STOREMART

ADVANCED DATABASE MANAGEMENT SYSTEM

GROUP PROJECT

PREPARED BY:

JUHI GUPTA

NIDHI PARASHAR

PANKAJ KUMAR

VAIBHAV MIGLANI

TABLE OF CONTENTS

INTRODUCTION

Storemart is a departmental store chain which currently has ten stores across ten different cities in the United States. Storemart is looking for expanding their business by opening more stores but first they need a centralized management system that can help them in capturing all the information at one place and keep all the stores in sync. This means, for example if a customer at the Atlanta outlet needs an electronic item which is out of stock in that store, supervisor of the store can check in the system if that item is available at their Alpharetta store and order it for that customer. For our project, we have designed a database management system for Storemart which will help them in maintaining the records of the daily sales, inventory, customers, employees, vendors at all their stores.

BASIC INFORMATION OF THE SYSTEM

For any business to be successful, capturing the relevant data and using it to expand their business is critical. Recognizing which data is useful and should be recorded is equally important. For this database management system, we have identified thirteen important and created tables for each of them in the database.

Few of these tables are

- Employees
- Products
- Stores
- Sales
- Payments
- Customers

The above-mentioned tables are the most critical ones for this system. The reasons for their importance are described below.

Recording the information of employees is important for any organization. This database management system will record the details of all the employees who are currently working or have worked in past at any of Storemart's outlets.

For a departmental store to run smoothly, holding the right quantity of stock in the inventory is vital. Products table will hold the information of all the products that Storemart offers to its customers. It will keep the record of what comes in the inventory and what goes out from the store (through sales).

Next most important thing for any business is clientele, especially its loyal customers. Customers table in our data base will hold the details of all the loyal customers at different outlets of Storemart. These customers are those who shop regularly and contribute significantly to the business. For this reason, these customers have a Loyalty card that earns them reward points each time they shop, and they can redeem their points to get good discounts.

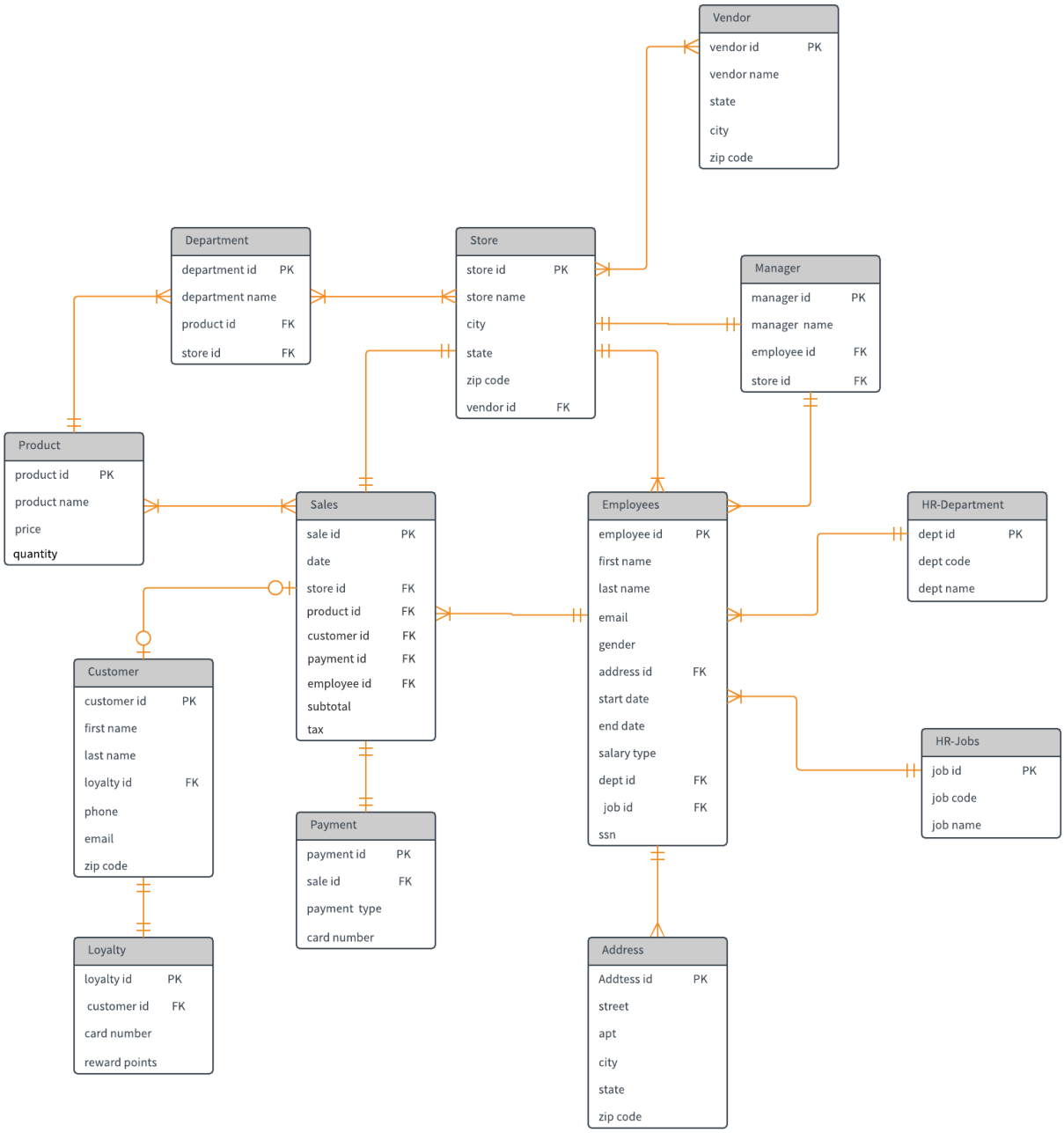
Stores table will maintain the information of all the store outlets of Storemart located in different cities. This will help all the stores to keep in sync with each other.

Sales table is for capturing all the purchases that take place at the stores. Information of what items were bought by which customer at which store, the subtotal and the tax amount will all be saved in the sales table.

Payments table will retain the information of each transaction that takes place at any of the store. The modes of payment, store where the transaction took place, all this information will be recorded in the payments table.

How these tables are related to each other, is shown through an Entity-Relationship Diagram of the database.

ENTITY-RELATIONSHIP DIAGRAM



Brief description relationships between entities

For this database design, relationships between the entities have been assigned, keeping in mind certain assumptions. Sales is the most important entity and is related to a lot of other entities such as Products, Employees, Customers, Payment and Stores.

- Sales records the details of every purchase that occurs at any of the outlets of Storemart. Every sale is recorded by one employee and an employee can record many sales, which is why there is a many-to-one relationship between the Sales and Employees table.

- Every sale record will have at least one or more than one product id in it and every product can be in one or more sales records, therefore, a many-to-many relationship exists between the Products and Sales tables.

- As mentioned earlier, the database will retain the information of each sales record, which is why it is important for each sales record to be linked with the store id to keep the track of which purchase took place at which store outlet. So, each sale record will have one and only one store id, but each store will have more than one sale records corresponding to it, hence one-to-many relationship exist between the Sales and Stores tables.

- For our database design, we have made some assumptions and one of them is that the Customers table will hold the details of only the loyal customers. The reason for this assumption is that in a departmental store, hundreds of customers come to shop in a day and not all of them come in the second time. There are lot of one-time customers and capturing every customers information will lead to data overflow. So, only the customers who are regular shoppers at Storemart or to put it in other way, those who have a loyalty card, will have their details recorded into the database. Now coming to the entity-relationship, sales record may or may not have a customer id therefore, there is a zero or more relationship between the Sales and Customers tables.

- Each sale record will have a payment id associated with it and each payment record will have one and only sale id associated to it thus, leading to a one-to-one relationship between the Sales and Payment tables.

- The Loyalty table will store the details of loyalty cards issued to customers. Based on our assumption that only the loyal customers' details will be recorded in the database, brings us to the conclusion that each customer record in the Customer table will have a loyalty card issued to their name and every customer will have only one loyalty card issued in their name. So, Customers and Loyalty tables have one-to-one relationship with each other.

- The Vendors table will store the details of all the vendors that supply the stock at the stores. It is obvious for a departmental store to have more than one supplier and a vendor can supply at more than one store thus, creating a many-to-many relationship between the Stores and Vendors tables.

- Another table called Managers, will store the information about the managers of each store and the employees working under them. We have assumed that each store outlet will have only one manager and hence the Store and Managers tables have one-to-one relationship between them. As for the Managers and Employees tables, there is a one-to-many relationship between them as a manager will have more than one employees working under him/her, but each employee will have only one manager.

- For creating normalization, we have designed an Address table which will store the address details of all the employees. The reason for creating a separate table for storing the address is to de-clutter the employees table. As for the relationship between the Employees and Address tables, it is many-to-one. An employee can have more than one address. But each record in the address table will have only one employee id associated with it.

- As in a general scenario, products are categorized broadly into departments. For example, in a departmental store there are various departments like food, clothes, electronics etc. We have created a Departments table which has a relationship with the Products table as one-to-many relationship, as one department will have more than one product under it. The relationship between Departments and Stores tables is many-to-many, as one store will have many departments and each department will be at all the stores.

After defining the functionalities and relationship of each table with other tables in the database, below are the snippets of the tables, their attributes and data types and keys.

1. CUSTOMER

Attribute_Name	Type	Null Allowed	Unique
Customer_ID	Number	No	Yes
First_Name	Varchar	No	No
Last_Name	Varchar	Yes	No
Phone_Number	Varchar2	No	Yes
Zip_Code	Number	No	No
Loyalty_Id	Varchar	Yes	Yes

Primary Key-Customer_Id

Foreign Key-Loyalty_Id

2. STORE

Attribute_Name	Type	Null Allowed	Unique
Store__Id	Number	No	Yes
StoreName	Varchar	No	Yes
City	Varchar	Yes	No
State	Varchar	Yes	No
ZipCode	Number	Yes	No
Vendor_Id	Number	No	No

Primary Key-Store_ID

Foreign Key-Vendor_Id

3. EMPLOYEE

Attribute_Name	Type	Null Allowed	Unique
Employee_Id	Number	No	yes
First_Name	Varchar	Yes	No
Last_Name	Varchar	No	No
Phone	Varchar	No	Yes
Email	Varchar	No	Yes
Gender	Varchar	yes	No
Address_Id	Number	Yes	No
Start_Date	Date	No	No
End_Date	Date	Yes	No
Salary_Type (Hourly = 0 & Salaried = 1)	Number	No	No
Dept_Id	Number	Yes	No
Job_Id	Number	Yes	No
SSN	Varchar	No	No

Primary Key-Employee_Id

Foreign Key-Address_Id, Dept_Id, Job_Id

4. LOYALTY

Attribute_Name	Type	Null Allowed	Unique
Loyalty_Id	Varchar	No	Yes
Customer_ID	Number	No	Yes
Loyalty_Card_Number	Number	No	Yes
Reward_Points	Number	No	No
Validity_Date	Date	No	No

Primary Key-Loyalty_Id

Foreign Key-Customer_Id

5. SALES

Attribute_Name	Type	Null Allowed	Unique
Sales_Id	Number	No	Yes
Date	Date	No	Yes
Store_Id	Number	No	No
Customer_Id	Number	Yes	No
Prodcut_Id	Number	No	No
Employee_Id	Number	Yes	No
Payment_Id	Number	No	No
Sub_Total	Number	No	No
Tax	Number	Yes	No

Composite Key- Sales_Id, Product_Id, Customer_Id

6. DEPARTMENT

Attribute_Name	Type	Null Allowed	Unique
Department_Id	Number	No	Yes
Store_Id	Varchar	No	Yes
Department_Name	Varchar	Yes	No
Product_Id	Varchar	Yes	No

Primary Key-Department_Id

Foreign Key-Product_Id

7. PRODUCT

Attribute_Name	Type	Null Allowed	Unique
Product_Name	Varchar	No	No
Product_Id	Number	No	Yes
Price	Number	No	No
Quantity	Number	Yes	No

Primary Key-Product_Id

8. PAYMENT

Atrribute_Name	Type	Null Allowed	Unique
Payment_Id	Number	No	Yes
Sale_Id	Number	No	Yes
Payment_Type	Varchar	No	No
Credit_Card_Number	Number	Yes	Yes
Credit_Card_Type	Varchar	Yes	No
Credit_Card_Expiry_Dt	Date	Yes	Yes

Primary Key-Payment_Id

Foreign Key-Sale_Id

9. VENDOR

Atrribute_Name	Type	Null Allowed	Unique
Vendor_Id	Number	No	Yes
Store_ Id	Number	No	Yes
Vendor_Company_Name	Varchar	No	Yes
Vendor_Address	Varchar	No	Yes
Vendor_City	Varchar	No	No
Vendor_Zip_Code	Number	No	No
Vendor_Phone	Number	No	Yes

Primary Key-Vendor_Id

Foreign Key-Store_Id

10. HR JOBS

Attribute_Name	Type	Null Allowed	Unique
Job_Id	Number	No	Yes
Job_Code	Varchar	No	Yes
Job_Name	Varchar	No	No
Job_Desc	Varchar	Yes	No
Job_Start_Date	Date	No	No
Job_End_Date	Date	Yes	No
Job_Set_Code	Varchar	No	No

Primary Key-Job_Id

11. HR DEPARTMENT

Attribute_Name	Type	Null Allowed	Unique
Department_Id	Number	No	Yes
Department_Code	Varchar	No	Yes
Department_Name	Varchar	No	No
Department_Desc	Varchar	Yes	No
Department_Start_Date	Date	No	No
Department_End_Date	Date	Yes	No
Department_Set_Code	Varchar	No	No

Primary Key-Department_Id

12. MANAGER

Attribute_Name	Type	Null Allowed	Unique
Manager_Id	Number	No	Yes
Manager_Name	Varchar	No	No
Employee_Id	Number	No	Yes
Store_Id	Number	No	Yes

Primary Key-Manager_Id

Foreign Key- Employee_Id, Store_Id

13. ADDRESS

Attribute_Name	Type	Null Allowed	Unique
Address_Id	Number	No	Yes
Street	Varchar	No	No
Apt	Varchar	No	No
City	Varchar	No	No
State	Varchar	No	No
Zip	Number	No	No

Primary Key-Address_Id

PHYSICAL DATABASE DESIGN

After creating the Entity-Relationship Diagram (ERD) for the database, we created few of the important tables in Oracle Database. We used CREATE query to create the tables with attributes. Below are the screenshots of the tables and their attributes created in the database using SQL.

I. Customer

```
CREATE TABLE CUSTOMERS(
Customer_ID NUMBER,
First_Name VARCHAR(255),
Last_Name VARCHAR(255),
Phone_Number VARCHAR(255),
ZIP_CODE VARCHAR(255),
LOYALTY_ID VARCHAR(255));
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CUSTOMER_ID	NUMBER	Yes	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(255 BYTE)	Yes	(null)	2	(null)
3	LAST_NAME	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
4	PHONE	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)
5	EMAIL	VARCHAR2(255 BYTE)	Yes	(null)	5	(null)
6	ZIP	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)
7	LOYALTY_ID	VARCHAR2(255 BYTE)	Yes	(null)	7	(null)

Constraints:

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	CUST_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	CUST_UNIQ	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

II. Employee

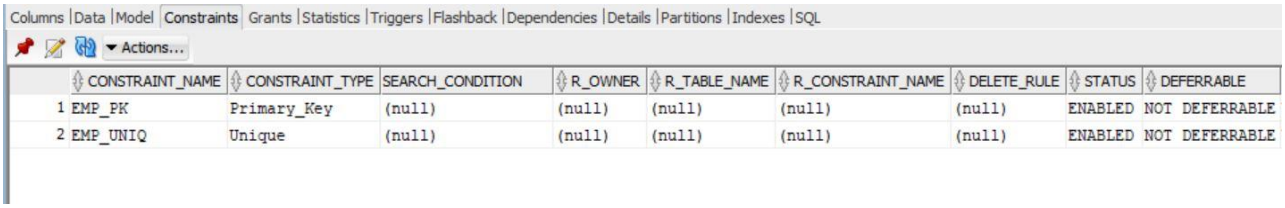
```
CREATE TABLE EMPLOYEE
  (    EMPLOYEE_ID NUMBER,
       FIRST_NAME VARCHAR2(255 BYTE),
       LAST_NAME VARCHAR2(255 BYTE),
       PHONE VARCHAR2(255 BYTE),
       EMAIL VARCHAR2(255 BYTE),
       GENDER VARCHAR2(255 BYTE),
       ADDRESS_ID NUMBER,
       START_DATE DATE,
       END_DATE DATE,
       SALARY_TYPE NUMBER,
       DEPT_ID NUMBER,
       JOB_ID NUMBER,
       SSN VARCHAR2(255 BYTE)
  );
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYEE_ID	NUMBER	Yes	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(255 BYTE)	Yes	(null)	2	(null)
3	LAST_NAME	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
4	PHONE	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)
5	EMAIL	VARCHAR2(255 BYTE)	Yes	(null)	5	(null)
6	GENDER	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)
7	ADDRESS_ID	NUMBER	Yes	(null)	7	(null)
8	START_DATE	DATE	Yes	(null)	8	(null)
9	END_DATE	DATE	Yes	(null)	9	(null)
10	SALARY_TYPE	NUMBER	Yes	(null)	10	(null)
11	DEPT_ID	NUMBER	Yes	(null)	11	(null)
12	JOB_ID	NUMBER	Yes	(null)	12	(null)
13	SSN	VARCHAR2(255 BYTE)	Yes	(null)	13	(null)

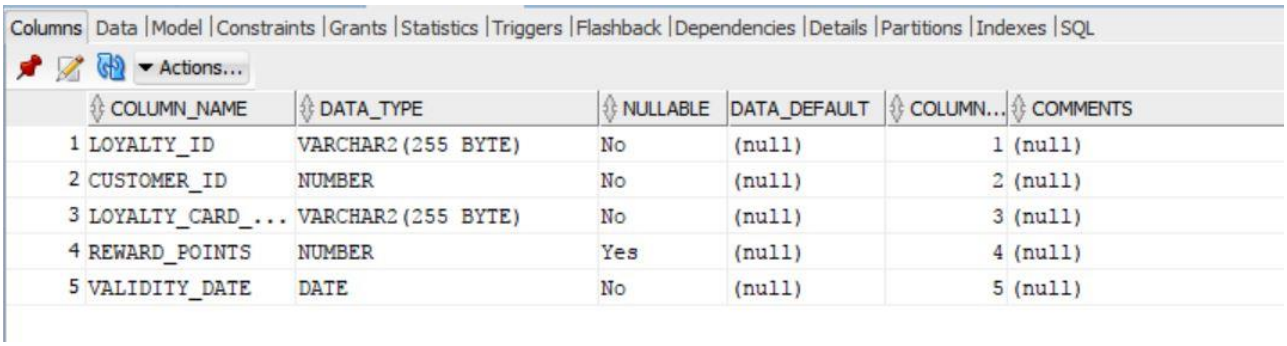
Constraints:



	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	EMP_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	EMP_UNIQ	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

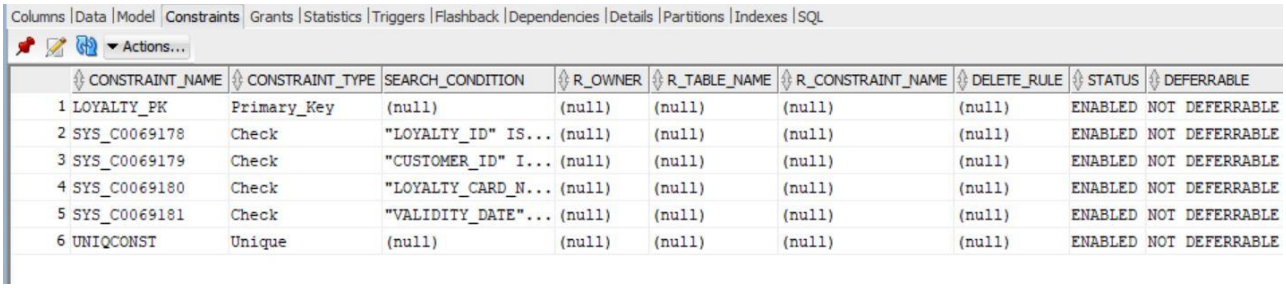
III. Loyalty

```
CREATE TABLE LOYALTY
(Loyalty_Id VARCHAR(255),
Customer_ID NUMBER,
Loyalty_Card_Number VARCHAR(255),
Reward_Points NUMBER,
Validity_Date DATE);
```



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN...	COMMENTS
1	LOYALTY_ID	VARCHAR2(255 BYTE)	No	(null)	1	(null)
2	CUSTOMER_ID	NUMBER	No	(null)	2	(null)
3	LOYALTY_CARD_...	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4	REWARD_POINTS	NUMBER	Yes	(null)	4	(null)
5	VALIDITY_DATE	DATE	No	(null)	5	(null)

Constraints:



	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	LOYALTY_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	SYS_C0069178	Check	"LOYALTY_ID" IS...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
3	SYS_C0069179	Check	"CUSTOMER_ID" I...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
4	SYS_C0069180	Check	"LOYALTY_CARD_N...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
5	SYS_C0069181	Check	"VALIDITY_DATE"...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
6	UNIQCONST	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

IV. Payment

```
CREATE TABLE PAYMENT
  (PAYMENT_ID NUMBER,
   SALE_ID NUMBER,
   PAYMENT_TYPE VARCHAR2(255 BYTE),
   CARD_TYPE VARCHAR2(255 BYTE),
   CARD_NUMBER VARCHAR2(255 BYTE),
   CARD_EXPIRY_DATE DATE
  );
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENT_ID	NUMBER	Yes	(null)	1	(null)
2	SALE_ID	NUMBER	Yes	(null)	2	(null)
3	PAYMENT_TYPE	VARCHAR2(255 BYTE)	Yes	(null)	3	(null)
4	CARD_TYPE	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)
5	CARD_NUMBER	VARCHAR2(255 BYTE)	Yes	(null)	5	(null)
6	CARD_EXPIRY_DATE	DATE	Yes	(null)	6	(null)

Constraints:

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	SYS_C0069429	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	SYS_C0069430	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

V. Vendor

```
CREATE TABLE VENDOR
(VENDOR_ID NUMBER,
STORE_ID NUMBER,
VENDOR_COMPANY_NAME VARCHAR2(255 BYTE) NOT NULL,
VENDOR_ADDRESS  VARCHAR2(255 BYTE),
VENDOR_CITY VARCHAR2(255 BYTE),
VENDOR_ZIP_CODE NUMBER NOT NULL,
VENDOR_PHONE NUMBER NOT NULL);
```

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN...	COMMENTS
1	VENDOR_ID	NUMBER	No	(null)	1	(null)
2	STORE_ID	NUMBER	Yes	(null)	2	(null)
3	VENDOR_COMPAN...	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4	VENDOR_ADDRESS	VARCHAR2(255 BYTE)	No	(null)	4	(null)
5	VENDOR_CITY	VARCHAR2(255 BYTE)	Yes	(null)	5	(null)
6	VENDOR_ZIP_CODE	NUMBER	No	(null)	6	(null)
7	VENDOR_PHONE	NUMBER	No	(null)	7	(null)

Constraints:

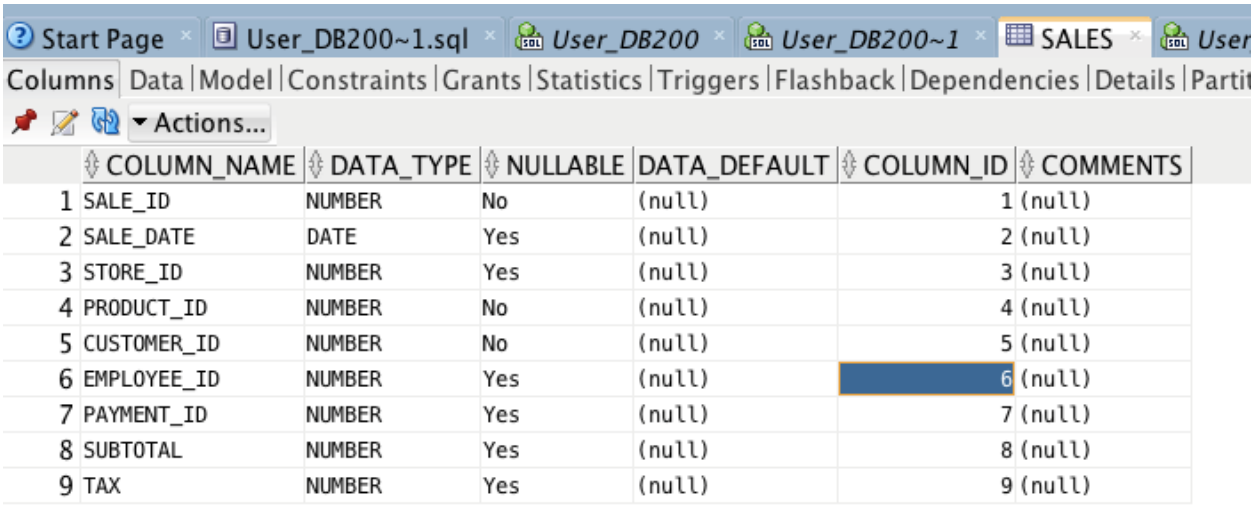
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	SYS_C0069185	Check	"VENDOR_COMPANY...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	SYS_C0069186	Check	"VENDOR_ADDRESS...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
3	SYS_C0069187	Check	"VENDOR_ZIP_COD...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
4	SYS_C0069188	Check	"VENDOR_PHONE" ...	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
5	SYS_C0069192	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
6	SYS_C0069197	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
7	UNIQUEVENDOR	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

Sales

```
CREATE TABLE SALES (
SALE_ID NUMBER NOT NULL,
SALE_DATE DATE,
STORE_ID NUMBER,
PRODUCT_ID NUMBER NOT NULL,
CUSTOMER_ID NUMBER NOT NULL,
EMPLOYEE_ID NUMBER,
PAYMENT_ID NUMBER,
SUBTOTAL NUMBER,
TAX NUMBER);
```



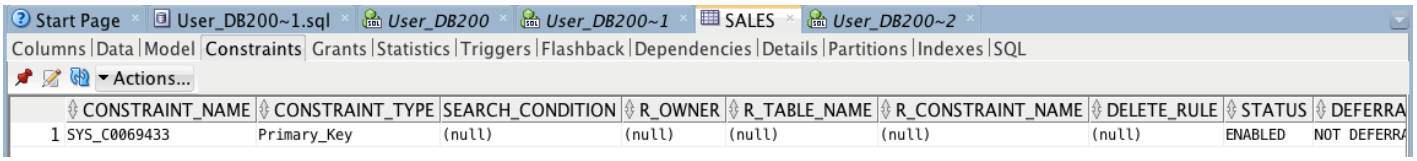
Start Page | User_DB200~1.sql | User_DB200 | User_DB200~1 | SALES | User

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partit

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SALE_ID	NUMBER	No	(null)	1	(null)
2	SALE_DATE	DATE	Yes	(null)	2	(null)
3	STORE_ID	NUMBER	Yes	(null)	3	(null)
4	PRODUCT_ID	NUMBER	No	(null)	4	(null)
5	CUSTOMER_ID	NUMBER	No	(null)	5	(null)
6	EMPLOYEE_ID	NUMBER	Yes	(null)	6	(null)
7	PAYMENT_ID	NUMBER	Yes	(null)	7	(null)
8	SUBTOTAL	NUMBER	Yes	(null)	8	(null)
9	TAX	NUMBER	Yes	(null)	9	(null)

Constraints:



Start Page | User_DB200~1.sql | User_DB200 | User_DB200~1 | SALES | User_DB200~2

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

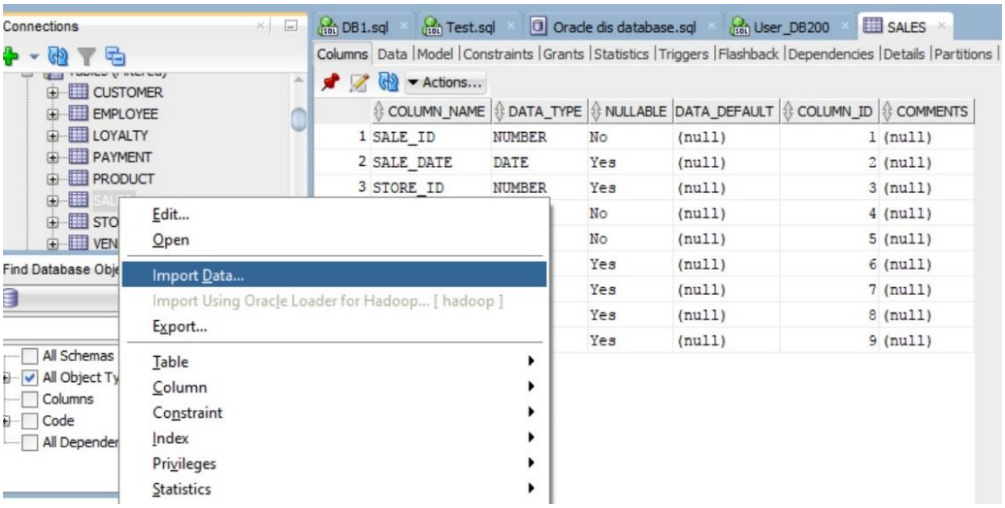
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRA
1	SYS_C0069433	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA

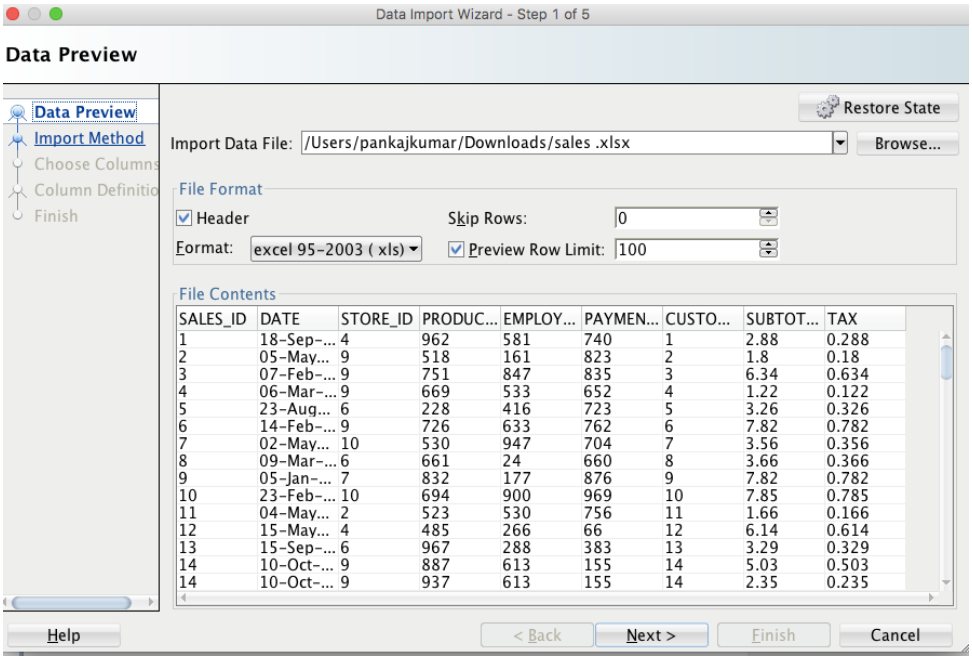
Data Creation and Import

For creating sample data for our project, we used an online data generator. We have populated six of the tables for now. After generating sample data in an excel file, next step is to import that data into the database. Steps for that are described below.

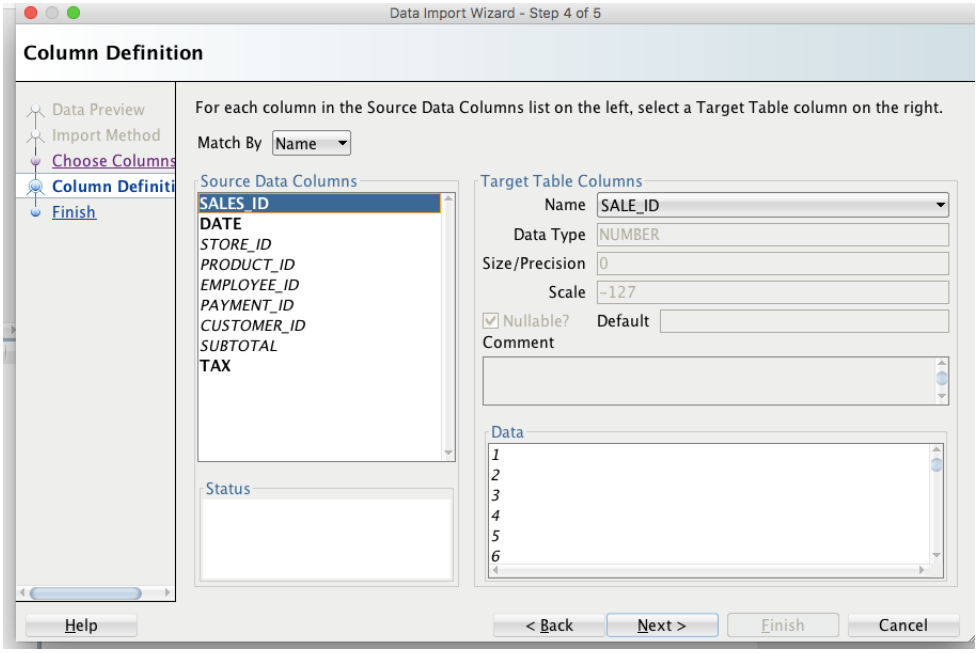
1. Right-click on the table in which data is to be imported and choose Import data.



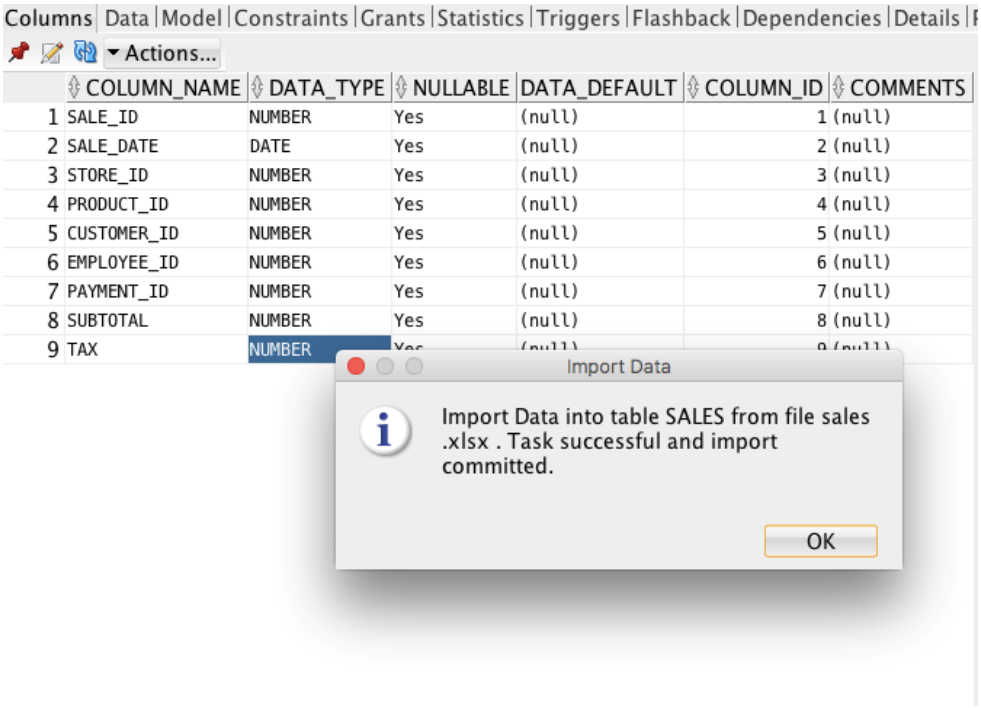
2. Choose the location of excel file which contains the data to be imported.



3. After choosing the Import Method, Choose the columns and check the column definition. Column definition is to verify if the data from a column name is associated with the right attribute in the table.



4. The final step is to click on Finish and the data gets imported.

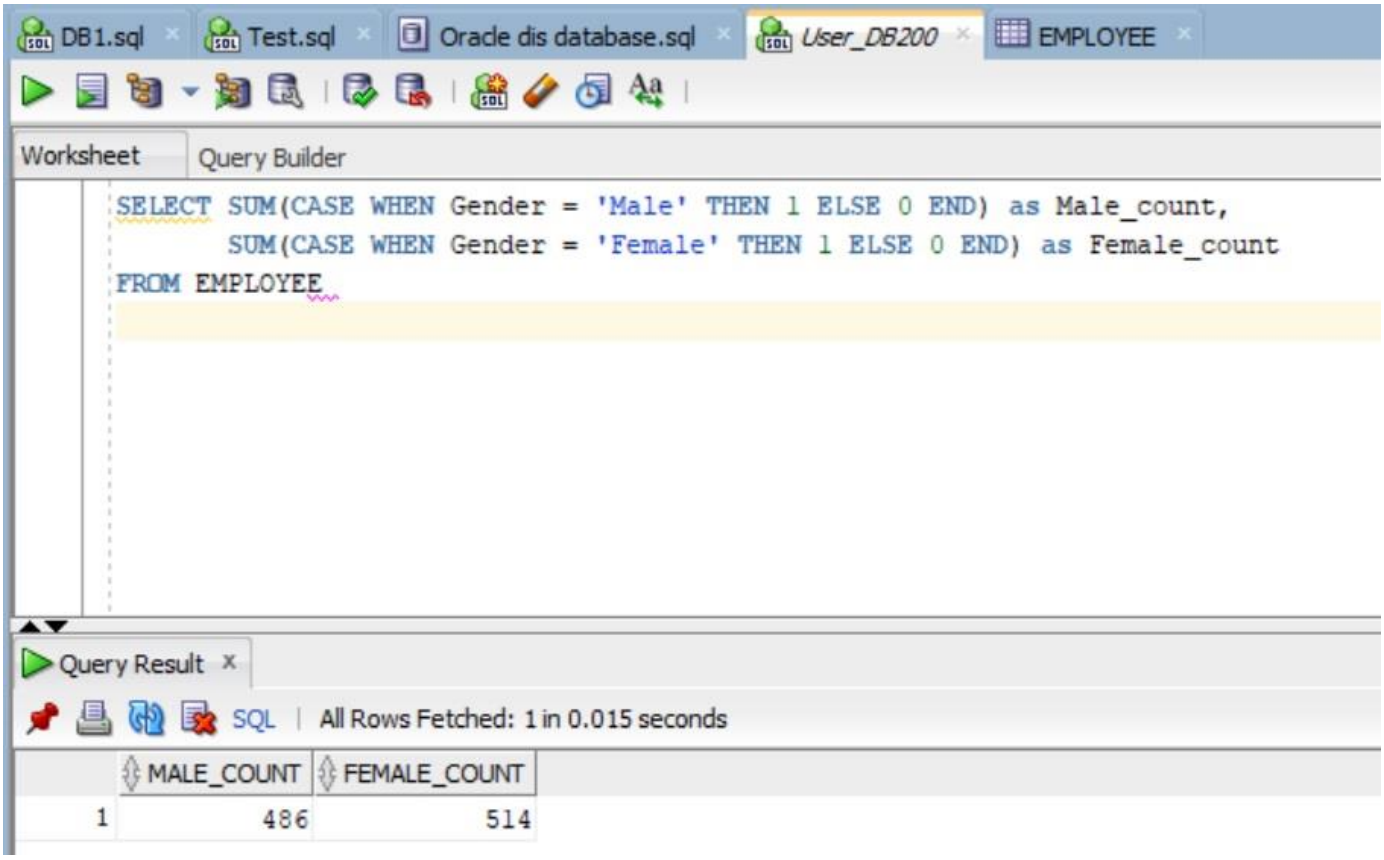


SQL QUERIES

Given below are few interesting SQL queries and their results.

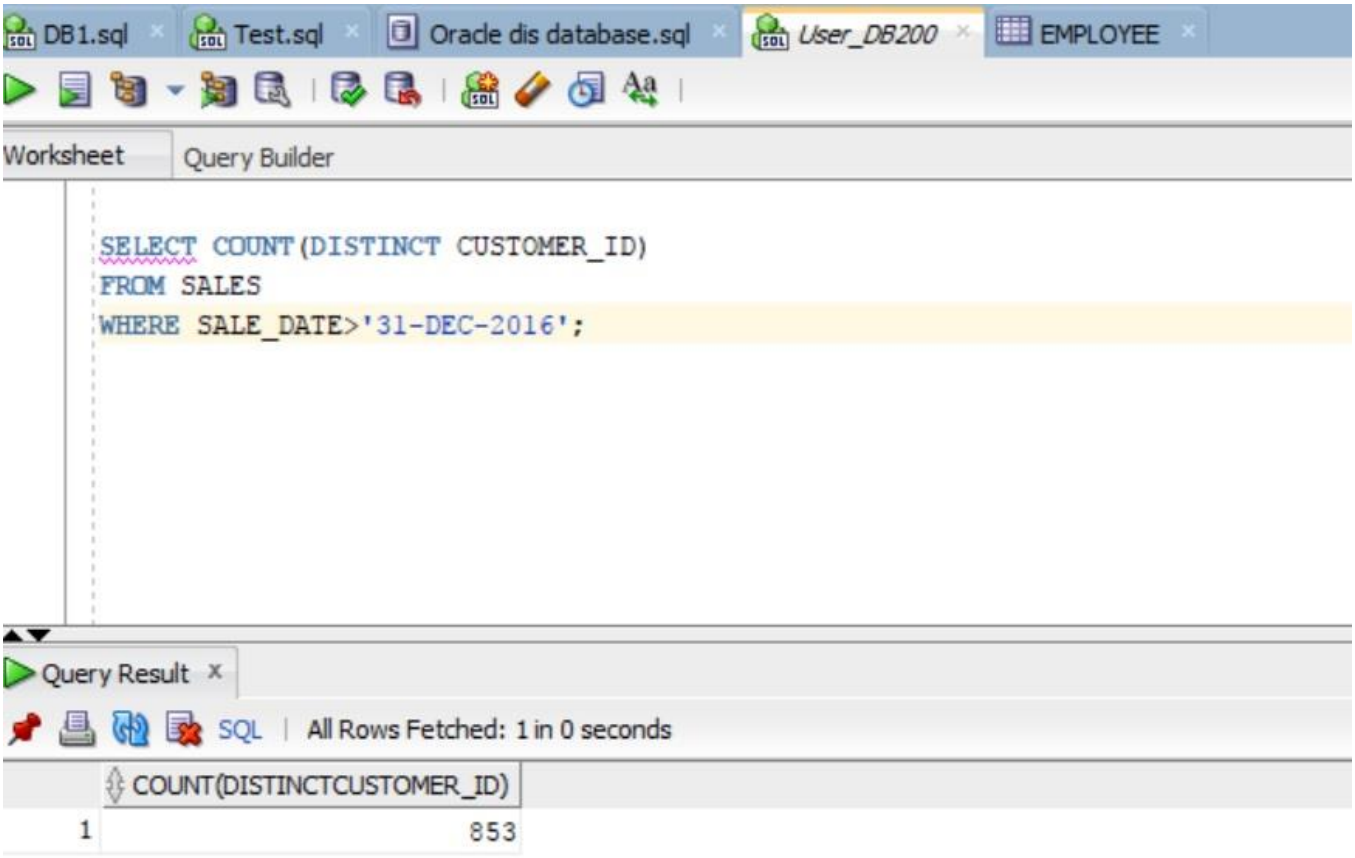
1. To find the count of Male and Female employees.

```
SELECT SUM(CASE WHEN Gender = 'Male' THEN 1 ELSE 0 END) as Male_count,
SUM(CASE WHEN Gender = 'Female' THEN 1 ELSE 0 END) as Female_count
FROM EMPLOYEE
```



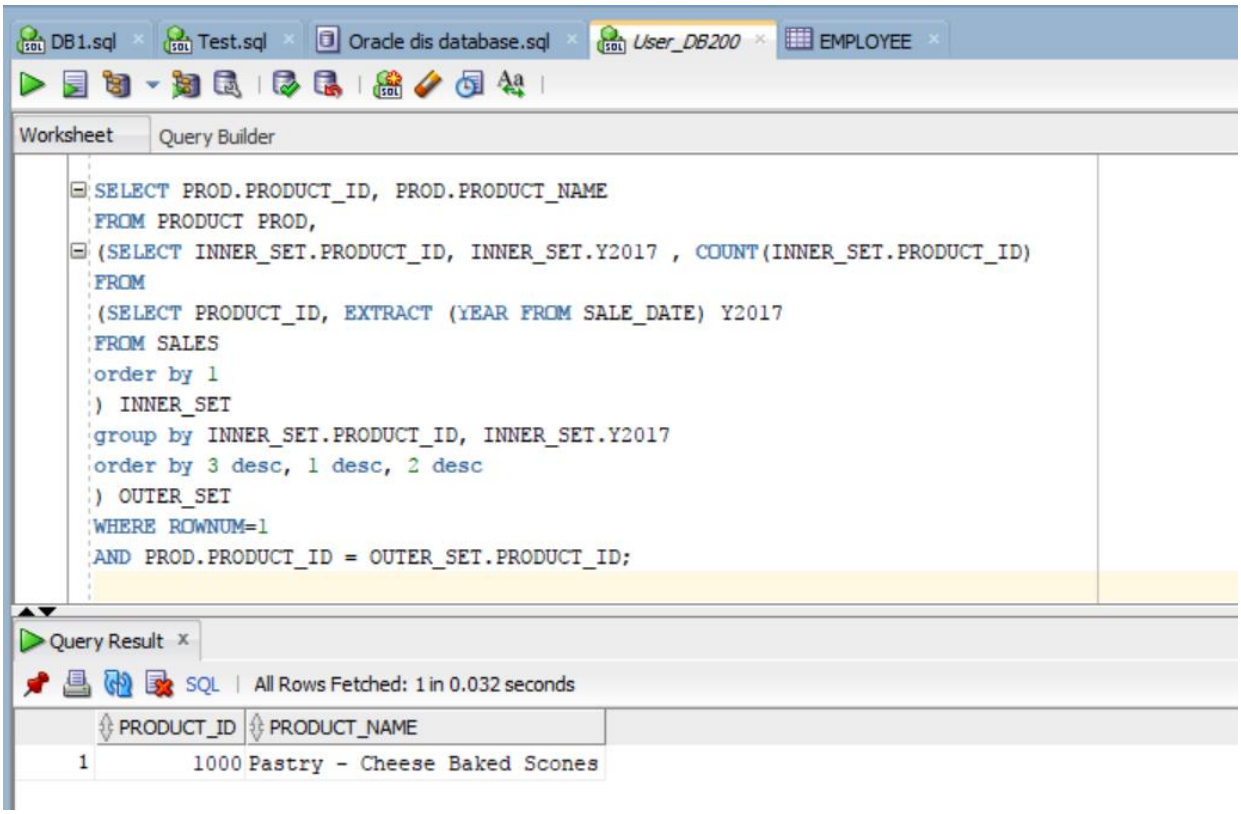
2. To find the total number of sales that happened in the year 2017.

```
SELECT COUNT(DISTINCT CUSTOMER_ID)
FROM SALES
WHERE SALE_DATE>'31-DEC-2016';
```



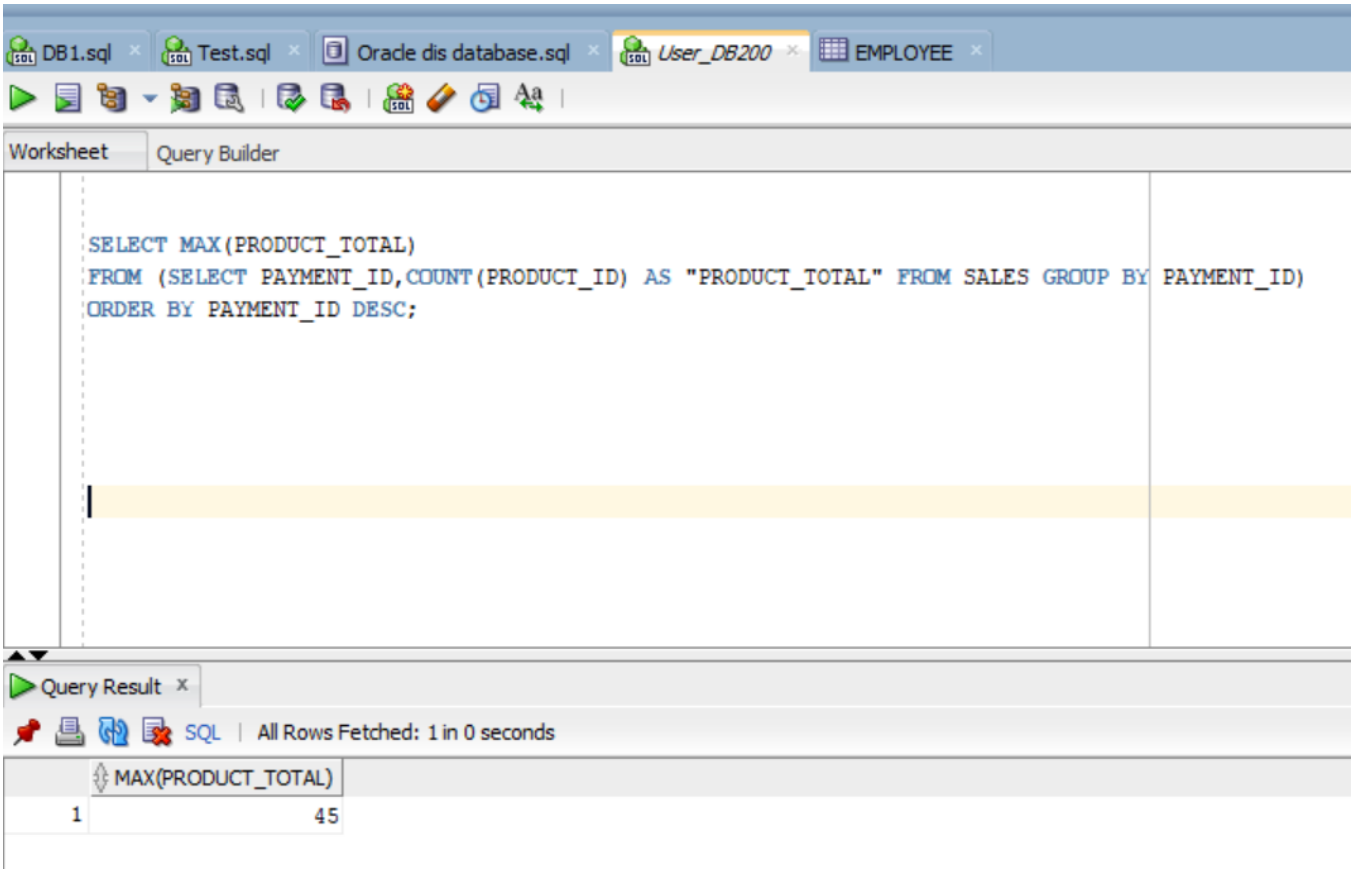
3. To find the highest selling product of 2017.

```
SELECT PROD.PRODUCT_ID, PROD.PRODUCT_NAME

FROM PRODUCT PROD, (SELECT INNER_SET.PRODUCT_ID, INNER_SET.Y2017 ,
COUNT(INNER_SET.PRODUCT_ID)

FROM(SELECT PRODUCT_ID, EXTRACT (YEAR FROM SALE_DATE) Y2017

FROM SALES

order by 1) INNER_SET

group by INNER_SET.PRODUCT_ID, INNER_SET.Y2017

order by 3 desc, 1 desc, 2 desc) OUTER_SET

WHERE ROWNUM=1

AND PROD.PRODUCT_ID = OUTER_SET.PRODUCT_ID;
```



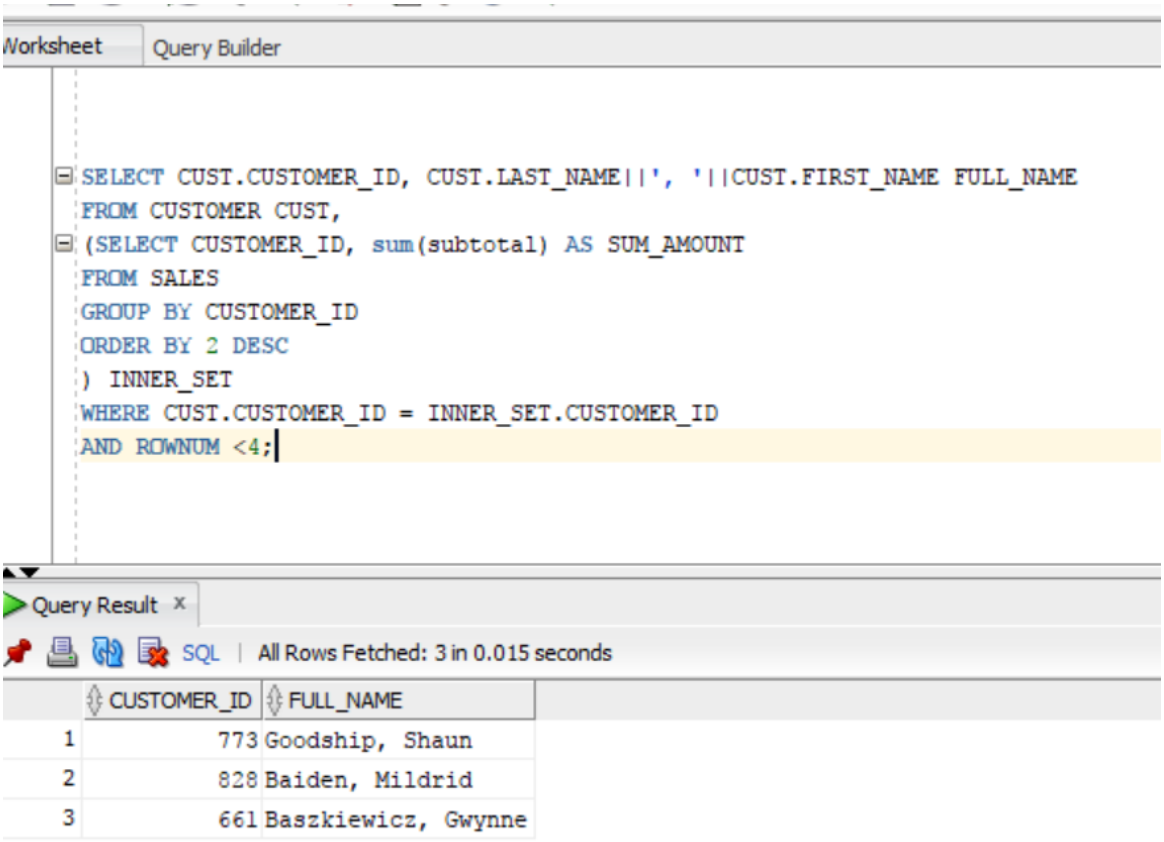
4. To find the maximum number of products bought in one purchase.

```
SELE CT MAX(PRODUCT_TOTAL)

FROM (SELECT PAYMENT_ID,COUNT(PRODUCT_ID) AS "PRODUCT_TOTAL"
FROM SALES GROUP BY PAYMENT_ID) ORDER BY PAYMENT_ID DESC;
```



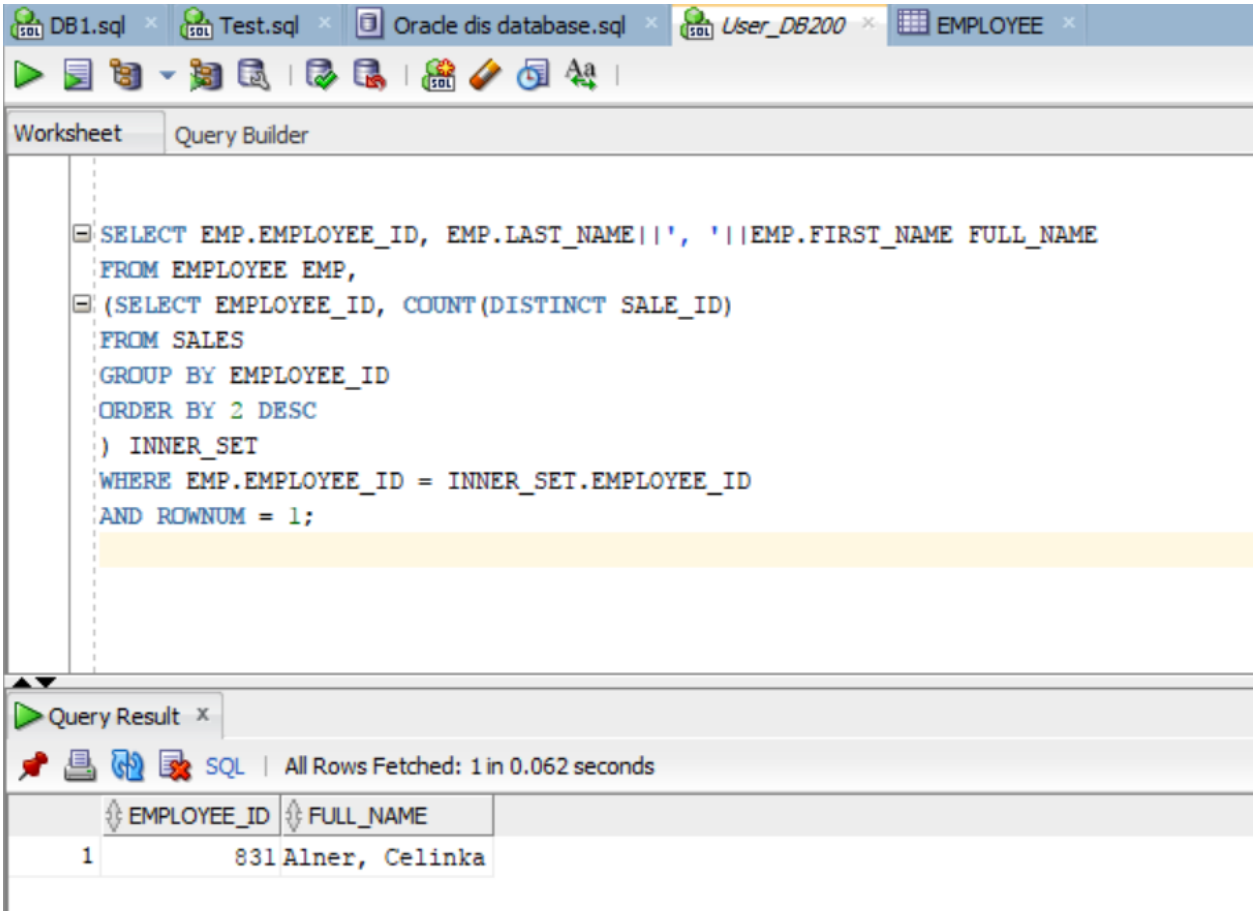
5. To find the top 3 customers who paid the highest bill.

```
SELECT CUST.CUSTOMER_ID, CUST.LAST_NAME||', '||CUST.FIRST_NAME FULL_NAME

FROM CUSTOMER CUST,

(SELECT CUSTOMER_ID, sum(subtotal) AS SUM_AMOUNT

FROM SALES

GROUP BY CUSTOMER_ID

ORDER BY 2 DESC) INNER_SET

WHERE CUST.CUSTOMER_ID = INNER_SET.CUSTOMER_ID

AND ROWNUM <4;
```



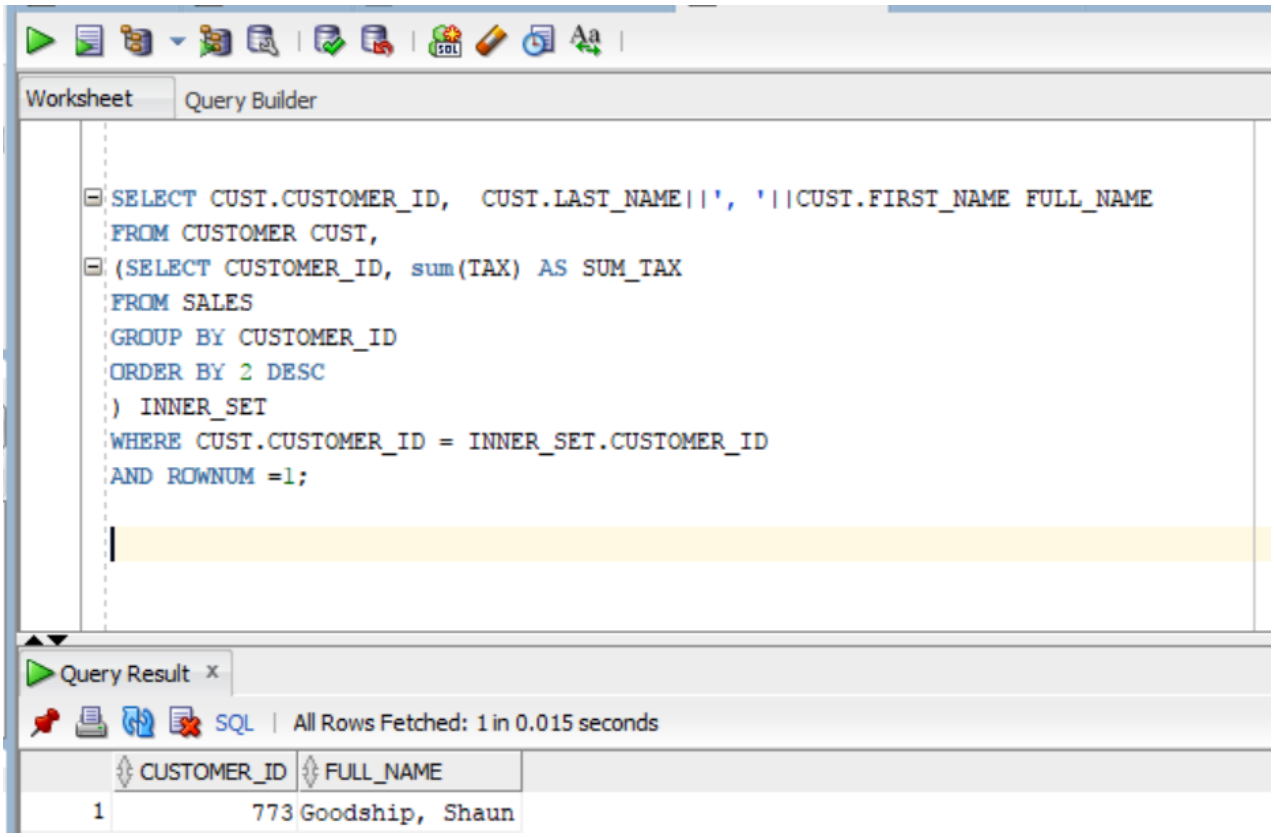
6. To find the employee who did maximum billing.

```
SELECT EMP.EMPLOYEE_ID, EMP.LAST_NAME||', '||EMP.FIRST_NAME FULL_NAME

FROM EMPLOYEE EMP,

(SELECT EMPLOYEE_ID, COUNT(DISTINCT SALE_ID)

FROM SALES

GROUP BY EMPLOYEE_ID

ORDER BY 2 DESC) INNER_SET

WHERE EMP.EMPLOYEE_ID = INNER_SET.EMPLOYEE_ID

AND ROWNUM = 1;
```



7. To find the customer who paid the highest tax amount.

```
SELECT CUST.CUSTOMER_ID, CUST.LAST_NAME||', '||CUST.FIRST_NAME FULL_NAME
FROM CUSTOMER CUST,
(SELECT CUSTOMER_ID, sum(TAX) AS SUM_TAX
FROM SALES
GROUP BY CUSTOMER_ID
ORDER BY 2 DESC) INNER_SET
WHERE CUST.CUSTOMER_ID = INNER_SET.CUSTOMER_ID
AND ROWNUM =1;
```



8. To find the store with maximum sales.

SELECT STORES.STORE_ID, STORES.STORE_NAME

FROM STORE STORES,

(SELECT STORE_ID, sum(subtotal) AS SUM_AMOUNT

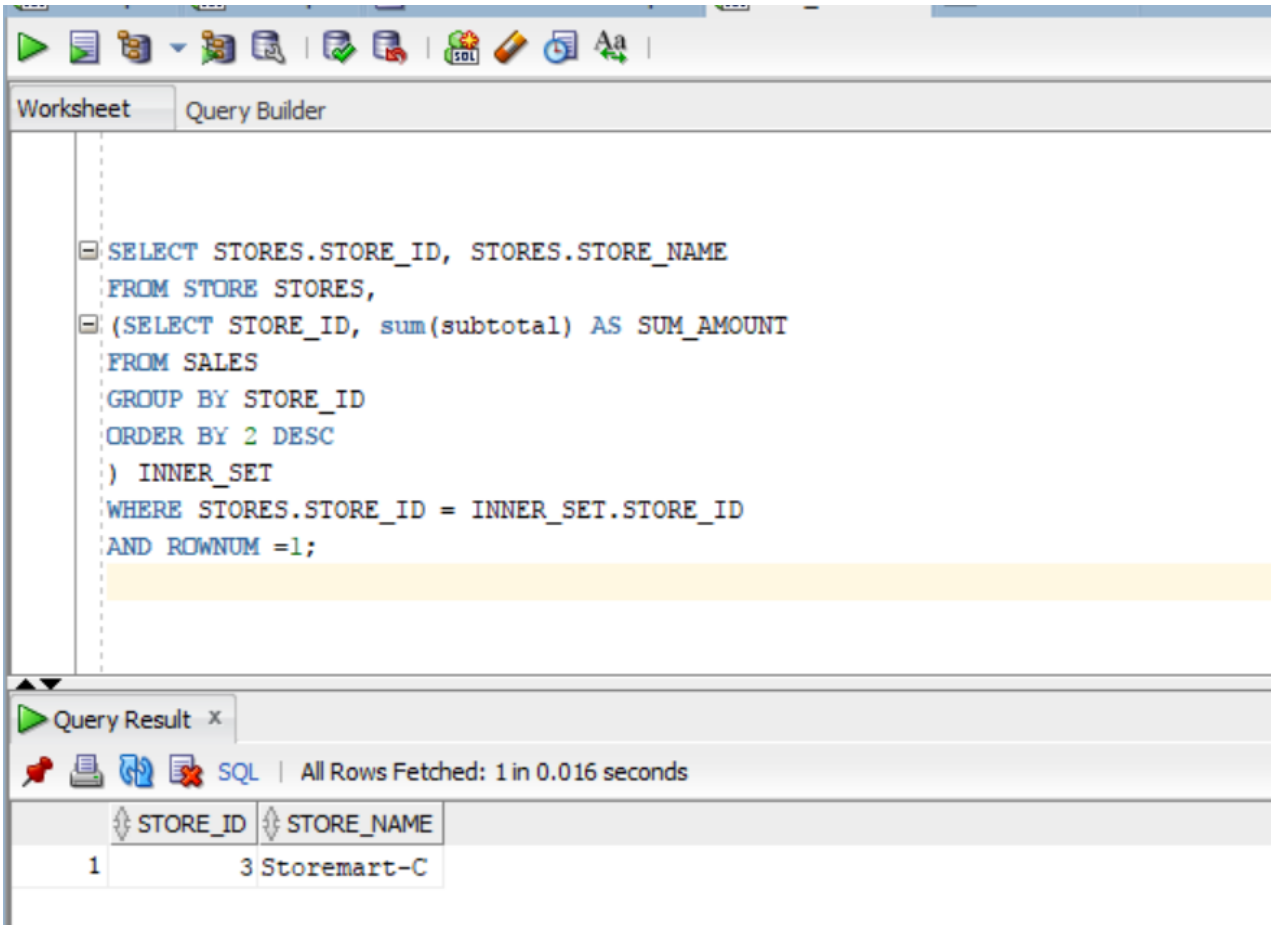
FROM SALES

GROUP BY STORE_ID

ORDER BY 2 DESC) INNER_SET

WHERE STORES.STORE_ID = INNER_SET.STORE_ID

AND ROWNUM =1;



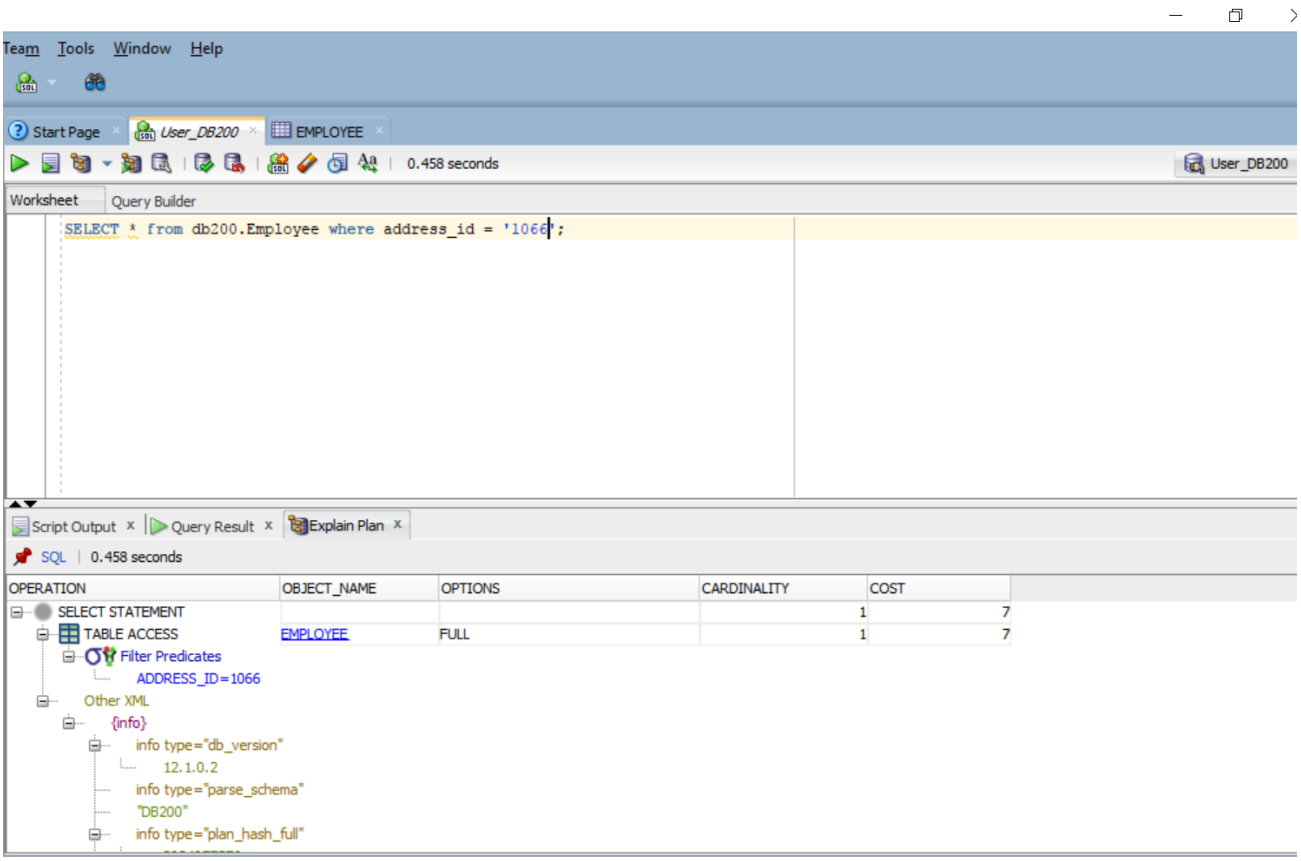
PERFORMANCE TUNING

INDEXING

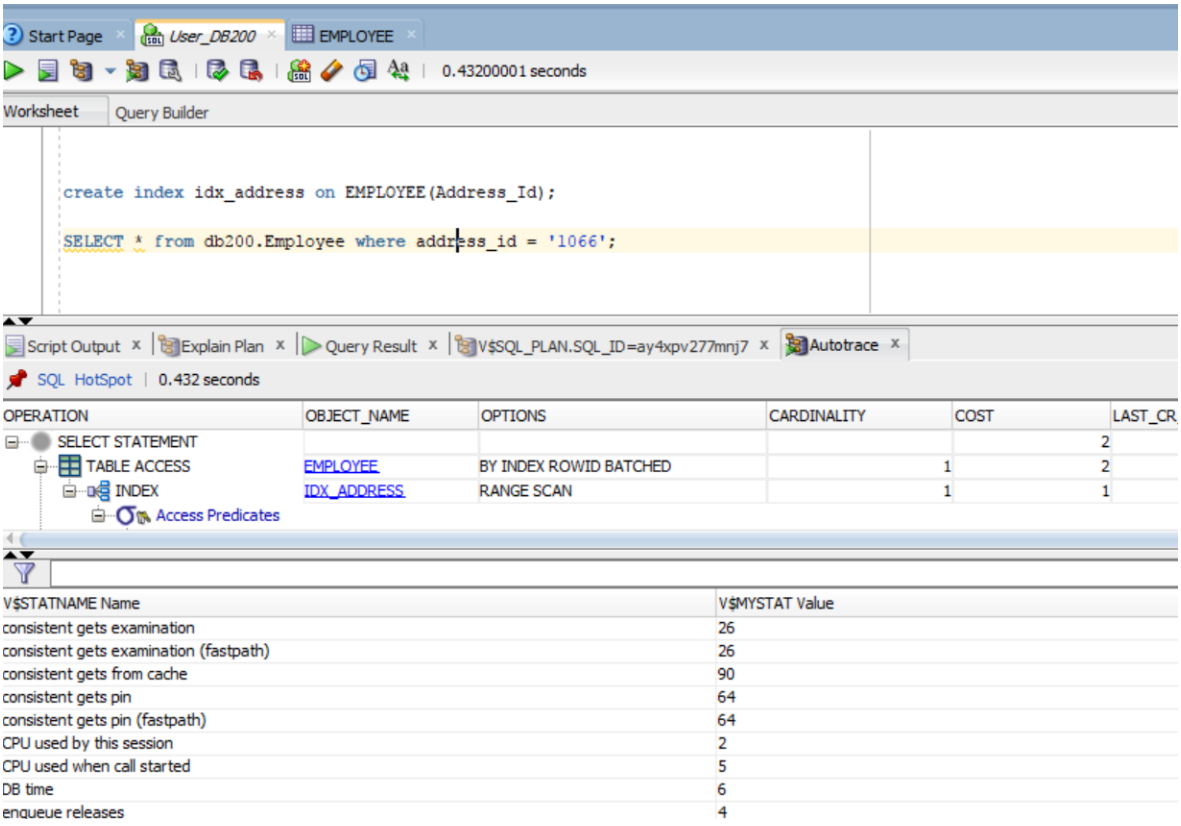
A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

SIMPLE INDEXING

Consider the following query on the Employee table:



After creating the index on the zip, we can observe that the cost of executing the query has decreased as well as the consistent gets are improved.

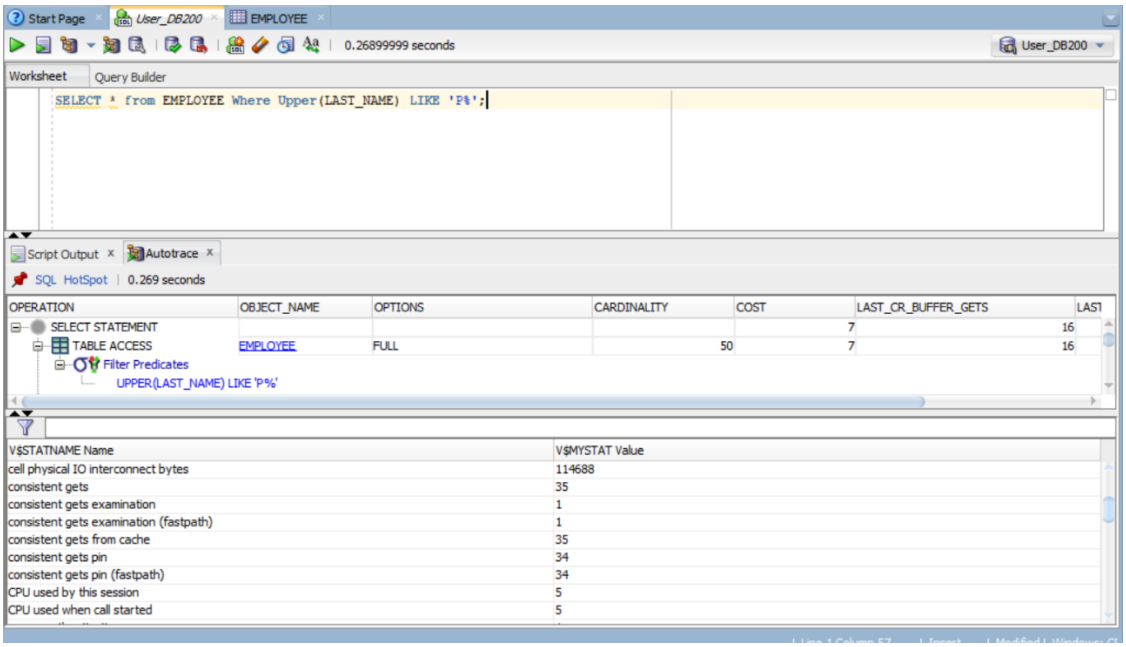


FUNCTION_BASED INDEXING

The default type of index is a b-tree index, which creates an index on one or two more columns. A function-based index is an index that is created on the results of a function or an expression. This helps in replacing unnecessary full table scans with faster index range scans.

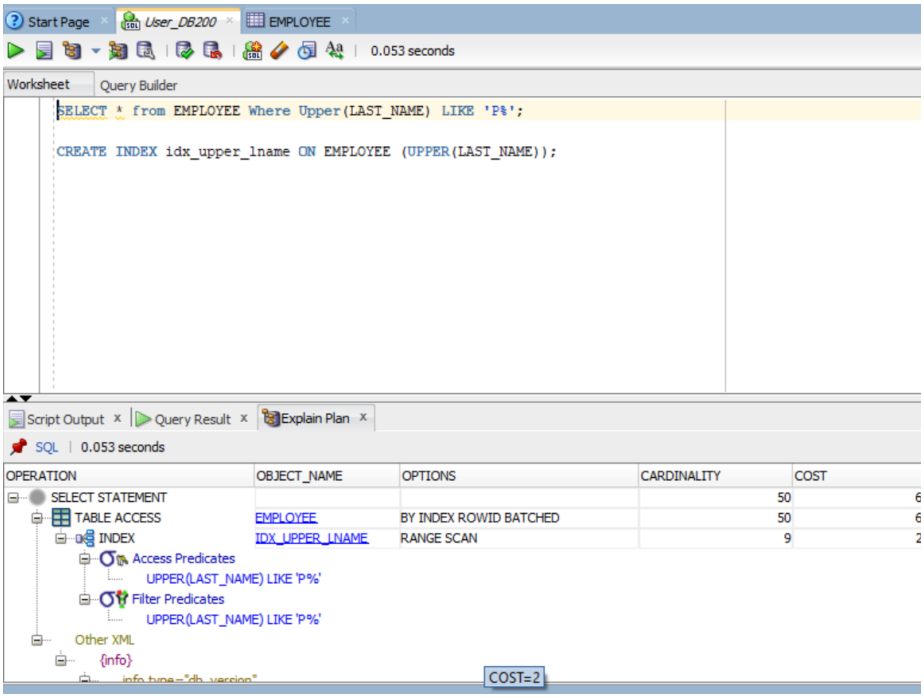
We will demonstrate with the following experiment:

Without function-based index, the cost of a query with a full scan is shown:



After creating the index using the following statement:

CREATE INDEX idx_upper_lname ON EMPLOYEE (UPPER(LAST_NAME));



The cost of the query has decreased after using the index.

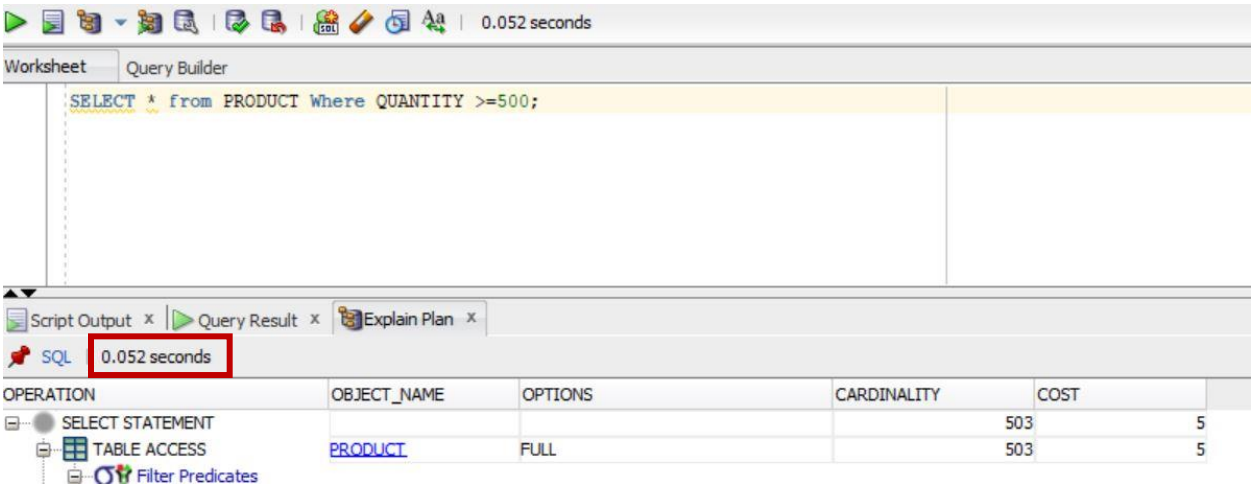
SQL TUNING

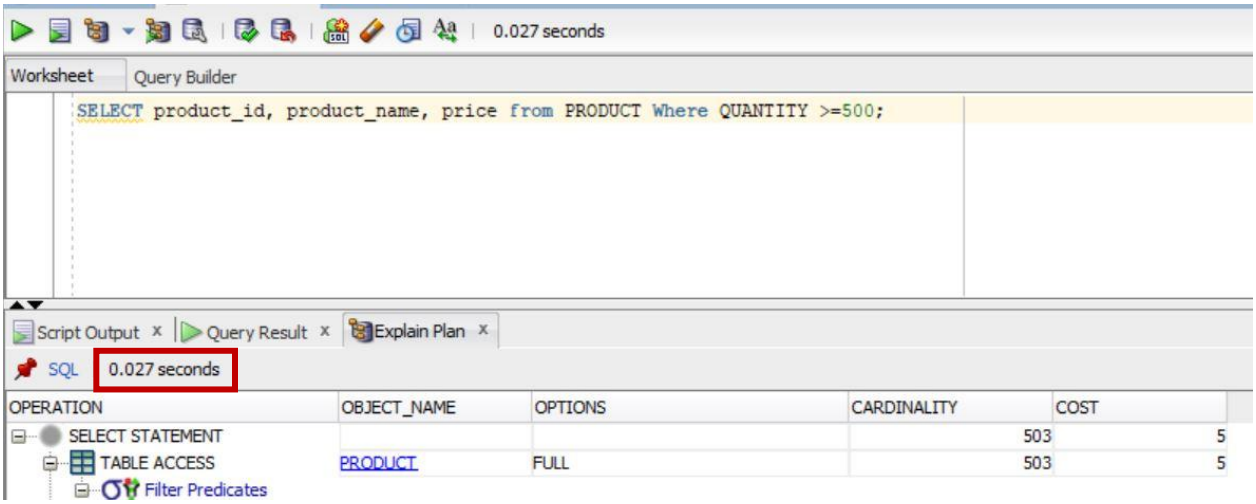
SQL tuning helps in the tuning of the system database performance. The objective of SQL tuning is to either reduce the response time for the end users or optimize the usage of resources required to generate results.

One method of SQL tuning is to improve SQL statement efficiency. It is easier to rewrite inefficient SQL statements if the purpose of performing the query is known.

1) Retrieval of results is faster when we use the actual names of the columns instead of an "*".

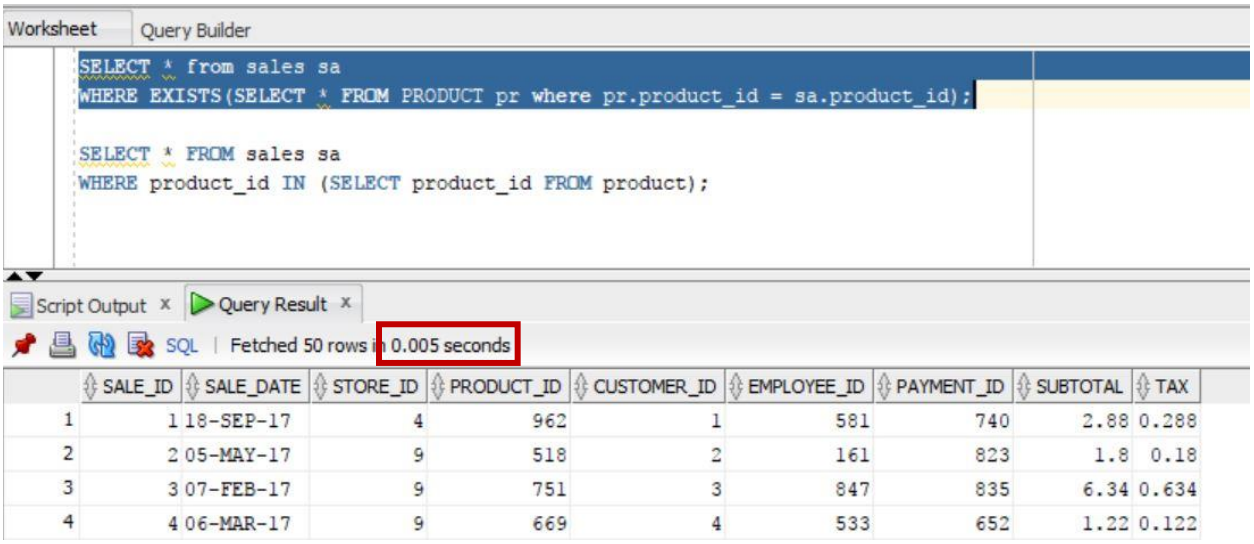
This is shown below:



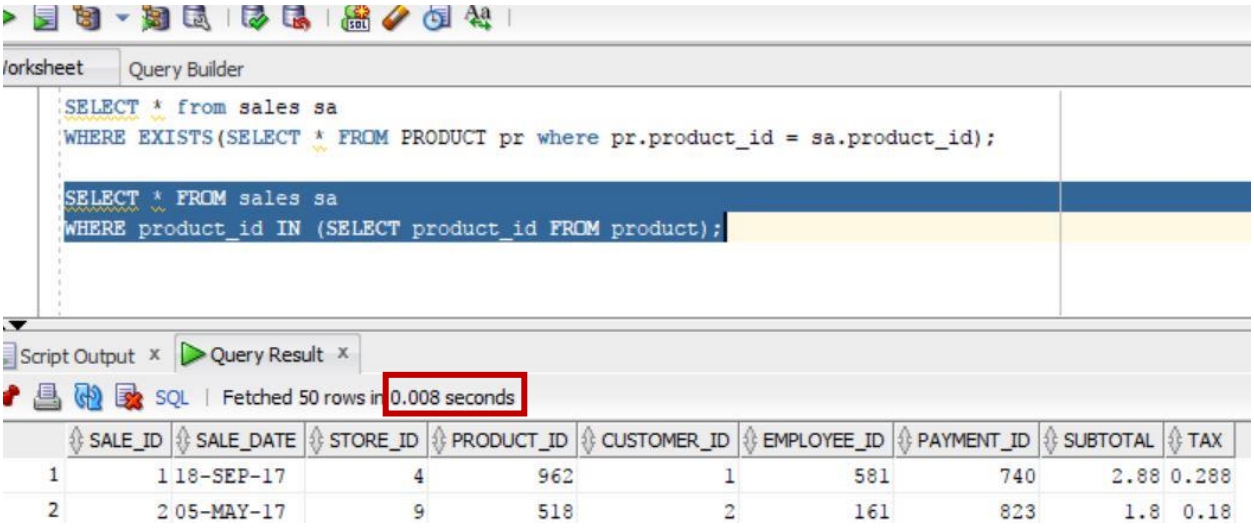


2) Use of Exists versus IN for subqueries

Consider the following examples where we will compare the time the query takes to fetch the results. It is better to use EXISTS when most of the filtering criteria is in the main query.



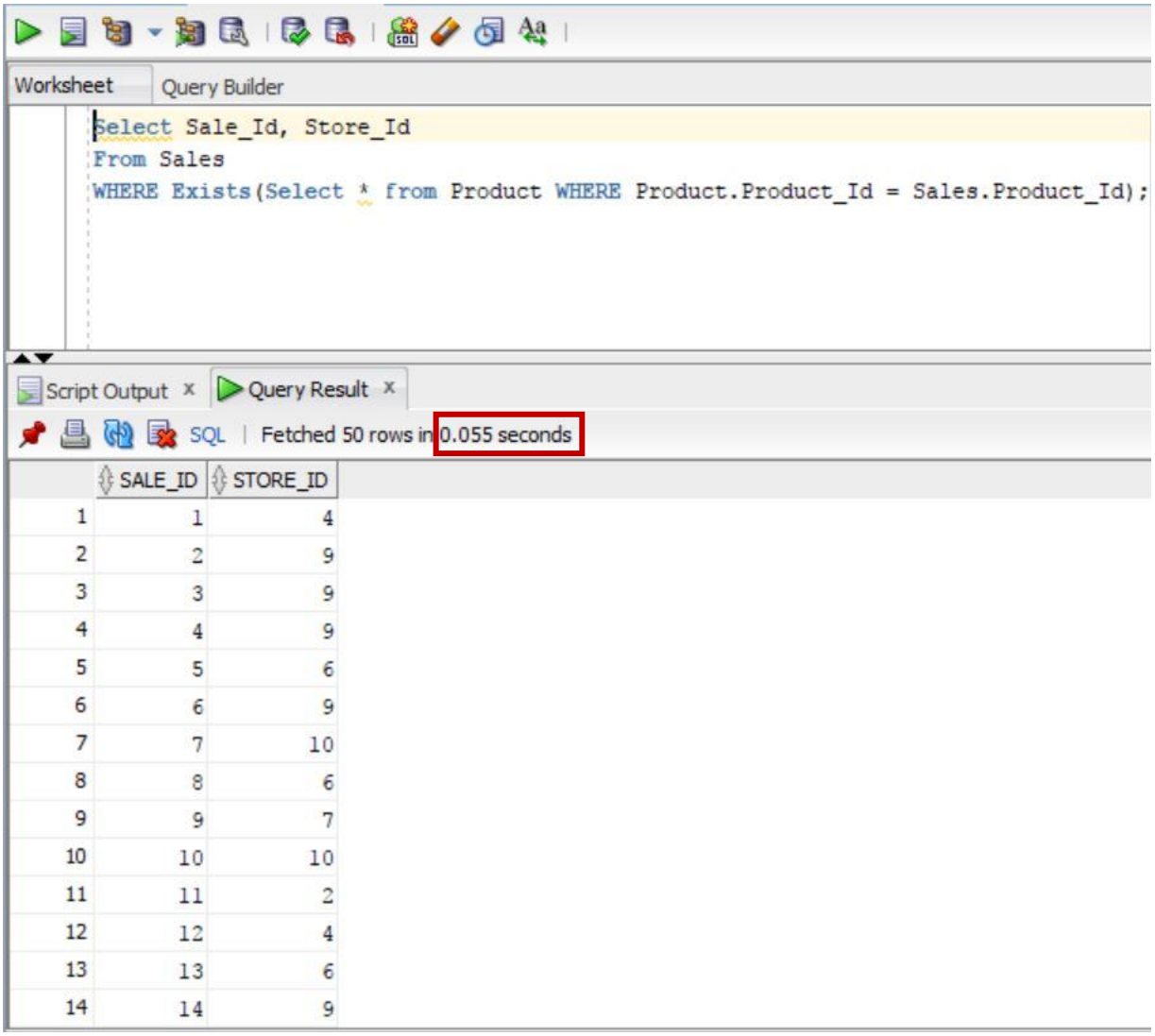
	SALE_ID	SALE_DATE	STORE_ID	PRODUCT_ID	CUSTOMER_ID	EMPLOYEE_ID	PAYMENT_ID	SUBTOTAL	TAX
1	1	18-SEP-17	4	962	1	581	740	2.88	0.288
2	2	05-MAY-17	9	518	2	161	823	1.8	0.18
3	3	07-FEB-17	9	751	3	847	835	6.34	0.634
4	4	06-MAR-17	9	669	4	533	652	1.22	0.122



	SALE_ID	SALE_DATE	STORE_ID	PRODUCT_ID	CUSTOMER_ID	EMPLOYEE_ID	PAYMENT_ID	SUBTOTAL	TAX
1	1	18-SEP-17	4	962	1	581	740	2.88	0.288
2	2	05-MAY-17	9	518	2	161	823	1.8	0.18

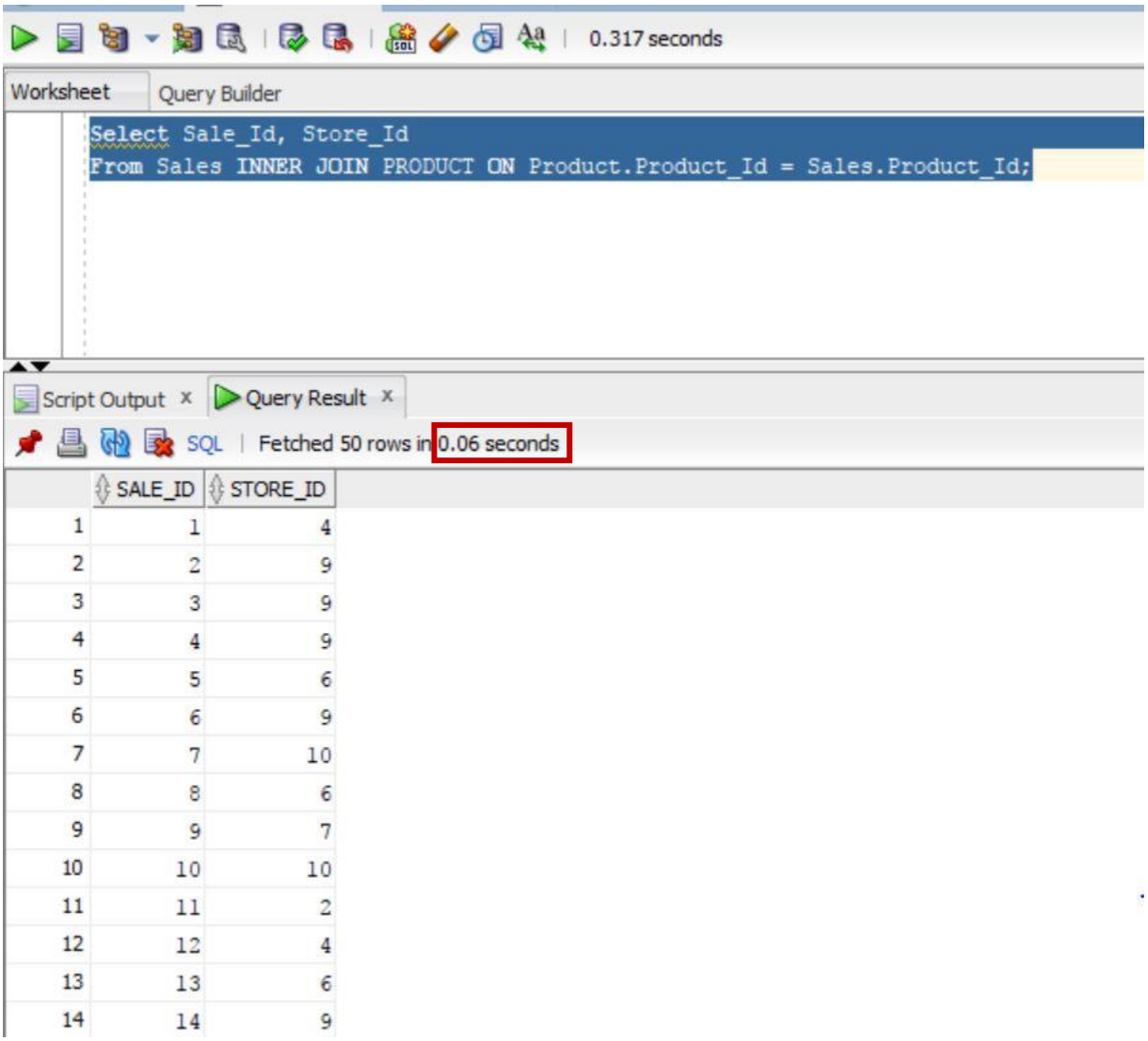
3) Reduce correlated subqueries

A correlated subquery is a subquery which depends on the outer query. It uses the results obtained from the outer query in its WHERE clause. Consider the following example.



	SALE_ID	STORE_ID
1	1	4
2	2	9
3	3	9
4	4	9
5	5	6
6	6	9
7	7	10
8	8	6
9	9	7
10	10	10
11	11	2
12	12	4
13	13	6
14	14	9

In the above case, the subquery runs once for each row and can cause inefficiency. Instead we can use a JOIN:



	SALE_ID	STORE_ID
1	1	4
2	2	9
3	3	9
4	4	9
5	5	6
6	6	9
7	7	10
8	8	6
9	9	7
10	10	10
11	11	2
12	12	4
13	13	6
14	14	9

4) Avoid wildcard characters at the beginning of a LIKE pattern.

Avoid using the LIKE pattern in the following way:

```
SELECT * FROM Payment WHERE Card_Type LIKE '%mas%';
```

The use of '%' wildcard prevents the database from using an index, if it exists. The database will perform a full scan and the query execution may slow down. Instead, the query can be rewritten as:

```
SELECT * FROM Payment WHERE Card_Type LIKE 'mas%';
```

5) Use of HAVING clause
HAVING clause is used to filter the values in a GROUP BY. It filters rows after all the rows are selected.

```
SELECT COUNT(Customer_ID), Zip
FROM Customer
GROUP BY Zip
HAVING COUNT(Customer_ID) > 5;
```

Instead of

```
SELECT COUNT(Customer_ID), zip
FROM Customer
WHERE Customer_ID > 5
GROUP BY zip;
```

PARALLEL PROCESSING

Parallel processing is used to reduce the response time of operations that are highly data-intensive. It is simply breaking down one process doing all the execution of a query into many sub processes which work at a same time. Parallel Execution reduces response time for operations such as which require

• Large table Scans, Joins

• Creation of Large Indexes

• Creation of Large Tables

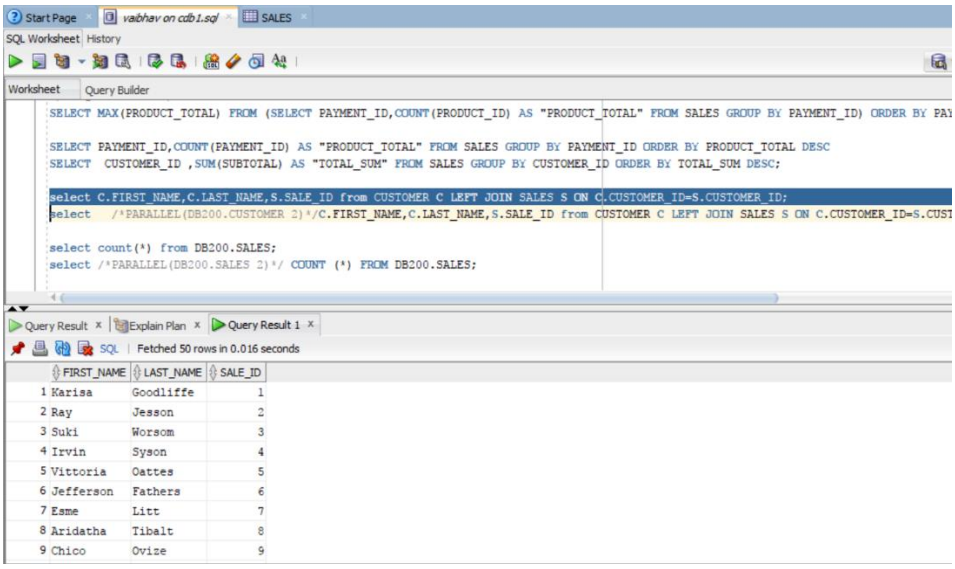
• Bulk Inserts, Updates, Merge, and Delete.

Parallel processing is suitable for long-running operations in low-concurrency environments. Parallel processing is less suitable for OLTP style databases.

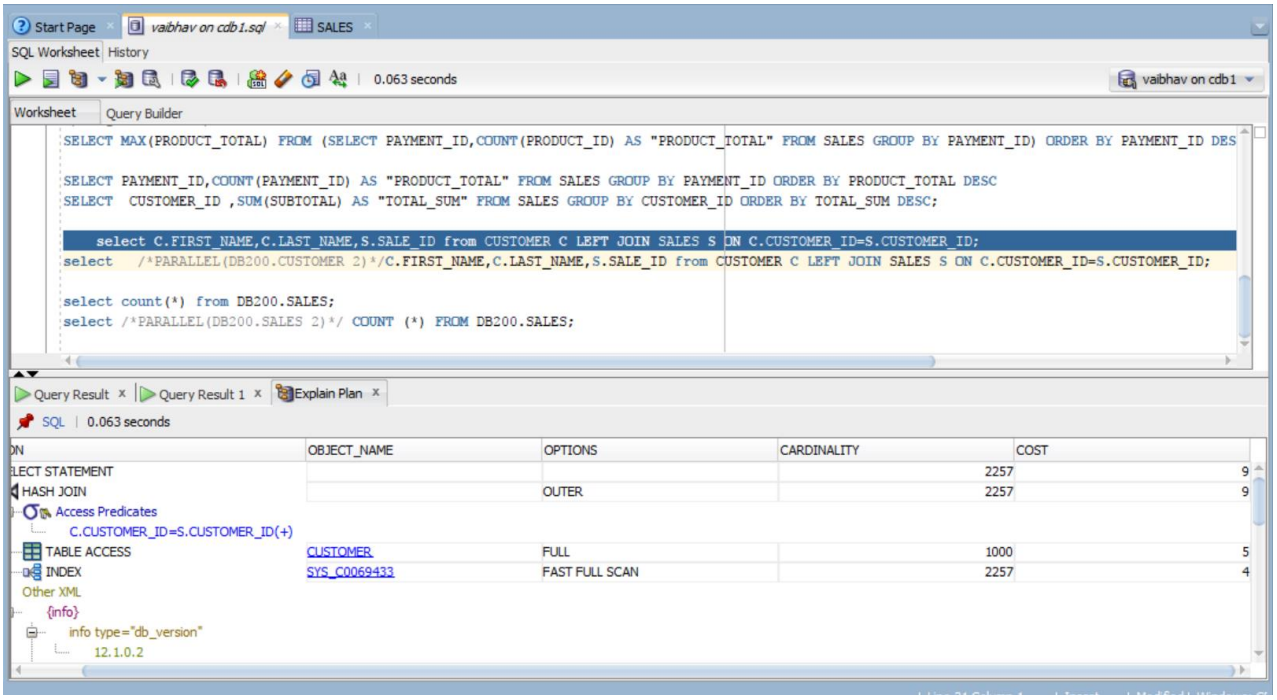
Like for example if we have to fetch the details of a customer's name and its corresponding sales id from sales table it will have a join operation which will be a data intensive operation.

So, let's Perform the execution without parallel processing.

SELECT C.FIRST_NAME,C.LAST_NAME,S.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=S.CUSTOMER_ID;

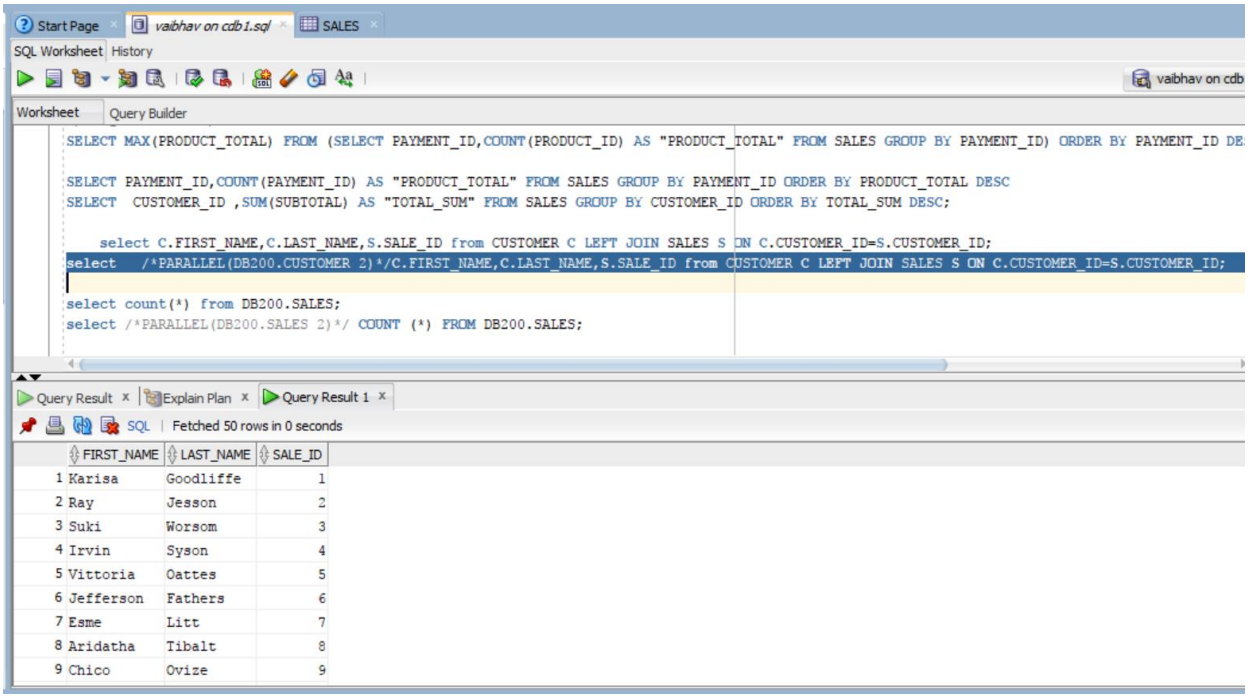


Its Explain Plan and cost are as follows-

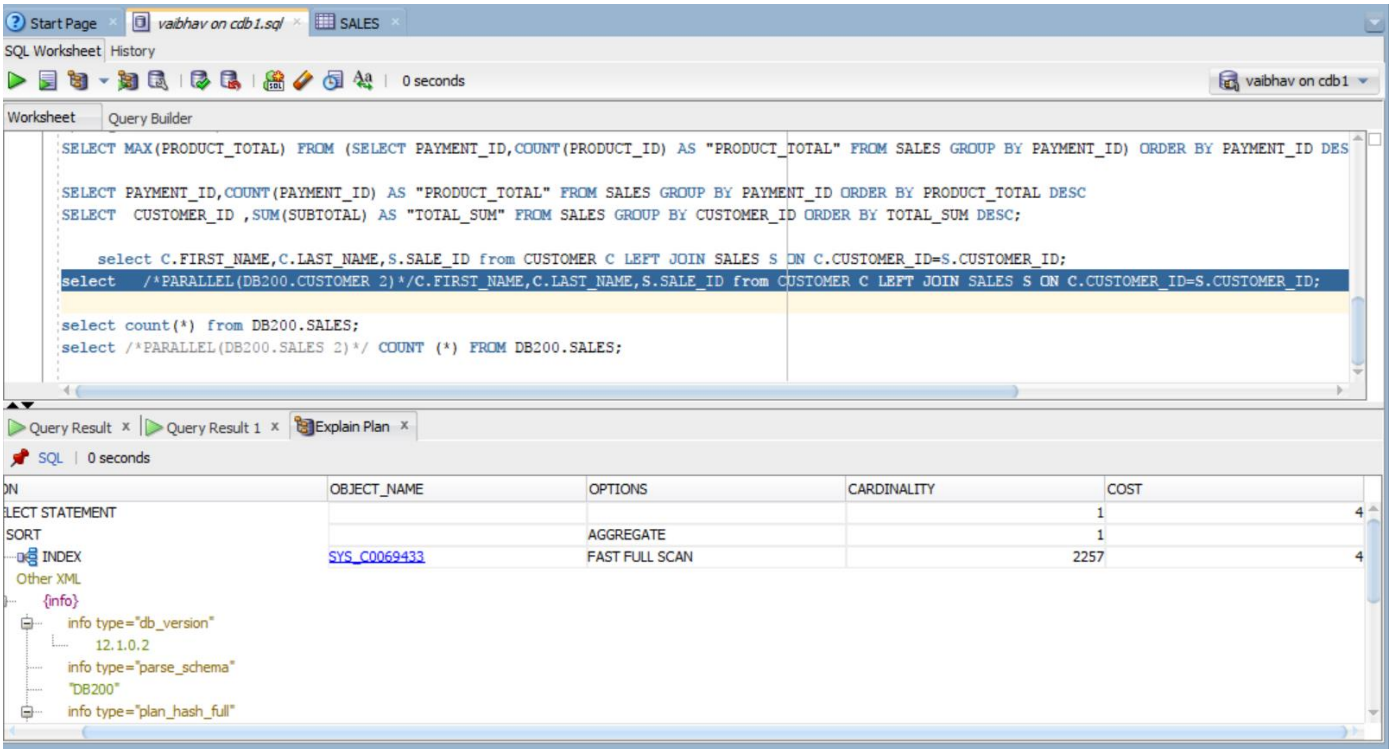


Now let us implement parallel processing.

select /*PARALLEL(DB200.CUSTOMER 2)*/C.FIRST_NAME,C.LAST_NAME,S.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=S.CUSTOMER_ID;



Its Explain Plan and cost is as follows



So, we can see the execution time and cost has been reduced greatly.

OTHER TOPICS

We have explored two topics in our project, first is database security using Encryption and second is DBA scripts.

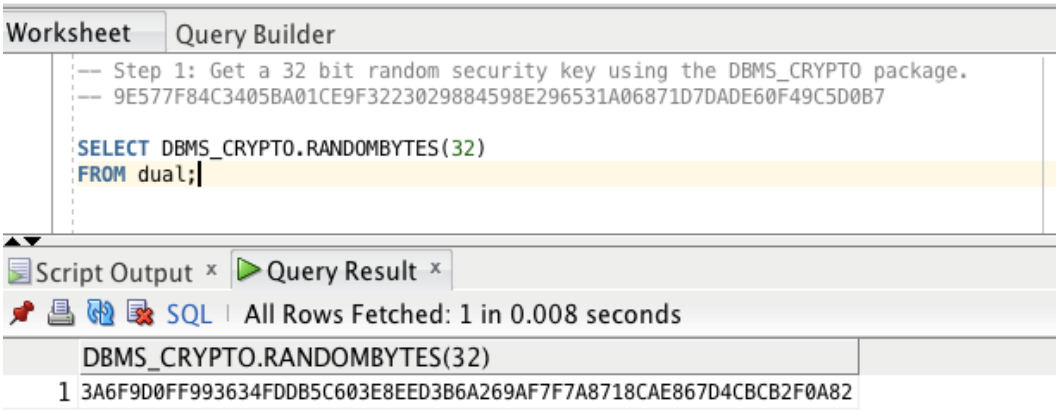
ENCRYPTION

There is always some sensitive data stored in the databases which should be protected at all costs. Encryption is one of the features used in database to protect such data. Encryption is the process of converting data into a code which is incomprehensible to humans until some technology is used to decrypt it into simple language.

Suppose, the manager needs to fetch the information of top five employees in the Employee table, a SQL query can be run to retrieve this data. Note that there is some sensitive data like SSN which is also stored in the table and the admin wants to keep this protected. So, SSN can be encrypted.

There are five steps to encrypt a column in a table.

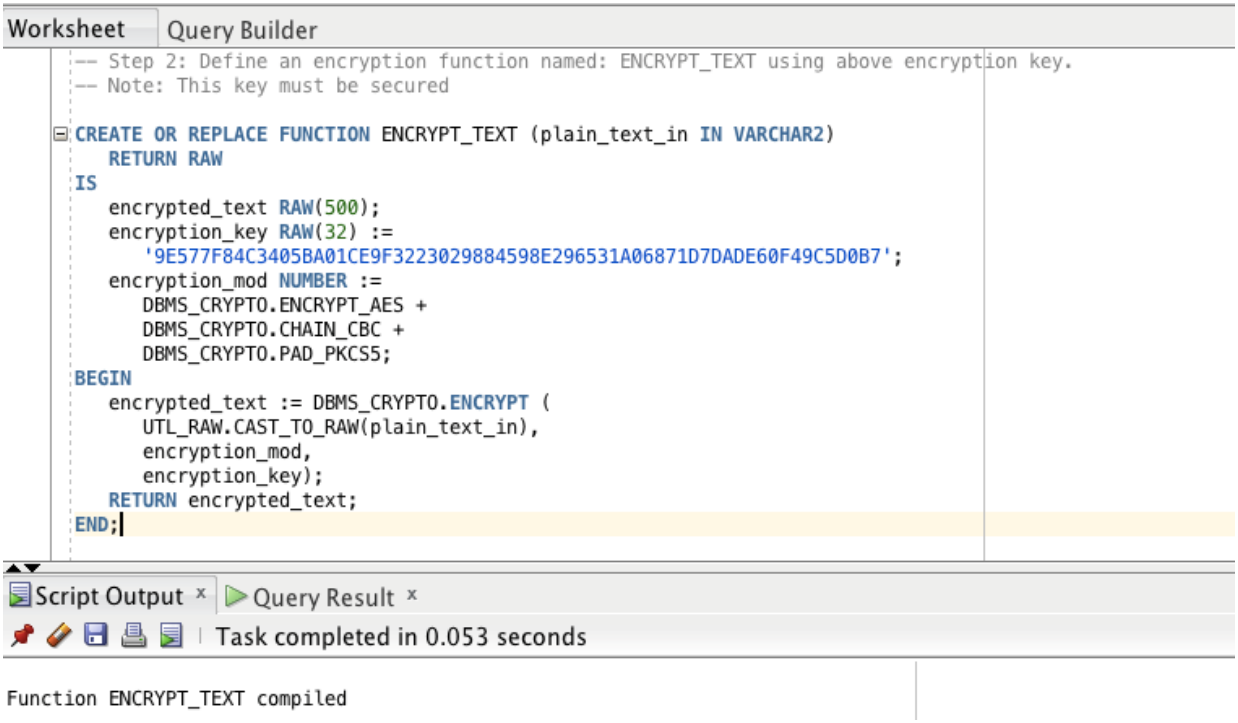
1. First step is to generate a 32-bit random security key using DBMS_CRYPTO package.



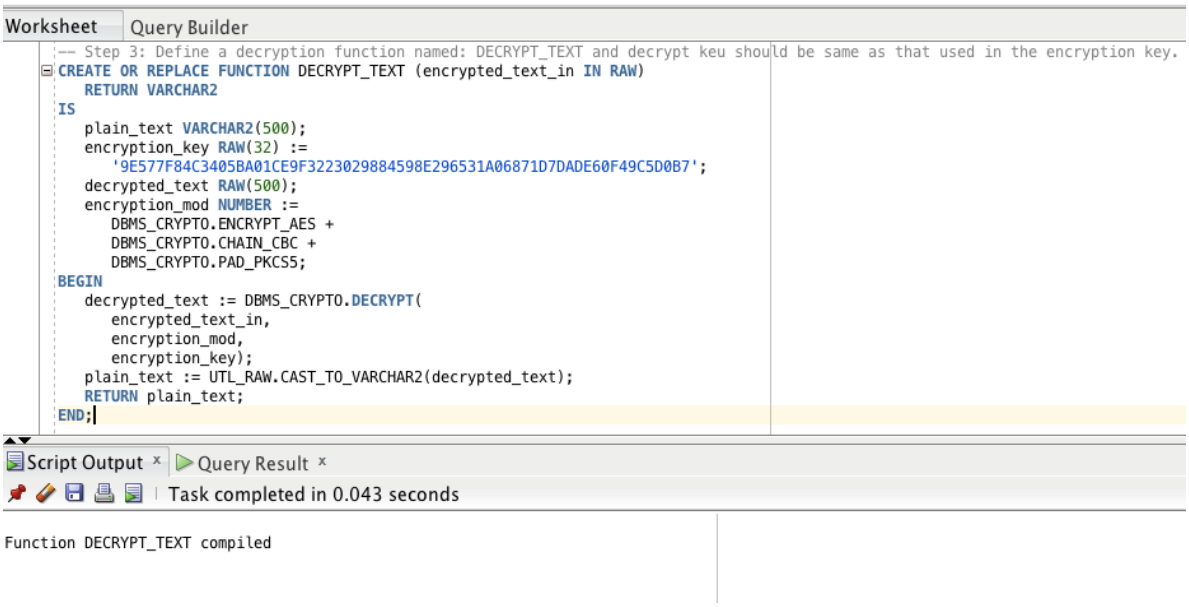
SELECT DBMS_CRYPTO.RANDOMBYTES(32)

FROM dual;

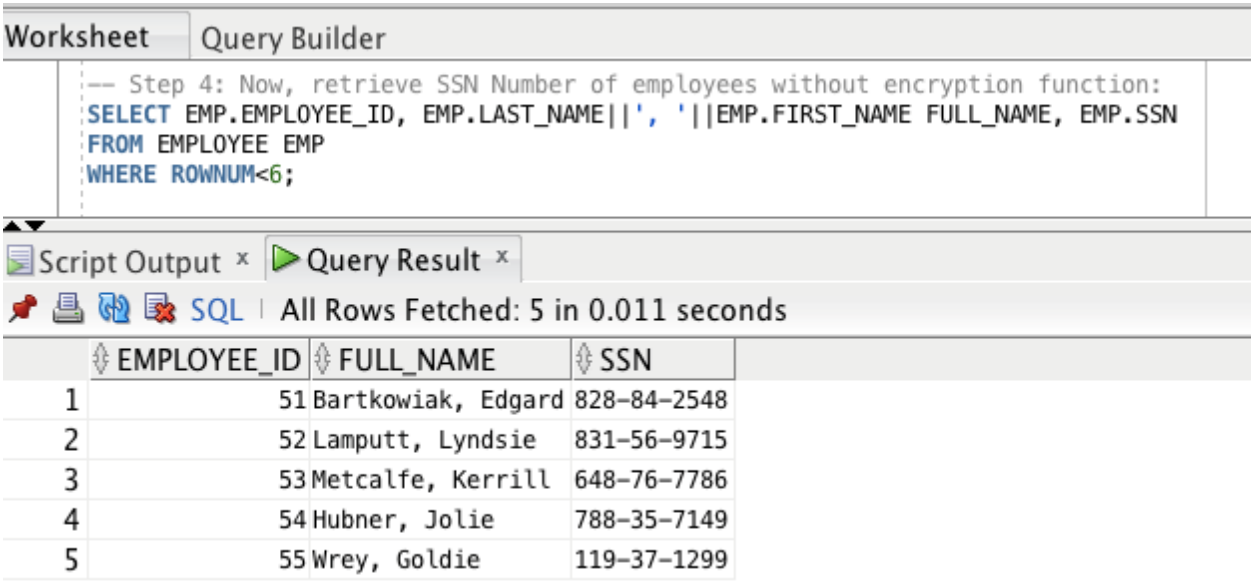
2. Second step is to define an encryption function using above generated encryption key.



3. Third step is to define a decryption function. decryption key should be same as that used in encryption.



4. Now, retrieve SSN number of employees without encryption function.



Worksheet | Query Builder

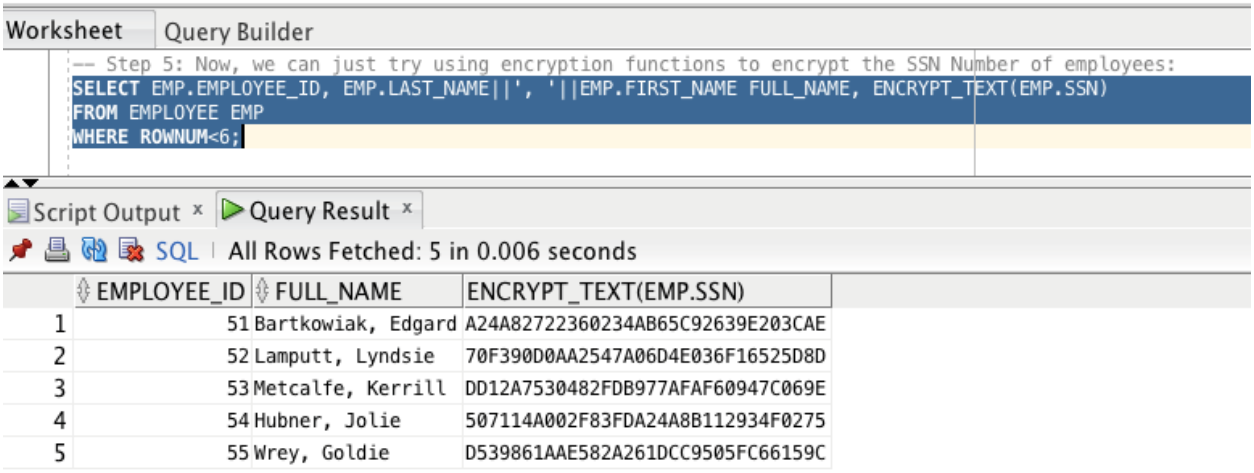
```
-- Step 4: Now, retrieve SSN Number of employees without encryption function:
SELECT EMP.EMPLOYEE_ID, EMP.LAST_NAME||', '||EMP.FIRST_NAME FULL_NAME, EMP.SSN
FROM EMPLOYEE EMP
WHERE ROWNUM<6;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.011 seconds

	EMPLOYEE_ID	FULL_NAME	SSN
1	51	Bartkowiak, Edgard	828-84-2548
2	52	Lamputt, Lyndsie	831-56-9715
3	53	Metcalfe, Kerrill	648-76-7786
4	54	Hubner, Jolie	788-35-7149
5	55	Wrey, Goldie	119-37-1299

5. Next, we can retrieve the same data using encryption function.



Worksheet | Query Builder

```
-- Step 5: Now, we can just try using encryption functions to encrypt the SSN Number of employees:
SELECT EMP.EMPLOYEE_ID, EMP.LAST_NAME||', '||EMP.FIRST_NAME FULL_NAME, ENCRYPT_TEXT(EMP.SSN)
FROM EMPLOYEE EMP
WHERE ROWNUM<6;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.006 seconds

	EMPLOYEE_ID	FULL_NAME	ENCRYPT_TEXT(EMP.SSN)
1	51	Bartkowiak, Edgard	A24A82722360234AB65C92639E203CAE
2	52	Lamputt, Lyndsie	70F390D0AA2547A06D4E036F16525D8D
3	53	Metcalfe, Kerrill	DD12A7530482FDB977AFAF60947C069E
4	54	Hubner, Jolie	507114A002F83FDA24A8B112934F0275
5	55	Wrey, Goldie	D539861AAE582A261DCC9505FC66159C

So, now when the query runs, the data retrieved shows SSN in an encrypted form.

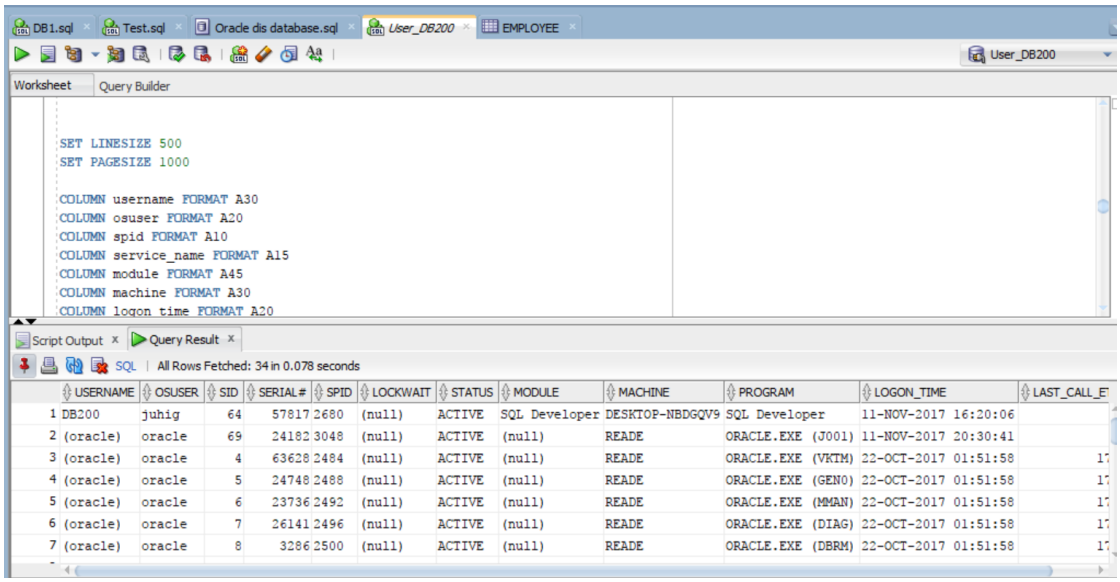
DBA SCRIPTS

DBA or Database scripts are used by admins to troubleshoot the database. There are different types of scripts available for monitoring, resource manager, constraints, security etc.

Here, we have explored few of these scripts in real time.

- **To find all the active database sessions**.

```
SET LINESIZE 500
SET PAGESIZE 1000
COLUMN username FORMAT A30
COLUMN osuser FORMAT A20
COLUMN spid FORMAT A10
COLUMN service_name FORMAT A15
COLUMN module FORMAT A45
COLUMN machine FORMAT A30
COLUMN logon_time FORMAT A20
SELECT NVL(s.username, '(oracle)') AS username,
s.osuser, s.sid, s.serial#, p.spid, s.lockwait, s.status, s.module, s.machine, s.program,
TO_CHAR(s.logon_Time,'DD-MON-YYYY HH24:MI:SS') AS logon_time,
s.last_call_et AS last_call_et_secs
FROM   v$session s, v$process p
WHERE  s.paddr  = p.addr
AND    s.status = 'ACTIVE'
ORDER BY s.username, s.osuser;
SET PAGESIZE 14
```

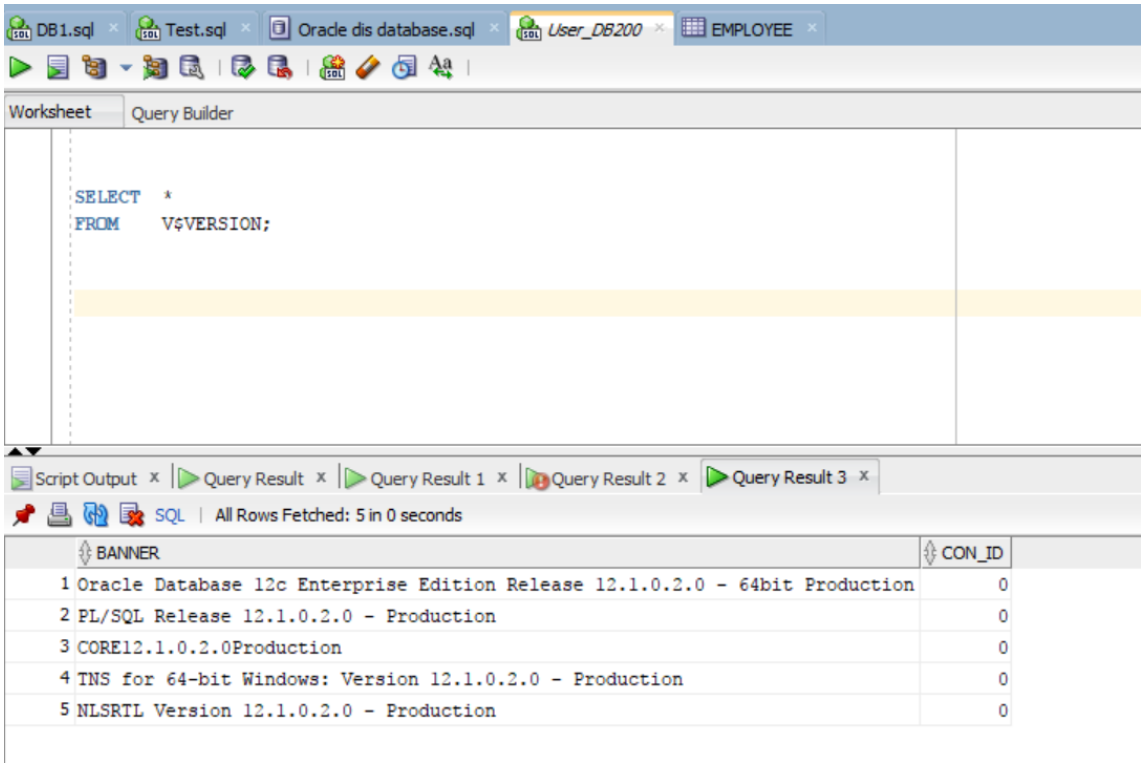


There are total 34 active sessions.

- **To find the version of the database.**

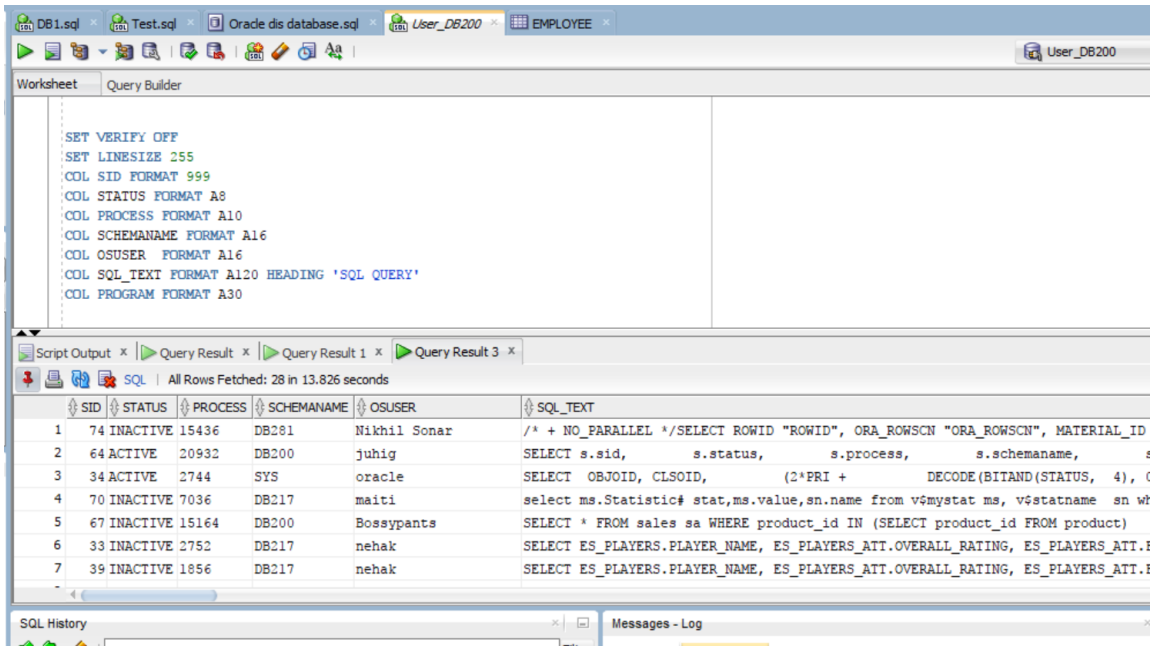
SELECT *

FROM V$VERSION;



- **To display the SQL statements for the current database session.**

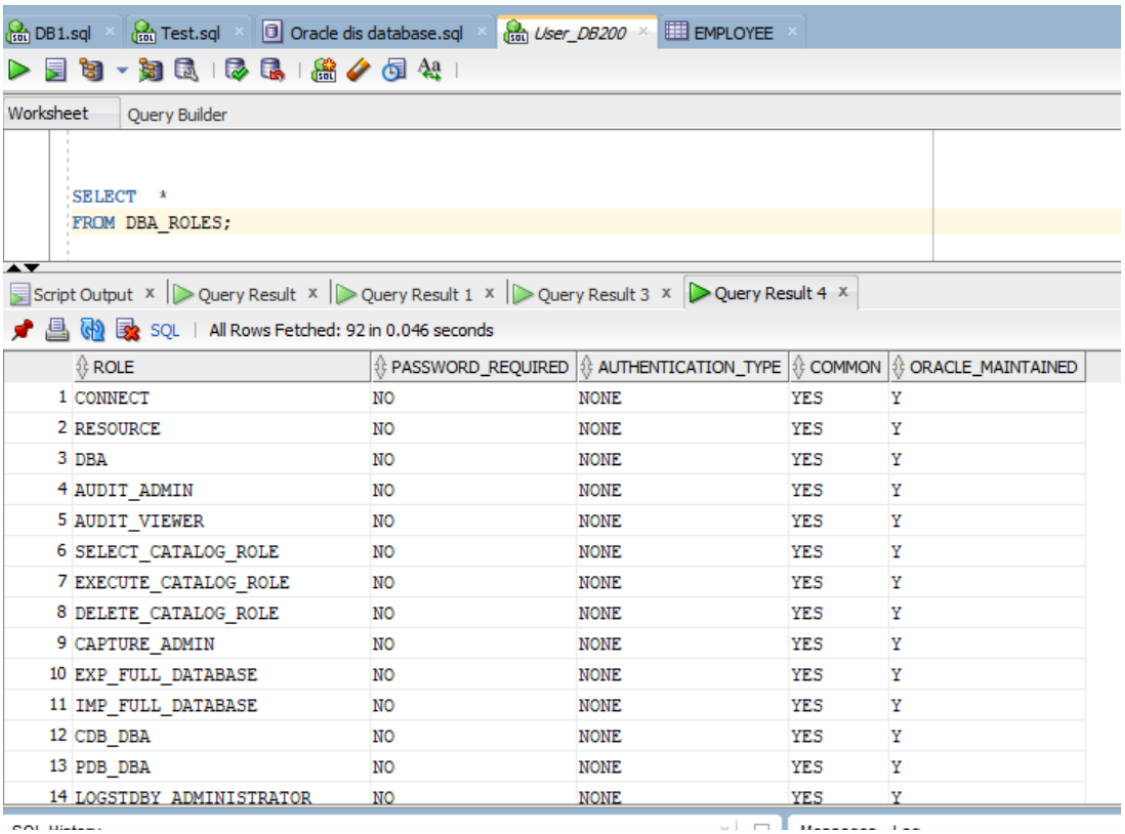
```
SET VERIFY OFF

SET LINESIZE 255

COL SID FORMAT 999

COL STATUS FORMAT A8

COL PROCESS FORMAT A10

COL SCHEMANAME FORMAT A16

COL OSUSER  FORMAT A16

COL SQL_TEXT FORMAT A120 HEADING 'SQL QUERY'

COL PROGRAM FORMAT A30

SELECT s.sid, s.status, s.process, s.schemaname, s.osuser, a.sql_text, p.program

FROM   v$session s, v$sqlarea a, v$process p

WHERE  s.SQL_HASH_VALUE = a.HASH_VALUE

AND    s.SQL_ADDRESS = a.ADDRESS

AND    s.PADDR = p.ADDR /

SET VERIFY ON

SET LINESIZE 255
```



The output shows result for both Active and Inactive sessions.

- **To find the list of roles.**

SELECT * FROM DBA_ROLES;



EVALUATION TABLE

Topic	Description	Evaluation
Entity- Relationship Diagram	**This section includes ERD and the description of relationships between the entities.**	**20**
Physical Database Design & Data Creation and Import	**This section includes the SQL queries used to create tables in the database. It also includes the procedure to generate sample data and how it is imported into the database.**	**20**
SQL Queries	**This section includes some interesting SQL Queries created for the database.**	**20**
Performance Tuning	**In this section, we have used three methods to improve the performance of our database management system.**	**25**
Other Topics	**In this section, we have explored two topics that we found interesting. One is related to database security and the other is DBA Scripts.**	**15**