MARMARA UNIVERSITY ENGINEERING FACULTY

COMPUTER ENGINEERING

CSE400 Part2

INTERNSHIP REPORT

HALE ŞAHİN – 150116841

ZİRAAT TEKNOLOJİ / DATA WAREHOUSE DEPARTMENT

08/17 – 09/07 (10 working days)

1. About The Firm

My CSE400 Internship was divided on three parts because of my summer school, and firm's availability. In second part, I did my internship at Ziraat Teknoloji, Yıldız Teknopark, Davutpaşa. This firm is big and there are nearly 600 workers in it. The company serves finance sector with their own information accumulation and human resource power. Firm was first built as Fintek Financial Technology Services A.C. in 2001 in order to serve Halkbank and Ziraat Bank information technology services. Then, in 2014, it is called Zıraat Technology A.C.(Anonymous Company), and now its capital fully belongs to the Ziraat Bank. The firm develop apps in a variety of platforms, system operating and management, project management and consulting services in addition of the information technology service.[1]

Vision
The vision of Ziraat Teknoloji A.C. is that to become a pioneer who value their customers with innovative products, solutions and services in the industry of financial technologies.[2]

Mission
The mission of Ziraat Teknoloji A.C. is that to become a company who continuously improve itself by its research and developing activities, developing customers technology needs in the standards of contemporary norms, and provide an advantage for competition.[2]

2. Technologies Used

My internship was at the department of Data WareHouse. My Mentors are Ms. Seda, Ms.Kübra. I didn't took any database course yet, that is why there was a problem that what will i do. I suggested them that even if I had only 10 days, I can learn a lot of things while this process. And they accepted. Then in first 7 days, I spent my time on YouTube and any other sites to learn what is SQL, why and how we use this language, the syntax and making tables, joins, searching on PLSQL. And they supply an environment in PLSQL for me as the user of education they create for interns. At the end, I have my own project with my own data tables and my own functions which I create random. I create joins to understand join clause better and get the result as what I need.

3. Work Done

a. Week 1

First week was only one day for me, because of the holiday. This day I get used to the company ad learn what projects they have at present. They were using Teredata SQL and they are changing it to Oracle SQL. Like my first intern of this year, there are many meetings and hard workings in this company too. I learned that SQL is composed of some tables which are including some crucial information as categorized. You can create table, connect tables together, get some data from a table with your own constraints, update a value in a table, delete an entry in table, and add new data to the table.

Data WareHouses are inventory for database of really big companies. For this company, there are 50 million users which means that when you try to update a data in a table, and someone tries to access that specific table, it can break. There are some solutions for that. Ms. Seda call for top 10 entries in a real big table in order to show me that if we called all of them we can lock the system. I learned that in Teredata, there are the words '*When*', '*Where*' and they are used for conditional statements when you use your constraints. Another information I learned that Left Join takes the first table as the base table. While I was waiting for my computer to be ready, I watch Ms. Seda and I learned much things. Join syntax is used for the combine columns of the selected tables according to the rule of the type of the join. To not lock the system in Teredata, we call tables as view format. The code for it will be like '*Show table; Show view*'. '*Select Distinct*' syntax is used for call some specific data.

My mentor Seda is using Teredata, and when I follow her working, I understand that the problems come to her via email, and she back emails the solutions. Firstly, she call the table which problem is in, find what is wrong or sometimes there is not problem, they just want to know some data or table access, she gives the answer for them via using Teredata and the database tables.

Then my computer has arrived, and my own PLSQL environment was opened. And I try to find my data in Ziraat Bank, and I couldn't find any, because in test environment, we can't access the all real data. But, in this way, I was made my first search in SQL. In PLSQL we select the code part we need to run and we run it with F8 key in keyboard. SQL helps us to access database.

An important point in tables is primary key. The 'Not Null' term is used and they should be unique, to identify entries from each other. Primary key is not mandatory, but if you can't tolerate two entry of the same entity, then you need to use it. For example, for customers table, customer id is used for identify them. '*Drop Table*' will delete table and all the records in it. '*Delete From TableName*' will delete all entries in that table except the table itself.

b. Week 2

There are functions which are used to do some calculations or changes and return a value. Procedures have codes, can call function or another procedure inside and they don't return a value. I learned them by examining of pre-made projects. '*Create or Replace Table*' helps us while creating a table, if we forgot that we already have this table and we can update it, if didn't have this table we become created it.

SELECT * FROM mytable;

This will bring you to all information in the mytable. * means all.

Entity is a term which has one or more attribute in it. Attributes are the column information of the tables. Entity is who has those information, entity can be a dog with foot number, skin color attributes, or it can be an employee with name, address and phone information of attributes.

There is a foreign key term which I come across a few times. I asked about it to Ms. Kübra, and I learned that it is unique like primary key and it is referencing for another table. For example, we have a trees and sales table. And we used tree as a foreign key in sales. Since there is no sales without tree, it should have used in here. And sales cannot have a tree which doesn't exist. That is why we should prevent deletion of the data in a table which has been referenced for another tables. There are indexes on the tables. It is used for data scan in a table or sort the data in ascending or descending order.

For example, let say in one table there employee attributes like hire date, salary, department information and in another table there are the information for promotion based on the hire date of each employee. Then, you want look which employee is which place in promotion, what is their data in that column of the table. Then you can use join based on the hire date which is the foreign key and you can narrow your columns with selecting just name and promotion status. There are types of joins; Left Join, Inner Join, Outer Join.

My first example was writing on a random empty table HelloWorld string. I used this code.

SELECT ‘HelloWorld’ FROM DUAL

I saw these examples on the video list of Caleb Curry on YouTube, ‘Oracle SQL Tutorials’. And my code worked, writes my string to the table below. DUAL means includes everything I guess like a default table.

SELECT 5+5 FROM DUAL

This code will gave the result of two rows first ‘5+5’ and the second ‘10’.

The value type for integer or double is NUMBER in PLSQL and for string we need to use VARCHAR2. after the type we can add constraint with parenthesis like ‘x NUMBER(5)’, x is a 5 digit number.

The word ‘check’ is like a boolean expression, true or false. For example in order to add a primary key constraint to the attribute of user_id, use the following syntax, or you can just write PRIMARY KEY after you write the type of it.

user_id NUMBER CONSTRAINT users_pk PRIMARY KEY

Foreign key definition within another line the declare of variable will be happen like below;

CONSTRAINT projects_usef_fk FOREIGN KEY (creator) REFERENCES users(username)

This line means that creator is a reference foreign key for projects users and its referenced by username attribute of the users table.

In a table, when we want to add new data for that entity, we will use the syntax below and we have to give a data for not null, PK and FK attributes. And the data for PK and that specified as unique ones should be different from each other.

INSERT INTO users VALUES(1, ‘FirstUser’);

Let say there is a user table and it has two attributes one thet is the identifier and declared as number, and the other is username and declared as a VARCHAR2.

To delete some entity's information we use this syntax;

DELETE FROM users WHERE user_id = 1;

This will delete all the data from which identified as 1 for the user id. When there foreign keys, references and connections and we try to delete a data where it used on other tables, this would be a problem. That is why there is a syntax 'ON DELETE CASCADE' which deletes all the connection data with it.

When you need to update a value which used by other tables as reference, then you need to 'ON UPDATE CASCADE' syntax. Then the new update will be done in the referenced parts too. Note that PK's should never update, FK's can be updated.

If I need to drop a table which is used for foreign keys then I need to do it like this;

DROP TABLE employees CASCADE CONSTRAINTS

After I learned basics on syntax I should learn about the joins which are really important in database systems. So again I watched YouTube videos on reference 4 in order to understand joins.

When columns which are participating in ON join condition have different name and same data type or same name and same data type.

When we use equality comparison operator for comparing columns over which we want to put join we can use USING clause supplying only that column name which we want to emphasize join. When the name and the data type of columns over which we want to put join in your query is the same, we can use USING clause.

We can use WHERE clause when we want to filter the result returned by our query.

There are four types of joins; Inner Join, Outer Join, Cross Join and Self Join. There are three outer joins as left, right and full outer join. Join conditions can be equality or nonequality.

Source table is the first table in the join tables and when you make left join then source table will be the determiner of number of rows. Target table is secondary table in join and it comes after Join clause.

Let say we want to join two attributes from two different tables. Department_name from departments table and city_name from locations table.

SELECT department_name,city FROM departments NATURAL JOIN locations.

We can use common columns as the determiner of join or we can use cross join which joins all the possibilities, it takes paired up products of the two table.

There is ON clause which you can add your join condition. For example let say there is a common and unique column manager_id for both employees and departments tables. And you can say that I want to do a join where in both tables manager_id matches. Then we need to add last part of our join syntax this;

SELECT employee_id, department_name FROM employess LEFT JOIN departments ON(employee.manager_id = departments.manager_id)

USING clause is just like ON clause but simpler. You can use it like following it will give same result with the last one.

SELECT employee_id, department_name FROM employees LEFT JOIN departments USING(manager_id)

ORDER BY means you can edit the results you selected as you wish. You can order them descending or ascending or according to the attribute you want.

Natural join only gives the matching result without any constraints. If we want to add some constraint we can use WHERE clause like this example;

SELECT emp_name, dept_name FROM employees NATURAL JOIN departments WHERE dept_name = 'IT'

EMPLOYEES

emp_id(PK)	emp_name
1	steve
2	nancy
3	john
4	ellen
5	julin

DEPARTMENTS

dept_id	dept_name	emp_id(FK)
1	sales	1
2	account	2
3	finance	3
4	IT	null
5	marketing	null

Result will be;

Emp_name	dept_name
Null	IT

After this I created my first table and another table I used all these information and I insert data, delete data, update data and I made natural join. But I don't have the record of them since they don't let me. So I just add my bigger project in the week 3, because I did that in my home computer again , I took notes for me to make it exactly same. But, I couldn't get to do for all of my work to put them in my report.

c. Week 3

Last week my join working was left unfinished. So I go on this week finished it and I did my project which I am asked to do.

Inner join returns the result that both source and target table satisfy the condition. We should use inner join with ON or USING clause. Example for ON clause;

SELECT emp_name, dept_name FROM employees INNER JOIN departments ON(employees.emp_id = departments.emp_id)

Emp_name	dept_name
Steve	sales
Nancy	account
John	finance

This means that there is a emp_id attribute for both of them and it can be used to matching condition. We are comparing values of emp_id column of employees table with the values of emp_id column in departments table.

SELECT emp_name, dept_name FROM employees INNER JOIN departments USING(emp_id)

This will return result exact same with the one with the ON clause.

Cross Join which can be called as Cartesian Join because it is like cartesian product. Each row in the first table is paired up with all the rows in the second table. Therefore total number of result row will be product of the row number of first table and the row number of second table.

SELECT emp_id,emp_name,dept_name FROM employees,departments

OR

SELECT emp_id,emp_name,dept_name FROM employees CROSS JOIN departments

In here we can’t use any ON,WHERE clause, because even if there is a condition it will pair all them up again, so no need for that.

Right outer join is a form of outer join returns all entries from second table(right side table) which in here considered as source table, and return only the condition satisfied values from the first table which is here considered as target table. If there is a row that first column from first table is not satisfied the condition but you have to right the other columns from second table since it is right outer join, then it will write ‘null’ instead of first table’s non-satisfied attribute value.

Left outer join is a form of outer join returns all entries from first table(left side table) which in here considered as source table, and return only the condition satisfied values from the second table which is here considered as target table. If there is a row that a column from second table is not satisfied the condition but you have to right the other columns from first table since it is left outer join, then it will write ‘null’ instead of second table’s non-satisfied attribute value. LEFT OUTER JOIN usually used like LEFT JOIN, and thehy are totally same with each other.

SELECT emp_name, dept_name FROM employees LEFT OUTER JOIN departments USING(emp_id)

Emp_name	dept_name
Steve	sales
Nancy	account
John	finance
Ellen	null
Julin	null

This returned all the entries in employee first column, and for the dept_name only satisfied values will be returned for the others it will write null.

SELECT emp_name, dept_name FROM departments LEFT JOIN employees ON(employees.emp_id = departments.dept_id)

Emp_name	dept_name
Steve	sales
Nancy	account
John	finance
null	IT
null	marketing

This returned the condition of emp id and dept id values are satisfied, they are same data type and different name which is why we can use them in ON clause.

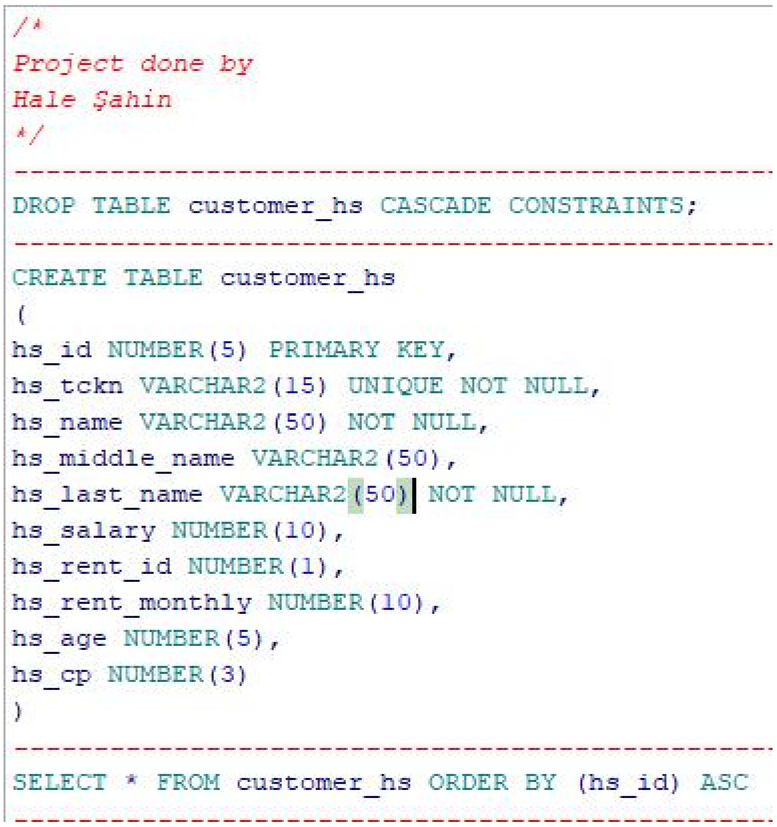
Full outer join is a kind of a combination of both left and right outer joins. And it will return all the rows from left and the right sided tables.

SELECT emp_name, dept_name FROM employees FULL OUTER JOIN departments ON(employees.emp_id = departments.emp_id);

Emp_name	dept_name
Steve	sales
Nancy	account
John	finance
Null	IT
Null	marketing
Ellen	null
Julin	null

This result till row 5 is similar to that of right outer join, as all the records from right side table which satisfy the join condition are here in the result. Followed by the remaining non-satisfied records of employees table.

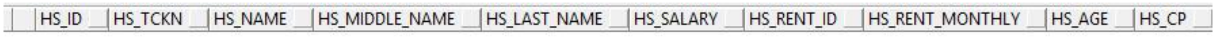
After I learned clearly enough to create my own project, I started to it.



```
/*
Project done by
Hale Şahin
*/
--------------------------------------------------
DROP TABLE customer_hs CASCADE CONSTRAINTS;
--------------------------------------------------
CREATE TABLE customer_hs
(
hs_id NUMBER(5) PRIMARY KEY,
hs_tckn VARCHAR2(15) UNIQUE NOT NULL,
hs_name VARCHAR2(50) NOT NULL,
hs_middle_name VARCHAR2(50),
hs_last_name VARCHAR2(50) NOT NULL,
hs_salary NUMBER(10),
hs_rent_id NUMBER(1),
hs_rent_monthly NUMBER(10),
hs_age NUMBER(5),
hs_cp NUMBER(3)
)
--------------------------------------------------
SELECT * FROM customer_hs ORDER BY (hs_id) ASC
--------------------------------------------------
```

Figure 1. Create the table

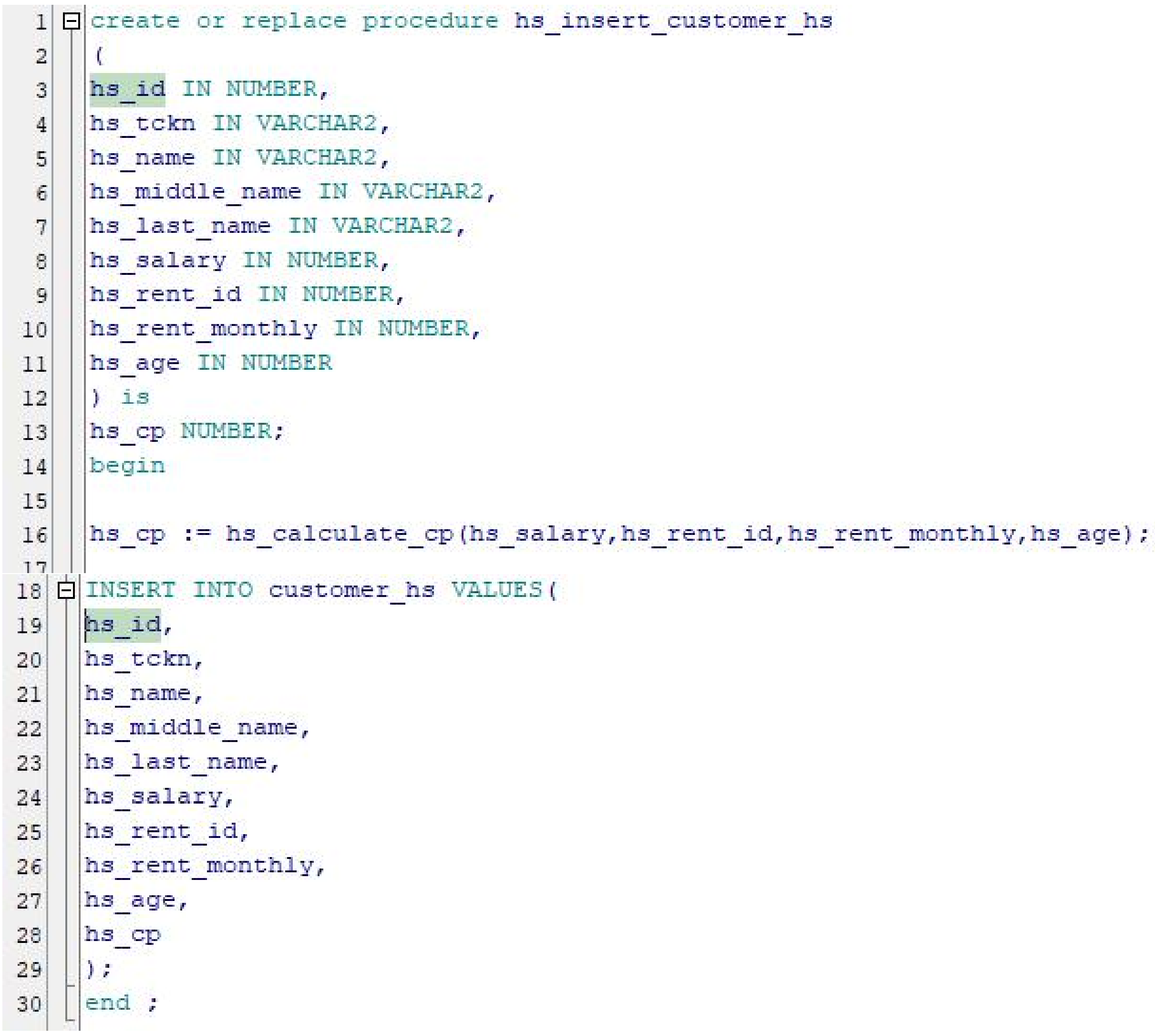
In figure 1, I create my table and since there were so many table, I named all my work with the start of my initials like a signature of me. This is a customer table with its values and declare value types. NUMBER(11) means 11 digit number. Since I used table informations for the other tables as reference, I need to used CASCADE clause to drop. My values are, hs_id means that the identifier of my table customer id and it is primary key which cannot be null and have to be unique. I have tckn value which is the social security number for Turkey and it is also not null and unique as I declared, but for this table it is not PK. Then I have name, middle name, last name where name and last name cannot be null. There is salary,rent_id, rent_monthly, age and cp attributes. Rent id determines if the customer is in rent or residence. Rent monthly is for if customer is rent then how much he/she pays for a month. And cp is stands for credit point for that customer.



HS_ID	HS_TCKN	HS_NAME	HS_MIDDLE_NAME	HS_LAST_NAME	HS_SALARY	HS_RENT_ID	HS_RENT_MONTHLY	HS_AGE	HS_CP

Figure 2.My tables first vision

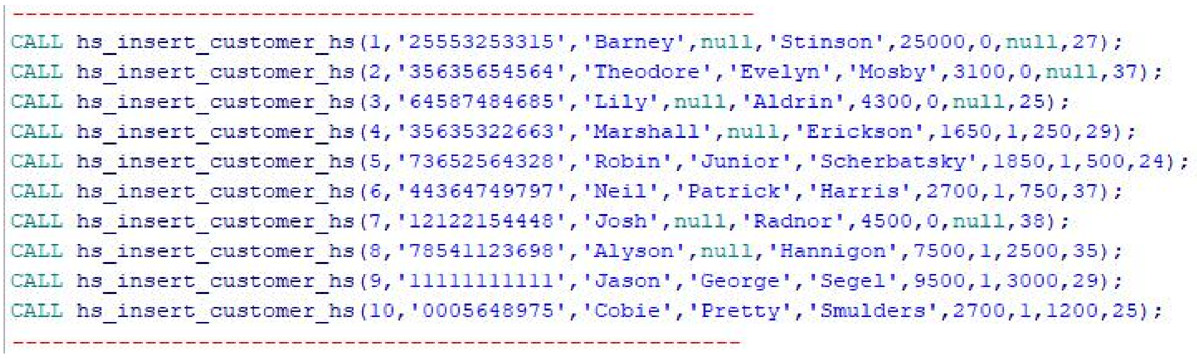
Now I can see that my table is created. And it is empty now, I will add some data into it. But first to make my work easy I did a procedure for insert data into customer table.



```
create or replace procedure hs_insert_customer_hs
(
hs_id IN NUMBER,
hs_tckn IN VARCHAR2,
hs_name IN VARCHAR2,
hs_middle_name IN VARCHAR2,
hs_last_name IN VARCHAR2,
hs_salary IN NUMBER,
hs_rent_id IN NUMBER,
hs_rent_monthly IN NUMBER,
hs_age IN NUMBER
) is
hs_cp NUMBER;
begin

hs_cp := hs_calculate_cp(hs_salary,hs_rent_id,hs_rent_monthly,hs_age);

INSERT INTO customer_hs VALUES(
hs_id,
hs_tckn,
hs_name,
hs_middle_name,
hs_last_name,
hs_salary,
hs_rent_id,
hs_rent_monthly,
hs_age,
hs_cp
);
end ;
```

Figure 3. Insert procedure for customer_hs table

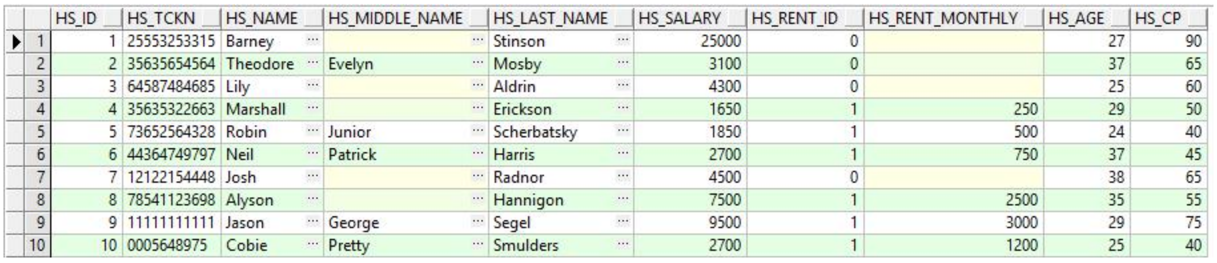
In customer_hs, credit point is calculated from the other attributes instead of taken from the user. First I calculated it and then I made insertion.



```
-------------------------------------------------------
CALL hs_insert_customer_hs(1,'25553253315','Barney',null,'Stinson',25000,0,null,27);
CALL hs_insert_customer_hs(2,'35635654564','Theodore','Evelyn','Mosby',3100,0,null,37);
CALL hs_insert_customer_hs(3,'64587484685','Lily',null,'Aldrin',4300,0,null,25);
CALL hs_insert_customer_hs(4,'35635322663','Marshall',null,'Erickson',1650,1,250,29);
CALL hs_insert_customer_hs(5,'73652564328','Robin','Junior','Scherbatsky',1850,1,500,24);
CALL hs_insert_customer_hs(6,'44364749797','Neil','Patrick','Harris',2700,1,750,37);
CALL hs_insert_customer_hs(7,'12122154448','Josh',null,'Radnor',4500,0,null,38);
CALL hs_insert_customer_hs(8,'78541123698','Alyson',null,'Hannigon',7500,1,2500,35);
CALL hs_insert_customer_hs(9,'11111111111','Jason','George','Segel',9500,1,3000,29);
CALL hs_insert_customer_hs(10,'0005648975','Cobie','Pretty','Smulders',2700,1,1200,25);
------------------------------------------------------
```

Figure 4. Insert some data into customer_hs

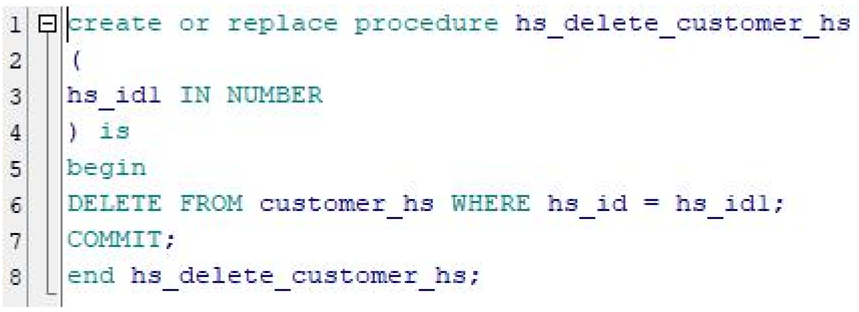
In figure 4, I called for the procedure and I made data insertion for 10 data, but in here we have to look out for PK values and cannot give empty values for an attribute, we only can write null for an empty part. Now my table looks like figure 5.



	HS_ID	HS_TCKN	HS_NAME	HS_MIDDLE_NAME	HS_LAST_NAME	HS_SALARY	HS_RENT_ID	HS_RENT_MONTHLY	HS_AGE	HS_CP
1	1	25553253315	Barney		Stinson	25000	0		27	90
2	2	35635654564	Theodore	Evelyn	Mosby	3100	0		37	65
3	3	64587484685	Lily		Aldrin	4300	0		25	60
4	4	35635322663	Marshall		Erickson	1650	1	250	29	50
5	5	73652564328	Robin	Junior	Scherbatsky	1850	1	500	24	40
6	6	44364749797	Neil	Patrick	Harris	2700	1	750	37	45
7	7	12122154448	Josh		Radnor	4500	0		38	65
8	8	78541123698	Alyson		Hannigon	7500	1	2500	35	55
9	9	11111111111	Jason	George	Segel	9500	1	3000	29	75
10	10	0005648975	Cobie	Pretty	Smulders	2700	1	1200	25	40

Figure 5. Data Inserted customer_hs table

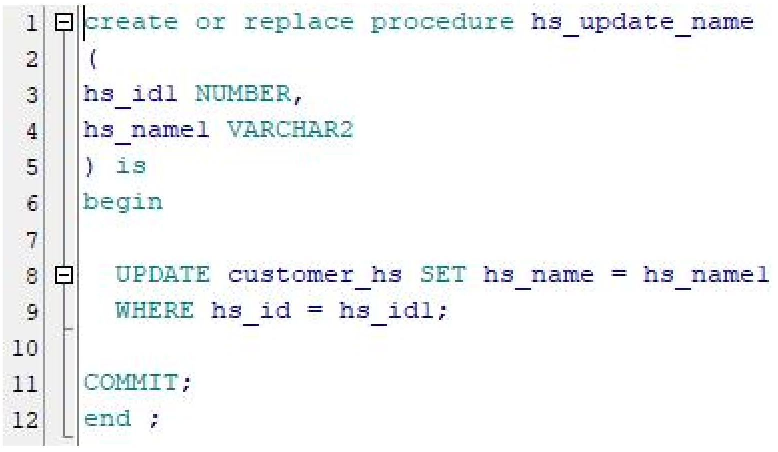
Then I made some changes on my data and to do it I write down a lot of procedures for each attributes update procedure.



```
create or replace procedure hs_delete_customer_hs
(
hs_id1 IN NUMBER
) is
begin
DELETE FROM customer_hs WHERE hs_id = hs_id1;
COMMIT;
end hs_delete_customer_hs;
```

Figure 6. Delete Procedure for customer_hs

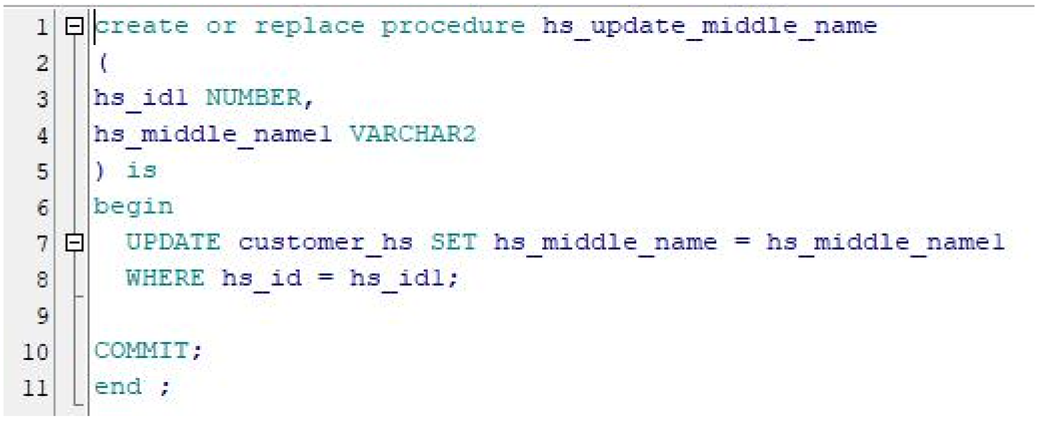
Figure 6 shows the delete procedure for an entry in customer_hs. It deletes by the primary key and it deletes all columns of a row.



```
create or replace procedure hs_update_name
(
hs_id1 NUMBER,
hs_name1 VARCHAR2
) is
begin

  UPDATE customer_hs SET hs_name = hs_name1
  WHERE hs_id = hs_id1;

COMMIT;
end ;
```

Figure 7. Update name attribute for customer_hs

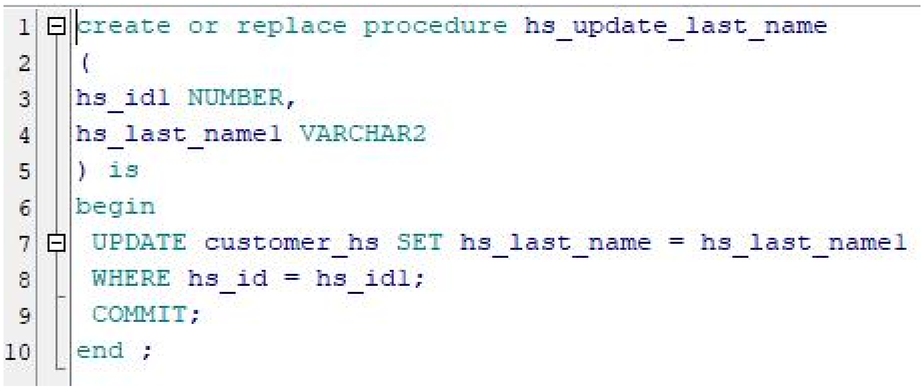
When we call hs_update_name procedure it will take two values as input, id for the identify whose name is going to change and the new name value.



```
create or replace procedure hs_update_middle_name
(
hs_id1 NUMBER,
hs_middle_name1 VARCHAR2
) is
begin
  UPDATE customer_hs SET hs_middle_name = hs_middle_name1
  WHERE hs_id = hs_id1;

COMMIT;
end ;
```

Figure 8. Middle name update

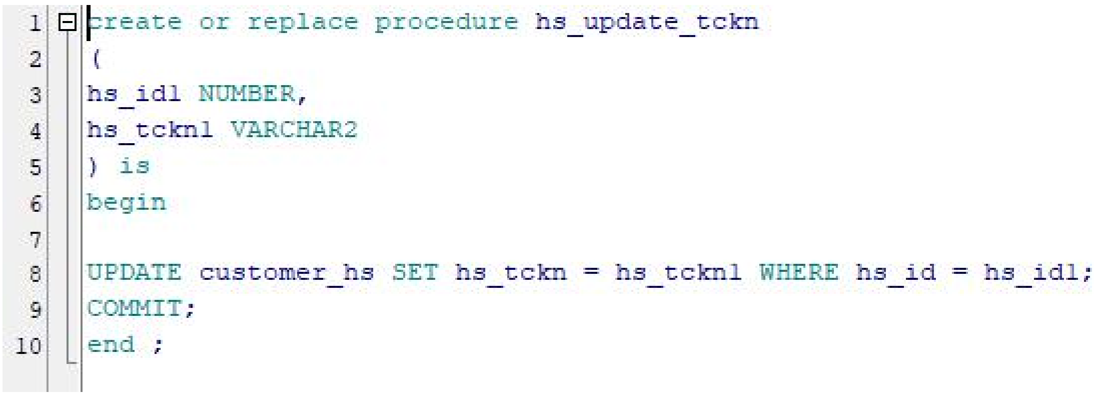
Figure 8 just like the name update procedure for middle name.



```
create or replace procedure hs_update_last_name
(
hs_id1 NUMBER,
hs_last_name1 VARCHAR2
) is
begin
 UPDATE customer_hs SET hs_last_name = hs_last_name1
 WHERE hs_id = hs_id1;
 COMMIT;
end ;
```

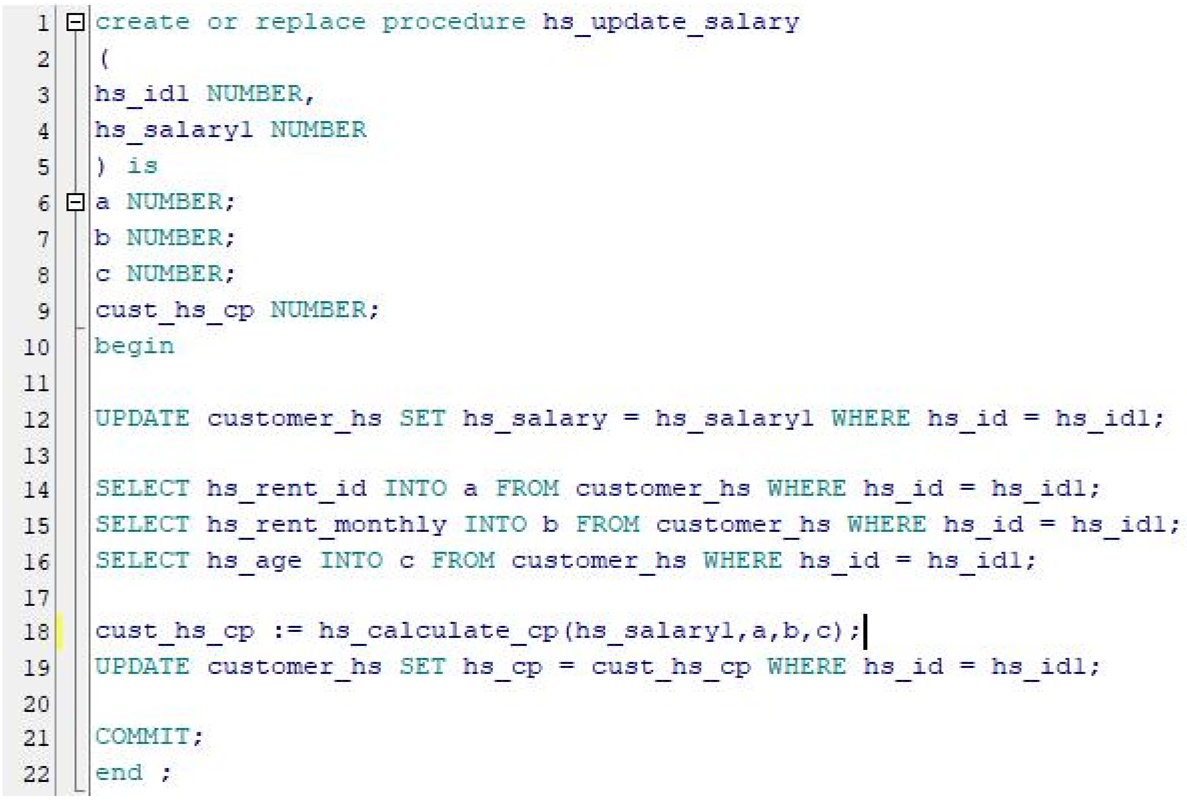
Figure 9. Last name update

Figure 9 is for last name changing just like figure 7 and 8.



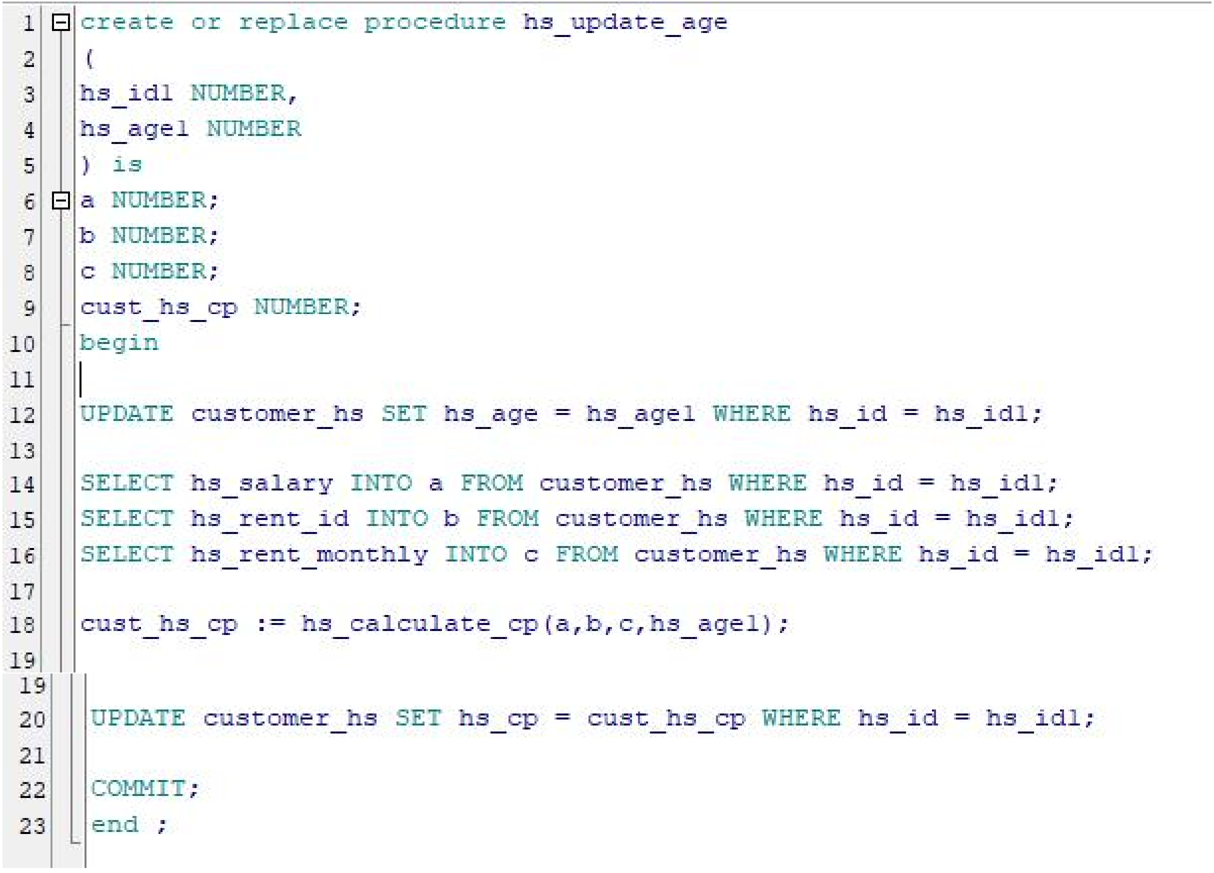
```
create or replace procedure hs_update_tckn
(
hs_id1 NUMBER,
hs_tckn1 VARCHAR2
) is
begin

UPDATE customer_hs SET hs_tckn = hs_tckn1 WHERE hs_id = hs_id1;
COMMIT;
end ;
```

Figure 10.Update tckn



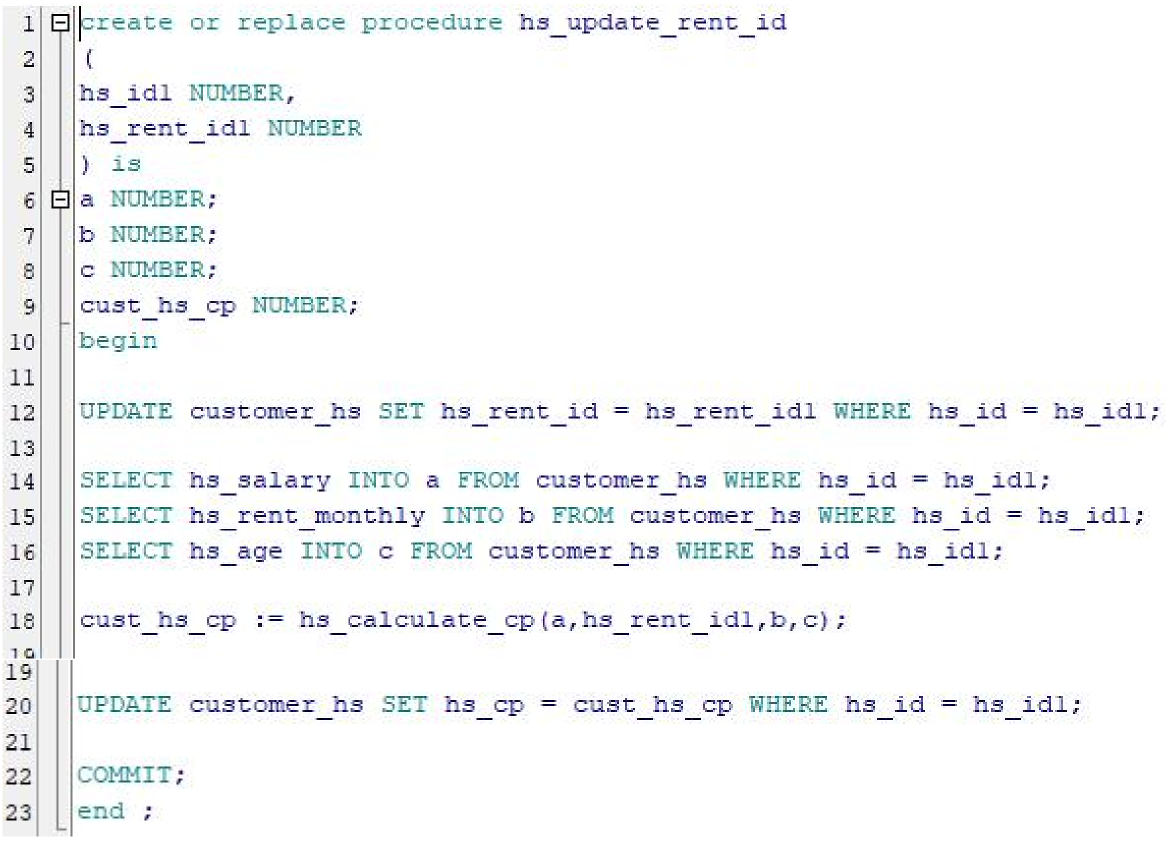
```
create or replace procedure hs_update_salary
(
hs_id1 NUMBER,
hs_salary1 NUMBER
) is
a NUMBER;
b NUMBER;
c NUMBER;
cust_hs_cp NUMBER;
begin

UPDATE customer_hs SET hs_salary = hs_salary1 WHERE hs_id = hs_id1;

SELECT hs_rent_id INTO a FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_rent_monthly INTO b FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_age INTO c FROM customer_hs WHERE hs_id = hs_id1;

cust_hs_cp := hs_calculate_cp(hs_salary1,a,b,c);
UPDATE customer_hs SET hs_cp = cust_hs_cp WHERE hs_id = hs_id1;

COMMIT;
end ;
```

Figure 11.Update salary of an entity



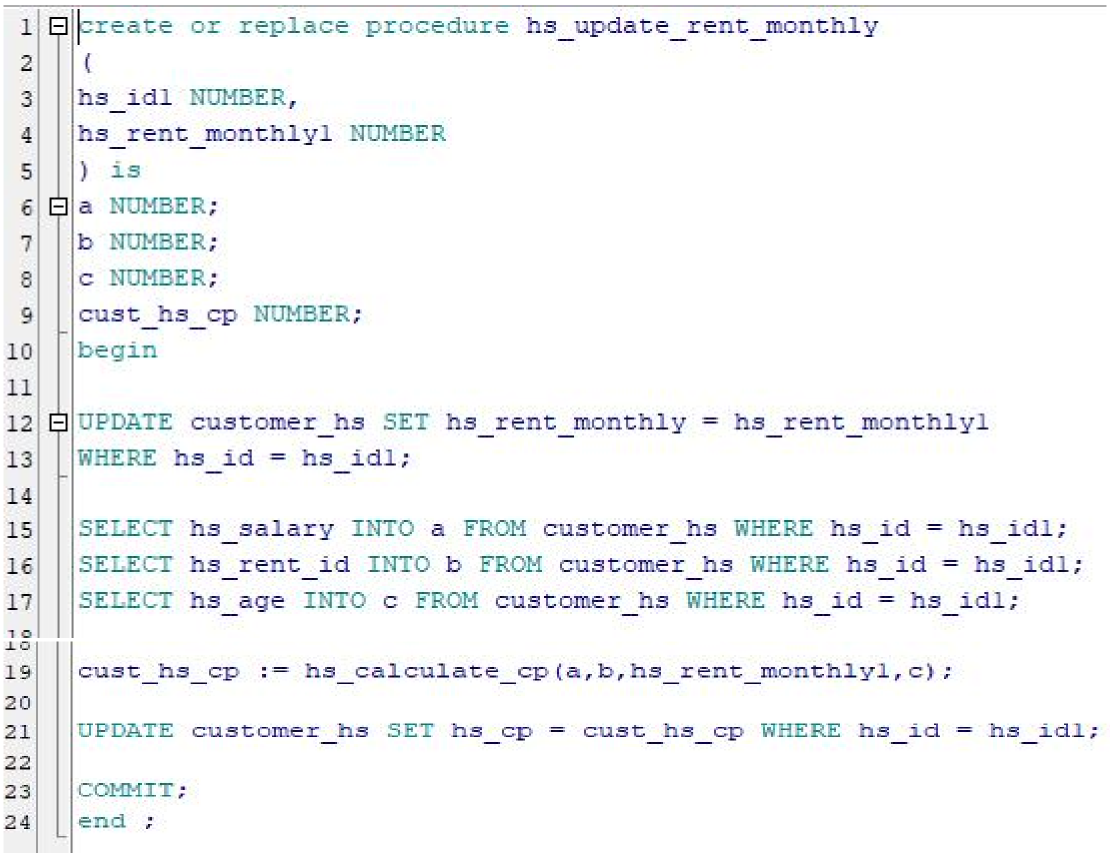
```
create or replace procedure hs_update_age
(
hs_id1 NUMBER,
hs_age1 NUMBER
) is
a NUMBER;
b NUMBER;
c NUMBER;
cust_hs_cp NUMBER;
begin

UPDATE customer_hs SET hs_age = hs_age1 WHERE hs_id = hs_id1;

SELECT hs_salary INTO a FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_rent_id INTO b FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_rent_monthly INTO c FROM customer_hs WHERE hs_id = hs_id1;

cust_hs_cp := hs_calculate_cp(a,b,c,hs_age1);


UPDATE customer_hs SET hs_cp = cust_hs_cp WHERE hs_id = hs_id1;

COMMIT;
end ;
```

Figure 12. Update age value for customer_hs



```
create or replace procedure hs_update_rent_id
(
hs_id1 NUMBER,
hs_rent_id1 NUMBER
) is
a NUMBER;
b NUMBER;
c NUMBER;
cust_hs_cp NUMBER;
begin

UPDATE customer_hs SET hs_rent_id = hs_rent_id1 WHERE hs_id = hs_id1;

SELECT hs_salary INTO a FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_rent_monthly INTO b FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_age INTO c FROM customer_hs WHERE hs_id = hs_id1;

cust_hs_cp := hs_calculate_cp(a,hs_rent_id1,b,c);

UPDATE customer_hs SET hs_cp = cust_hs_cp WHERE hs_id = hs_id1;

COMMIT;
end ;
```

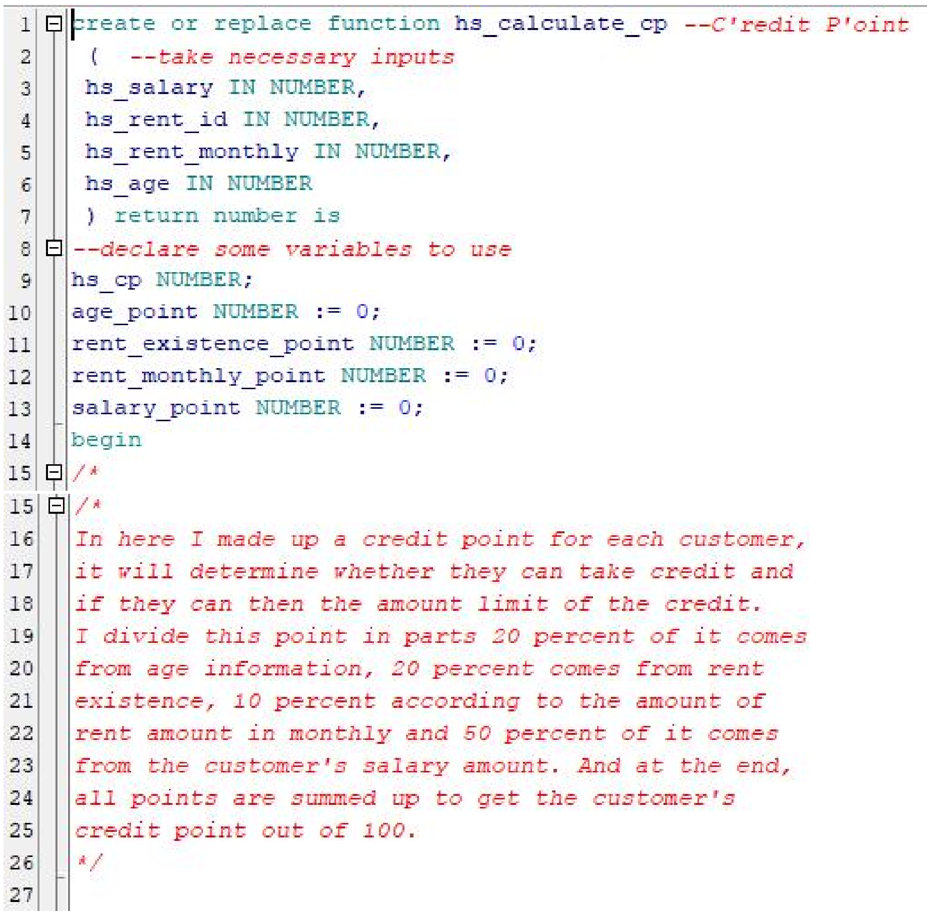
Figure 13. Update rent id for customer_hs entity



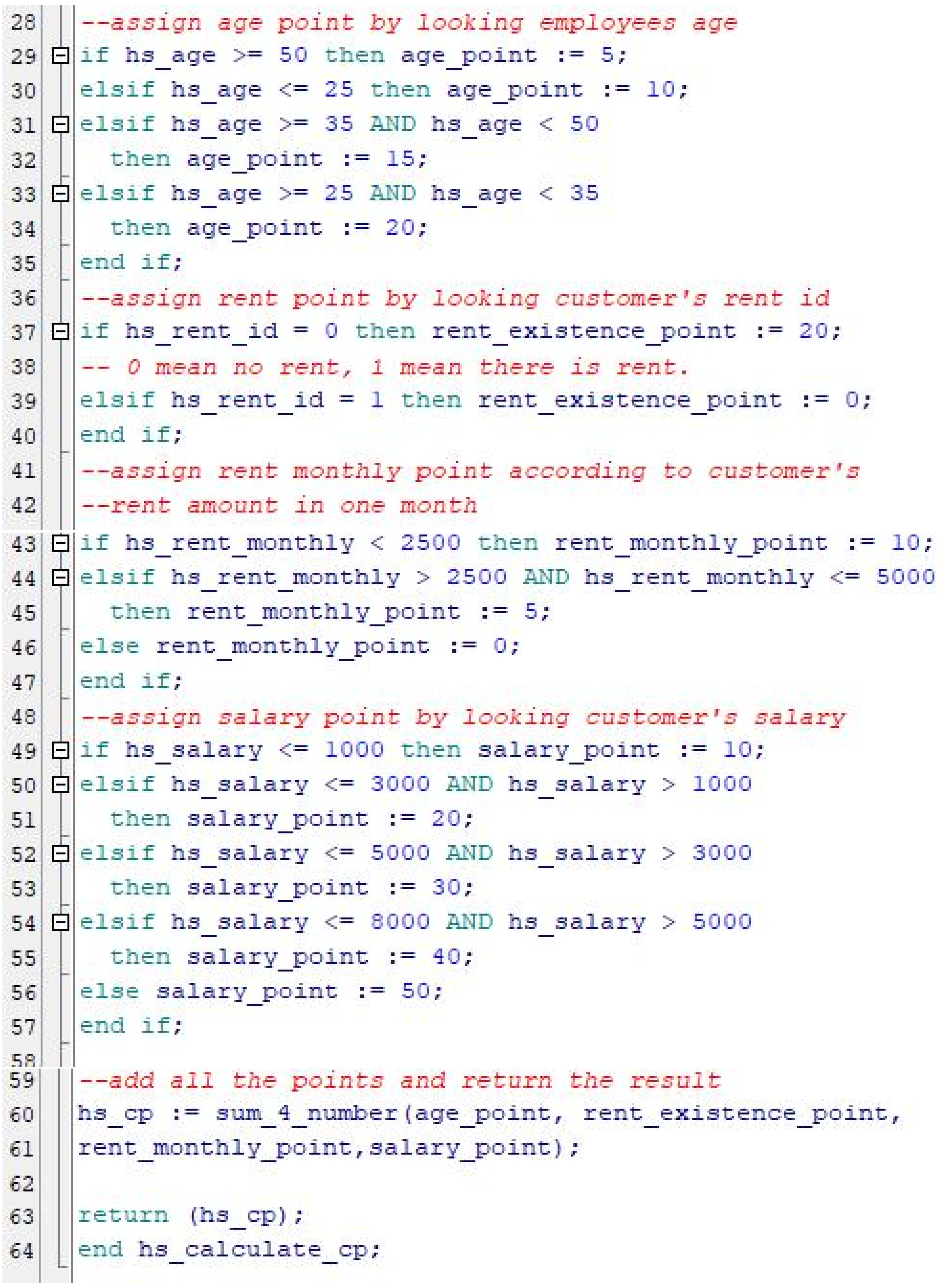
```
create or replace procedure hs_update_rent_monthly
(
hs_id1 NUMBER,
hs_rent_monthly1 NUMBER
) is
a NUMBER;
b NUMBER;
c NUMBER;
cust_hs_cp NUMBER;
begin

UPDATE customer_hs SET hs_rent_monthly = hs_rent_monthly1
WHERE hs_id = hs_id1;

SELECT hs_salary INTO a FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_rent_id INTO b FROM customer_hs WHERE hs_id = hs_id1;
SELECT hs_age INTO c FROM customer_hs WHERE hs_id = hs_id1;

cust_hs_cp := hs_calculate_cp(a,b,hs_rent_monthly1,c);

UPDATE customer_hs SET hs_cp = cust_hs_cp WHERE hs_id = hs_id1;

COMMIT;
end ;
```

Figure 14.Update rent_monthly attribute for customer_hs

Figure 11, Figure 12, Figure 13 and Figure 14 are similar to each other because I use the value of salary, age, rent_id and rent_monthly in order to calculate credit point. That is why when one of them changes, credit point should have to change(it remains same, it can increase or decrease) and I updated them with the new credit value. And I don't need to get the four values from the user let say for update age, I don't need to get rent_monthly too, I just can access it with the id. So I get all my needed information with just an id, and took from user id and the new values.



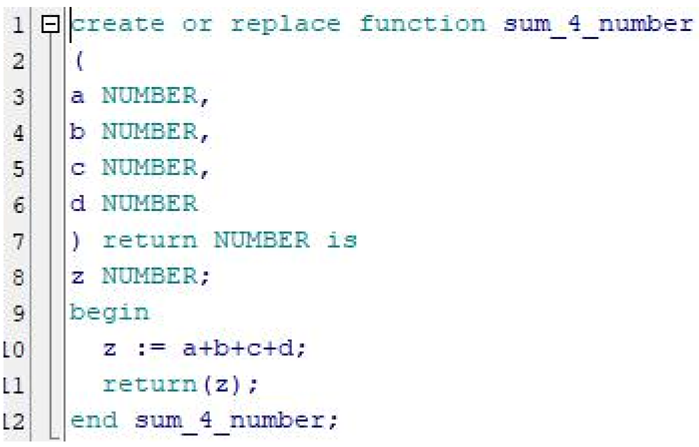
```
create or replace function hs_calculate_cp --C'redit P'oint
 (  --take necessary inputs
 hs_salary IN NUMBER,
 hs_rent_id IN NUMBER,
 hs_rent_monthly IN NUMBER,
 hs_age IN NUMBER
 ) return number is
--declare some variables to use
hs_cp NUMBER;
age_point NUMBER := 0;
rent_existence_point NUMBER := 0;
rent_monthly_point NUMBER := 0;
salary_point NUMBER := 0;
begin
/*
/*
In here I made up a credit point for each customer,
it will determine whether they can take credit and
if they can then the amount limit of the credit.
I divide this point in parts 20 percent of it comes
from age information, 20 percent comes from rent
existence, 10 percent according to the amount of
rent amount in monthly and 50 percent of it comes
from the customer's salary amount. And at the end,
all points are summed up to get the customer's
credit point out of 100.
*/
```



```
--assign age point by looking employees age
if hs_age >= 50 then age_point := 5;
elsif hs_age <= 25 then age_point := 10;
elsif hs_age >= 35 AND hs_age < 50
  then age_point := 15;
elsif hs_age >= 25 AND hs_age < 35
  then age_point := 20;
end if;
--assign rent point by looking customer's rent id
if hs_rent_id = 0 then rent_existence_point := 20;
-- 0 mean no rent, 1 mean there is rent.
elsif hs_rent_id = 1 then rent_existence_point := 0;
end if;
--assign rent monthly point according to customer's
--rent amount in one month
if hs_rent_monthly < 2500 then rent_monthly_point := 10;
elsif hs_rent_monthly > 2500 AND hs_rent_monthly <= 5000
  then rent_monthly_point := 5;
else rent_monthly_point := 0;
end if;
--assign salary point by looking customer's salary
if hs_salary <= 1000 then salary_point := 10;
elsif hs_salary <= 3000 AND hs_salary > 1000
  then salary_point := 20;
elsif hs_salary <= 5000 AND hs_salary > 3000
  then salary_point := 30;
elsif hs_salary <= 8000 AND hs_salary > 5000
  then salary_point := 40;
else salary_point := 50;
end if;

--add all the points and return the result
hs_cp := sum_4_number(age_point, rent_existence_point,
rent_monthly_point,salary_point);

return (hs_cp);
end hs_calculate_cp;
```

Figure 15. Calculation procedure for Credit Point

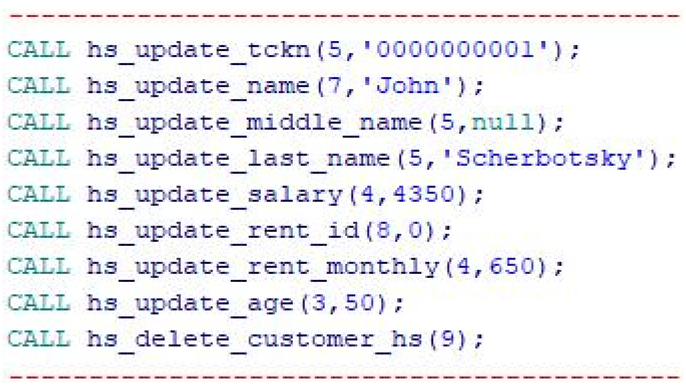
I explained which formula I used for to calculate credit point in figure 15. The values are not related to company, I just used them randomly, I made up the rules. I used if,elsif and else clause here. Ms. Kübra taught me them.



```
create or replace function sum_4_number
(
a NUMBER,
b NUMBER,
c NUMBER,
d NUMBER
) return NUMBER is
z NUMBER;
begin
  z := a+b+c+d;
  return(z);
end sum_4_number;
```

Figure 16. Sum 4 number

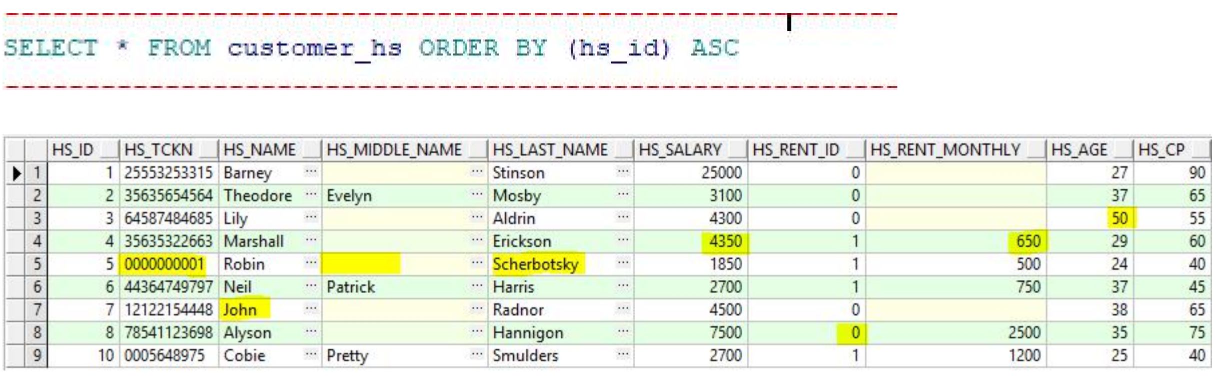
I used figure 16, sum 4 number procedure in calculate credit point procedure.



```
------------------------------------------
CALL hs_update_tckn(5,'0000000001');
CALL hs_update_name(7,'John');
CALL hs_update_middle_name(5,null);
CALL hs_update_last_name(5,'Scherbotsky');
CALL hs_update_salary(4,4350);
CALL hs_update_rent_id(8,0);
CALL hs_update_rent_monthly(4,650);
CALL hs_update_age(3,50);
CALL hs_delete_customer_hs(9);
------------------------------------------
```

Figure 17. Make changes on data

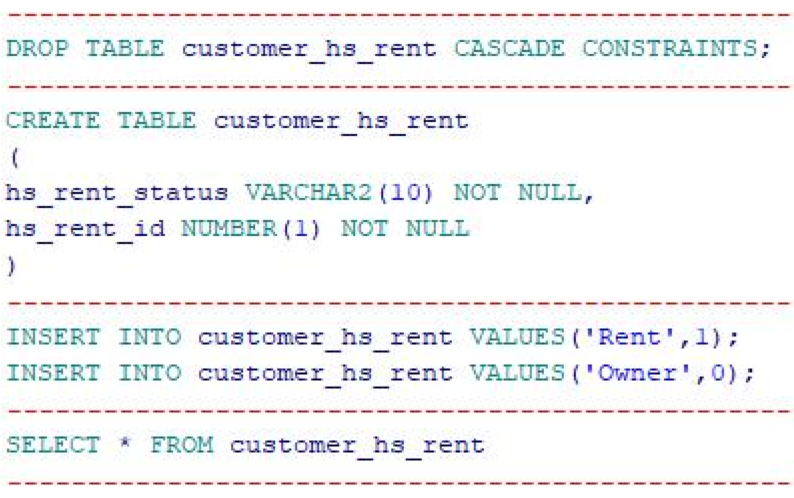
I called some random updates for some random entities. The result will show them changing in figure 18, differences from figure 5.



```
-------------------------------------------------------
SELECT * FROM customer_hs ORDER BY (hs_id) ASC
-------------------------------------------------------
```

	HS_ID	HS_TCKN	HS_NAME	HS_MIDDLE_NAME	HS_LAST_NAME	HS_SALARY	HS_RENT_ID	HS_RENT_MONTHLY	HS_AGE	HS_CP
1	1	25553253315	Barney		Stinson	25000	0		27	90
2	2	35635654564	Theodore	Evelyn	Mosby	3100	0		37	65
3	3	64587484685	Lily		Aldrin	4300	0		50	55
4	4	35635322663	Marshall		Erickson	4350	1	650	29	60
5	5	0000000001	Robin		Scherbotsky	1850	1	500	24	40
6	6	44364749797	Neil	Patrick	Harris	2700	1	750	37	45
7	7	12122154448	John		Radnor	4500	0		38	65
8	8	78541123698	Alyson		Hannigon	7500	0	2500	35	75
9	10	0005648975	Cobie	Pretty	Smulders	2700	1	1200	25	40

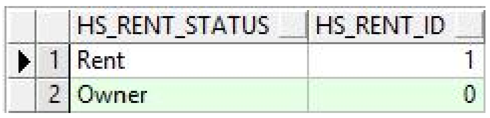
Figure 18. Result of the updates



```
--------------------------------------------------
DROP TABLE customer_hs_rent CASCADE CONSTRAINTS;
--------------------------------------------------
CREATE TABLE customer_hs_rent
(
hs_rent_status VARCHAR2(10) NOT NULL,
hs_rent_id NUMBER(1) NOT NULL
)
--------------------------------------------------
INSERT INTO customer_hs_rent VALUES('Rent',1);
INSERT INTO customer_hs_rent VALUES('Owner',0);
--------------------------------------------------
SELECT * FROM customer_hs_rent
--------------------------------------------------
```

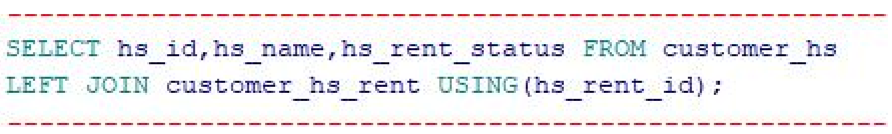
Figure 19. Rent table creation

I created another table as customer_hs_rent in figure 19. It has rent id and rent status. It took two data which shows that rent id 1 for 'rent' status and rent id 0 is for 'Owner' status. The result table is like this.



	HS_RENT_STATUS	HS_RENT_ID
1	Rent	1
2	Owner	0

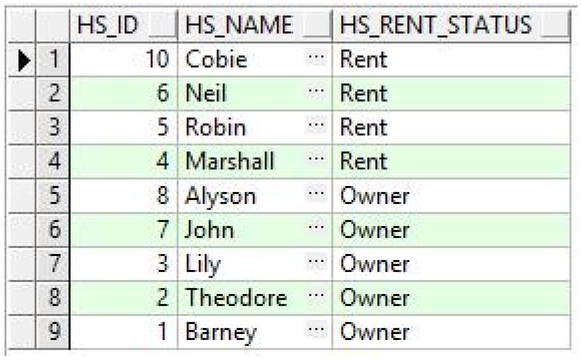
Figure 20. Rent table



```
-------------------------------------------------------
SELECT hs_id,hs_name,hs_rent_status FROM customer_hs
LEFT JOIN customer_hs_rent USING(hs_rent_id);
-------------------------------------------------------
```

Figure 21. Left Join example

In figure 21, I wanted to name of customers and rent status of each of them. That is why, I used left join to take all the customer name, and write only the join condition satisfied rent status. Here is the result;



	HS_ID	HS_NAME	HS_RENT_STATUS
1	10	Cobie ...	Rent
2	6	Neil ...	Rent
3	5	Robin ...	Rent
4	4	Marshall ...	Rent
5	8	Alyson ...	Owner
6	7	John ...	Owner
7	3	Lily ...	Owner
8	2	Theodore ...	Owner
9	1	Barney ...	Owner

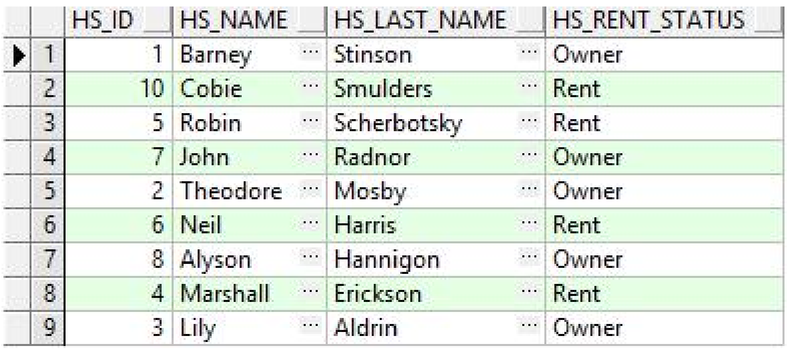
Figure 22. Result of join in figure 20

We can see all the customers in customer_hs and their rent status joined with rent id reference attribute in figure 22.

```
-------------------------------------------------------
SELECT hs_id,hs_name,hs_last_name,hs_rent_status
FROM customer_hs INNER JOIN customer_hs_rent
ON (customer_hs.hs_rent_id = customer_hs_rent.hs_rent_id)
ORDER BY(hs_last_name) DESC;
-------------------------------------------------------
```

Figure 23. Join Example

In figure 23, I used inner join again according to the rent id but this time with the ON clause. I will order the result in descending order for last name. The result is in figure 24.



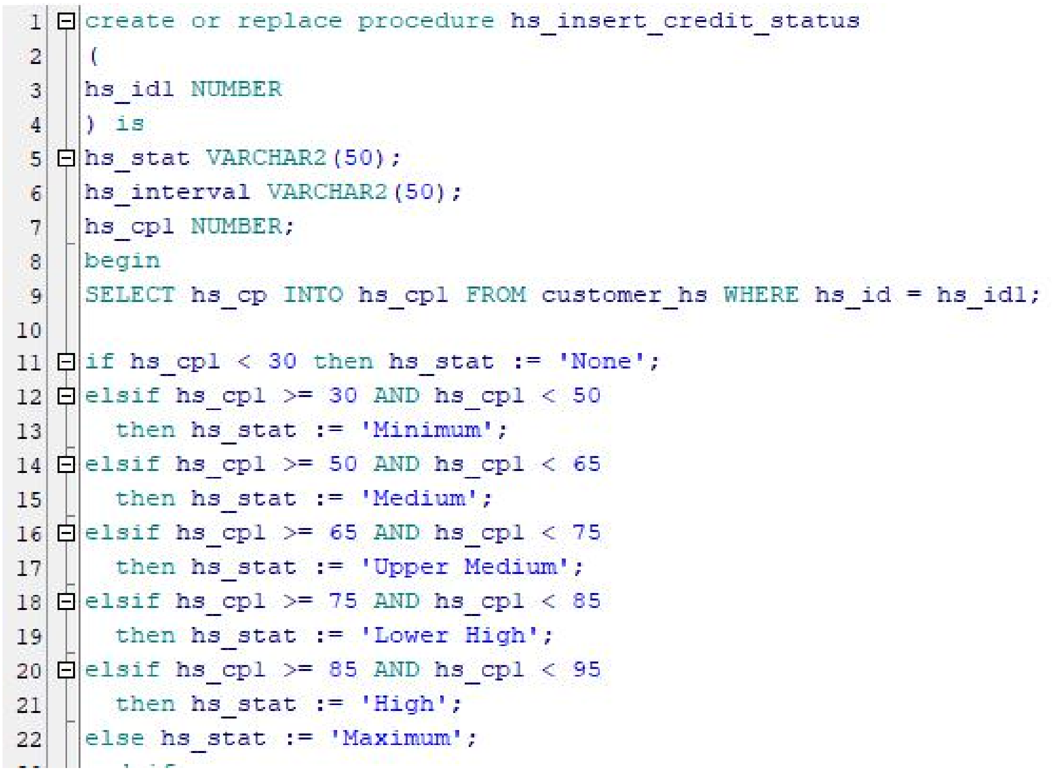
	HS_ID	HS_NAME	HS_LAST_NAME	HS_RENT_STATUS
1	1	Barney	Stinson	Owner
2	10	Cobie	Smulders	Rent
3	5	Robin	Scherbotsky	Rent
4	7	John	Radnor	Owner
5	2	Theodore	Mosby	Owner
6	6	Neil	Harris	Rent
7	8	Alyson	Hannigon	Owner
8	4	Marshall	Erickson	Rent
9	3	Lily	Aldrin	Owner

Figure 24. Result of join in figure 22

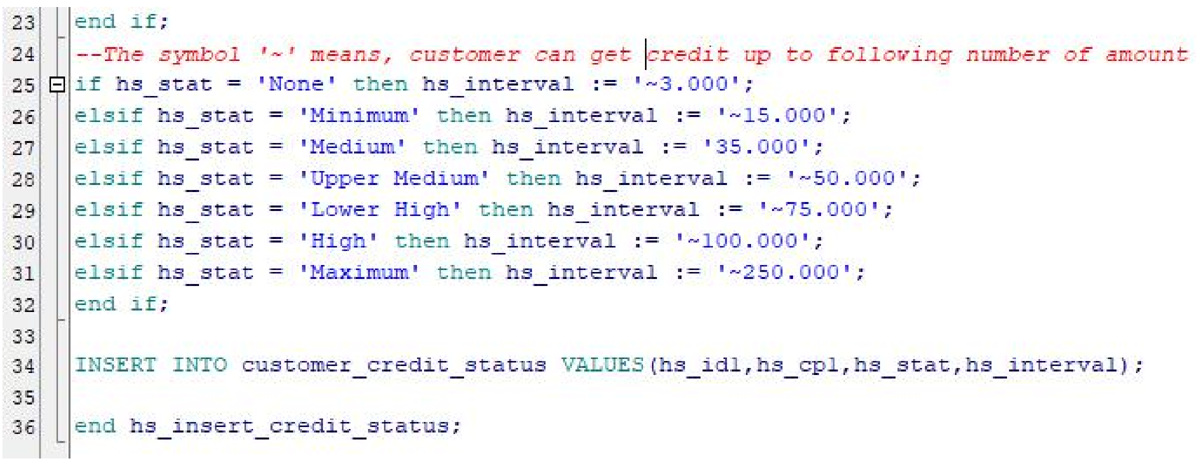
```
-------------------------------------------------------
DROP TABLE customer_credit_status CASCADE CONSTRAINTS;
-------------------------------------------------------
CREATE TABLE customer_credit_status
(
hs_id NUMBER(5) NOT NULL,
hs_cp NUMBER(3) NOT NULL,
hs_credit_status VARCHAR2(50) NOT NULL,
hs_credit_interval VARCHAR2(50)
)
-------------------------------------------------------
```

Figure 25. Credit status table creation

In figure 25, I created a new table similar like rent status, but this time I will assign credit status according to each credit point.



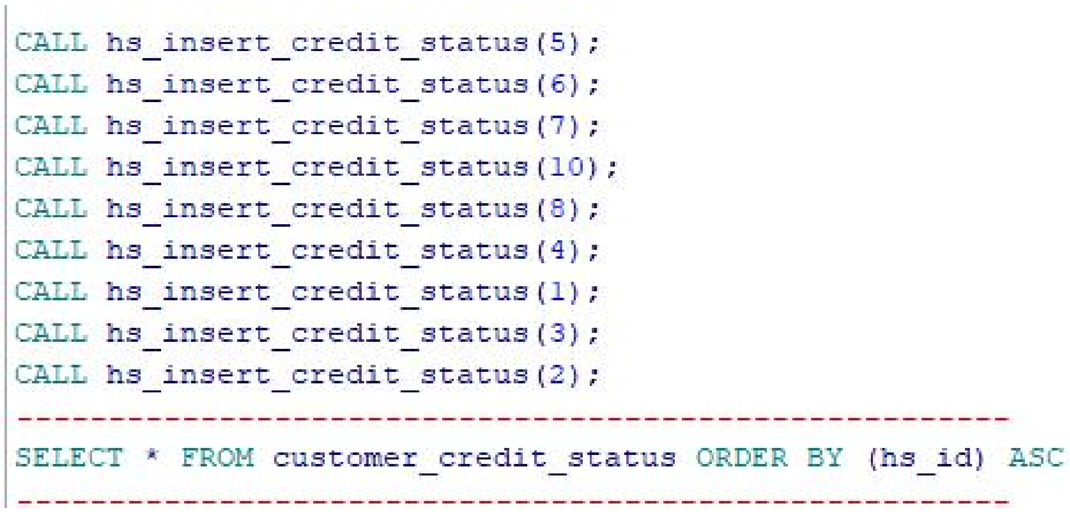
```
create or replace procedure hs_insert_credit_status
(
hs_id1 NUMBER
) is
hs_stat VARCHAR2(50);
hs_interval VARCHAR2(50);
hs_cp1 NUMBER;
begin
SELECT hs_cp INTO hs_cp1 FROM customer_hs WHERE hs_id = hs_id1;

if hs_cp1 < 30 then hs_stat := 'None';
elsif hs_cp1 >= 30 AND hs_cp1 < 50
  then hs_stat := 'Minimum';
elsif hs_cp1 >= 50 AND hs_cp1 < 65
  then hs_stat := 'Medium';
elsif hs_cp1 >= 65 AND hs_cp1 < 75
  then hs_stat := 'Upper Medium';
elsif hs_cp1 >= 75 AND hs_cp1 < 85
  then hs_stat := 'Lower High';
elsif hs_cp1 >= 85 AND hs_cp1 < 95
  then hs_stat := 'High';
else hs_stat := 'Maximum';
```



```
end if;
--The symbol '~' means, customer can get credit up to following number of amount
if hs_stat = 'None' then hs_interval := '~3.000';
elsif hs_stat = 'Minimum' then hs_interval := '~15.000';
elsif hs_stat = 'Medium' then hs_interval := '35.000';
elsif hs_stat = 'Upper Medium' then hs_interval := '~50.000';
elsif hs_stat = 'Lower High' then hs_interval := '~75.000';
elsif hs_stat = 'High' then hs_interval := '~100.000';
elsif hs_stat = 'Maximum' then hs_interval := '~250.000';
end if;

INSERT INTO customer_credit_status VALUES(hs_id1,hs_cp1,hs_stat,hs_interval);

end hs_insert_credit_status;
```

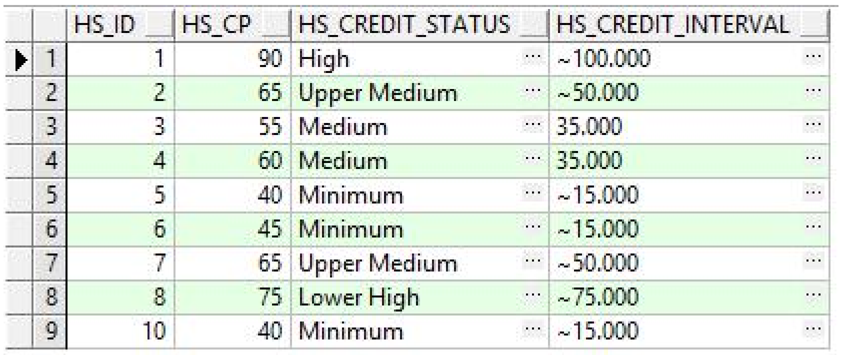
Figure 26. Insert credit status table

Figure 26, explains insert procedure for credit status table. It just need customer's id for the input. I can calculate all of them inside the procedure. I access credit point of that id of entity. And I just assign some random variables on credit status by looking at the credit point. In figure 27, I add some data into the credit status table by calling them with the customer id. The result of the inserted table is in figure 28. The insert procedure worked successfully.



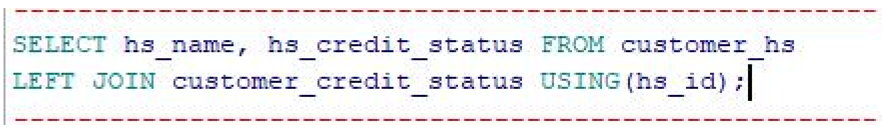
```
CALL hs_insert_credit_status(5);
CALL hs_insert_credit_status(6);
CALL hs_insert_credit_status(7);
CALL hs_insert_credit_status(10);
CALL hs_insert_credit_status(8);
CALL hs_insert_credit_status(4);
CALL hs_insert_credit_status(1);
CALL hs_insert_credit_status(3);
CALL hs_insert_credit_status(2);
------------------------------------------------------------
SELECT * FROM customer_credit_status ORDER BY (hs_id) ASC
------------------------------------------------------------
```

Figure 27. Insert data into credit status table



	HS_ID	HS_CP	HS_CREDIT_STATUS	HS_CREDIT_INTERVAL
1	1	90	High	~100.000
2	2	65	Upper Medium	~50.000
3	3	55	Medium	35.000
4	4	60	Medium	35.000
5	5	40	Minimum	~15.000
6	6	45	Minimum	~15.000
7	7	65	Upper Medium	~50.000
8	8	75	Lower High	~75.000
9	10	40	Minimum	~15.000

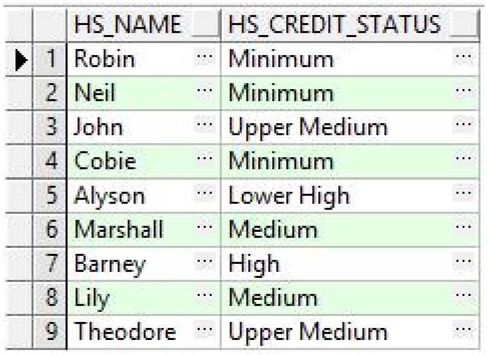
Figure 28. Credit status table



```
------------------------------------------------------------
SELECT hs_name, hs_credit_status FROM customer_hs
LEFT JOIN customer_credit_status USING(hs_id);
------------------------------------------------------------
```

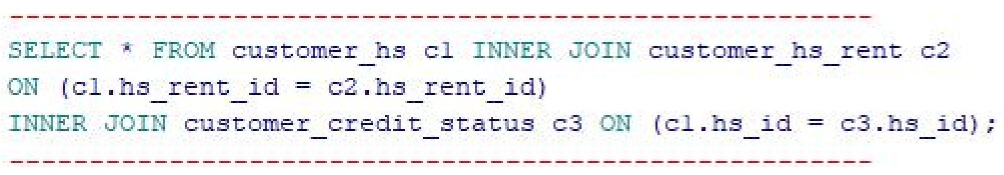
Figure 29. Join of two table

Figure 29 is a join example for customer and credit status table. Here is the result.



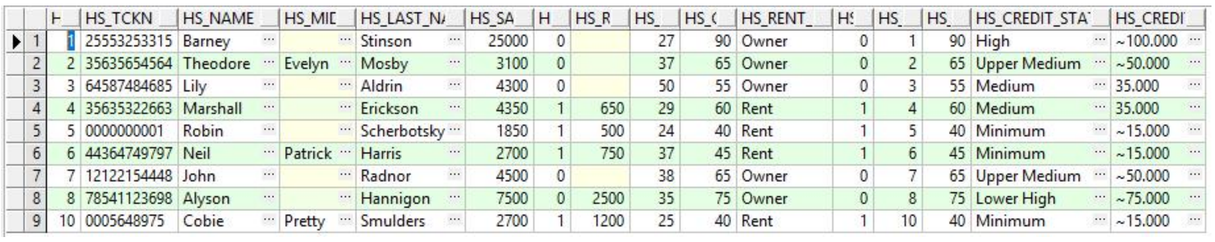
	HS_NAME	HS_CREDIT_STATUS
1	Robin	Minimum
2	Neil	Minimum
3	John	Upper Medium
4	Cobie	Minimum
5	Alyson	Lower High
6	Marshall	Medium
7	Barney	High
8	Lily	Medium
9	Theodore	Upper Medium

Figure 30. Result of the join figure 29



```
----------------------------------------------------
SELECT * FROM customer_hs c1 INNER JOIN customer_hs_rent c2
ON (c1.hs_rent_id = c2.hs_rent_id)
INNER JOIN customer_credit_status c3 ON (c1.hs_id = c3.hs_id);
----------------------------------------------------
```

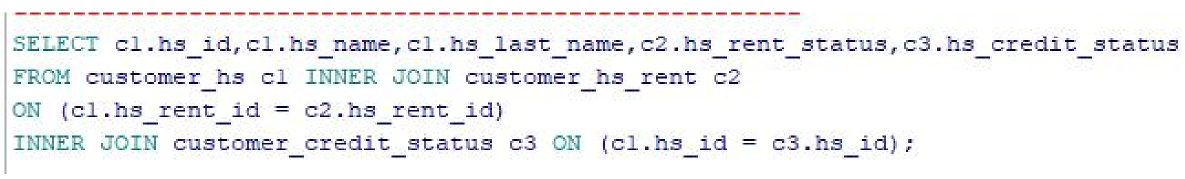
Figure 31. Join example for 3 table



	H	HS_TCKN	HS_NAME	HS_MIE	HS_LAST_N/	HS_SA	H	HS_R	HS_	HS_(	HS_RENT_	HS	HS_	HS_	HS_CREDIT_STA	HS_CREDI
1	1	25553253315	Barney		Stinson	25000	0		27	90	Owner	0	1	90	High	~100.000
2	2	35635654564	Theodore	Evelyn	Mosby	3100	0		37	65	Owner	0	2	65	Upper Medium	~50.000
3	3	64587484685	Lily		Aldrin	4300	0		50	55	Owner	0	3	55	Medium	35.000
4	4	35635322663	Marshall		Erickson	4350	1	650	29	60	Rent	1	4	60	Medium	35.000
5	5	0000000001	Robin		Scherbotsky	1850	1	500	24	40	Rent	1	5	40	Minimum	~15.000
6	6	44364749797	Neil	Patrick	Harris	2700	1	750	37	45	Rent	1	6	45	Minimum	~15.000
7	7	12122154448	John		Radnor	4500	0		38	65	Owner	0	7	65	Upper Medium	~50.000
8	8	78541123698	Alyson		Hannigon	7500	0	2500	35	75	Owner	0	8	75	Lower High	~75.000
9	10	0005648975	Cobie	Pretty	Smulders	2700	1	1200	25	40	Rent	1	10	40	Minimum	~15.000

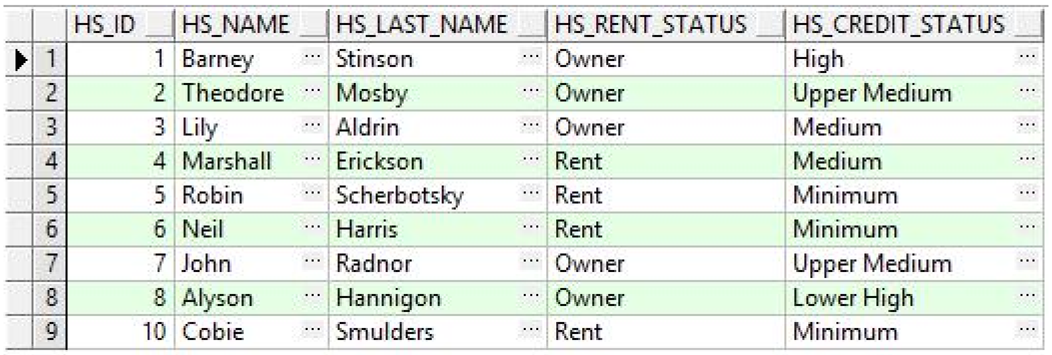
Figure 32. Result of the join figure 31

I try to join three tables together instead of selecting columns, I took all columns of all tables, and it worked properly. I can see every information about every entity.



```
----------------------------------------------------
SELECT c1.hs_id,c1.hs_name,c1.hs_last_name,c2.hs_rent_status,c3.hs_credit_status
FROM customer_hs c1 INNER JOIN customer_hs_rent c2
ON (c1.hs_rent_id = c2.hs_rent_id)
INNER JOIN customer_credit_status c3 ON (c1.hs_id = c3.hs_id);
```

Figure 33. Another Join example for three table



	HS_ID	HS_NAME	HS_LAST_NAME	HS_RENT_STATUS	HS_CREDIT_STATUS
1	1	Barney	Stinson	Owner	High
2	2	Theodore	Mosby	Owner	Upper Medium
3	3	Lily	Aldrin	Owner	Medium
4	4	Marshall	Erickson	Rent	Medium
5	5	Robin	Scherbotsky	Rent	Minimum
6	6	Neil	Harris	Rent	Minimum
7	7	John	Radnor	Owner	Upper Medium
8	8	Alyson	Hannigon	Owner	Lower High
9	10	Cobie	Smulders	Rent	Minimum

Figure 34. Result of the join example in figure 33

In figure 33, I used some selected columns for three table joins.

As a result, I can apply my information in PLSQL. I can search data with some specific information, I can add, delete, update a data, and add, delete, update a table. I can join the columns of table I chose, in which way I want, which rule I choose. This internship was so educative for me. I have the opportunity to apply my knowledge on a system for the first time.

4. References

[1] Ziraat Teknoloji, Access Date : October 7, 2018,

http://www.ziraatteknoloji.com/hakkimizda/hakkimizda.html

[2] Ziraat Teknoloji, Access Date : October 7, 2018,

http://www.ziraatteknoloji.com/hakkimizda/vizyon.html

[3] Caleb Curry, Oracle SQL Tutorials, Access Date : September 3, 2018

https://www.youtube.com/watch?v=QHYuuXPdQNM&list=PL_c9BZzLwBRJ8f9-pSPbxSS G6lNgxQ4m9

[4] Manish Sharma, SQL Tutorials, Access Date : September 5, 2018,

https://www.youtube.com/watch?v=6fky1a7gFOs&list=PLMlNiWEoh5Qpy-AzQAJdt26M0L XS7DgxQ