# Pabna University of Science and Technology


B.Sc.(Engineering) 3rd Year 1st Semester
Session: 2017-2018

**Department of**
**Information and Communication Engineering**

**Faculty of Engineering and Technology**

Course Title: Database Management Systems Sessional

Course Code: ICE-3106

## Lab report

**Submitted to:**

Md. Tofail Ahmed
Lecturer
Department of
Information and Communication Engineering
Pabna University of Science and Technology
Date: 19-02-2022

**Submitted by:**

Laila Binta Mustafiz
Roll: 180610
Registration no: 1065293
Department of
Information and Communication Engineering
Pabna University of Science and Technology

19.2.22

# INDEX

Experiment no. : 01

Name of the Experiment: Study and Implementation of DML commands of SQL with Suitable example.

- Insert
- Delete
- Update

~~Theory~~:

Objectives:

1. To understand and use of Data manipulation Language to write query for database.
2. To study how to insert, delete and update data in the database.

Theory:

Data Manipulation Language (DML): A data-manipulation language is a language that enables users to access or manipulates data as organized by the appropriate data model.

The types of access are:

i) Retrieval of information stored in the database.
ii) Insertion of new information into the database.
iii) Deletion of information from the database.
iv) Modification of information stored in the database.

a) **Insertion:** To insert data into a relation we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

The simplest "insert" statement is a request to insert one tuple. Suppose we want to insert the fact that there is a course CS-437 in the Computer Science department with the title "Database systems" and four credit hours in a 'course' table. The statement will be:

```
insert into course
        values ('CS-437', 'Database Systems', 'Comp. Sci', 4);
```

In this example, the values are specified in the order in which the corresponding attributes are listed in the relation schema.

It also can be written as:

```
insert into course (course_id, title, dept_name, credits)
        values ('CS-437', 'Database systems', 'Comp. Sci', 4);
```

## b) Deletions :

A delete request is expressed in much the same way as a query. We can delete only whole tuples; we cannot delete values on only particular attributes.

SQL expression of delete operation :

```
delete from r
where P;
```

where P represents Predicate and r represents a relation.

The delete statement first finds all tuples t in r for which P(t) is true, and then deletes them from r. The where clause can be omitted; in which case all tuples in r are deleted.

A delete command operates on only one relation. If we want to delete tuples from several relations, we must use one delete command for each relation.

Example :

```
delete from instructor;
```

where 'instructor' is a table name.

Another example :

Delete all tuples in the instructor relation pertaining to instructors in the Finance department.

Query →

```
delete from instructor
where dept_name = 'Finance';
```

c) Update :

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose, the update statement can be used.

For example,

Suppose that annual salary increases are being made, and salaries of all instructors are to be increased by 5 percent. We write:

```
update instructor
set salary = salary * 1.05 ;
```

Source code :

```
use master
create database University_180610
--To use the University_180610 database
use University_180610

-- Create a table named department which has 3 attribute
create table department (
dept_name varchar(20),
building varchar(15),
budget numeric(12,2),
Primary key(dept_name));
```

```
insert into department values('Biology', 'Watson', 90000)
insert into department values('Comp.Sci', 'Taylor', 100000)
insert into department values('Elec.Eng', 'Taylor', 85000)
insert into department values('Finance', 'Painter', 120000)
insert into department values('History', 'Painter', 50000)
insert into department values('Music', 'Packard', 80000)
insert into department values('Physics', 'Watson', 70000)

-- To show the department table with attributes and values
-- i insert into it
select * from department

-- To delete one tuple
delete from department where dept_name='Biology'

-- To update department
update department set budget = budget * 1.05 where budget < 85000
```

Output:

| | dept_name | building | budget |
|---|---|---|---|
| 1 | ~~Comp.Sci~~ Biology | ~~Taylor~~ Watson | 90000.00 |
| 2 | Comp.Sci. | Taylor | 100000.00 |
| 3 | Elec. Eng. | Taylor | 85000.00 |
| 4 | Finance | Painter | 120000.00 |
| 5 | History | Painter | 50000.00 |
| 6 | Music | Packard | 80000.00 |
| 7 | physics | Watson | 70000.00 |

| | dept_name | building | budget |
|---|---|---|---|
| 1 | Comp. Sci. | Taylor | 100000.00 |
| 2 | Elec. Eng. | Taylor | 85000.00 |
| 3 | Finance | Painter | 120000.00 |
| 4 | History | Painter | 50000.00 |
| 5 | Music | Packard | 80000.00 |
| 6 | Physics | Watson | 70000.00 |

| | dept_name | building | budget |
|---|---|---|---|
| 1 | Comp. Sci. | Taylor | 100000.00 |
| 2 | Elec. Eng. | Taylor | 85000.00 |
| 3 | Finance | Painter | 120000.00 |
| 4 | History | Painter | 52500.00 |
| 5 | Music | Packard | 84000.00 |
| 6 | Physics | Watson | 73500.00 |

Experiment no. : 02

Name of the experiment: Study and Implementation of DDL commands of SQL with suitable example.

- Create
- Alter
- Droop

Objectives:

1. To understand and use of Data Definition Language to write query for database.
2. To study how to create, alter and droop table in database.

Theory:

We define SQL relation by using the create table command.

Data Definition Language: We specify a database schema by a set of definitions expressed by a special language called a data definition language (DDL). The DDL is also used to specify additional properties of the data.

## Create :

The following command creates a relation department in the database :

```
create table department
    (dept_name varchar(20),
     building  varchar(15),
     budget    numeric(12,2),
     Primary key (dept_name));
```

The general form of create table command is :

```
create table r
    (A1 D1,
     A2 D2,
     ----
     An Dn,
     (integrity constraint1),
     ---,
     (integrity constraintn));
```

where r is the name of the relation, each $A_i$ is the name of an attribute in the schema of relation r and $D_i$ is the domain of attribute $A_i$ ; that is, $D_i$ specifies the type of attribute $A_i$ along with the optional constraints that restrict the set of allowed values for $A_i$.

The semicolon shows the end of the create table statement.

## Drop:

The drop command deletes all information about the dropped relation from the database.

The command is:

```
drop table r;
```

## Alter:

We use the alter command to add attribute to an existing relation. All tuples in the relation are assigned null as the value for the new attribute. The form of the alter table command is:

```
alter table r add A D;
```

where r is the name of an existing relation,
A is the name of the attribute to be added
and D is the type of the added attribute.

We can drop attributes from a relation by the command:

```
alter table r drop A;
```

where r is the name of the existing relation
A is the name of an attribute of the relation.

Source code :

```
use master
create database University
use University
-- Create a table named instructor
create table instructor (
  ID varchar (5),
  name  varchar (20) not Null ,
  dept_name varchar (20),
  Salary  numeric (8,2),
  Primary key (ID)
  for );

insert into instructor values ('10101', 'Srinivasan', 'Comp. Sci.', 65000)
insert into instructor values ('12121', 'Wu', 'Finance', 90000)
insert into instructor values ('15151', 'Mozart', 'Music', 40000)
insert into instructor values ('22222', 'Einstein', 'Physics', 95000)
insert into instructor values ('32343', 'El Said', 'History', 60000)
insert into instructor values ('33456', 'Gold', 'Physics', 87000)

select * from instructor

alter table instructor add course_no char (20) ;

drop table instructor
```

Output:

| | ID | name | dept_name | Salary |
|---|---|---|---|---|
| 1 | 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 2 | 12121 | Wu | Finance | 90000.00 |
| 3 | 15151 | Mozart | Music | 40000.00 |
| 4 | 22222 | Einstein | physics | 95000.00 |
| 5 | 32343 | El said | History | 60000.00 |
| 6 | 33456 | Gold | physics | 87000.00 |

| | ID | name | dept_name | Salary | course_no |
|---|---|---|---|---|---|
| 1 | 10101 | Srinivasan | Comp. Sci. | 65000.00 | NULL |
| 2 | 12121 | Wu | Finance | 90000.00 | NULL |
| 3 | 15151 | Mozart | Music | 40000.00 | NULL |
| 4 | 22222 | Einstein | physics | 95000.00 | NULL |
| 5 | 32343 | El Said | History | 60000.00 | NULL |
| 6 | 33456 | Gold | physics | 87000.00 | NULL |

Experiment no. : 03

Name of the Experiment: Study and Implementation of DML Commands of

- Select Clause
- From clause
- Where clause

Objectives:

1. To understand and use of the SQL queries.
2. To study how to implement select, from, where clause in database.

Theory:

The basic structure of an SQL queries consists of three clauses:

- select
- from
- where.

A query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses and then produces a relation as the result.

The role of each clause is as follows:

- The select clause is used to list the attributes desired in the result of a query.
- The from clause is a list of the relations to be accessed in the evaluation of the query.
- The where clause is a Predicate involving attributes of the relation in the from clause.

A typical SQL query has the form:

```
select  A1, A2, ---, An
from r1, r2, ----, rm
where P;
```

Each Ai represents an attribute and each ri a relation and P is a Predicate. If the where clause is omitted, the Predicate P is true.

## Queries on a single relation

Let us consider an example,

"Find the names of all the instructors". Instructor names are found in the instructor relation, so we put that relation in the from clause. The instructor's name appears in the name attribute, so we put that in the select clause.

```
select name
from instructor ;
```

The result is a relation consisting of a single attribute with the heading name.

## Queries on Multiple Relations

Suppose we want to answer the following query:
"Retrieve the names of all instructors along with their department names and department building name".

To answer the query, each tuple in the instructor relation must be matched with the tuple in the department relation whose dept_name value matches the dept_name value of the instructor tuple.

The query can be written in SQL as –

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name = department.dept_name;
```

The select clause may contain arithmetic expressions involving the operations +, –, * and / operating on constants or attributes of tuples. For example,

```
select ID, salary*1.1
from instructor;
```

Source code:

```
use University
select dept_name from instructor;
select name
from instructor
where dept_name = 'Comp. Sci' 'physics';
```

Output:

| | dept_name |
|---|---|
| 1 | Comp. Sci. |
| 2 | Finance |
| 3 | Music |
| 4 | physics |
| 5 | History |
| 6 | physics |

| | name |
|---|---|
| 1 | Enstein |
| 2 | Gold |

Experiment no. : 04

Name of the Experiment: Study and Implementation of DML commands of

- Group By & Having Clause
- Order By clause
- Create view, Indexing & Procedure clause.

Objectives:

1. To understand and use of the SQL queries.
2. To study how to implement group by, Having, order By clause in SQL code.
3. To create view, Indexing and Procedure clause in database.

Theory:

There are circumstances where we would like to apply aggregate function, in these case we use group by and having clause.

The aggregate functions are:

- Average : avg
- Maximum : max
- Minimum : min

- Total : sum
- Count : count

## Group By Clause :

In some cases we apply aggregate function not only to a single set of tuples, but also to a group of sets of tuples ; we specify this in SQL using group by clause. The attribute or attributes given in the group by clause are placed in one group.

Example:

Find the average salary in each department.

Query :

```
select dept_name, avg(salary) as avg-salary
from instructor
group by dept_name;
```

## Having Clause :

It is useful to state a condition that applies to groups rathere than to tuples.

For example, we want to see only those departments where the average salary of the instructors is more than 42000. This condition does not apply to a single tuple, rather it applies to each group constructed by the group by clause. To express such a query, we use having clause.

We express this query:

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary) > 42000;
```

## Order By Clause:

SQL offers the users to some control over the order in which tuples in a relation are displayed. The "order By" clause causes the tuples in the result of a query to appear in sorted order.

For example,

```
select *
from instructor
order by salary desc, name asc;
```

By default, the order By clause lists items in ascending order. To specify the sort order, we may specify "desc" for descending order, "asc" for ascending order.

## Create view:

We define a view in SQL by using the create view command. To define a view, we must give the view a name and must state the query that computes the view.

The form of the create view command is:

create view v as <query expression>;

where <query expression> is any legal query expression. The view name is represented by v.

## Indexing:

An index on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.

We create an index with the create index command, which takes the form:

create index <index-name> on <relation-name> (<attribute-list>);

create index dept_index on instructor (dept_name);

## Procedure:

A stored Procedure is a set of SQL statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.

Syntax:

```
CREATE PROCEDURE Procedure_name
AS
BEGIN
SQL QUEURY
END
EXEC procedure_name;
```

Source code:

```
use University
select dept_name, avg(salary) as avg_salary
from instructors
group by dept_name;

select dept_name, avg(salary) as avg_salary
from instructors
group by dept_name
having avg(salary) > 42000;

select * from instructors orders by salary desc, name asc;

create view faculty as
select ID, name, dept_name
from instructors.

create index dept_index on instructors (dept_name);

CREATE PROCEDURE instruct_Proc
AS
BEGIN
select name as authors_name from instructors where ID='15151'
END
EXEC instruct_Proc

select * from instructors.
```

Output:

| | dept_name | avg_salary |
|---|---|---|
| 1 | Comp. Sci. | 65000.000000 |
| 2 | Finance | 90000.000000 |
| 3 | History | 60000.000000 |
| 4 | Music | 40000.000000 |
| 5 | physics | 91000.000000 |

| | dept_name | avg_salary |
|---|---|---|
| 1 | Comp. Sci. | 65000.000000 |
| 2 | Finance | 90000.000000 |
| 3 | History | 60000.000000 |
| 4 | physics | 91000.000000 |

| | ID | name | dept_name | salary |
|---|---|---|---|---|
| 1 | 22222 | Einstein | Physics | 95000.00 |
| 2 | 12121 | Wu | Finance | 90000.00 |
| 3 | 33456 | Gold | physics | 87000.00 |
| 4 | 10101 | Srinivasan | Comp. Sci | 65000.00 |
| 5 | 32343 | El Said | History | 60000.00 |
| 6 | 15151 | Mozart | Music | 40000.00 |

| | authors_name |
|---|---|
| 1 | Mozart |

Experiment no. : 05

Name of the Experiment: Study and Implementation of SQL Commands of Join Operations with Example.

- Cartesian Product
- Natural Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

Objectives :

1. To understand and use of the Cartesian Product in SQL queries on database.
2. To study the commands of Join operations and their implementation on database.

Theory :

Join Operation : Join operations that allow the Programmer to write some queries in a more natural way and to express some queries that are difficult to do with only the Cartesian Product.

The goal of creating a join condition is that it helps to combine the data from two or more DBMS tables.

It is denoted by ⋈.

## The Natural Join :

The natural join operation operates on two relations and produces a relation as the result. Unlike the Cartesian Product of two relations which concatenates each tuple of the first relation with every tuple of the second, natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.

For example,

We write the query for all students in the university who have taken some course, find their names and the course ID of all courses they took" as :

```
select name, course_id
from student natural join takes;
```

## Syntax :

```
select A1, A2 ... An
from r1 natural join r2 natural join ... natural join rm
where P;
```

## Outer-join:

The outer-join operation works in a manner similar to the join operations, but it preserves those tuples that would be lost in a join by creating tuples in the result containing null values.

### There are three forms of outer join:

1) The Left-outer join –

The left outer join preserves tuples only in the relation named before the left outer join operation.

Example:

```
select *
from r1 natural left outer join r2;
```

2) The Right-Outer join —

The Right-outer join preserves tuples only in the relation named after the right outer join operation.

Example:

```
select A1, A2, ..., An
from r1 natural right outer join r2;
```

### iii) The Full Outer Join —

The Full Outer Join preserves tuples in both relations. It is the union of a left outer join and the corresponding right outer join.

```
select *
from r1 natural full outer join r2;
```

**Cartesian Product:** The Cartesian Product operation allow us to combine information from any two relation.

**Source code:**

```
use University
select * from instructors
select * from departments
~~select building~~
--- Cartesian Product
select building, departments.dept_name, salary ~~from~~
from departments, instructors
where departments.dept_name = instructors.dept_name
--- join operation
select salary, building
from departments join instructors on
departments.dept_name = instructors.dept_name
```

--- left outer join

```
select * from departments left outer join instractors on
departments.dept_name = instructors.dept_name
```

--- Right Outer join

```
select * from instructors right outer join departments on
departments.dept_name = instructors.dept_name
```

--- Full Outer join

```
select * from instructors full outer join departments on
departments.dept_name = instructions.dept_name
```

Output:

Cartesian Product

| | building | dept_name | salary |
|---|---|---|---|
| 1 | Taylor | Comp. Sci. | 65000.00 |
| 2 | Painter | Finance | 90000.00 |
| 3 | Packard | Music | 40000.00 |
| 4 | Watson | physics | 95000.00 |
| 5 | Painter | History | 60000.00 |
| 6 | Watson | physics | 87000.00 |

## join operation:

| | Salary | building |
|---|---|---|
| 1 | 65000.00 | Taylor |
| 2 | 90000.00 | Painter |
| 3 | 40000.00 | Packard |
| 4 | 95000.00 | Painter Watson |
| 5 | 60000.00 | Painter |
| 6 | 87000.00 | Watson |

## left outer join:

| | dept_name | building | budget | ID | name | dept_name | Salary |
|---|---|---|---|---|---|---|---|
| 1 | Biology | Watson | 90000.00 | NULL | NULL | NULL | NULL |
| 2 | Comp. Sci. | Taylor | 100000.00 | 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 3 | Elec. Eng. | Taylor | 85000.00 | NULL | NULL | NULL | NULL |
| 4 | Finance | Painter | 1200000.00 | 12121 | Wu | Finance | 90000.00 |
| 5 | History | Painter | 50000.00 | 32343 El Said | El Said | History | 60000.00 |
| 6 | Music | Packard | 80000.00 | 15151 Mozart | Mozart | Music | 40000.00 |
| 7 | Physics | Watson | 70000.00 | 22222 Einstein | Einstein | physics | 95000.00 |
| 8 | physics | Watson | 70000.00 | 33456 Gold | physics Gold | physics | 87000.00 |

## Right outer join:

| | ID | name | dept_name | salary | dept_name | building | budget |
|---|---|---|---|---|---|---|---|
| 1 | NULL | NULL | NULL | NULL | Biology | Watson | 90000.00 |
| 2 | 10101 | Srinivasan | Comp. Sci. | 65000.00 | Comp. Sci. | Taylor | 100000.00 |
| 3 | NULL | NULL | NULL | NULL | Elec. Eng. | Taylor | 85000.00 |
| 4 | 12121 | Wu | Finance | 90000.00 | Finance | Painter | 1200000.00 |
| 5 | 32343 | El Said | History | 60000.00 | History | Painter | 50000.00 |
| 6 | 15151 | Mozart | Music | 40000.00 | Music | Packard | 80000.00 |
| 7 | 22222 | Einstein | physics | 95000.00 | physics | Watson | 70000.00 |
| 8 | 33456 | Gold | Physics | 87000.00 | Physics | Watson | 70000.00 |

Full Outer join :

| | ID | name | dept_name | salary | dept_name | buiding | budget |
|---|---|---|---|---|---|---|---|
| 1 | 10101 | Srinivasan | Comp. Sci. | 65000.00 | Comp. Sci | Taylor | 100000.00 |
| 2 | 12121 | Wu | Finance | 90000.00 | Finance | Painter | 120000.00 |
| 3 | 15151 | Mozart | Music | 40000.00 | Music | Packard | 80000.00 |
| 4 | 22222 | Einstein | Physics | 95000.00 | Physics | Watson | 70000.00 |
| 5 | 32343 | El Said | History | 60000.00 | History | Painter | 50000.00 |
| 6 | 33456 | Gold | Physics | 87000.00 | Physics | Watson | 70000.00 |
| 7 | NULL | NULL | NULL | NULL | Biology | Watson | 90000.00 |
| 8 | NULL | NULL | NULL | NULL | Elec. Eng. | Taylor | 85000.00 |

Experiment no. : 06

Name of the experiment: Study and Implementation of Aggregate Function with example

- Count Function
- Max Function
- Min Function
- Avg Function.

Objectives:

1. To understand and use of the aggregate function on database.
2. To study how to implement count(), max(), min(), avg() function in SQL on database.

Theory:

Aggregate functions are functions that take a collection of values as input and return a single value.

SQL offers five standard built-in aggregate functions.

- Average : avg
- Minimum : min
- Maximum : max
- Total : sum

• count : count

The input to sum and avg must be a collection of numbers, but the other operations can operate on collections of nonnumeric data types, such as string as well.

# Avg(): This avg() function returns the average value of some collection of numbers in a data table.

Example:

```
select avg(salary) as average_salary
from instructor
where dept_name = "physics";
```

# Min(): This min() function returns the minimum value from some collection of numbers under an attribute in a relation.

Example:

```
select min(salary) as minimum_salary
from instructor
where dept_name = "physics";
```

This query returns the minimum salary of a physics department.

# max(): max() function returns the maximum value from some collection of numeric values under an attribute in a relation.

Example:

```
select max(salary) as maximum_salary
from instructor
where dept_name = "physics";
```

This query returns the maximum salary of a physics department.

# count(): count() function returns number of tuples in a relation usually.

The notation for this function in SQL is count(*).

Example:

To find the number of tuples in the instructor relation we can write:

```
select count(*)
from instructor;
```

There are some cases where we ~~want~~ must eleminate duplicates before computing an aggregate function.

For example:

```
select count(distinct ID)
from teaches
where semester = 'Spring' and year = 2018;
```

This returns the total number of instructors who teach a course in the Spring 2018 semester.

Source code:

```
use University
select count (ID) as count_ID
from instructor;

select max(salary) as max_salary from instructors;

select min(salary) as min_salary from
from instructor;

select avg(salary) as avg_salary
from instructor;
```

Output:

| | count_ID |
|---|---|
| 1 | 6 |

| | max_salary |
|---|---|
| 1 | 95000.00 |

| | min_salary |
|---|---|
| 1 | 40000.00 |

| | avg_salary |
|---|---|
| 1 | 72833.333333 |

Experiment no. : 07

Name of the experiment: Study and Implementation of Triggering System on Database Table using SQL commands with example.

Objectives:

1. To understand the triggering system on Database table.
2. To understand how trigger can be used for automatically updating a table when an insert/update/delete statement takes place in another table.

Theory:

Trigger is a special type of Procedure which is attached to a table and is only executed when an Insert, Update or Delete occurs on that table. One has to specify the modification actions that fire the triggers when it is created.

Unlike stored Procedures, trigger can not be explicitly executed.

Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

Trigger cannot usually perform updates outside the database, Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL.

Syntax:

```
CREATE TRIGGER trigger_name on table_name
FOR INSERT | DELETE | UPDATE
AS
BEGIN
"TRIGGER BODY"
END
```

Where create trigger creates or replaces an existing trigger with the trigger_name.

on table_name : This specifies the name of the table associated with the trigger.

FOR INSERT | DELETE | UPDATE : This specifies the DML operation.

Trigger_body : This Provides the operation to be performed as trigger is fired.

## Source code :

```
create database ~~temp~~ storetable
use storetable
create table Customers
(cust-id char(5) Primary key check (cust_id like('[cs][0-9][0-9]
                                                  [0-9][0-9]')),
 cust-fname char(12) NOT NULL,
 cust-lname varchar(12),
 cust_address TEXT);

insert Customer values ('c0001', 'Laila', 'Binta', 'Pabna')

create table Items
(
 item_id char(5) Primary key check (item_id like('[P][0-9][0-9]
                                                  [0-9][0-9]')),
 item_name char(12),
 item_category char(10),
 item_Price float(12) check (item_Price>=0),
 item_qoh int check (item_qoh>=0),
 item_last_sold datetime default getdate()
)

select * from Items
insert Items values ('P0001', 'Jamuna', 'laptop', 125.5, 26, '10-2-2022')
insert Items values ('P0003', 'Realme Jamy', 'phone', 80.5, 30, '5-14-2022')
insert Items values ('P0004', 'HP ~~Ambra~~', 'laptop', 130.5, 20, '2-18-2022')
```

```
create table transact
(
tran_id char(8) check (tran_id like ('[T][0-9][0-9][0-9][0-9]
                                      [0-9][0-9][0-9]')),
item_id char(5) foreign key references Items (item_id),
cust_id char(5) foreign key references Customer(cust_id),
tran_type char(1),
tran_quantity int check (tran_quantity >0),
tran_date datetime default getdate()
)
select * from transact
-- Triggers
Create trigger test on Items FOR INSERT
AS
BEGIN
Declare @item_id char(5), @amount char(12), @tran_type char(1)
select @item_id = item_id, @amount = tran_quantity,
@tran_type = tran_type from inserted
IF (@tran_type = 'S')
    Update Items set item_qoh = item_qoh - @amount where
     item_id = @item_id
ELSE
   Update Items set item_qoh = item_qoh + @amount where
    item_id = @item_id
END
insert transact values ('T00000001', 'P0003', 'C0001', 'S', 5, '3-12-2022')
select * from transact
select * from Items
```

Output:

Before ~~tran~~ triggering "Items" table

| | item_id | item_name | item_category | item_price | item_qoh | item_last_sold |
|---|---|---|---|---|---|---|
| 1 | P0001 | Jamuna | laptop | 125.5 | 26 | 2022-10-02 |
| 2 | P0003 | Realme | phone | 80.5 | 30 | 2022-05-14 |
| 3 | P0004 | HP | laptop | 130.5 | 20 | 2022-02-18 |

Before triggering "transact" table

| tran_id | item_id | cust_id | tran_type | tran_quantity | tran_date |
|---|---|---|---|---|---|

After triggering "transact" table

| | tran_id | item_id | cust_id | tran_type | tran_quantity | tran_date |
|---|---|---|---|---|---|---|
| 1 | T00000001 | P0003 | C0001 | S | 5 | 2022-3-12 |

After triggering "Items" table

| | item_id | item_name | item_category | item_price | item_qoh | item_last_sold |
|---|---|---|---|---|---|---|
| 1 | P0001 | Jamuna | laptop | 125.5 | 26 | 2022-10-02 |
| 2 | P0003 | Realme | phone | 80.5 | 25 | 2022-05-14 |
| 3 | P0004 | HP | laptop | 130.5 | 20 | 2022-02-18 |