use employee

select 10/2

select 10/0

begin try

select 10/0

end try

begin catch

print 'its error'

end catch

begin try

select 10/2

print 'Line'

end try

begin catch

print 'its error'

end catch

select \* from emp

begin try

update emp set salary ='aaa'

end try

begin catch

print 'Cant'

end catch

begin try

update emp set salary ='aaa'

end try

begin catch

select error\_state() As "Error State",

error\_number() as [Error NUmber],

ERROR\_LINE() as [Error Line],

ERROR\_MESSAGE() as [Message],

ERROR\_PROCEDURE() as [SP Name]

print 'Cant'

end catch

alter proc A1

As

begin

begin try

declare @n1 int

declare @n2 int

Set @n1 = 10

set @n2 = 0

if (@n2 = 0)

RAISERROR('No can not be 0',1,1)

end try

begin catch

select ERROR\_MESSAGE() As "Error Message",

ERROR\_LINE() [Line Number],

ERROR\_SEVERITY() [Error Sev],

ERROR\_PROCEDURE() [Procedure Name]

end catch

end

Transactions

Transactions in SQL Server

For any business, transactions that may be comprised of many individual operations and even other transactions, play a key role.

Transactions are essential for maintaining data integrity, both for multiple related operations and when multiple users that update the database concurrently.

## What Is a Transaction?

A transaction is a set of operations performed so all operations are guaranteed to succeed or fail as one unit.

**Transaction is all or none**

A common example of a transaction is the process of transferring money from a checking account to a savings account.

This involves two operations:

1. Deducting money from the checking account and  
     
   **Note**: in the USA a checking account is like a current account in India
2. Adding it to the savings account.

Both must succeed together and the changes must be committed to the accounts, or both must fail together and rolled back so that the accounts are maintained in a consistent state. Under no circumstances should money be deducted from the checking account but not added to the savings account (or vice versa), you would at least not want this to happen with the transactions occurring with your bank accounts.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | item | qty |  |  |  |  | C1 |  | update customer |
|  |  |  | i1 | 90 |  |  |  | i2 | 40 |  |  |
|  |  |  | i2 | 40 |  |  |  |  |  |  |  |
|  |  |  | i3 | 12 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | in customet t able : insert item |  |  |  |  |  |  |  |
|  |  |  |  | in stcok table update |  |  |  |  |  |  |  |
|  |  |  |  | pay amount : amount is dedeucted from customer |  |  |  |  |  |  |  |
|  |  |  |  | amount I added yo item |  |  |  |  |  |  |  |

By using a transaction concept, both the operations, namely debit and credit, can be guaranteed to succeed or fail together. So both accounts remain in a consistent state all the time.

## When to Use Transactions

You should use transactions when several operations must succeed or fail as a unit. The following are some frequent scenarios where use of transactions is recommended:

* In batch processing, where multiple rows must be inserted, updated, or deleted as a single unit
* Whenever a change to one table requires that other tables be kept consistent
* When modifying data in two or more databases concurrently
* In distributed transactions, where data is manipulated in databases on various servers

When you use transactions, you put locks on data that is pending for permanent change to the database. No other operations can take place on locked data until the acquired lock is released. You could lock anything from a single row up to the entire database. This is called concurrency, which means how the database handles multiple updates at one time.

In the bank example above, locks will ensure that two separate transactions don't access the same accounts at the same time. If they do then either deposits or withdrawals could be lost.

A transaction has only two results: **success** or **failure**.

Transaction follows ACID Rule

A database transaction must be atomic, consistent, isolated and durable. Bellow we have discussed these four points.

**Atomic :** A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed.

**Consistent :** At the end of the transaction, all data must be left in a consistent state.

**Isolated :** Modifications of data performed by a transaction must be independent of another transaction. Unless this happens, the outcome of a transaction may be erroneous.

**Durable :** When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

Often these four properties of a transaction are acronymed as **ACID**.

Commands used in Transactions are known as TCL

Commit

Rollback

Savepoint

update data set name ='Deepak' where id=2

We want to undo this, how do we so this

begin transaction

update data set name ='Deepak' where id=2

commit trans

rollback trans

Open other Window

--set transaction isolation level read uncommitted

select \* from data

Trasactions with try-catch

Begin try

Begin transcation

commit transaction

End try

drop table data

create table data (id int primary key, name varchar(20), salary int check(salary between 10000 and 25000))

Begin try

Begin transaction

insert into data(id,name,salary) values(1,'Ajay',21000)

insert into data(id,name,salary) values(2,'Ajay',21000)

insert into data(id,name,salary) values(3,'Ajay',21000)

commit transaction

print 'NO Error'

End try

begin catch

rollback transaction

print 'ERROR'

Select ERROR\_MESSAGE() as [ERROR MSG]

end catch

select \* from data

Triggers

--set transaction isolation level read uncommitted

select \* from data

truncate table data

insert into data values(1,'Ajay',12000)

insert into data values(2,'Dilip',21000)

insert into data values(3,'Dev',24000)

insert into data values(4,'Lalit',12000)

insert into data values(5,'Ravi',21000)

insert into data values(6,'Sagar',24000)

insert into data values(7,'Manoj',12000)

insert into data values(8,'Girish',21000)

insert into data values(9,'Lalita',22000)

insert into data values(10,'Pawan',18000)

create trigger T1 on data

after insert

AS

Begin

print 'Record has been inserted'

select \* from inserted

end

create table data\_audit(audit\_id int identity, audit\_info varchar(max))

create trigger data\_insert\_audit

on data

after insert

As

begin

declare @id int

select @id = id from inserted

insert into data\_audit values('Record with id ' + Convert(varchar(20), @id) + ' is inserted')

-- insert into data\_audit values('Record with id ' + Cast(@id as varchar(20)) + ' --is inserted')

-- insert into data\_audit values('Record with id ' + Cast(@id as varchar(20)) + ' --is inserted' + ' at ' + cast(GetDate() as varchar(50)))

end

select \* from data\_audit

creater trigger data\_delete\_audit

on data

after delete

As

begin

declare @id int

select @id = id from deleted

insert into data\_audit values('Existimg Record with Id '

+ Cast(@id as varchar(20)) + ' is ideleted' + ' at ' + cast(GetDate() as varchar(50)))

end

create trigger T2 on data

instead of insert

AS

Begin

print 'Record cannot be inserted'

end

**We can have more than one after trigger on a table for one operation but we can have only one instead of trigger on a table for one operation**

**DDL Triggers**

create trigger T100

on database

for create\_table

As

begin

print 'new table created'

end

create table tt(id int)

create trigger T101

on database

for create\_procedure

As

Begin

rollback

print ‘ You can not create procedure’

end

create trigger T101

on database

with encryption

for create\_procedure

As

begin

rollback

print 'new proc created'

end

**disable trigger triggername on database**

**enable trigger triggername on database**

create trigger T102

on database

for rename

As

begin

print ‘you just renamed a coulm or a table'

end

Syntax of Sub Query is

Main Query / Outer Query OPERATOR (Inner Query / Subquery / Nested Query)

Operators : > < == <> >= <= IN ANY ALL

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

## Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

**Give all employees who gets salary more than average salary**

select \* from employee where salary > (Select avg(salary) from employee)

What is subquery in SQL?

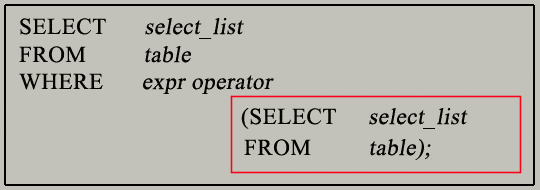
A subquery is a SQL query nested inside a larger query.

* A subquery may occur in :
  + - A SELECT clause
  + - A FROM clause
  + - A WHERE clause
* The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
* A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
* You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
* A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
* The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

* Compare an expression to the result of the query.
* Determine if an expression is included in the results of the query.
* Check whether the query selects any rows.

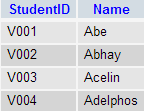
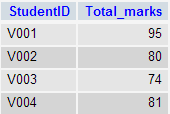
**Syntax :**



* The subquery (inner query) executes once before the main query (outer query) executes.
* The main query (outer query) use the subquery result.
* set transaction isolation level read uncommitted
* select \* from employee
* -- Give me names of employees who gets salry more than Ajay
* Select \* from employee where salary > (Select salary from
* employee where name = 'Deepak')
* -- Give me names of employees who gets salary what Ajay and Deepak are getting
* Select \* from employee where salary IN (Select salary from
* employee where name IN ('Deepak','Ajay'))
* -- Give me names of employees who gets salary more than Ajay and Deepak both
* Select \* from employee where salary > ALL (Select salary from
* employee where name IN ('Deepak','Ajay'))
* -- Give me names of employees who gets salary more than the avaerage salary
* Select \* from employee where salary > (Select avg(salary) from employee)

SQL Subqueries Example :

In this section, you will learn the requirements of using subqueries. We have the following two tables 'student' and 'marks' with common field 'StudentID'.

  
           
            student                                        marks

create table student (rn int primary key , name varchar(20))

create table marks (rn int references student(rn), marks int)

insert into student values(1,'Ajay'),(2,'Deepak'),(3,'Sagar')

insert into marks values(1,90), (2,23), (3,80)

Who has got marks more than rn 1

select \* from student , marks where student.rn = marks.rn and

marks.marks > (Select marks from marks where rn=1)

Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.  
- To solve the problem, we require two queries. One query returns the marks (stored in Total\_marks field) of 'V002' and a second query identifies the students who get better marks than the result of the first query.

**First query:**

SELECT \*

FROM `marks`

WHERE studentid = 'V002';

Copy

**Query result:**

student query

The result of the query is 80.  
- Using the result of this query, here we have written another query to identify the students who get better marks than 80. Here is the query :

**Second query:**

SELECT a.studentid, a.name, b.total\_marks

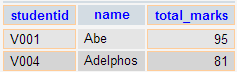
FROM student a, marks b

WHERE a.studentid = b.studentid

AND b.total\_marks >80;

Copy

**Query result:**



Above two queries identified students who get the better number than the student who's StudentID is 'V002' (Abhay).

You can combine the above two queries by placing one query inside the other. The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

**SQL Code:**

SELECT a.studentid, a.name, b.total\_marks

FROM student a, marks b

WHERE a.studentid = b.studentid AND b.total\_marks >

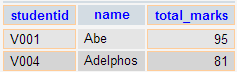
(SELECT total\_marks

FROM marks

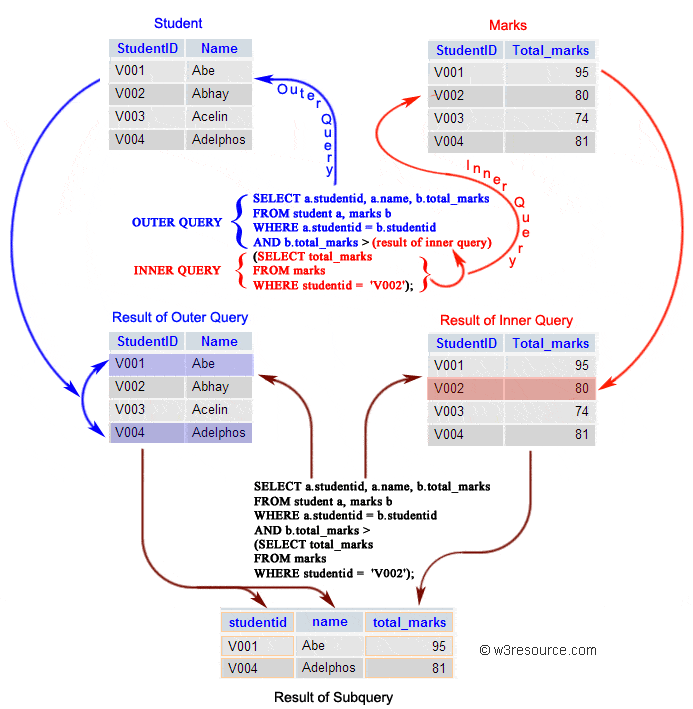
WHERE studentid = 'V002');

Copy

**Query result:**



**Pictorial Presentation of SQL Subquery:**

[](https://www.w3resource.com/sql/subqueries/sql-subqueries.gif)

Subqueries: General Rules

A subquery SELECT statement is almost similar to the SELECT statement and it is used to begin a regular or outer query. Here is the syntax of a subquery:

**Syntax:**

(SELECT [DISTINCT] subquery\_select\_argument

FROM {table\_name | view\_name}

{table\_name | view\_name} ...

[WHERE search\_conditions]

[GROUP BY aggregate\_expression [, aggregate\_expression] ...]

[HAVING search\_conditions])

Subqueries: Guidelines

There are some guidelines to consider when using subqueries :

* A subquery must be enclosed in parentheses.
* A subquery must be placed on the right side of the comparison operator.
* Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
* Use single-row operators with single-row subqueries.
* If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Type of Subqueries

* Single row subquery : Returns zero or one row.
* Multiple row subquery : Returns one or more rows.
* Multiple column subqueries : Returns one or more columns.
* Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
* Nested subqueries : Subqueries are placed within another subquery.

In the next session, we have thoroughly discussed the above topics. Apart from the above type of subqueries, you can use a subquery inside INSERT, UPDATE and DELETE statement. Here is a brief discussion :

Subqueries with INSERT statement

INSERT statement can be used with subqueries. Here are the syntax and an example of subqueries using INSERT statement.

**Syntax:**

INSERT INTO table\_name [ (column1 [, column2 ]) ]

SELECT [ \*|column1 [, column2 ]

FROM table1 [, table2 ]

[ WHERE VALUE OPERATOR ];

create table student\_temp (rn int primary key , name varchar(20))

Insert into student\_temp

select \* from student

where rn in (2,3)

If we want to insert those orders from 'orders' table which have the advance\_amount 2000 or 5000 into 'neworder' table the following SQL can be used:

Sample table: orders

**SQL Code:**

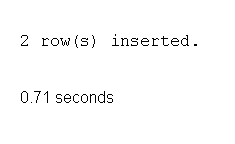
INSERT INTO neworder

SELECT \* FROM orders

WHERE advance\_amount in(2000,5000);

Copy

Output:



To see more details of subqueries using INSERT statement [click here](https://www.w3resource.com/sql/insert-statement/insert-using-subqueries.php).

Subqueries with UPDATE statement

In a UPDATE statement, you can set new column value equal to the result returned by a single row subquery. Here are the syntax and an example of subqueries using UPDATE statement.

**Syntax:**

UPDATE table SET column\_name = new\_value

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

If we want to update that ord\_date in 'neworder' table with '15-JAN-10' which have the difference of ord\_amount and advance\_amount is less than the minimum ord\_amount of 'orders' table the following SQL can be used:

Sample table: neworder

**SQL Code:**

UPDATE neworder

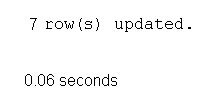
SET ord\_date='15-JAN-10'

WHERE ord\_amount-advance\_amount<

(SELECT MIN(ord\_amount) FROM orders);

Copy

Output:



update student\_temp

set name ='Deepak'

where rn = (select rn from student where rn =1)

Subqueries with DELETE statement

DELETE statement can be used with subqueries. Here are the syntax and an example of subqueries using DELETE statement.

**Syntax:**

DELETE FROM TABLE\_NAME

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

If we want to delete those orders from 'neworder' table which advance\_amount are less than the maximum advance\_amount of 'orders' table, the following SQL can be used:

Sample table: neworder

**SQL Code:**

DELETE FROM neworder

WHERE advance\_amount<

(SELECT MAX(advance\_amount) FROM orders);

delete from student\_temp

where rn > (Select rn from student where name='Ajay')

Multi Value Subqueries

select \* from student where rn in (select rn from marks)

insert into student values(4,'Ajay'),(5,'Deepak'),(6,'Sagar')

select \* from student where rn > ANY (select rn from marks)

select \* from student where rn = ANY (select rn from marks)

select \* from student where rn > ALL (select rn from marks)

**Inner query is executed once, what it returns is used by the outer query**

**A correlated subquery is a subquery that uses values from the outer query**

**This means that the subquery is executed repeatedly, once for each row that might be selected by the outer query**

create table empl (id int primary key identity, name varchar(20), deptno int , salary int)

insert into empl values('Karan',30,32000),

('Ravi',20,13000),('Ajay',10,27000),('Deepak',20,23000),('Vijay',10,19000),('Sagar',20,20000),('Ajay',30,19000)

select \* from empl

**Display the names of those employees whose salary is greater than average salary of all their own department**

**1 Karan 30 32000**

**2 Ravi 20 13000**

**3 Ajay 10 27000**

**4 Deepak 20 23000**

**5 Vijay 10 19000**

**6 Sagar 20 20000**

**7 Ajay 30 19000**

Select name, salary, deptno from empl e

where salary > (select avg(salary) from empl

where deptno = e.deptno)

**Here inner query executes multiple times , outer query will keep same**

**Reference is taken from outer table**

select avg(salary) from empl

select avg(salary) from empl where deptno=10 -- 23000

select avg(salary) from empl where deptno=20 -- 18666

select avg(salary) from empl where deptno=30 -- 25500

**Give employee whose salary is more than salary of Sagar**

This is inner query

select \*from empl where salary > (Select salary from empl where name ='Sagar')