INLS 623
4/8/2013

Phase Two Design Report

TeamB : Jennifer Martin, Koushyar Rajavi, Lei Wang, Shumeng Gu

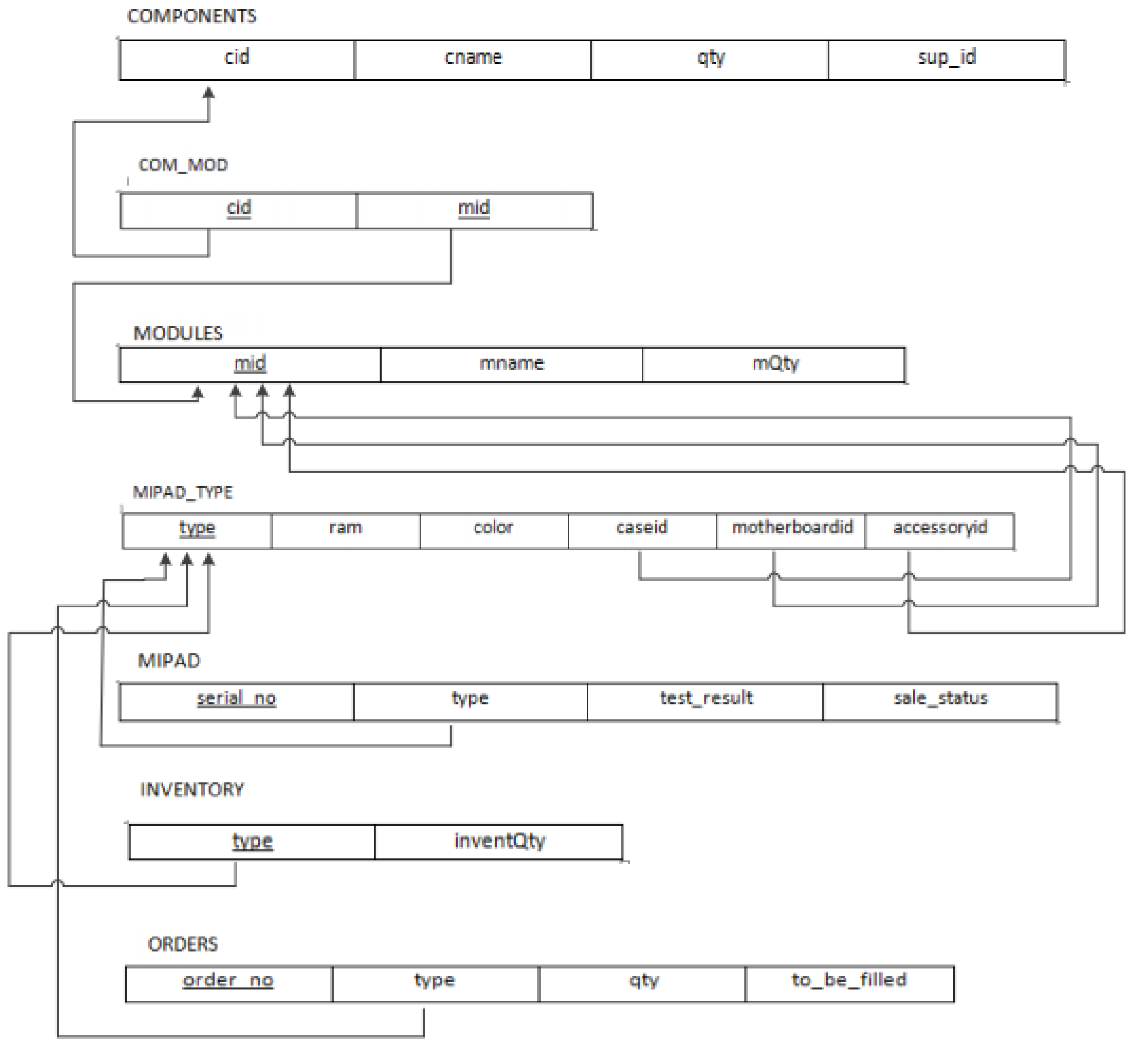
Overall Description

Our database captures at an abstract level the processes of taking components, manufacturing modules from the components, assembling the modules into different types of miPads, testing the miPads, and fulfilling orders for the miPads. Because of the bi-directional linear flow of information and materials (information mostly back, materials forward), the relationships among the tables most closely resemble a chain. Triggers are in place to automate the processing of orders, automatically passing completed miPads or modules forward to be used in the next stage and passing information on what is needed back to trigger creation. A trigger is also used to test newly created miPads and to order replacement components when we are running low.

The goal of our design was to have a database that could react to orders given to us by Team C, fulfilling the orders automatically and without needing our intervention, exchanging information/materials all the way back through to placing orders with Team A. Our primary interaction with Team C flows two ways: they place orders into our Orders table, and we fulfill those orders by writing to their inventory table. Our primary interaction with Team A also flows two ways: we write orders to their Order table, and they fulfill the order by increasing our inventory of components in our Components table. Thus, for the interactions with both teams, orders are passed back and physical materials passed forward.

We chose to ignore the human elements of the supply chain, as they did not contribute to the conceptual model we were working with, which focused on the flow of components, modules, and miPads. Obviously, in real life actual workers would be required to carry out many of the tasks alluded to in our database, but they essentially disappear at the level of abstraction we are working with.

Table Schema and Descriptions



```
CREATE TABLE Components(
cid INT NOT NULL,
cname VARCHAR(25) NOT NULL,
qty INT NOT NULL,
sup_id Char(5) NOT NULL,
PRIMARY KEY(cid),
CONSTRAINT fk_xx
FOREIGN KEY(sup_id)
REFERENCES Supplier(sup_id ) ON DELETE SET NULL
);
```

The table Components records the components we purchase from Team A's suppliers and use in creating modules. The table records a unique identifier for each component (cid)--the value of which is the same across Team A and Team B--the name of each component (cname), the amount of each component we have in stock (qty), and the id of the supplier that we order each component from (supp). When our orders to Team A are fulfilled, the qty field will be updated to reflect the number of components added. The table references Team A's Supplier table to keep the supplier names consistent. Components also has a relationship with Com_mod, specifying which components are used for which modules.

```
CREATE TABLE Com_mod(
cid INT NOT NULL,
mid INT NOT NULL,
CONSTRAINT com_mod_pk PRIMARY KEY(cid, mid),
CONSTRAINT fk_com
FOREIGN KEY(cid)
REFERENCES Components(cid) ON DELETE SET NULL,
CONSTRAINT fk_mod
FOREIGN KEY(mid)
REFERENCES Modules(mid) ON DELETE SET NULL
);
```

The table Com_mod connects the components to the modules they are used to create. The entities "components" and "modules" have a many-to-many relationship, so this table bridges that.

```
CREATE TABLE Modules(
mid INT NOT NULL,
mname VARCHAR(25) NOT NULL,
mQty INT NOT NULL,
CONSTRAINT mod_pk PRIMARY KEY(mid, qty)
);
```

The table Modules records information about each module type that is manufactured. It includes a unique id for each type of module (mid), the name of the module (mname), and the number of each type of module we currently have in stock (qty). It is related to the table Com_mod, which records what components go into making each module, and to the table Types, which records what modules are used to create which types of miPad.

```
CREATE TABLE Types(
type CHAR(5) NOT NULL,
ram CHAR(5) NOT NULL,
color CHAR(5) NOT NULL,
caseid INT NOT NULL,
motherboardid INT NOT NULL,
accessoryid INT NOT NULL,
CONSTRAINT types_pk PRIMARY KEY(type),
CONSTRAINT fk_mod1
FOREIGN KEY(caseid)
REFERENCES Modules(mid),
CONSTRAINT fk_mod2
FOREIGN KEY(motherboardid)
REFERENCES Modules(mid),
CONSTRAINT fk_mod3
FOREIGN KEY(accessoryid)
REFERENCES Modules(mid),
);
```

The table Types records information about the types of miPads we assemble. Each type has a unique name (type), a given amount of RAM (ram), a color (color), and is made up of three modules: a case (caseid), an integrated motherboard (motherboardid), and a module holding several other parts and accessories (accessoryid). The modules fields reference the Modules table. The Types table is also related to both the MiPad table, which records information about the individual miPads produced, and to the inventory table, which tracks the number of miPads of each type we have.

```
CREATE TABLE MiPad(
serial_no INT NOT NULL,
type VARCHAR(5) NOT NULL,
test_result VARCHAR(2) NOT NULL,
sale_status VARCHAR(2) NOT NULL,
CONSTRAINT mipad_pk PRIMARY KEY(pid),
CONSTRAINT fk_type
FOREIGN KEY(type)
REFERENCES Types(type),
);
```

The table MiPad records information for each individual miPad we make. Each miPad has a unique identifier (pid), is of a given type (type), and is sold or unsold (sale_status). The table also records whether or not a miPad has been tested yet and, if it has been tested, if it passed or failed the test (test_result). MiPad references the Types table to indicate what types of miPad can be and are present.

```
CREATE TABLE Inventory(
type VARCHAR(5) NOT NULL,
qty INT,
CONSTRAINT Inventory_pk PRIMARY KEY(type),
CONSTRAINT fk_type2
FOREIGN KEY(type)
REFERENCES Types(type),
);
```

The table Inventory records numbers of currently-in-stock, ready-to-be-sold miPads. The table includes the type of miPad (type) and the number of that type of miPad that are available for immediate sale (inventQty). Like the MiPad table, Inventory references the Types table for referential integrity on the types of miPads available.

```
CREATE TABLE Orders(
order_no INT NOT NULL,
type VARCHAR(5) NOT NULL,
qty INT NOT NULL,
to_be_filled INT NOT NULL,
CONSTRAINT order_pk PRIMARY KEY(oid),
CONSTRAINT fk_inv
FOREIGN KEY(type)
REFERENCES Inventory(type) ON DELETE SET NULL
);
```

The table Orders records the orders that we receive from Team C. Each order has a unique, identifying number (oid), is for a type of miPad (type), gives the number of miPads wanted (qty), and records how many miPads still need to be provided to Team C to complete the order (to_be_filled). This is the table Team C will insert into the order miPads from us.

Triggers and Trigger Descriptions

Initial set of Triggers

Below is the initial set of triggers we created along with each trigger's description. We initially tried to have everything done via triggers as we were expected to do so according to the project description; however, we were having some troubles implementing them into Oracle, because these triggers were causing deadlocks with themselves. That is why we are changing the initial ones into the set of new triggers and stored procedures.

1. This trigger would be called when Team C places a new order into our Orders table. After the insertion, the trigger would check whether there are quantities of the ordered type in the INVENTORY table.

```
CREATE OR REPLACE TRIGGER new_order_group_c
After INSERT OR UPDATE on Orders
FOR EACH ROW
Begin(
        If (:New.to_be_filled>0) (
        Update Inventory set qty=qty-1 where type=:new.type;
        update orders set to_be_filled=to_be_filled-1 where type=:new.type and id=:new.id
        )
        end if;
)
END;
```

2. This trigger would be called when the quantity column in the Inventory table changed. If the quantity decreases, it means one miPad in the inventory is to be delivered to Team C, and the to_be_filled value in the Order Table will also decrease by one. If the quantity increases, it indicates that a new miPad has been produced and passed the test and so has moved into the inventory.

```
CREATE OR REPLACE TRIGGER inventory
After UPDATE on Inventory
FOR EACH ROW
declare x varchar;
declare y number;
declare z number;
BEGIN
[*** this is the case when we are adding newly produced items to the inventory ***]
if ( :New.Qty > (select I.qty from inventory I where I.type=:New.type) )
( // addition to inventory will be done by the trigger that called it, here we don't need to worry about it... )
        select o.to_be_filled into z from oders o where o.type=:new.type;
        if (z=:new.qty) then update orders set to_be_filled=0 where type=:new.type;
                update inventory set qty=0 where type=:new.type;
                call stored proc to set the last z mipads to sold
        end if;
end if;

else if ( :New.Qty < (select I.qty from inventory I where I.type=:New.type) ) {
[*** this is when we are delivering orders to group C ***]
select o.to_be_filled into z from oders o where o.type=:new.type;
if (:New.Qty=0 and z>0) (then update Mipad set Mipad.sale_status=sold where mipad.id=(select
min(m.id) from mipad m where m.type=:new.type and m.sale_status='not sold');
:New.qty=:New.qty+1;) end if;
}
end if;
End
```

3. This trigger would be called when no remaining miPads are left in the Inventory table and a certain type of miPad needs to be produced. We suppose that to create type A we need modules with id# 1, 3, and 5; to create type B, we need modules 1, 4, and 5; to create type C, we need modules 2, 3, and 5; and to create type C, we need modules 2, 4, and 5.
To produce a certain type of miPad, we will check the remaining numbers of modules, and if the remaining modules are able to produce the miPad we need, their quantities will decrease by 1.

```
CREATE OR REPLACE TRIGGER Mipad
After update of sale_status on Mipad
declare y int;
declare z int;
```

```
for each row
Begin
select count(*) into y from mipad M where M.type=:New.type and M.status=”success” and
M.sale_status=”not sold”;
select o.to_be_filled into z from oders o where o.type=:new.type;

if (z=0) Exit;
elseif (y=0) THEN  {
        if (:New.type=”A”) then {
                update modules set qty=qty-1 where id=1;
                update modules set qty=qty-1 where id=3;
                update modules set qty=qty-1 where id=5;
        }
        elseif (:New.type=”B”) then {
                update modules set qty=qty-1 where id=1;
                update modules set qty=qty-1 where id=4;
                update modules set qty=qty-1 where id=5;
        }
        elseif (:New.type=”C”) then {
                update modules set qty=qty-1 where id=2;
                update modules set qty=qty-1 where id=3;
                update modules set qty=qty-1 where id=5;
        }
        if (:New.type=”D”) then {
                update modules set qty=qty-1 where id=2;
                update modules set qty=qty-1 where id=4;
                update modules set qty=qty-1 where id=5;
        }
        end if;
}
END IF;

END
```

4. This trigger would be called during the testing process. When a new row is inserted into the miPad table, indicating that a new miPad has been produced, the trigger will wait for 5 seconds and then print out sentences which mean that the miPad is in the testing process.

```
CREATE OR REPLACE TRIGGER Mipad_add
After Insert of type on Mipad
For each row
declare x number;
Begin
//wait for 2 or 3 seconds
```

```
dbms_job.submit
    ( job => l_job, what => 'myproc(:new.id);',, next_date => sysdate+5/24/60/60 -- 5 seconds later);
//print on the console that you are waiting for test
DBMS_OUTPUT.put_line('mipad is being tested');
update Mipad set status="success" where id=:new.id;
update Mipad set sale_status="for sale" where id=:new.id;
DBMS_OUTPUT.put_line('mipad test was successful');
update Inventory set qty=qty+1 where type=:new.type;
DBMS_OUTPUT.put_line('mipad sent to inventory after test');
END
```

5. This trigger would be called when a certain type of module needs to be produced. We suppose that to produce module id 1, we need components 1 and 2; for module id 2, components 3 and 4; for module 3, components 5 and 6; for module 4, components 7 and 8; and for module 5, components 9 and 10. When a certain type of module is out of stock, this trigger would decrease each component which was needed to produce the module by 1 and increase the module quantity by 1. When the quantity of the module increases, the trigger would check which miPads use the module. That miPad quantity would then be increased by 1, indicating that a miPad of that type had been produced, and the given module quantity, as well as the quantities of the other modules used in creating the miPad, would be decreased by 1.

```
CREATE OR REPLACE TRIGGER Modules
Before update of qty in modules
for each row
declare x number; declare y number;
declare a number; declare b number; declare c number; declare e number; declare f number;
declare aa number, declare bb number, declare cc number; declare dd number;
Begin
if (:New.qty=0) then [
        if (:New.Id=1) then {
                update components set qty=qty-1 where id=1;
                update components set qty=qty-1 where id=2;
        }end if;
        if (:New.Id=2) then {
                update components set qty=qty-1 where id=3;
                update components set qty=qty-1 where id=4;
        }end if;
        if (:New.Id=3) then {
                update components set qty=qty-1 where id=5;
                update components set qty=qty-1 where id=6;
        }end if;
        if (:New.Id=4) then {
                update components set qty=qty-1 where id=7;
                update components set qty=qty-1 where id=8;
        }end if;
        if (:New.Id=5) then {
                update components set qty=qty-1 where id=9;
                update components set qty=qty-1 where id=10;
        }end if;
] end if;
```

// for when the components have made a new module.

```
else if (:New.qty>0) then {
        select modules.qty into x from modules where modules.id=:New.id;
        if (:New.qty > x) then (
                select m.qty into a from modules m where m.id=1;
                select m.qty into b from modules m where m.id=2;
                select m.qty into c from modules m where m.id=3;
                select m.qty into d from modules m where m.id=4;
                select m.qty into e from modules m where m.id=5;
                //
                select i.qty into aa from inventory i where i.type=”A”;
                select i.qty into bb from inventory i where i.type=”B”;
                select i.qty into cc from inventory i where i.type=”C”;
                select i.qty into dd from inventory i where i.type=”D”;
                if (:New.id ==1) then [
                        if (c>0 and e>0 and aa==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”A”,’not tested’,’not available’);
                        )end if;
                        if (d>0 and e>0 and bb==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”B”,’not tested’,’not available’);
                        )end if;
                ]
                elseif (:New.id ==2) then [
                        if (c>0 and e>0 and cc==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”C”,’not tested’,’not available’);
                        )end if;
                        if (d>0 and e>0 and dd==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”D”,’not tested’,’not available’);
                        )end if;
                ]
                elseif (:New.id ==3) then [
                        if (a>0 and e>0 and aa==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”A”,’not tested’,’not available’);
                        )end if;
                        if (b>0 and e>0 and cc==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”C”,’not tested’,’not available’);
                        )end if;
                ]
                elseif (:New.id ==4) then [
                        if (a>0 and e>0 and bb==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”B”,’not tested’,’not available’);
                        )end if;
                        if (b>0 and e>0 and dd==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
```

```
                                y=y+1;
                                insert into mipad values (y,”D”,’not tested’,’not available’);
                        )end if;
                ]
                elseif (:New.id ==5) then [
                        if (a>0 and c>0 and aa==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”A”,’not tested’,’not available’);
                        )end if;
                        if (a>0 and d>0 and bb==0) then (
                                update modules set qty=qty-1 where modules.id=1;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”B”,’not tested’,’not available’);
                        )end if;
                        if (b>0 and c>0 and cc==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=3;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”C”,’not tested’,’not available’);
                        )end if;
                        if (b>0 and d>0 and dd==0) then (
                                update modules set qty=qty-1 where modules.id=2;
                                update modules set qty=qty-1 where modules.id=4;
                                update modules set qty=qty-1 where modules.id=5;
                                select max(m.id) into y from mipad m;
                                y=y+1;
                                insert into mipad values (y,”D",’not tested’,’not available’);
                        )end if;
                ]
        ) end if;

}end if;

End
```

6. This trigger would be called when the quantity of each component has changed. When the quantity reaches 0, it would place an order to Team A. Otherwise, when the quantity of the component decreases, it would check whether the other component which is needed to produce a module has remaining quantity. If so, a new module will be produced, and the number of these two components will be decreased by 1.

```
CREATE OR REPLACE TRIGGER Components
Before update of qty in components
For Each Row
declare x number; declare y number; declare xx number;
declare a number; declare b number; declare c number; declare d number; declare e number; declare f
number; declare g number; declare h number; declare i number; declare j number;
Begin
        select max(o.to_be_filled) into xx from orders o;
        xx=2*xx;

        if (:New.qty==0) then Insert into order_group_A set ID=:New.ID and qty=xx;

        select c.qty into x from components c where c.id=:New.id;
        elseif (:new.qty>0 and :new.qty<x) then
                select c.qty into a from components a where a.id=1;
                select c.qty into b from components a where a.id=2;
                select c.qty into c from components a where a.id=3;
                select c.qty into d from components a where a.id=4;
                select c.qty into e from components a where a.id=5;
                select c.qty into f from components a where a.id=6;
                select c.qty into g from components a where a.id=7;
                select c.qty into h from components a where a.id=8;
                select c.qty into i from components a where a.id=9;
                select c.qty into j from components a where a.id=10;
                if (:new.id=1 and b>0) then (
                        update modules set qty=qty+1 where id=1;
                        update components set qty=qty-1 where id=1;
                        update components set qty=qty-1 where id=2;
                )
                elseif (:new.id=2 and a>0) then (
                        update modules set qty=qty+1 where id=1;
                        update components set qty=qty-1 where id=1;
                        update components set qty=qty-1 where id=2;
                )
                elseif (:new.id=3 and d>0) then (
                        update modules set qty=qty+1 where id=2;
                        update components set qty=qty-1 where id=3;
                        update components set qty=qty-1 where id=4;
                )
                elseif (:new.id=4 and c>0) then (
                        update modules set qty=qty+1 where id=2;
                        update components set qty=qty-1 where id=3;
                        update components set qty=qty-1 where id=4;
                )
                elseif (:new.id=5 and f>0) then (
                        update modules set qty=qty+1 where id=3;
                        update components set qty=qty-1 where id=5;
                        update components set qty=qty-1 where id=6;
                )
                elseif (:new.id=6 and e>0) then (
                        update modules set qty=qty+1 where id=3;
                        update components set qty=qty-1 where id=5;
                        update components set qty=qty-1 where id=6;
                )
                elseif (:new.id=7 and h>0) then (
                        update modules set qty=qty+1 where id=4;
                        update components set qty=qty-1 where id=7;
                        update components set qty=qty-1 where id=8;
                )
                elseif (:new.id=8 and g>0) then (
                        update modules set qty=qty+1 where id=4;
                        update components set qty=qty-1 where id=7;
                        update components set qty=qty-1 where id=8;
                )
                elseif (:new.id=9 and j>0) then (
                        update modules set qty=qty+1 where id=5;
                        update components set qty=qty-1 where id=9;
                        update components set qty=qty-1 where id=10;
                )
                elseif (:new.id=10 and i>0) then (
                        update modules set qty=qty+1 where id=5;
                        update components set qty=qty-1 where id=9;
                        update components set qty=qty-1 where id=10;
                )
                end if;
```

// when group A has delivered components and we want to make modules out of them

```
        elseif (:New.qty>0) then [
                select c.qty into a from components a where a.id=1;
                select c.qty into b from components a where a.id=2;
                select c.qty into c from components a where a.id=3;
                select c.qty into d from components a where a.id=4;
                select c.qty into e from components a where a.id=5;
```

```
select c.qty into f from components a where a.id=6;
select c.qty into g from components a where a.id=7;
select c.qty into h from components a where a.id=8;
select c.qty into i from components a where a.id=9;
select c.qty into j from components a where a.id=10;
if (:New.qty>x) then (
        if (:new.id==1) then (
                if (b>0) then(
                        update components set qty=qty-1 where id=1;
                        update components set qty=qty-1 where id=2;
                        update modules set qty=qty+1 where id=1;
                ) end if;
        )
        elseif (:new.id==2) then (
                if (a>0) then(
                        update components set qty=qty-1 where id=1;
                        update components set qty=qty-1 where id=2;
                        update modules set qty=qty+1 where id=1;
                ) end if;
        )
        elseif (:new.id==3) then (
                if (d>0) then(
                        update components set qty=qty-1 where id=3;
                        update components set qty=qty-1 where id=4;
                        update modules set qty=qty+1 where id=2;
                ) end if;
        )
        elseif (:new.id==4) then (
                if (c>0) then(
                        update components set qty=qty-1 where id=3;
                        update components set qty=qty-1 where id=4;
                        update modules set qty=qty+1 where id=2;
                ) end if;
        )
        elseif (:new.id==5) then (
                if (f>0) then(
                        update components set qty=qty-1 where id=5;
                        update components set qty=qty-1 where id=6;
                        update modules set qty=qty+1 where id=3;
                ) end if;
        )
        elseif (:new.id==6) then (
                if (e>0) then(
                        update components set qty=qty-1 where id=5;
```

```
                    update components set qty=qty-1 where id=6;
                    update modules set qty=qty+1 where id=3;
                ) end if;
            )
            elseif (:new.id==7) then (
                if (h>0) then(
                    update components set qty=qty-1 where id=7;
                    update components set qty=qty-1 where id=8;
                    update modules set qty=qty+1 where id=4;
                ) end if;
            )
            elseif (:new.id==8) then (
                if (g>0) then(
                    update components set qty=qty-1 where id=7;
                    update components set qty=qty-1 where id=8;
                    update modules set qty=qty+1 where id=4;
                ) end if;
            )
            elseif (:new.id==9) then (
                if (j>0) then(
                    update components set qty=qty-1 where id=9;
                    update components set qty=qty-1 where id=10;
                    update modules set qty=qty+1 where id=5;
                ) end if;
            )
            elseif (:new.id==10) then (
                if (i>0) then(
                    update components set qty=qty-1 where id=9;
                    update components set qty=qty-1 where id=10;
                    update modules set qty=qty+1 where id=5;
                ) end if;
            )
            end if;
        )end if;

    ] end if;

END;
```

Therefore, I separated the two conditions:

```
create or replace
TRIGGER
Components
```

```
Before
update of qty on components
For
Each Row
DECLARE

x
number;
y
number;
a
number;
b
number;
c
number;
d
number;
e
number;
f
number;
g
number;
h
number;
i
number;
j
number;
Begin

        if :New.qty>:old.qty then
                        if :new.cid=1 then
                                select qty into b from components where cid=2;
if b>0 then
                                        update components set qty=qty-1 where cid=1;
                                        update components set qty=qty-1 where cid=2;
                                        update modules set qty=qty+1 where mid=1;
                                end if;
                        end if;
                        if :new.cid=2 then
                                if a>0 then
                                select qty into a from components where cid=1;
```

```
                    update components set qty=qty-1 where cid=1;
                    update components set qty=:new.qty-1 where cid=2;
                    update modules set qty=qty+1 where mid=1;
                end if;
            end if;
            if :new.cid=3 then
select qty into d from components where cid=4;
                if d>0 then
                    update components set qty=:new.qty-1 where cid=3;
                    update components set qty=qty-1 where cid=4;
                    update modules set qty=qty+1 where mid=2;
                end if;
            end if;
            if :new.cid=4 then
select qty into c from components where cid=3;
                if c>0 then
                    update components set qty=qty-1 where cid=3;
                    update components set qty=:new.qty-1 where cid=4;
                    update modules set qty=qty+1 where mid=2;
                end if;
            end if;
            if :new.cid=5 then
select qty into f from components where cid=6;
                if f>0 then
                    update components set qty=:new.qty-1 where cid=5;
                    update components set qty=qty-1 where cid=6;
                    update modules set qty=qty+1 where mid=3;
                end if;
            end if;
            if :new.cid=6 then
select qty into e from components where cid=5;
                if e>0 then
                    update components set qty=qty-1 where cid=5;
                    update components set qty=:new.qty-1 where cid=6;
                    update modules set qty=qty+1 where mid=3;
                end if;
            end if;
            if :new.cid=7 then
select qty into h from components where cid=8;
                if h>0 then
                    update components set qty=:new.qty-1 where cid=7;
                    update components set qty=qty-1 where cid=8;
                    update modules set qty=qty+1 where mid=4;
                end if;
```

```
                                end if;
                               if :new.cid=8 then
select qty into g from components where cid=7;
                                          if g>0 then
                                                    update components set qty=qty-1 where cid=7;
                                                    update components set qty=:new.qty-1 where cid=8;
                                                    update modules set qty=qty+1 where mid=4;
                                          end if;
                                end if;
                               if :new.cid=9 then
select qty into j from components where cid=10;
                                          if j>0 then
                                                    update components set qty=:new.qty-1 where cid=9;
                                                    update components set qty=qty-1 where cid=10;
                                                    update modules set qty=qty+1 where mid=5;
                                          end if;
                               end if;
                               if :new.cid=10 then
select qty into i from components where cid=9;                                  if i>0 then
                                                    update components set qty=qty-1 where cid=9;
                                                    update components set qty=:new.qty-1 where cid=10;
                                                    update modules set qty=qty+1 where mid=5;
                                           end if;
                                end if;
                     end if;
End;
```

Revised Triggers, Stored Procedures and Descriptions

Below are the revised triggers and stored procedures. We are still testing them.

1. This trigger will be called when teamC places an order and insert a new row into our Order Table. First it will print out the notice that Group C has placed an order with its information. Then it will call the procInventory stored procedure and pass the quantity, type and Order Id of the new order into the procedure as the parameters.

```
CREATE OR REPLACE TRIGGER Orders
After INSERT on Orders
FOR EACH ROW
Begin
          (
          DBMS_OUTPUT.put_line('Group C has ordered: ' || :New.qty || ' type' || :New.type || ' mipads');
          procInventory (:new.qty, :new.type, :new.oid);
          )
```

```
)
END;
```

2. This procedure means that if the quantity of the mipad in our inventory is larger than the required quantity in the new order, then we would call the deliver procedure, which will complete the order and notify teamC. However, if the remaining mipads in the inventory cannot complete the order, it will call the searchMipadTable and pass in the quantity of mipads that we need to produce, the type of mipad and order id as parameters.

```
CREATE OR REPLACE PROCEDURE procInventory (qty IN INT, typeInput IN VARCHAR, orderId
IN INT)
IS
BEGIN
        DBMS_OUTPUT.put_line(‘start of ProcInventory function’);
        declare a int;
        declare b int;
        select i.qty into a from inventory i where i.type=typeInput;
        if (qty<=a) then deliver( qty , typeInput , orderId ); // we have enough mipads to deliver
        elseif (qty>a) then b:=qty-a;  // we want ‘b’ more mipads.
        searchMipadTable(b,typeInput,orderId);
        end if;
        COMMIT;
END;
```

3. If the remaining quantity in the inventory cannot complete the new order, this procedure will be called. It will check in our mipad table to see if there are still some remaining mipads. These mipads should be as the same type as what the order wants, passed the test, and haven’t been sold.
If there is any, it will be added to the quantity of the inventory table. If there is no enough mipad, it will call the searchModulesTable procedure to produce new mipads.

```
CREATE OR REPLACE PROCEDURE searchMipadTable (qtyWeNeed IN INT, typeInput IN
VARCHAR, orderId IN INT)
IS
BEGIN
        declare a, b, c int;
        select count(m.serial_no) into a from mipad m where m.type=typeInput AND
m.test_result=’success’ AND m.sale_status=’not_sold’;
        if (a>=qtyWeNeed) then addToInventory(qtyWeNeed, typeInput, orderId);
        select o.qty into c from orders o where o.oid=orderId;
        //deliver ( c , typeInput , orderId ) ;
        elseif (a<qtyWeNeed) then b:=qtyWeNeed - a;
        searchModulesTable (b, typeInput, orderId);
        end if;
        COMMIT;
END;
```

4. This procedure will be called when we need to produce a new mipad. It will first check if there are enough quantity of the modules which are required to produce the mipad. If so, a assembleModules procedure will be called to assemble the modules. If not, it will call the searchComponentTable to see if there are enough quantity of components which are needed to produce the module. For the searchComponentTable, the extra needed quantity of component, module type, order id, and component id will be passed in as the parameters.

```
CREATE OR REPLACE PROCEDURE searchModulesTable (qty IN INT, typeInput IN VARCHAR,
orderId IN INT)
IS
BEGIN
        declare a, b, c, d, e, x, y INT;
        select m.qty into a from modules m where m.mid=1;
        select m.qty into b from modules m where m.mid=2;
        select m.qty into c from modules m where m.mid=3;
        select m.qty into d from modules m where m.mid=4;
        select m.qty into e from modules m where m.mid=5;
        if (typeInput='A') then
                if ( a>=qty AND c>=qty AND e>=qty ) then
                        assembleModules (qty, A, orderId);
                else y:=qty-a;
                        if (y>0) searchComponentsTable (y, A, orderId, 1);
                        end if;
                        y:=qty-c;
                        if (y>0) searchComponentsTable (y, A, orderId, 3);
                        end if;
                        y:=qty-e;
                        if (y>0) searchComponentsTable (y, A, orderId, 5);
                        end if;
                end if;
        elseif (typeInput='B') then
                if ( a>=qty AND d>=qty AND e>=qty ) then
                        assembleModules (qty, B, orderId);
                else y:=qty-a;
                        if (y>0) searchComponentsTable (y, B, orderId, 1);
                        end if;
                        y:=qty-d;
                        if (y>0) searchComponentsTable (y, B, orderId, 4);
                        end if;
                        y:=qty-e;
                        if (y>0) searchComponentsTable (y, B, orderId, 5);
                        end if;
                end if;
        elseif (typeInput='C') then
                if ( b>=qty AND c>=qty AND e>=qty ) then
                        assembleModules (qty, C, orderId);
                else y:=qty-b;
                        if (y>0) searchComponentsTable (y, C, orderId, 2);
                        end if;
                        y:=qty-c;
                        if (y>0) searchComponentsTable (y, C, orderId, 3);
                        end if;
                        y:=qty-e;
                        if (y>0) searchComponentsTable (y, C, orderId, 5);
                        end if;
                end if;
        elseif (typeInput='D') then
                if ( b>=qty AND d>=qty AND e>=qty ) then
                        assembleModules (qty, D, orderId);
                else y:=qty-b;
                        if (y>0) searchComponentsTable (y, D, orderId, 2);
                        end if;
                        y:=qty-d;
                        if (y>0) searchComponentsTable (y, D, orderId, 4);
                        end if;
                        y:=qty-e;
                        if (y>0) searchComponentsTable (y, D, orderId, 5);
                        end if;
                end if;
        end if;
        COMMIT;
END;
```

5. This trigger will be called when the remaining modules cannot assembly to enough number of mipad to complete the order. If there are enough quantity of components which are required to build the lacking module, the assembleComponents procedures will be called. Otherwise the orderFromGroupA will be called to ask for more components from group a.

CREATE OR REPLACE PROCEDURE **searchComponentsTable** (qtyy IN INT, typeInput IN VARCHAR, orderId IN INT, moduleId IN INT)

```
IS
BEGIN
        declare a, b, c, d, e, f, g, h, i, j INT;
        select c.qty into a from components c where c.cid=1;
        select c.qty into b from components c where c.cid=2;
        select c.qty into c from components c where c.cid=3;
        select c.qty into d from components c where c.cid=4;
        select c.qty into e from components c where c.cid=5;
        select c.qty into f from components c where c.cid=6;
        select c.qty into g from components c where c.cid=7;
        select c.qty into h from components c where c.cid=8;
        select c.qty into i from components c where c.cid=9;
        select c.qty into j from components c where c.cid=10;
        declare x, y, z, xyz INT;
        select min(o.qty) into xyz from orders o where o.to_be_filled>0
        if (moduleId=1) then
                if(a>=qtyy AND b>=qtyy)then assembleComponents (qtyy, orderId, 1);
                else    x:=qtyy-a;
                        if (x>0) orderFromGroupA ( xyz, 1, 1);
                        end if;
                        x:=qtyy-b;
                        if (x>0) orderFromGroupA (xyz,2,2);
                        end if;
                end if;
        elseif (moduleId=2) then
                if(c>=qtyy AND d>=qtyy)then assembleComponents (qtyy, orderId, 2);
                else x:=qtyy-c;
                if (x>0) orderFromGroupA (xyz,3,1);
                end if;
                x:=qtyy-d;
                if (x>0) orderFromGroupA (xyz,4,2);
                end if;
        elseif (moduleId=3) then
                if(e>=qtyy AND f>=qtyy)then assembleComponents (qtyy, orderId, 3);
                else x:=qtyy-e;
                if (x>0) orderFromGroupA (xyz,5,1);
                end if;x:=qtyy-f;
                if (x>0) orderFromGroupA (xyz,6,2);
                end if;
        elseif (moduleId=4) then
                if(g>=qtyy AND h>=qtyy)then assembleComponents (qtyy, orderId, 4);
                else x:=qtyy-g;
                if (x>0) orderFromGroupA (xyz,7,1);
                end if;
                x:=qtyy-h;
                if (x>0) orderFromGroupA (xyz,8,2);
                end if;
        elseif (moduleId=5) then
```

```
                    if(i>=qtyy AND j>=qtyy)then assembleComponents (qtyy, orderId, 5);
                    else x:=qtyy-i;
                    if (x>0) orderFromGroupA (xyz,9,1);
                    end if;
                    x:=qtyy-j;
                    if (x>0) orderFromGroupA (xyz,10,2);
                    end if;
          end if;
END;
```

6. When we need more components from group A, we will insert the needed quantity into their corresponding supplier’s tables.

```
CREATE OR REPLACE PROCEDURE orderFromGroupA ( qtyy IN INT, compId IN INT, suppId IN
INT)
IS
BEGIN
          if (compId=1 OR compId=3 OR compId=5 OR compId=7 OR compId=9) then INSERT INTO
EX_ORDERS(Part_id, Buyer_id, Quantity) VALUES(compId,’groupB’, qtyy);
          else INSERT INTO FN_ORDERS(Part_id, Buyer_id, Quantity) VALUES(compId,’groupB’,
qtyy);
END;
```

7. If we have got enough quantity of mipads to satisfy the requirment of the order, this procedure will be called. It will first print out the notice that we have enought mipads and we are delivering them to group C. Then a setSaleStatus will be called, and change the sale_status to “sold”.

```
CREATE OR REPLACE PROCEDURE deliver (qtyy IN INT, typeInput IN VARCHAR, orderId IN INT)
IS
BEGIN
          declare x INT;
          select i.qty into x from inventory i where i.type=typeInput;
          if (x>=qtyy) then DBMS_OUTPUT.put_line(‘we have enough mipads in our inventory’);
          update inventory set qty:=qty-qtyy where type=typeInput;
          update orders set to_be_filled:=0 where type=typeInput;
          DBMS_OUTPUT.put_line(‘delivering ‘ || qtyy ||’ type ‘ || typeInput || ‘ mipads to group C’);
          DBMS_OUTPUT.put_line(‘setting status of delivered mipads to sold’);
          setSaleStatus (qtyy , typeInput);
          COMMIT;
END;
```

8. This procedure will be called when the mipads have been sold. It will change the sale_status from “not sold” to “sold”.

```
CREATE OR REPLACE PROCEDURE setSaleStatus (qty IN INT , typeInput IN VARCHAR)
IS
BEGIN
        declare i, a int;
        FOR i IN 1..qty LOOP
                select min(m.serial_no) into a from mipad m where m.type=typeInput AND
m.sale_status=’not sold’ and m.test_result=’successful’;
                UPDATE Mipad set sale_status:=’sold’ where serial_no=a;
                DBMS_OUTPUT.put_line(‘mipad number ’ || a || ‘ of type ‘ || typeInput || ‘ was sold’);
        END LOOP;
        COMMIT;
END;
```

9. This procedure is used to assemble the components into modules.

```
CREATE OR REPLACE PROCEDURE assembleComponents (qtyy IN INT, orderId IN INT, moduleId
IN INT)
IS
BEGIN
        declare x, y, z INT;
        if (moduleId=1) then
                Update components set qty:=qty-qtyy where cid=1 OR cid=2;
                DBMS_OUTPUT.put_line(‘number of components 1 and 2 decreased by ’|| qtyy);
                Update modules set qty:=qty+qtyy where mid=1;
                DBMS_OUTPUT.put_line(qtyy||’ module type 1 was made...total number: ’|| z);
        elseif (moduleId=2) then
                Update components set qty:=qty-qtyy where cid=3 OR cid=4;
                DBMS_OUTPUT.put_line(‘number of components 3 and 4 decreased by ’|| qtyy);
                Update modules set qty:=qty+qtyy where mid=2;
                DBMS_OUTPUT.put_line(qtyy||’ module type 2 was made...total number: ’|| z);
        elseif (moduleId=3) then
                Update components set qty:=qty-qtyy where cid=5 OR cid=6;
                DBMS_OUTPUT.put_line(‘number of components 5 and 6 decreased by ’|| qtyy);
                Update modules set qty:=qty+qtyy where mid=3;
                DBMS_OUTPUT.put_line(qtyy||’ module type 3 was made...total number: ’|| z);
        elseif (moduleId=4) then
                Update components set qty:=qty-qtyy where cid=7 OR cid=8;
                DBMS_OUTPUT.put_line(‘number of components 7 and 8 decreased by ’|| qtyy);
                Update modules set qty:=qty+qtyy where mid=4;
                DBMS_OUTPUT.put_line(qtyy||’ module type 4 was made...total number: ’|| z);
        elseif (moduleId=5) then
                Update components set qty:=qty-qtyy where cid=9 OR cid=10;
```

```
                    DBMS_OUTPUT.put_line(‘number of components 9 and 10 decreased by ’|| qtyy);
                    Update modules set qty:=qty+qtyy where mid=5;
                    DBMS_OUTPUT.put_line(qtyy||’ module type 5 was made...total number: ’|| z);
          end if;
END;
```

10. this procedure is used to assemble modules.

```
CREATE OR REPLACE PROCEDURE assembleModules (qtyy IN INT, typeInput IN VARCHAR,
orderId IN INT)
IS
BEGIN
          if (typeInput=’A’) then update modules set qty:=qty-qtyy where mid=1 OR mid=3 OR mid=5;
          if (typeInput=’B’) then update modules set qty:=qty-qtyy where mid=1 OR mid=4 OR mid=5;
          if (typeInput=’C’) then update modules set qty:=qty-qtyy where mid=2 OR mid=3 OR mid=5;
          if (typeInput=’D’) then update modules set qty:=qty-qtyy where mid=2 OR mid=4 OR mid=5;
          DBMS_OUTPUT.put_line(qty || ‘ new mipads of type ‘ || typeInput || ‘ has been made’ );
          makeNewMipads (qtyy, typeInput, orderId);
          COMMIT;
END;
```

11. After a new mipad is made, this procedure will be called. it will generate a new serial_no to the mipad table, and then insert its type and set the test and sale status to “not tested” and “not available”. Then it will called the testMipad procedure to finish the testing process, and call the addtoInventory procedure to increase the inventory quantity, if needed.

```
CREATE OR REPLACE PROCEDURE makeNewMipads (qtyy IN INT, typeInput IN VARCHAR,
orderId IN INT)
IS
BEGIN
          declare a, b, x INT;
          select min(m.serial_no) into a from mipad m;
          DBMS_OUTPUT.put_line(‘last mipad has serial-no: ‘ ||a);
          for b IN 1..qtyy LOOP
                    x:=a+b;
                    INSERT INTO Mipad values (x, typeInput, ‘not tested’, not available’);
                    DBMS_OUTPUT.put_line(‘A mipad with serial number of ’ ||x|| ‘ has been made’);
                    testMipad (x);
          END LOOP;
          DBMS_OUTPUT.put_line(‘All the tests were done successfully’);
          COMMIT;
          addToInventory(qtyy, typeInput, orderId);
END;
```

12. This procedure is to do the testing process. When it is called, it will first print our the serial number of the mipad that it will test, and sleep for 5 seconds. Then it will update the test-status to “successful” and set the sale_status to “for sale”.

```
CREATE OR REPLACE PROCEDURE testMipad (mipadNum IN INT)
IS
BEGIN
        DBMS_OUTPUT.put_line(‘testing mipad with serial number ‘ || mipadNum);
        dbms_job.submit ( job => l_job, what => 'myproc(:new.id);',, next_date => sysdate+3/24/60/60 );
        DBMS_OUTPUT.put_line(‘test took 3 seconds’);
        Update Mipad set test_result:=’successful’ where serial_no=mipadNum;
        Update Mipad set sale_status:=’for sale’ where serial_no=mipadNum;
END;
```

13. This trigger will be called once group A sent us the components that we need.

```
CREATE OR REPLACE TRIGGER deliveredComponents
After update on Components
FOR EACH ROW
declare x, y int;
BEGIN
        select c.qty into x from components c where c.id=:new.id;
        if (:new.qty<x) then DBMS_OUTPUT.put_line(‘no need to do anything!’);
        elseif (:new.qty>x)then y:=:new.qty-x;
        DBMS_OUTPUT.put_line(y||‘ type ’|| :new.id || ‘ components delivered to us’);
        checkComponents (:new.id);
END;
```

14. After group A sends us the components that we need, this procedure will be called. It will first check whether the rest of the components, combined with the new delivered components, are sufficient to build the new modules we want. If so, the new modules will be made.

```
CREATE OR REPLACE PROCEDURE checkComponents (compId IN INT)
IS
BEGIN
        declare x, y int;
        declare z varchar(5);
        select min(o.id) into y from orders o where o.to_be_filled>0
        select o.qty into x from orders o where o.id=y;
        select o.type into z from orders o where o.id=y;
        declare a, b, c, d, e, f, g, h, i, j INT;
        select c.qty into a from components c where c.cid=1;
```

```
    select c.qty into b from components c where c.cid=2;
    select c.qty into c from components c where c.cid=3;
    select c.qty into d from components c where c.cid=4;
    select c.qty into e from components c where c.cid=5;
    select c.qty into f from components c where c.cid=6;
    select c.qty into g from components c where c.cid=7;
    select c.qty into h from components c where c.cid=8;
    select c.qty into i from components c where c.cid=9;
    select c.qty into j from components c where c.cid=10;
    if (compId=1 or compId=2)
        if (a>=x AND b>=x) then assembleComponents (x, y, 1);
        checkModules (x, z);
        end if;
    elseif (compid=3 or compId=4)
        if (c>=x AND d>=x) then assembleComponents (x, y, 2);
        checkModules (x, z);
        end if;
    elseif (compid=5 or compId=6)
        if (e>=x AND f>=x) then assembleComponents (x, y, 3);
        checkModules (x, z);
        end if;
    elseif (compid=7 or compId=8)
        if (g>=x AND h>=x) then assembleComponents (x, y, 4);
        checkModules (x, z);
        end if;
    elseif (compid=9 or compId=10)
        if (i>=x AND j>=x) then assembleComponents (x, y, 5);
        checkModules (x, z);
        end if;
    end if;
END;
```

15. When a certain type of mipad is to be made, it will check whether there are sufficient quantity of modules. If so, it will start to build the mipad with assembleModules procedure.

```
CREATE OR REPLACE PROCEDURE checkModules ( qtyy IN INT, mipadType IN VARCHAR )
IS
BEGIN
    declare a, b, c, d, e INT;
    select m.qty into a from modules m where m.mid=1;
    select m.qty into b from modules m where m.mid=2;
    select m.qty into c from modules m where m.mid=3;
    select m.qty into d from modules m where m.mid=4;
    select m.qty into e from modules m where m.mid=5;
```

```
        declare x, y, xx, yy int;
        declare z varchar(5);
        select min(o.id) into y from orders o where o.to_be_filled>0
        select o.qty into x from orders o where o.id=y;
        select o.type into z from orders o where o.id=y;
        select i.qty into xx from inventory i where i.type=z;
        z:=x-z;
        if (mipadType='A') then
                if (a>=z AND c>=z AND e>=z) then assembleModules (z, A, y);
                end if;
        elseif (mipadType='B') then
                if (a>=z AND d>=z AND e>=z) then assembleModules (z, B, y);
                end if;
        elseif (mipadType='C') then
                if (b>=z AND c>=z AND e>=z) then assembleModules (z, C, y);
                end if;
        elseif (mipadType='D') then
                if (b>=z AND d>=z AND e>=z) then assembleModules (z, D, y);
                end if;
        end if;
END;
```

16. When certain number of mipad has been made, it will be added to the inventory table.

```
CREATE OR REPLACE PROCEDURE addToInventory (qtyy IN INT, typeInput IN VARCHAR,
orderID IN INT)
IS
BEGIN
        declare x INT;
        update inventory set qty:=qty+qtyy where type=typeInput;
        select o.qty into x from orders o where o.order_no=orderId;
        deliver (x, typeInput , orderId);
END;
```

Create View Statements

CEO View

From the CEO's perspective, he/she won't need to worry about the whole manufacturing and assembly process; all he/she cares is that how many components we have bought for producing miPads, as well as how many mipads have been delivered to teamC. The following two view tables are for the CEO to monitor these two processes.

(the following two views are created using the table from teamA)

```
create VIEW CEO_VIEW1 as
select part_id,  sum (quantity)
 from EX_ORDERS E
group by part_id;

create VIEW CEO_VIEW2 as
select part_id sum (quantity)
from FN_ORDERS
group by part_id;
```

CEO_VIEW1 and CEO_VIEW2provides the information about the total quantities of each component we purchased. As we don't have a table in our team which records the total numbers of components we bought, we need to refer to teamA's order table to realize that.

```
create view CEO_VIEW3 as
select type,  sum (qty)
from ORDERS
where to_be_filled>0
group by type;
```

CEO_VIEW3 view offers the CEO the information about how many mipads of each type have been delivered to teamC. We did not include the information about the order which hadn't been delivered.

Supply Chain Manager View

As for the supply chain manager, he/she not only needs to care about the information that the CEO cares, but also needs to monitor the entire manufacturing and assembly processes, including the semi-finished products (i.e the modules) and the current quantities of components and modules in the manufacturing and assembly lines.

(this view is created using the table from teamA)

```
create view SCM_VIEW1 as
create VIEW CEO_VIEW1 as
select part_id, sum (quantity)
from EX_ORDERS
group by part_id;

create VIEW CEO_VIEW2 as
select part_id sum (quantity)
from FN_ORDERS
group by part_id;
```

```
create view SCM_VIEW3 as
select cid, qty
from COMPONENTS;

create view SCM_VIEW4 as
select mid, mQty
from MODULES;

create view SCM_VIEW5 as
select*
from INVENTORY;

create view SCM_VIEW6 as
select type,  sum (qty)
from ORDERS
where to_be_filled>0
group by type;
```

Demonstration Outline

At the beginning of the demonstration, we BEGIN the database session and then will use SELECT * FROM [each table] to show what is already in our database and demonstrate the presence of the various types of modules and miPads.

After that, our database will come into use when Team C inserts orders into our table. Our triggers and stored procedures will pass the orders back and the parts and miPads forward as needed; this activity will be marked by the triggers printing their activity to the screen.

After all that activity is finished, we will use the SELECT * FROM [each table] to show that miPads were created and tested, inventory changed, and orders were filled. Then we will COMMIT and END the session.

Statement on Teamwork

We worked fairly well as a team. Koushyar had the most programming experience, so he took lead in making sure the triggers worked. Shumeng focused on creation of the database structures, the view statements and the explanations of triggers and procedures. Lei focused on creation of the database structures, data entry and testing. Jennifer lead the writing part of the project. Everyone contributed conceptually to the overall design of the project. The biggest issue we had was coordinating times to meet; it was difficult to find sufficient time among four very busy schedules.