Advanced Database Management
Group Project Report-Online Grocery Store

Submitted By
Suganth Kumar Thangavel
Sai Seetha ram Nomula
Vamsi Krishna Vandana
Gauri Naik

Contents

INTRODUCTION:

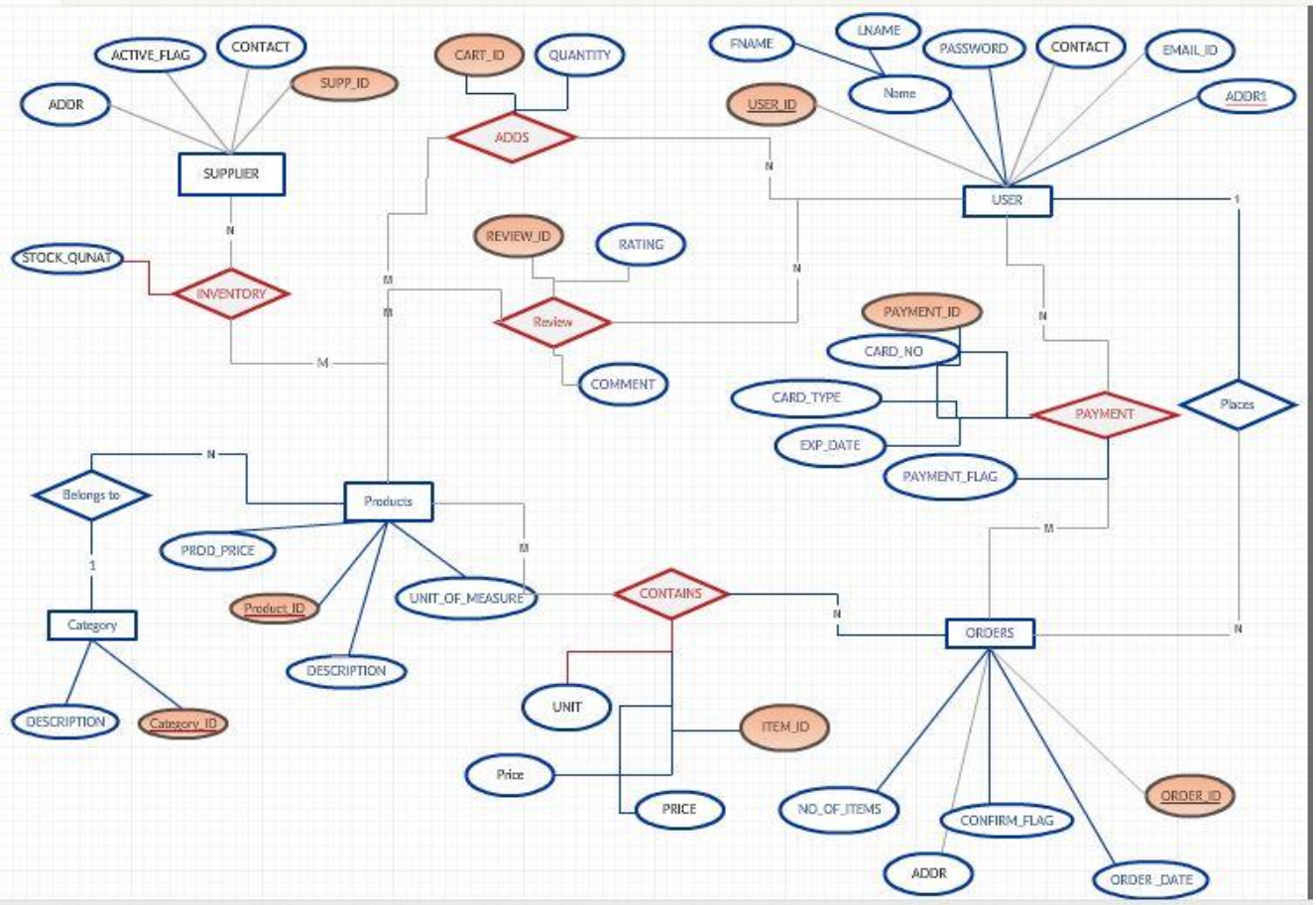
The Online Grocery Store database is the practical implementation of E-commerce for grocery goods to facilitate suppliers to sell their product under one window. This database aims to keep track of inventories, sales, online orders, product delivery and to allow close monitoring of many aspects such as order shipping, receiving and invoicing. Entities such as users, suppliers, products, orders, order items and payment are included in the database.

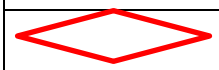
Assumptions and Information:

1. The payment status can be tracked by values ranging from (0-2) where 0 indicates payment failed, 1 indicates successful payment and 2 indicates in progress.
2. The confirmation of the user order can be tracked by CONFIRM flag from ORDERS table.
3. An order can be processed by multiple payments. i.e. one ORDER_ID can be associated with multiple PAYMENT_ID in case of online payment mode.
4. Suppliers status i.e. whether he supplies products or not at current time, can be tracked by ACTIVE_FLAG. '0' value indicated supplier is inactive and '1' value indicated supplier is active.

CONCEPTUAL DESIGN:

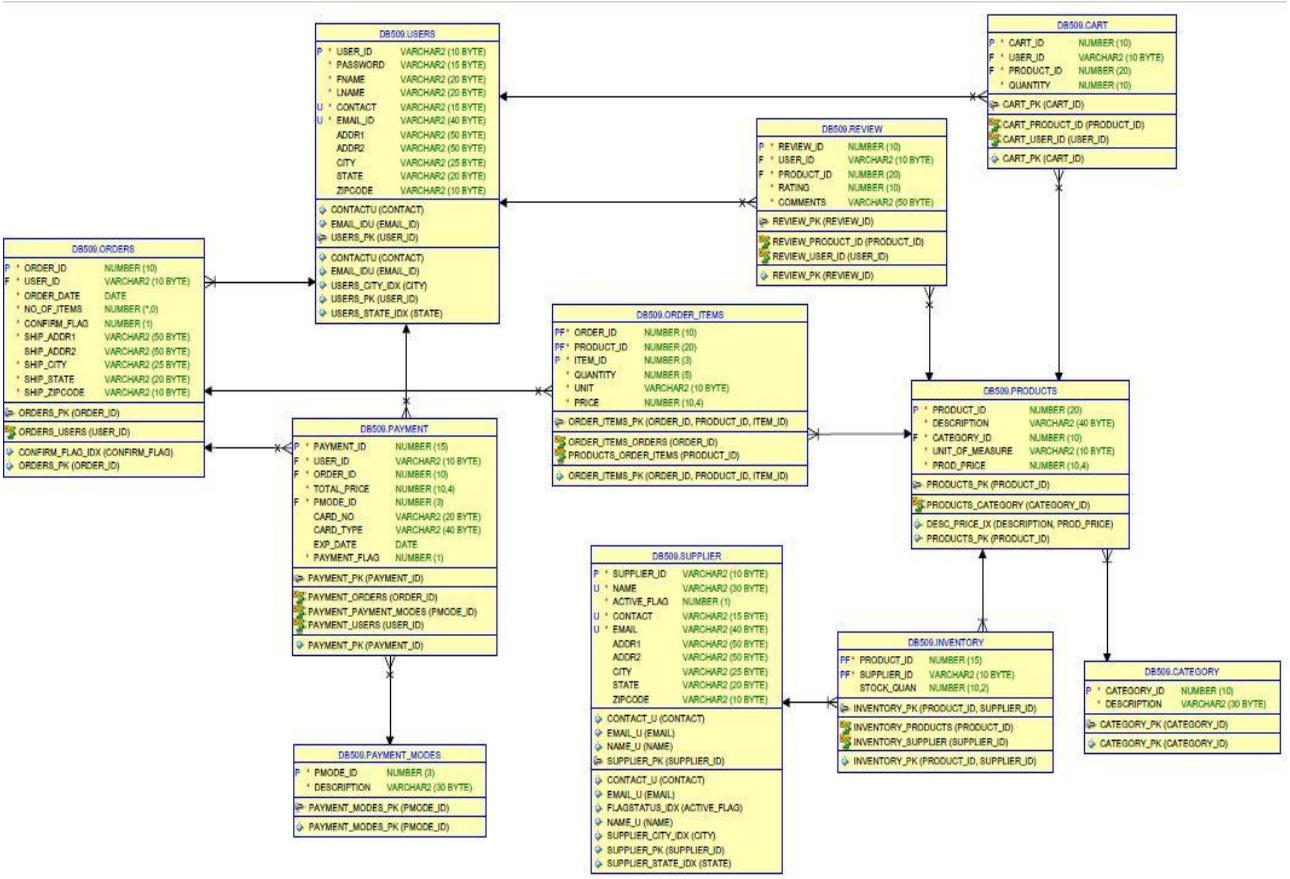
Initially, based on the requirements of the business, a conceptual design is made with few attributes and relations. While developing and implementing the database model, new attributes will be added as per the requirements and some of the redundant are deleted. When all the entity- relationships are well defined, we'll go ahead with the implementation of the project by developing database model. Below is the model developed with platform independent i.e. irrespective of the technology used, database management system version etc.

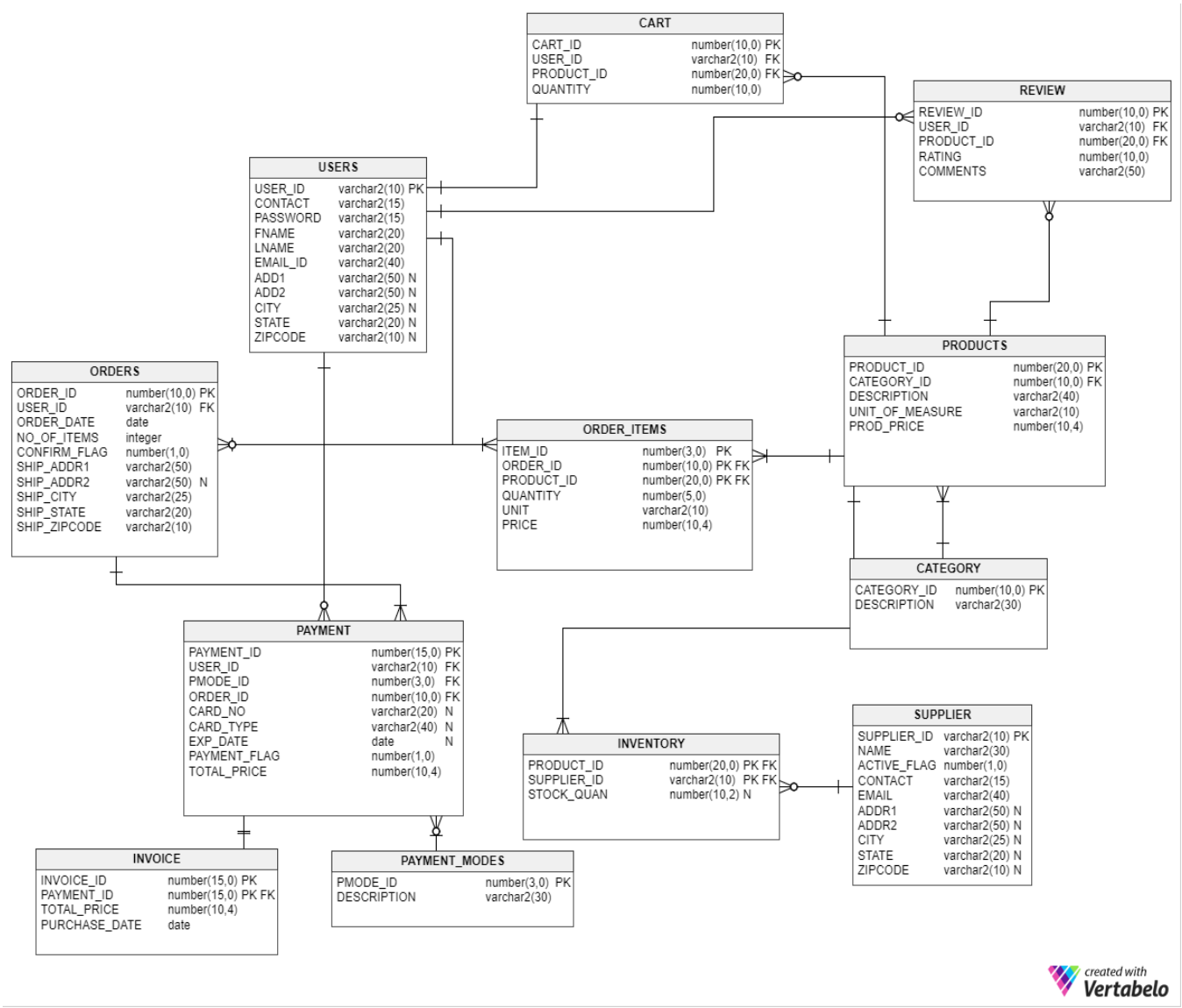


Symbols	Indication
	Relationship (m:n)
	Primary key
	Relationship (1:n)

LOGICAL DESIGN:

Once the requirements are finalized, a Business Requirements Document (BRD) is published i.e. document detailing all the information regarding entities, relationships between entities, keys, constraints etc. a logical model is developed. All the tables and their associated views, indexes etc. are created which are derived based on the logical model. Below is the diagram that shows the logical design developed based on the tables created and used along with its constraints.




CART
CART_ID number(10,0) PK
USER_ID varchar2(10) FK
PRODUCT_ID number(20,0) FK
QUANTITY number(10,0)
REVIEW
REVIEW_ID number(10,0) PK
USER_ID varchar2(10) FK
PRODUCT_ID number(20,0) FK
RATING number(10,0)
COMMENTS varchar2(50)
USERS
USER_ID varchar2(10) PK
CONTACT varchar2(15)
PASSWORD varchar2(15)
FNAME varchar2(20)
LNAME varchar2(20)
EMAIL_ID varchar2(40)
ADD1 varchar2(50) N
ADD2 varchar2(50) N
CITY varchar2(25) N
STATE varchar2(20) N
ZIPCODE varchar2(10) N
PRODUCTS
PRODUCT_ID number(20,0) PK
CATEGORY_ID number(10,0) FK
DESCRIPTION varchar2(40)
UNIT_OF_MEASURE varchar2(10)
PROD_PRICE number(10,4)
ORDERS
ORDER_ID number(10,0) PK
USER_ID varchar2(10) FK
ORDER_DATE date
NO_OF_ITEMS integer
CONFIRM_FLAG number(1,0)
SHIP_ADDR1 varchar2(50)
SHIP_ADDR2 varchar2(50) N
SHIP_CITY varchar2(25)
SHIP_STATE varchar2(20)
SHIP_ZIPCODE varchar2(10)
ORDER_ITEMS
ITEM_ID number(3,0) PK
ORDER_ID number(10,0) PK FK
PRODUCT_ID number(20,0) PK FK
QUANTITY number(5,0)
UNIT varchar2(10)
PRICE number(10,4)
CATEGORY
CATEGORY_ID number(10,0) PK
DESCRIPTION varchar2(30)
PAYMENT
PAYMENT_ID number(15,0) PK
USER_ID varchar2(10) FK
PMODE_ID number(3,0) FK
ORDER_ID number(10,0) FK
CARD_NO varchar2(20) N
CARD_TYPE varchar2(40) N
EXP_DATE date N
PAYMENT_FLAG number(1,0)
TOTAL_PRICE number(10,4)
INVENTORY
PRODUCT_ID number(20,0) PK FK
SUPPLIER_ID varchar2(10) PK FK
STOCK_QUAN number(10,2) N
SUPPLIER
SUPPLIER_ID varchar2(10) PK
NAME varchar2(30)
ACTIVE_FLAG number(1,0)
CONTACT varchar2(15)
EMAIL varchar2(40)
ADDR1 varchar2(50) N
ADDR2 varchar2(50) N
CITY varchar2(25) N
STATE varchar2(20) N
ZIPCODE varchar2(10) N
INVOICE
INVOICE_ID number(15,0) PK
PAYMENT_ID number(15,0) PK FK
TOTAL_PRICE number(10,4)
PURCHASE_DATE date
PAYMENT_MODES
PMODE_ID number(3,0) PK
DESCRIPTION varchar2(30)
created with Vertabelo

PHYSICAL DATABASE DESIGN:

Physical Database design refers to the conversion of relations in the logical data base into corresponding database objects. We are using the SQL Developer for the creation and development of the database objects based on the logical model developed. All the requirements are gathered based on the business and logical model is developed. Physical design involved in creating various database objects like tables, indexes and views etc. based on entities and relations in logical design.

TABLE: USERS

```
CREATE TABLE users
 (
   user_id  VARCHAR2(10) NOT NULL,
   password VARCHAR2(15) NOT NULL,
   fname    VARCHAR2(20) NOT NULL,
   lname    VARCHAR2(20) NOT NULL,
   contact  VARCHAR2(15) NOT NULL,
   email_id VARCHAR2(40) NOT NULL,
   addr1    VARCHAR2(50) NULL,
   addr2    VARCHAR2(50) NULL,
   city     VARCHAR2(25) NULL,
   state    VARCHAR2(20) NULL,
   zipcode  VARCHAR2(10) NULL,
   CONSTRAINT contactu UNIQUE (contact),
   CONSTRAINT email_idu UNIQUE (email_id),
   CONSTRAINT users_pk PRIMARY KEY (user_id)
 );
```

DATA DICTIONARY:

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	USER_ID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	PASSWORD	VARCHAR2(15 BYTE)	No	(null)	2	(null)
3	FNAME	VARCHAR2(20 BYTE)	No	(null)	3	(null)
4	LNAME	VARCHAR2(20 BYTE)	No	(null)	4	(null)
5	CONTACT	VARCHAR2(15 BYTE)	No	(null)	5	(null)
6	EMAIL_ID	VARCHAR2(40 BYTE)	No	(null)	6	(null)
7	ADDR1	VARCHAR2(50 BYTE)	Yes	(null)	7	(null)
8	ADDR2	VARCHAR2(50 BYTE)	Yes	(null)	8	(null)
9	CITY	VARCHAR2(25 BYTE)	Yes	(null)	9	(null)
10	STATE	VARCHAR2(20 BYTE)	Yes	(null)	10	(null)
11	ZIPCODE	VARCHAR2(10 BYTE)	Yes	(null)	11	(null)

CONSTRAINTS:

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

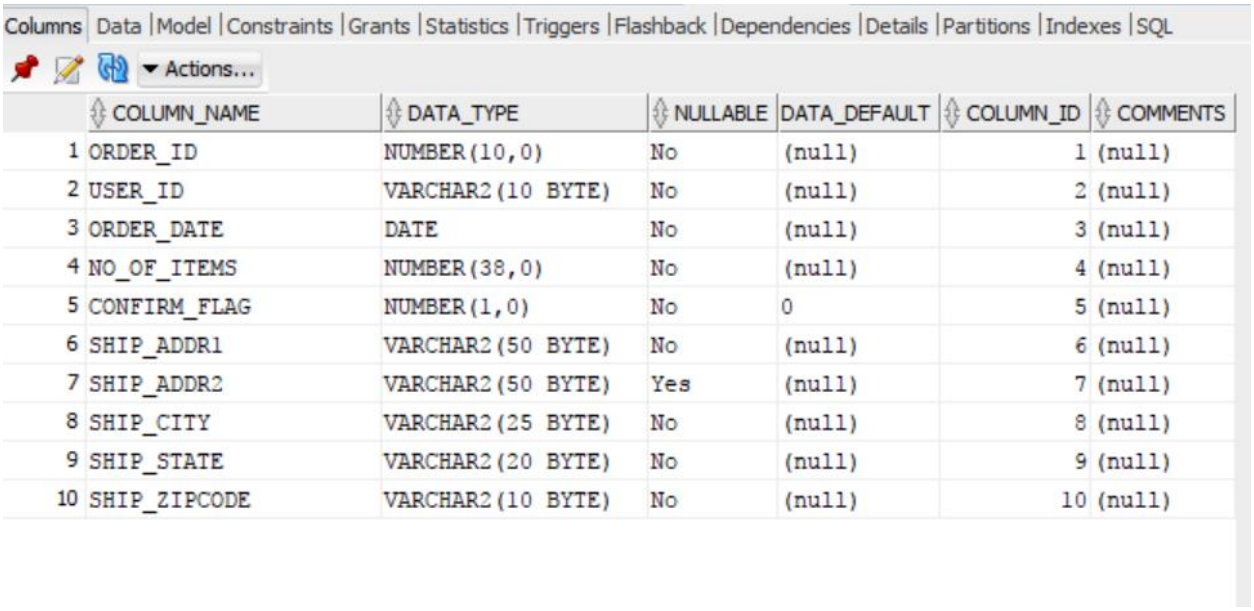
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE	VAL
1	CONTACTU	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
2	EMAIL_IDU	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
3	SYS_C0065391	Check	"USER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
4	SYS_C0065392	Check	"PASSWORD" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
5	SYS_C0065393	Check	"FNAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
6	SYS_C0065394	Check	"LNAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
7	SYS_C0065395	Check	"CONTACT" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
8	SYS_C0065396	Check	"EMAIL_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI
9	USERS_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE	VALI

TABLE: ORDERS

```
CREATE TABLE orders
 (
   order_id     NUMBER(10, 0) NOT NULL,
   user_id      VARCHAR2(10) NOT NULL,
   order_date   DATE NOT NULL,
   no_of_items  INTEGER NOT NULL,
   confirm_flag NUMBER(1, 0) DEFAULT 0 NOT NULL,
   ship_addr1   VARCHAR2(50) NOT NULL,
   ship_addr2   VARCHAR2(50) NULL,
   ship_city    VARCHAR2(25) NOT NULL,
   ship_state   VARCHAR2(20) NOT NULL,
   ship_zipcode VARCHAR2(10) NOT NULL,
   CONSTRAINT flag_check CHECK (confirm_flag IN (1, 2)),
   CONSTRAINT orders_pk PRIMARY KEY (order_id)
 );
```

DATA DICTIONARY:



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	ORDER_ID	NUMBER(10,0)	No	(null)	1	(null)
2	USER_ID	VARCHAR2(10 BYTE)	No	(null)	2	(null)
3	ORDER_DATE	DATE	No	(null)	3	(null)
4	NO_OF_ITEMS	NUMBER(38,0)	No	(null)	4	(null)
5	CONFIRM_FLAG	NUMBER(1,0)	No	0	5	(null)
6	SHIP_ADDR1	VARCHAR2(50 BYTE)	No	(null)	6	(null)
7	SHIP_ADDR2	VARCHAR2(50 BYTE)	Yes	(null)	7	(null)
8	SHIP_CITY	VARCHAR2(25 BYTE)	No	(null)	8	(null)
9	SHIP_STATE	VARCHAR2(20 BYTE)	No	(null)	9	(null)
10	SHIP_ZIPCODE	VARCHAR2(10 BYTE)	No	(null)	10	(null)

FOREIGN KEY:

```
ALTER TABLE orders
  ADD CONSTRAINT orders_users FOREIGN KEY (user_id) REFERENCES users (user_id);
```

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	FLAG_CHECK	Check	CONFIRM_FLAG IN (1,2)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
2	ORDERS_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
3	ORDERS_USERS	Foreign_Key	(null)	DB579	USERS	USERS_PK	NO ACTION	ENABLED	NOT DEFERRABL
4	SYS_C0065400	Check	"ORDER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
5	SYS_C0065401	Check	"USER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
6	SYS_C0065402	Check	"ORDER_DATE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
7	SYS_C0065403	Check	"NO_OF_ITEMS" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
8	SYS_C0065404	Check	"CONFIRM_FLAG" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL
9	SYS_C0065405	Check	"SHIP_ADDR1" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABL

TABLE: ORDER_ITEMS

```
CREATE TABLE order_items
 (
   order_id   NUMBER(10, 0) NOT NULL,
   product_id NUMBER(20, 0) NOT NULL,
   item_id    NUMBER(3, 0) NOT NULL,
   quantity   NUMBER(5, 0) NOT NULL,
   unit       VARCHAR2(10) NOT NULL,
   price      NUMBER(10, 4) NOT NULL,
   CONSTRAINT order_items_pk PRIMARY KEY (order_id, product_id, item_id)
 );
```

DATA DICTIONARY:

Start Page | Myconnection1~1 | Myconnection~3 | ORDER_ITEMS

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

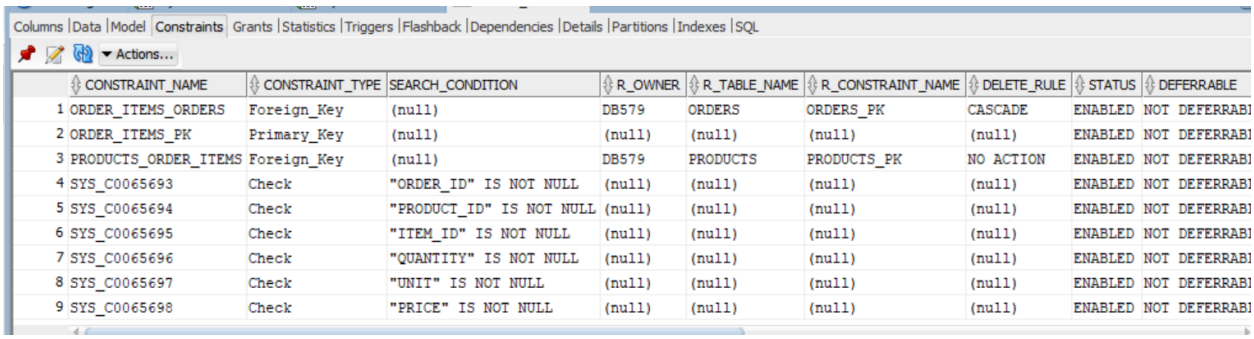
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ORDER_ID	NUMBER(10,0)	No	(null)	1	(null)
2	PRODUCT_ID	NUMBER(20,0)	No	(null)	2	(null)
3	ITEM_ID	NUMBER(3,0)	No	(null)	3	(null)
4	QUANTITY	NUMBER(5,0)	No	(null)	4	(null)
5	UNIT	VARCHAR2(10 BYTE)	No	(null)	5	(null)
6	PRICE	NUMBER(10,4)	No	(null)	6	(null)

FOREIGN KEY:

```
ALTER TABLE order_items
  ADD CONSTRAINT order_items_orders FOREIGN KEY (order_id) REFERENCES orders
(order_id) ON DELETE CASCADE;

ALTER TABLE order_items
  ADD CONSTRAINT products_order_items FOREIGN KEY (product_id) REFERENCES
  products (product_id);
```

CONSTRAINTS:



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

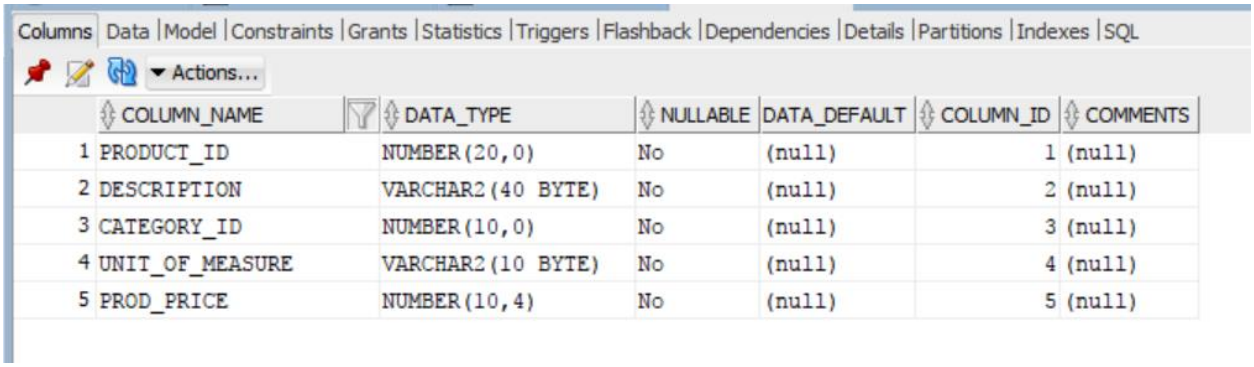
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	ORDER_ITEMS_ORDERS	Foreign_Key	(null)	DB579	ORDERS	ORDERS_PK	CASCADE	ENABLED	NOT DEFERRABI
2	ORDER_ITEMS_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
3	PRODUCTS_ORDER_ITEMS	Foreign_Key	(null)	DB579	PRODUCTS	PRODUCTS_PK	NO ACTION	ENABLED	NOT DEFERRABI
4	SYS_C0065693	Check	"ORDER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
5	SYS_C0065694	Check	"PRODUCT_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
6	SYS_C0065695	Check	"ITEM_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
7	SYS_C0065696	Check	"QUANTITY" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
8	SYS_C0065697	Check	"UNIT" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI
9	SYS_C0065698	Check	"PRICE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABI

TABLE: PRODUCTS

```
CREATE TABLE products
  (
    product_id      NUMBER(20, 0) NOT NULL,
    description     VARCHAR2(40) NOT NULL,
    category_id     NUMBER(10, 0) NOT NULL,
    unit_of_measure VARCHAR2(10) NOT NULL,
    prod_price      NUMBER(10, 4) NOT NULL,
    CONSTRAINT products_pk PRIMARY KEY (product_id)
  );
```

DATA DICTIONARY:



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

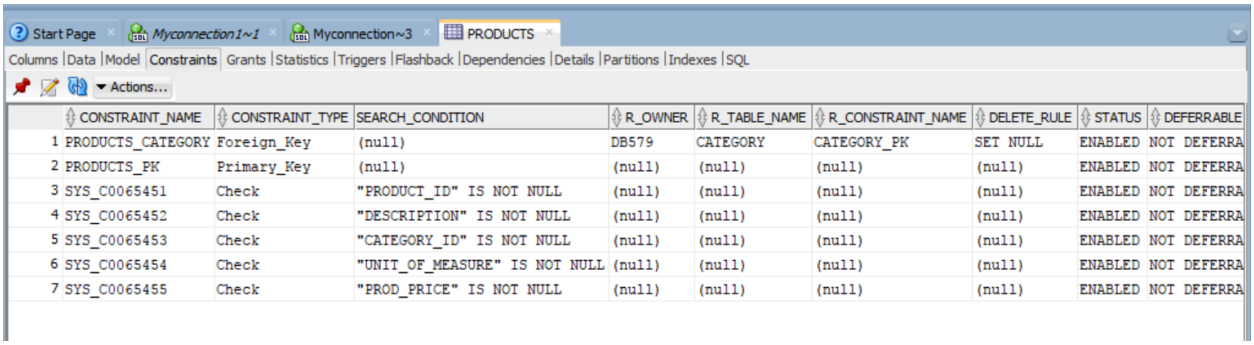
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PRODUCT_ID	NUMBER(20,0)	No	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(40 BYTE)	No	(null)	2	(null)
3	CATEGORY_ID	NUMBER(10,0)	No	(null)	3	(null)
4	UNIT_OF_MEASURE	VARCHAR2(10 BYTE)	No	(null)	4	(null)
5	PROD_PRICE	NUMBER(10,4)	No	(null)	5	(null)

FOREIGN KEY:

```
ALTER TABLE products
  ADD CONSTRAINT products_category FOREIGN KEY (category_id) REFERENCES
 category
  (category_id) ON DELETE SET NULL;
```

CONSTRAINTS:



Start Page | Myconnection1~1 | Myconnection~3 | PRODUCTS

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

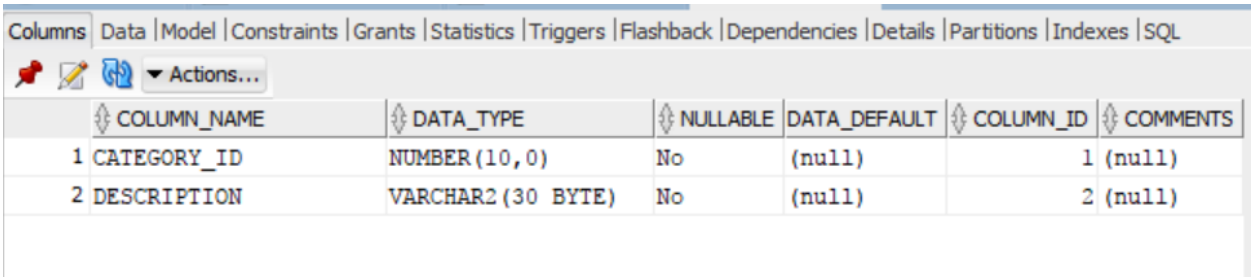
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	PRODUCTS_CATEGORY	Foreign_Key	(null)	DB579	CATEGORY	CATEGORY_PK	SET NULL	ENABLED	NOT DEFERRA
2	PRODUCTS_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA
3	SYS_C0065451	Check	"PRODUCT_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA
4	SYS_C0065452	Check	"DESCRIPTION" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA
5	SYS_C0065453	Check	"CATEGORY_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA
6	SYS_C0065454	Check	"UNIT_OF_MEASURE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA
7	SYS_C0065455	Check	"PROD_PRICE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRA

TABLE: CATEGORY

```
CREATE TABLE category
  (
    category_id NUMBER(10, 0) NOT NULL,
    description VARCHAR2(30) NOT NULL,
    CONSTRAINT category_pk PRIMARY KEY (category_id)
  );
```

DATA DICTIONARY:

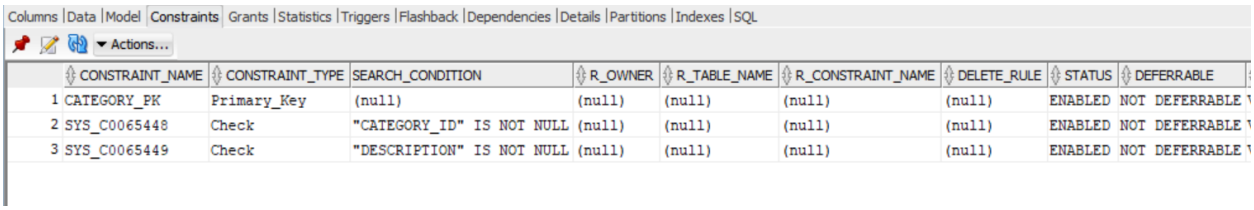


Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CATEGORY_ID	NUMBER(10,0)	No	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(30 BYTE)	No	(null)	2	(null)

CONSTRAINTS:



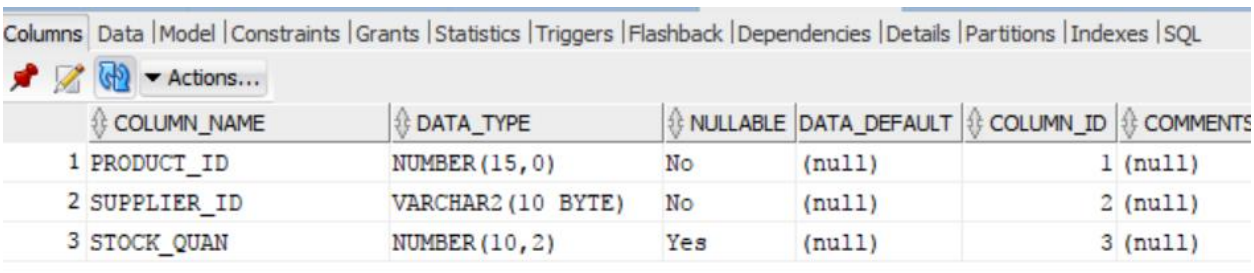
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	CATEGORY_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	SYS_C0065448	Check	"CATEGORY_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
3	SYS_C0065449	Check	"DESCRIPTION" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

TABLE: INVENTORY:

```
CREATE TABLE inventory
 (
   product_id  NUMBER(15, 0) NOT NULL,
   supplier_id VARCHAR2(10) NOT NULL,
   stock_quan  NUMBER(10, 2) NULL,
   CONSTRAINT inventory_pk PRIMARY KEY (product_id, supplier_id)
 );
```



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

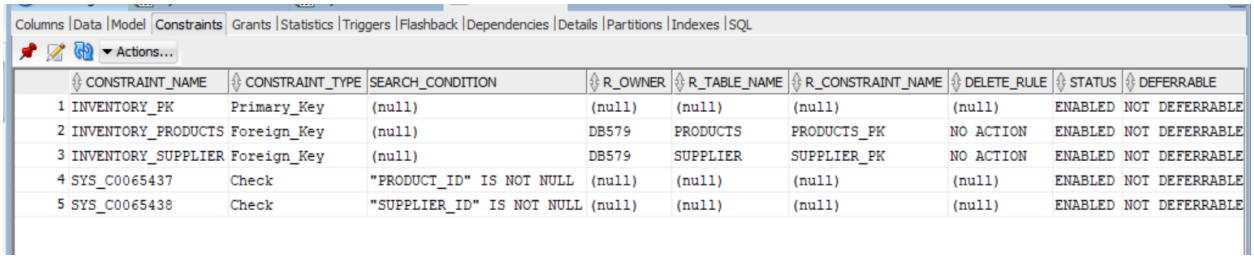
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PRODUCT_ID	NUMBER(15,0)	No	(null)	1	(null)
2	SUPPLIER_ID	VARCHAR2(10 BYTE)	No	(null)	2	(null)
3	STOCK_QUAN	NUMBER(10,2)	Yes	(null)	3	(null)

FOREIGN KEY:

```
ALTER TABLE inventory
 ADD CONSTRAINT inventory_products FOREIGN KEY (product_id) REFERENCES
products
 (product_id);

ALTER TABLE inventory
 ADD CONSTRAINT inventory_supplier FOREIGN KEY (supplier_id) REFERENCES
 supplier (supplier_id);
```

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	INVENTORY_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	INVENTORY_PRODUCTS	Foreign_Key	(null)	DB579	PRODUCTS	PRODUCTS_PK	NO ACTION	ENABLED	NOT DEFERRABLE
3	INVENTORY_SUPPLIER	Foreign_Key	(null)	DB579	SUPPLIER	SUPPLIER_PK	NO ACTION	ENABLED	NOT DEFERRABLE
4	SYS_C0065437	Check	"PRODUCT_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
5	SYS_C0065438	Check	"SUPPLIER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

TABLE: SUPPLIER

```
CREATE TABLE supplier
 (
   supplier_id VARCHAR2(10) NOT NULL,
   NAME        VARCHAR2(30) NOT NULL,
   active_flag NUMBER(1, 0) DEFAULT 0 NOT NULL,
   contact     VARCHAR2(15) NOT NULL,
   email       VARCHAR2(40) NOT NULL,
   addr1       VARCHAR2(50) NULL,
   addr2       VARCHAR2(50) NULL,
   city        VARCHAR2(25) NULL,
   state       VARCHAR2(20) NULL,
   zipcode     VARCHAR2(10) NULL,
   CONSTRAINT contact_u UNIQUE (contact),
   CONSTRAINT email_u UNIQUE (email),
   CONSTRAINT name_u UNIQUE (NAME),
   CONSTRAINT act_check CHECK (active_flag IN(0, 1)),
   CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
 );
```

DATA DICTIONARY:

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	PRODUCT_ID	NUMBER(15,0)	No	(null)	1	(null)
2	SUPPLIER_ID	VARCHAR2(10 BYTE)	No	(null)	2	(null)
3	STOCK_QUAN	NUMBER(10,2)	Yes	(null)	3	(null)

CONSTRAINTS:

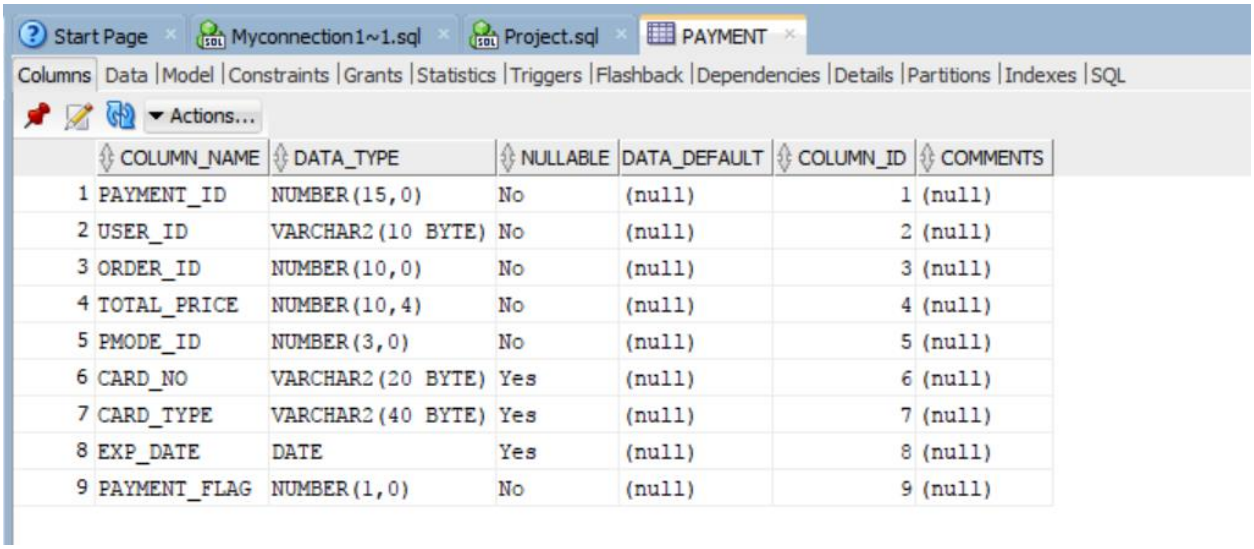
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRABLE
1	ACT_CHECK	Check	ACTIVE_FLAG IN(0,1)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
2	CONTACT_U	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
3	EMAIL_U	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
4	NAME_U	Unique	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
5	SUPPLIER_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
6	SYS_C0065422	Check	"SUPPLIER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
7	SYS_C0065423	Check	"NAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
8	SYS_C0065424	Check	"ACTIVE_FLAG" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE
9	SYS_C0065425	Check	"CONTACT" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE

TABLE: PAYMENT:

```
CREATE TABLE payment
 (
   payment_id   NUMBER(15, 0) NOT NULL,
   user_id      VARCHAR2(10) NOT NULL,
   order_id     NUMBER(10, 0) NOT NULL,
  total_price     NUMBER(10, 4) NOT NULL,
   pmode_id     NUMBER(3, 0) NOT NULL,
   card_no      VARCHAR2(20) NULL,
   card_type    VARCHAR2(40) NULL,
   exp_date     DATE NULL,
   payment_flag NUMBER(1, 0) NOT NULL,
   CONSTRAINT payflag_check CHECK (payment_flag IN (0, 1, 2)),
   CONSTRAINT payment_pk PRIMARY KEY (payment_id)
 );
```

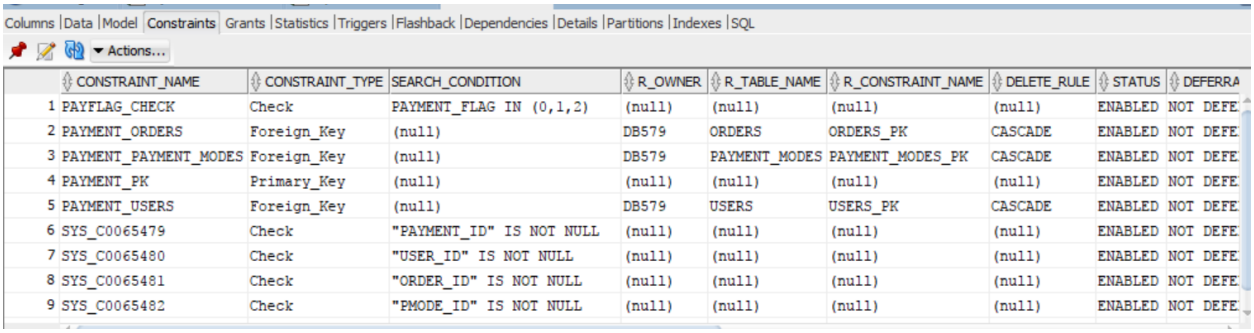
DATA DICTIONARY:



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENT_ID	NUMBER(15,0)	No	(null)	1	(null)
2	USER_ID	VARCHAR2(10 BYTE)	No	(null)	2	(null)
3	ORDER_ID	NUMBER(10,0)	No	(null)	3	(null)
4	TOTAL_PRICE	NUMBER(10,4)	No	(null)	4	(null)
5	PMODE_ID	NUMBER(3,0)	No	(null)	5	(null)
6	CARD_NO	VARCHAR2(20 BYTE)	Yes	(null)	6	(null)
7	CARD_TYPE	VARCHAR2(40 BYTE)	Yes	(null)	7	(null)
8	EXP_DATE	DATE	Yes	(null)	8	(null)
9	PAYMENT_FLAG	NUMBER(1,0)	No	(null)	9	(null)

```
ALTER TABLE payment
  ADD CONSTRAINT payment_orders FOREIGN KEY (order_id) REFERENCES orders (
  order_id) ON DELETE CASCADE;


ALTER TABLE payment
  ADD CONSTRAINT payment_payment_modes FOREIGN KEY (pmode_id) REFERENCES
  payment_modes (pmode_id) ON DELETE CASCADE;


ALTER TABLE payment
  ADD CONSTRAINT payment_users FOREIGN KEY (user_id) REFERENCES users (user_id)

  ON DELETE CASCADE;
```

CONSTRAINTS:



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

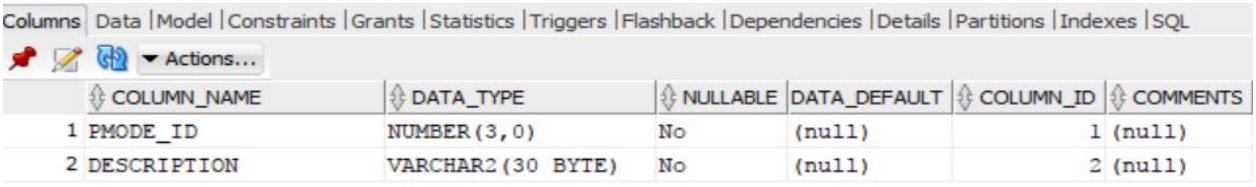
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERRA
1	PAYFLAG_CHECK	Check	PAYMENT_FLAG IN (0,1,2)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE
2	PAYMENT_ORDERS	Foreign_Key	(null)	DB579	ORDERS	ORDERS_PK	CASCADE	ENABLED	NOT DEFE
3	PAYMENT_PAYMENT_MODES	Foreign_Key	(null)	DB579	PAYMENT_MODES	PAYMENT_MODES_PK	CASCADE	ENABLED	NOT DEFE
4	PAYMENT_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE
5	PAYMENT_USERS	Foreign_Key	(null)	DB579	USERS	USERS_PK	CASCADE	ENABLED	NOT DEFE
6	SYS_C0065479	Check	"PAYMENT_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE
7	SYS_C0065480	Check	"USER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE
8	SYS_C0065481	Check	"ORDER_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE
9	SYS_C0065482	Check	"PMODE_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFE

TABLE: PAYMENT_MODES:

```
CREATE TABLE payment_modes
 (
   pmode_id    NUMBER(3, 0) NOT NULL,
   description VARCHAR2(30) NOT NULL,
   CONSTRAINT payment_modes_pk PRIMARY KEY (pmode_id)
 );
```

DATA DICTIONARY:

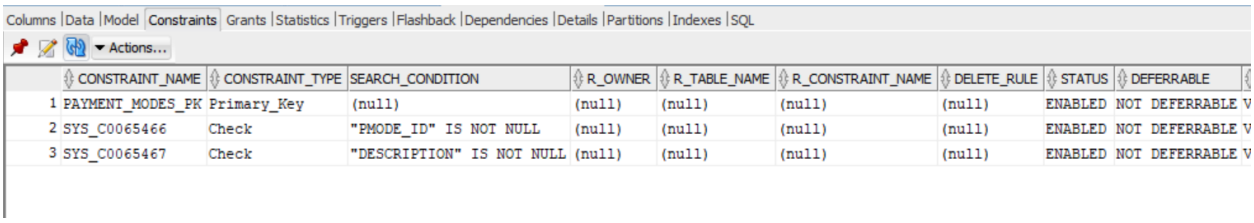


Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PMODE_ID	NUMBER(3,0)	No	(null)	1	(null)
2	DESCRIPTION	VARCHAR2(30 BYTE)	No	(null)	2	(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	PAYMENT_MODES_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE V
2	SYS_C0065466	Check	"PMODE_ID" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE V
3	SYS_C0065467	Check	"DESCRIPTION" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEFERRABLE V

TABLE: CART

```
CREATE TABLE cart
 (
   cart_id    NUMBER(10, 0) NOT NULL,
   user_id    VARCHAR2(10) NOT NULL,
   product_id NUMBER(20, 0) NOT NULL,
   quantity   NUMBER(10, 0) NOT NULL,
   CONSTRAINT cart_pk PRIMARY KEY (cart_id)
 );
```

CONSTRAINTS:

```
ALTER TABLE cart
 ADD CONSTRAINT cart_user_id FOREIGN KEY (user_id) REFERENCES users (user_id)
 ON DELETE CASCADE;

ALTER TABLE cart
 ADD CONSTRAINT cart_product_id FOREIGN KEY (product_id) REFERENCES products (
 product_id) ON DELETE CASCADE;
```

TABLE AND CONSTRAINTS: REVIEW

```
CREATE TABLE review
 (
   review_id  NUMBER(10, 0) NOT NULL,
   user_id    VARCHAR2(10) NOT NULL,
   product_id NUMBER(20, 0) NOT NULL,
   rating     NUMBER(10, 0) NOT NULL,
   comments   VARCHAR2(50) NOT NULL,
   CONSTRAINT review_pk PRIMARY KEY (review_id),
   CONSTRAINT review_user_id FOREIGN KEY (user_id) REFERENCES users (user
_id)
   ON DELETE CASCADE,
   CONSTRAINT review_product_id FOREIGN KEY (product_id) REFERENCES

products ( product_id) ON DELETE CASCADE
 );
```

DATA GENERATION AND LOADING:

The data is created mainly from data generating website https://www.mockaroo.com/ .

The grocery product data is scraped from the web. We also wrote Python scripts to generate Order Item details table from orders and products table. Below is the Python script used to create

Data is loaded for below tables:

Reviews and Cart data is not loaded.

TABLE	Number of Records
USERS	2000
ORDERS	2000
ORDER_ITEMS	4998
PRODUCTS	10527
CATEGORY	304
INVENTORY	19889
SUPPLIER	2000
PAYMENT	2322
PAYMENT_MODES	3

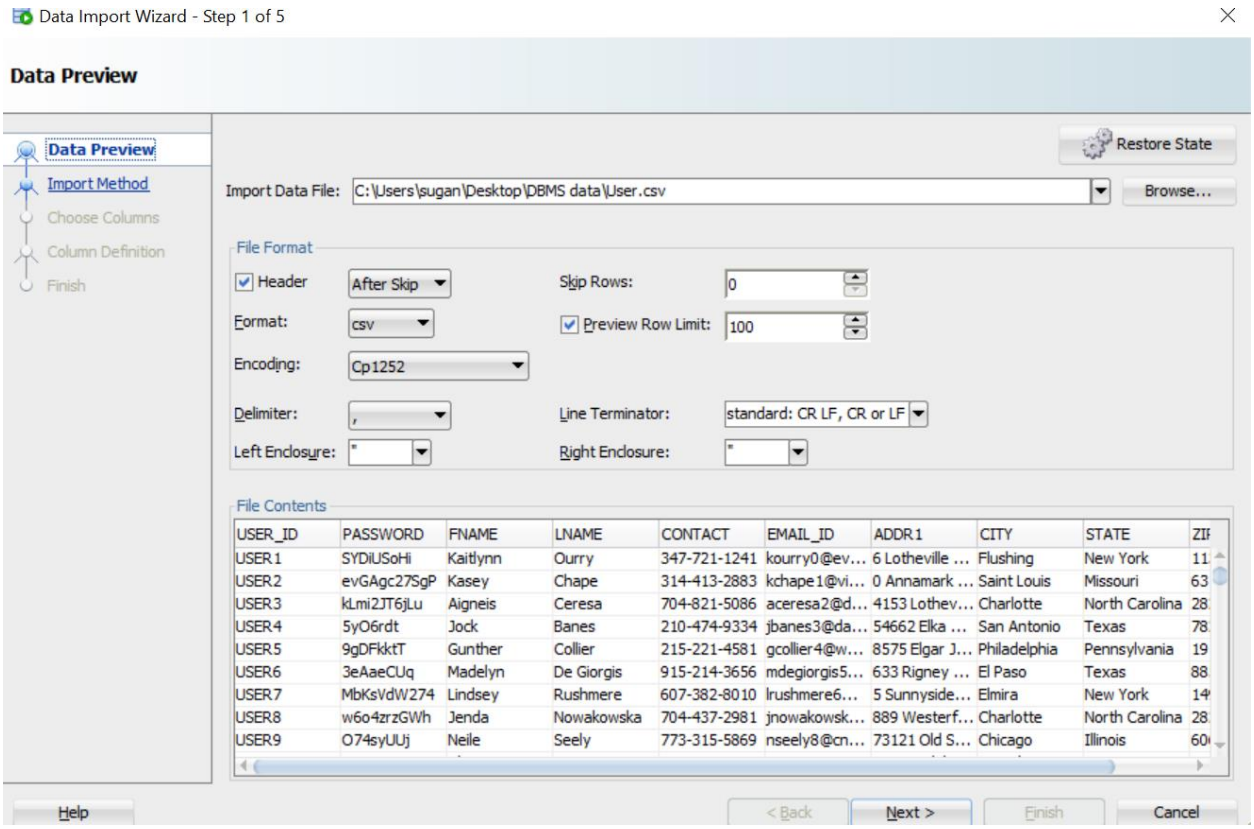
Python script used to load Order item tables:

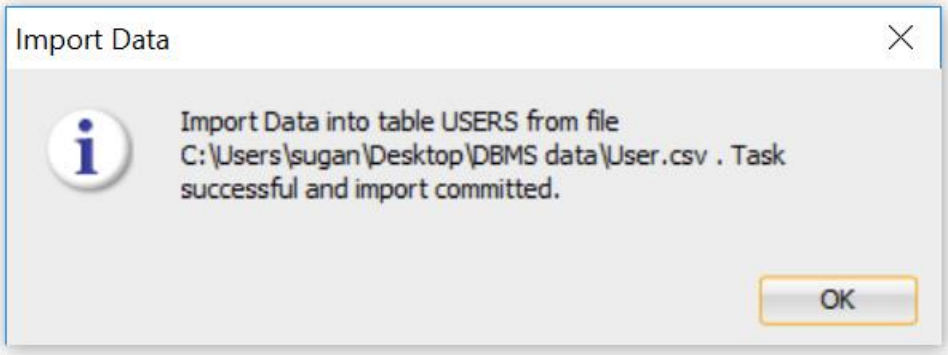
```
orders = pd.ExcelFile('orderid.xlsx')
product = pd.ExcelFile('ORDERS_ITEMS.xlsx')
```

In [ ]:
```
product = product.parse('Sheet1')
orders = orders.parse('Sheet1')
orderlist = orders['ORDER_ID']
f1_data = pd.DataFrame(columns = ('ORDER_ID','PRODUCT_ID','ITEM_ID','QUANTITY','UNIT','PRICE'))
final_data = f1_data
```

In [56]:
```
for i in orderlist:
    count = np.random.randint(1,5)
    for j in range(1,count+1):
        oid = i
        ran = np.random.randint(1,len(product['PRODUCT_ID']))
        pid = product['PRODUCT_ID'][ran]
        itid = j
        q = np.random.randint(1,10)
        qt = q
        ut = product['UNIT'][ran]
        pr = product['PRICE'][ran]*q
        temp = pd.DataFrame([[oid,pid,itid,qt,ut,pr]],columns=('ORDER_ID','PRODUCT_ID','ITEM_ID','QUANT
        final_data = final_data.append(temp,ignore_index=True)
```

The data is loaded into DB using import wizard as shown below.





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.

B + TREE INDEXES:

Real case scenario: To improve the speed of the data access or to decrease the amount of time to retrieve information by constructing B+ tree index on products table. This indexing strategy will be helpful when users of grocery store search information online based on a text in a string (i.e. Chocolate in Pastry - Chocolate Marble Tea). This indexing will be done on the products table.

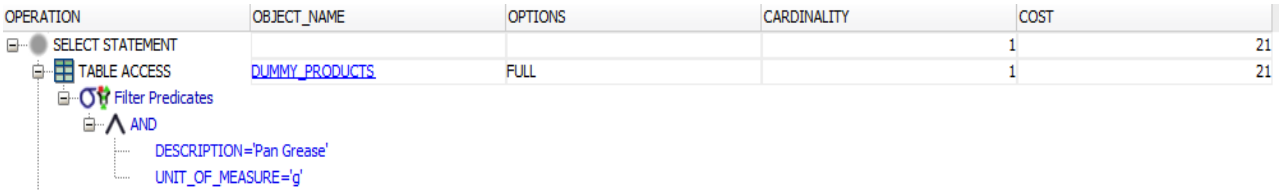
Steps for analyzing the cost benefits of using user defined indexes.

Analyzing the execution plan of the query which will search for a specific product in the database.

a) **When a specific product is being searched.**

Before indexing:

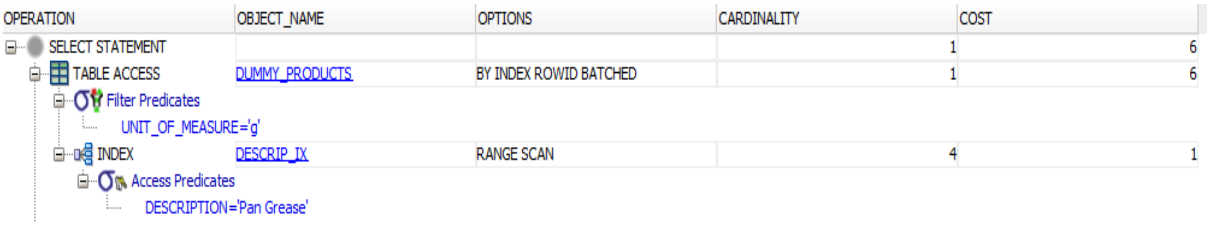
```
SELECT *
FROM   products
WHERE  description = 'Pan Grease'
       AND unit_of_measure = 'g';
```



Creating Index:

```
CREATE INDEX descrip_ix
  ON  products (description);
```

After indexing:

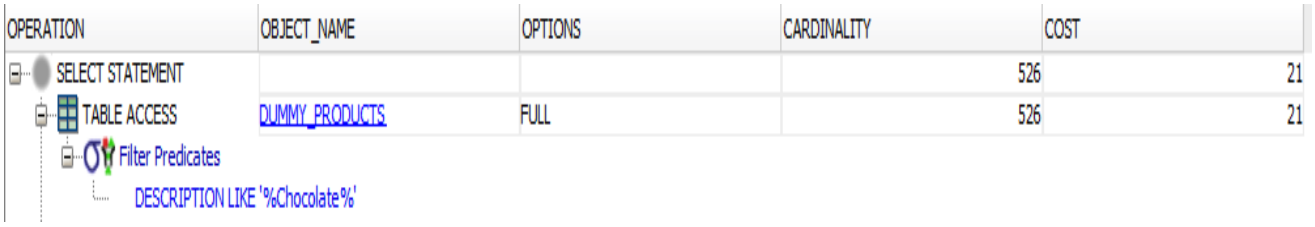


From the above explain plan of the query, it can be clearly seen that index is used in the operation to search for a specific product, and due to the presence of index the query cost has been decreased from 21 to 6.

b) **When a specific string in the product description is being searched.**

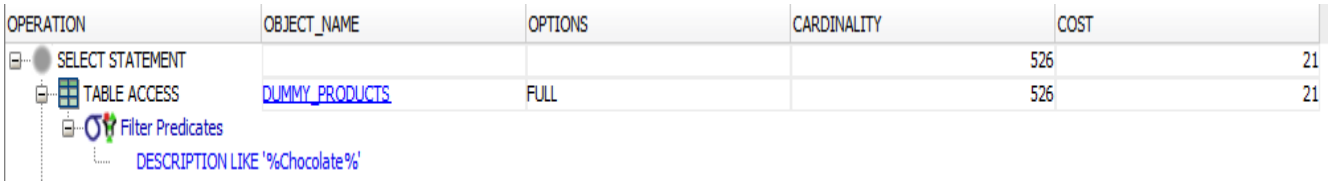
```
SELECT *
FROM   products
WHERE  description LIKE '%Chocolate%';
```

Before indexing



```
CREATE INDEX descrip_ix
  ON  products (description);
```

After indexing:



Here in the above case for specific string in a product description**,** there is no change in the cost because B+ tree indexing uses binary search algorithm, the algorithm sorts the description based on the first letter used in the description, as we are searching for a specific part of text, the indexing strategy will not be of any help.

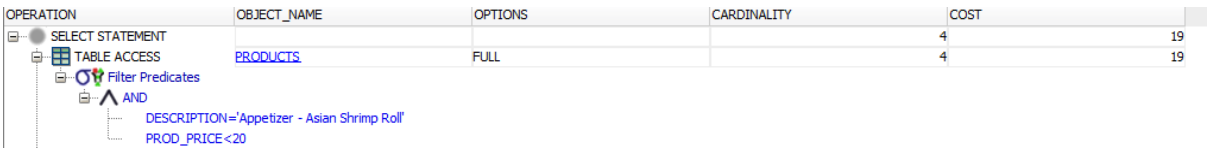
Multi column Based Indexing:

Real case scenario: Sorting the products based on their product description and product price. In Real case scenario, usually customers use product description and the prices of products to buy them based on their payment capability, so it would be wise to use an multi column indexing for faster output of results to customers for providing a better customer experience.

SQL query for selecting a product based on the price:

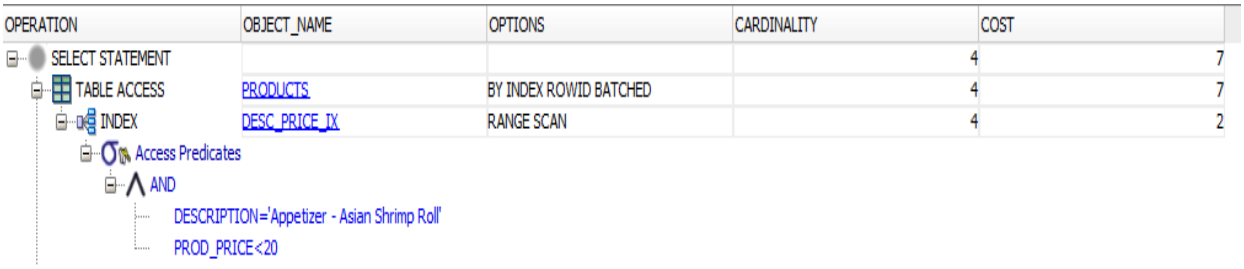
```
SELECT *
FROM   products
WHERE  description = 'Appetizer - Asian Shrimp Roll'
     AND prod_price < 20;
```

Before indexing



```
CREATE INDEX descrip_price_ix
  ON  products (price);
```

After indexing



In the above two steps we can see that, indexing on frequently used combination of columns has reduced the cost of the query as well as the execution time of the query.

BITMAP INDEXES:

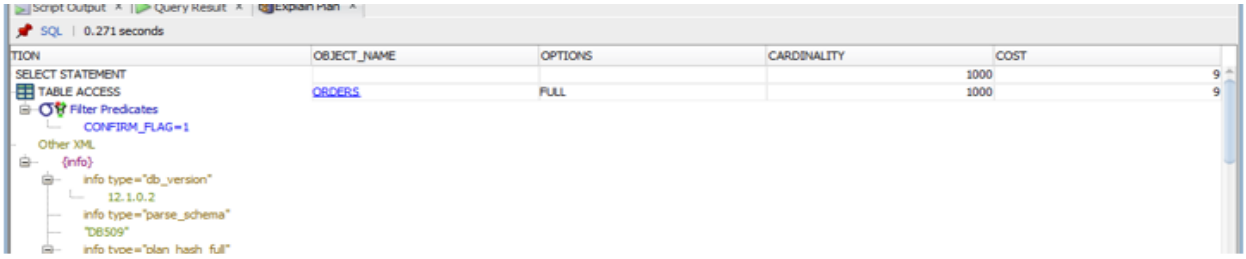
Real case scenario: To improve the speed of the data access or to decrease the amount of time to retrieve information by constructing BITMAP index on products table. This indexing strategy will be helpful when we want to find out the order_ids for the orders which are confirmed from the orders table. This indexing will be done on the Confirmed_flag column which belongs to Orders table.

Steps for analyzing the cost benefits of using user defined indexes.

Analyzing the execution plan of the query which will search for a specific product in the database.

c) When a confirmed order_id is being searched.

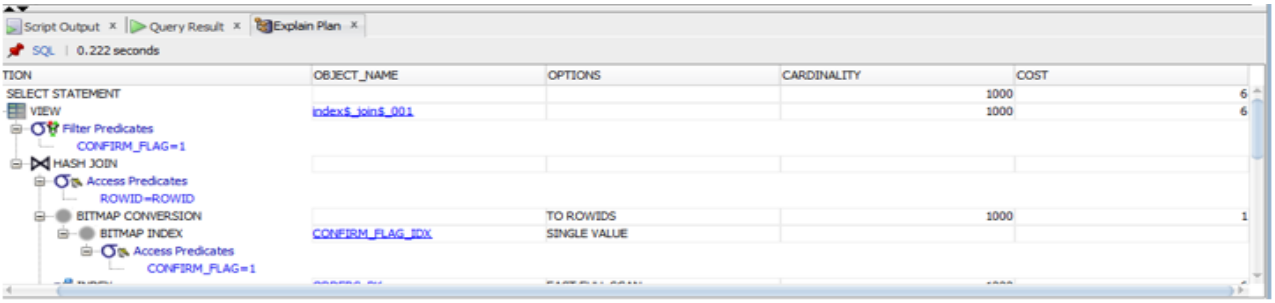
```
SELECT order_id
FROM  orders
WHERE  confirm_flag='1';
```



Creating Index:

CREATE BITMAP INDEX CONFIRM_FLAG_IDX on orders(confirm_flag)

After indexing:



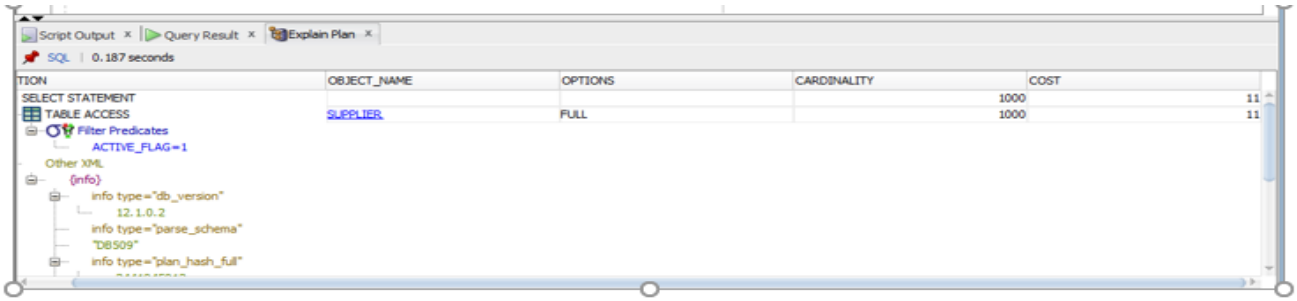
From the above explain plan of the query, it can be clearly seen that index is used in the operation to search order_id, and due to the presence of index the query cost has been decreased from 9 to 6.

Real case scenario: To improve the speed of the data access or to decrease the amount of time to retrieve information by constructing Bit map index. This indexing strategy will be helpful when we need to find active supplier of grocery store search information online based on a text in a string based on active flag. This indexing will be done on the supplier table.

d) When an active supplier_id in the supplier is being searched.

```
SELECT supplier_id
FROM   supplier
WHERE  activeflag = 1;
```

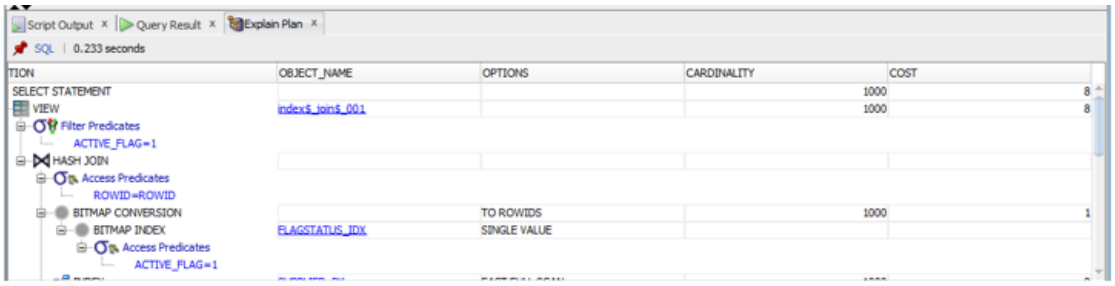
Before indexing:



Creating Index:

CREATE BITMAP INDEX FLAGSTATUS_IDX on supplier(active flag) ;

After indexing:



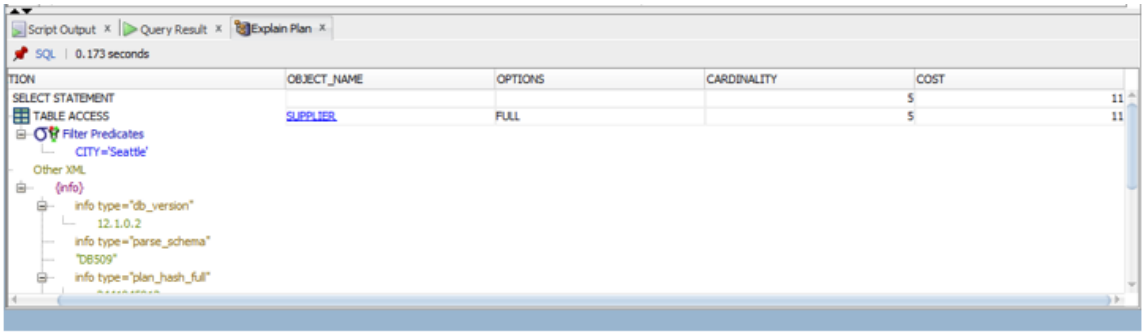
Here in the above case for finding out active suppliers**, there** is change in the cost because of BITMAP indexing, due to the presence of index the query cost has been decreased from 11 to 8.

Real case scenario: To improve the speed of the data access or to decrease the amount of time to retrieve information by constructing Bit map index. This indexing strategy will be helpful when we need to find supplier of grocery store from city. This indexing will be done on the supplier table on city column.

e) When the details of suppliers from a city are searched.

Before indexing:

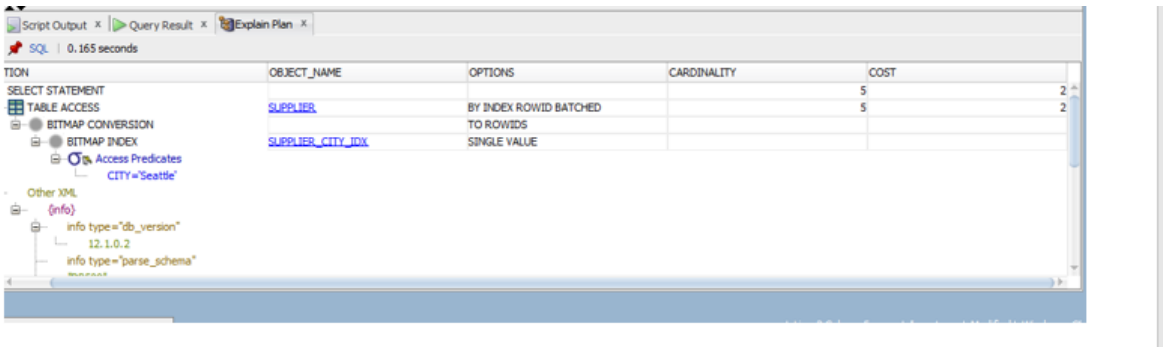
```
SELECT *
FROM   supplier
WHERE  city = 'seattle';
```



Creating Index:

CREATE BITMAP INDEX SUPPLIER_CITY_IDX on supplier(city) ;

After indexing:

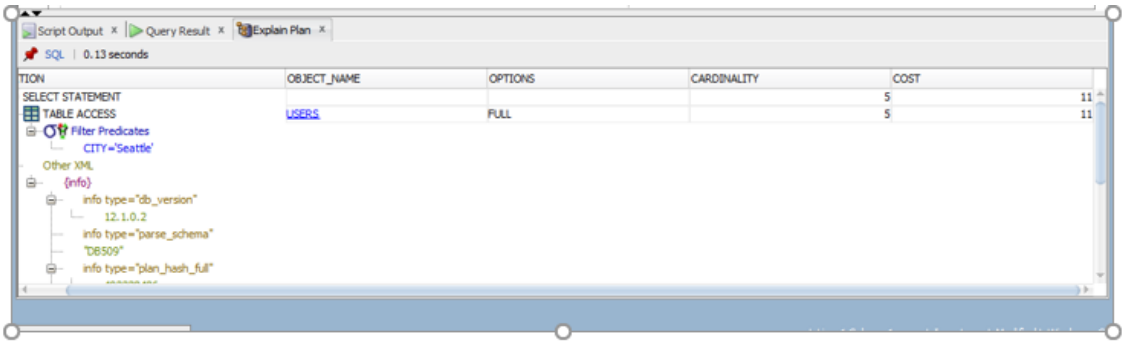


Here in the above case for finding out supplier details from a particular city**, there** is a change in the cost because of BITMAP indexing, due to the presence of index the query cost has been decreased from 11 to 2.

Real case scenario: To improve the speed of the data access or to decrease the amount of time to retrieve information by constructing Bit map index. This indexing strategy will be helpful when we need to find users of grocery store from city. This indexing will be done on the supplier table on city column

f) When the details of users from a city are searched.

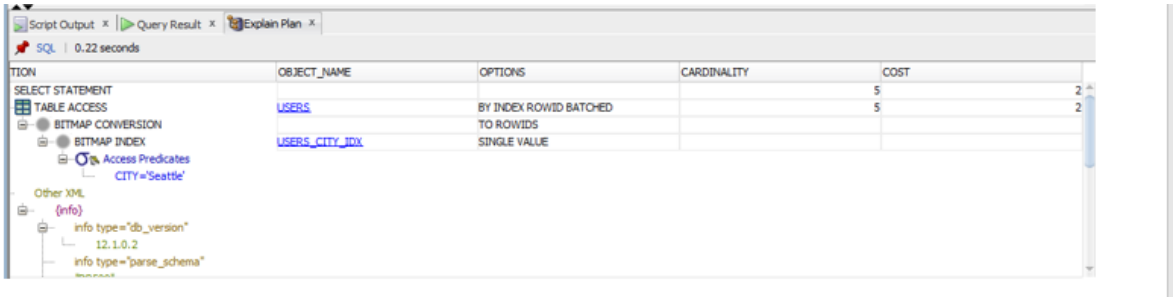
```
SELECT *
FROM   users
WHERE  city = ‘seattle’;
```



Creating Index:

CREATE BITMAP INDEX USERS _CITY_IDX on user(city) ;

After indexing:



Here in the above case for finding out users details from a particular city**, there** is change in the cost because of BITMAP indexing, due to the presence of index the query cost has been decreased from 11 to 2.

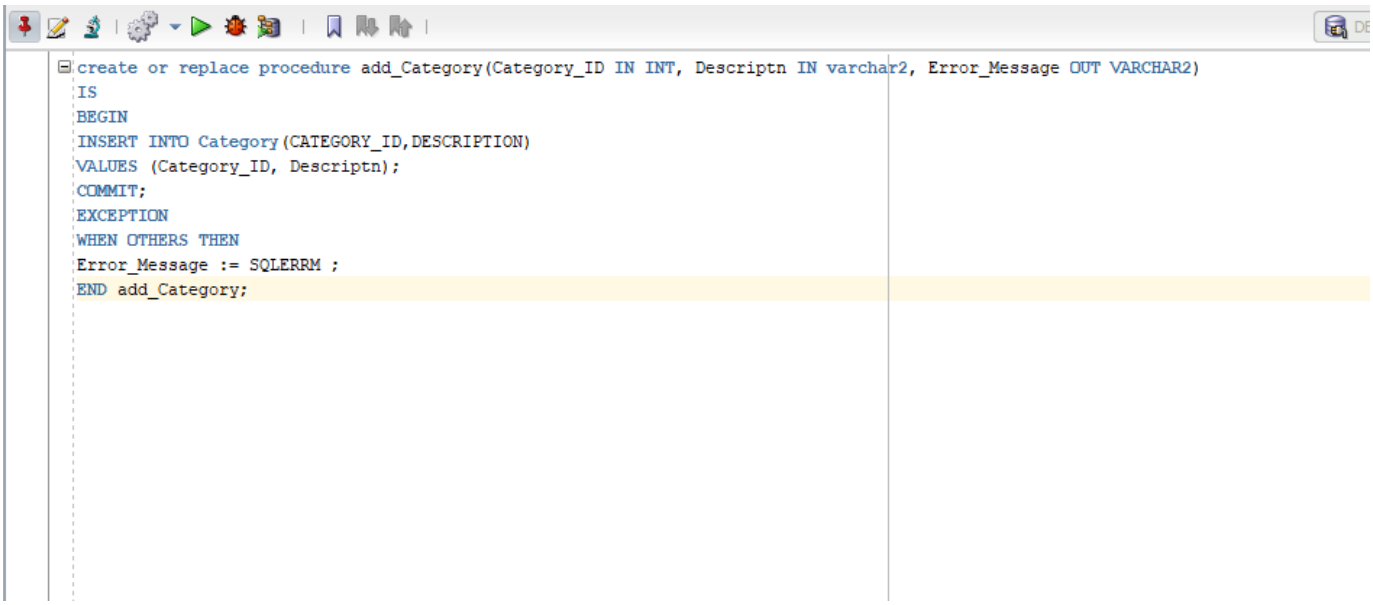
DATABASE PROGRAMMING:

Stored Procedures:

A **stored procedure** is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in the SQL developer. Procedures are compiled only once, and it can be run many times. It does not return any value.
Function performs an action or a complex calculation and returns the output. A function is compiled and executed every time when it is called.

Procedures:

1. In the below **add_category** procedure, we have created a set of statements to insert the values into category table. With the help of this procedure we can insert the values by calling the procedure name and passing the arguments in it.



```
create or replace procedure add_Category(Category_ID IN INT, Descriptn IN varchar2, Error_Message OUT VARCHAR2)
IS
BEGIN
INSERT INTO Category(CATEGORY_ID,DESCRIPTION)
VALUES (Category_ID, Descriptn);
COMMIT;
EXCEPTION
WHEN OTHERS THEN
Error_Message := SQLERRM ;
END add_Category;
```

2. In the below **Raise_Price** procedure, we have created a set of statements to increase the amount of price for the product price column in the Products table. With the help of this procedure we can increase the values by calling the procedure name and passing the appropriate arguments in it.

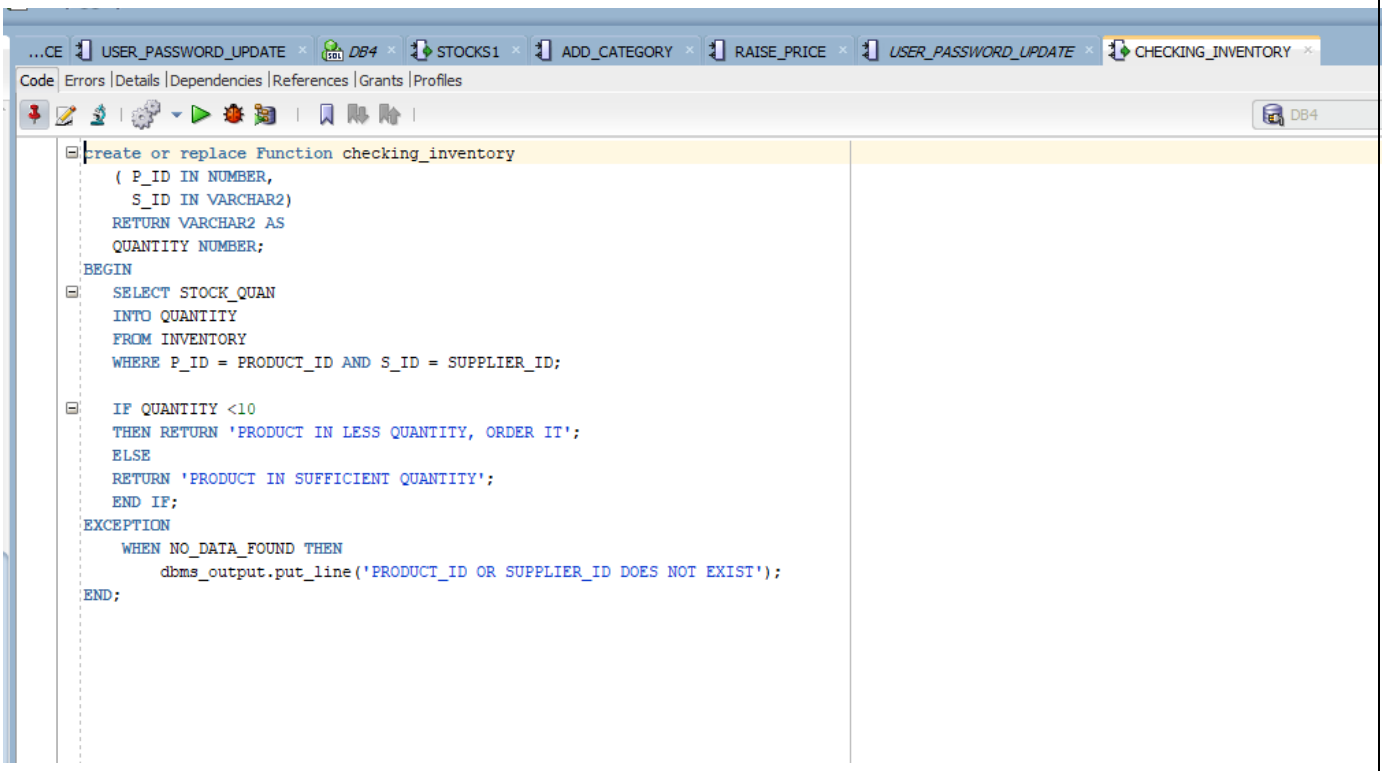
```
create or replace procedure Raise_Price(P_ID IN NUMBER ,AMT IN NUMBER,Error_Message OUT VARCHAR2)
IS
BEGIN
UPDATE PRODUCTS SET PROD_PRICE= PROD_PRICE + AMT
          WHERE PRODUCT_ID=P_ID;
COMMIT;
EXCEPTION
WHEN OTHERS THEN
Error_Message := SQLERRM ;
END Raise_Price;
```

3. In the below **User_Password_Update** procedure, we have created a set of statements to update the password in the Users table. With the help of this procedure we can Update the password by calling the procedure name and passing the appropriate arguments in it.

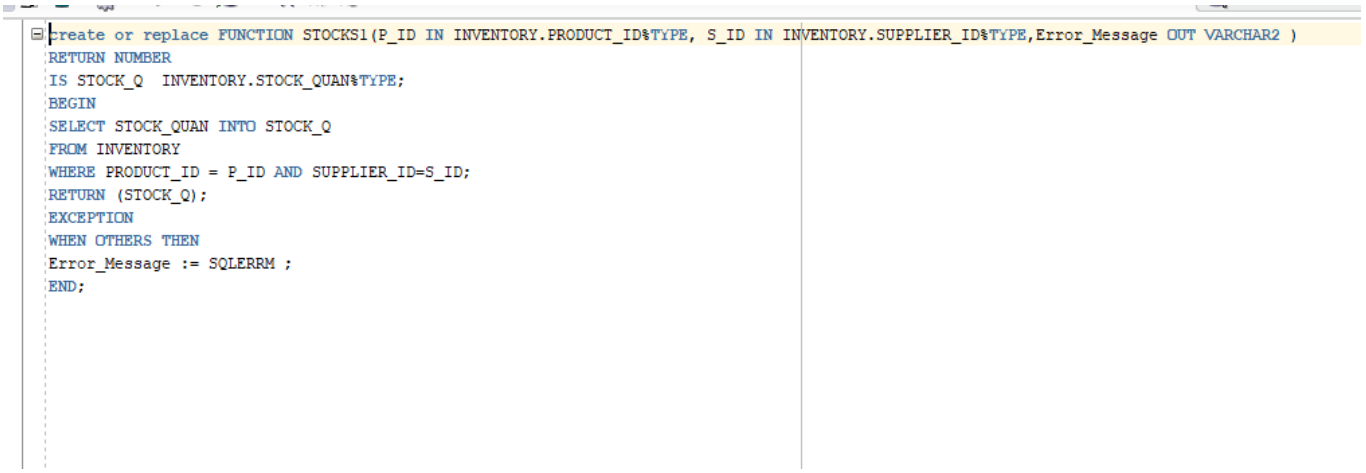
```
create or replace PROCEDURE User_Password_Update (
    U_ID       VARCHAR2,
    U_password      VARCHAR2,
    U_Email          VARCHAR2,
    U_Message        OUT VARCHAR2
)  AS
BEGIN
    UPDATE USERS
        SET
            PASSWORD = U_password
    WHERE
            EMAIL_ID = U_Email
        AND
            USER_ID = U_ID;

    IF
        ( SQL%rowcount >= 1 )
    THEN
        U_Message := 'Successfully Updated';
    ELSE
        U_Message := 'Enter Valid USER_ID and EMAIL_ID';
    END IF;

END User_Password_Update;
```

Functions:

1. The **checking_inventory** function is used for checking the stocks in the inventory and it gives out a message whether to refill the stock for a product or if the product is in sufficient quantity.



2. The **STOCKS1** function produces the quantity of stock, when the appropriate Supplier_Id and Product_ID is given.



SQL PERFORMANCE TUNING

SQL statements are used to fetch the data from database. We can write the queries in many ways to fetch the data but writing in a best way to is important when we consider the performance.

1) Retrieval becomes faster when we use the actual names of columns of the tables instead id '*'

```
SELECT user_id,
      fname,
      lname,
      email_id
FROM   users
```

INSTEAD OF

```
SELECT *
FROM   users;
```

2) Usage of HAVING clause in the filter condition

```
SELECT   order_id,
        Count(order_id)
FROM     order_items
GROUP BY order_id
HAVING   order_id > 1703;
```

INSTEAD OF

```
SELECT   order_id,
        Count(order_id)
FROM     order_items
WHERE    order_id > 4
GROUP BY order_id;
```

3) Minimize the usage of sub queries as much as possible

```
SELECT product_id,
       item_id
FROM   order_items
WHERE  ( quantity, price ) = (SELECT Max(quantity),
                                    Max(price)
                             FROM   order_items);
```

Instead of

```
SELECT product_id,
       item_id
FROM   order_items
WHERE  quantity = (SELECT Max(quantity)
                   FROM   order_items)
       AND price = (SELECT Max(price)
                    FROM   order_items)
```

4) Usage of EXISTS operator instead of IN when most of the filter criteria is in the main query

```
SELECT *
FROM   orders fr
WHERE  EXISTS (SELECT *
               FROM   users m
               WHERE  m.user_id = fr.user_id)
```

INSTEAD of

```
SELECT *
FROM   orders dr
WHERE  user_id IN (SELECT user_id
                   FROM   users);
```

5) Use UNION ALL in place of UNION

```
SELECT user_id
FROM   users
UNION ALL
SELECT user_id
FROM   orders
```

INSTEAD of

```
SELECT user_id
FROM   users
UNION
SELECT user_id
FROM   orders
```

SQL QUERYING

1.Top Selling Items by Price

```
SELECT description,
       Sum(order_items.price)    AS SALES_TOTAL,
       Sum(order_items.quantity) AS QUANTITY_TOTAL
FROM   order_items
       INNER JOIN products
               ON order_items.product_id = products.product_id
GROUP  BY products.description
ORDER  BY Sum(order_items.price) DESC;
```

Query Result 2 ×

SQL | Fetched 50 rows in 0.044 seconds

	DESCRIPTION	SALES_TOTAL	QUANTITY_TOTAL
1	Oil - Coconut	832.7	53
2	Gelatine Powder	658.46	38
3	Coffee - Decaffeinato Coffee	600.64	46
4	Tomatoes Tear Drop Yellow	576.73	33
5	Appetizer - Sausage Rolls	569.78	39
6	Mace	567.42	45
7	Muffin - Mix - Creme Brule 151	561.32	40
8	Fish - Base, Bouillion	538.36	49
9	Bread - Triangle White	535.58	41
10	Chicken - Thigh, Bone In	531.25	40

2.Top Selling Items by Quantity

```
SELECT products.description,
       Sum(order_items.price)    AS SALES_TOTAL,
       Sum(order_items.quantity) AS QUANTITY_TOTAL
FROM   order_items
       INNER JOIN products
               ON order_items.product_id = products.product_id
GROUP  BY products.description
ORDER  BY Sum(order_items.quantity) DESC;
```

Query Result

SQL | Fetched 50 rows in 0.048 seconds

	DESCRIPTION	SALES_TOTAL	QUANTITY_TOTAL
1	Oil - Coconut	832.7	53
2	Fish - Base, Bouillion	538.36	49
3	Bread - Burger	305.42	47
4	Oil - Olive Bertolli	397.15	47
5	Coffee - Decaffeinato Coffee	600.64	46
6	Crackers - Graham	438.4	46
7	Potatoes - Peeled	512.86	45
8	Icecream - Dstk Strw Chseck	426.1	45
9	Mace	567.42	45
10	Veal - Shank, Pieces	324.87	44

3. Top customers BY purchase amount

```
SELECT users.user_id,
       users.fname,
       users.lname,
       ( order_items.quantity * order_items.price ) AS TOTAL_PRICE
FROM   users
       INNER JOIN orders
               ON users.user_id = orders.user_id
       INNER JOIN order_items
               ON order_items.order_id = orders.order_id
GROUP  BY users.user_id,
          users.fname,
          users.lname,
          order_items.quantity,
          order_items.price
ORDER  BY Sum(total_price) DESC;
```

Query Result | Query Result 1

SQL | Fetched 50 rows in 0.062 seconds

	USER_ID	FNAME	LNAME	TOTAL_PRICE
1	USER1207	Colleen	Yellop	1617.57
2	USER1671	Jarvis	Hargreaves	1616.76
3	USER59	Lorianne	Bondar	1614.33
4	USER1275	Patin	Shilstone	1610.28
5	USER1102	Sinclare	Kelsell	1609.47
6	USER1397	Izak	Ortner	1609.47
7	USER1992	Derick	De Vere	1607.85
8	USER1645	Yetty	Giovanizio	1603.8
9	USER59	Lorianne	Bondar	1602.18
10	USER1186	Isa	Barley	1599.75

4. **Top Customers by Number of Transactions**

```
SELECT     users.user_id,
           users.fname,
           users.lname,
           Count(orders.order_id) AS TOTAL_TRANSACTIONS
FROM       orders
INNER JOIN users
ON         users.user_id = orders.user_id
GROUP BY   users.user_id,
           users.fname,
           users.lname
ORDER BY   Count(orders.order_id) DESC;
```

Query Result | Query Result 1 | Query Result 2

SQL | Fetched 50 rows in 0.072 seconds

	USER_ID	FNAME	LNAME	TOTAL_TRANSACTIONS
1	USER1773	Stormy	Castelot	4
2	USER1890	Sharl	Drissell	3
3	USER164	Tressa	Sandercock	3
4	USER1405	Fidelio	Aizikov	3
5	USER304	Waylan	Dickman	3
6	USER1685	Matias	Ragborne	3
7	USER109	Gilligan	Hulse	2
8	USER157	Carina	Humbee	2
9	USER168	Josselyn	Behnecken	2
10	USER169	Stephi	Colleer	2

5. **Recent Best-Selling Items (Quantity)**

```
SELECT     products.description,
           Sum(order_items.quantity * order_items.price) AS SALES_TOTAL,
           Sum(order_items.quantity)                     AS QUANTITY_TOTAL
FROM       orders
INNER JOIN order_items
ON         orders.order_id = order_items.order_id
INNER JOIN products
ON         order_items.product_id = products.product_id
WHERE      orders.order_date BETWEEN Add_months(Trunc(sysdate,'mm'),-
1) AND     Last_day(Add_months(Trunc(sysdate,'mm'),-1))
GROUP BY   products.description
ORDER BY   Sum(order_items.quantity) DESC
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | Fetched 50 rows in 0.048 seconds

	DESCRIPTION	SALES_TOTAL	QUANTITY_TOTAL
1	Wiberg Super Cure	2158.72	28
2	Nantuket Peach Orange	1381.99	17
3	Loquat	1354.66	17
4	Coconut - Creamed, Pure	653.95	17
5	Pasta - Lasagne, Fresh	684.17	16
6	Water - Aquafina Vitamin	998.29	15
7	Salmon - Sockeye Raw	1796.22	15
8	Wine - Masi Valpolocell	1256.72	14
9	Wine - Prosecco Valdobiaddene	1333.21	14
10	Table Cloth 62x120 White	1603.77	14

6. **Supplier details as per product and available stock**

```
SELECT     products.description AS PRODUCT_NAME,
           category.description AS CATEGORY_NAME,
           supplier.NAME        AS SUPPLIER_NAME,
           inventory.stock_quan AS STOCK_COUNT,
           supplier.contact,
           supplier.email
FROM       products
INNER JOIN category
ON         category.category_id=products.category_id
INNER JOIN inventory
ON         inventory.product_id=products.product_id
INNER JOIN supplier
ON         supplier.supplier_id=inventory.supplier_id
ORDER BY   inventory.stock_quan
```

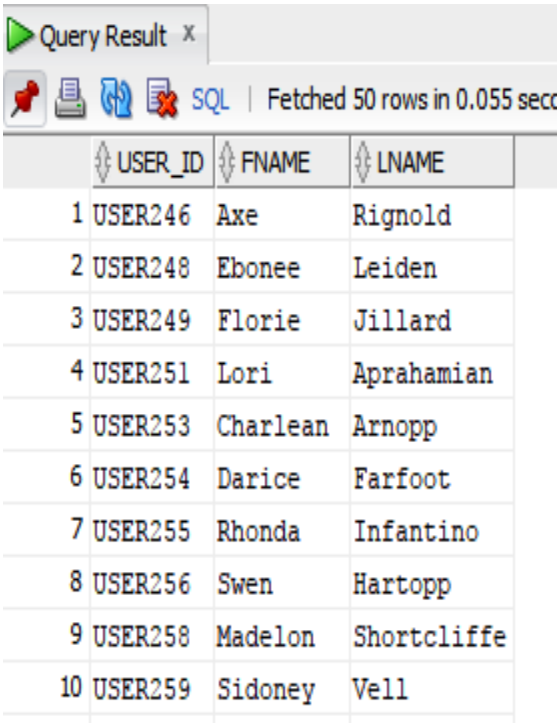
Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6

SQL | Fetched 50 rows in 0.047 seconds

	PRODUCT_NAME	CATEGORY_NAME	SUPPLIER_NAME	STOCK_COUNT	CONTACT	EMAIL
1	Tea - Earl Grey	NON-FRUIT DRINKS - SS	Henrie Naris	1	+1 501 905 5591	hnaris2u@cargocollective.com
2	Ezy Change Mophandle	SPORTS DRINKS	Ryann Alejandre	1	+1 716 894 6209	ralejandrei9@youtube.com
3	Sesame Seed	SMOKELESS TOBACCO	Lucho Caseborne	1	+1 404 198 6607	lcaseborneg8@netscape.com
4	Napkin - Beverage 1 Ply	WATER FILTERS/DEVICES	Hildegaard Fowlston	1	+1 504 302 0443	hfowlstonos@chron.com
5	Pepper - Cayenne	TOOTHPASTE	Basia Edmott	1	+1 757 812 0895	bedmott63@gnu.org
6	Basil - Thai	BEER/ALE/ALCOHOLIC CIDER	Lanette Spellman	1	+1 209 296 0120	lspellmanml@hud.gov
7	Cheese - Cottage Cheese	COLD/ALLERGY/SINUS LIQUIDS	Gil Sancto	1	+1 414 356 4448	gsanctoal@globo.com
8	Liners - Baking Cups	OTHER GROOMING SUPPLIES	Nate Larmet	1	+1 702 557 5035	nlarmet41@blogs.com
9	Cup - 3.5oz, Foam	POOL/SPA CHEMICALS	Kandy Meace	1	+1 916 748 9141	kmeacet@ucoz.com
10	Carrots - Jumbo	BAKED BEANS/CANNED BREAD	Marcelline Awdry	1	+1 202 232 0764	mawdryrl@infoseek.co.jp

7. **Customers with order not yet confirmed**

```
SELECT     users.user_id,
           users.fname,
           users.lname
FROM       orders
INNER JOIN users
ON         orders.user_id=users.user_id
WHERE      orders.confirm_flag=2
```



	USER_ID	FNAME	LNAME
1	USER246	Axe	Rignold
2	USER248	Ebonee	Leiden
3	USER249	Florie	Jillard
4	USER251	Lori	Aprahamian
5	USER253	Charlean	Arnopp
6	USER254	Darice	Farfoot
7	USER255	Rhonda	Infantino
8	USER256	Swen	Hartopp
9	USER258	Madelon	Shortcliffe
10	USER259	Sidoney	Vell

8. Users with pending shipping order

```
SELECT users.user_id,
       users.fname,
       users.lname,
       orders.ship_addr1,
       orders.ship_city,
       orders.ship_state,
       orders.ship_zipcode,
       Count(*) AS PENDING
FROM   orders
       INNER JOIN users
               ON users.user_id = orders.user_id
       INNER JOIN payment
               ON orders.user_id = payment.user_id
WHERE  payment.payment_flag = 2
GROUP  BY users.user_id,
          users.fname,
          users.lname,
          orders.ship_addr1,
          orders.ship_city,
          orders.ship_state,
          orders.ship_zipcode
ORDER  BY pending DESC
```

Query Result

SQL | Fetched 50 rows in 0.152 seconds

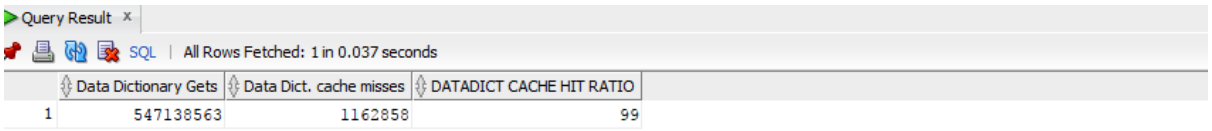
	USER_ID	FNAME	LNAME	SHIP_ADDR1	SHIP_CITY	SHIP_STATE	SHIP_ZIPCODE	PENDING
1	USER290	Roselin	Millbank	4468 Esch Road	Los Angeles	California	90060	3
2	USER1849	Flint	Burd	41 Almo Road	Honolulu	Hawaii	96835	3
3	USER94	Ethelin	Devonshire	235 Straubel Junction	Fort Worth	Texas	76110	3
4	USER358	Amalia	Pighills	9 Lien Plaza	Allentown	Pennsylvania	18105	3
5	USER1158	Lorant	Frickey	3395 Nancy Way	Mobile	Alabama	36616	3
6	USER1539	Bonnee	Phonix	132 Maple Wood Way	South Bend	Indiana	46634	3
7	USER1875	Gustie	Philippe	184 Summerview Trail	Washington	District of Columbia	20022	3
8	USER1918	Margie	Coyett	56 Eliot Hill	Beaumont	Texas	77705	3
9	USER250	Vince	Olesen	2 Crowley Trail	Buffalo	New York	14263	3
10	USER989	Prudy	Pherps	66 Di Loreto Point	Bakersfield	California	93381	3

DATABASE SCRIPTS

Database scripts mainly executed by the DBA'S (Database administrators). DBA's monitor the performance of database with the help of scripts.

1) Script to obtain information about the data dictionary cache hit ratio, Data dictionary cache Gets and Data Dictionary Cache Misses.

```
SELECT SUM(gets)
    "Data Dictionary Gets",
    SUM(getmisses)
    "Data Dict. cache misses",
    Trunc(( 1 - ( SUM(getmisses) / SUM(gets) ) ) * 100)
    "DATADICT CACHE HIT RATIO"
FROM   v$rowcache;
```



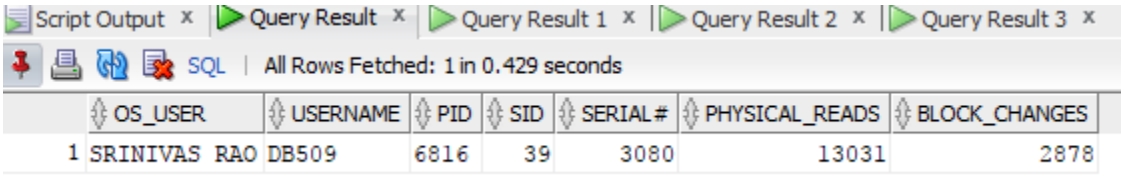
2) Database script use to get information regarding all the files currently present in the existing tablespace and data regarding the current file size used free space present in the file size.

```
SELECT a.tablespace_name,
       a.bytes / 1024 / 1024   Mbytes_used,
       b.bytes / 1024 / 1024   Mbytes_free,
       Round(( ( a.bytes - b.bytes ) / a.bytes ) * 100, 2) percent_used
FROM   (SELECT tablespace_name,
               SUM(bytes) BYTES
        FROM   dba_data_files
        GROUP  BY tablespace_name) a
       left outer join (SELECT tablespace_name,
                               SUM(bytes) BYTES,
                               Max(bytes) largest
                        FROM   dba_free_space
                        GROUP  BY tablespace_name) b
                    ON a.tablespace_name = b.tablespace_name
WHERE  1 = 1
       AND a.tablespace_name LIKE '%'
ORDER  BY ( ( a.bytes - b.bytes ) / a.bytes ) DESC;
```

	TABLESPACE_NAME	MBYTES_USED	MBYTES_FREE	PERCENT_USED
1	SYSTEM	930	8.625	99.07
2	EXAMPLE	1260.625	41.625	96.7
3	USERS	1889	94.5	95
4	SYSAUX	2130	141.0625	93.38
5	STUDENTS	10240	3474.125	66.07
6	COLORS	2579	2315.25	10.23
7	UNDOTBS1	2445	2405.5625	1.61

3) DBA script displays the information regarding current session Active database users in the descending order of physical reads.

```
SELECT osuser  os_user,
       username,
       process pid,
       ses.sid sid,
       serial#,
       physical_reads,
       block_changes
FROM   v$session ses,
       v$sess_io sio
WHERE  ses.sid = sio.sid
       AND username IS NOT NULL
       AND status = 'ACTIVE'
ORDER  BY physical_reads;
```

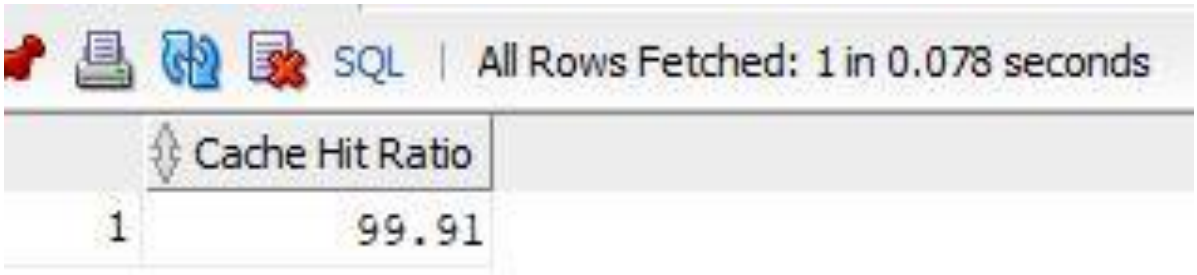


	OS_USER	USERNAME	PID	SID	SERIAL#	PHYSICAL_READS	BLOCK_CHANGES
1	SRINIVAS RAO	DB509	6816	39	3080	13031	2878

Only active user is SRINIVAS RAO (i.e. Myself) with physical reads of 13031 and Block changes 2878.

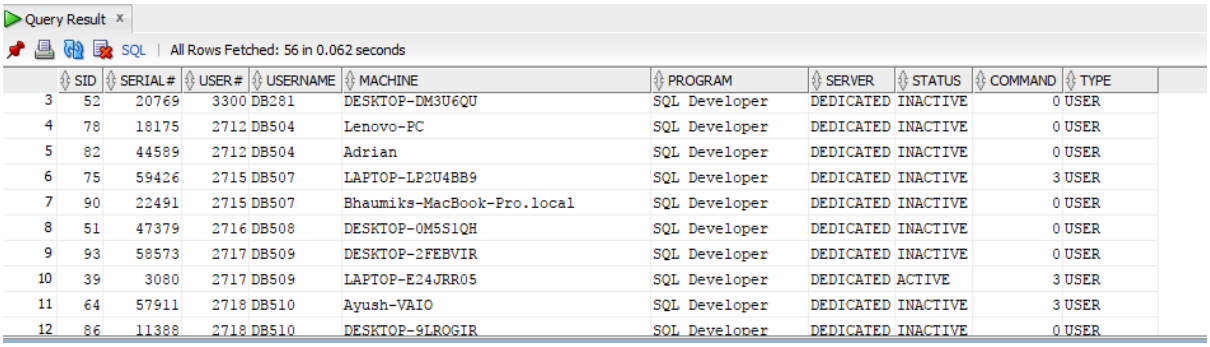
4) The following query will help to find out the Database Buffer cache hit ratio. If the cache hit ratio is very high, then the database is highly likely to store the most recently accessed data. It also in turn depends on the amount of data queried by the previous queries.

```
SELECT Round(( 1 - ( phy.value / ( cur.value + con.value ) ) ) * 100, 2)
    "Cache Hit Ratio"
FROM   v$sysstat cur,
    v$sysstat con,
    v$sysstat phy
WHERE  cur.name = 'db block gets'
    AND con.name = 'consistent gets'
    AND phy.name = 'physical reads';
```



5) Script to check the list of active and inactive users for a database. This script will be highly useful in real case scenarios to check how users are active in the existing application of a product.

```
SELECT sid,
       serial#,
       user#,
       username,
       machine,
       program,
       server,
       status,
       command,
       TYPE
FROM   v$session
ORDER  BY username;
```



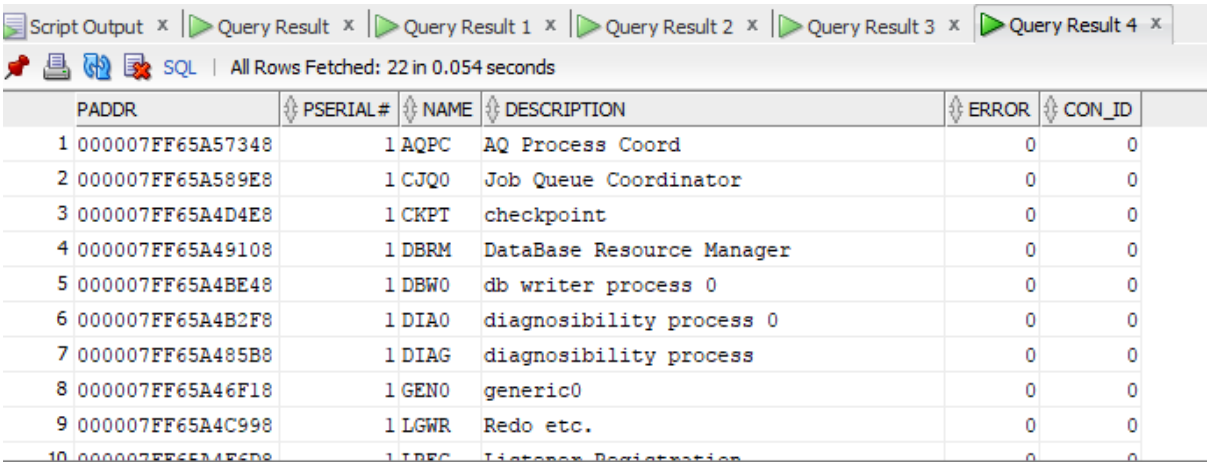
Query Result

SQL | All Rows Fetched: 56 in 0.062 seconds

	SID	SERIAL#	USER#	USERNAME	MACHINE	PROGRAM	SERVER	STATUS	COMMAND	TYPE
3	52	20769	3300	DB281	DESKTOP-DM3U6QU	SQL Developer	DEDICATED	INACTIVE	0	USER
4	78	18175	2712	DB504	Lenovo-PC	SQL Developer	DEDICATED	INACTIVE	0	USER
5	82	44589	2712	DB504	Adrian	SQL Developer	DEDICATED	INACTIVE	0	USER
6	75	59426	2715	DB507	LAPTOP-LP2U4BB9	SQL Developer	DEDICATED	INACTIVE	3	USER
7	90	22491	2715	DB507	Bhaumiks-MacBook-Pro.local	SQL Developer	DEDICATED	INACTIVE	0	USER
8	51	47379	2716	DB508	DESKTOP-0M5S1QH	SQL Developer	DEDICATED	INACTIVE	0	USER
9	93	58573	2717	DB509	DESKTOP-2FEBVIR	SQL Developer	DEDICATED	INACTIVE	0	USER
10	39	3080	2717	DB509	LAPTOP-E24JRR05	SQL Developer	DEDICATED	ACTIVE	3	USER
11	64	57911	2718	DB510	Ayush-VAIO	SQL Developer	DEDICATED	INACTIVE	3	USER
12	86	11388	2718	DB510	DESKTOP-9LROGIR	SQL Developer	DEDICATED	INACTIVE	0	USER

6) DBA Script to provide information regarding the background processes currently running and description of the process.

```
SELECT *
FROM   v$bgprocess
WHERE  paddr <> '00'
ORDER  BY name;
```



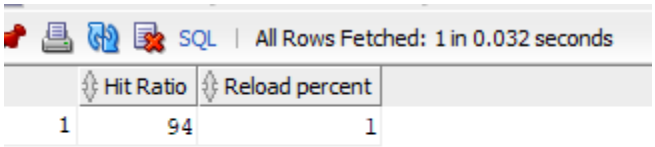
Script Output x | Query Result x | Query Result 1 x | Query Result 2 x | Query Result 3 x | Query Result 4 x

SQL | All Rows Fetched: 22 in 0.054 seconds

	PADDR	PSERIAL#	NAME	DESCRIPTION	ERROR	CON_ID
1	000007FF65A57348	1	AQPC	AQ Process Coord	0	0
2	000007FF65A589E8	1	CJQ0	Job Queue Coordinator	0	0
3	000007FF65A4D4E8	1	CKPT	checkpoint	0	0
4	000007FF65A49108	1	DBRM	DataBase Resource Manager	0	0
5	000007FF65A4BE48	1	DBW0	db writer process 0	0	0
6	000007FF65A4B2F8	1	DIA0	diagnosibility process 0	0	0
7	000007FF65A485B8	1	DIAG	diagnosibility process	0	0
8	000007FF65A46F18	1	GEN0	generic0	0	0
9	000007FF65A4C998	1	LGWR	Redo etc.	0	0
10	000007FF65A4E6D8	1	LREG	Listener Registration	0	0

7) DBA Script to find out information about the library cache hit ratio.

```
SELECT Round(SUM(pinhits) / SUM(pins), 2) * 100 "Hit Ratio",
       Round(SUM(reloads) / SUM(pins), 2) * 100 "Reload percent"
FROM   v$librarycache
WHERE  namespace IN ( 'SQL AREA', 'TABLE/PROCEDURE', 'BODY', 'TRIGGE
R' );
```



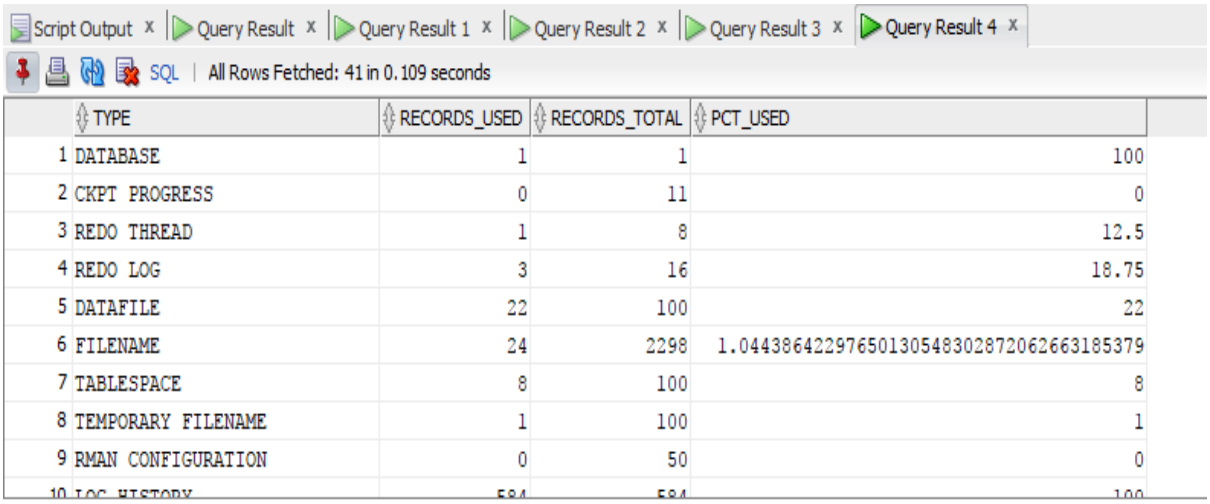
8) DBA Script to find out all blocking sessions and which users are blocking them. This will help DBA in resolving the blocking issues and improve the performance of database.

```
SELECT sid,
       blocking_session,
       username,
       sql_id,
       event,
       machine,
       osuser,
       program,
       last_call_et
FROM   v$session
WHERE  blocking_session > 0;
```

The above query result is empty because none of the sessions are blocked in the current database.

9) DBA SCRIPT to list all the files that are managed by the control file and physical structure of the database.

```
set pages 50000
col pct_used format 990.09
SELECT TYPE,
      records_used,
      records_total,
      records_used / records_total * 100 "PCT_USED"
FROM   sys.v_$controlfile_record_section;
```



Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4

SQL | All Rows Fetched: 41 in 0.109 seconds

	TYPE	RECORDS_USED	RECORDS_TOTAL	PCT_USED
1	DATABASE	1	1	100
2	CKPT PROGRESS	0	11	0
3	REDO THREAD	1	8	12.5
4	REDO LOG	3	16	18.75
5	DATAFILE	22	100	22
6	FILENAME	24	2298	1.044386422976501305483028720626631853 79
7	TABLESPACE	8	100	8
8	TEMPORARY FILENAME	1	100	1
9	RMAN CONFIGURATION	0	50	0
10	LOG HISTORY	584	584	100

VISUALIZATION USING PYTHON:

Data visualization allows user to have visual access to huge amounts of data. It helps to understand hidden patterns and important insights of data. We have integrated database to python for visualizing data insights for grocery database.

Code for fetching the Top 10 Selling Categories:

This will help get top category details and data can be used to give recommendations to customer. Offers can be made which will have low selling products with these category products and in turn will increase business.

```
# Check if Oracle instant client is installed - 32 or 64 bit based on sys config
# Include the path of instant client in path environmental variable.
# Ensure python is also same 32 or 64 bit. Install cx_Oracle library
#Then connect to the Oracle database


import cx_Oracle
ip = 'reade.forest.usf.edu'
port = 1521
SID = 'cdb9'
dsn_tns = cx_Oracle.makedsn(ip, port, SID)
d = cx_Oracle.connect('db579','db5pass',dsn_tns)
f = d.cursor()

# query for fetching the categories and their count
e =f.execute("SELECT c.DESCRIPTION, count(*) FROM "
             "order_items o inner join products p  ON o.product_id = p.product_id "
             "inner join category c  ON p.category_id = c.category_id "
             "group by c.DESCRIPTION")
ROWS = e.fetchall()
```

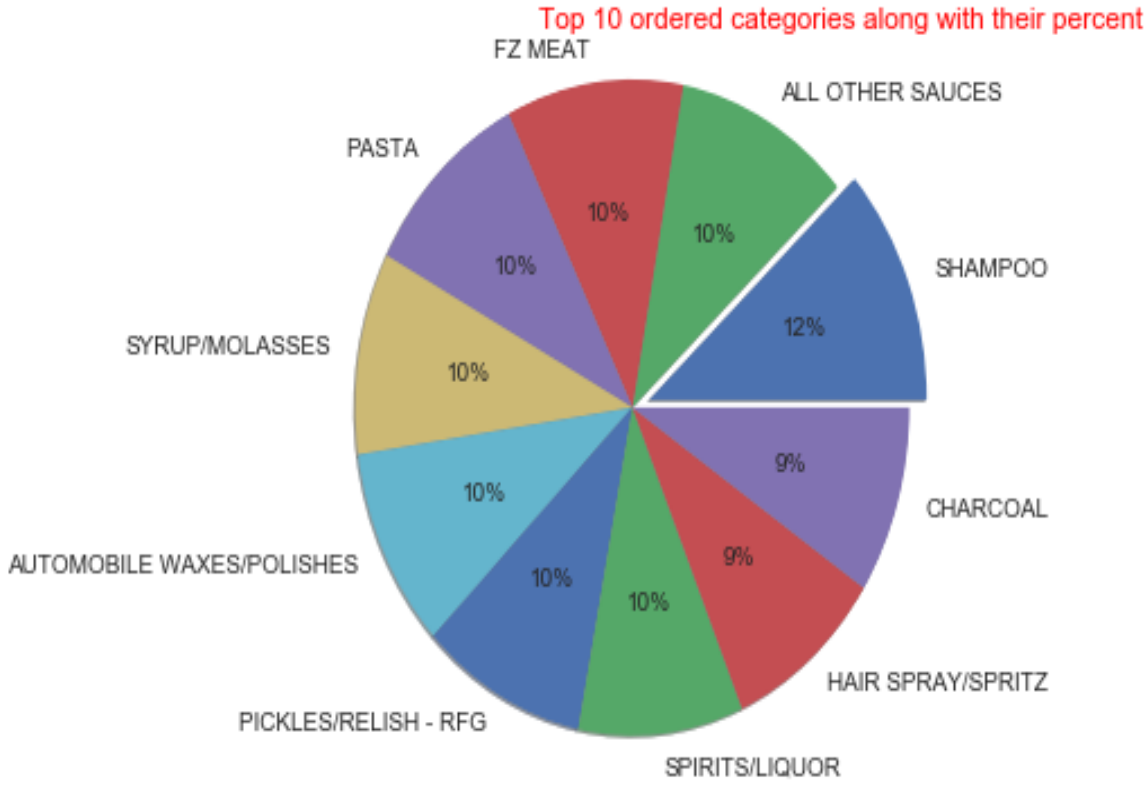
```
## Visualizing the top 5 most selling categories
```

```
category_dict = {}
for i, row in enumerate(ROWS):
    category_dict[i] = list(row)
cat_df = pd.DataFrame(category_dict)
cat_df = cat_df.transpose()
cat_df.columns = ['category_name', 'count_of_orders']
cat_df.sort_values(by ='count', ascending = False, inplace = True)
cat_df.head(10)
```

	category_name	count
193	SHAMPOO	33
200	ALL OTHER SAUCES	29
27	FZ MEAT	29
52	PASTA	28
59	SYRUP/MOLASSES	28
11	AUTOMOBILE WAXES/POLISHES	28
136	PICKLES/RELISH - RFG	27
38	SPIRITS/LIQUOR	27
248	HAIR SPRAY/SPRITZ	26
277	CHARCOAL	26

```
category_names =[]
category_percent = []
for cat_name, cat_price in zip(cat_df['category_name'].values[:10], cat_df['count_of_orders'].values[:10]):
    category_names.append(cat_name)
    category_percent.append(round((cat_price*100)/sum(cat_df['count_of_orders'].values[:10]), 2))
category_percent[9] = category_percent[9] +0.02
labels = category_names
fracs = category_percent
explode = (0.2, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
fig = plt.figure(figsize=(8,5))
plt.pie(fracs, explode=explode, labels=labels, autopct='%.0f%%', shadow=True, startangle= 360, radius = 3)

plt.axis('equal')
plt.title("Top 10 ordered categories along with their percent", fontsize = 12, loc ='right', color ='red')
```

```
<matplotlib.text.Text at 0x2150dab90b8>
```



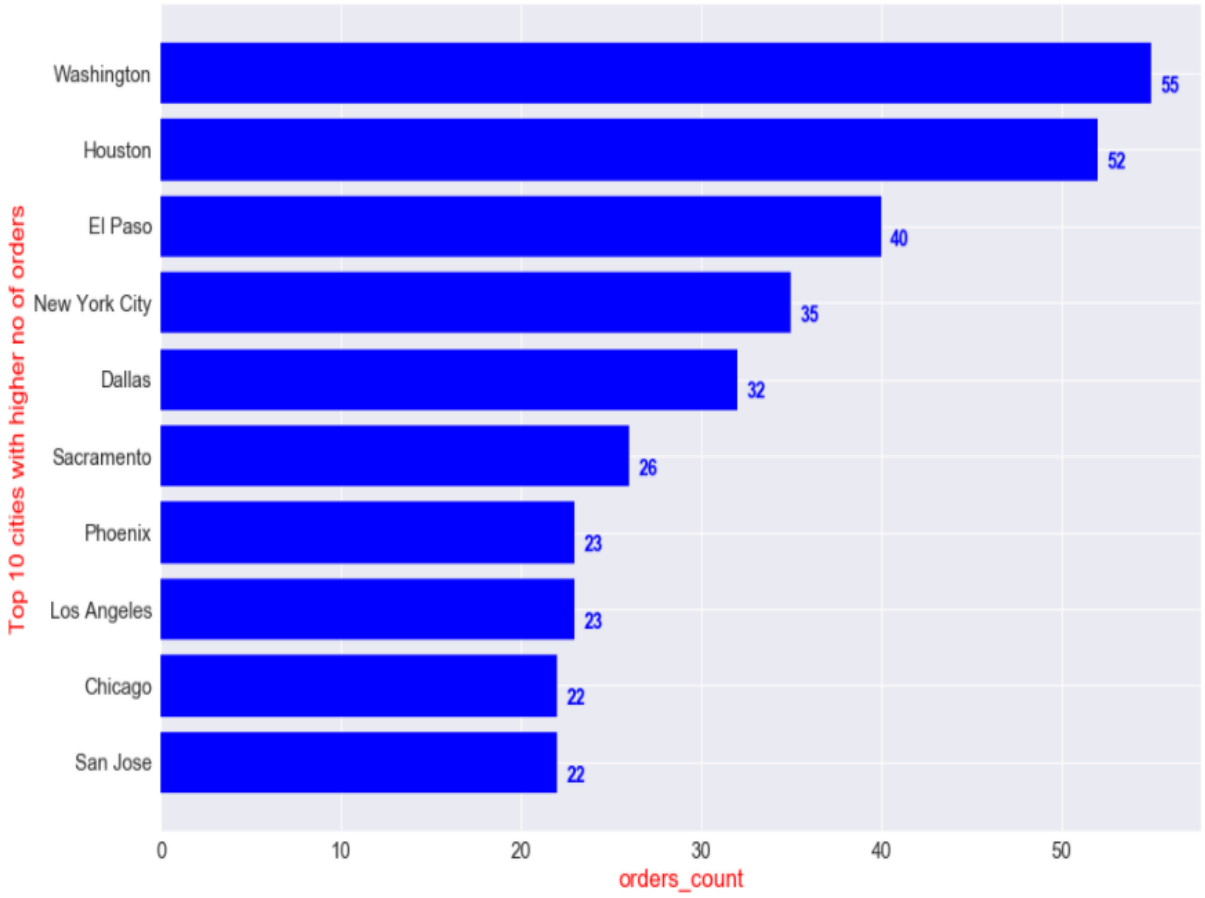
Cities with Highest No of Orders:

This will help management team to track inventory count. Inventory can be supplied based on city location before demand exceeds which can help to manage inventory efficiently.

```
z =f.execute("select ship_city,count(*) from orders "
                "group by ship_city "
            "order by count(*) desc")
all_rows = z.fetchall()
city_dict = {}
for i, row in enumerate(all_rows):
    city_dict[i] = list(row)
city_df = pd.DataFrame(city_dict)
city_df = city_df.transpose()
city_df.columns = ['city_name', 'orders_count']
city_df.sort_values(by ='orders_count', ascending = False, inplace = True)
city_df.head(10)
```

	city_name	orders_count
0	Washington	55
1	Houston	52
2	El Paso	40
3	New York City	35
4	Dallas	32
5	Sacramento	26
6	Phoenix	23
7	Los Angeles	23
8	Chicago	22
9	San Jose	22

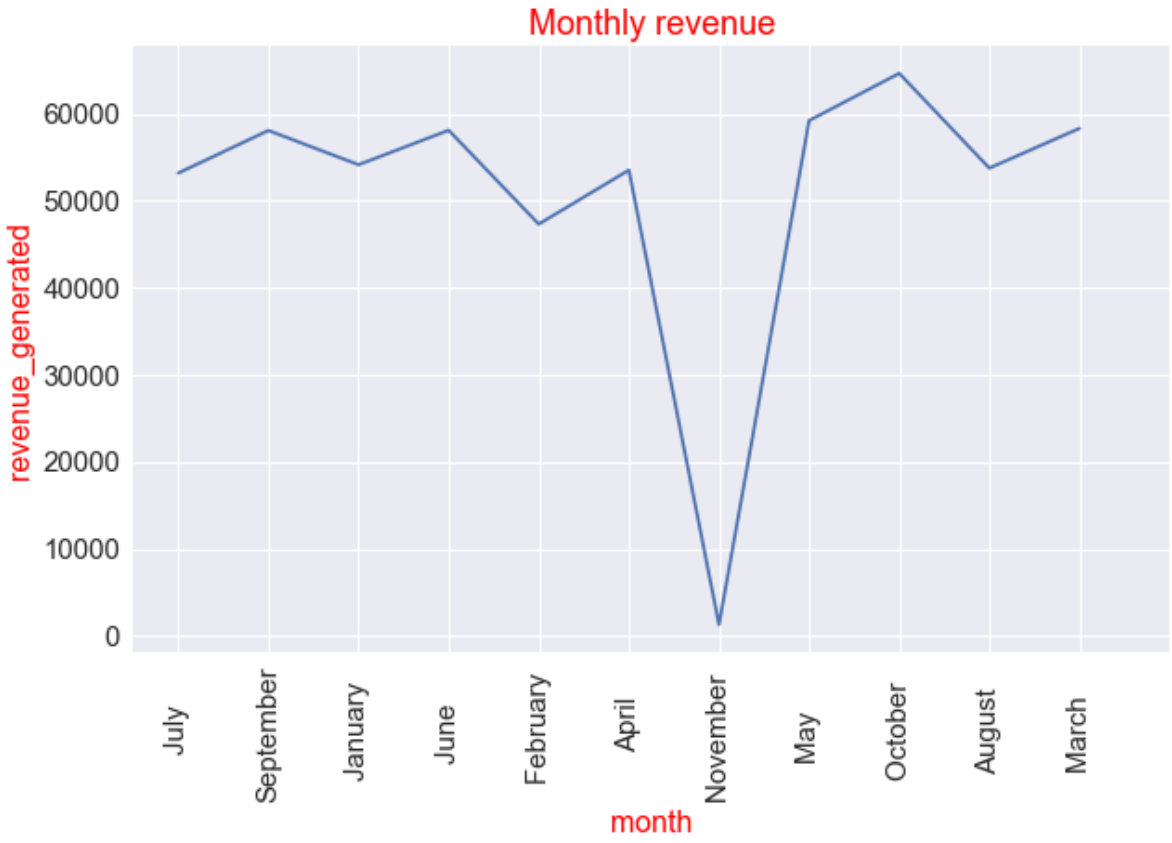
```
plt.figure(figsize=(15,10))
plt.barh(bottom= np.arange(0, 10), width= city_df['orders_count'].values[:10], tick_label = city_df['city_name'].values[:10], co
lor = 'blue')
plt.xlabel('orders_count', fontsize = 'xx-large', color = 'red')
plt.ylabel('Top 10 cities with higher no of orders', fontsize = 'xx-large', color = 'red')
plt.tick_params(labelsize =15.0)
ax = plt.gca()
ax.invert_yaxis()
for i, v in enumerate(city_df['orders_count'].values[:10]):
    ax.text(v + 0.5 , i + .25, str(v), fontsize = 14, color='blue', fontweight='bold')
```



Tracking the revenue of grocery store by month in 2017

By getting this details revenue for current year can be analyze. This revenue detail can be use to predict next year's possible revenue and necessary steps can be taken.

```
e = f.execute("select to_char(to_date(ORDER_DATE, 'DD-MM-YYYY'), 'Month') AS month_ , sum(TOTAL_PRICE) "
              "from (select ORDER_DATE, TOTAL_PRICE "
               "from orders o, payment P "
               "WHERE o.ORDER_ID = P.order_ID and order_date > TO_DATE('31/12/16', 'DD/MM/YY')) "
               "group by to_char(to_date(ORDER_DATE, 'DD-MM-YYYY'), 'Month')")
all_rows = e.fetchall()
revenue_dict = {}
for i, row in enumerate(all_rows):
    revenue_dict[i] = list(row)
revenue_df = pd.DataFrame(revenue_dict)
revenue_df = revenue_df.transpose()
revenue_df.columns = ['month', 'revenue_generated']
revenue_df.head()

plt.figure(figsize=(10, 6))
plt.title('Monthly revenue', color = 'red', size = 18)
plt.plot(revenue_df['revenue_generated'].values)
plt.xticks(np.arange(12), revenue_df.month.values.tolist(), size = 15, rotation = 'vertical')
plt.yticks(size= 15)
plt.xlabel('month', color = 'red', size = 16)
plt.ylabel('revenue_generated', size = 16, color ='red')
```



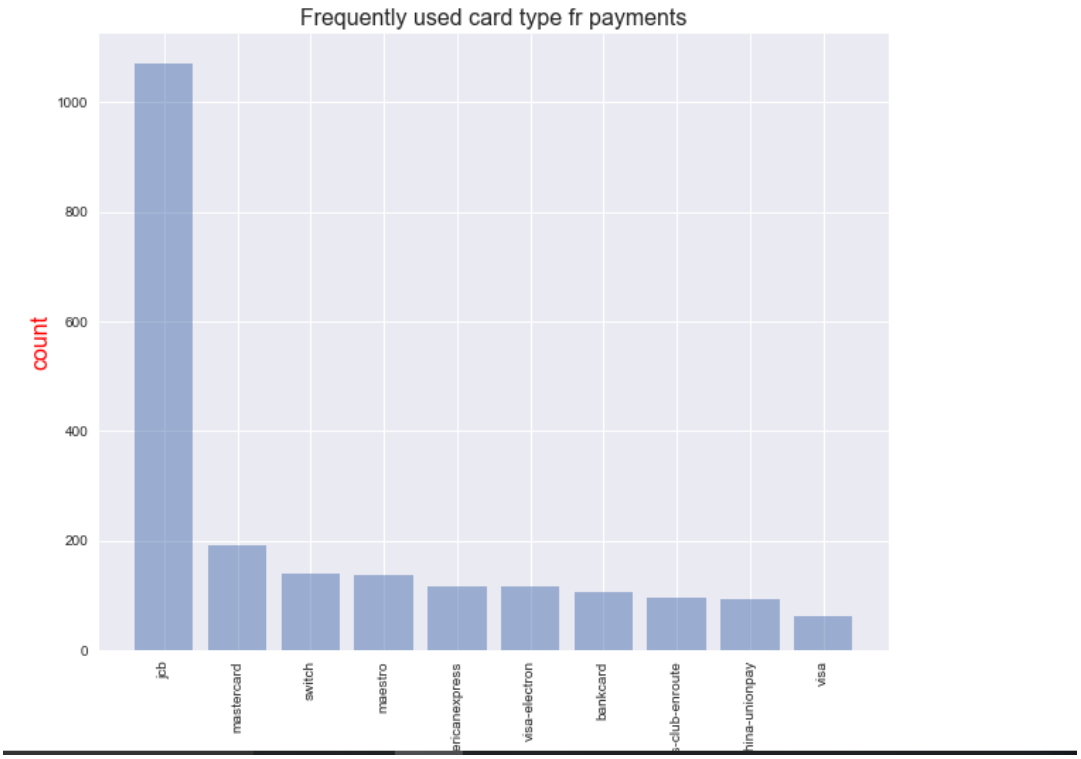
Frequently used card for payments:

This visualization will help retailers to find which card is used by users most of the time so that they can collaborate with card dealers and provide offers for users to retain customer base.

```
z =f.execute('SELECT CARD_TYPE, COUNT(*) FROM PAYMENT GROUP BY CARD_TYPE ORDER BY COUNT(*) DESC')
all_rows = z.fetchall()
card_dict = {}
for i, row in enumerate(all_rows):
    card_dict[i] = list(row)
card_df = pd.DataFrame(card_dict)
card_df = card_df.transpose()
card_df.columns = ['card_type', 'count']
card_df.sort_values(by ='count', ascending = False, inplace = True)
card_df.head(10)
```

	card_type	count
0	jcb	1071
1	mastercard	192
2	switch	140
3	maestro	137
4	americanexpress	117
5	visa-electron	116
6	bankcard	106
7	diners-club-enroute	95
8	china-unionpay	93
9	visa	62

```
objects = card_df['card_type'].values.tolist()[:10]
y_pos = np.arange(0, 10)
performance = card_df['count'].values.tolist()[:10]
plt.figure(figsize =(10, 8))
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects, rotation = 'vertical')
plt.ylabel('count', color = 'red', size = 16)
plt.title('Frequently used card type fr payments', size = 16)

plt.show()
```



EVALUATION TABLE

Topic / Section	Description	Evaluation
Logical database design	The logical design section should include entity-relationship diagrams (ERDs) and data dictionaries for your database design, as well as any design assumptions. You might include a few high-level diagrams that highlight interesting sections of your project (include textual descriptions with these). There should also be a complete ERD for your entire project. There is no expectation that you implement all of your design, just indicate the areas built. You are expected to add additional design work as part of the project.	15

Physical database design	This section should cover implementation-level issues. For instance, you should discuss predicted usage and indexing strategies that support expected activities. In addition, you may wish to discuss architecture issues, including distributed database issues (even though you may not implement anything in these areas). Artifacts could include capacity planning, storage subsystems, and data placement (e.g., tablespace / file system arrangements), indexing strategies, transaction usage maps, etc.	15
Data generation and loading	Though some data was provided, there may have been interesting queries, stored procedures, desktop tools (e.g., MS Excel) that were used to populate the database. You may have used queries with mod function, data arithmetic, number sequences, lookup tables, and even data from the Web. Any / these are interesting additions to the project.	7
Performance tuning	In this section, highlight any experiments run as part of the project related to performance tuning. Experiments with different indexing strategies, optimizer changes, transaction isolation levels, function-based indexes, and table partitioning can all be interesting. Remember to look at different types of queries (e.g., point, range, and scan), execution plans, and I/O burden.	16
Querying	You may also choose to focus on writing SQL queries (analytic SQL extensions can also be explored). Include interesting queries that highlight the types of questions that can be answered by the database. These queries may also be used to illustrate performance tuning.	15
DBA scripts		10

	Throughout the semester, we looked at example DBA scripts	
Database programming	For this section, highlight any stored procedures, functions, or triggers that were created that are not included in the data generation and loading topic.	10
Data visualization	Though interface issues are not typically the focus of the project, you are free to add emphasis here. You can do everything from sketches and mock-ups, to using HTML and other web-enabled tools to build an interface. You can also experiment with creating visualizations for your data using a variety of freely-available tools such as Tableau Public.	12