**DHAVAL GOGRI**

**47444609**

**CSE 5330/7330 Fall 2017**

**Project Definition**

**Do your own work. Reference any material used.**

**(Alternative Projects will be considered by Dr. Moore. Please contact him with suggestions.)**

This is an individual project to be completed by each student. In addition, the project is to be completed in 3 phases. At the end of each phase, you are to submit that portion of the project for grading. At that time, you will receive information needed to complete the next phase of the project.

*You are to design and implement a database system to keep track of software builds performed by* ***Our Software Factory*** *(OSF). Your user contact (and person to approach with any questions) is Dr. Moore.*

Any DBMS may be used for the project as long as a SQL command line interface is available for customer acceptance testing. The last phase of the project will require that Dr. Moore be able to test some unseen queries against your database, thus he must have access to your database. If you use the SMU implementation of Oracle or MySQL, this can easily be accomplished using appropriate security settings. If, however, you use another DBMS then you will have to provide instructions for its use or schedule a time for hands-on interaction.

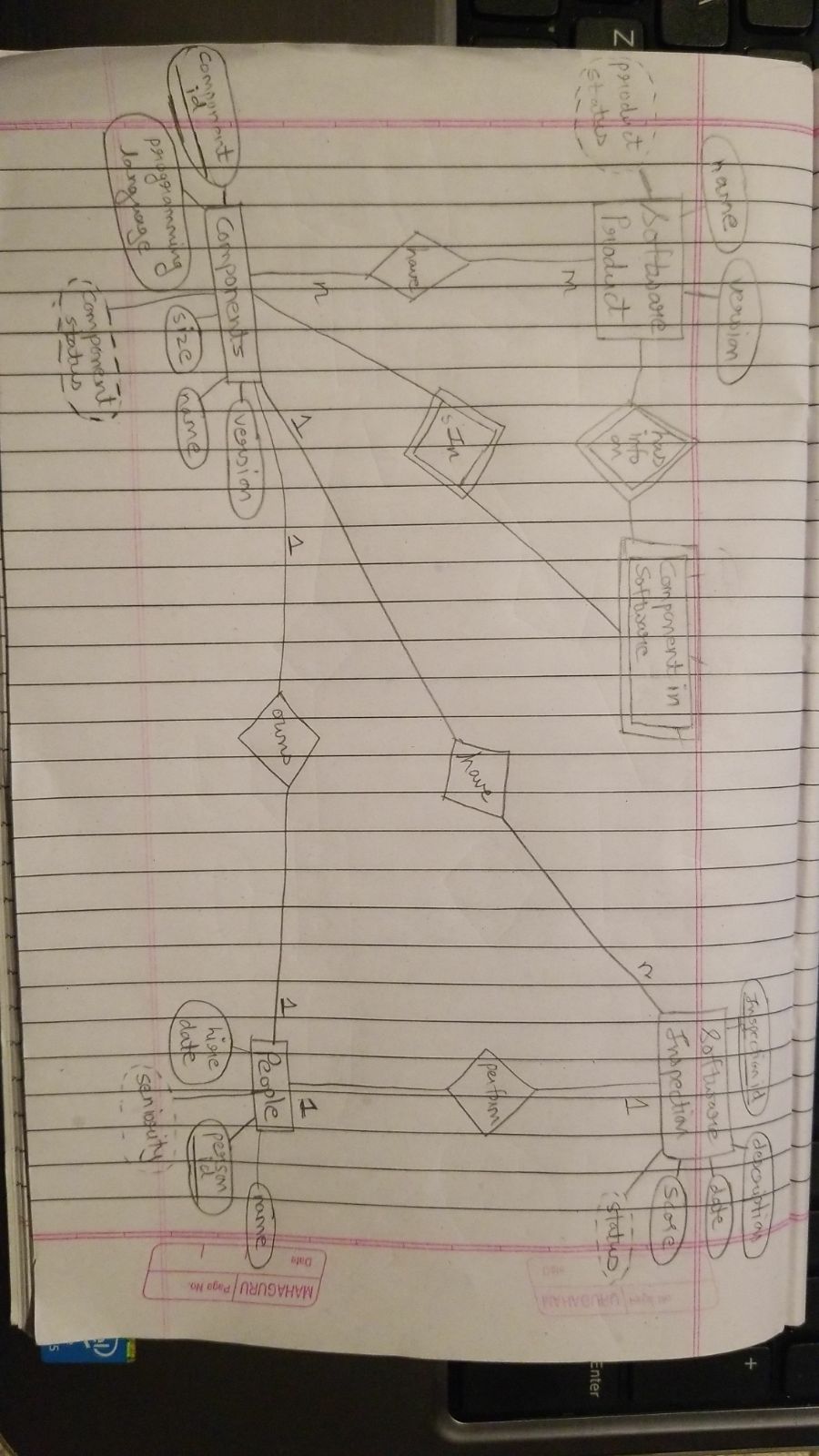
Our Software Factory develops various software products, each one requiring a software build. A software product is identified by its name and version. A software build identifies the components needed for a particular product. Components may be shared among products. The build status for a product is the lowest status of any of its components. Each component has a status of ready, usable, and not-ready. Information required about each component includes size, programming language (C, C++, C#, Java, or PHP), component name and 3 character version. Each component has one person identified as its owner. People have a unique ID (5 digit number), name (60 characters) and seniority. Components get their status as a result of a software inspection (peer review). An inspection event covers one component and results in an inspection score (0 .. 100). If the score is greater than 90, the component is considered “Ready”. If the score is less than 75, it is “not-ready”; otherwise it is “usable”. Inspection data includes what was inspected, the date of inspection, who conducted the inspection, inspection score, and textual description. The textual description can be updated later, but the score can never be changed. Components may be inspected multiple times.

The actual applications to be run against the database have not yet been determined by the user; however you have been asked to start the development of the database to get it ready. Your assignment is divided into three phases. At the end of each phase, you will be provided with any missing information needed to complete the next phase

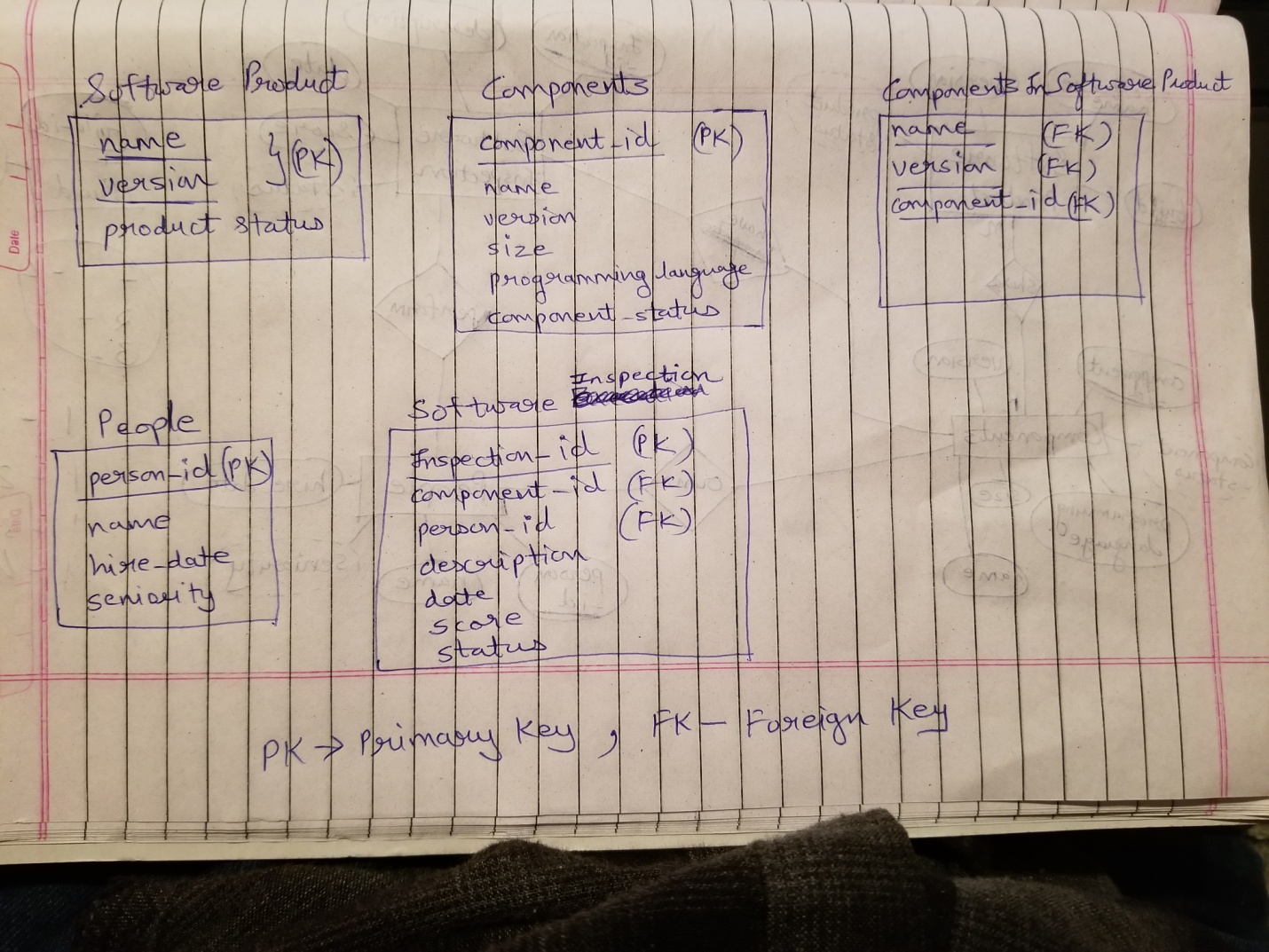
**Project Phases**

1. **ER Diagram and Initial Relational Design (20 pts; Due: 10/26):**

* 1. Construct an ER Diagram with attributes: The ER diagram you create must support all requirements stated above. If you add any restrictions or information not stated above, please explain. Indicate any design requirements that are not included in the ER diagram.



* 1. Produce an initial Relational design: Given your ER diagram, provide an initial description of your relations, keys and foreign keys.



There would be 5 tables

1. Software Product - Contains Software Product information

2. Components - Contains Component Information

3. Components in Software Product - Contains Components information for a software product

4. People - People there in Our Software Factory

5. Software Inspection - Contains the score needed to give component its Working Status

Here, when a Software Inspection is completed and a score is given, a trigger is fired and changes are made to the component accordingly. When the status is changed for the component, all the Software Product associated with the component are also checked for updating of Product Status. The "Components in Software Product" table keeps tracks of all the components used in a particular software. When the "component\_status" changes a trigger would run and check in "Components in Software Product" for that component and associated software products to make any changes if necessary.

* 1. Submit for grading your ER diagram and relational schemas.

**2.a Database Implementation** **(30 pts; Due: 11/16):**

For phase 2 I have implemented the suggestions given by you during phase 1. I have made name and version as the primary key in Components.The table 'Components\_in\_software\_products' will have (name, version and comp\_id) as primary key.

* 1. Using SQL DDL statements, create the relations as designed in phase 1. You must include any needed data constraints and keys (primary and foreign) to ensure design requirements are met.

create table programming\_language

(

language\_name varchar(20) primary key,

language\_status ENUM('current', 'future') not null

)

create table software\_products

(

name varchar(40),

version varchar(10),

software\_status ENUM('Ready', 'not-ready', 'usable') not null default 'not-ready',

primary key(name,version)

)

create table Employees

(

id int primary key,

name varchar(30),

hire\_date timestamp,

mgr\_id int,

seniority varchar(10)

)

create table Components

(

comp\_id int auto\_increment,

component\_name varchar(40),

version varchar(10),

component\_size int,

prog\_language varchar(20),

comp\_owner int,

component\_status ENUM('Ready', 'not-ready', 'usable') not null default 'not-ready',

primary key(component\_name, version),

FOREIGN KEY (prog\_language) REFERENCES programming\_language(language\_name),

FOREIGN KEY (comp\_owner) REFERENCES Employees(id),

key (comp\_id)

)

create table components\_in\_software\_products

(

name varchar(40),

version varchar(10),

comp\_id int,

primary key(name,version, comp\_id),

FOREIGN KEY (name, version) REFERENCES software\_products(name, version),

FOREIGN KEY (comp\_id) REFERENCES Components(comp\_id)

)

create table Inspection

(

inspection\_id int primary key auto\_increment,

component\_name varchar(40),

version varchar(10),

inspection\_date timestamp not null,

by\_who int,

score int not null,

description varchar(4000),

status ENUM('Ready', 'not-ready', 'usable') not null default 'not-ready',

FOREIGN KEY (by\_who) REFERENCES Employees(id),

FOREIGN KEY (component\_name, version) REFERENCES Components(component\_name, version),

key (inspection\_id)

)

* 1. Populate the relations using data provided by the user.

-- Insert Programming languages

insert into programming\_language values('C','current');

insert into programming\_language values('C++','current');

insert into programming\_language values('C#','current');

insert into programming\_language values('Java','current');

insert into programming\_language values('PHP','current');

insert into programming\_language values('Python','Future');

insert into programming\_language values('assembly','Future');

-- Insert Into Employees

insert into employees(id, name, hire\_date, mgr\_id) values(10100, 'Employee-1', STR\_TO\_DATE( '08/11/1984', '%m/%d/%Y'), null);

insert into employees(id, name, hire\_date, mgr\_id) values(10200, 'Employee-2', STR\_TO\_DATE( '08/11/1994', '%m/%d/%Y'),10100);

insert into employees(id, name, hire\_date, mgr\_id) values(10300, 'Employee-3', STR\_TO\_DATE( '08/11/2004', '%m/%d/%Y'),10200);

insert into employees(id, name, hire\_date, mgr\_id) values(10400, 'Employee-4', STR\_TO\_DATE( '01/11/2008', '%m/%d/%Y'),10200);

insert into employees(id, name, hire\_date, mgr\_id) values(10500, 'Employee-5', STR\_TO\_DATE( '01/11/2015', '%m/%d/%Y'),10400);

insert into employees(id, name, hire\_date, mgr\_id) values(10600, 'Employee-6', STR\_TO\_DATE( '01/11/2015', '%m/%d/%Y'),10400);

insert into employees(id, name, hire\_date, mgr\_id) values(10700, 'Employee-7', STR\_TO\_DATE( '01/11/2016', '%m/%d/%Y'),10400);

insert into employees(id, name, hire\_date, mgr\_id) values(10800, 'Employee-8', STR\_TO\_DATE( '01/11/2017', '%m/%d/%Y'),10200);

-- Insert into Components

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(1, 'Keyboard Driver', 'K11', 1200, 'C', 10100);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(2, 'Touch Screen Driver', 'T00', 4000, 'C++', 10100);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(3, 'Dbase Interface', 'D00', 2500, 'C++', 10200);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(4, 'Dbase Interface', 'D01', 2500,'C++', 10300);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(5, 'Chart generator', 'C11', 6500, 'java', 10200);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(6, 'Pen Driver', 'P01', 3575, 'C', 10700);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(7, 'Math unit', 'A01', 5000, 'C', 10200);

insert into Components(comp\_id, component\_name, version, component\_size, prog\_language, comp\_owner) values(8, 'Math unit', 'A02', 3500, 'Java', 10200);

-- Insert into Software Products

insert into software\_products(name, version) values('Excel', '2010');

insert into software\_products(name, version) values('Excel', '2015');

insert into software\_products(name, version) values('Excel', '2018beta');

insert into software\_products(name, version) values('Excel', 'secret');

-- Insert into Components in Software

insert into components\_in\_software\_products values('Excel', '2010', 1);

insert into components\_in\_software\_products values('Excel', '2010', 3);

insert into components\_in\_software\_products values('Excel', '2015', 1);

insert into components\_in\_software\_products values('Excel', '2015', 4);

insert into components\_in\_software\_products values('Excel', '2015', 6);

insert into components\_in\_software\_products values('Excel', '2018beta', 1);

insert into components\_in\_software\_products values('Excel', '2018beta', 2);

insert into components\_in\_software\_products values('Excel', '2018beta', 5);

insert into components\_in\_software\_products values('Excel', 'secret', 1);

insert into components\_in\_software\_products values('Excel', 'secret', 2);

insert into components\_in\_software\_products values('Excel', 'secret', 5);

insert into components\_in\_software\_products values('Excel', 'secret', 8);

-- insert into Inspection

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(1, 'Keyboard Driver', 'K11', STR\_TO\_DATE('02/14/2010', '%m/%d/%Y'), 10100, 100, 'legacy code which is already approved');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(2, 'Touch Screen Driver', 'T00', STR\_TO\_DATE('06/01/2017', '%m/%d/%Y'), 10200, 95, 'initial release ready for usage');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(3, 'Dbase Interface', 'D00', STR\_TO\_DATE('02/22/2010', '%m/%d/%Y'), 10100, 55, 'too many hard coded parameters, the software must be more maintainable and configurable because we want to use this in other products.');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(4, 'Dbase Interface', 'D00', STR\_TO\_DATE('02/24/2010', '%m/%d/%Y'), 10100, 78, 'improved, but only handles DB2 format');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(5, 'Dbase Interface', 'D00', STR\_TO\_DATE('02/26/2010', '%m/%d/%Y'), 10100, 95, 'Okay, handles DB3 format.');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(6, 'Dbase Interface', 'D00', STR\_TO\_DATE('02/28/2010', '%m/%d/%Y'), 10100, 100, 'satisfied');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(7, 'Dbase Interface', 'D01', STR\_TO\_DATE('05/01/2011', '%m/%d/%Y'), 10200, 100, 'Okay ready for use');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(8, 'Pen Driver', 'P01', STR\_TO\_DATE('07/15/2017', '%m/%d/%Y'), 10300, 80, 'Okay ready for beta testing');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(9, 'Math unit', 'A01', STR\_TO\_DATE('06/10/2014', '%m/%d/%Y'), 10100, 90, 'almost ready');

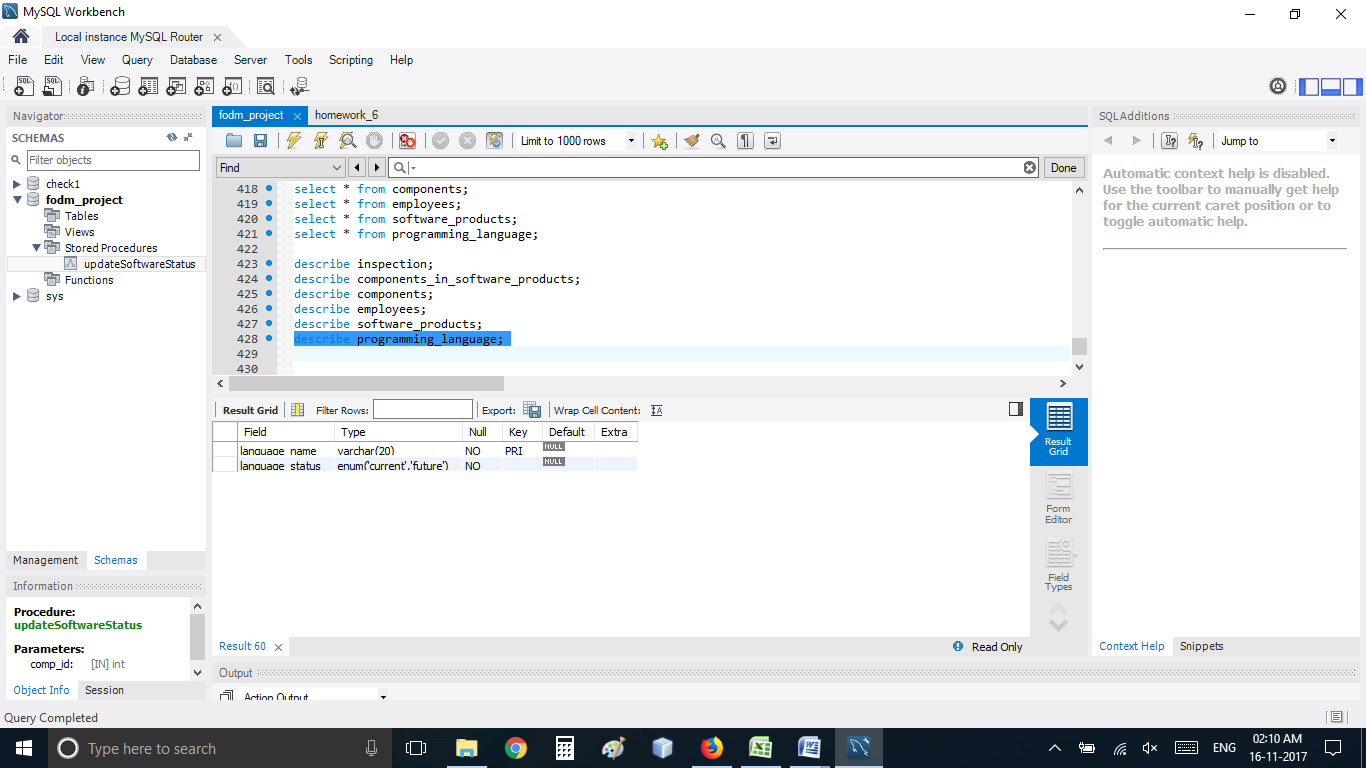
insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(10, 'Math unit', 'A02', STR\_TO\_DATE('06/15/2014', '%m/%d/%Y'), 10100, 70, 'Accuracy problems!');

insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(11, 'Math unit', 'A02', STR\_TO\_DATE('06/30/2014', '%m/%d/%Y'), 10100, 100, 'Okay problems fixed');

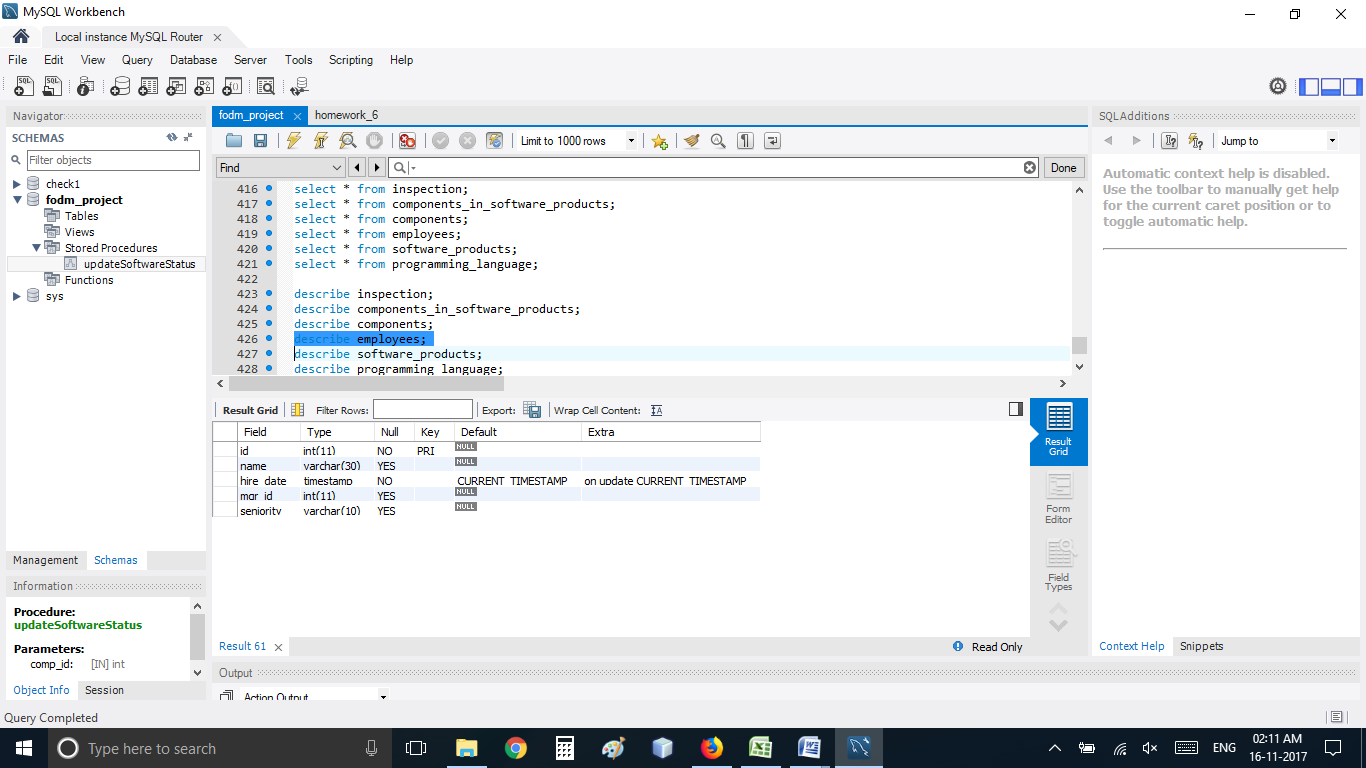
insert into inspection(inspection\_id, component\_name, version, inspection\_date, by\_who, score, description) values(12, 'Math unit', 'A02', STR\_TO\_DATE('11/02/2016', '%m/%d/%Y'), 10700, 100, 're-review for new employee to gain experience in the process.');

* 1. Submit for grading proof of creation of the relations and their population. This could be output of ‘Describe’ and SELECT \* statements. Be sure to indicate your selection of DBMS and location of implementation.

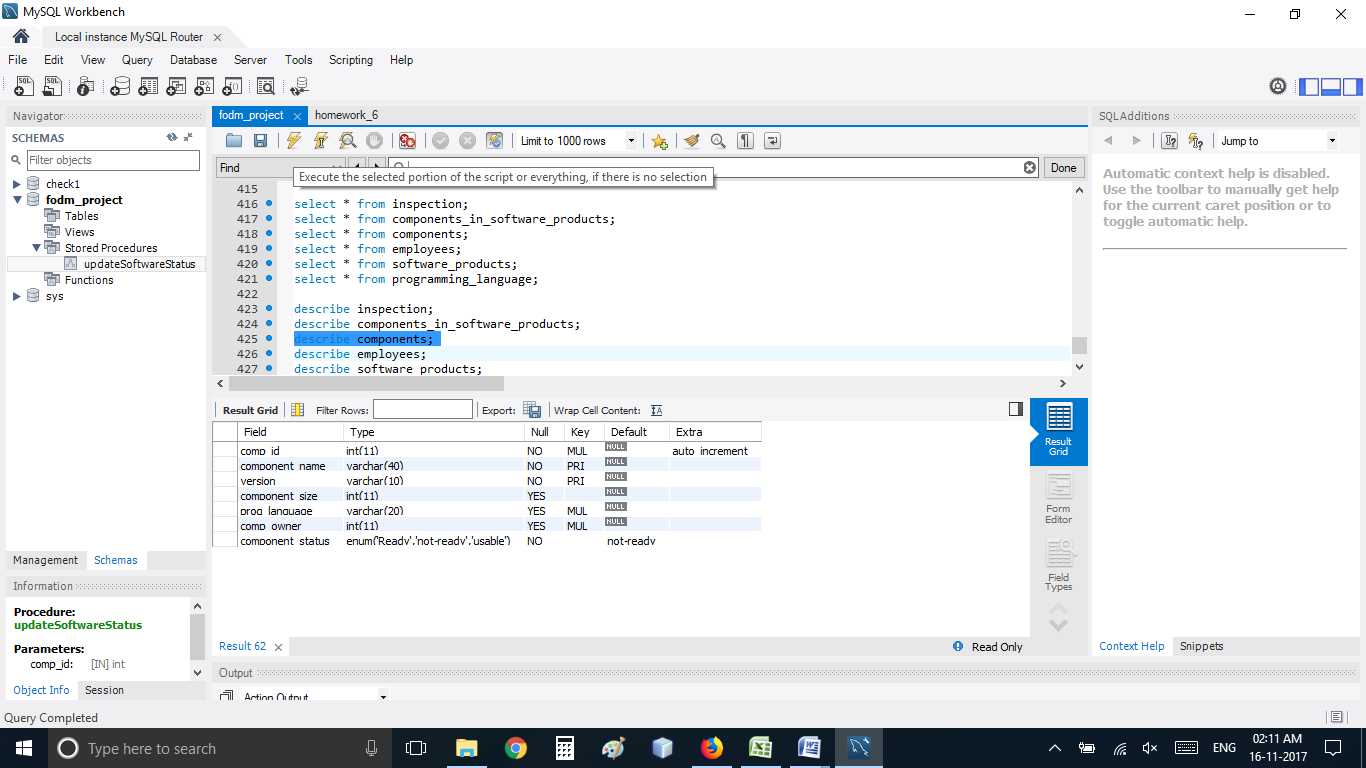
describe programming\_language;



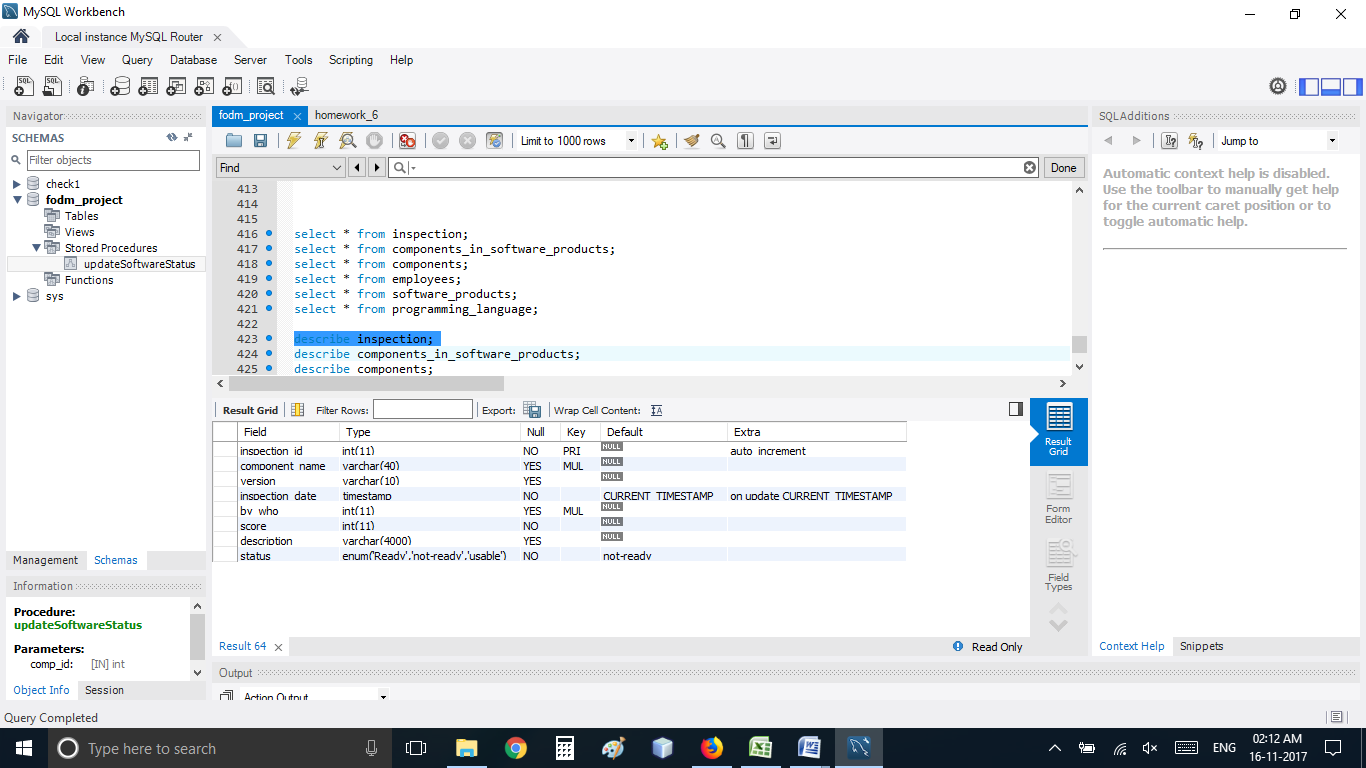
describe employees;



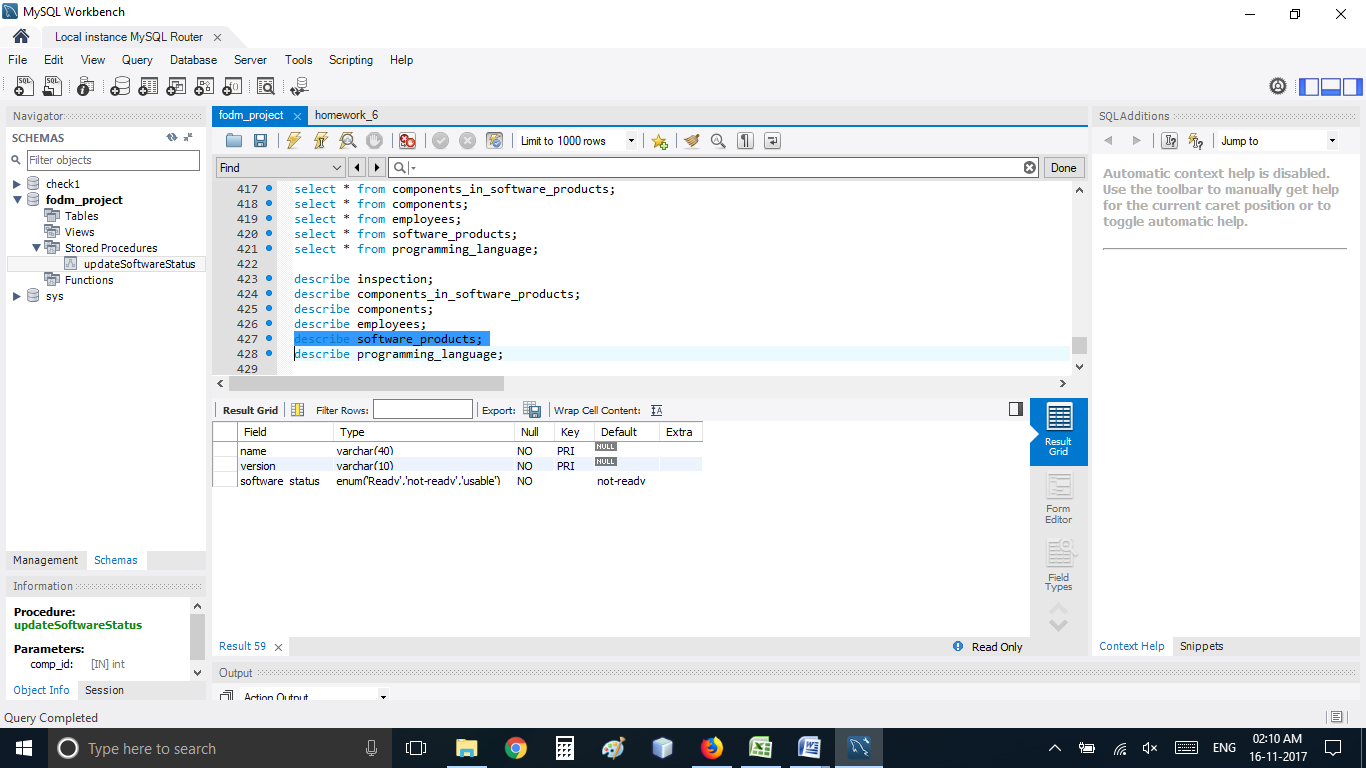
describe components;



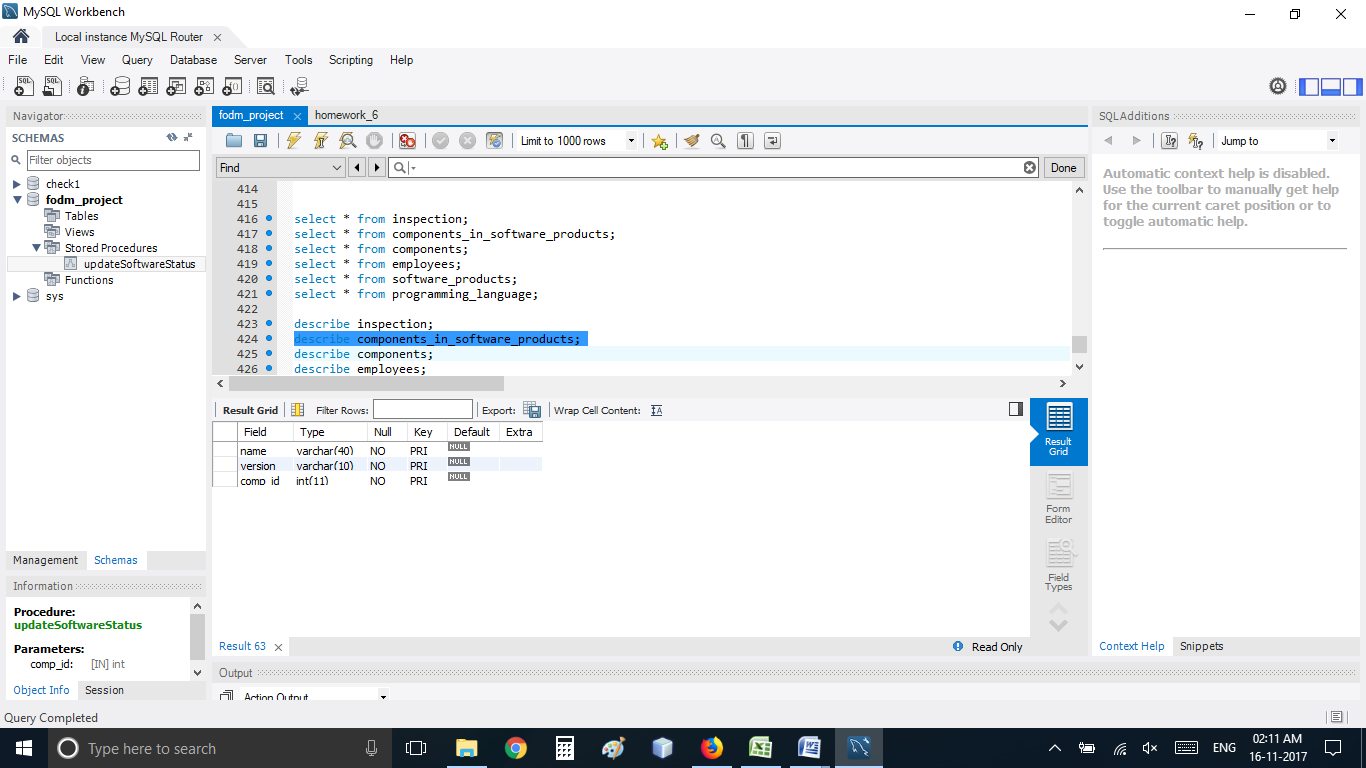
describe inspection;



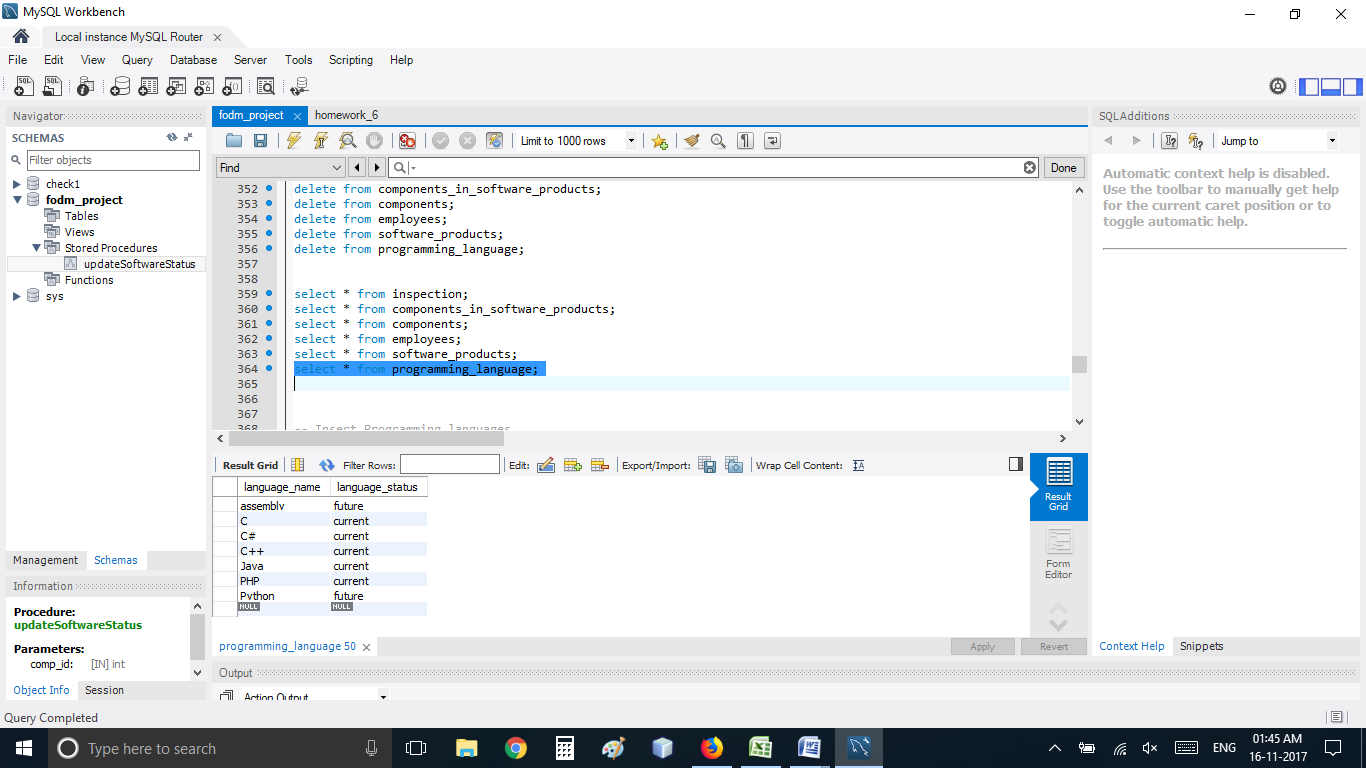
describe software\_products;



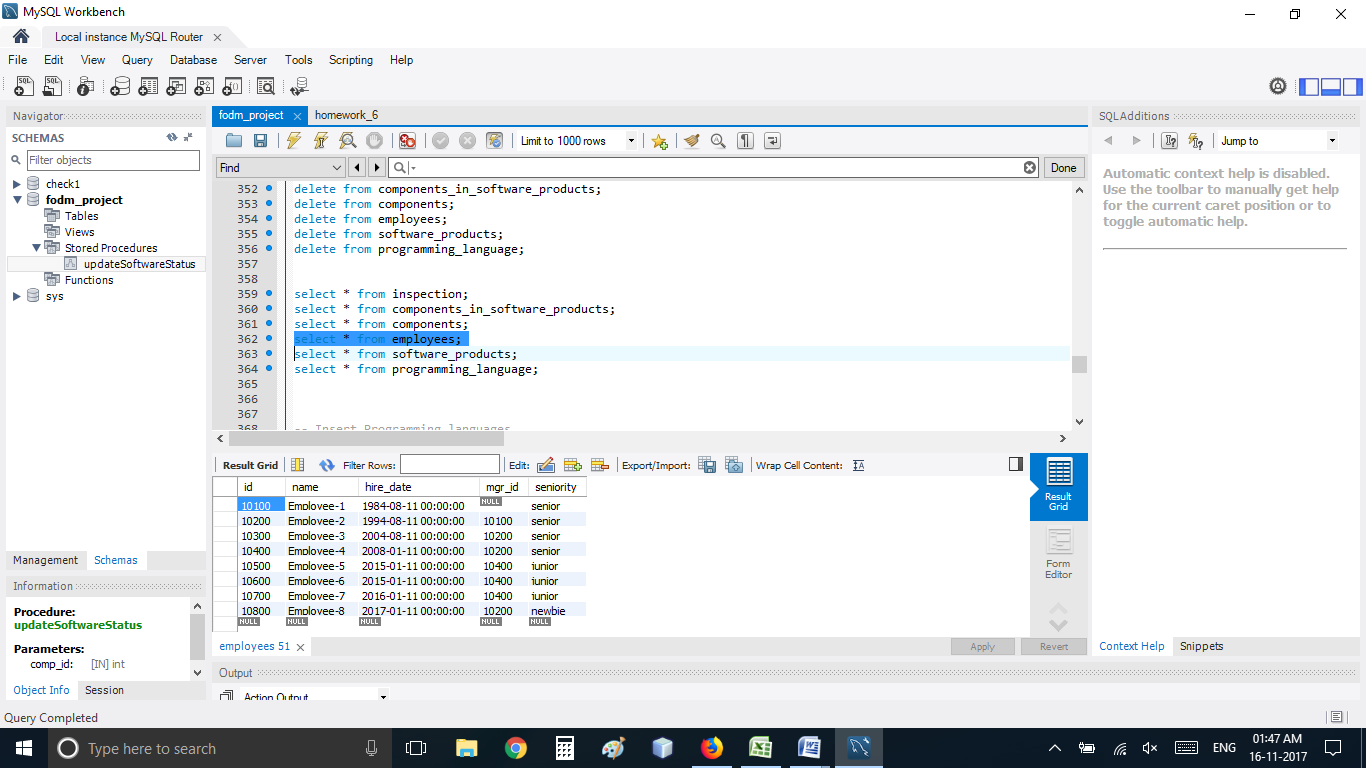
describe components\_in\_software\_products;



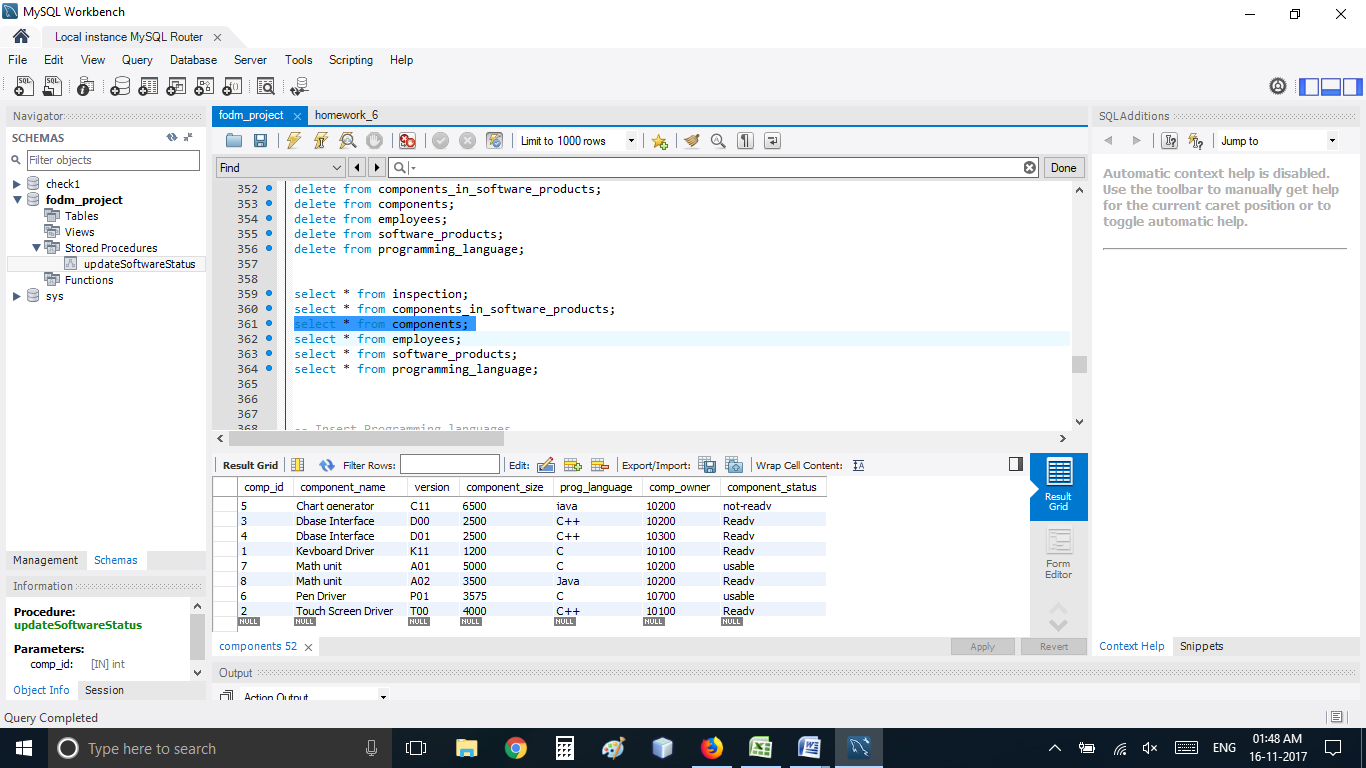
select \* from programming\_language;



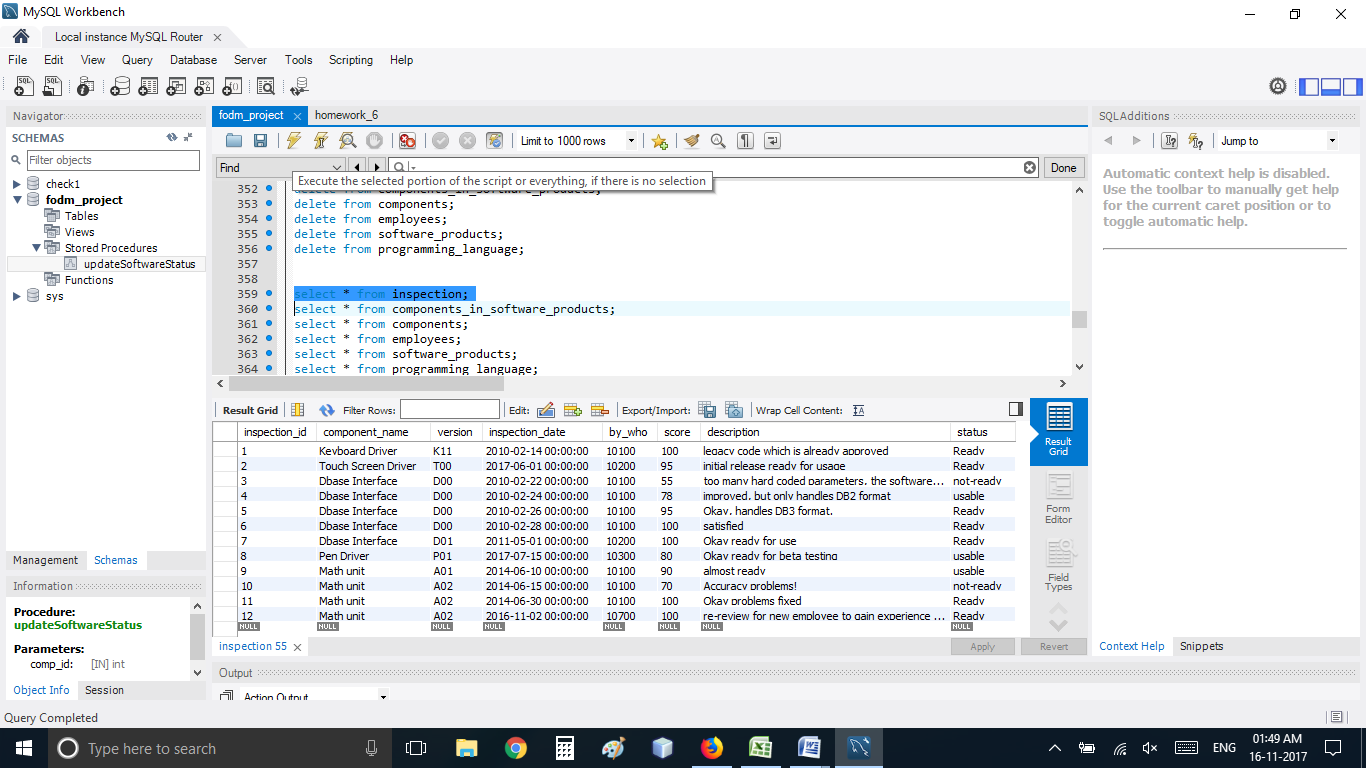
select \* from employees;



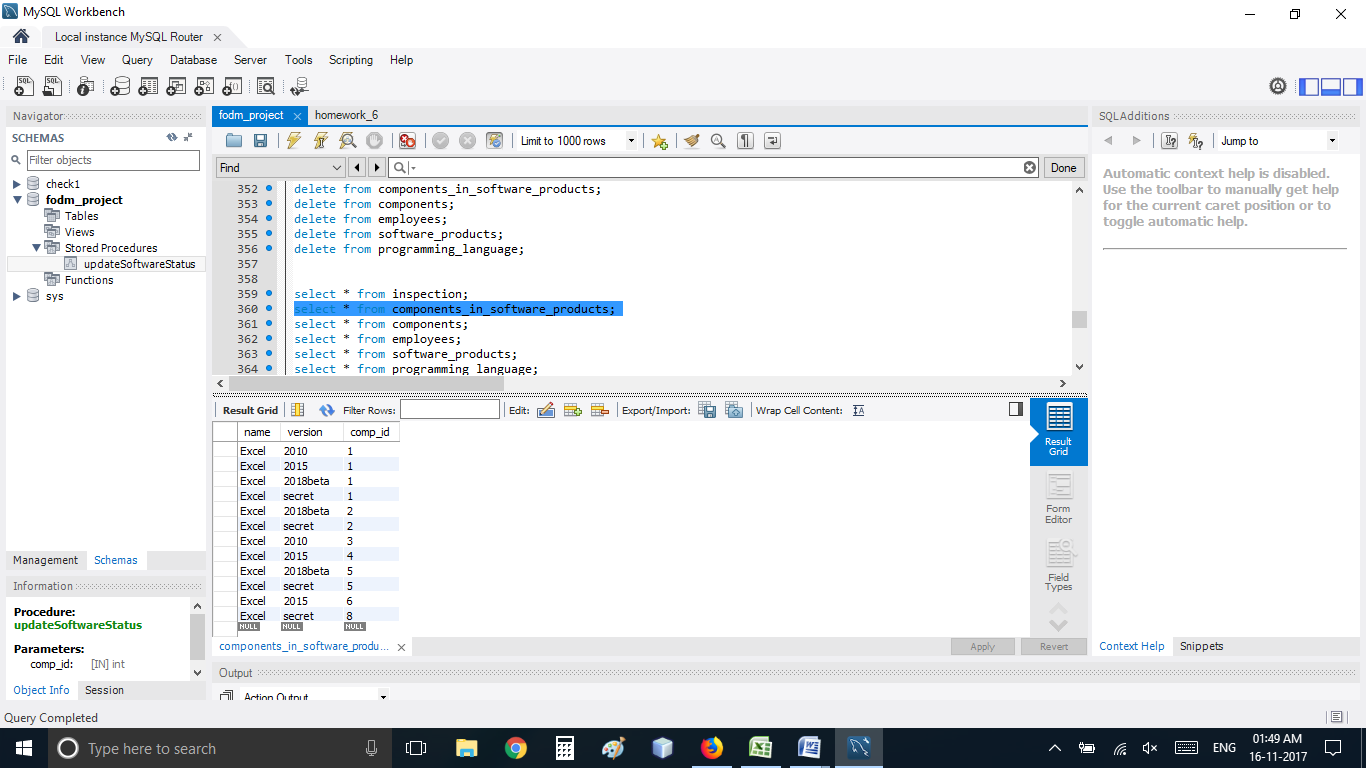
select \* from components;



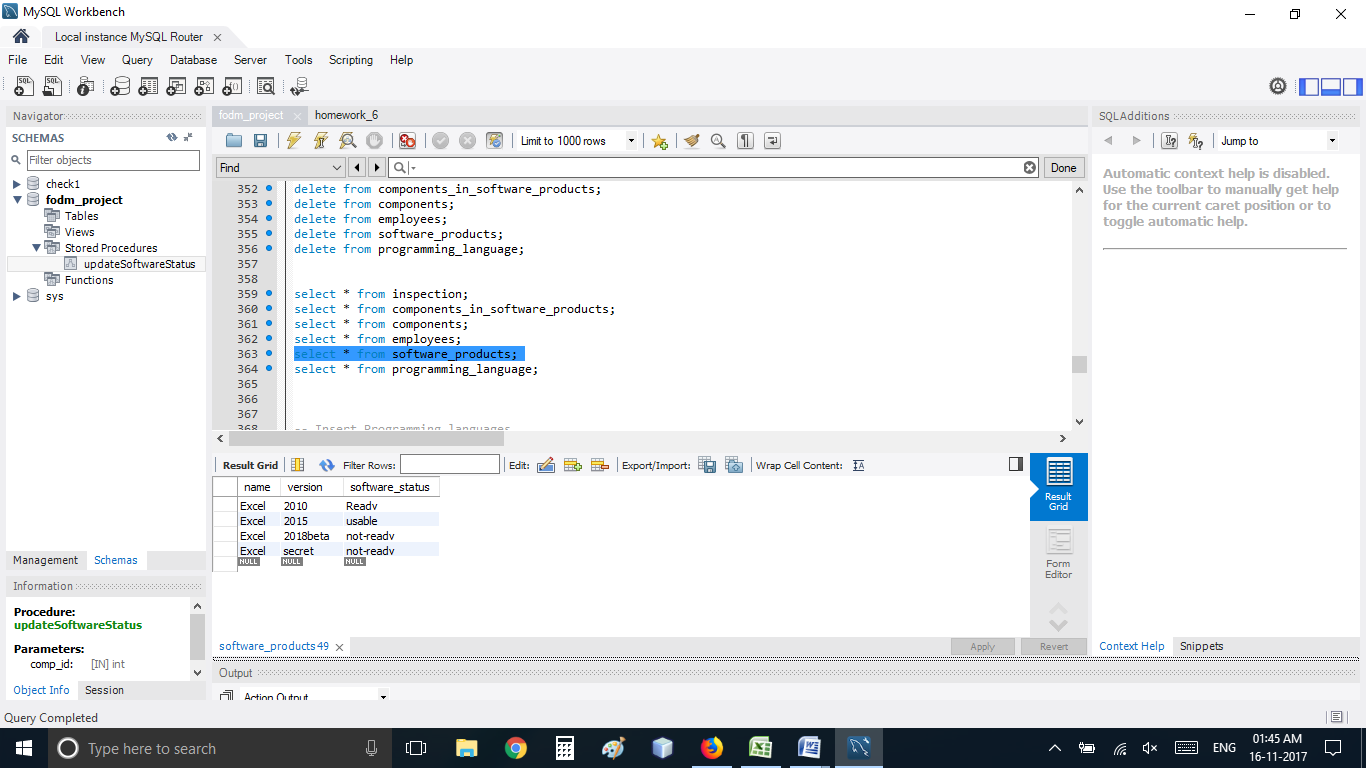
select \* from inspection;



select \* from components\_in\_software\_products;



select \* from software\_products;



**2.b Triggers** **(10 pts; Due: 11/16 ):**

* 1. Select one nontrivial trigger that is needed to ensure data requirements are met.

-- Trigger on Mgr Id

-- INSERT

This trigger checks for mgr\_id. For CEO i.e. for employee id 10100. Either he should be the manager himself or the field left null. For other employees the manager should be an existing employee of the company.

delimiter $$

create trigger empolyee\_manager\_validate\_insert

before insert on Employees

for each row

begin

DECLARE count\_occ INT default 0;

if (new.id = 10100) then -- 10100 is the CEO as per the requirement

if (new.mgr\_id = 10100 or new.mgr\_id = null) then

set new.mgr\_id = new.mgr\_id;

else

signal sqlstate '45000'

set message\_text = 'The CEO can not have a subordinate as a manager or enter his own id or null as his manager.';

end if;

else

if (new.id = new.mgr\_id) then

signal sqlstate '45000'

set message\_text = 'An employee cannot be his own manager';

end if;

(select count(\*) into @count\_occ from Employees group by Employees.id having Employees.id = new.id);

if (count\_occ = 0) then

signal sqlstate '45000'

set message\_text = 'Manager should be an existing employee';

end if;

end if;

end;

$$

delimiter ;

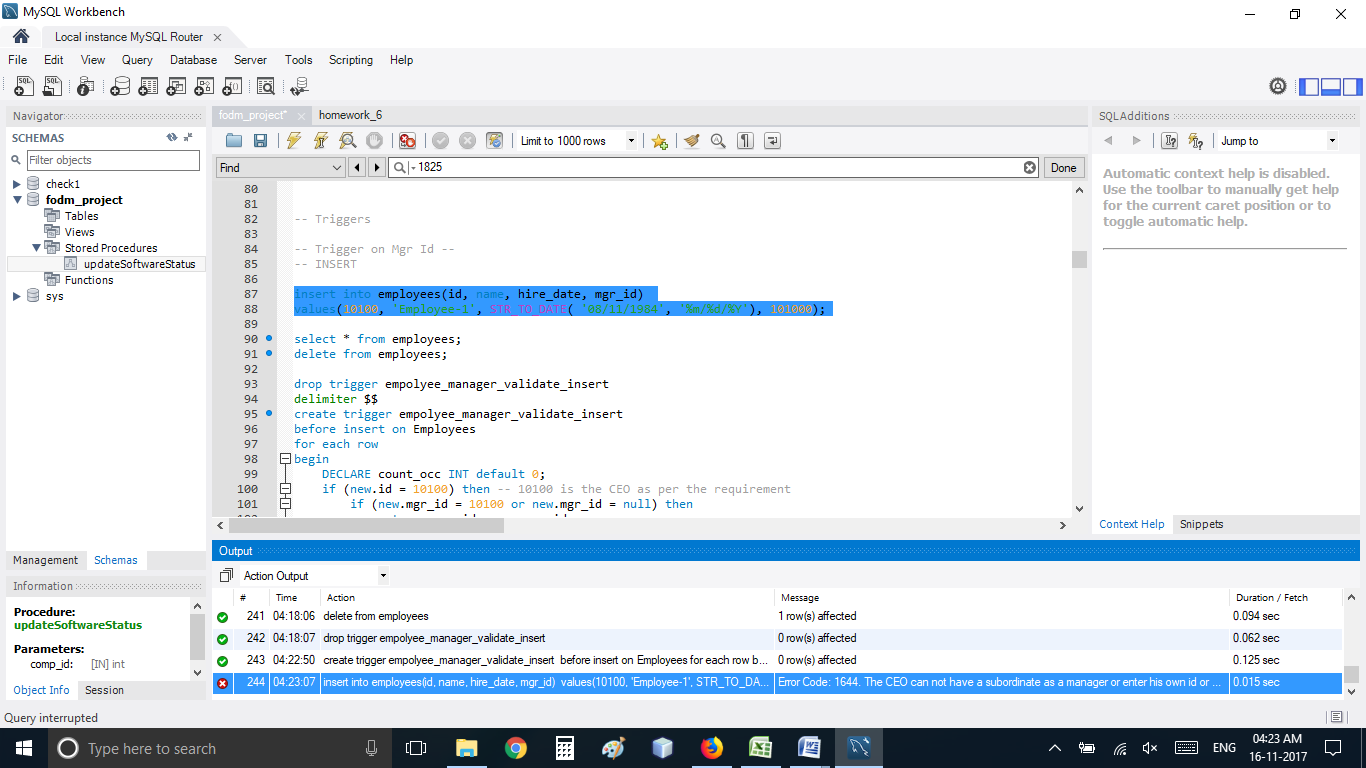
* 1. Show the implementation of that trigger along with the results of your testing to confirm the trigger works as expected in an efficient manner. Continue to implement all required triggers.

This trigger checks for mgr\_id. For CEO i.e. for employee id 10100. Either he should be the manager himself or the field left null.

insert into employees(id, name, hire\_date, mgr\_id)

values(10100, 'Employee-1', STR\_TO\_DATE( '08/11/1984', '%m/%d/%Y'), 101000);

I enter the wrong manager id and trigger fires and doesn't allow the row to be added in the table.



I have also implemented several other triggers, event for updating seniority and procedures to support the triggers.

Below are the codes for those triggers, procedures and events.

-- Tiggers and Stored Procedure for Update in Components

-- Procedure to change the status of the component

Delimiter $$

CREATE PROCEDURE updateComponentsStatus (IN component\_name varchar(20), IN version varchar(10), IN status varchar(10))

BEGIN

Declare id INT;

update Components set Components.component\_status = status where Components.component\_name = component\_name and Components.version = version;

set id = (select comp\_id from Components where Components.component\_name = component\_name and Components.version = version);

CALL updateSoftwareProductStatus(id);

END $$

Delimiter ;

Delimiter $$

CREATE PROCEDURE updateSoftwareProductStatus (IN id int)

BEGIN

DECLARE current\_streak int;

DECLARE rowcount int;

DECLARE Name VARCHAR(40);

DECLARE Version VARCHAR(10);

DECLARE updateDone INT DEFAULT 0;

DECLARE cur CURSOR FOR SELECT components\_in\_software\_products.name,components\_in\_software\_products.version FROM components\_in\_software\_products where components\_in\_software\_products.comp\_id = id;

-- DECLARE EXIT HANDLER FOR NOT FOUND

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET updateDone = 1;

set current\_streak=0;

open cur;

select FOUND\_ROWS() into rowcount ;

start\_loop: loop

IF updateDone =1 THEN

LEAVE start\_loop;

END IF;

fetch cur into Name,Version;

set current\_streak = current\_streak +1;

if ((select count(\*) from Components where Components.component\_status like 'not-ready' and Components.comp\_id in (SELECT components\_in\_software\_products.comp\_ID FROM components\_in\_software\_products where components\_in\_software\_products.name = Name and components\_in\_software\_products.version = Version)) >0 ) then

update software\_products set software\_products.software\_status = 'not-ready' where software\_products.name = name and software\_products.version = version;

else if ((select count(\*) from Components where Components.component\_status like 'usable' and Components.comp\_id in (SELECT components\_in\_software\_products.comp\_ID FROM components\_in\_software\_products where components\_in\_software\_products.name = Name and components\_in\_software\_products.version = Version)) >0 ) then

update software\_products set software\_products.software\_status = 'usable' where software\_products.name = name and software\_products.version = version;

else

update software\_products set software\_products.software\_status = 'ready' where software\_products.name = name and software\_products.version = version;

end if;

end if;

if (current\_streak<=rowcount) then

leave start\_loop;

end if;

end loop;

close cur;

END $$

Delimiter ;

-- Triggers on Status (Inspection) --> WORKING

-- INSERT

delimiter $$

create trigger inspection\_status\_insert

before insert on Inspection

for each row

begin

if (new.score > 90 ) then

set new.status = 'ready';

else if (new.score < 75) then

set new.status = 'not-ready';

else

set new.status = 'usable';

end if;

end if;

CALL updateComponentsStatus(new.component\_name, new.version, new.status);

end;

$$

delimiter ;

-- Triggers on Status (Inspection) --> WORKING

-- UPDATE

delimiter $$

create trigger inspection\_status\_update

before update on Inspection

for each row

begin

DECLARE score\_value INT;

SET score\_value = (select score from Inspection where Inspection.inspection\_id = new.inspection\_id);

if (score\_value != new.score) then

signal sqlstate '45000'

set message\_text = 'Cannot update score';

end if;

end;

$$

delimiter ;

-- Triggers for Employees

-- Seniority

SET GLOBAL event\_scheduler = ON;

delimiter $$

CREATE EVENT seniority\_update

ON SCHEDULE

EVERY 1 day

DO

BEGIN

DECLARE current\_streak int;

DECLARE rowcount int;

Declare hire\_date timestamp;

Declare id int;

Declare date\_diff int;

DECLARE seniority\_temp VARCHAR(10);

DECLARE updateDone INT DEFAULT 0;

DECLARE cur CURSOR FOR SELECT id, hire\_date from employees;

-- DECLARE EXIT HANDLER FOR NOT FOUND

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET updateDone = 1;

set current\_streak=0;

open cur;

select FOUND\_ROWS() into rowcount ;

start\_loop: loop

IF updateDone =1 THEN

LEAVE start\_loop;

END IF;

fetch cur into id, hire\_date;

set current\_streak = current\_streak +1;

set date\_diff = ((UNIX\_TIMESTAMP(current\_date()) - UNIX\_TIMESTAMP(hire\_date))/60/60/24);

if (day\_diff < 365) then

update Employees set seniority = 'newbie' where Employees.id = id;

else if (day\_diff > 365 and day\_diff < 1825) then

update Employees set seniority = 'junior' where Employees.id = id;

else if (day\_diff > 1825) then

update Employees set seniority = 'senior' where Employees.id = id;

end if;

end if;

end if;

if (current\_streak<=rowcount) then

leave start\_loop;

end if;

end loop;

close cur;

END

$$

delimiter ;

-- Triggers on Employee Seniority --> WORKING

-- Insert

delimiter $$

create trigger employee\_seniority\_update

before insert on employees

for each row

begin

DECLARE day\_diff INT;

set day\_diff = ((UNIX\_TIMESTAMP(current\_date()) - UNIX\_TIMESTAMP(new.hire\_date))/60/60/24);

if (day\_diff < 365) then

set new.seniority = 'newbie';

else if (day\_diff > 365 and day\_diff < 1825) then

set new.seniority = 'junior';

else if (day\_diff > 1825) then

set new.seniority = 'senior';

end if;

end if;

end if;

end;

$$

delimiter ;

**3.a Applications (20 pts; Due: 12/6)**

* 1. You will be provided a list of application requirements
  2. Using the relations populated in Phase 2, you are to create SQL code to implement a set of queries against the database.
  3. Submit for grading, proof of execution of these queries and documentation of any changes made to your implementation.

**3.b Final Testing (20 pts; Due: 12/6):**

* 1. When you have submitted Phase 3.a for grading, Dr. Moore will perform his own testing of the database. Each student will have the same tests performed.

NOTES:

1. Requirements are subject to change at any time at the discretion of the user.
2. If you change details of an earlier phase implementation, please provide detail of this with your submission during the next phase.
3. You will receive a written grade on each phase while you are working on the next phase.