMO. You can download source code for that application in my projects page.

Apart from these libraries many third party libraries are also available to access SQL Server. In .Net you can directly access SQL server using Data.SqlClient namespace.

Now you have learned how SQL Server and SQL Client tools are exists in a network. It is time to learn some Client-Server technology rules

- 1) There may be more than one connection to server at any time
- 2) Client should request only what it wants
- 3) Client requests should be short

The first point is related to Server so leave it. The second point says that the client should retrieve only the data that it needs. This is because transferring data from Server to Client utilizes server's resource and also increases network traffic.

Here is an example, you have an employee table in the server and you want to see Employee's name who has joined before '1-1-1990'. You can do it using the following three queries. The last query is best for Client-Server topology.

```
SELECT * FROM Employee
```

```
SELECT * FROM Employee WHERE DOJ < '1-1-1990'
```

```
SELECT EmpName FROM Employee WHERE DOJ < '1-1-1990'
```

# DDL (Data Definition Language)

## *Data Types*

**What is data type?** – Classification of data into similar groups. Names, Money, Date, Time, etc are examples for data type.

**What is the use of classification or data type?** – It increases the performance, reduces the space needed to store the data.

I can't explain how it increases the performance and how it reduces the space in a single paragraph. However I can direct you to find the solution – Try writing a C program which should store Employee Name, DateOfBirth, Salary, EmployeeID into file and provide option to search the records.

The following data types are available in SQL.

TINYINT, SMALLINT, INT, BIGINT  
DECIMAL/NUMERIC  
FLOAT, REAL

SMALLMONEY, MONEY

BIT

CHAR, NCHAR, VARCHAR, NVARCHAR, TEXT, NTEXT

BINARY, VARBINARY, IMAGE

CURSOR  
SQL\_VARIANT  
TABLE  
TIMESTAMP  
UNIQUEIDENTIFIER

The main difference between the data type is the capacity of the type. To see the capacity of the data type refer "Books Online".

CREATE TABLE, Constraints, Relationships,

All your definitions of the tables, views, stored procedures, functions are stored in system tables. Books online - The information used by Microsoft® SQL Server™ 2000 and its components is stored in special tables known as system tables.

You can query the system tables to obtain the inside structure of your schema. The following system tables are the some of the most used system tables – sysobjects, syscolumns, systypes, sysusers, sysindexes. Some of the tables are stored only in master database because they contain contents which are global across the databases – syslogins, syslockinfo, sysdatabases etc.

For example to get list of databases in the connected server, you can issue –

**SELECT \* FROM sysdatabases**

The following query will return all the table names in the connected database.

```
SELECT TableName = [Name] FROM sysobjects WHERE xtype='U'
```

The following query will return all the column details for a given Table.

**SELECT \***

**FROM sysobjects so(NOLOCK)**

**INNER JOIN syscolumns sc(NOLOCK)**

**ON so.[id]=sc.[id]**

**WHERE so.[Name]='TableName'**

You can query the system tables but you should not add, modify or delete anything directly. To add, delete and modify system table contents SQL Server provides some stored procedures.

**sp\_databases** – This stored procedure will return all the databases in the connected server.

**sp\_tables** – This will return all the tables in the database.

**sp\_columns** – This stored procedure will return the column details of a given Table.

## **DML (Data Manipulation Language)**

Data Manipulation Languages contains command used to query, change, and add data from/to the database. This includes the following commands - SELECT, INSERT, DELETE. All these commands are deals with set of data. So I want to refresh the Set theory concepts before going further.

### ***Set***

A set is a collection of objects considered as a whole.

### ***Subset***

If every member of the set A is also a member of the set B, then A is said to be a subset of B

### ***Union of two sets***

The union of A and B, denoted by  $A \cup B$ , is the set of all things which are members of either A or B.

### ***Intersection of two sets***

The intersection of A and B, denoted by  $A \cap B$ , is the set of all things which are members of both A and B. If  $A \cap B = \emptyset$ , then A and B are said to be disjoint.

To read more about Set theory visit <http://en.wikipedia.org/wiki/Set>

In SQL server, all data is handled as sets – rowset or recordset. Hereafter, the word rowset or recordset refers a collection of data.

## **SELECT**

This selects/extracts/returns only one rowset from the datasource. The datasource may be anything. In short, if you want to extract any data from SQL server you should use SELECT. Similarly, if you want to return any data to user or client application you should use SELECT. (Note - SELECT is also capable of inserting records into a table or XML file)

### **SELECT \* FROM Employee**

The above command instructs the SQL Server to extract all the columns and all the rows from the Employee table. But in practice you should not write a query like this – because it extracts all the columns and all the rows which may return GBs of data. Consider the employee table has the following fields Name, Address and Photo. Since the photo column is a binary image the column size may be more than 100KB. If there are one thousand employees and then the picture column alone will take  $1000 * 100KB = 100MB$  and the remaining name and address fields will take less than 1MB. If you want to see only Employee Name and address then the above query will return 101MB of data from which only 1MB is worth for you. So select only the fields which are required.

```
SELECT Name, Address FROM Employee
```

This command instructs the SQL Server to extract only name and address columns and all the rows from the Employee table. Again you should not use queries like this without row constraint. You can add row constraint to a query in two ways – TOP and WHERE clause.

### **TOP Clause**

This query will return the first seven Employees from the database. Although it is not very useful in any real time projects, it is very handy during debugging or data analysis.

```
SELECT TOP 7 Name, Address FROM Employee
```

The TOP clause is very useful when you combine it with ORDER BY clause. For example the following query will return the 7 senior most employees in the organization.

```
SELECT TOP 7 Name, Address, DateOfJoin, BasicPay FROM Employee ORDER BY  
DateOfJoin
```

As the query starts growing the complication starts because the clarity in reading the query is lost. To minimize this some kind of coding standard should be followed. I am following a standard which has the following rules

- 1) All SQL keywords should be in capital letter
- 2) Command should in the first line (with Top or any other clause)
- 3) Each column names should be in a separate line with one indent
- 4) The data source should be in a separate line

So the above query can be written as follows

```
SELECT TOP 7  
    Name,  
    Address,  
    DateOfJoin,  
    BasicPay  
FROM Employee  
ORDER BY DateOfJoin
```

SET ROWCOUNT option can also used to restrict the number of rows returned.

## **WHERE clause**

To add one and/or more condition to a query the WHERE clause is used.

**SELECT**

**Name,**  
**Address,**  
**DateOfJoin**

**FROM Employee**

**WHERE DateOfJoin > '1-Jan-2000'**

The above query will return who has joint after 1-Jan-2000. The following query will return who has joined after 1-Jan-2000 and before 12-Dec-2000.

**SELECT**

Name,  
Address,  
DateOfJoin,  
BasicPay

**FROM Employee**

**WHERE DateOfJoin > '1-Jan-2000' AND DateOfJoin < '12-Dec-2000'**

Or you can write it as

**SELECT**

Name,  
Address,  
DateOfJoin

**FROM Employee**

**WHERE DateOfJoin BETWEEN '1-Jan-2000' AND '12-Dec-2000'**

Now try to write a query that should return the last employee details who has joined before '1-Jan-2000'

**SELECT TOP 1**

Name,  
Address,  
DateOfJoin

**FROM Employee**

**WHERE DateOfJoin > '1-Jan-2000'**

**ORDER BY DateOfJoin DESC**

## GROUP BY

It is a clause which is used to summarize or group the data. For example, if you look into a cricket score board you may want to know the total runs instead of individual batsman scores.

```
SELECT
    Department,
    TotalPay = SUM(BasicPay)
```

FROM Employee

### **GROUP BY Department**

Here I have summarized the salary amount paid for each department.

```
SELECT
    Department,
    TotalEmployees = COUNT(*)
```

FROM Employee

### **GROUP BY Department**

The above query will return total employees in a department.

Now try to write a query that should give DepartmentName and average salary without using the **AVG()** function.

## INTO

The INTO keyword is used to create a table and insert the selected records into the newly created table.

```
SELECT
    Department,
    TotalEmployees = COUNT(*)
```

### **INTO #Result**

FROM Employee

GROUP BY Department

## **Multiple Rowsets**

Up to now I have shown you how to deal with a single result set. Now let try to add or subtract one or more rowsets to create new rowsets. For this assume you have 3 tables with employee information – Employee\_India, Employee\_USA, Employee\_China.

Now write query to select all the employee name from the 3 employee tables.

```
SELECT Name FROM Employee_India
UNION
SELECT Name FROM Employee_USA
UNION
SELECT Name FROM Employee_China
```

The above query will return all the name from the three employee tables. Now take only the names which starts with the letter 'A'.

You can do it in two ways adding WHERE clause to all the three tables or adding one WHERE clause to the final rowset

```
SELECT Name FROM Employee_India WHERE Name LIKE 'A%'
UNION
SELECT Name FROM Employee_USA WHERE Name LIKE 'A%'
UNION
SELECT Name FROM Employee_China WHERE Name LIKE 'A%'
```

Or

```
SELECT Name
FROM
    (
        SELECT Name FROM Employee_India
        UNION
        SELECT Name FROM Employee_USA
        UNION
        SELECT Name FROM Employee_China
    )
WHERE Name LIKE 'A%'
```

## JOINS

Relations between two row set is referred as JOIN in SQL. JOIN can be any one of the following types – one to one, one to many, many to many. Learn more about these types of relations in Discrete Mathematics. Similarly a function may be surjective, injective or bijective.

When I was start learning T-SQL I was ignoring the JOIN keyword syntax used by SQL Server, instead I was used to join tables using WHERE condition. I have seen many junior developers are doing this way. This is because the JOIN syntax is seem to be slightly confusing (at the starting stage) and it is only specific to MS-SQL Server. But once you understood the JOIN keyword syntax it is much very much useful to maintain the query.

To explain joins assume the following table structure.

Student	
Student_id	Int
Name	Varchar(20)

If you are not able to understand this table structure then you have learn lot of fundamentals before continuing.

Subject	
Subject_id	Int
Name	Varchar(20)

Mark	
Student_id	Int
Subject_id	Int
Mark	Tinyint

Here the relation between the Student and Mark table is Student\_id, similarly the relation between Subject and Mark table is Subject\_id.

You can extract marks obtained by student using the following query.

```
SELECT s.Name, m.Mark  
FROM Student s, Mark m  
WHERE s.Student_id = m.Student_id
```

The following query will return the same result set but using the JOIN syntax which is SQL-92 standard

```
SELECT s.Name, m.Mark  
FROM Student s INNER JOIN Mark m  
ON s.Student_id = m.Student_id
```

The difference between relationship and condition is very clear in JOIN syntax where it is not clear in the ordinary query. For example you have to retrieve Student Names who has scored more than 50 then you can write the following queries.

```
SELECT s.Name, m.Mark  
FROM Student s, Mark m  
WHERE  
    s.Student_id = m.Student_id  
    AND m.Mark > 50
```

In this query the relationship (s.Student\_id = m.Student\_id) and condition is (m.Mark > 50) is specified in WHERE condition which will make the query complicated. The following query is replacement of the above query.

```
SELECT s.Name, m.Mark  
FROM Student s INNER JOIN Mark m  
    ON s.Student_id = m.Student_id  
WHERE  
    m.Mark > 50
```

## OUTER JOIN

Some times we may found that the relation between two set is surjective. That is there may be elements in either or both side of the sets that is not linked with other set. For example consider the following example -

Emp_Id	Emp_Name	Dept_Id
1	A	1
2	B	1
3	C	2
4	D	NULL
5	E	1

Dept_Id	Dept_Name
1	X
2	Y
3	Z

In the employee table the Employee D (Emp Id - 4) does not related to the Department table. Similarly the department Z is not related the Employee table.

Now try to write to a query that should return the following fields - Employee Name, Department Name. The following query will miss out the Employee D.

```
SELECT
    EmployeeName,
    DeptName
FROM Employee E
INNER JOIN Department D
    ON E.Emp_id = D. Emp_id
```

To include the all the records in the left table you have to use LEFT OUTER JOIN, similarly you have to use RIGHT OUTER JOIN to include all the values in the right table.

```
SELECT
    EmployeeName,
    DeptName
FROM Employee E
LEFT OUTER JOIN Department D
    ON E.Emp_id = D. Emp_id
```

In this query instructs to include all the values from the Employee table and then link it to department table. LEFT OUTER JOIN works based on the position of the table name that is the table placed left to the OUTER JOIN key word will include in the whole. Sometimes when you join more than 2 tables you may need to use RIGHT OUTER JOIN because the table may come RIGHT to your query.

Now try to write a query which should include all employee records and department names even if they are not related.

The following query will display all the employee name and department name.

```
SELECT
    EmployeeName,
    DeptName
FROM Employee E
FULL OUTER JOIN Department D
    ON E.Emp_id = D. Emp_id
```

The result will be

EmployeeName	DeptName
A	X
B	X
C	Y
<b>D</b>	<b>NULL</b>
E	X
<b>NULL</b>	<b>Z</b>

## CROSS JOIN

Cross join will produce Cartesian product from two set. That is it will generate all the possible combination of relation between two set. The difference between CROSS JOIN and FULL OUTER JOIN is full outer join tries to link the elements and include the elements, but cross join links each element in one set with all other element in the other set. CROSS JOIN is equivalent to linking two tables without any join condition.

The following two queries will produce same result

```
SELECT EmpName, DeptName FROM Employee, Department
```

```
SELECT EmpName, DeptName FROM Employee CROSS JOIN Department
```

The result will be

EmployeeName	DeptName
A	X
A	Y
A	Z
B	X
B	Y
B	Z
C	X
C	Y
C	Z
D	X
D	Y
D	Z
E	X
E	Y
E	Z

## **UPDATE**

UPDATE statement is used to modify one set of records at a time. For example you can add title to the name of employees using the following query.

**UPDATE Employee SET EmpName = 'Mr.' + EmpName**

But it is not very useful until you add a condition. You can add a condition by using the WHERE condition.

**UPDATE Employee SET EmpName = 'Mr.' + EmpName WHERE Sex='M'**

**UPDATE Employee SET EmpName = 'Mrs.' + EmpName WHERE Sex='F'**

The above query will update the Employee name with the prefixed title. The title will be Mr. for all male employees and Mrs. for all female employees. (I assumed the employer has a policy of recruiting only married persons). You can write the above two queries in a single statement using CASE keyword.

**UPDATE Employee**

```
    SET EmpName =  
        CASE Sex  
            WHEN 'M' THEN 'Mr.'  
            WHEN 'F' THEN 'Mrs.'  
        END  
    + EmpName
```

Note you can CASE statement in any SQL statement not only in UPDATE statements.

Another important feature of UPDATE statement is you need not use any temporary variable to swap value between two fields. For example, assume in your employee table the DateOfJoin field contains DateOfBirth data and DateOfBirth field contains DateOfJoin field data. Now you have to swap the data, the following query will do that -

**UPDATE Employee**

```
    SET  
        DateOfJoin = DateOfBirth,  
        DateOfBirth = DateOfJoin
```

In very rare occasion you will need to write simple update queries like this, the more frequent occasion will be combining two or more tables and updating one table. Remember you can't update more than one table at a time. SQL server simply doesn't support. You can update only one table at a time. Remember the Student Table

structure that I have mentioned, now add another field called "Percentage (float)" to the Student table. The following query will update it.

**UPDATE s**

**SET Percentage = SUM(m.Marks) / COUNT(m.Marks)**

**FROM Student s**

**INNER JOIN Mark m**

**ON s.Student\_id = m.Student\_id**

**GROUP BY s.Student\_id**

In the above query you are updating the student table since you have mentioned the alias name.

## ***DELETE***

Delete is easiest statement in the SQL, I think. To delete all records from a table you can issue **DELETE FROM TableName**. Here we have not used **DELETE \* FROM TableName** because the \* is used to specify all columns, since we can delete only rows and not columns. The DELETE operation creates logs and can be undoable. Since it performs under transaction it is safer but slower. The alternative command to delete records from a table is **TRUNCATE TABLE TableName**. The TRUNCATE command deletes all records from the table you cannot control it. In DELETE command you control the delete operation by specifying the WHERE condition.

## ***INSERT***

Insert command is used to insert data into table.

The following commands will do the same thing. It will insert a new student name into the table with student id 10001.

```
INSERT INTO Student (Student_id, StudentName)  
VALUES (10001, 'Vimal')
```

If you are going to insert all values you can omit the field names.

```
INSERT INTO Student  
VALUES (10001, 'Vimal')
```

(If you try to execute the same query second time you should get Primary Key violation, if you have set primary key on the student\_id field)

Assume you have table called OldStudent which has old student names and you want to transfer it to the student table, the following query will do it.

```
INSERT INTO Student (Student_id, StudentName)  
SELECT Student_id, StudentName FROM OldStudent
```

Similarly assume you have student information in the following tables – ClassA, ClassB, ClassC and you want to import into a Student Table you can use the following query. (Add a field Class (Varchar) to student table, if you want to store Class name also.)

```
INSERT INTO Student (Student_id, StudentName, Class)  
(  
SELECT Student_id, StudentName, 'A' FROM ClassA  
UNION  
SELECT Student_id, StudentName, 'B' FROM ClassB  
UNION  
SELECT Student_id, StudentName, 'C' FROM ClassC  
)
```

The above queries will work fine only if the student\_ids are unique across the table. If the student\_ids are auto-generated you can't use them to insert into the Student table because they will generate duplicate values and the query will fail. To avoid that you have to generate Student\_ids automatically, the following query will do that.

```
SELECT
    IDENTITY(int, 10000) AS Student_id,
    StudentName
INTO #tmp
FROM OldStudent
```

```
INSERT INTO Student (Student_id, StudentName)
SELECT Student_id, StudentName FROM #tmp
```

The identity function inserts serial numbers starting from 10000 into the Student Table.

If you have inserted a single row into a table, then you may want to know the last inserted identity value. For that SQL server provides the following one function and one global variable - `IDENT_CURRENT` and `@@IDENTITY`

**IDENT\_CURRENT** (TableName) returns the last identity value generated for a specific table in any session and any scope.

**@@IDENTITY** returns the last identity value generated for any table in the current session, across all scopes.

# Transactions

A transaction is a sequence of operations performed as a single logical unit of work. A logical unit of work must exhibit four properties, called the **ACID** (Atomicity, Consistency, Isolation, and Durability) properties, to qualify as a transaction.

## *Atomicity and Consistency*

SQL Server provides Data Control Language for controlling transactions. **BEGIN TRANSACTION** is used to start a transaction. You can use **ROLLBACK TRANSACTION** to rollback your transaction. ROLLBACK TRANSACTION erases all data modifications made since the start of the transaction or to a savepoint. It also frees resources held by the transaction. **COMMIT TRANSACTION** can be used to commit your changes to the database. SQL server supports nested transactions.

### **BEGIN TRANSACTION**

```
UPDATE Student SET StudentName = 'Test' WHERE StudentID = 10000
SELECT StudentName FROM Student WHERE StudentID = 10000
```

### **ROLLBACK TRANSACTION**

```
SELECT StudentName FROM Student WHERE StudentID = 10000
```

The above query will return Test and OldStudentName, since the transaction is cancelled.

### **BEGIN TRANSACTION**

```
UPDATE Student SET StudentName = 'Test' WHERE StudentID = 10000
SELECT StudentName FROM Student WHERE StudentID = 10000
```

### **COMMIT TRANSACTION**

```
SELECT StudentName FROM Student WHERE StudentID = 10000
```

The above query will return Test and Test since the transaction is updated.

## ***Isolation***

SQL Server provides locking hints to isolate a unit of work. A range of table-level locking hints can be specified using the SELECT, INSERT, UPDATE, and DELETE statements to direct Microsoft® SQL Server™ 2000 to the type of locks to be used. NOLOCK, HOLDLOCK, ROWLOCK, TABLOCKX and UPDLOCK are the some of the examples for lock hints.

1)

```
BEGIN TRANSCATION
    UPDATE Student WITH (HOLDLOCK)
        SET StudentName='Test'
        WHERE Student_id= 10000
    SELECT StudentName FROM Student
        WHERE Student_id = 10000
COMMIT TRANSACTION
```

2)

```
SELECT StudentName FROM Student WITH (NOLOCK)
    WHERE Student_id = 10000
```

To test the isolation, run the 1<sup>st</sup> query (comment out the COMMIT statement) in a query analyzer, open another window of query analyzer and try to run the following command – SELECT StudentName FROM Student WHERE Student\_id = 10000. You will be blocked until the transaction running in 1<sup>st</sup> window is committed or roll backed. If you want retrieve records without being blocked use the 2<sup>nd</sup> query WITH NOLOCK.

## ***Durability***

It is all about internal database logs. So it may not useful for a fresher.

## Views

Abstraction and Security is the main purpose for which I have used Views.

**Abstraction** – You can hide the internal table structure from the client applications and give a better interface through views. For example you can combine Student and Marks table to get results and define it as a view as follows.

```
CREATE VIEW Result AS
SELECT
    StudentName = s.StudentName,
    TotalMark = SUM(m.Mark),
    Grade = CASE
        WHEN SUM(m.Mark)/COUNT(m.Mark) > '80' THEN 'First Class'
        WHEN SUM(m.Mark)/COUNT(m.Mark) > '60' THEN 'Second
    Class'
        WHEN SUM(m.Mark)/COUNT(m.Mark) > '50' THEN 'Third Class'
        ELSE 'FAIL'
    END
FROM Student s
INNER JOIN Mark m
    ON s.Student_id = m.Student_id
```

Here the Relationship between tables and the calculations are abstracted.

Views are constructed as virtual table or logical table. This means that they does not exists in the physical data page, they exists only in the code form. But when you run a query on a view it is constructed as a virtual table in the TempDB.

You can create Clustered indexes on the Views.

## Stored Procedures

A stored procedure is a group of Transact-SQL statements compiled into a single execution plan. It is like a function or procedure in ordinary programming language. You can return one more values from a stored procedure.

```
CREATE PROCEDURE sp_Test  
AS  
    SELECT 'Test'  
    SELECT StudentName, Student_id FROM Student  
GO
```

The above stored procedure will return the following values – one 'Test' and all the StudentName, Student\_id from the student table.

```
CREATE PROCEDURE sp_StudentInsert  
@StudentName AS VARCHAR(50), @StudentID AS INT OUTPUT  
AS  
    IF LTRIM(@StudentName) = ''  
    BEGIN  
        RAISERROR('Student Name should be atleast one character')  
        RETURN  
    END  
    IF EXISTS(SELECT 1 FROM Student WHERE StudentName = @StudentName)  
    BEGIN  
        RAISERROR('Student Name already exists')  
        RETURN  
    END  
    INSERT INTO Student(StudentName) VALUES(@StudentName)  
    IF @@Error = 0  
    BEGIN  
        SET @StudentID = @@Identity  
        RETURN  
    END  
    RAISERROR('Insert operation failed. Error Code - %d', @@Error )  
GO
```

The above stored procedure will insert a new student name into the student table and return the studentid as the output parameter.

## ***GLOBAL Variables***

I found the following variables as most useful.

@@Error - Returns the error number for the last Transact-SQL statement executed.

@@FETCH\_STATUS - Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

@@IDENTITY - Returns the last-inserted identity value.

@@ROWCOUNT - Returns the number of rows affected by the last statement.

## Indexes

Indexes are created for the fast execution of Query. Unnecessary creation of indexes will slow down your application, so don't create indexes until you finished creating the queries. INSERT statement will work faster if there is no index at all. To learn you have to read about SCAN and SEEK operations first. SCAN – It is an operation which will move sequentially from one record to another record to compare a value. SEEK – This operation will move to the record position based on the group of record. SEEK operations are faster for query execution.

Only one index per query can be used by SQL server. So be careful about creating index for a query.

### ***Execution Plan***

You can view execution plan using SQL query analyzer. Using the execution plan you can redesign the table or create, modify drop indexes.

### ***Table Hints***

Well you learned to create indexes and created an index that may made your query execution faster because SQL server will pick up indexes automatically for query execution. But in few cases, if there are more than one index on table you have specify the index you want to use in the query, otherwise SQL server may pickup some other index which may not give the full performance. (It don't mean SQL server will randomly pickup the indexes but meant SQL server may miss to pickup the index which you want).

The following query will use the index IX\_EmployeeEmpName defined on the Employee table extract the employee names.

```
SELECT EmpName  
FROM Employee WITH (INDEX (IX_EmployeeEmpName))  
WHERE EmpName LIKE 'Test%'
```

The following query will use the index IX\_EmployeeEmpId to extract employee name.

```
SELECT EmpName  
FROM Employee WITH (NOLOCK, INDEX (IX_EmployeeEmpId))  
WHERE EmpId = 10000
```

## DTS (Data Transformation Services)

Data Transformation Services are used to transfer and transform data from one datasource to another datasource in the same server or another server.

Some applications of DTS

- 1) Copying **data** from one database to another database
- 2) Copying **data structure** from one database to another database
- 3) Migrating data from other datasources such as Flat File, Excel, Access, Oracle to SQL Server.
- 4) Migrating data from SQL server to other datasources.

Since the DTS topic is huge you cannot get to know what it is until you do actual task that is related to DTS. So try the following things and you will get to know something about DTS. Create an Excel file with the following columns EmployeeName, Address, DateOfJoin, PhoneNumber, MaritalStatus, and Department. Fill this excel sheet with some meaningful information and then try to transfer the contents from Excel to SQL using DTS Import/Export Wizard.

You can also create DTS packages which can be scheduled to run at future. DTS package programming allows mapping source fields to different destination fields and also provides error control.

## **SQL Profiler**

SQL Profiler is a handy tool used to analyze what is happening inside and outside of a SQL Server. Simply it is a tool which extracts the log information from SQL server. These logs will help you debug applications, optimize queries, redesign database structure.

SQL Profiler is a powerful which has no alternate tool to extract trace information from the SQL Server. You can change the trace settings of SQL Server by using SQL Profiler or by using system stored procedures such as `sp_trace_create`, `sp_trace_setfilter`, `sp_trace_generateevent`, `sp_trace_setstatus` and `sp_trace_setevent`. SQL server writes one record in the `tempdb.TraceQueue` table for each trace it makes.

Running SQL Profiler is very easy, Click SQL profiler from the SQL Server group menu. Then select New Trace from the file menu. It will open the connection dialog box. Type your server name, user name and password. Now you have to set options for your profile. The options include what **events** you want to trace, what **columns** you want in the output, filters if any.

For example you may want to see all **applications** that access a **table**. Then you have add the **Object:Opened** event and also add **ApplicationName** in the Data Columns. This trace will track all the events raised against all objects, but we want only a particular table in a specific database, to do that you have to add a filter to this trace. So add two filters **ObjectName Like** and **DataBaseName Like**. Then click Run. It will start tracing after that.

## **DBCC**

Database Console Commands are referred as DBCC. DBCC contains some special commands through which you can accomplish certain DB operations which is not possible through normal SQL commands.

For example you can check the allocation and structural integrity of all the objects in the specified database using the DBCC CHECKDB command.

I have used the following commands and found they are very useful.

DBCC INDEXDEFRAG - Defragments clustered and secondary indexes of the specified table or view.

DBCC SHRINKDATABASE - Shrinks the size of the data files in the specified database. This command is very useful to reduce the size of the data files and log files.

DBCC INPUTBUFFER - Displays the last statement sent from a client. You have to pass the spid of the client to this command. To get the client id use the following query – SELECT \* FROM sysprocesses

DBCC CHECKIDENT - Checks the current identity value for the specified table and, if needed, corrects the identity value.

**The End**

## ***About Me***

I am not a DBA. But I have some working knowledge in SQL. Although my interest is in system side, I always want to make some documentations/articles. And I hope I will move into system side programming soon so I make this article as my contribution to application side programming. I like to give my thanks to Mathiazhagan and Chandra Babu for giving early feedback on this article.

You can get the latest version of this document from the following URL:

<http://www.samueldotj.com/Articles.asp>