DBMS2 - FINAL PROJECT

RESOURCES: <https://www.kaggle.com/ekrembayar/fifa-21-complete-player-dataset>

GIT HUB

FRONT END & BACK END: <https://github.com/rooohaa/dbms-project.git>

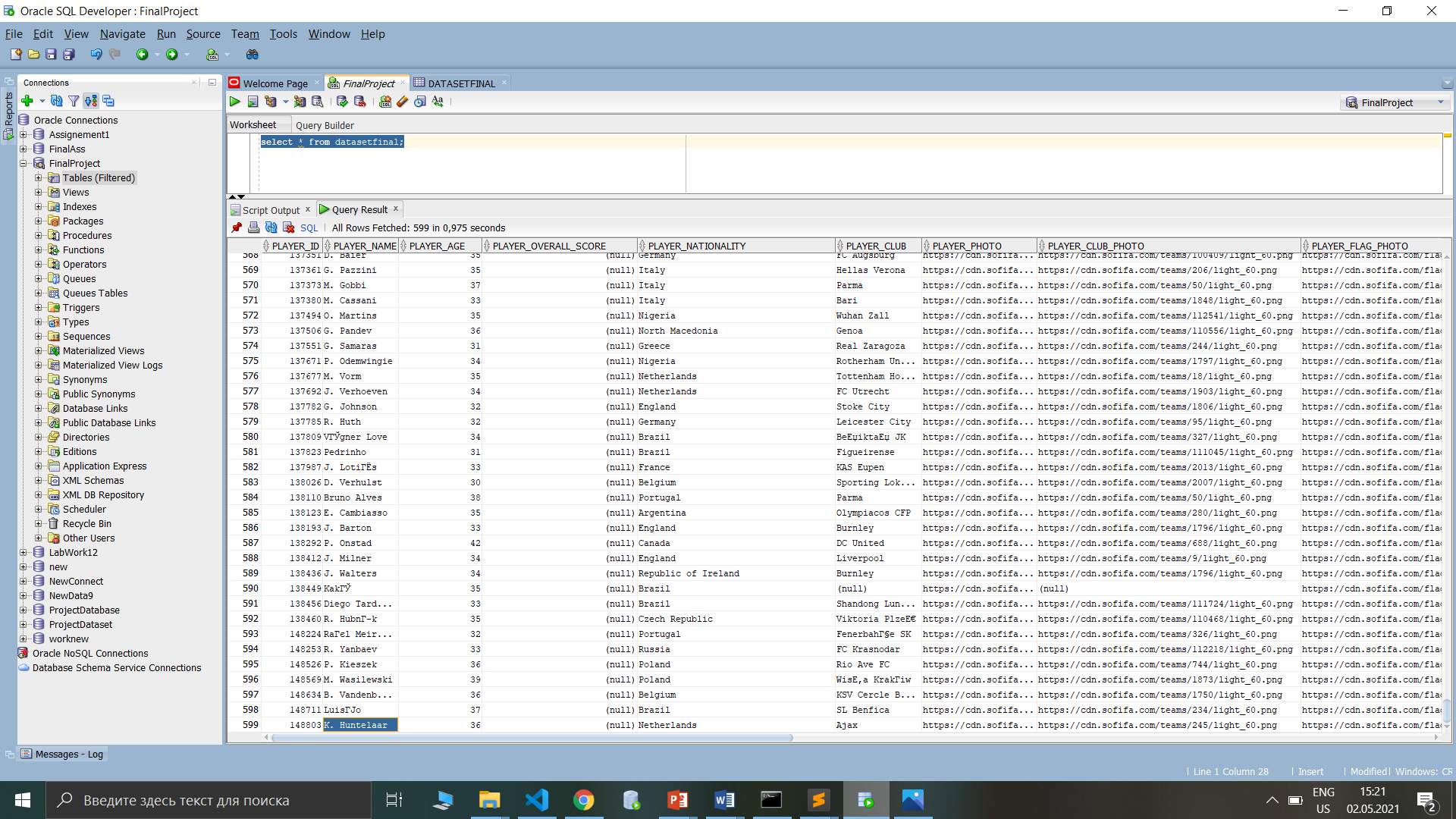
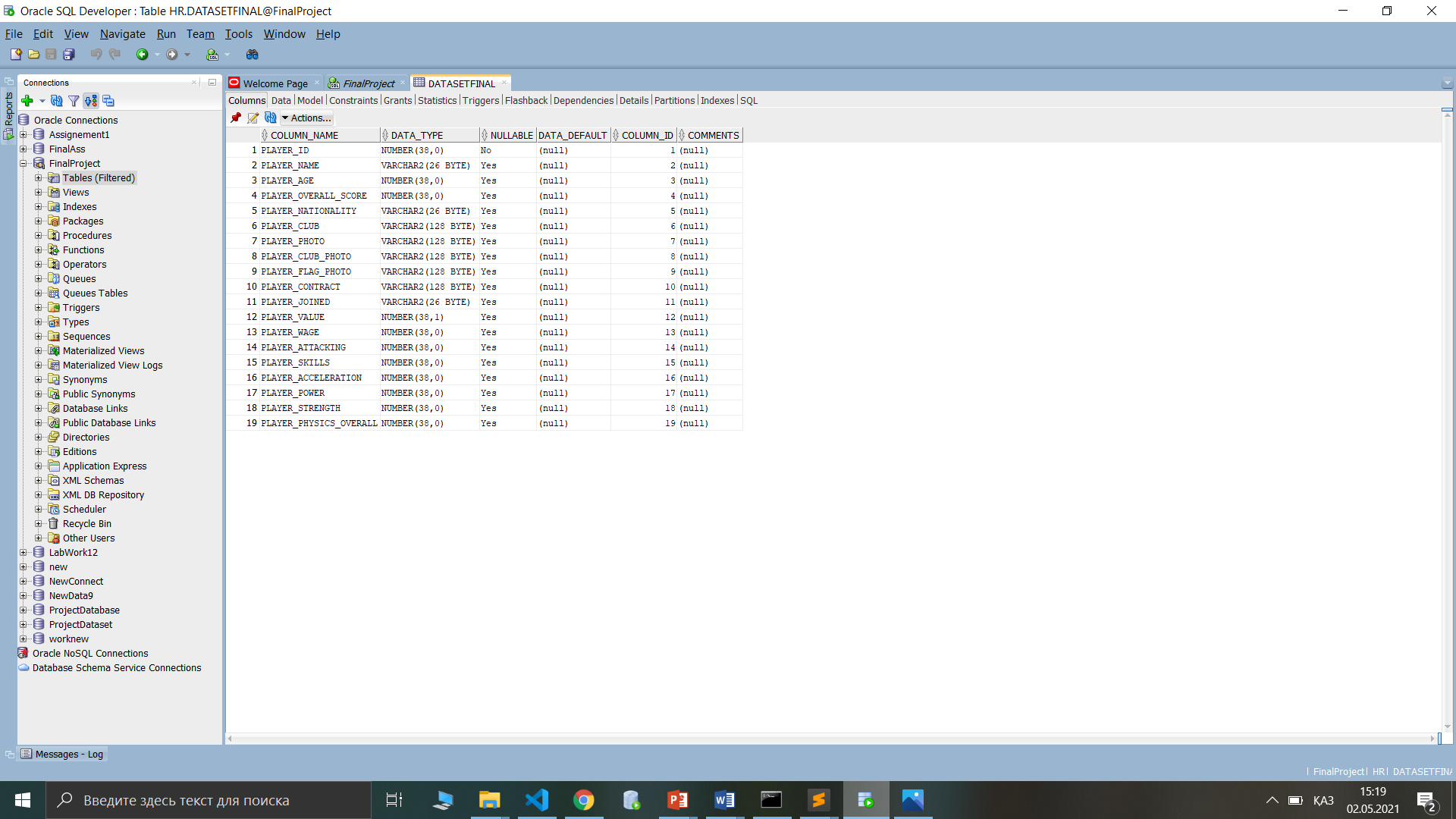
DATABASE: <https://github.com/MukhammedAli/dbms-project-database.git>

This project is about displaying information of football players and manipulating some data by using insert/update/delete operations. We took some style of analyzing data and displaying it to client from famous fifa players record. As you can see in the front end part, on the main page it gives ability to see all players based by their id’s which contains personal information about them. Also if client wants to find special data about player it is possible to search by using input button. In addition , we’ve completed bonus task where we need to add recommendation . Every time when client interacts with application by using DML statements our display table will change . Sorting and graph processing are done in our project . Under the hood of this software we’ve used modern new technologies like React.js library on frontend , NodeJs + Express for backend side, also Chart.js library to construct specific graphs, charts from our data and Oracle database which builds main logic for our application.

Let’s dive into the main core of our project , that is oracle implementation.

ATTENTION: All requirements are fulfilled, except image processing.

For this project we use two tables. The main table is datasetfinal which contains more than 500 rows and 10 columns. And second one is recommendation table which servers as bonus part of our project implementation. By the way recommendation table has the same construct . DATASETFINAL contains numeric , varchar2 data types . Player\_id has primary key constraint.



--FIRST PACKAGE

create or replace package first\_package as

procedure create\_index;

cursor help\_cursor is select player\_id , player\_attacking ,

player\_skills from datasetfinal;

new\_rec help\_cursor%rowtype;

type t\_newtype is table of help\_cursor%rowtype

index by binary\_integer;

new\_record t\_newtype;

v\_var number;

end first\_package;

create or replace package body first\_package as

procedure create\_index

as

begin

open help\_cursor;

loop

fetch help\_cursor into new\_rec;

new\_record(new\_rec.player\_id):= new\_rec;

exit when help\_cursor%notfound;

end loop;

close help\_cursor;

for i in new\_record.first .. new\_record.last

loop

if(new\_record.exists(i))

then

v\_var := new\_record(i).player\_attacking +

new\_record(i).player\_skills;

update datasetfinal

set player\_overall\_score = v\_var where player\_id = new\_record(i).player\_id;

end if;

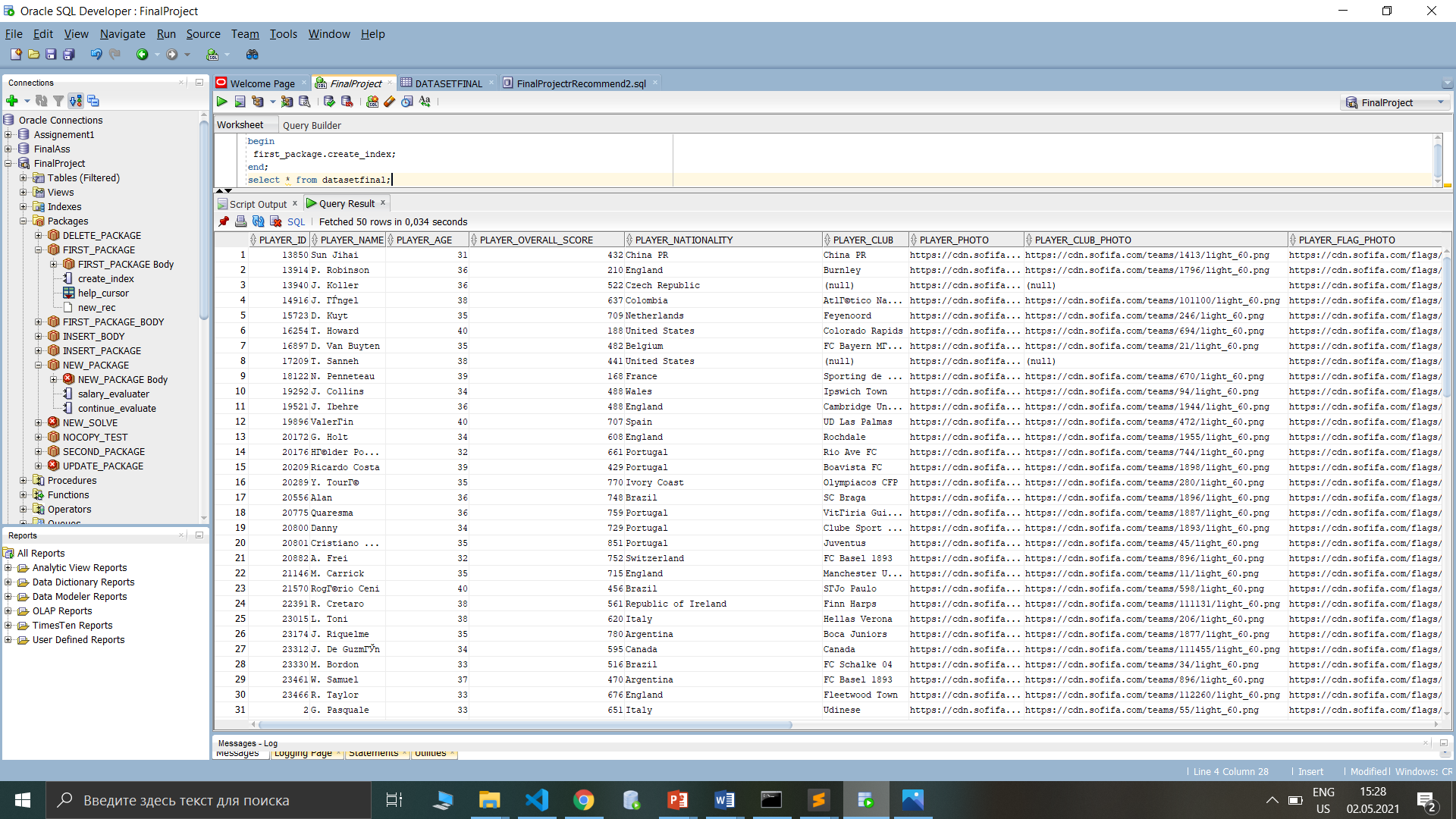
end loop;

end create\_index;

end first\_package;

first\_package contains create\_index procedure , help\_cursor and index by table of records which initialized as new\_record.

The main purpose of this package is computing first derived column called PLAYER\_OVERALL\_SCORE . This column is summation of two single columns which are player\_attacking and player\_skills . These columns are represented as OVA column in front part’s table .By making some population and initialization staff at the end we will be able to sum up all necessary columns and update our datasetfinals table in appropriate places.



--SECOND PACKAGE

create or replace package second\_package as

procedure create\_physic;

cursor physic\_cursor is select player\_id , player\_acceleration ,

player\_power, player\_strength from datasetfinal;

new\_col physic\_cursor%rowtype;

type t\_physictype is table of physic\_cursor%rowtype

index by binary\_integer;

physic\_record t\_physictype;

v\_var number;

end second\_package;

create or replace package body second\_package as

procedure create\_physic

as

begin

open physic\_cursor;

loop

fetch physic\_cursor into new\_col;

physic\_record(new\_col.player\_id):= new\_col;

exit when physic\_cursor%notfound;

end loop;

close physic\_cursor;

for i in physic\_record.first .. physic\_record.last

loop

if(physic\_record.exists(i))

then

v\_var := physic\_record(i).player\_acceleration +

physic\_record(i).player\_power + physic\_record(i).player\_strength;

update datasetfinal

set player\_physics\_overall = v\_var where player\_id = physic\_record(i).player\_id;

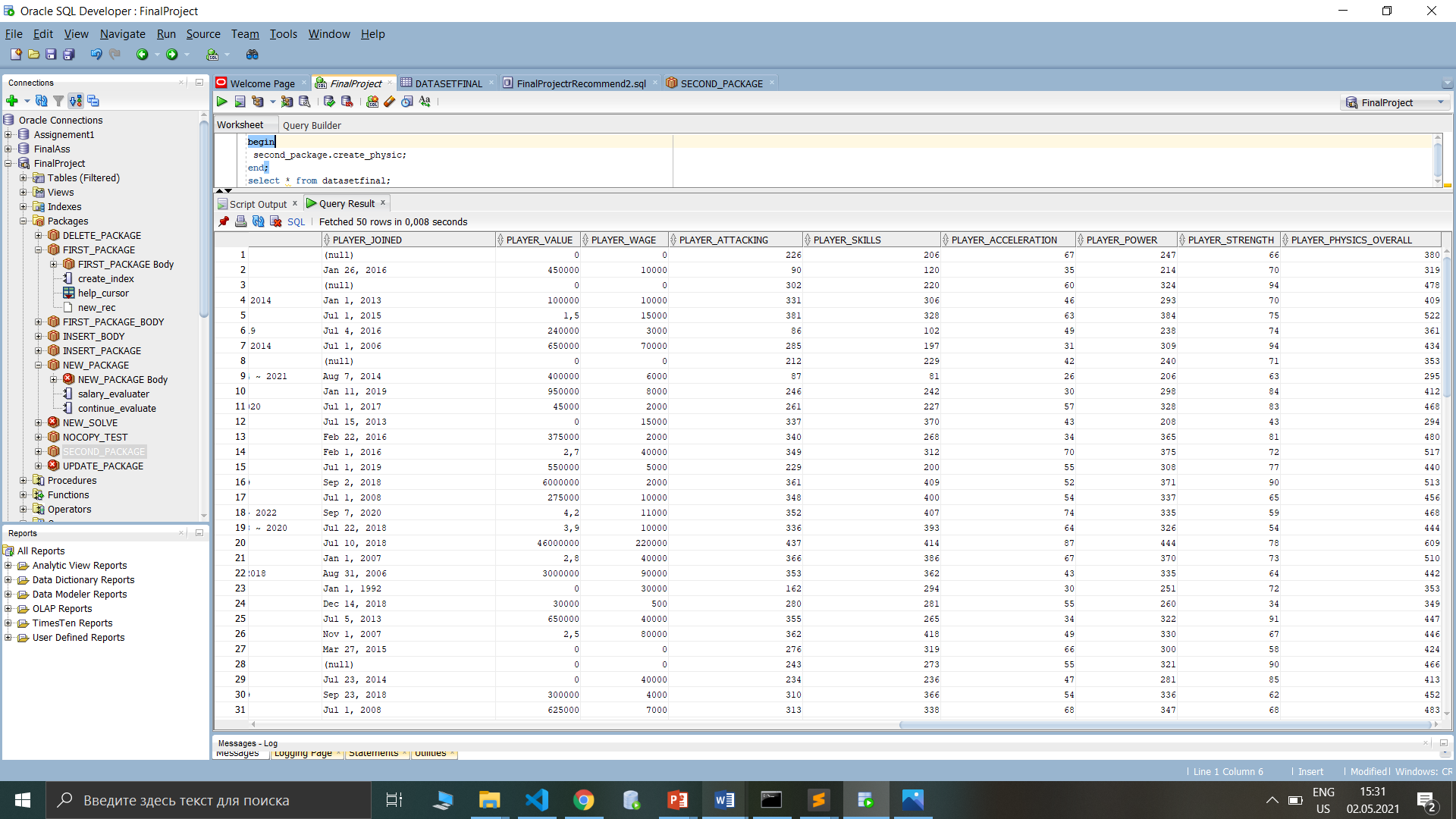
end if;

end loop;

end create\_physic;

end second\_package;

The second package uses the same technology as in the previous package except that here we compute PLAYER\_PHYSICS\_OVERALL which is the second derived column. The structure is the same at first glance. We use player\_acceleration , player\_power , player\_strength and sum up them into single derived column.



Of course, since we are filling derived columns we need some triggers to autocomplete each new changes.

--FIRST TRIGER

create or replace trigger after\_insert\_player

after insert on datasetfinal

begin

first\_package.create\_index;

second\_package.create\_physic;

end;

--SECOND TRIGGER

create or replace trigger after\_update\_player

after update of player\_id,

player\_name, player\_age,

player\_nationality, player\_club,

player\_contract, player\_joined,

player\_value, player\_wage,

player\_attacking, player\_skills,

player\_acceleration, player\_power,

player\_strength on datasetfinal

begin

first\_package.create\_index;

second\_package.create\_physic;

end;

These two triggers are responsible for fulfilling derived columns. Each time when we insert some new record or make changes they will fire and call necessary packages to sum up all appropriate columns.

In the next explanation you’ll see basic operations which gives ability to client manipulate with data by calling ready procedures. All of them contains DYNAMIC SQL so that by making insert/update/delete we’ve tried to use it .

--THIRD PACKAGE INSERT PACKAGE

create or replace package insert\_package as

procedure insert\_procedure(new\_id datasetfinal.player\_id%type, new\_name datasetfinal.player\_name%type,

new\_age datasetfinal.player\_age%type, new\_nation datasetfinal.player\_nationality%type, new\_club datasetfinal.player\_club%type,

new\_contract datasetfinal.player\_contract%type, new\_join datasetfinal.player\_joined%type, new\_value\_salary datasetfinal.player\_value%type,

new\_wage datasetfinal.player\_wage%type, new\_attacking datasetfinal.player\_attacking%type, new\_skills datasetfinal.player\_skills%type,

new\_acceleration datasetfinal.player\_acceleration%type, new\_power datasetfinal.player\_power%type, new\_strength datasetfinal.player\_strength%type);

end insert\_package;

create or replace package body insert\_package as

procedure insert\_procedure(new\_id datasetfinal.player\_id%type, new\_name datasetfinal.player\_name%type,

new\_age datasetfinal.player\_age%type, new\_nation datasetfinal.player\_nationality%type, new\_club datasetfinal.player\_club%type,

new\_contract datasetfinal.player\_contract%type, new\_join datasetfinal.player\_joined%type, new\_value\_salary datasetfinal.player\_value%type,

new\_wage datasetfinal.player\_wage%type, new\_attacking datasetfinal.player\_attacking%type, new\_skills datasetfinal.player\_skills%type,

new\_acceleration datasetfinal.player\_acceleration%type, new\_power datasetfinal.player\_power%type, new\_strength datasetfinal.player\_strength%type)

as

v\_execute varchar2(10000);

begin

v\_execute := 'INSERT INTO ' || 'DATASETFINAL ' || '(player\_id, player\_name, player\_age, player\_nationality,

player\_club, player\_contract, player\_joined, player\_value, player\_wage, player\_attacking, player\_skills,

player\_acceleration, player\_power, player\_strength)' || ' VALUES' || '(' || new\_id ||

' , ' || '''' || new\_name || '''' || ' , ' || new\_age || ' , ' || '''' || new\_nation || ''''

|| ' , ' || '''' || new\_club || '''' || ' , ' || '''' || new\_contract || '''' || ' , ' || '''' || new\_join ||

'''' || ' , ' || new\_value\_salary || ' , ' || new\_wage || ' , ' || new\_attacking || ' , ' || new\_skills ||

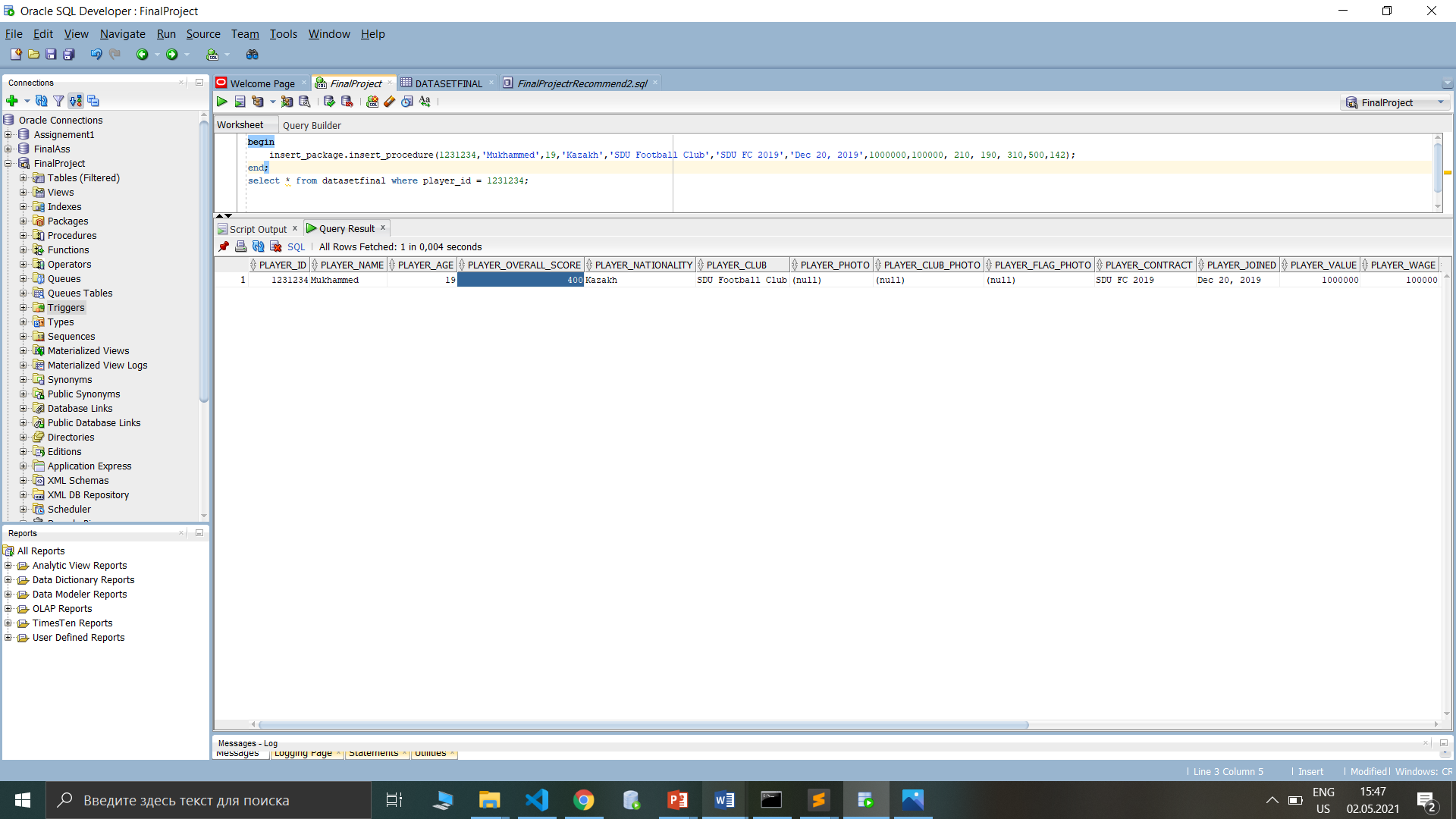
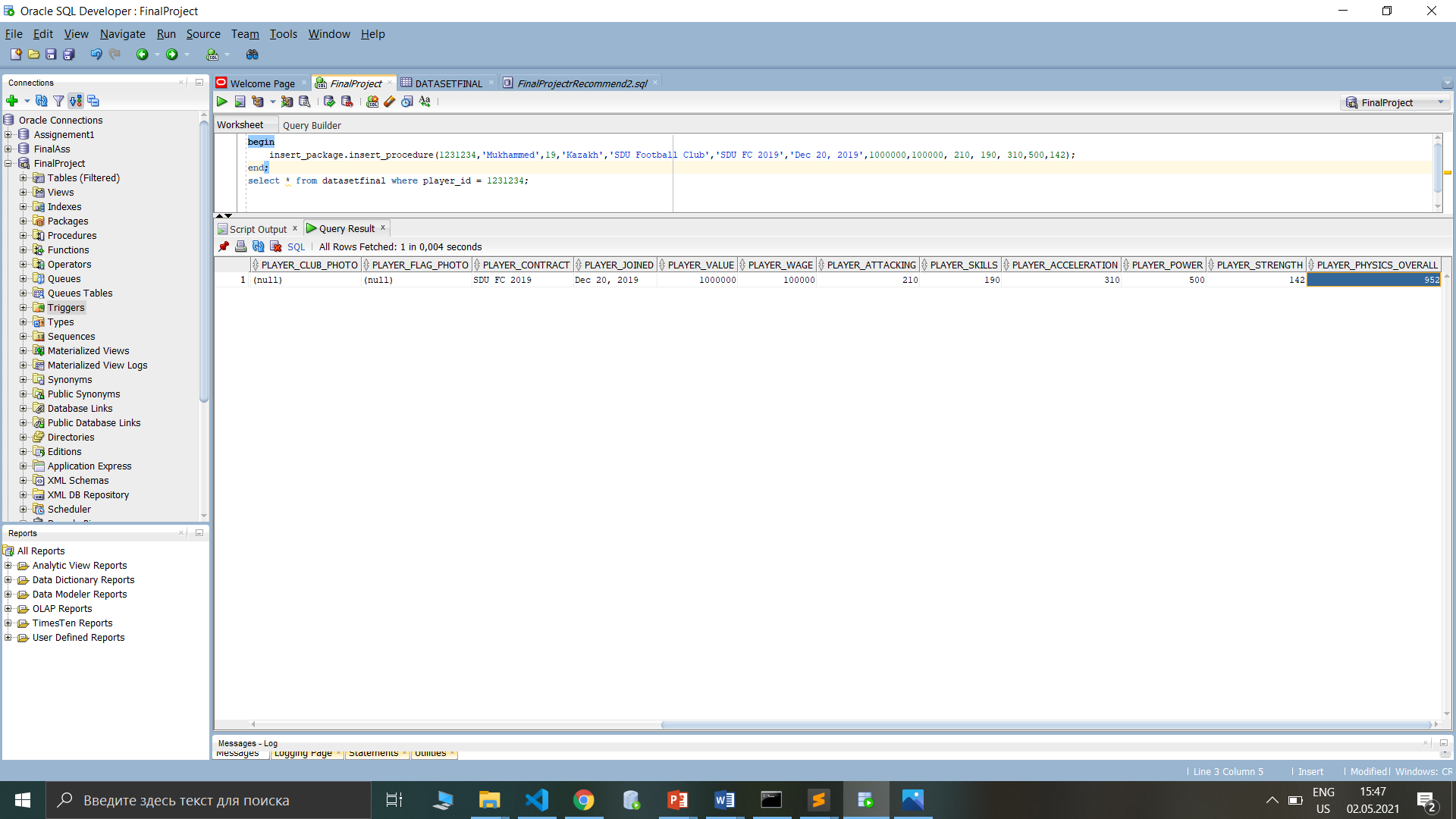
' , ' || new\_acceleration || ' , ' || new\_power || ' , ' || new\_strength ||')';

execute immediate v\_execute;

end insert\_procedure;

end insert\_package;

This package contains insert procedure which is done by DYNAMIC SQL statement. If client wants to insert some new data it is easy to call ready procedure from oracle. From back end side we put necessary parameters and call it.



So also as you can notice , we didn’t insert derived columns. Instead of it we just compute them automatically by two triggers . The first trigger is for insert procedure and second one is for update .

--FOURTH PACKAGE UPDATE PACKAGE

create or replace package update\_package as

procedure update\_procedure(new\_id datasetfinal.player\_id%type, new\_name datasetfinal.player\_name%type,

new\_age datasetfinal.player\_age%type, new\_nation datasetfinal.player\_nationality%type, new\_club datasetfinal.player\_club%type,

new\_contract datasetfinal.player\_contract%type, new\_join datasetfinal.player\_joined%type, new\_value\_salary datasetfinal.player\_value%type,

new\_wage datasetfinal.player\_wage%type, new\_attacking datasetfinal.player\_attacking%type, new\_skills datasetfinal.player\_skills%type,

new\_acceleration datasetfinal.player\_acceleration%type, new\_power datasetfinal.player\_power%type, new\_strength datasetfinal.player\_strength%type);

end update\_package;

create or replace package body update\_package as

procedure update\_procedure(new\_id datasetfinal.player\_id%type, new\_name datasetfinal.player\_name%type,

new\_age datasetfinal.player\_age%type, new\_nation datasetfinal.player\_nationality%type, new\_club datasetfinal.player\_club%type,

new\_contract datasetfinal.player\_contract%type, new\_join datasetfinal.player\_joined%type, new\_value\_salary datasetfinal.player\_value%type,

new\_wage datasetfinal.player\_wage%type, new\_attacking datasetfinal.player\_attacking%type, new\_skills datasetfinal.player\_skills%type,

new\_acceleration datasetfinal.player\_acceleration%type, new\_power datasetfinal.player\_power%type, new\_strength datasetfinal.player\_strength%type)

as

v\_string\_store varchar(10000);

begin

v\_string\_store := 'UPDATE ' || 'DATASETFINAL ' || 'SET ' || 'player\_name' || ' = ' || '''' || new\_name || '''' || ' , '

|| 'player\_age' || ' = ' || new\_age || ' , ' || 'player\_nationality' || ' = ' || '''' || new\_nation || '''' || ' , ' ||

'player\_club' || ' = ' || '''' || new\_club || '''' || ' , ' || 'player\_contract' || ' = ' || '''' || new\_contract || '''' || ' , ' ||

'player\_joined' || ' = ' || '''' || new\_join || '''' || ' , ' || 'player\_value' || ' = ' || new\_value\_salary || ' , ' ||

'player\_wage' || ' = ' || new\_wage || ' , ' || 'player\_attacking' || ' = ' || new\_attacking || ' , ' ||

'player\_skills' || ' = ' || new\_skills || ' , ' || 'player\_acceleration' || ' = ' || new\_acceleration || ' , ' ||

'player\_power' || ' = ' || new\_power || ' , ' || 'player\_strength' || ' = ' || new\_strength || ' WHERE ' || 'player\_id' || ' = ' || new\_id;

execute immediate v\_string\_store;

end update\_procedure;

end update\_package;

UPDATE PACKAGE has the same structure as insert package. It also uses update procedure which includes DYNAMIC SQL .

--FIFTH PACKAGE DELETE PACKAGE

create or replace package delete\_package

as

procedure delete\_procedure(p\_del\_id datasetfinal.player\_id%type);

end delete\_package;

create or replace package body delete\_package as

procedure delete\_procedure(p\_del\_id datasetfinal.player\_id%type)

is

v\_new\_st varchar(10000);

begin

v\_new\_st := 'DELETE ' || 'FROM' || ' DATASETFINAL ' || 'WHERE' || ' player\_id ' || ' = ' || p\_del\_id;

execute immediate v\_new\_st;

end delete\_procedure;

end delete\_package;

DELETE PACKAGE CONTAINS delete procedure which takes id as parameter and executes some delete operation by DYNAMIC SQL;

--NEW SOLVE PACKAGE FOR FIXING PLAYER\_VALUE AND PLAYER\_WAGE

create or replace package new\_solve as

cursor change\_value

is select player\_id, player\_value

from datasetfinal

where player\_value like '%,%';

cursor change\_wage

is select player\_id , player\_wage

from datasetfinal

where player\_wage like '%,%';

procedure change\_procedure;

end new\_solve;

create or replace package body new\_solve as

procedure change\_procedure is

begin

for temp\_cursor in change\_value loop

update datasetfinal set datasetfinal.player\_value = temp\_cursor.player\_value \* 1000000

where datasetfinal.player\_id = temp\_cursor.player\_id;

end loop;

for temp\_cursor in change\_wage loop

update datasetfinal set datasetfinal.player\_wage = temp\_cursor.player\_wage \* 1000000

where datasetfinal.player\_id = temp\_cursor.player\_id;

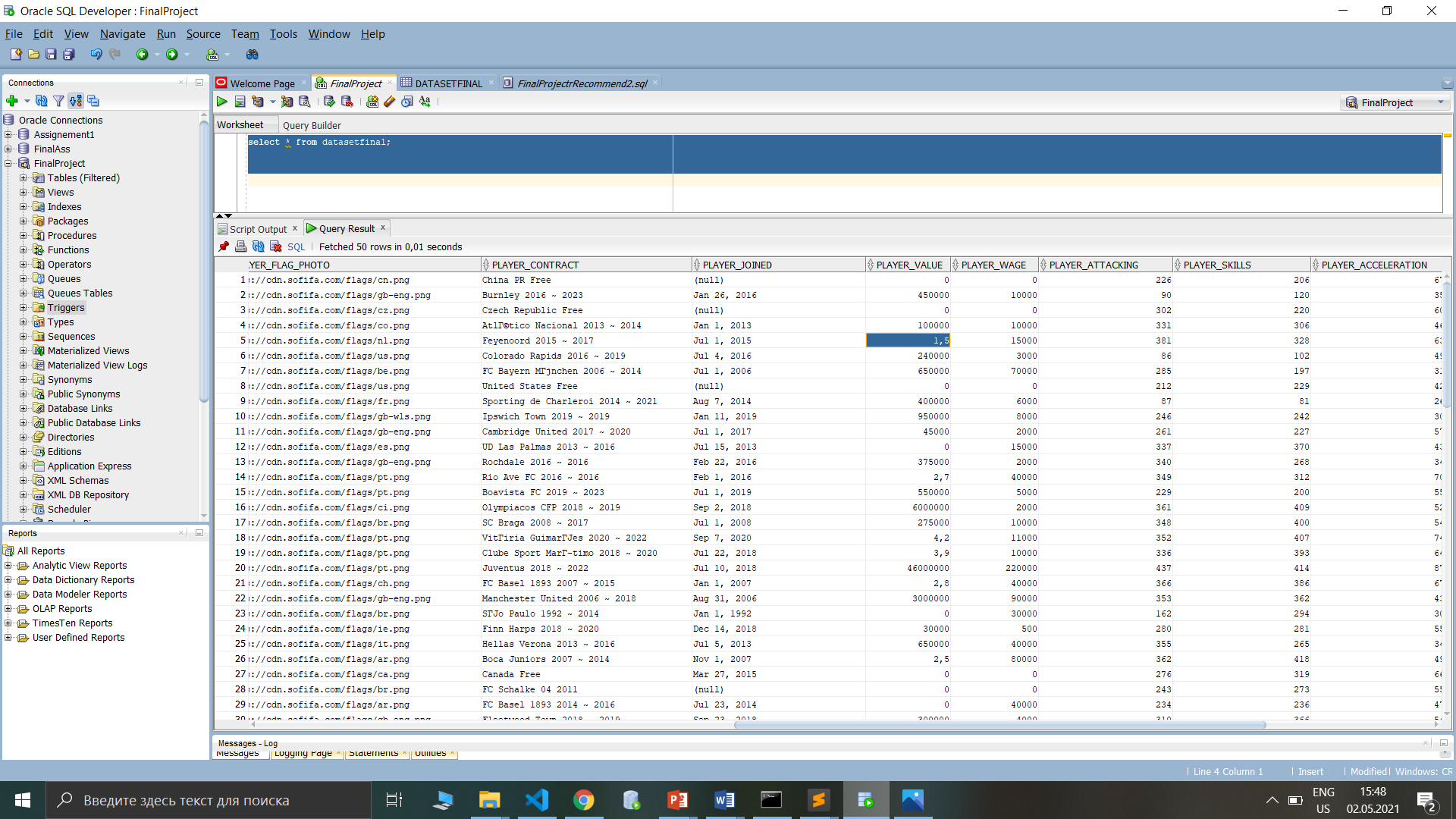
end loop;

end change\_procedure;

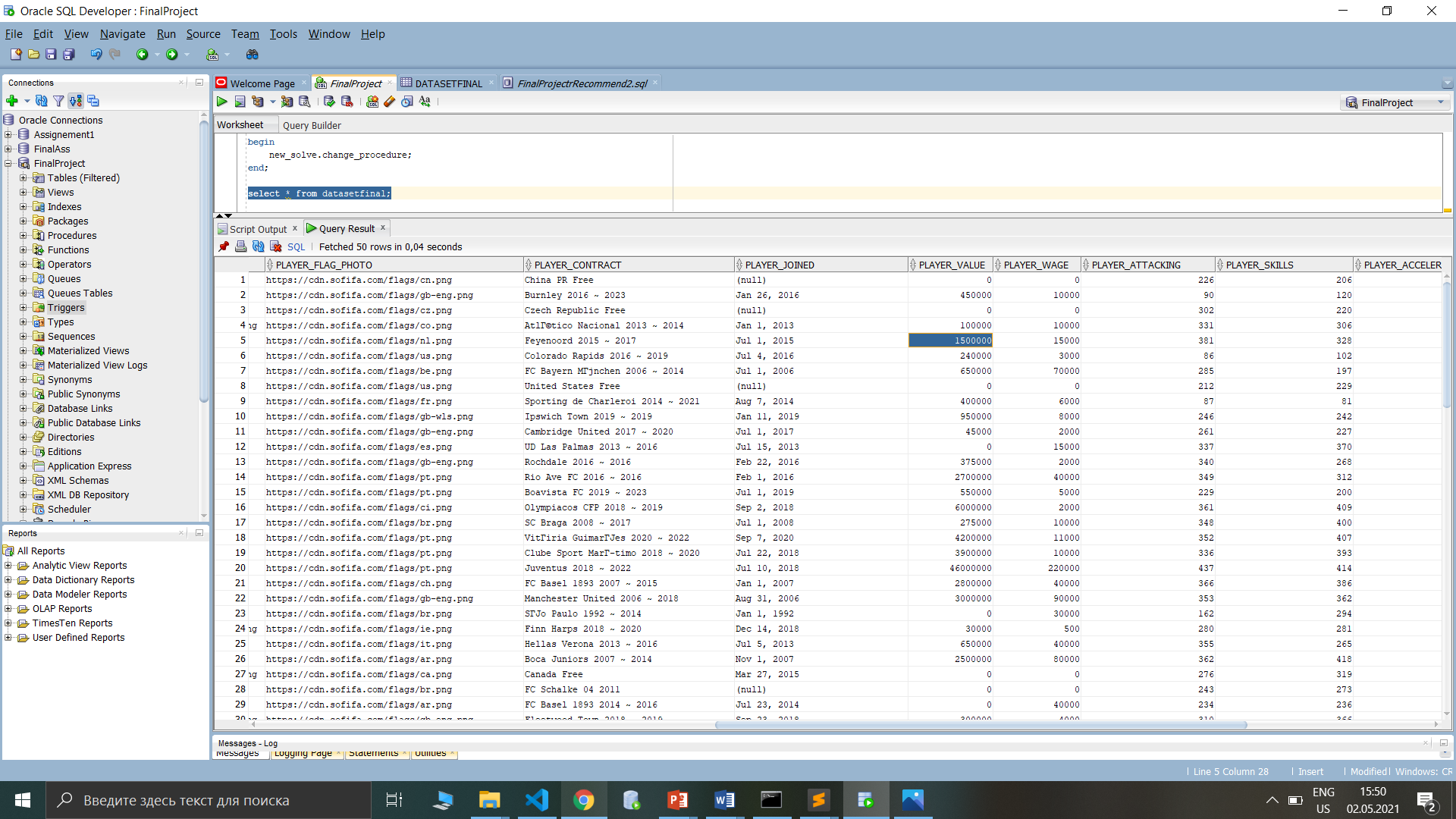
end new\_solve;

PLAYER\_VALUE is the cost of player at certain football club. And PLAYER\_WAGE is salary of player per each months. Our dataset contains integers and floats in these columns , represented by number data type. In addition , we noticed a little bug which removes zero digits at the end of floating point numbers. To prevent some errors and inconveniences , we’ve decided to make some special package with procedure , so we can see the clear data in our front part. Generally , the feature of this procedure is fixing our special bug. By using cursors and DML statements we multiplied cut zeroes to floating point numbers. Actually, procedure is just for our database part for consistency and clarification . Besides , client will be confused unless there is any clear data.

BEFORE NEW SOLVE



AFTER NEW SOLVE



--RECOMMENDATION PROCEDURE TOP TEN PLAYERS BONUS PART

Create table recommendation\_proc a s select \* from datasetfinal;

Delete from datasetfinal;

create or replace procedure recommendation\_proc

is

new\_var number;

count\_var number := 0;

cursor new\_recommend is select \* from datasetfinal

order by (player\_overall\_score + player\_physics\_overall) desc;

begin

delete from recommendation\_table;

for i in new\_recommend loop

insert into recommendation\_table (player\_id, player\_name, player\_age, player\_nationality,player\_club, player\_contract, player\_joined, player\_value, player\_wage, player\_attacking, player\_skills,player\_acceleration, player\_power, player\_strength,

player\_overall\_score, player\_physics\_overall)

values(i.player\_id,i.player\_name, i.player\_age, i.player\_nationality,

i.player\_club, i.player\_contract, i.player\_joined, i.player\_value, i.player\_wage, i.player\_attacking, i.player\_skills,

i.player\_acceleration, i.player\_power, i.player\_strength,i.player\_overall\_score, i.player\_physics\_overall);

count\_var := count\_var + 1;

exit when count\_var > 9;

end loop;

end;

RECOMMENDATION is essential part of our project. Because it is just for client convenience. We created special table for this one. Working principle is very easy. We select top ten players which have the highest points by computing PLAYER\_OVERALL\_SCORE and PLAYER\_OVERALL\_PHYSICS .Also it is already included in new\_recommend cursor. Initially we delete all previous records from recommendation\_table , then we insert data into recommendation\_table by fetching records from cursor.

--THIRD TRIGGER FOR\_RECOMMEND

create or replace trigger for\_recommend

after insert or update or delete on datasetfinal

declare

v\_var varchar2(5);

begin

if inserting then

recommendation\_proc;

elsif updating then

recommendation\_proc;

elsif deleting then

recommendation\_proc;

else

v\_var := 'no';

end if;

end;

For\_recommend trigger is continuation of our bonus task. It is dependent on actions that will be done by manipulating datasetfinals table . So if you perform DML statement , it immediately changes records inside of recommendation\_table which is logically correct.

