Ex No : 6 **QUERY TUNING**

Date : 31.03.2020

**Aim :**

To analyze the different ways to optimize the queries for minimum impact on database performance.

**COMMANDS:**

### Only use lowercase letters, numbers and underscores

Don’t use dots, spaces or dashes in database, schema, table or column names. Dots are for identifying objects, usually in the database .schema .table. column pattern. Having dots in names of objects will cause confusion. Queries are harder to write if you use capital letters in table or column names.

Eg: Use the query as:

SELECT ORD\_NO FROM order\_table;

Instead of:

SELECT ‘ORD NO’ FROM order\_table;

### Use simple, descriptive column names

Descriptive names make it easier for others to understand the schema, which is vital to maintaining efficiency as the team grows. Don’t use ambiguous names for polymorphic data. Don’t prefix column names with the name of the containing table.

Eg: Use the query as:

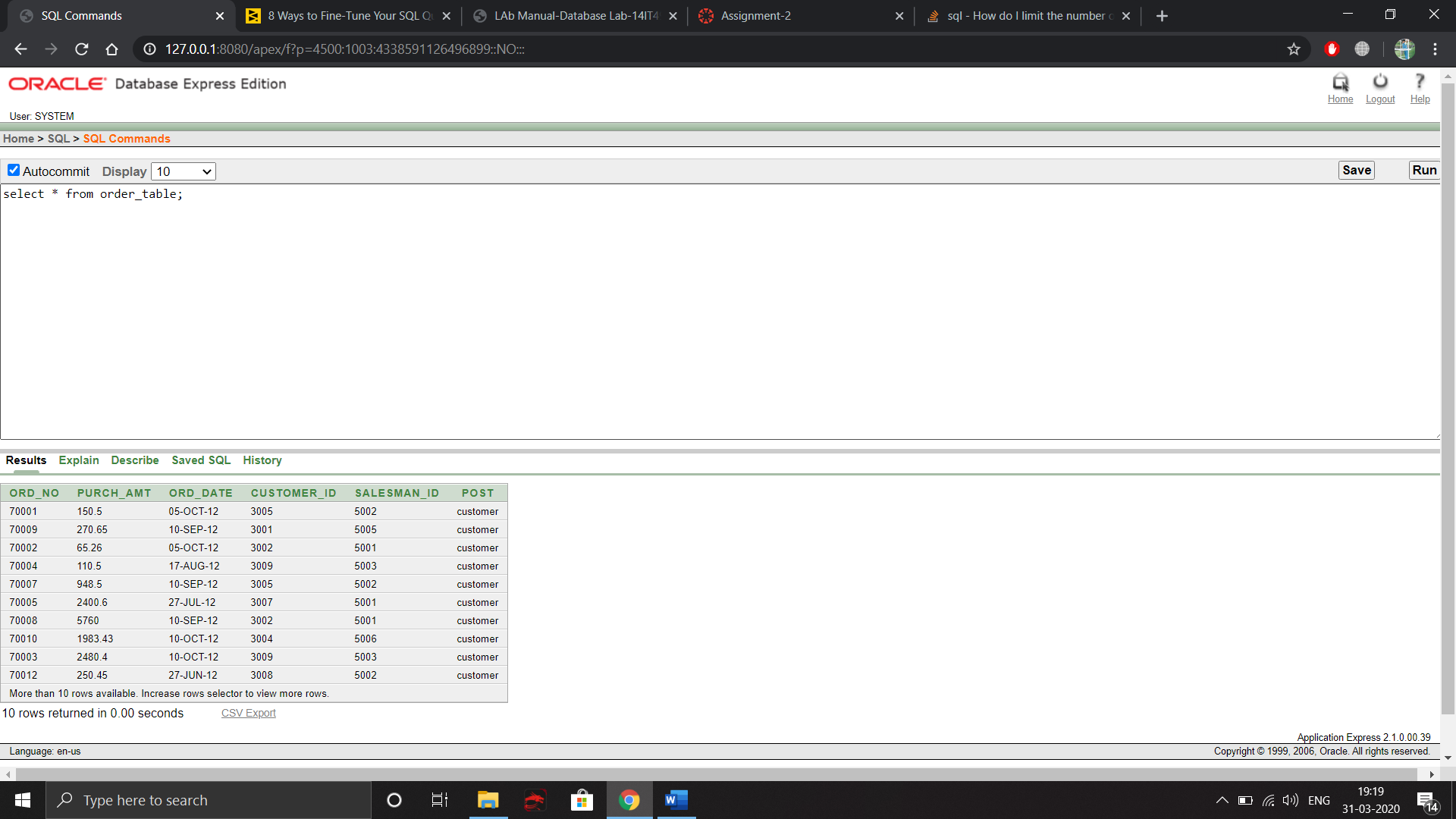
SELECT SUM(purch\_amt) FROM order\_table;

Instead of:

SELECT SUM(purch\_amt) AS vote\_count

FROM order\_table

WHERE post=’customer’;



### Use simple, descriptive table names and have an integer primary key

If the table name is made up of multiple words, use underscores to separate the words. Don’t prefix tables to imply a schema. This kind of primary key makes certain analyses much easier, like selecting only the first row of a group. Avoid multi-column primary keys. Use an integer primary key, a multi-column unique constraint, and several single-column indexes instead.

And if an import job ever duplicates data, this key will be a life-saver because you’ll be able to delete specific rows.

Eg: Use the query as

DELETE FROM Faculty

WHERE facuty\_id IN (SELECT …….) AS duplicate\_ids;

1. Use DECODE to avoid the scanning of same rows or joining the same table repetitively. DECODE can also be made used in place of GROUP BY or ORDER BY clause.

**For example**, Use the query as

SELECT Student\_id,Contact\_no FROM Student

WHERE Student\_name LIKE 'Re%'

and Dept= 'ECE';

Instead of

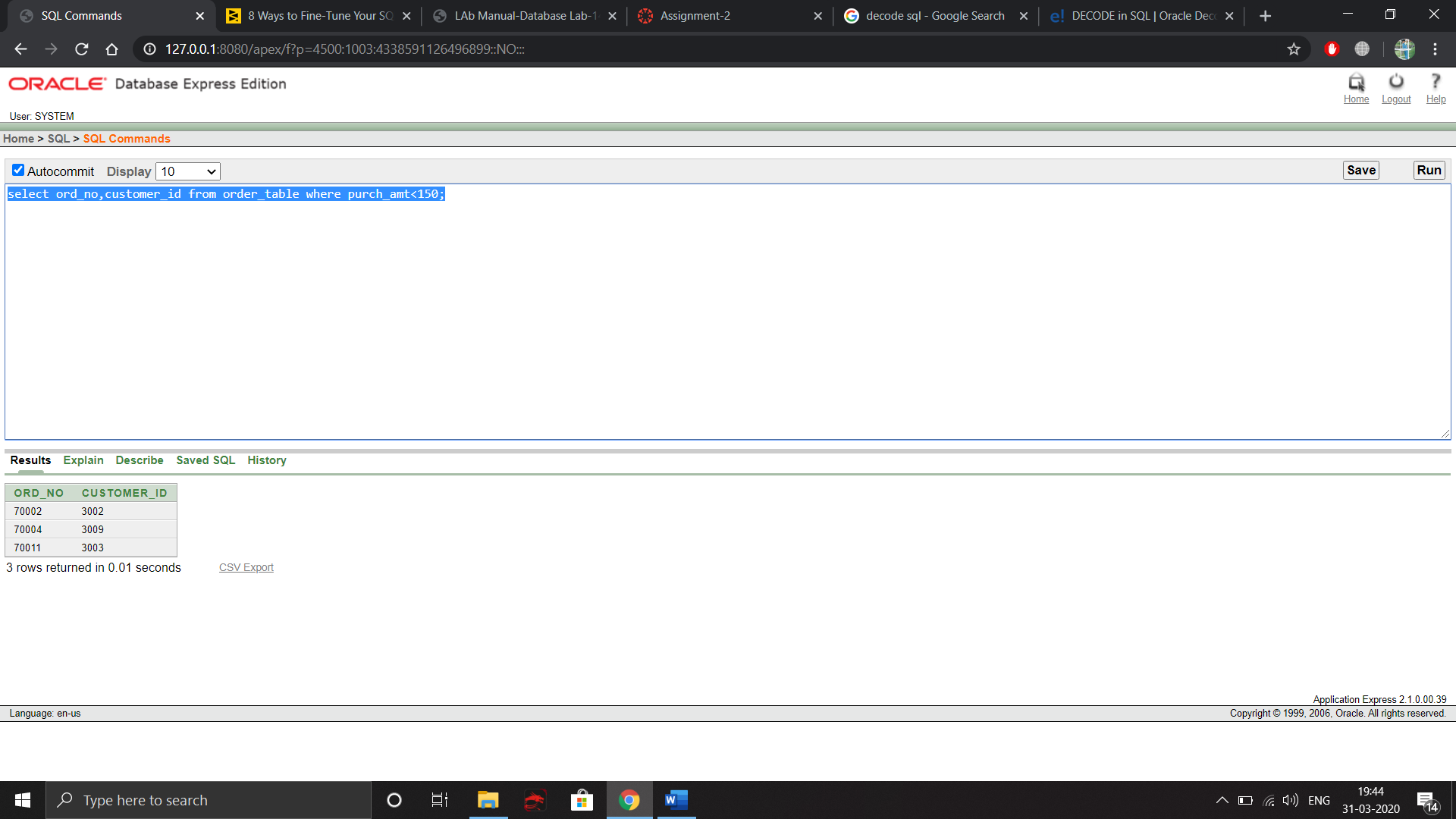
SELECT DECODE(Dept,'ECE',Student\_id ,NULL) Student\_id FROM Student

WHERE Student\_name LIKE 'Re%';

1. Use non-column expression on one side of the query because it will be processed earlier.

**For example** ,Use the query as

select ord\_no,customer\_id from order\_table where purch\_amt<150;



Instead of

select ord\_no,customer\_id from order\_table where purch\_amt+100<50;

1. Be careful while using conditions in WHERE clause. The statements mentioned below does not mean the same query, just a small in the operator can change the whole query.

**For example,** Use the query as

select ord\_no,customer\_id from order\_table where purch\_amt<50;

Instead of,

select ord\_no,customer\_id from order\_table where purch\_amt!=50;

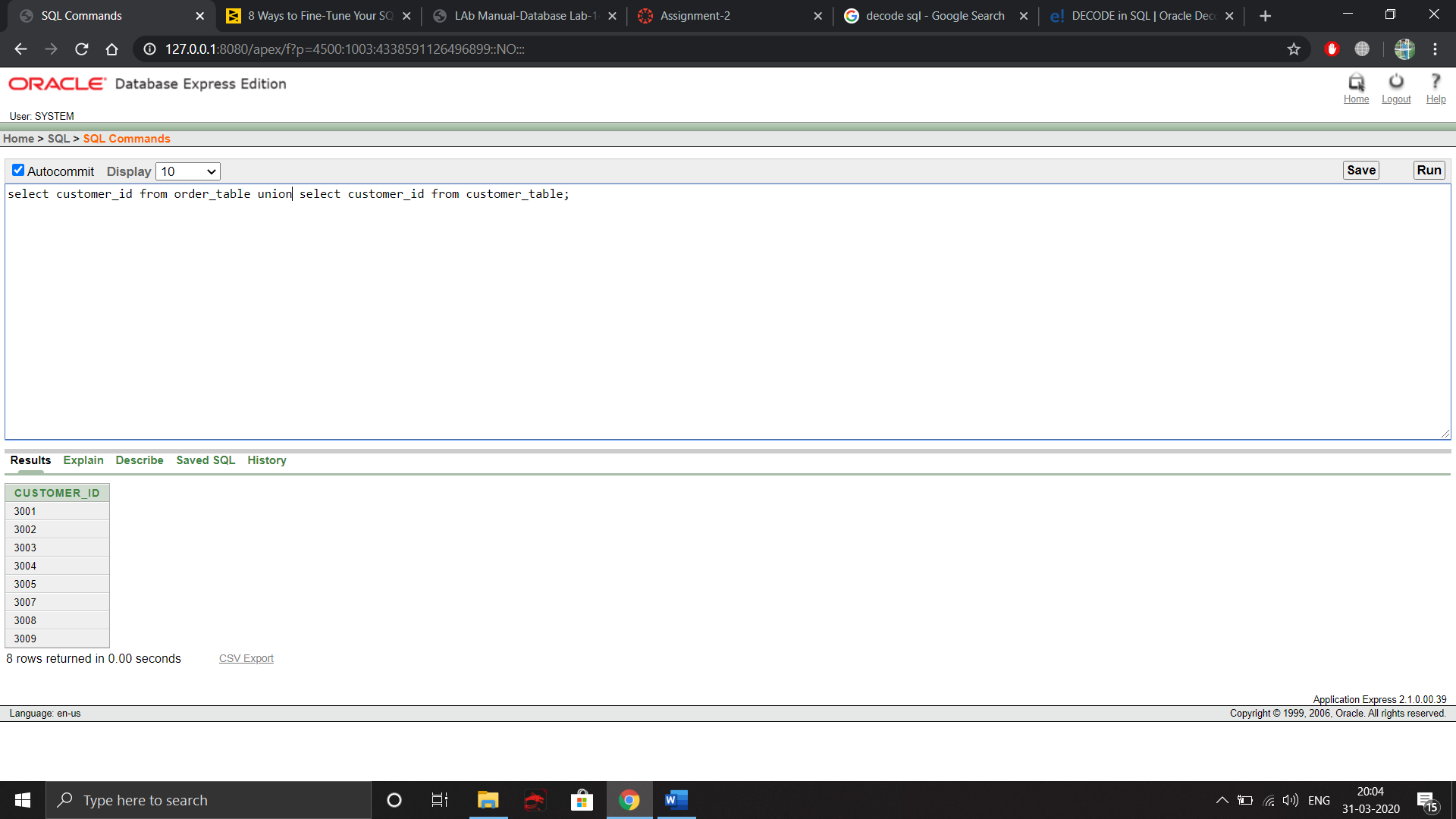
1. Try to use UNION ALL in place of UNION. This saves the time and also it displays the records in a order which can be sorted easily. This also reduces execution time.

**For example** , Use the query as

select customer\_id from order\_table union all select customer\_id from customer\_table;

Instead of,

select customer\_id from order\_table union select customer\_id from customer\_table;



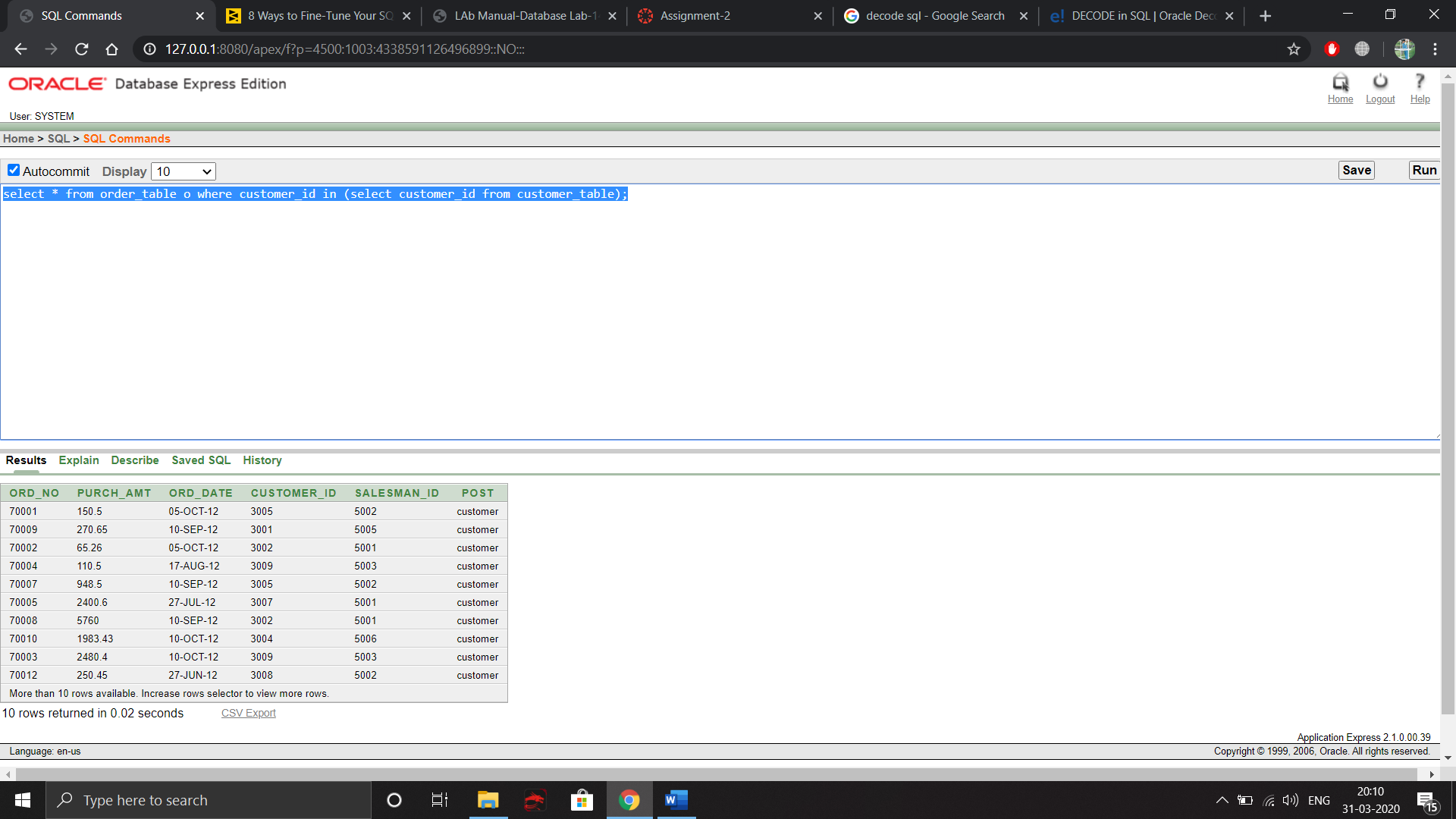
1. Use operator EXISTS, IN and table joins appropriately in your query.  
   **a)**Usually IN has the slowest performance.  
   **b)**IN is efficient when most of the filter criteria is in the sub-query.  
   **c)**EXISTS is efficient when most of the filter criteria is in the main query.

**For example**, Use the query as

select \* from order\_table o where exists( select \* from customer\_table c where c.customer\_id=o.customer\_id);

Instead of,

select \* from order\_table o where customer\_id in (select customer\_id from customer\_table);



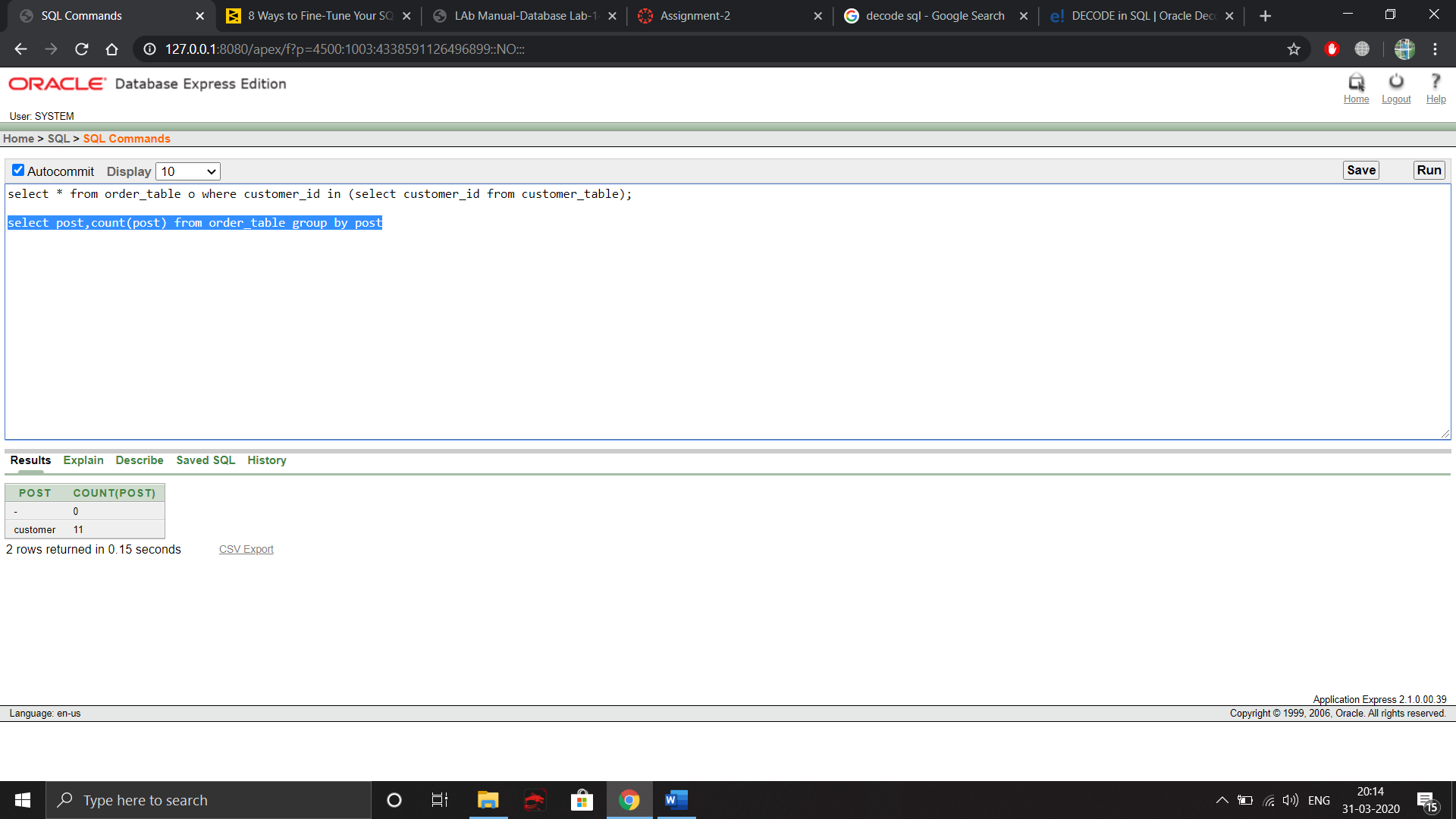
1. HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.

**For example**, Use the query as

select post,count(post) from order\_table where post=’customer’ group by post

Instead of,

select post,count(post) from order\_table group by post having post=’customer’



1. **Implicit aggregation:**

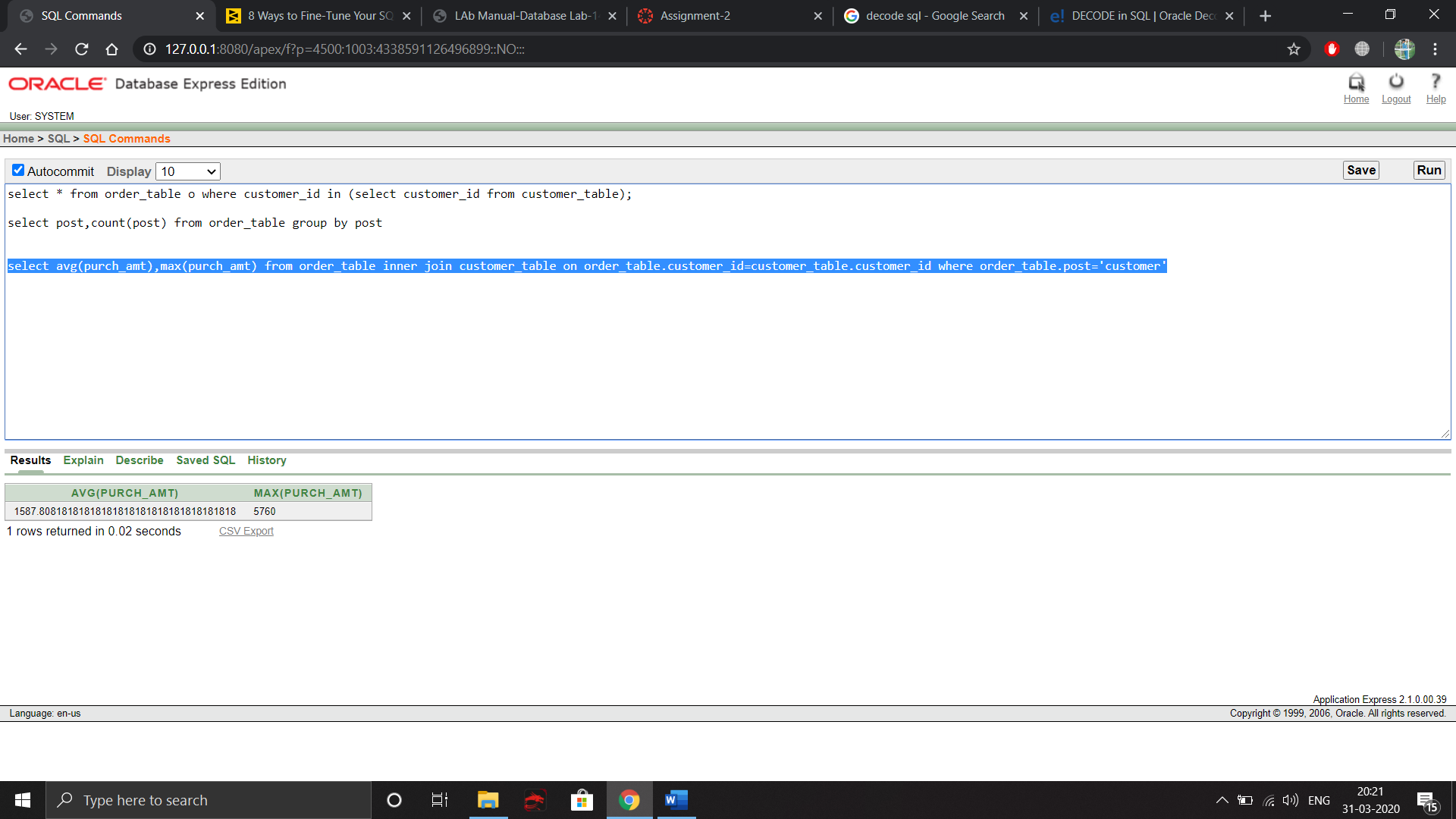
There is one case where you can take an aggregation without using a group by.

When you are aggregating the full table there is an implied group by.

This is known as the in SQL standards documentation.

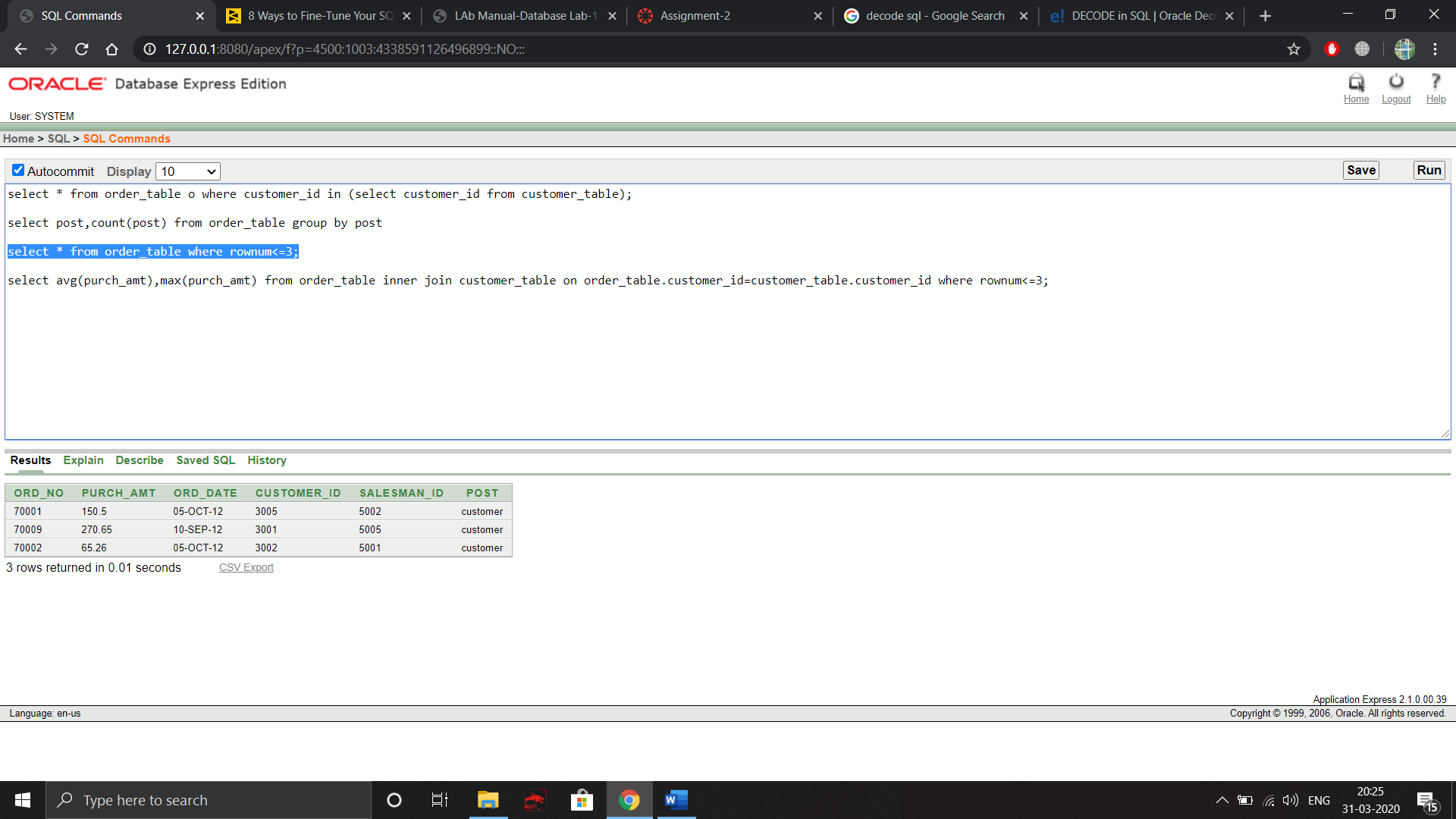
**For example,** Use the query as

select avg(purch\_amt),max(purch\_amt) from order\_table inner join customer\_table on order\_table.customer\_id=customer\_table.customer\_id where order\_table.post='customer'



1. Before running a query for the first time, ensure the results will be desirable and meaningful by using a **LIMIT** statement. The LIMIT statement returns only the number of records specified. Using a **LIMIT** statement prevents taxing the production database with a large query, only to find out the query needs editing or refinement.

**For example**,

select \* from order\_table where rownum<=3;

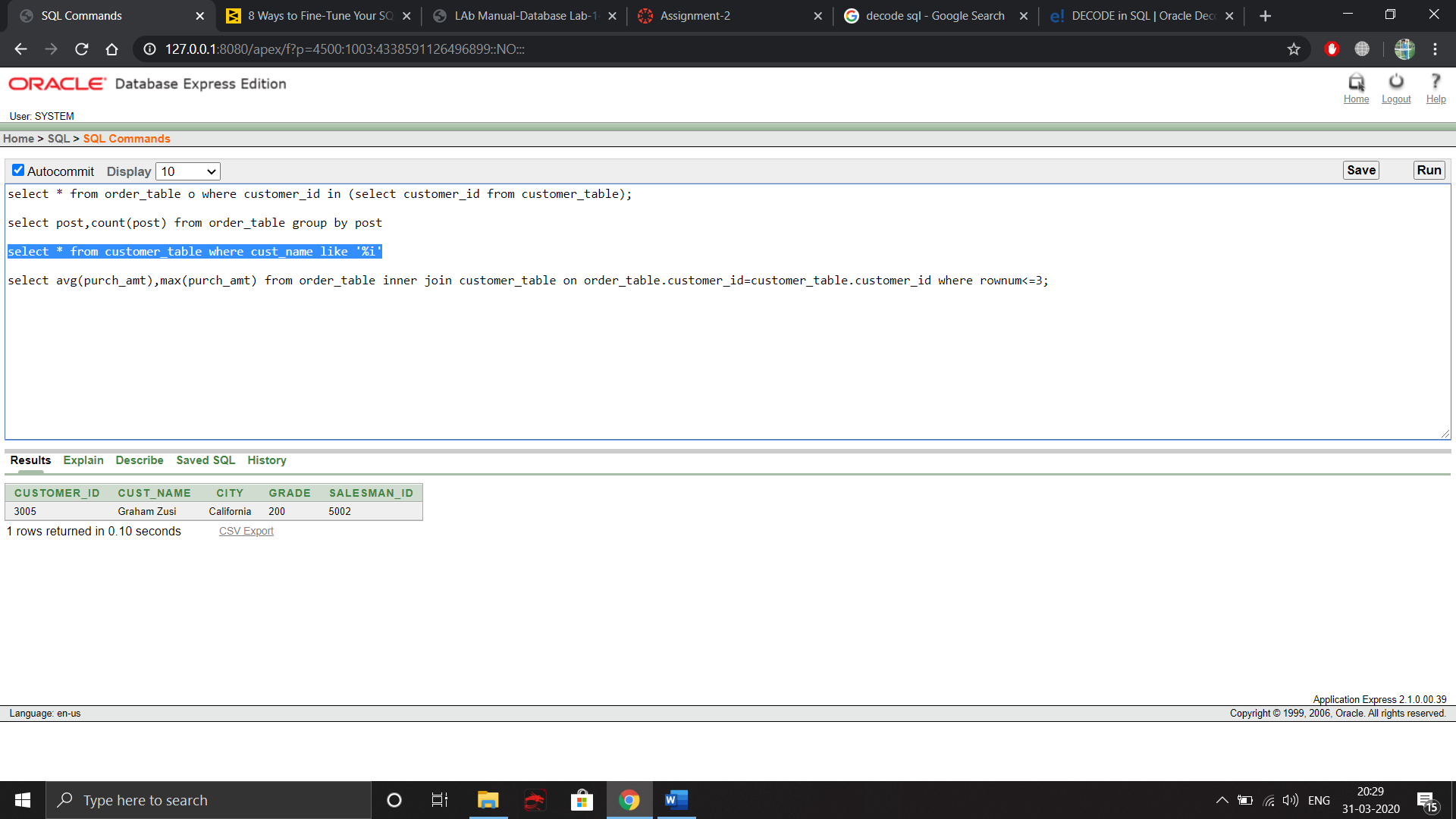
1. When searching plaintext data, such as cities or names, wildcards create the widest search possible. However, the widest search is also the most inefficient search.When a leading wildcard is used, especially in combination with an ending wildcard, the database is tasked with searching all records for a match anywhere within the selected field.

**For example,** Use the query as

select \* from customer\_table where cust\_name like '%I'

Instead of,

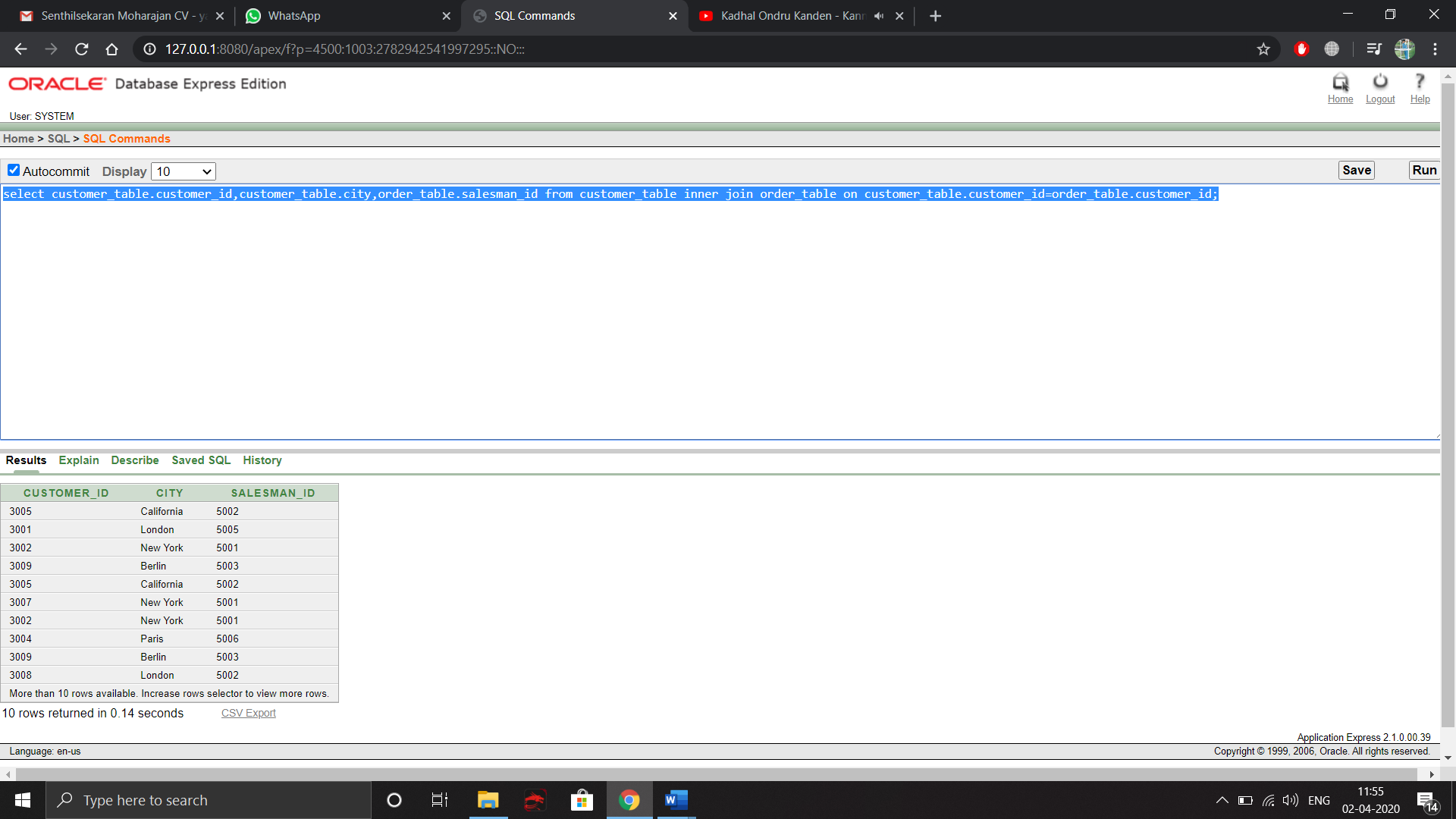
select \* from customer\_table where cust\_name like '%i'



1. Some SQL developers prefer to make joins with **WHERE** clauses,this type of join creates a Cartesian Join, also called a Cartesian Product or **CROSS JOIN**. In a Cartesian Join, all possible combinations of the variables are created.To prevent creating a Cartesian Join, use **INNER JOIN** instead.Using inner join will reduce the execution time for your database.

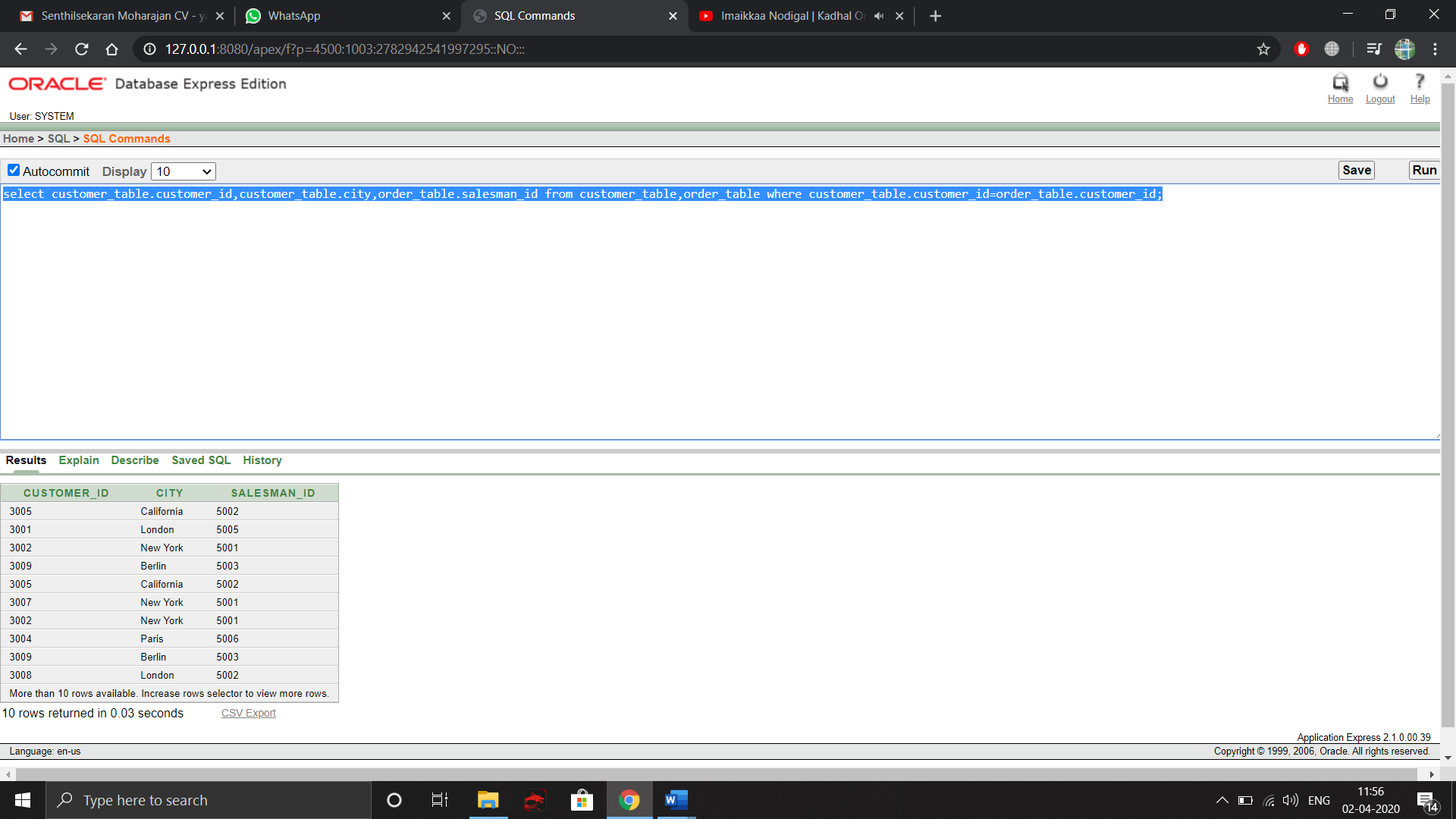
**For example**, Use the query as

select customer\_table.customer\_id,customer\_table.city,order\_table.salesman\_id from customer\_table inner join order\_table on customer\_table.customer\_id=order\_table.customer\_id;



Instead of,

select customer\_table.customer\_id,customer\_table.city,order\_table.salesman\_id from customer\_table,order\_table where customer\_table.customer\_id=order\_table.customer\_id;



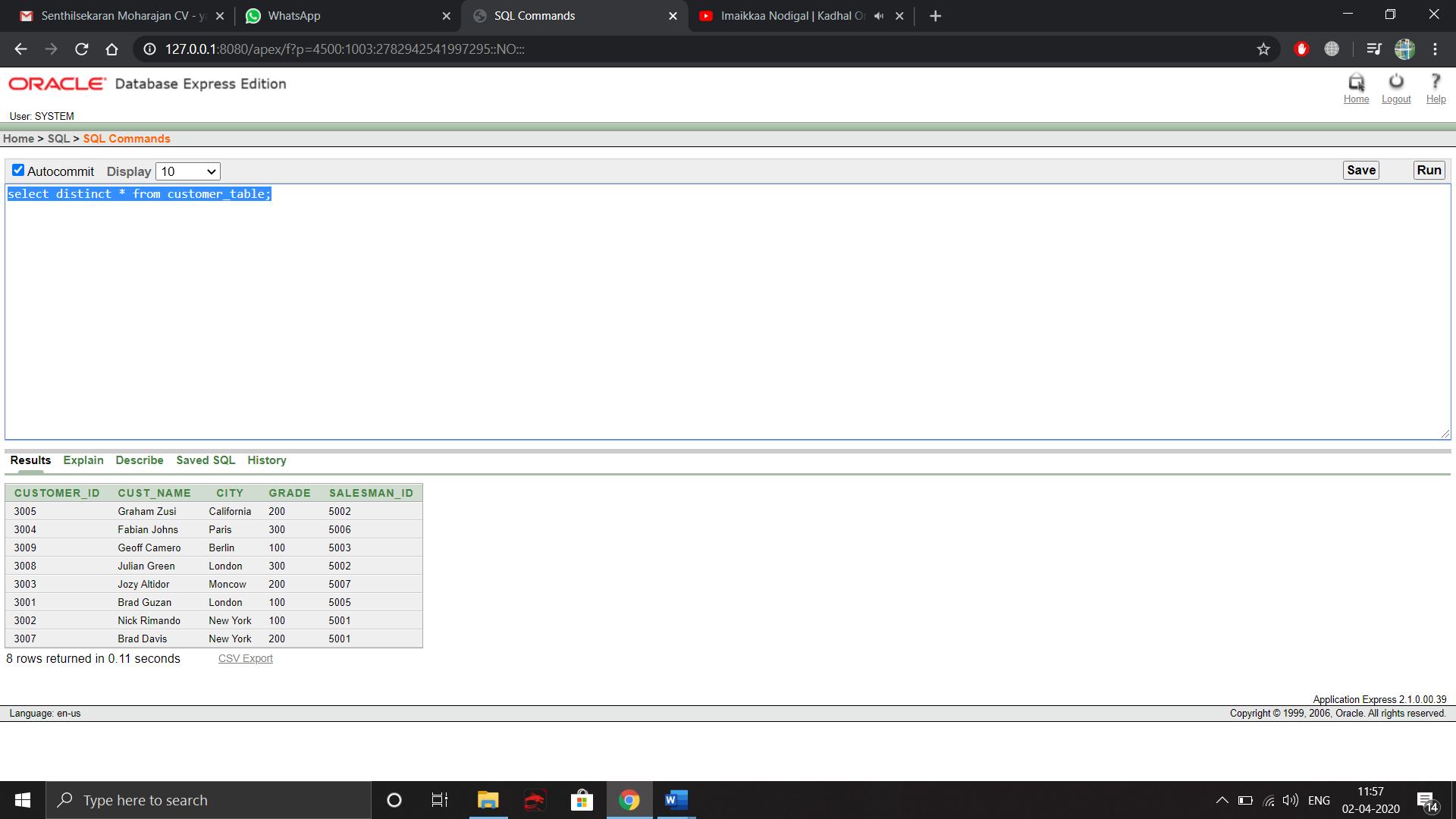
1. **SELECT DISTINCT** is a handy way to remove duplicates from a query. **SELECT DISTINCT** works by **GROUP**ing all fields in the query to create distinct results. To accomplish this goal however, a large amount of processing power is required. Additionally, data may be grouped [to the point of being inaccurate](https://www.sisense.com/blog/understanding-simpsons-paradox-to-avoid-faulty-conclusions/). To avoid using **SELECT DISTINCT**, select more fields to create unique results. By adding more fields, unduplicated records were returned without using SELECT DISTINCT. The database does not have to group any fields, and the number of records is accurate.

**For example,** Use the query as

select \* from customer\_table;

Instead of,

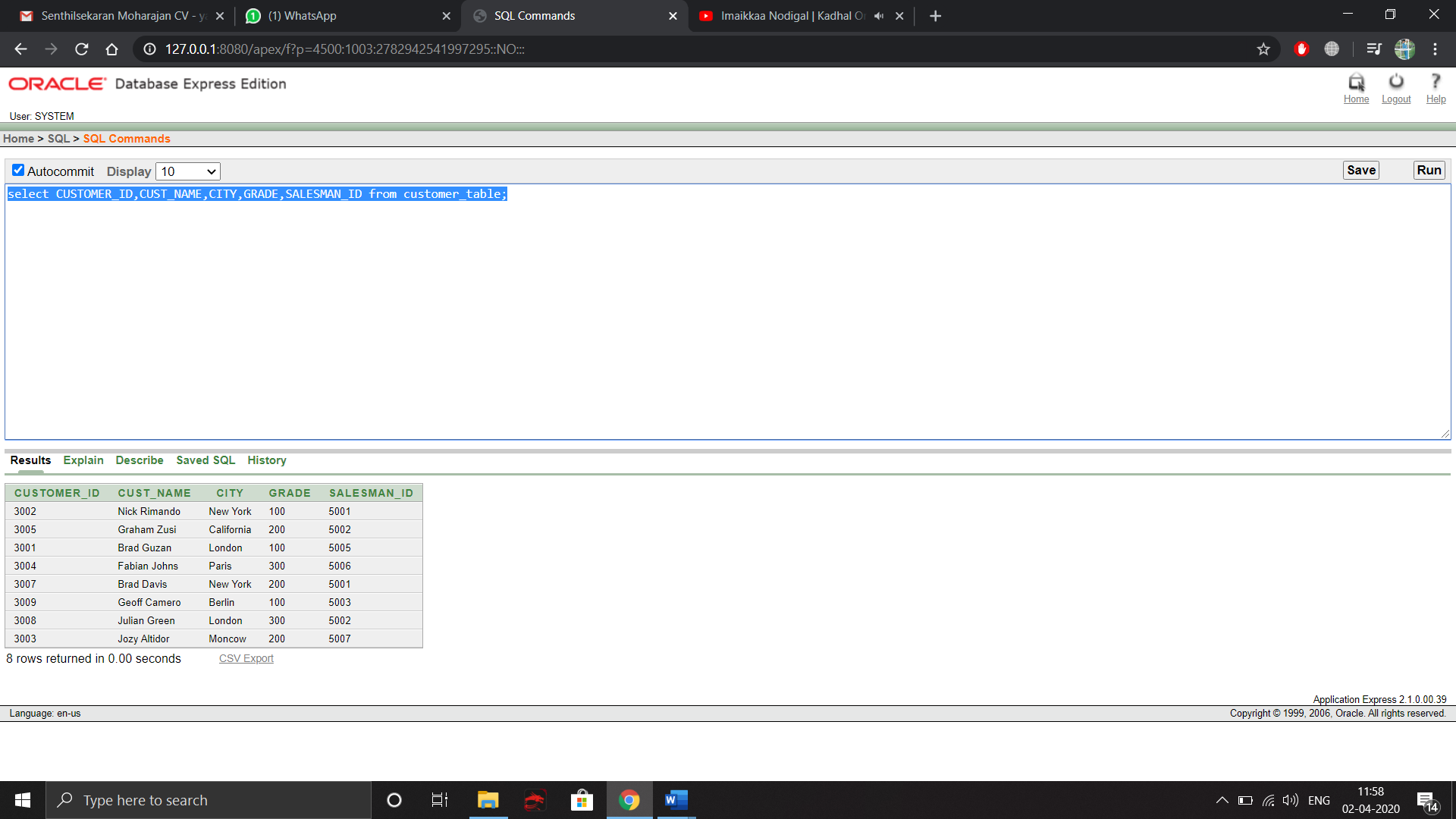
select distinct \* from customer\_table;



1. When running exploratory queries, many SQL developers use **SELECT \***  as a [shorthand to query](https://www.sisense.com/blog/sql-symbol-cheatsheet/) all available data from a table. However, if a table has many fields and many rows, this taxes database resources by querying a lot of unnecessary data.Using the **SELECT** statement will point the database to querying only the data you need to meet the business requirements.

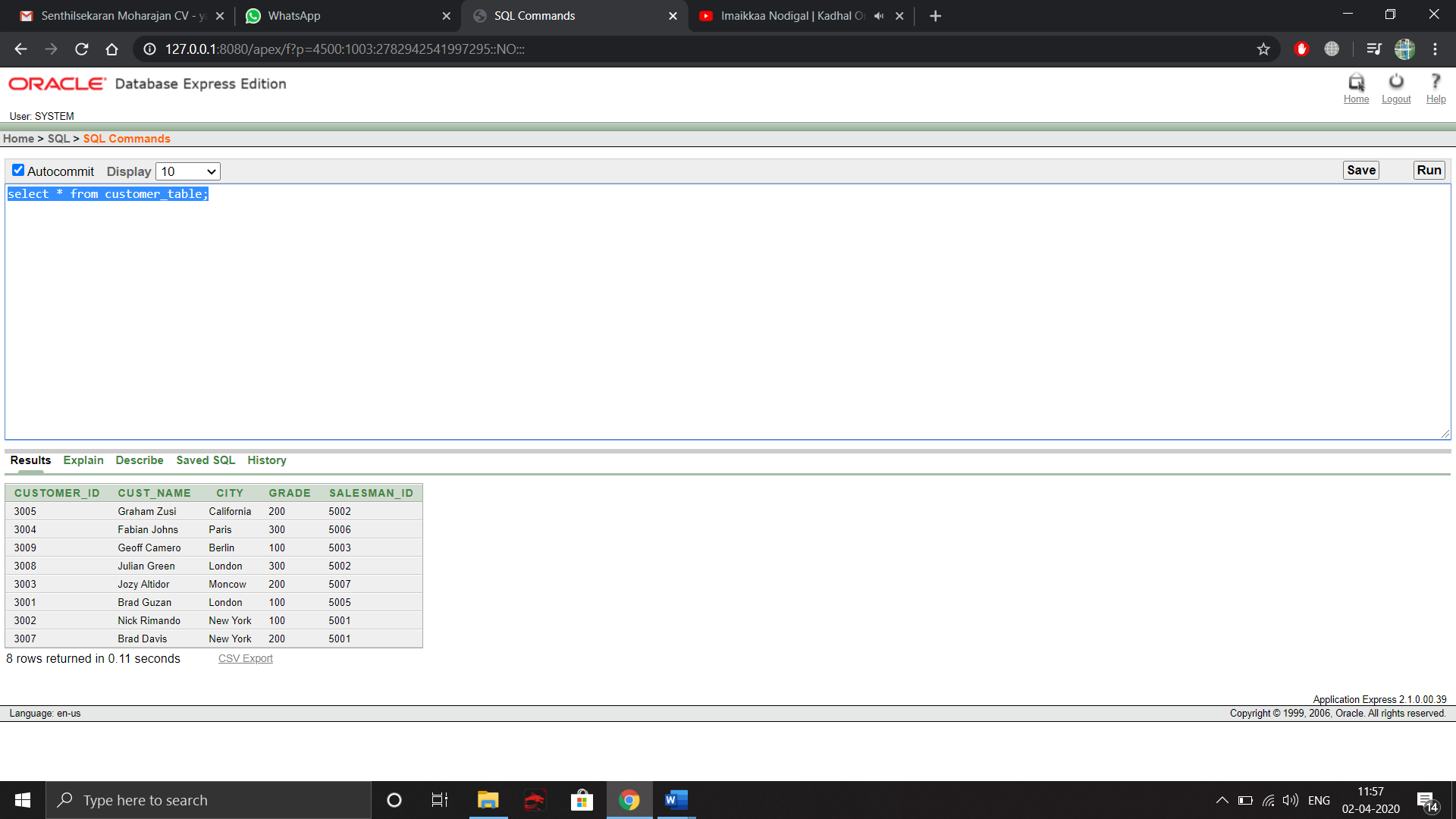
**For example,** Use the query as

select CUSTOMER\_ID,CUST\_NAME,CITY,GRADE,SALESMAN\_ID from customer\_table;



Instead of,

select \* from customer\_table;



**Result:**

Thus optimizing the queries are very much essential to keep the performance and execution time intact for the users.

**EXERCISE-07 (Block Creation and Use of various Data types)**

1. **Execute any 5 sample PL/SQL programs**

**PROGRAM 1**

**CODE**

DECLARE

username varchar(20);

password varchar(20);

textuser varchar(20);

textpass varchar(20);

BEGIN

username:='yajith';

password:='vishwa';

textuser:='&textuser';

textpass:='&textpass';

IF textuser=username THEN

IF textpass=password THEN

DBMS\_OUTPUT.PUT\_LINE('WELCOME');

ELSE

DBMS\_OUTPUT.PUT\_LINE('WRONG PASSWORD');

END IF;

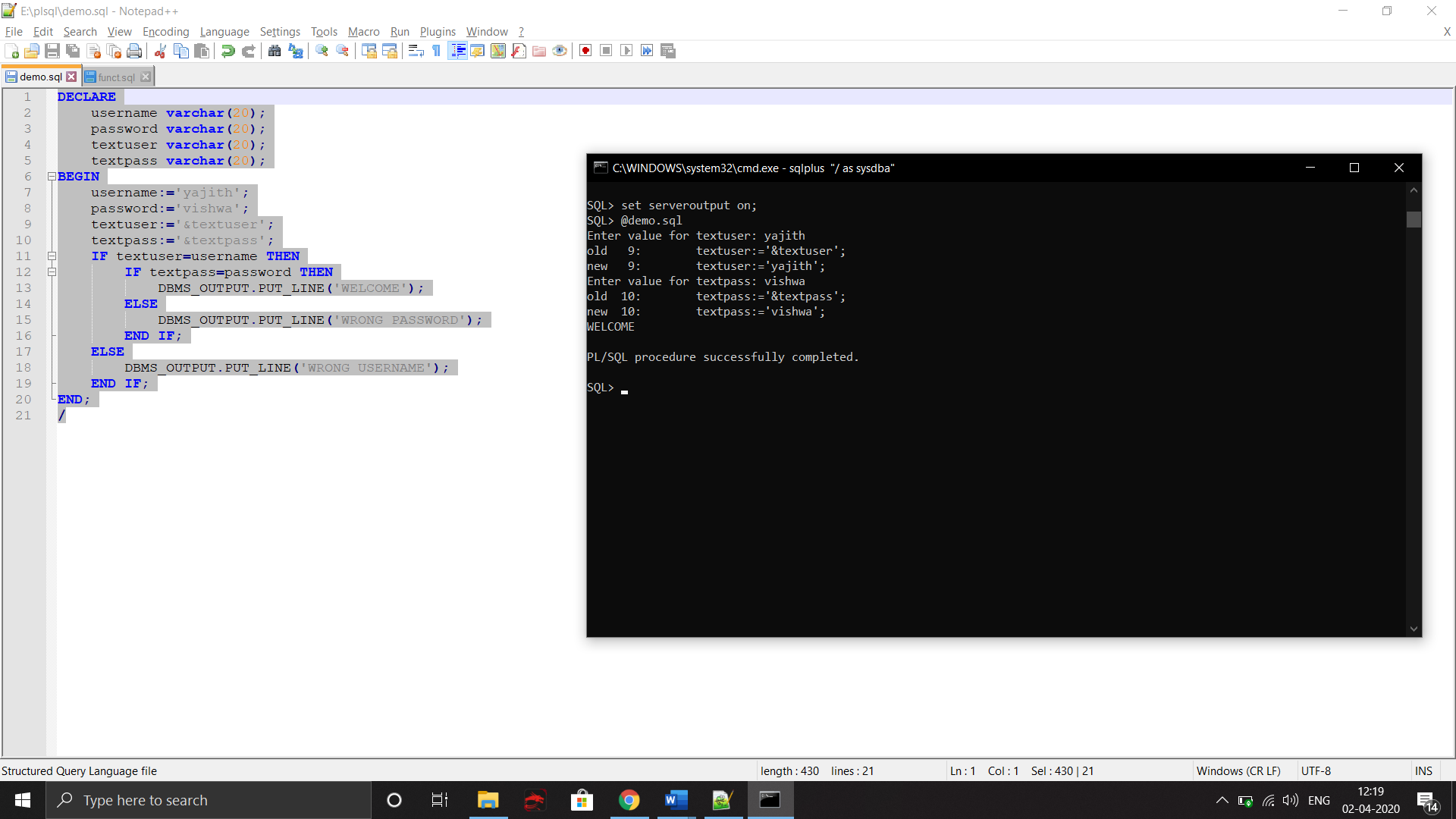
ELSE

DBMS\_OUTPUT.PUT\_LINE('WRONG USERNAME');

END IF;

END;

/



Program 2

Code

DECLARE

str varchar(20);

str1 varchar(20);

leng number;

BEGIN

str:='&str';

leng:=length(str);

FOR i IN REVERSE 1.. leng LOOP

str1:=str1||SUBSTR(str,i,1);

END LOOP;

dbms\_output.put\_line('The REVERSE String is '||str1);

END;

/

Program 3

Code

DECLARE

a number;

b number;

c number;

BEGIN

a:=&a;

b:=&b;

c:=a+b;

dbms\_output.put\_line('Sum: '||c);

c:=a-b;

dbms\_output.put\_line('Difference: '||c);

c:=a\*b;

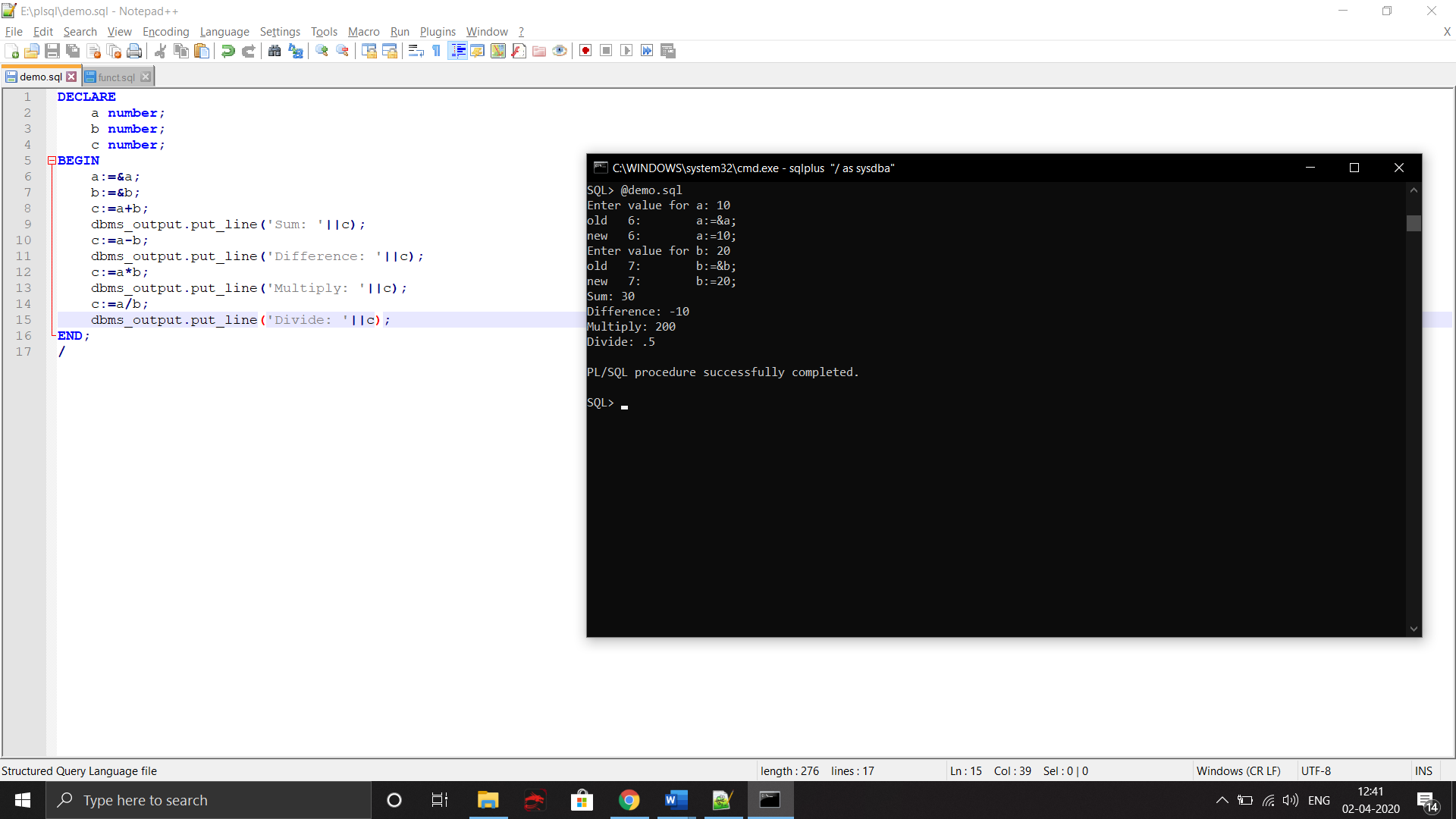
dbms\_output.put\_line('Multiply: '||c);

c:=a/b;

dbms\_output.put\_line('Divide: '||c);

END;

/



Program 4

Code

DECLARE

todate date;

query varchar(30);

BEGIN

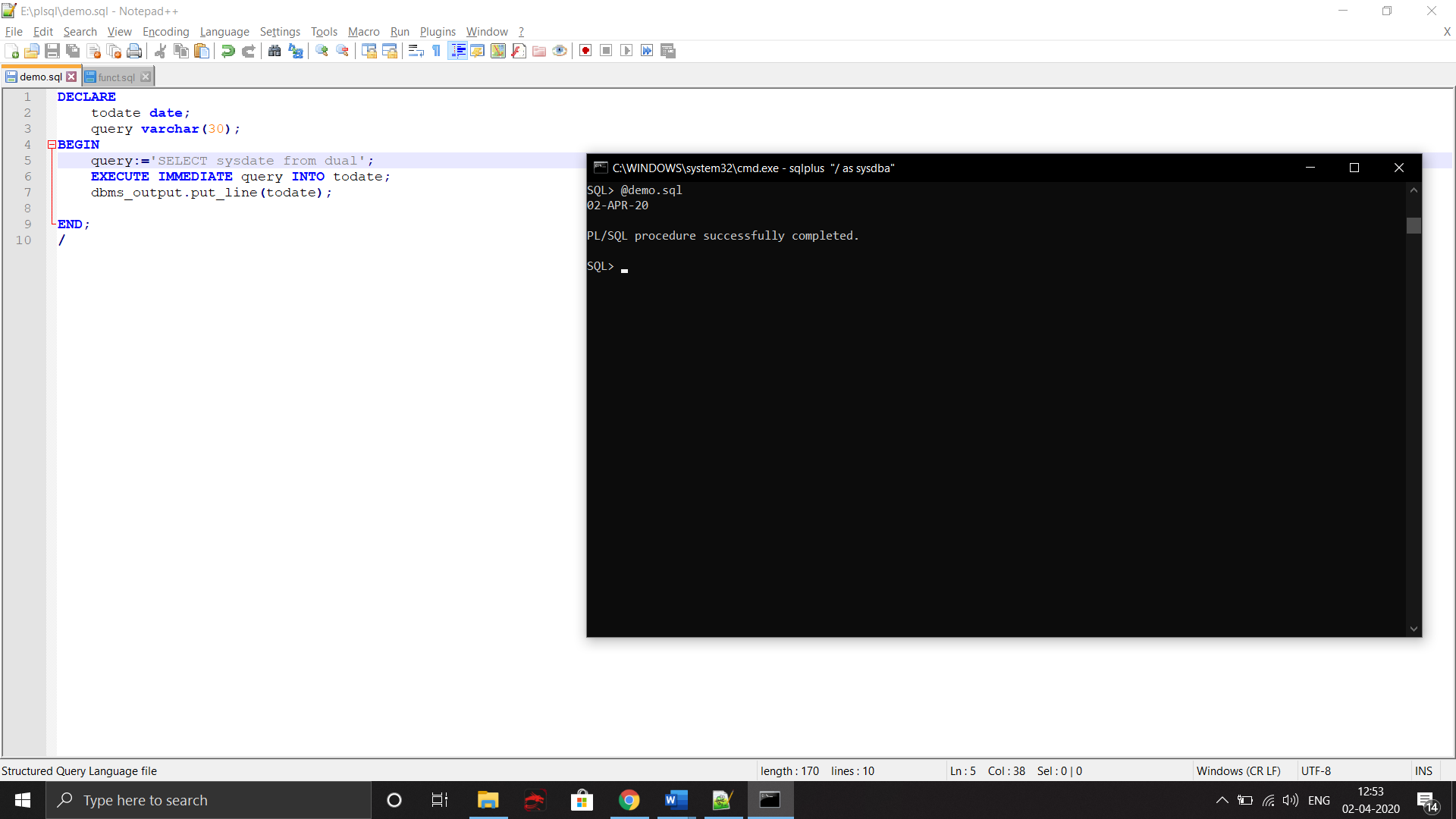
query:='SELECT sysdate from dual';

EXECUTE IMMEDIATE query INTO todate;

dbms\_output.put\_line(todate);

END;

/



Program 5

Code

DECLARE

a number;

b number;

sumvar number;

c number;

d number;

BEGIN

a:=&a;

b:=a\*a;

c:=trunc(b/10);

d:=mod(b,10);

sumvar:=c+d;

IF a=sumvar THEN

dbms\_output.put\_line('Neon NUMBER '||sumvar);

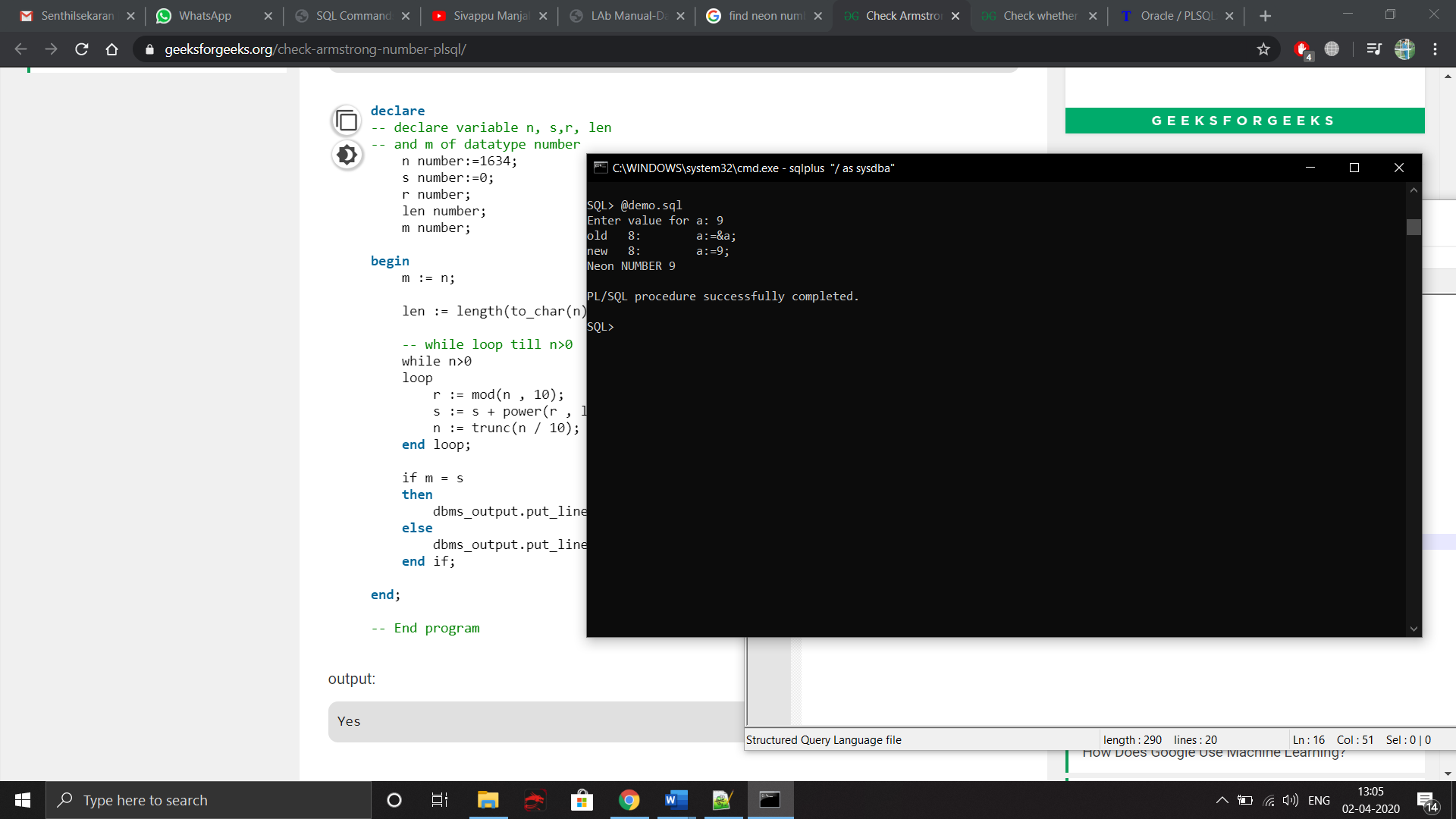
ELSE

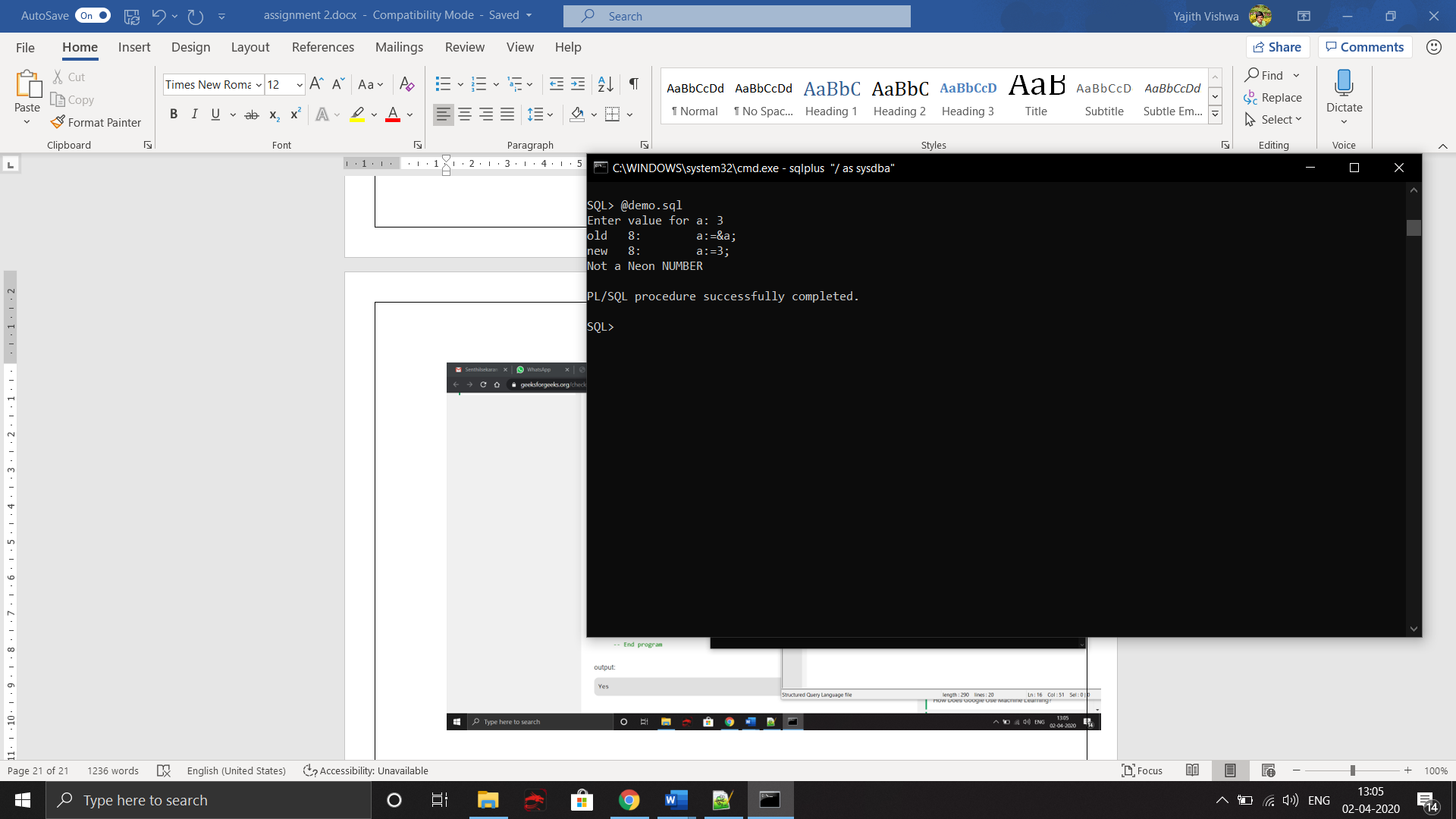
dbms\_output.put\_line('Not a Neon NUMBER');

END IF;

END;

/





**Execute any 5 PL/SQL program for your application**

**PROGRAM 1**

**CODE**

DECLARE

product number;

gst number;

total number;

tax number;

BEGIN

dbms\_output.put\_line('Enter the product cost');

product:=&product;

dbms\_output.put\_line('Enter the gst');

gst:=&gst;

dbms\_output.put\_line('Enter the tax');

tax:=&tax;

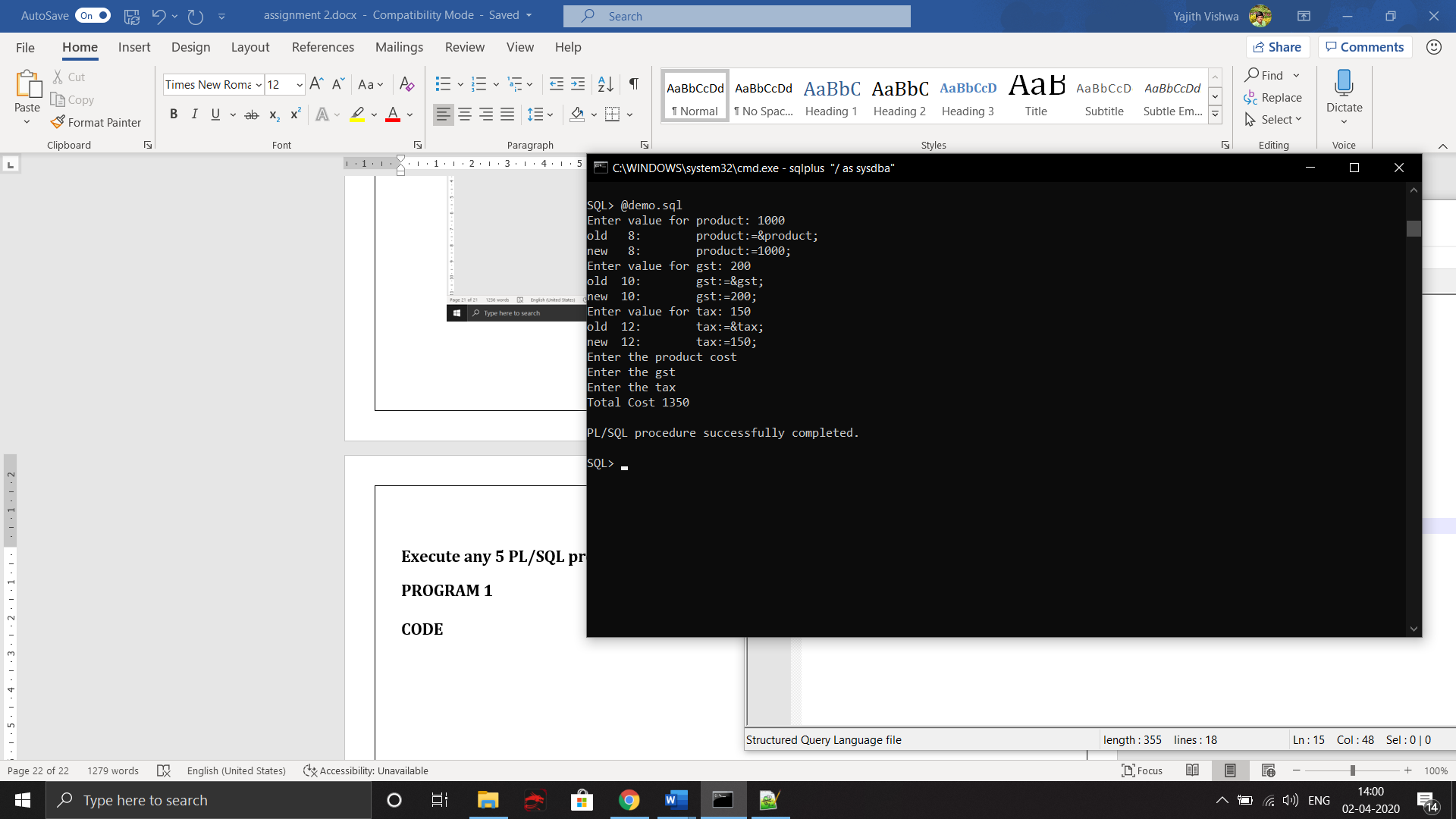
total:=product+gst;

total:=total+tax;

dbms\_output.put\_line('Total Cost '||total);

END;

/



**Program 2**

**Code**

**DECLARE**

**total number;**

**query varchar(70);**

**BEGIN**

**dbms\_output.put\_line('The Total Purchases');**

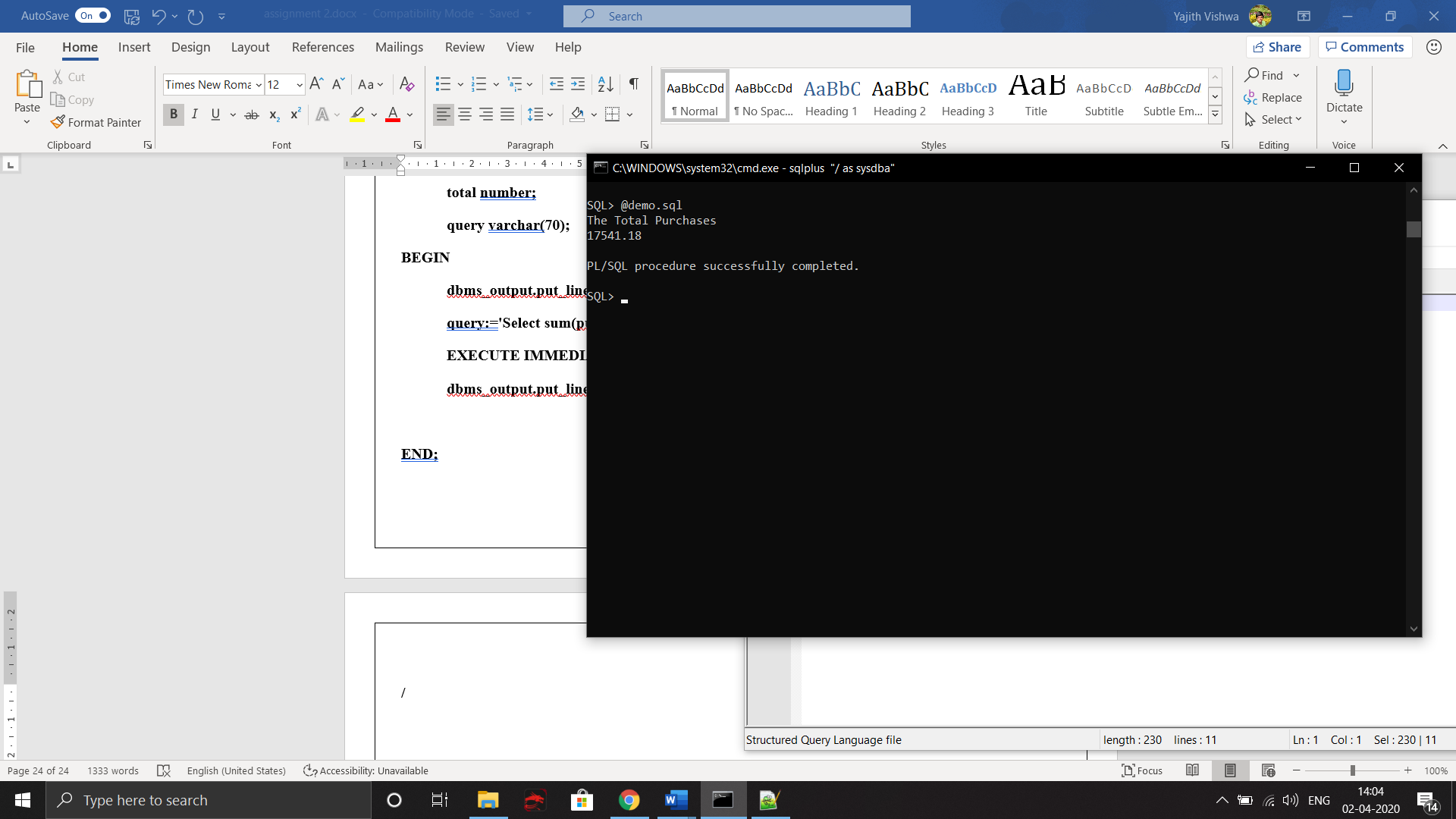
**query:='Select sum(purch\_amt) from order\_table';**

**EXECUTE IMMEDIATE query INTO total;**

**dbms\_output.put\_line(total);**

**END;**

**/**



**Program 3**

**Code**

DECLARE

name number;

grade number;

query varchar(70);

BEGIN

dbms\_output.put\_line('Enter the id');

name:=&name;

query:='Select grade from customer\_table where customer\_id=';

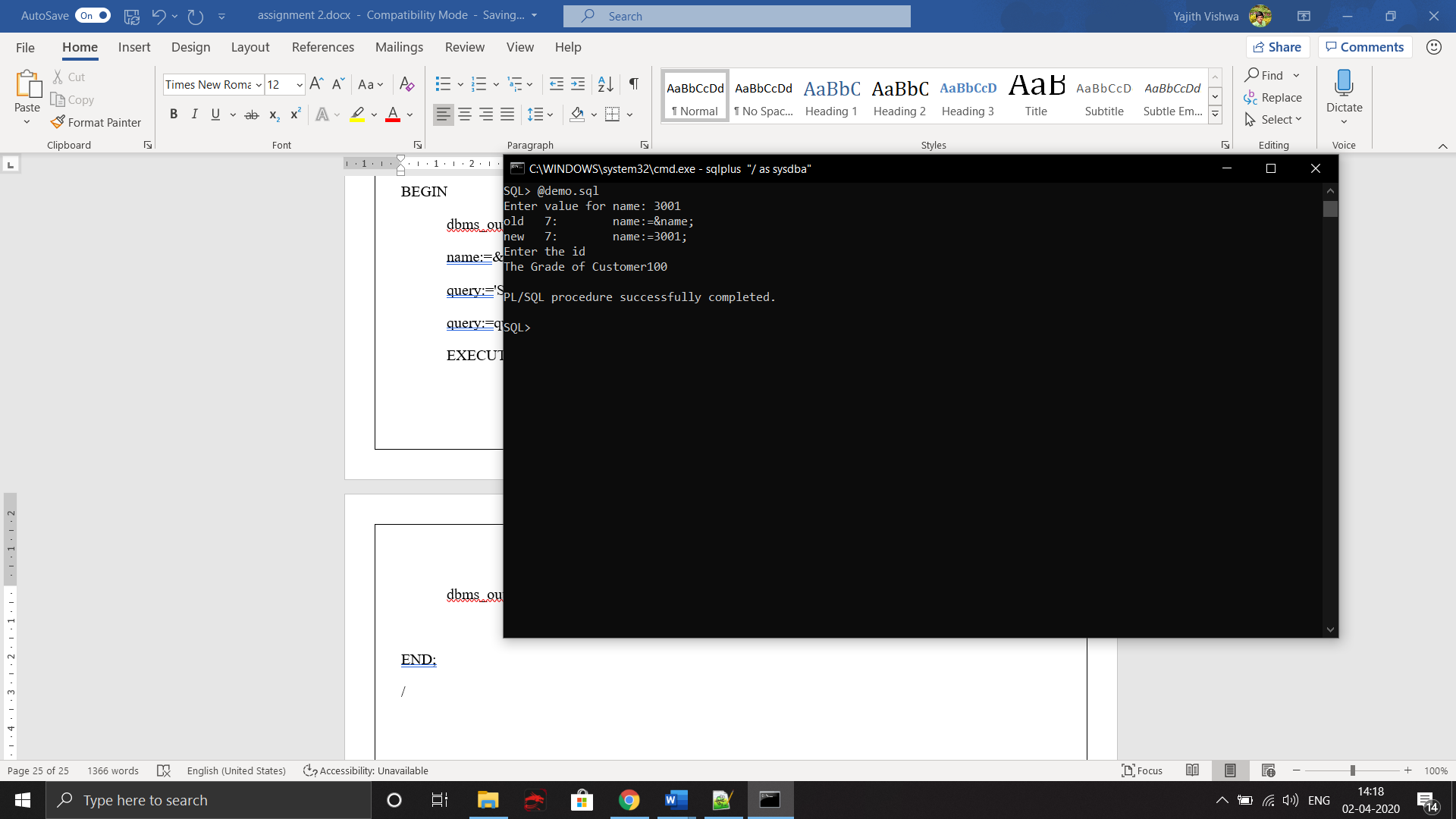
query:=query||name;

EXECUTE IMMEDIATE query INTO grade;

dbms\_output.put\_line('The Grade of Customer'||grade);

END;

/



**Program 4**

**Code**

declare

dname string(25) := 'yajith';

rating number(10);

begin

rating :=&rating;

dbms\_output.put\_line('Name of the driver ');

dbms\_output.put\_line(dname);

dbms\_output.put\_line('Performance');

if rating = 5 then

dbms\_output.put\_line('Excellent');

ELSIF rating = 4 then

dbms\_output.put\_line('Best');

ELSIF rating = 3then

dbms\_output.put\_line('Good');

ELSIF rating = 2 then

dbms\_output.put\_line('Better');

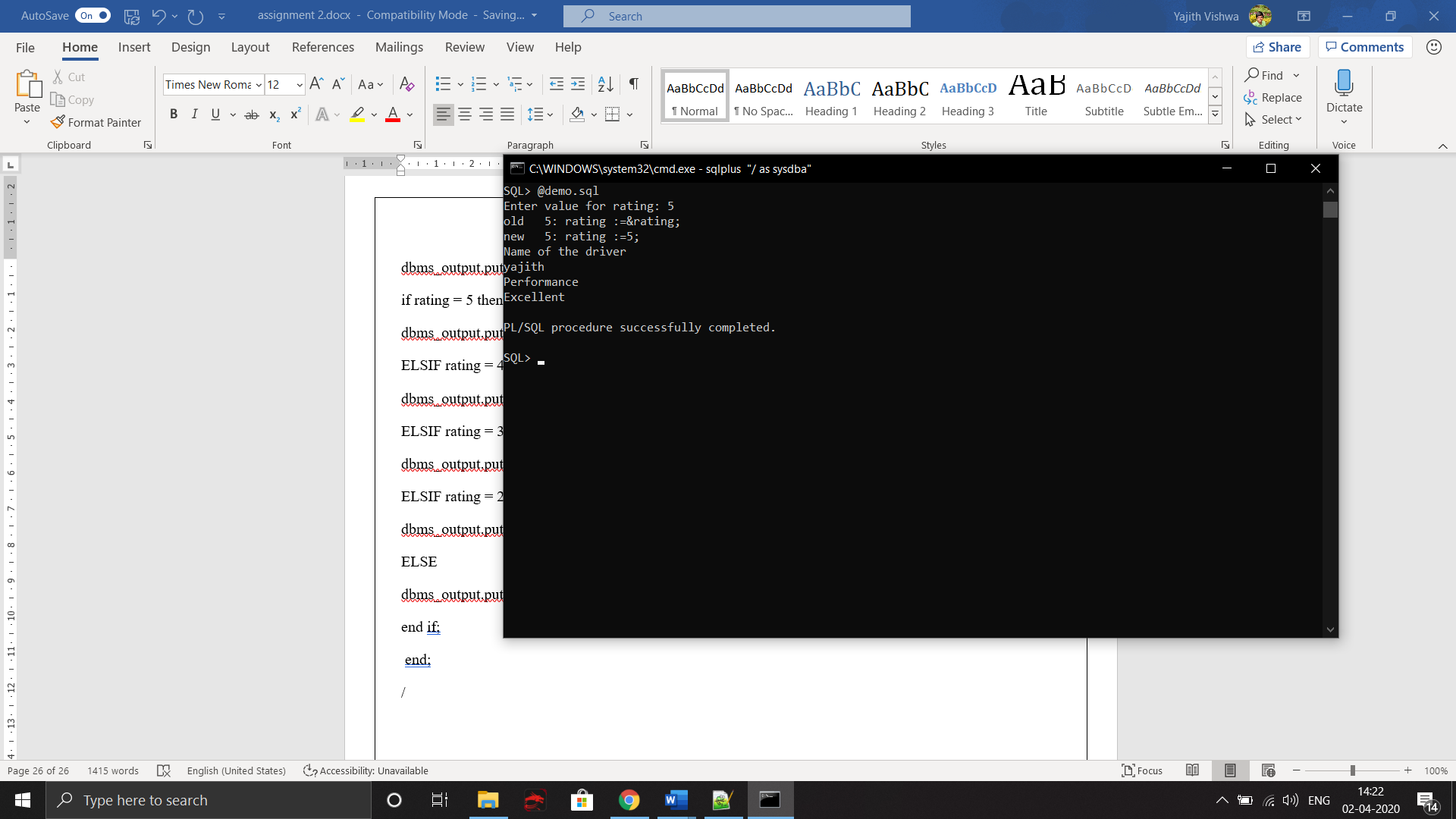
ELSE

dbms\_output.put\_line('Worst');

end if;

end;

/



**Program 5**

**User Defined DataType**

**Code**

DECLARE

SUBTYPE name is varchar(15);

SUBTYPE age IS NUMBER;

ag age;

fullname name;

BEGIN

ag:=&ag;

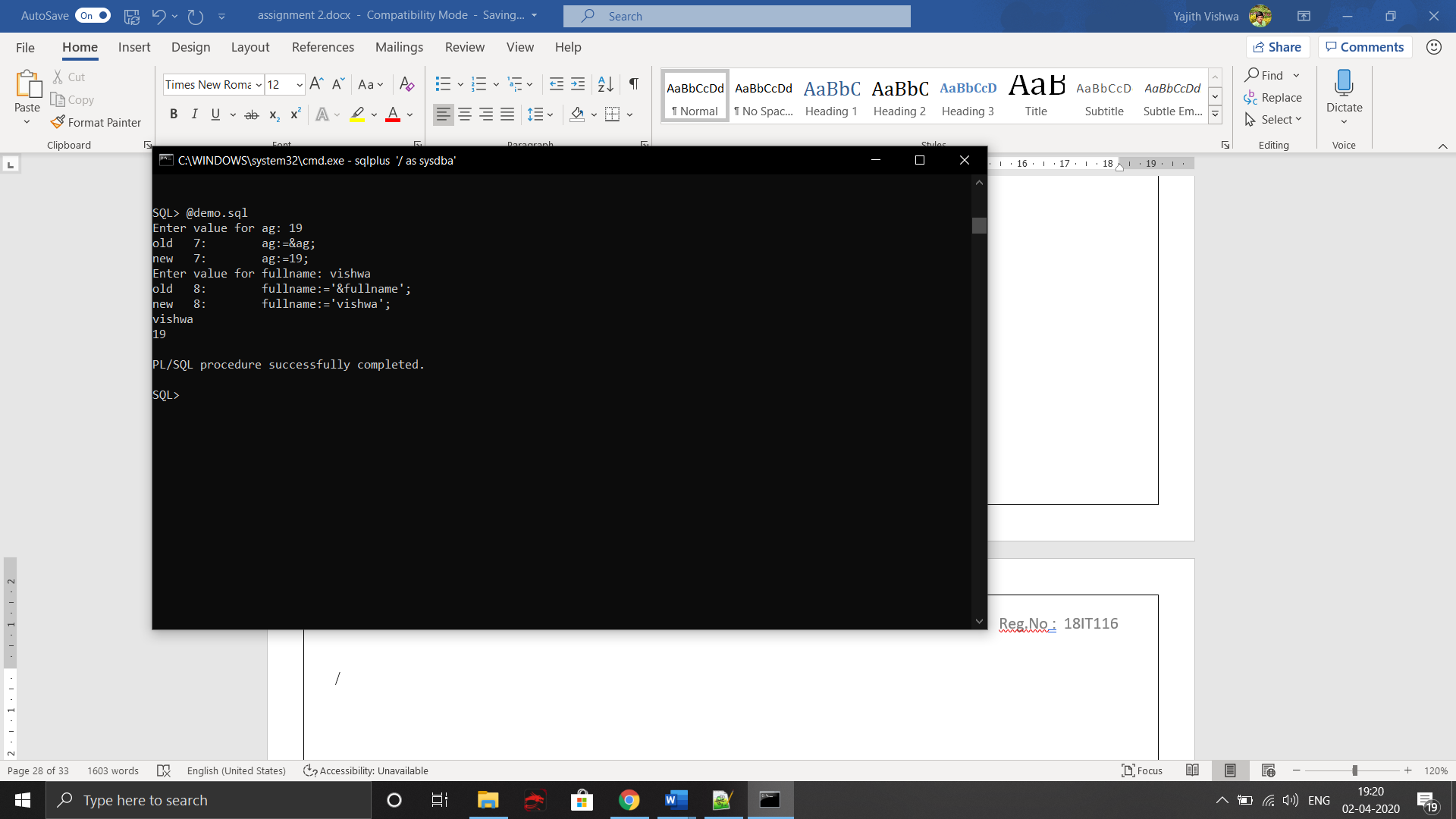
fullname:='&fullname';

dbms\_output.put\_line(fullname);

dbms\_output.put\_line(ag);

END;

/



**Ex No : 7 Procedures, Functions and packages in PL/SQL Aim: To execute the functions, procedures and Packages in SQL**

**Code**

1.Count no of customers

Function:

create or replace function customercount

return number is

counts number;

begin

select count(\*) into counts from customer\_table;

return counts;

end;

/

PLSQL:

DECLARE

a number;

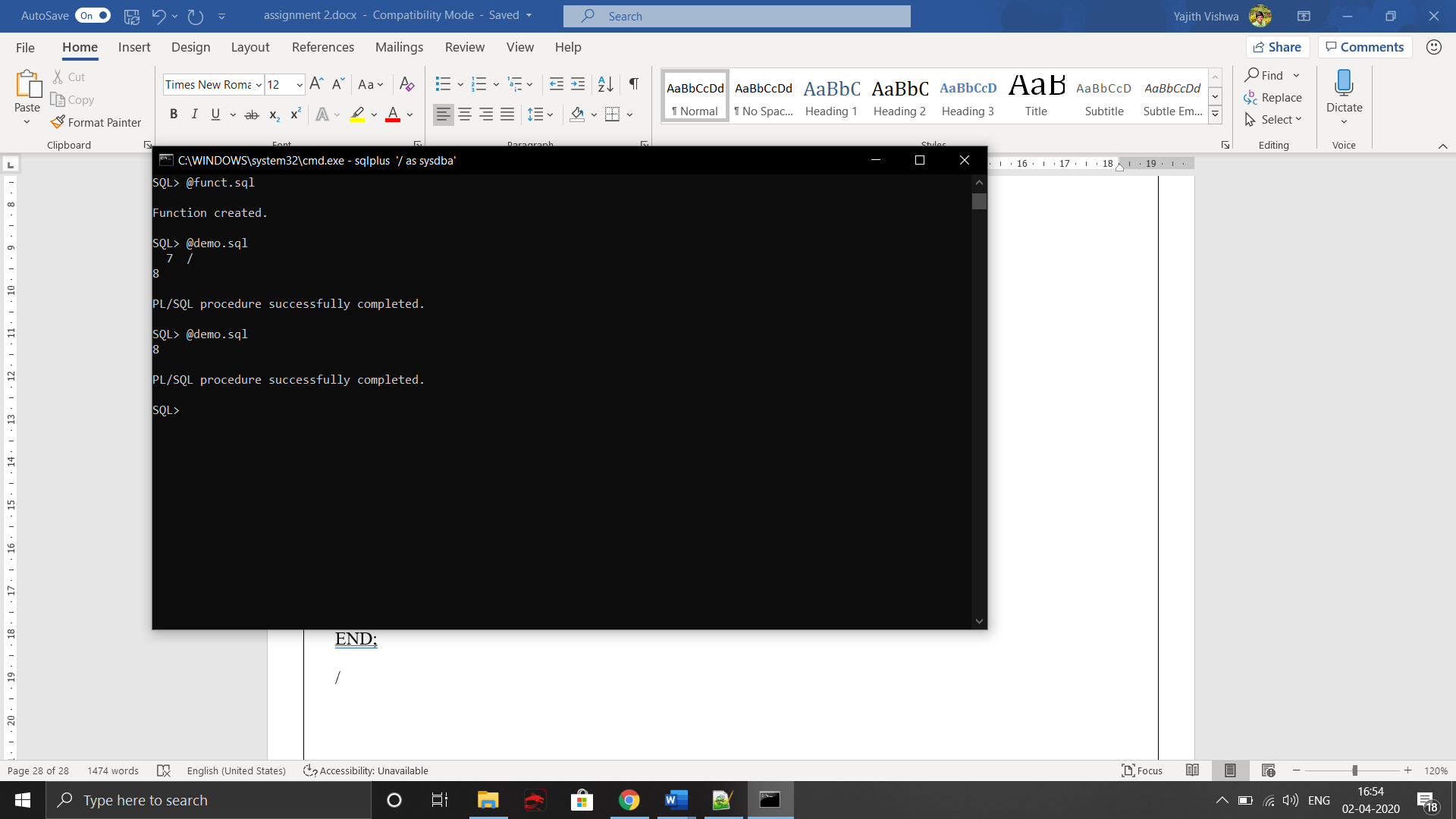
BEGIN

a:=customercount();

dbms\_output.put\_line(a);

END;

/



2.Find the grade and suggestion

Function:

create or replace function grade(x number)

return varchar is

counts varchar(30);

begin

case x

when 100 then counts:='WORK HARD';

when 200 then counts:='GOOD';

else counts:='KEEP GOING';

end case;

return counts;

end;

/

PLSQL:

DECLARE

a number;

query varchar(80);

c number;

b varchar(30);

BEGIN

a:=&a;

query:='select grade from customer\_table where customer\_id=';

query:=query||a;

EXECUTE IMMEDIATE query into c;

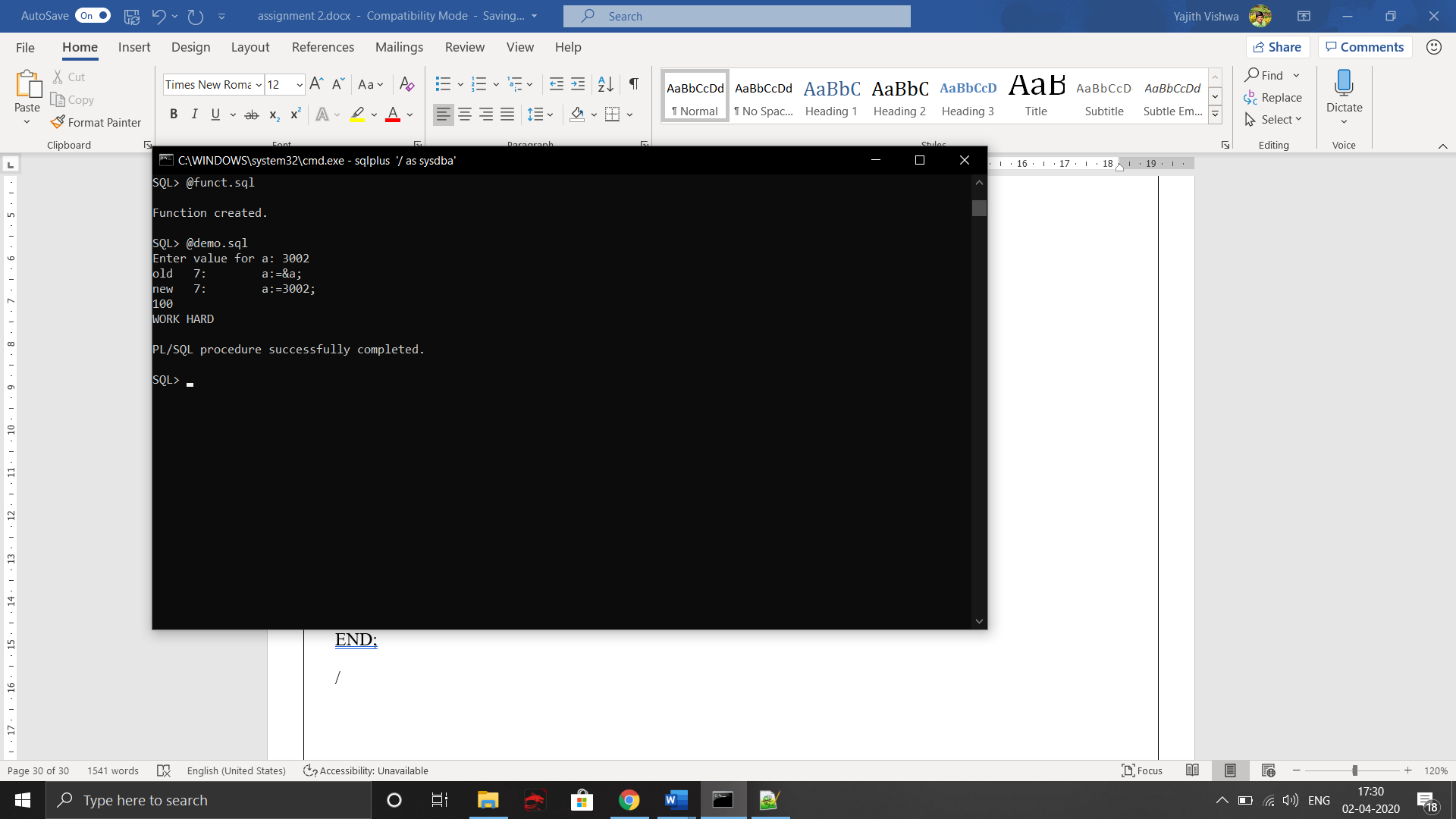
dbms\_output.put\_line(c);

b:=grade(c);

dbms\_output.put\_line(b);

END;

/



**PROCEDURE**

**Code 1:**

Inserting table with help of procedure

Procedure:

CREATE OR REPLACE PROCEDURE squareNum

(a in out number,b in out varchar2) is

BEGIN

INSERT INTO pack VALUES(a,b);

END;

/

PLSQL:

DECLARE

a number;

b varchar2(20);

BEGIN

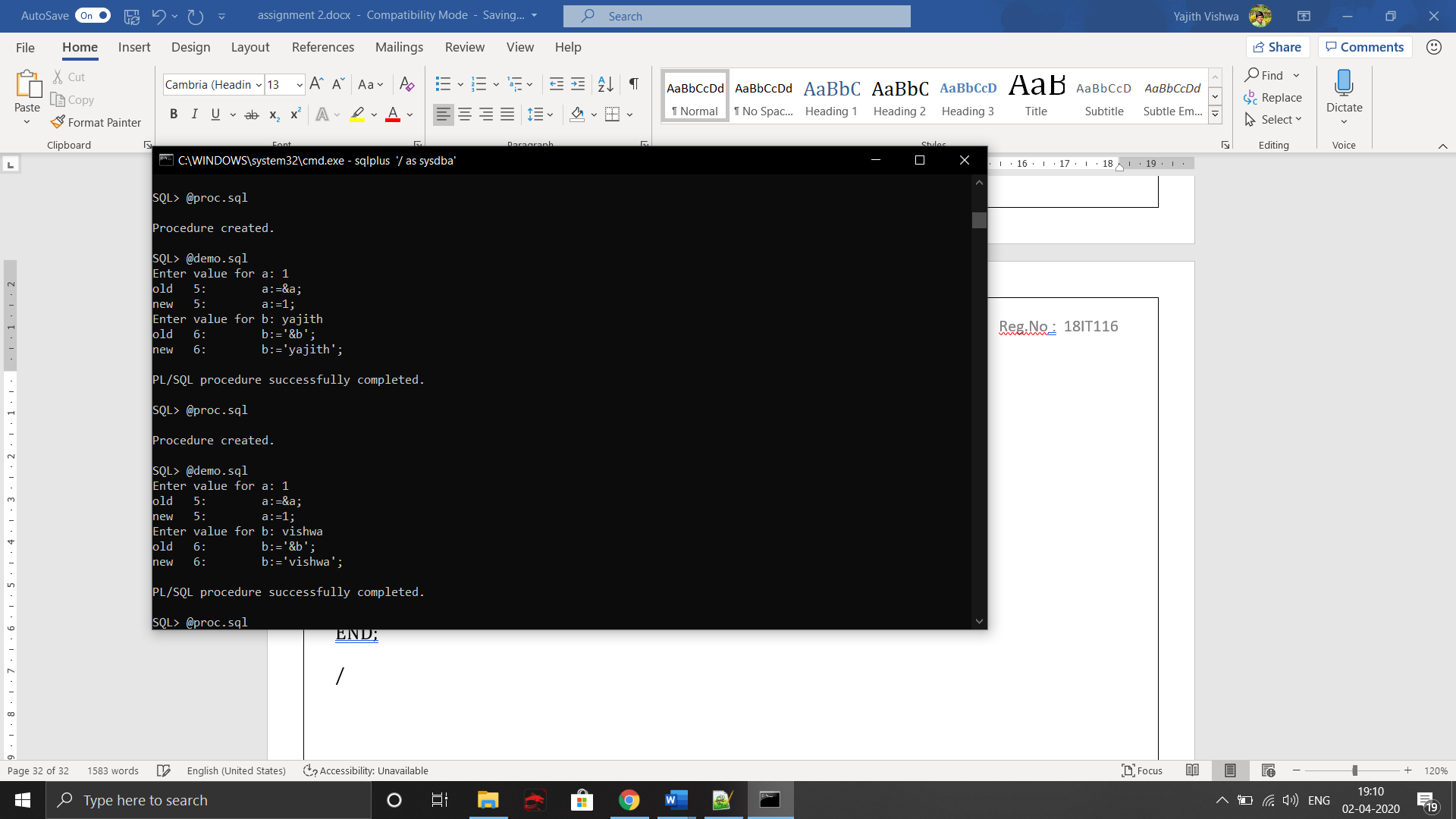
a:=&a;

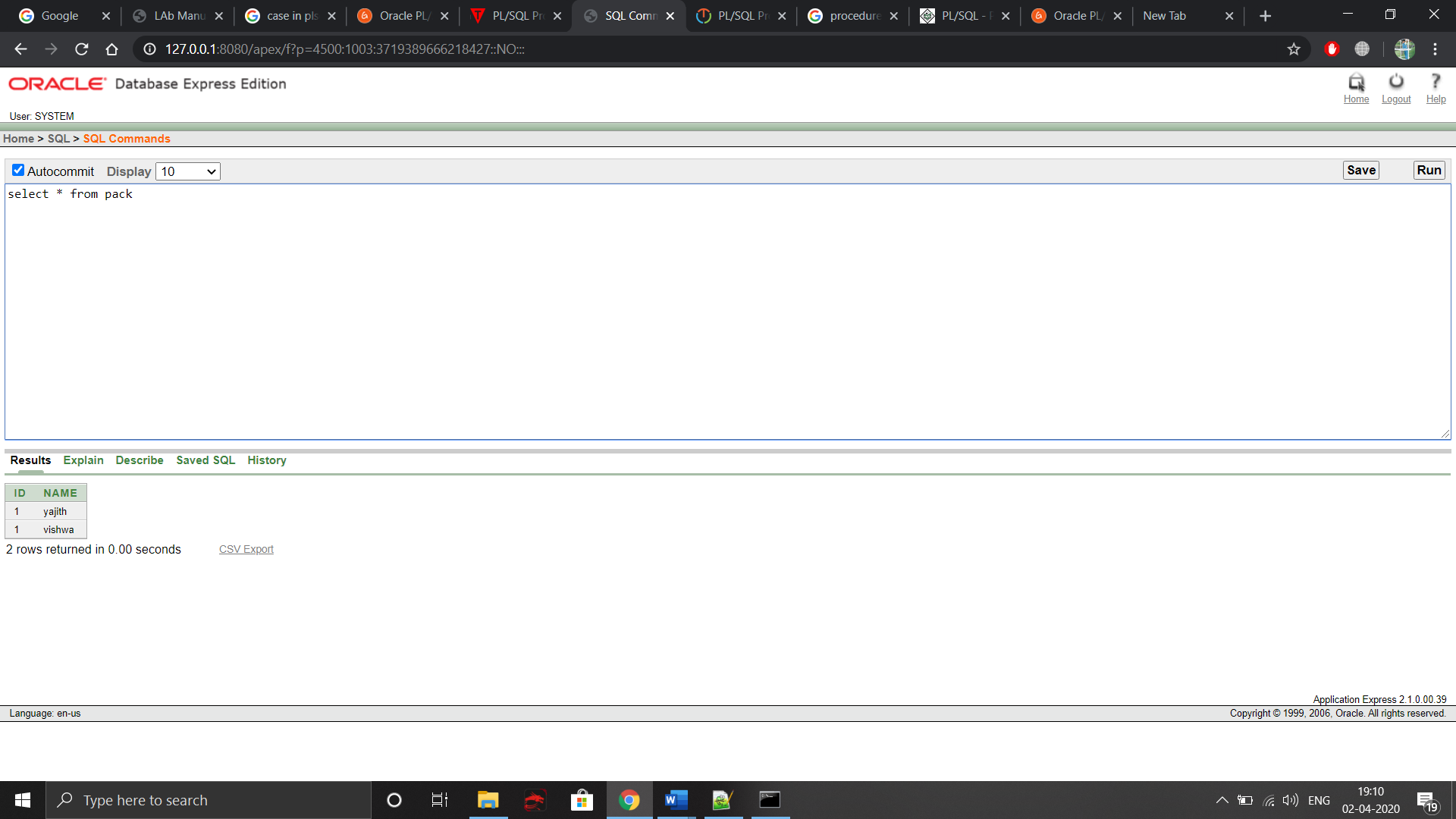
b:='&b';

squareNum(a,b);

END;

/





Code 2:

Sum of Purchased Amount in Ordertable

Procedure:

CREATE OR REPLACE PROCEDURE sumofpurch

(a in out number) is

BEGIN

SELECT sum(purch\_amt) INTO a FROM order\_table;

END;

/

PLSQL:

DECLARE

a number;

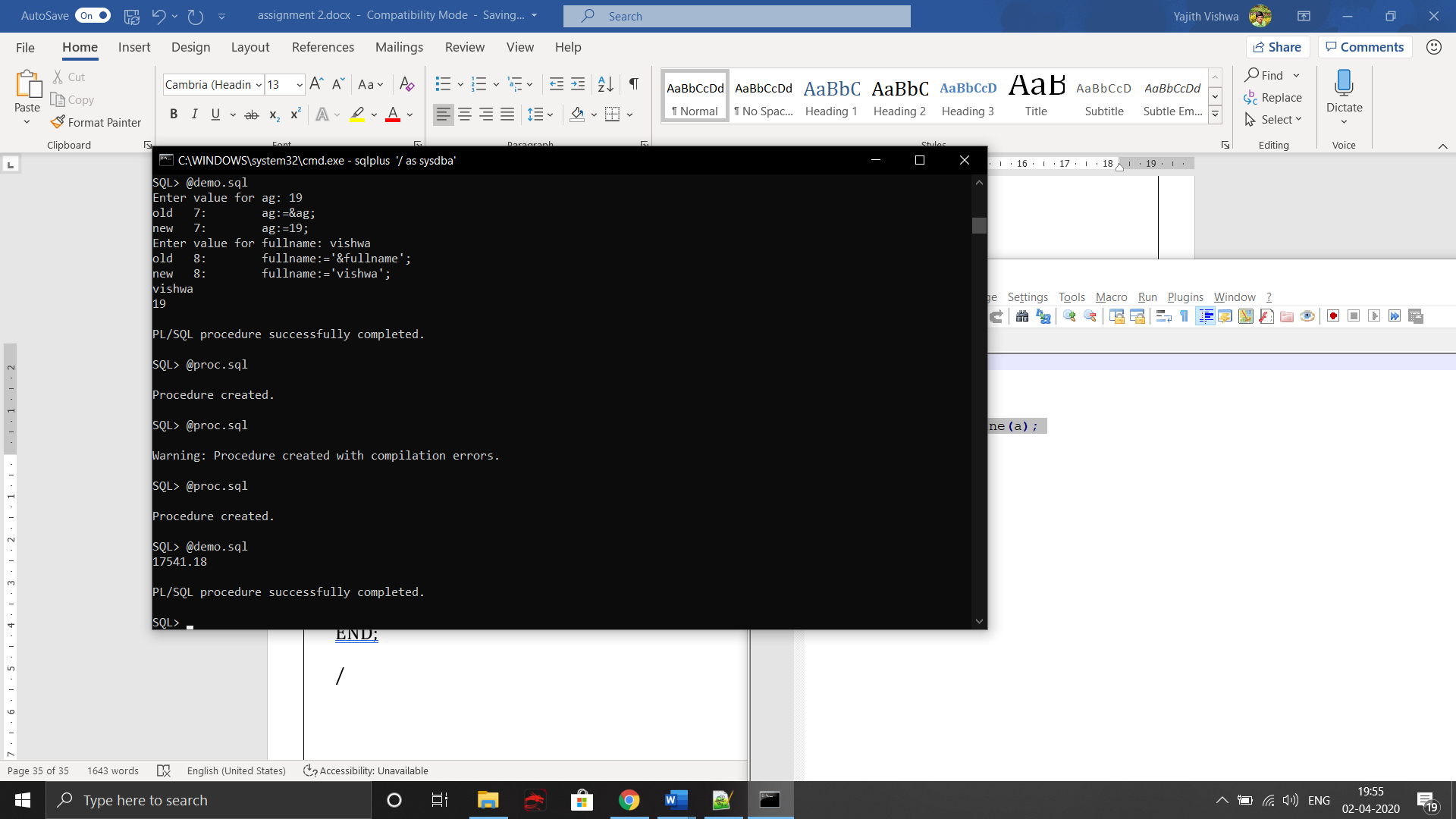
BEGIN

sumofpurch(a);

dbms\_output.put\_line(a);

END;

/



Package:

**Procedure:**

CREATE OR REPLACE PROCEDURE FIND\_AREA\_DETAIL(CODE IN LOCATION.AREA\_CODE%TYPE) IS

AREA LOCATION.AREA\_NAME%TYPE;

DISTRICT LOCATION.DISTRICT%TYPE;

STATE LOCATION.STATE%TYPE;

POPULATION LOCATION.POPULATION%TYPE;

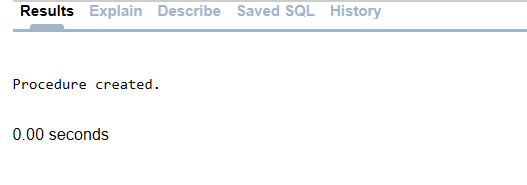
BEGIN

SELECT AREA\_NAME,DISTRICT,STATE,POPULATION INTO AREA,DISTRICT,STATE,POPULATION FROM LOCATION WHERE AREA\_CODE=CODE;

DBMS\_OUTPUT.PUT\_LINE('AREA NAME >>> '||AREA ||' DISTRICT >>> '|| DISTRICT || ' STATE >>> '|| STATE || ' POPULATION >>> '|| POPULATION);

END;

**Output:**



DECLARE

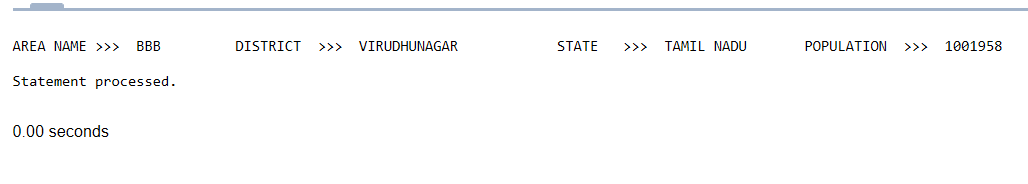
CODE LOCATION.AREA\_CODE%TYPE:='TN-101';

BEGIN

FIND\_AREA\_DETAIL(CODE);

END;

**Output:**



**EXERCISE-08 (Cursor Management and Creation of Triggers)**

**IMPLICIT CURSOR**

DECLARE

count\_customer number;

query varchar(90);

BEGIN

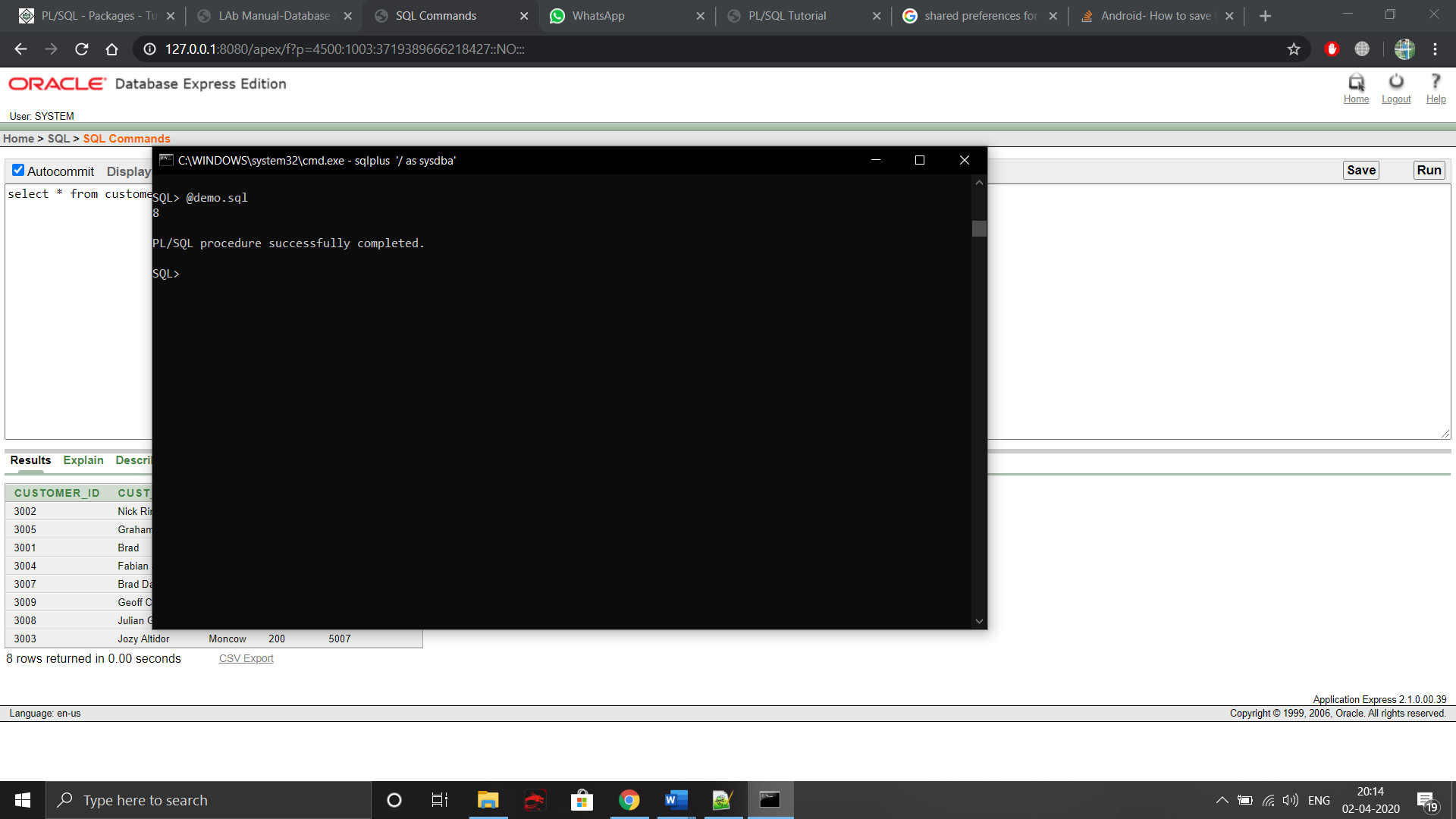
query:='SELECT COUNT(\*) FROM customer\_table';

EXECUTE IMMEDIATE query INTO count\_customer;

dbms\_output.put\_line(count\_customer);

END;

/



EXPLICIT CURSOR:

DECLARE

cust\_name customer\_table.cust\_name%type;

CURSOR name IS SELECT cust\_name FROM customer\_table;

BEGIN

OPEN name;

LOOP

FETCH name INTO cust\_name;

EXIT WHEN name%NOTFOUND;

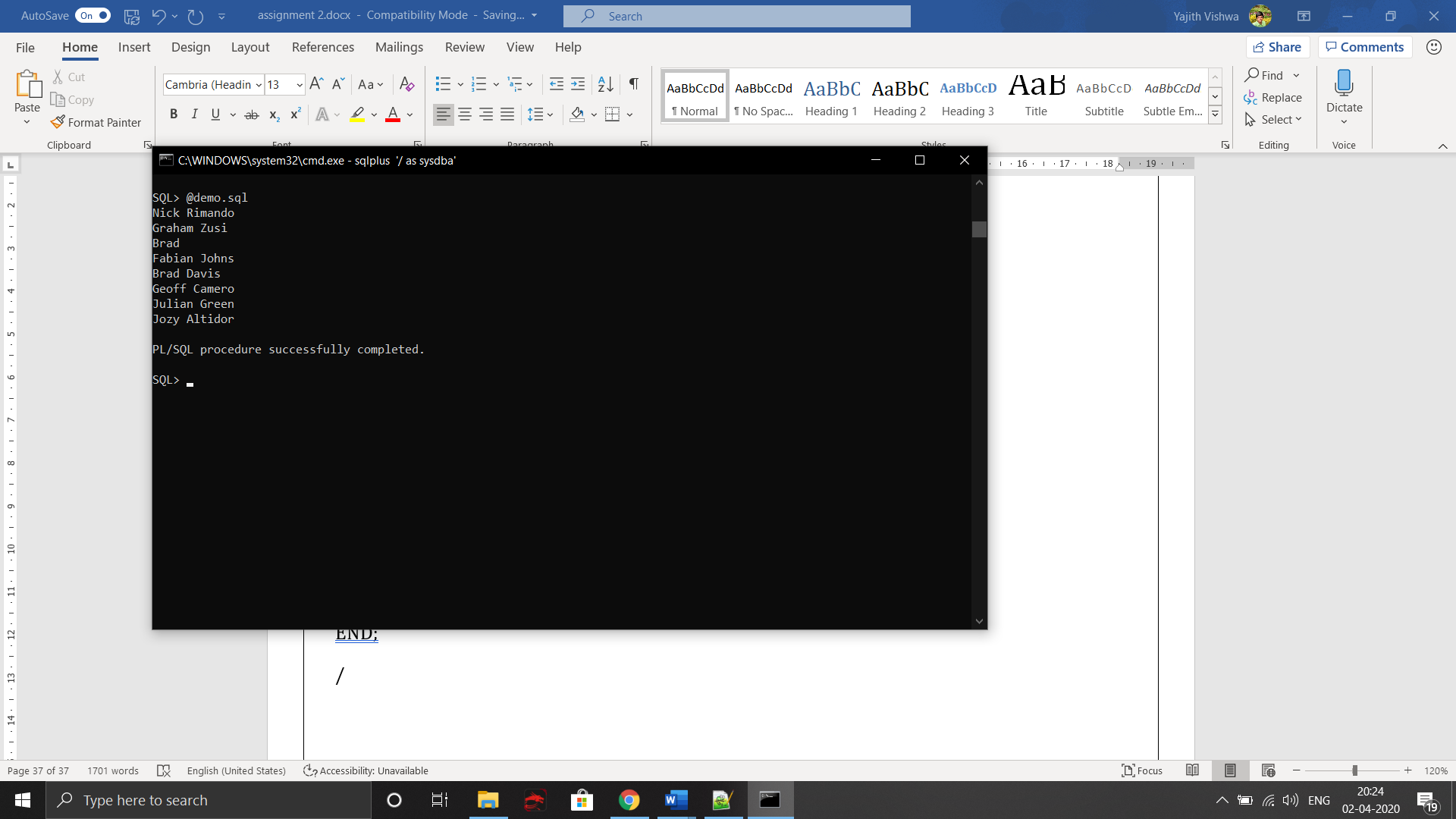
dbms\_output.put\_line(cust\_name);

END LOOP;

CLOSE name;

END;

/



TRIGGER:

CREATE OR REPLACE TRIGGER trg\_LOCATION

AFTER UPDATE

OF TOTAL\_WASTE

ON employee\_salary

FOR EACH ROW

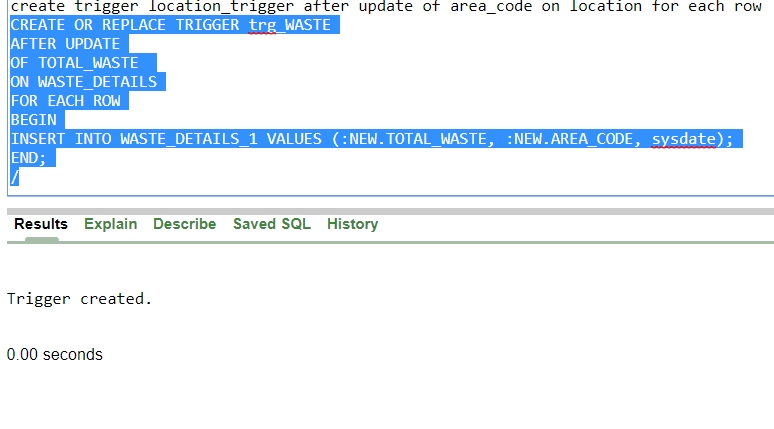
BEGIN

INSERT INTO WASTE\_DETAILS\_1 VALUES (:NEW.TOTAL\_WASTE, :NEW.AREA\_CODE, sysdate);

END;

/

**Output:**

****