

# STOREMART

## ADVANCED DATABASE MANAGEMENT SYSTEM GROUP PROJECT

**PREPARED BY:**  
**JUHI GUPTA**  
**NIDHI PARASHAR**  
**PANKAJ KUMAR**  
**VAIBHAV MIGLANI**

## **TABLE OF CONTENTS**

1.	Introduction.....	3
2.	Basic information of the system.....	3
3.	Entity-Relation Diagram (ERD).....	5
	i) Brief description of the relationships between entities.....	6
4.	Physical Database design.....	14
	i) Data Creation and Import.....	20
5.	SQL Queries.....	22
6.	Performance Tuning.....	33
	i) Indexing.....	33
	ii) SQL Tuning.....	36
	iii) Parallel Processing.....	38
7.	Other Topics.....	41
	i) Encryption.....	41
	ii) DBA Scripts.....	44
8.	Evaluation Table.....	48

## **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.

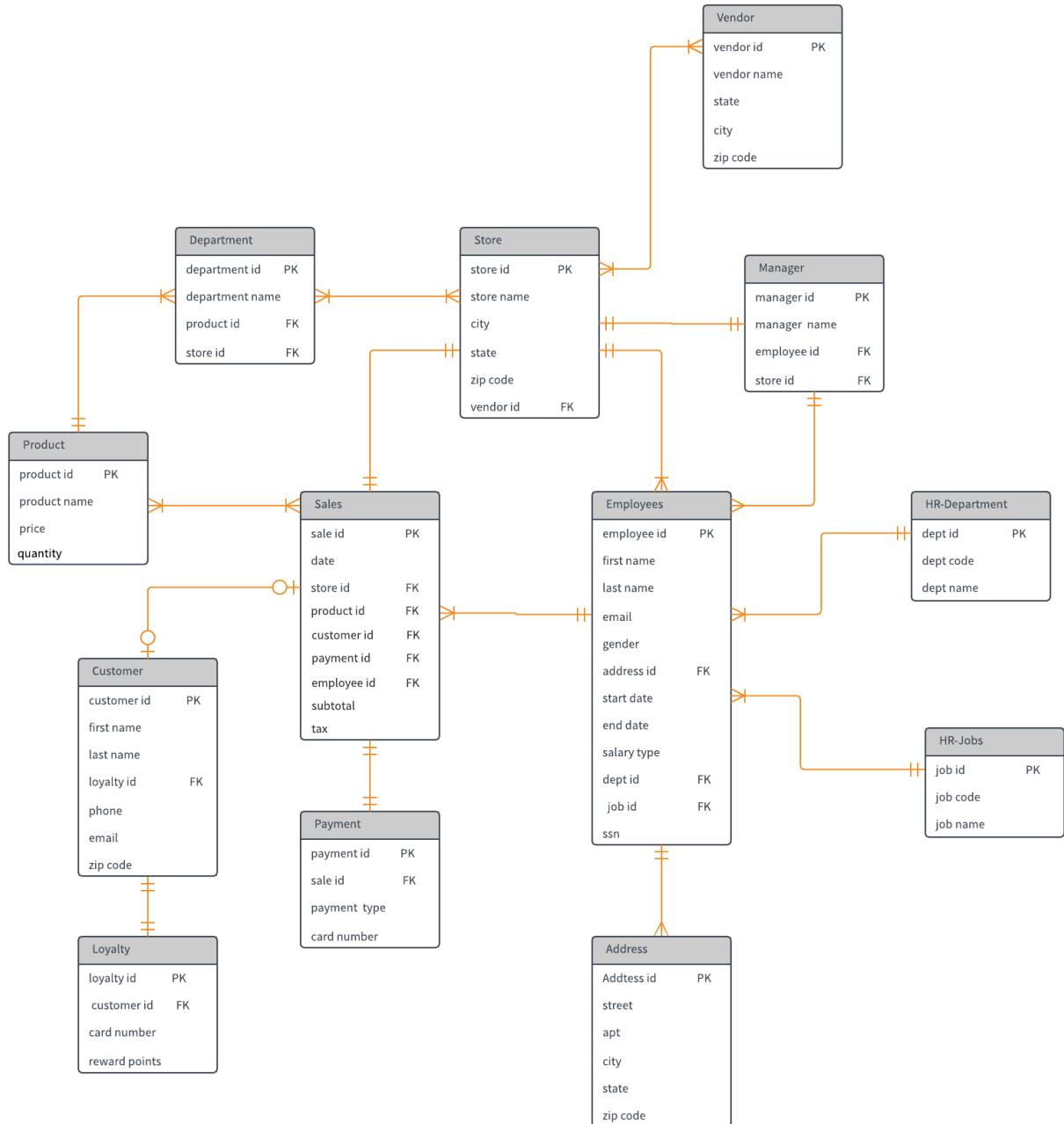
Stores table will maintain the information of all the store outlets of Storemart located in different cities. This will help all the stores to keep in sync with each other.

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**

<b>Attribute_Name</b>	<b>Type</b>	<b>Null Allowed</b>	<b>Unique</b>
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**

<b>Attribute_Name</b>	<b>Type</b>	<b>Null Allowed</b>	<b>Unique</b>
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**

<b>Attribute_Name</b>	<b>Type</b>	<b>Null Allowed</b>	<b>Unique</b>
Sales_Id	Number	No	Yes
Date	Date	No	Yes
Store_Id	Number	No	No
Customer_Id	Number	Yes	No
Product_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**

<b>Attribute_Name</b>	<b>Type</b>	<b>Null Allowed</b>	<b>Unique</b>
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**

Attribute_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**

Attribute_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));
```

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:

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:

Actions...									
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);
```

Actions...					
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:

Actions...									
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)	(null)	ENABLED	NOT DEFERRABLE	
3 SYS_C0069179	Check	"CUSTOMER_ID" I... (null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	
4 SYS_C0069180	Check	"LOYALTY_CARD_N... (null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	
5 SYS_C0069181	Check	"VALIDITY_DATE"... (null)	(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  
);
```

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:

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);
```

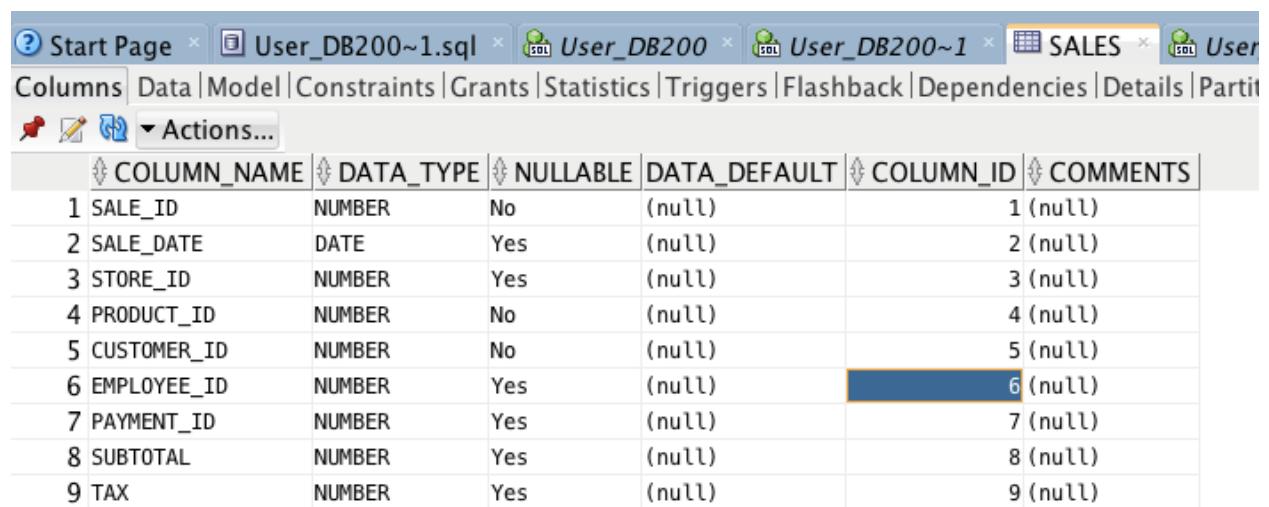
Actions...					
COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN...	COMMENTS
1 VENDOR_ID	NUMBER	No	(null)	1	(null)
2 STORE_ID	NUMBER	Yes	(null)	2	(null)
3 VENDOR_COMPANY_NAME	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4 VENDOR_ADDRESS	VARCHAR2(255 BYTE)	No	(null)	4	(null)
5 VENDOR_CITY	VARCHAR2(255 BYTE)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

### Constraints:

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)	(null)	ENABLED	NOT DEFERRABLE	
2 SYS_C0069186	Check	"VENDOR_ADDRESS..." (null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	
3 SYS_C0069187	Check	"VENDOR_ZIP_CODE..." (null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	
4 SYS_C0069188	Check	"VENDOR_PHONE" ... (null)	(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	

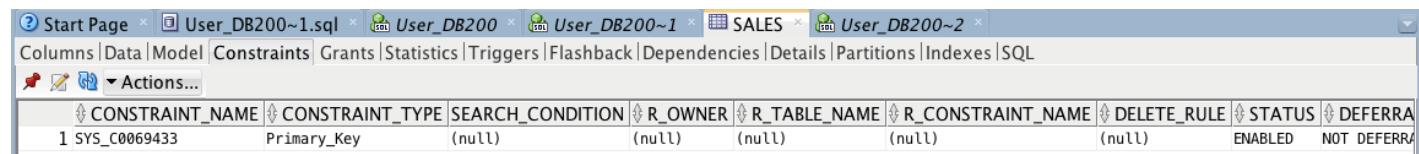
## Sales

```
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);
```



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:

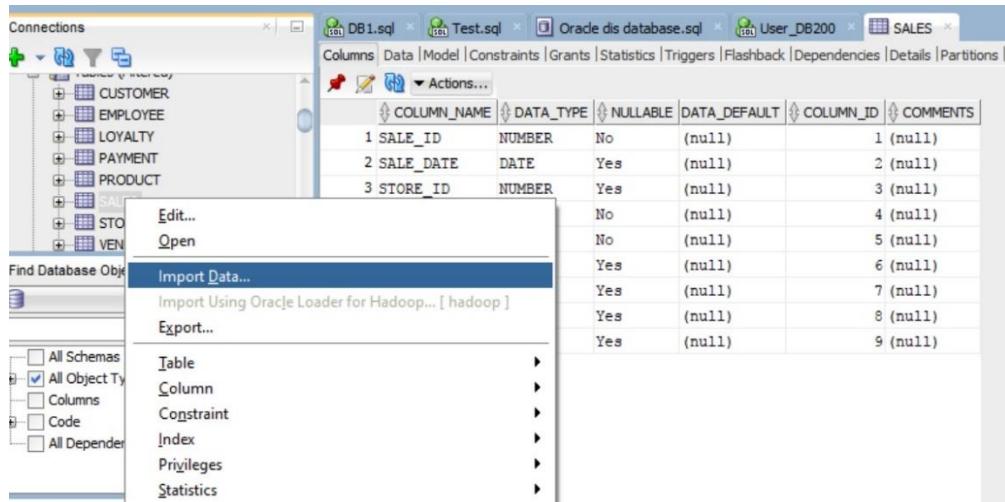


CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRA
1 SYS_C0069433	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA

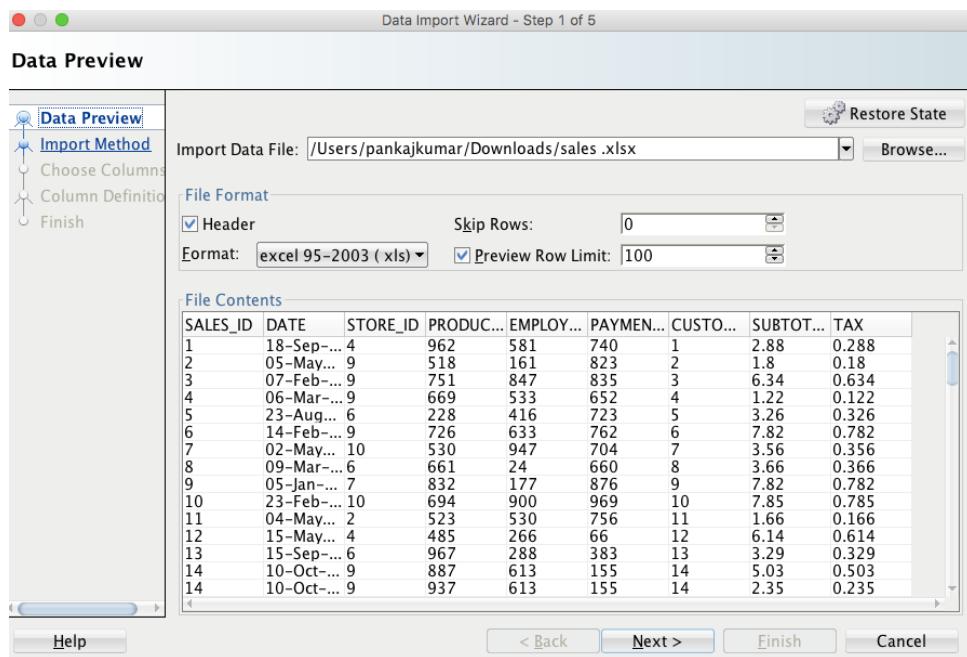
## Data Creation and Import

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.

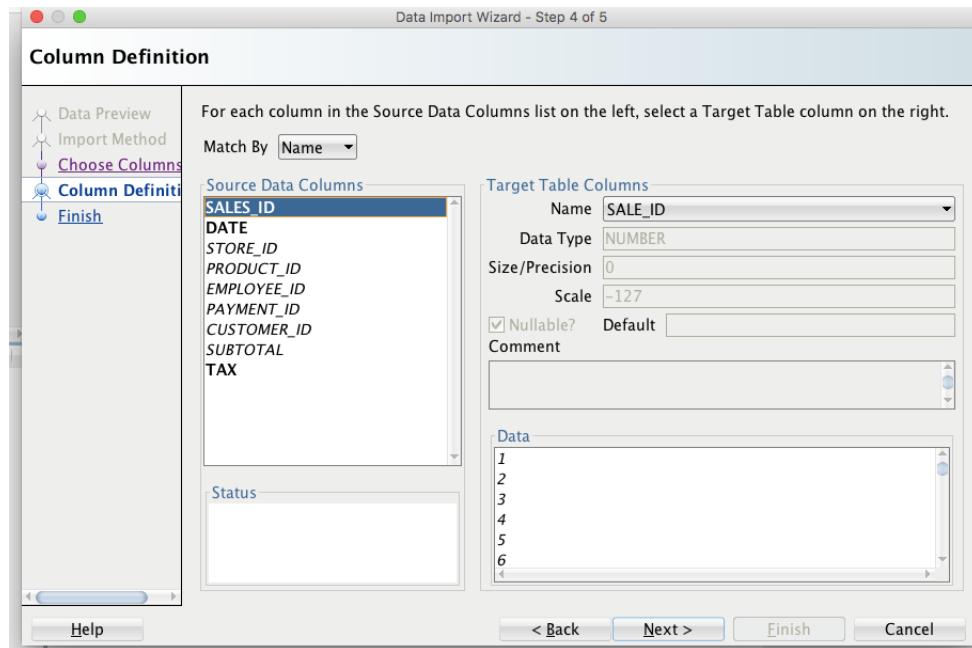
1. Right-click on the table in which data is to be imported and choose Import data.



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.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 SALE_ID	NUMBER	Yes	(null)	1	(null)
2 SALE_DATE	DATE	Yes	(null)	2	(null)
3 STORE_ID	NUMBER	Yes	(null)	3	(null)
4 PRODUCT_ID	NUMBER	Yes	(null)	4	(null)
5 CUSTOMER_ID	NUMBER	Yes	(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)

Import Data

Import Data into table SALES from file sales.xlsx . Task successful and import committed.

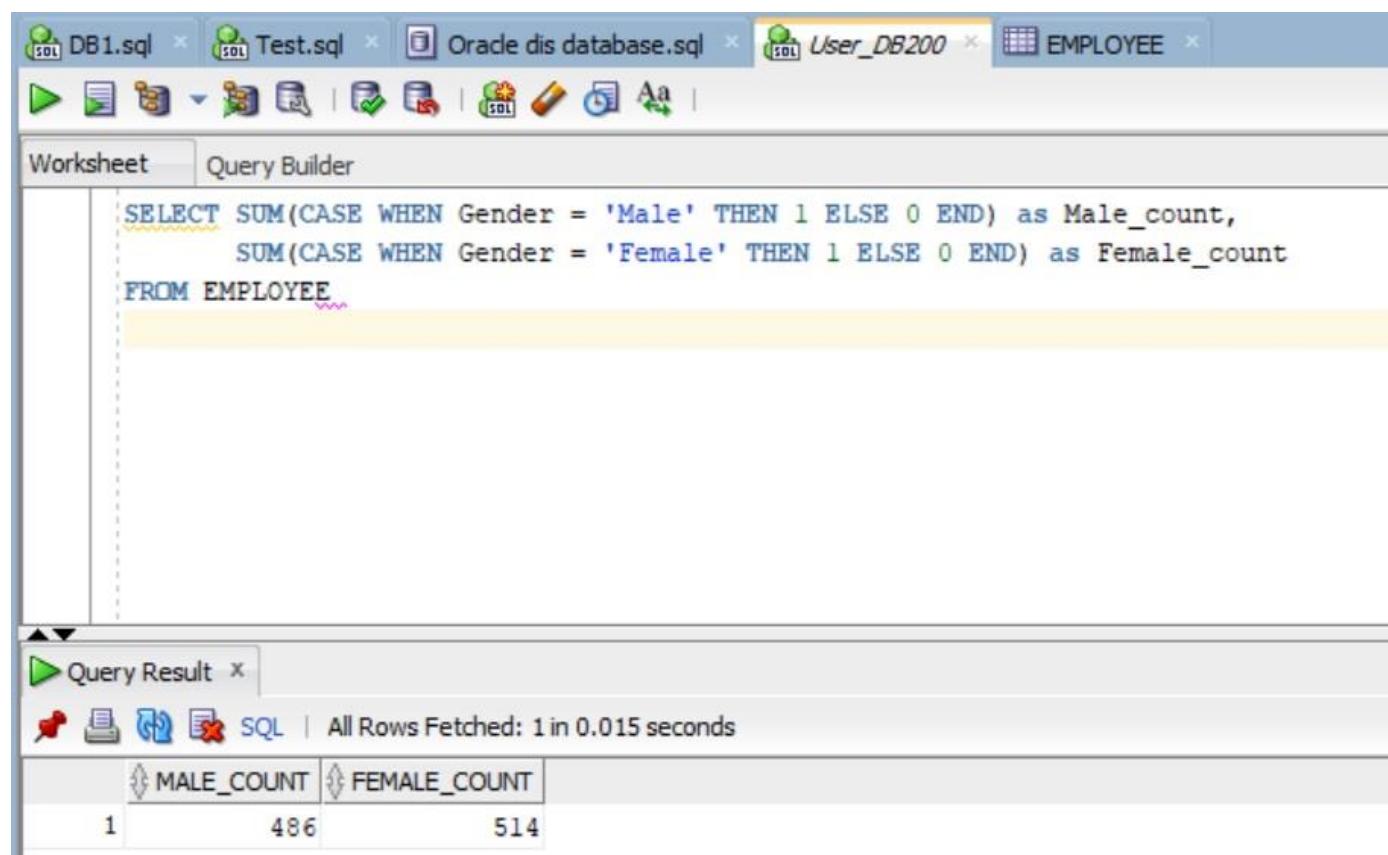
OK

## 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
```



The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs for DB1.sql, Test.sql, Oracle dis database.sql, User\_DB200, and EMPLOYEE. Below the tabs is a toolbar with various icons. The main area is divided into two tabs: Worksheet and Query Builder. The Worksheet tab is active, displaying the SQL query:

```
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
```

Below the worksheet is a Query Result window. It has a toolbar with icons for refresh, print, and cancel, followed by a SQL button and the text "All Rows Fetched: 1 in 0.015 seconds". The result table has two columns: MALE\_COUNT and FEMALE\_COUNT. The data row shows 1 in the MALE\_COUNT column and 486 in the FEMALE\_COUNT column.

MALE_COUNT	FEMALE_COUNT
1	486

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';
```

The screenshot shows the Oracle SQL Developer interface. At the top, there are several tabs: DB1.sql, Test.sql, Oracle dis database.sql, User\_DB200 (which is the active tab), and EMPLOYEE. Below the tabs is a toolbar with various icons. The main workspace is divided into two panes: 'Worksheet' (active) and 'Query Builder'. The Worksheet pane contains the SQL query:

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

The Query Result pane at the bottom shows the output of the query:

COUNT(DISTINCT CUSTOMER_ID)
1
853

Below the table, it says "All Rows Fetched: 1 in 0 seconds".

**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;
```

The screenshot shows the Oracle SQL Developer interface. The top navigation bar has tabs for DB1.sql, Test.sql, Oracle db database.sql, User\_DB200, and EMPLOYEE. Below the tabs is a toolbar with various icons. The main area has two tabs: Worksheet and Query Builder. The Worksheet tab contains the SQL query provided above. The Query Result tab shows the output of the query:

PRODUCT_ID	PRODUCT_NAME
1	1000 Pastry - Cheese Baked Scones

Below the table, it says "All Rows Fetched: 1 in 0.032 seconds".

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

SELECT MAX(PRODUCT\_TOTAL)

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

The screenshot shows the Oracle SQL Developer interface. The top navigation bar has tabs for DB1.sql, Test.sql, Oracle dis database.sql, User\_DB200, and EMPLOYEE. Below the tabs is a toolbar with various icons. The main area is divided into two panes: 'Worksheet' on the left and 'Query Builder' on the right. The 'Worksheet' pane contains the SQL query:

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

The 'Query Result' tab is selected in the bottom panel, showing the output:

	MAX(PRODUCT_TOTAL)
1	45

The status bar at the bottom of the result pane indicates "All Rows Fetched: 1 in 0 seconds".

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;
```

Worksheet    Query Builder

```
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;
```

Query Result    x

SQL | All Rows Fetched: 3 in 0.015 seconds

	CUSTOMER_ID	FULL_NAME
1	773	Goodship, Shaun
2	828	Baiden, Mildrid
3	661	Baszkiewicz, Gwynne

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;
```

The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs for DB1.sql, Test.sql, Oracle db database.sql, User\_DB200, and EMPLOYEE. Below the tabs is a toolbar with various icons. The main area is divided into two panes: 'Worksheet' (selected) and 'Query Builder'. The 'Worksheet' pane contains the SQL query from above. The 'Query Result' pane at the bottom shows the output of the query:

EMPLOYEE_ID	FULL_NAME
1	831 Alner, Celinka

Below the table, it says 'All Rows Fetched: 1 in 0.062 seconds'.

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;

```

The screenshot shows the Oracle SQL Developer interface. The top menu bar includes icons for running, saving, and zooming. Below the menu is a toolbar with various buttons. The main window has two tabs: 'Worksheet' and 'Query Builder'. The 'Worksheet' tab contains the SQL query shown above. The 'Query Result' tab shows the output of the query:

	CUSTOMER_ID	FULL_NAME
1	773	Goodship, Shaun

Below the table, it says 'All Rows Fetched: 1 in 0.015 seconds'.

#### 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;

```

The screenshot shows the Oracle SQL Developer interface. The top part is the 'Worksheet' tab, which contains the SQL query. The bottom part is the 'Query Result' tab, which displays the output of the query.

```

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;

```

STORE_ID	STORE_NAME
1	3 Storemart-C

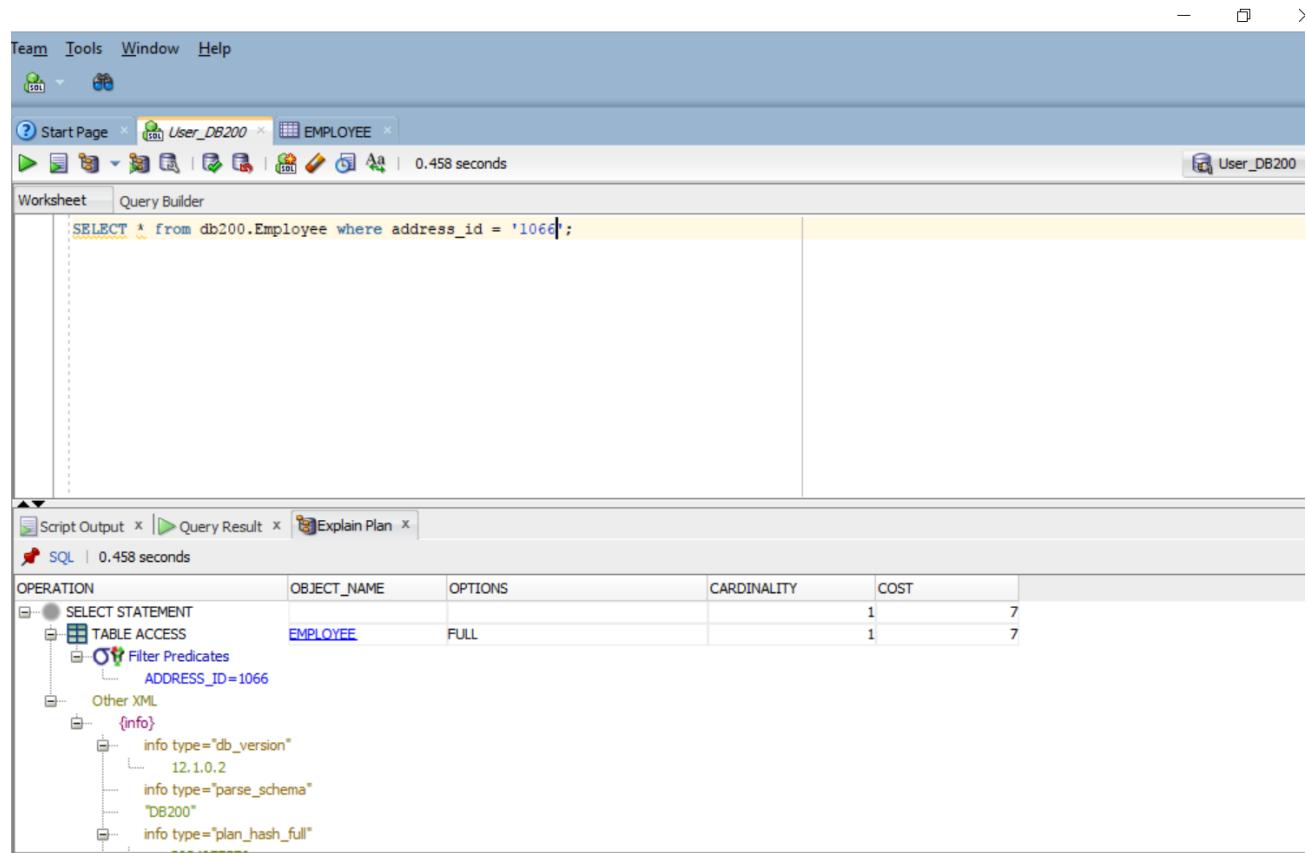
# PERFORMANCE TUNING

## 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:



The screenshot shows the Oracle SQL Developer interface. The top window displays a query in the Worksheet tab:

```
SELECT * from db200.Employee where address_id = '1066';
```

The bottom window shows the Explain Plan for this query, with the following details:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	7
TABLE ACCESS	EMPLOYEE	FULL	1	7
Filter Predicates				
ADDRESS_ID=1066				
Other XML				
{info}				
info type="db_version"				
12.1.0.2				
info type="parse_schema"				
"DB200"				
info type="plan_hash_full"				

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.

The screenshot shows the Oracle SQL Developer interface with the following details:

- Worksheet Tab:** Contains the SQL code:
 

```
create index idx_address on EMPLOYEE(Address_Id);
SELECT * from db200.Employee where address_id = '1066';
```
- Script Output Tab:** Shows the execution time: 0.43200001 seconds.
- Explain Plan Tab:** Displays the execution plan:
 

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CRTD
SELECT STATEMENT				2	
TABLE ACCESS	EMPLOYEE	BY INDEX ROWID BATCHED	1	2	
INDEX	IDX_ADDRESS	RANGE SCAN	1	1	
Access Predicates					
- V\$STATNAME Statistics:** A table showing various statistics:
 

V\$STATNAME Name	V\$MYSTAT Value
consistent gets examination	26
consistent gets examination (fastpath)	26
consistent gets from cache	90
consistent gets pin	64
consistent gets pin (fastpath)	64
CPU used by this session	2
CPU used when call started	5
DB time	6
enqueue releases	4

## 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:

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are tabs for 'Start Page', 'User\_DB200', and 'EMPLOYEE'. The status bar at the bottom indicates a duration of '0.2689999 seconds'. The main area is a 'Worksheet' tab with the following content:

```
SELECT * from EMPLOYEE Where Upper(LAST_NAME) LIKE 'P%';
```

Below the worksheet, the 'Autotrace' tab shows the execution plan:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST
SELECT STATEMENT				7	16	
TABLE ACCESS	EMPLOYEE	FULL	50	7	16	
Filter Predicates				UPPER(LAST_NAME) LIKE 'P%'		

At the bottom of the interface, there is a 'V\$STATNAME' table:

V\$STATNAME Name	V\$MYSTAT Value
cell physical IO interconnect bytes	114688
consistent gets	35
consistent gets examination	1
consistent gets examination (fastpath)	1
consistent gets from cache	35
consistent gets pin	34
consistent gets pin (fastpath)	34
CPU used by this session	5
CPU used when call started	5

After creating the index using the following statement:

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

The screenshot shows the Oracle SQL Developer interface. The top navigation bar has tabs for 'Start Page', 'User\_DB200', and 'EMPLOYEE'. The status bar at the bottom indicates a duration of '0.053 seconds'. The main area is a 'Worksheet' tab with the following content:

```
SELECT * from EMPLOYEE Where Upper(LAST_NAME) LIKE 'P%';
```

Below the worksheet, the 'Autotrace' tab shows the execution plan:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50	6
TABLE ACCESS	EMPLOYEE	BY INDEX ROWID BATCHED	50	6
INDEX	IDX_UPPER_LNAME	RANGE SCAN	9	2
Access Predicates				
UPPER(LAST_NAME) LIKE 'P%'				
Filter Predicates				
UPPER(LAST_NAME) LIKE 'P%'				

At the bottom of the interface, there is an 'Other XML' section with an 'info' entry:

```
inf_time="db_version"
```

A 'COST=2' label is also visible at the bottom right.

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:

The image shows two separate sessions in Oracle SQL Developer. Both sessions have a 'Worksheet' tab active and show the same SQL query:

```
SELECT * from PRODUCT Where QUANTITY >=500;
```

The top session shows an execution time of 0.052 seconds. Its explain plan is as follows:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			503	5
TABLE ACCESS	PRODUCT	FULL	503	5
Filter Predicates				

The bottom session shows an execution time of 0.027 seconds. Its explain plan is as follows:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			503	5
TABLE ACCESS	PRODUCT	FULL	503	5
Filter Predicates				

## 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.

The screenshot shows the Oracle SQL Developer interface with two queries in the Worksheet tab:

```

Worksheet Query Builder
SELECT * from sales sa
WHERE EXISTS(SELECT * FROM PRODUCT pr where pr.product_id = sa.product_id);

SELECT * FROM sales sa
WHERE product_id IN (SELECT product_id FROM product);

```

The second query is highlighted with a yellow background. Below the queries is the Query Result tab, which displays the fetched data:

Script Output x Query Result x

SQL | Fetched 50 rows in 0.005 seconds

	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

The screenshot shows the Oracle SQL Developer interface with the same two queries in the Worksheet tab:

```

Worksheet Query Builder
SELECT * from sales sa
WHERE EXISTS(SELECT * FROM PRODUCT pr where pr.product_id = sa.product_id);

SELECT * FROM sales sa
WHERE product_id IN (SELECT product_id FROM product);

```

The second query is highlighted with a yellow background. Below the queries is the Query Result tab, which displays the fetched data:

Script Output x Query Result x

SQL | Fetched 50 rows in 0.008 seconds

	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.

The screenshot shows a SQL query builder interface with the following details:

- Worksheet Tab:** The tab is selected, showing the following SQL code:

```
Select Sale_Id, Store_Id
From Sales
WHERE Exists(Select * from Product WHERE Product.Product_Id = Sales.Product_Id);
```
- Query Result Tab:** This tab is active, displaying the results of the executed query. The results are presented in a table with two columns: SALE\_ID and STORE\_ID. The data is as follows:

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

- Execution Details:** Below the table, it says "Fetched 50 rows in 0.055 seconds".

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

The screenshot shows a SQL query execution environment. At the top, there's a toolbar with various icons. Below it, a status bar displays "0.317 seconds". The main area has tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The query window contains the following SQL code:

```
Select Sale_Id, Store_Id  
From Sales INNER JOIN PRODUCT ON Product.Product_Id = Sales.Product_Id;
```

Below the query window is a "Script Output" tab and a "Query Result" tab. The "Query Result" tab is active, showing the results of the query. A message at the top of this tab says "Fetched 50 rows in 0.06 seconds". The results are presented in a table with columns labeled "SALE\_ID" and "STORE\_ID". The data is as follows:

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

- 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

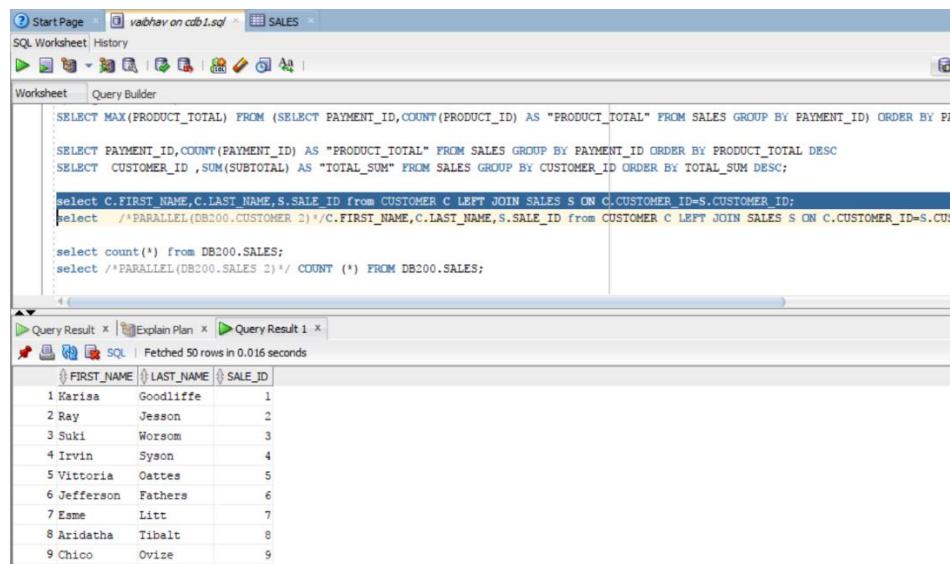
- Large table Scans, Joins
- Creation of Large Indexes
- Creation of Large Tables
- Bulk Inserts, Updates, Merge, and Delete.

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;
```



The screenshot shows the Oracle SQL Developer interface. The SQL Worksheet tab is active, displaying the following SQL code:

```
SELECT MAX(PRODUCT_TOTAL) FROM (SELECT PAYMENT_ID,COUNT(PRODUCT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID) ORDER BY PAYMENT_ID DESC;  
SELECT PAYMENT_ID,COUNT(PAYMENT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID ORDER BY PRODUCT_TOTAL DESC;  
SELECT CUSTOMER_ID ,SUM(SUBTOTAL) AS "TOTAL_SUM" FROM SALES GROUP BY CUSTOMER_ID ORDER BY TOTAL_SUM DESC;  
select C.FIRST_NAME,C.LAST_NAME,S.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=S.CUSTOMER_ID;  
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;  
select count(*) from DB200.SALES;  
select /*+PARALLEL(DB200.SALES 2)*/ COUNT (*) FROM DB200.SALES;
```

Below the code, the Explain Plan tab is visible, showing the execution plan for the query. The Query Result tab shows the final output:

FIRST_NAME	LAST_NAME	SALE_ID
1 Karisa	Goodliffe	1
2 Ray	Jesson	2
3 Suki	Worsom	3
4 Irvin	Syson	4
5 Vittoria	Oattees	5
6 Jefferson	Fathers	6
7 Esme	Litt	7
8 Aridatha	Tibalt	8
9 Chico	Ovize	9

Its Explain Plan and cost are as follows-

The screenshot shows the Oracle SQL Developer interface. In the top window, a query named 'SALES' is displayed:

```
SELECT MAX(PRODUCT_TOTAL) FROM (SELECT PAYMENT_ID,COUNT(PRODUCT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID) ORDER BY PAYMENT_ID DESC;
SELECT PAYMENT_ID,COUNT(PAYMENT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID ORDER BY PRODUCT_TOTAL DESC;
SELECT CUSTOMER_ID ,SUM(SUBTOTAL) AS "TOTAL_SUM" FROM SALES GROUP BY CUSTOMER_ID ORDER BY TOTAL_SUM DESC;
select C.FIRST_NAME,C.LAST_NAME,S.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=S.CUSTOMER_ID;
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;
select count(*) from DB200.SALES;
select /*PARALLEL(DB200.SALES 2)*/ COUNT (*) FROM DB200.SALES;
```

In the bottom window, the 'Explain Plan' tab is selected, showing the execution plan details:

ON	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				2257
HASH JOIN		OUTER		2257
Access Predicates				9
C.CUSTOMER_ID=S.CUSTOMER_ID(+)				
TABLE ACCESS	CUSTOMER	FULL	1000	5
INDEX	SYS_C0069433	FAST FULL SCAN	2257	4
Other XML	{info}			
{info type="db_version"}	12.1.0.2			

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;
```

The screenshot shows the Oracle SQL Developer interface. In the Worksheet tab, a complex query is written, utilizing parallel execution hints to process multiple customer rows in parallel. The results of the query are displayed in the Query Result tab, showing a list of customer names and their corresponding sale IDs.

```

SELECT MAX(PRODUCT_TOTAL) FROM (SELECT PAYMENT_ID,COUNT(PRODUCT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID) ORDER BY PAYMENT_ID DESC;
SELECT PAYMENT_ID,COUNT(PAYMENT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID ORDER BY PRODUCT_TOTAL DESC;
SELECT CUSTOMER_ID ,SUM(SUBTOTAL) AS "TOTAL_SUM" FROM SALES GROUP BY CUSTOMER_ID ORDER BY TOTAL_SUM DESC;
select c.FIRST_NAME,c.LAST_NAME,s.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=s.CUSTOMER_ID;
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;
select count(*) from DB200.SALES;
select /*+PARALLEL(DB200.SALES 2)*/ COUNT (*) FROM DB200.SALES;

```

FIRST_NAME	LAST_NAME	SALE_ID
1 Karisa	Goodliffe	1
2 Ray	Jesson	2
3 Suki	Worsom	3
4 Irvin	Syson	4
5 Vittoria	Oattes	5
6 Jefferson	Fathers	6
7 Esme	Litt	7
8 Aridatha	Tibalt	8
9 Chico	Ovize	9

Its Explain Plan and cost is as follows

The screenshot shows the Oracle SQL Developer interface with the Explain Plan tab selected. It displays the execution plan for the query, detailing the parallel execution strategy and costs for each step. The cost column indicates the relative cost of each part of the query execution.

```

SELECT MAX(PRODUCT_TOTAL) FROM (SELECT PAYMENT_ID,COUNT(PRODUCT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID) ORDER BY PAYMENT_ID DESC;
SELECT PAYMENT_ID,COUNT(PAYMENT_ID) AS "PRODUCT_TOTAL" FROM SALES GROUP BY PAYMENT_ID ORDER BY PRODUCT_TOTAL DESC;
SELECT CUSTOMER_ID ,SUM(SUBTOTAL) AS "TOTAL_SUM" FROM SALES GROUP BY CUSTOMER_ID ORDER BY TOTAL_SUM DESC;
select c.FIRST_NAME,c.LAST_NAME,s.SALE_ID from CUSTOMER C LEFT JOIN SALES S ON C.CUSTOMER_ID=s.CUSTOMER_ID;
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;
select count(*) from DB200.SALES;
select /*+PARALLEL(DB200.SALES 2)*/ COUNT (*) FROM DB200.SALES;

```

ON	OBJECT_NAME	OPTIONS	CARDINALITY	COST
LECT STATEMENT			1	4
SORT		AGGREGATE	1	4
INDEX	SYS_C0069433	FAST FULL SCAN	2257	4
Other XML				
{info}				
info type="db_version"	12.1.0.2			
info type="parse_schema"	"DB200"			
info type="plan_hash_full"				

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.

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is active, displaying a SQL query:

```
-- Step 1: Get a 32 bit random security key using the DBMS_CRYPTO package.  
-- 9E577F84C3405BA01CE9F3223029884598E296531A06871D7DADE60F49C5D0B7  
SELECT DBMS_CRYPTO.RANDOMBYTES(32)  
FROM dual;
```

The 'Query Result' tab is open below, showing the output:

DBMS_CRYPTO.RANDOMBYTES(32)
1 3A6F9D0FF993634FDD5C603E8EED3B6A269AF7F7A8718CAE867D4CBCB2F0A82

```
SELECT DBMS_CRYPTO.RANDOMBYTES(32)  
FROM dual;
```

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

Worksheet | Query Builder

```
-- Step 2: Define an encryption function named: ENCRYPT_TEXT using above encryption key.  
-- Note: This key must be secured  
  
CREATE OR REPLACE FUNCTION ENCRYPT_TEXT (plain_text_in IN VARCHAR2)  
  RETURN RAW  
IS  
  encrypted_text RAW(500);  
  encryption_key RAW(32) :=  
    '9E577F84C3405BA01CE9F3223029884598E296531A06871D7DADE60F49C5D0B7';  
  encryption_mod NUMBER :=  
    DBMS_CRYPTO.ENCRYPT_AES +  
    DBMS_CRYPTO.CHAIN_CBC +  
    DBMS_CRYPTO.PAD_PKCS5;  
BEGIN  
  encrypted_text := DBMS_CRYPTO.ENCRYPT (  
    UTL_RAW.CAST_TO_RAW(plain_text_in),  
    encryption_mod,  
    encryption_key);  
  RETURN encrypted_text;  
END;
```

Script Output | Query Result | Task completed in 0.053 seconds

Function ENCRYPT\_TEXT compiled

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

Worksheet | Query Builder

```
-- Step 3: Define a decryption function named: DECRYPT_TEXT and decrypt key should be same as that used in the encryption key.  
CREATE OR REPLACE FUNCTION DECRYPT_TEXT (encrypted_text_in IN RAW)  
  RETURN VARCHAR2  
IS  
  plain_text VARCHAR2(500);  
  encryption_key RAW(32) :=  
    '9E577F84C3405BA01CE9F3223029884598E296531A06871D7DADE60F49C5D0B7';  
  decrypted_text RAW(500);  
  encryption_mod NUMBER :=  
    DBMS_CRYPTO.ENCRYPT_AES +  
    DBMS_CRYPTO.CHAIN_CBC +  
    DBMS_CRYPTO.PAD_PKCS5;  
BEGIN  
  decrypted_text := DBMS_CRYPTO.DECRYPT(  
    encrypted_text_in,  
    encryption_mod,  
    encryption_key);  
  plain_text := UTL_RAW.CAST_TO_VARCHAR2(decrypted_text);  
  RETURN plain_text;  
END;
```

Script Output | Query Result | Task completed in 0.043 seconds

Function DECRYPT\_TEXT compiled

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
```

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, several files are open: DB1.sql, Test.sql, Oracle db database.sql, User\_DB200, and EMPLOYEE. The EMPLOYEE tab is currently selected. Below the tabs is a toolbar with various icons. The main area is divided into two panes: 'Worksheet' on the left and 'Query Builder' on the right. The Worksheet pane contains the following SQL code:

```

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

```

The Query Result pane below shows a table of active sessions:

USERNAME	OSUSER	SID	SERIAL#	SPID	LOCKWAIT	STATUS	MODULE	MACHINE	PROGRAM	LOGON_TIME	LAST_CALL_E
1 DB200	juhig	64	57817 2680	(null)		ACTIVE	SQL Developer	DESKTOP-NBDGQV9	SQL Developer	11-NOV-2017 16:20:06	
2 (oracle)	oracle	69	24182 3048	(null)		ACTIVE	(null)	READE	ORACLE.EXE (J001)	11-NOV-2017 20:30:41	
3 (oracle)	oracle	4	63628 2484	(null)		ACTIVE	(null)	READE	ORACLE.EXE (VKTM)	22-OCT-2017 01:51:58	17
4 (oracle)	oracle	5	24748 2488	(null)		ACTIVE	(null)	READE	ORACLE.EXE (GEN0)	22-OCT-2017 01:51:58	17
5 (oracle)	oracle	6	23736 2492	(null)		ACTIVE	(null)	READE	ORACLE.EXE (MMAN)	22-OCT-2017 01:51:58	17
6 (oracle)	oracle	7	26141 2496	(null)		ACTIVE	(null)	READE	ORACLE.EXE (DIAG)	22-OCT-2017 01:51:58	17
7 (oracle)	oracle	8	32862500	(null)		ACTIVE	(null)	READE	ORACLE.EXE (DBRM)	22-OCT-2017 01:51:58	17

There are total 34 active sessions.

- To find the version of the database.

SELECT \*

FROM V\$VERSION;

The screenshot shows the Oracle SQL Developer interface. The top tab bar has files DB1.sql, Test.sql, Oracle db database.sql, User\_DB200, and EMPLOYEE open. The EMPLOYEE tab is selected. The Worksheet pane contains the following SQL query:

```

SELECT *
FROM V$VERSION;

```

The Query Result pane shows the output of the query:

BANNER	CON_ID
1 Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production	0
2 PL/SQL Release 12.1.0.2.0 - Production	0
3 CORE12.1.0.2.0Production	0
4 TNS for 64-bit Windows: Version 12.1.0.2.0 - Production	0
5 NLSRTL Version 12.1.0.2.0 - Production	0

- 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
```

```

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

```

Script Output | Query Result | Query Result 1 | Query Result 3 | All Rows Fetched: 28 in 13.826 seconds

SID	STATUS	PROCESS	SCHEMANAME	OSUSER	SQL_TEXT
1	74 INACTIVE	15436	DB281	Nikhil Sonar	/* + NO_PARALLEL */SELECT ROWID "ROWID", ORA_ROWSCN "ORA_ROWSCN", MATERIAL_ID
2	64 ACTIVE	20932	DB200	juhig	SELECT s.sid, s.status, s.process, s.schemaname, s.
3	34 ACTIVE	2744	SYS	oracle	SELECT OBJOID, CLSOID, (2*PRI + DECODE(BITAND(STATUS, 4), 0,
4	70 INACTIVE	7036	DB217	maiti	select ms.Statistic# stat,ms.value,sn.name from v\$mystat ms, v\$statname sn w
5	67 INACTIVE	15164	DB200	Bossypants	SELECT * FROM sales sa WHERE product_id IN (SELECT product_id FROM product)
6	33 INACTIVE	2752	DB217	nehak	SELECT ES_PLAYERS.PLAYER_NAME, ES_PLAYERS_ATT.OVERALL_RATING, ES_PLAYERS_ATT.I
7	39 INACTIVE	1856	DB217	nehak	SELECT ES_PLAYERS.PLAYER_NAME, ES_PLAYERS_ATT.OVERALL_RATING, ES_PLAYERS_ATT.I

SQL History | Messages - Log

The output shows result for both Active and Inactive sessions.

- To find the list of roles.

`SELECT * FROM DBA_ROLES;`

```

SELECT *
FROM DBA_ROLES;

```

Script Output | Query Result | Query Result 1 | Query Result 3 | Query Result 4 | All Rows Fetched: 92 in 0.046 seconds

ROLE	PASSWORD_REQUIRED	AUTHENTICATION_TYPE	COMMON	ORACLE_MAINTAINED
1 CONNECT	NO	NONE	YES	Y
2 RESOURCE	NO	NONE	YES	Y
3 DBA	NO	NONE	YES	Y
4 AUDIT_ADMIN	NO	NONE	YES	Y
5 AUDIT_VIEWER	NO	NONE	YES	Y
6 SELECT_CATALOG_ROLE	NO	NONE	YES	Y
7 EXECUTE_CATALOG_ROLE	NO	NONE	YES	Y
8 DELETE_CATALOG_ROLE	NO	NONE	YES	Y
9 CAPTURE_ADMIN	NO	NONE	YES	Y
10 EXP_FULL_DATABASE	NO	NONE	YES	Y
11 IMP_FULL_DATABASE	NO	NONE	YES	Y
12 CDB_DBA	NO	NONE	YES	Y
13 PDB_DBA	NO	NONE	YES	Y
14 LOGSTDBY_ADMINISTRATOR	NO	NONE	YES	Y

## EVALUATION TABLE

Topic	Description	Evaluation
<b>Entity- Relationship Diagram</b>	This section includes ERD and the description of relationships between the entities.	20
<b>Physical Database Design &amp; Data Creation and Import</b>	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
<b>SQL Queries</b>	This section includes some interesting SQL Queries created for the database.	20
<b>Performance Tuning</b>	In this section, we have used three methods to improve the performance of our database management system.	25
<b>Other Topics</b>	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