ISM6218.001F16

ADVANCED DATABSE MANAGEMENT SYSTEMS

FINAL GROUP PROJECT

"E-COMMERCE SYSTEMS"

Group Members

Himank Vats (U87964346)

Jijo Johny (U12605351)

Tess Thomas (U66417789)

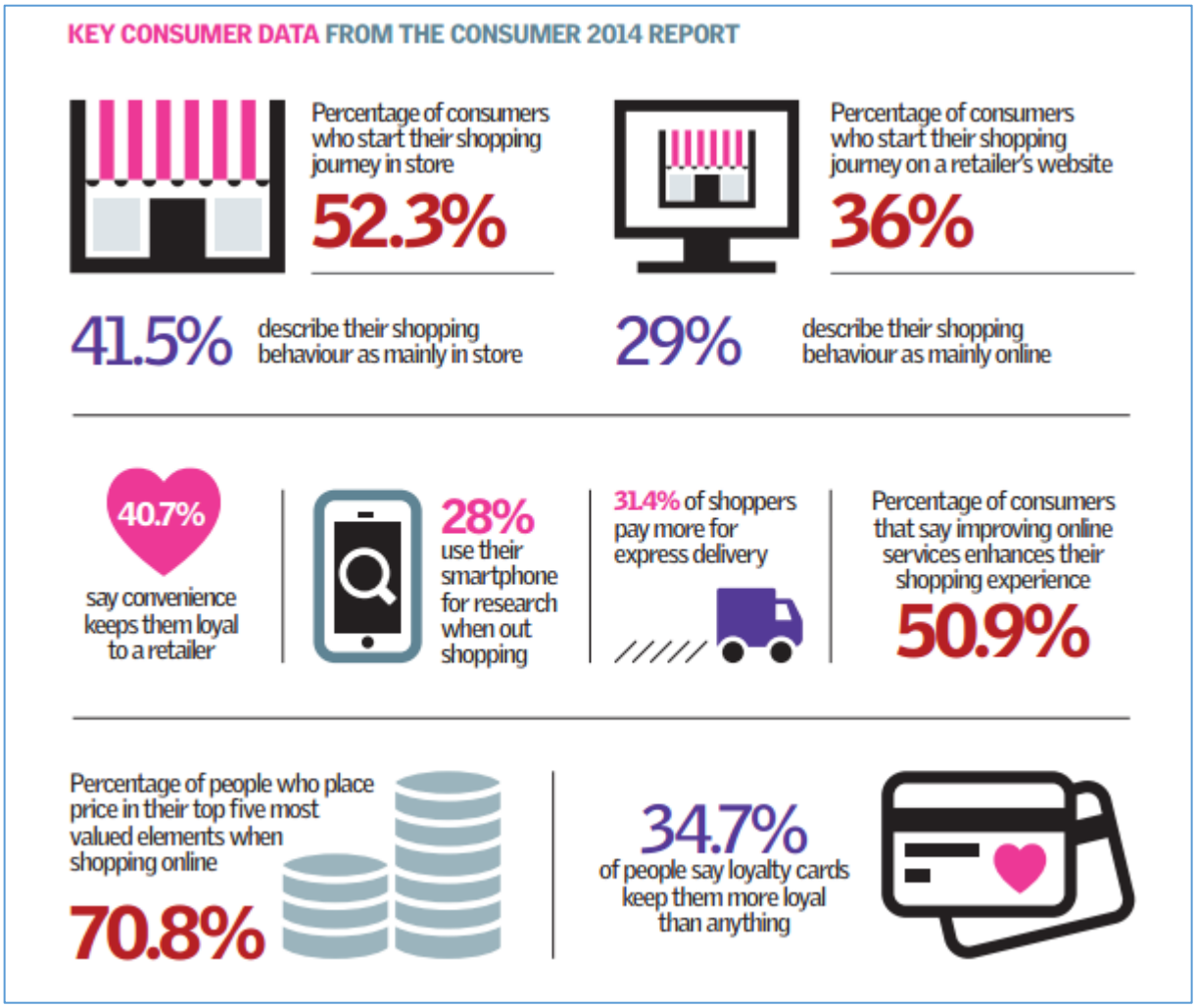
Vrishali Giri (U40856469)

CONTENTS

Executive Summary

The use of internet technology for performing business transaction has grown exponentially over the decade or so. Almost all companies have migrated their transactions to e-commerce and remaining ones are on their way to do so! While B2B ecommerce platforms are making billions of revenues, B2C has become successful in establishing good relation with huge consumer followers nonetheless.

The rise of online shopping have turned shoppers into promiscuous deal-hunters armed with mobile technology that allows them to shop whenever they want, however they want with whomever they want.



The goal of this project is to build an E-Commerce system for online shoppers giving them flexibility to shop their favourite items without having to visit any shop. An e-commerce

value chain represents a set of sequences which involves the interactions between online shoppers and e-commerce systems. We have focused on easy-to-understand user interface and query optimization in order to reduce the query processing time and give a user an hasslefree shopping experience. User role privileges are assigned in order to restrict the access between normal user and administrator. The system will generate up-to-date data based on number of items sold, availabe inventory and changes in prices.

Below is the weights distribution according to the areas covered –

Topic Area	Description	Points
Database Design	This part should include a logical database design (for the relational model), using normalization to control redundancy and integrity constraints for data quality.	
Query Writing	- SQL Queries - Stored Procedure	
Performance Tuning	- Indexing - Parallelism	
Other Topics	- DBA scripts - Database security - Data mining	

Assumptions

1. INV_Item_Number from the Inventory Item table should be present in the Order Item but not as primary key. Because if it is declared as foreign key then the referential integrity will be violated when INV_Item_Number from the Inventory Item table is deleted.

2. Join ORDER_ITEM and INVENTORY_ITEM by outer join rather than a natural join so that order item with deleted product will be included.

3. Credit card table has the 16-digit credit number as primary key this it is unique.

4. Order table provides detail on products and how many were ordered for a given invoice.

5. Residential and shipping address are stored for customers.

6. Customers can have multiple credit cards tied to their account. It can be different names and billing address.

7. Invoice contains the order details, shipping information and address of customer.

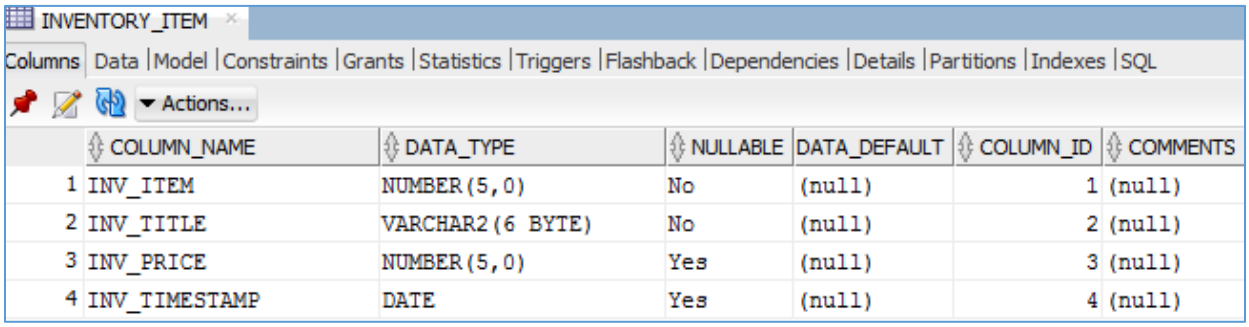
Part 1: Database Design

Section 1.1: Data Integrity

Below are the backend table creation queries which we have used.

A. INVENTORY_ITEM

```
CREATE TABLE "DB215"."INVENTORY_ITEM"
  (     "INV_ITEM" NUMBER(5,0) NOT NULL ENABLE,
        "INV_TITLE" VARCHAR2(6 BYTE) NOT NULL ENABLE,
        "INV_PRICE" NUMBER(5,0),
        "INV_TIMESTAMP" DATE,
         CONSTRAINT "INV_ITEM_PK" PRIMARY KEY ("INV_ITEM")      );
```


INVENTORY_ITEM

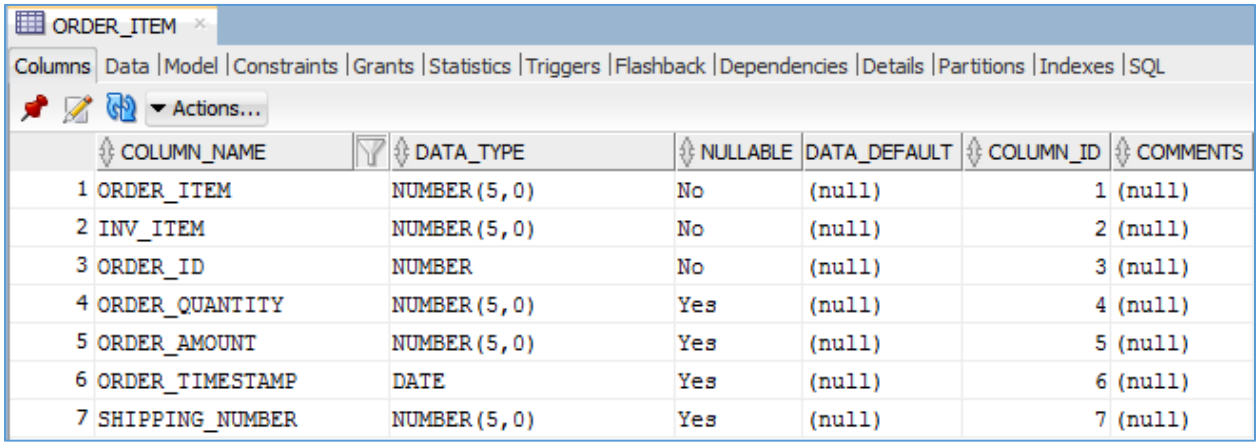
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	INV_ITEM	NUMBER(5,0)	No	(null)	1	(null)
2	INV_TITLE	VARCHAR2(6 BYTE)	No	(null)	2	(null)
3	INV_PRICE	NUMBER(5,0)	Yes	(null)	3	(null)
4	INV_TIMESTAMP	DATE	Yes	(null)	4	(null)

B. ORDER_ITEM

```
CREATE TABLE "DB215"."ORDER_ITEM"
  (    "ORDER_ITEM" NUMBER(5,0) NOT NULL ENABLE,
       "INV_ITEM" NUMBER(5,0) NOT NULL ENABLE,
       "ORDER_ID" NUMBER NOT NULL ENABLE,
       "ORDER_QUANTITY" NUMBER(5,0),
       "ORDER_AMOUNT" NUMBER(5,0),
       "ORDER_TIMESTAMP" DATE,
       "SHIPPING_NUMBER" NUMBER(5,0),
        CONSTRAINT "ORDER_ITEM_PK" PRIMARY KEY ("ORDER_ITEM")
        CONSTRAINT "INV_ITEM_FK" FOREIGN KEY ("INV_ITEM")
            REFERENCES "DB215"."INVENTORY_ITEM" ("INV_ITEM") ENABLE,
       CONSTRAINT "ORDER_ID_FK" FOREIGN KEY ("ORDER_ID")
            REFERENCES "DB215"."ORDERS" ("ORDER_ID") ENABLE,
       CONSTRAINT "SHIPPING_NUMBER_FK1" FOREIGN KEY ("SHIPPING_NUMBER")
            REFERENCES "DB215"."SHIPPING" ("SHIPPING_NUMBER") ENABLE    );
```



ORDER_ITEM

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_ITEM	NUMBER(5,0)	No	(null)	1	(null)
2	INV_ITEM	NUMBER(5,0)	No	(null)	2	(null)
3	ORDER_ID	NUMBER	No	(null)	3	(null)
4	ORDER_QUANTITY	NUMBER(5,0)	Yes	(null)	4	(null)
5	ORDER_AMOUNT	NUMBER(5,0)	Yes	(null)	5	(null)
6	ORDER_TIMESTAMP	DATE	Yes	(null)	6	(null)
7	SHIPPING_NUMBER	NUMBER(5,0)	Yes	(null)	7	(null)

C. INVOICE

```
CREATE TABLE "DB215"."INVOICE"
(       "INVOICE_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
        "ORDER_ID" NUMBER NOT NULL ENABLE,
        "ADDRESS_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
        "INVOICE_CREATION_DATE" DATE,
        "INVOICE_STATE" VARCHAR2(5 BYTE),
        "SHIPPING_NUMBER" NUMBER(5,0),
         CONSTRAINT "INVOICE_NUMBER_PK" PRIMARY KEY ("INVOICE_NUMBER")
         CONSTRAINT "ADDRESS_NUMBER_FK1" FOREIGN KEY ("ADDRESS_NUMBER")
                REFERENCES "DB215"."USER_INFO" ("ADDRESS_NUMBER") ENABLE,
         CONSTRAINT "ORDER_ID_FK2" FOREIGN KEY ("ORDER_ID")
                REFERENCES "DB215"."ORDERS" ("ORDER_ID") ENABLE,
         CONSTRAINT "SHIPPING_NUMBER_FK" FOREIGN KEY ("SHIPPING_NUMBER")
                REFERENCES "DB215"."SHIPPING" ("SHIPPING_NUMBER") ENABLE    );
```

INVOICE

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	INVOICE_NUMBER	NUMBER(5,0)	No	(null)	1	(null)
2	ORDER_ID	NUMBER	No	(null)	2	(null)
3	ADDRESS_NUMBER	NUMBER(5,0)	No	(null)	3	(null)
4	INVOICE_CREATION_DATE	DATE	Yes	(null)	4	(null)
5	INVOICE_STATE	VARCHAR2(5 BYTE)	Yes	(null)	5	(null)
6	SHIPPING_NUMBER	NUMBER(5,0)	Yes	(null)	6	(null)

D. USER_SESSION

```
CREATE TABLE "DB215"."USER_SESSION"
(       "SESSION_ID" NUMBER(5,0) NOT NULL ENABLE,
        "SESSION_CLICKS" NUMBER(5,0),
        "SESSION_TIME" DATE,
        "SESSION_IPADDRESS" VARCHAR2(100 BYTE),
        "CART_ID" NUMBER(5,0) NOT NULL ENABLE,
        "ACC_NUMBER" NUMBER(5,0),
         CONSTRAINT "USER_SESSION_PK" PRIMARY KEY ("SESSION_ID")
         CONSTRAINT "CART_ID_FK" FOREIGN KEY ("CART_ID")
                REFERENCES "DB215"."SHOPPING_CART" ("CART_ID") ENABLE,
        CONSTRAINT "ACC_NUMBER_FK2" FOREIGN KEY ("ACC_NUMBER")
                REFERENCES "DB215"."USER_ACCOUNT" ("ACC_NUMBER") ENABLE  );
```

USER_SESSION

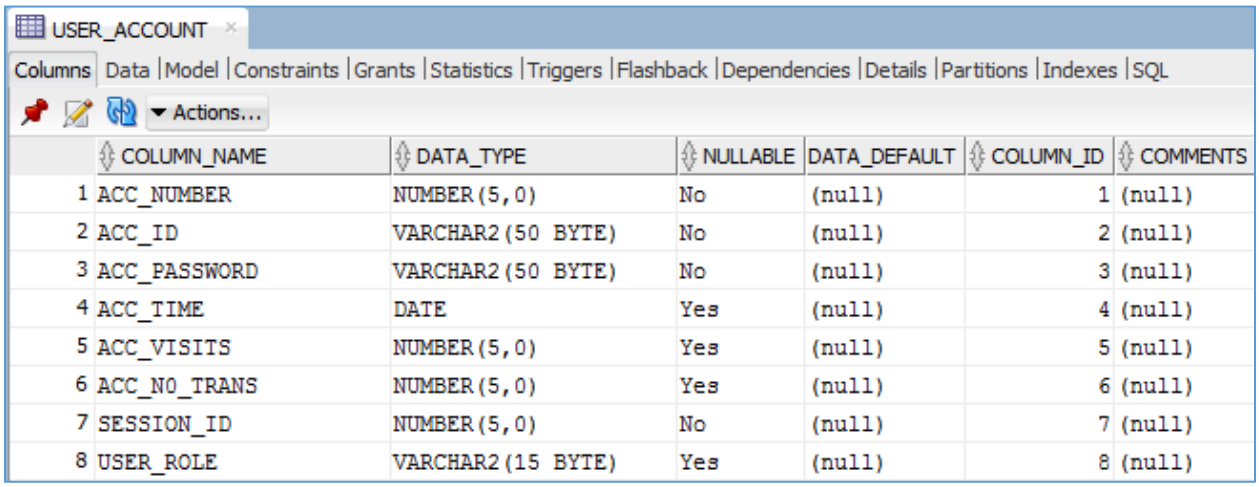
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	SESSION_ID	NUMBER(5,0)	No	(null)	1	(null)
2	SESSION_CLICKS	NUMBER(5,0)	Yes	(null)	2	(null)
3	SESSION_TIME	DATE	Yes	(null)	3	(null)
4	SESSION_IPADDRESS	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	CART_ID	NUMBER(5,0)	No	(null)	5	(null)
6	ACC_NUMBER	NUMBER(5,0)	Yes	(null)	6	(null)

E. USER_ACCOUNT

```
CREATE TABLE "DB215"."USER_ACCOUNT"
(       "ACC_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
        "ACC_ID" VARCHAR2(50 BYTE) NOT NULL ENABLE,
        "ACC_PASSWORD" VARCHAR2(50 BYTE) NOT NULL ENABLE,
        "ACC_TIME" DATE,
        "ACC_VISITS" NUMBER(5,0),
        "ACC_N0_TRANS" NUMBER(5,0),
        "SESSION_ID" NUMBER(5,0) NOT NULL ENABLE,
        "USER_ROLE" VARCHAR2(15 BYTE),
         CONSTRAINT "USER_ACCOUNT_PK" PRIMARY KEY ("ACC_NUMBER")
         CONSTRAINT "SESSION_ID_FK" FOREIGN KEY ("SESSION_ID")
                 REFERENCES "DB215"."USER_SESSION" ("SESSION_ID") ENABLE        );
```



USER_ACCOUNT

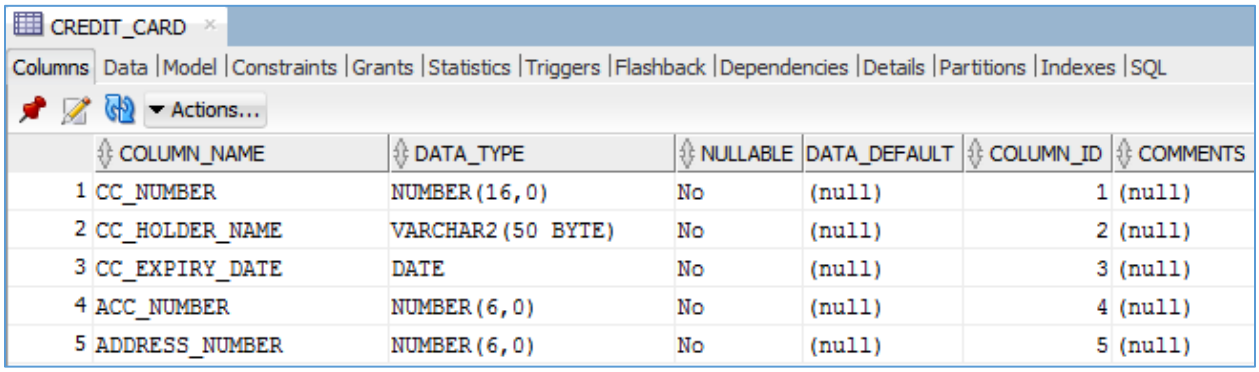
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ACC_NUMBER	NUMBER(5,0)	No	(null)	1	(null)
2	ACC_ID	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	ACC_PASSWORD	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	ACC_TIME	DATE	Yes	(null)	4	(null)
5	ACC_VISITS	NUMBER(5,0)	Yes	(null)	5	(null)
6	ACC_N0_TRANS	NUMBER(5,0)	Yes	(null)	6	(null)
7	SESSION_ID	NUMBER(5,0)	No	(null)	7	(null)
8	USER_ROLE	VARCHAR2(15 BYTE)	Yes	(null)	8	(null)

F. CREDIT_CARD

```
CREATE TABLE "DB215"."CREDIT_CARD"
(       "CC_NUMBER" NUMBER(16,0) NOT NULL ENABLE,
        "CC_HOLDER_NAME" VARCHAR2(50 BYTE) NOT NULL ENABLE,
        "CC_EXPIRY_DATE" DATE NOT NULL ENABLE,
        "ACC_NUMBER" NUMBER(6,0) NOT NULL ENABLE,
        "ADDRESS_NUMBER" NUMBER(6,0) NOT NULL ENABLE,
         CONSTRAINT "CREDIT_CARD_PK" PRIMARY KEY ("CC_NUMBER")
         CONSTRAINT "ACC_NUMBER_FK" FOREIGN KEY ("ACC_NUMBER")
                REFERENCES "DB215"."USER_ACCOUNT" ("ACC_NUMBER") ENABLE,
        CONSTRAINT "ADDRESS_NUMBER_FK" FOREIGN KEY ("ADDRESS_NUMBER")
                REFERENCES "DB215"."USER_INFO" ("ADDRESS_NUMBER") ENABLE );
```



CREDIT_CARD

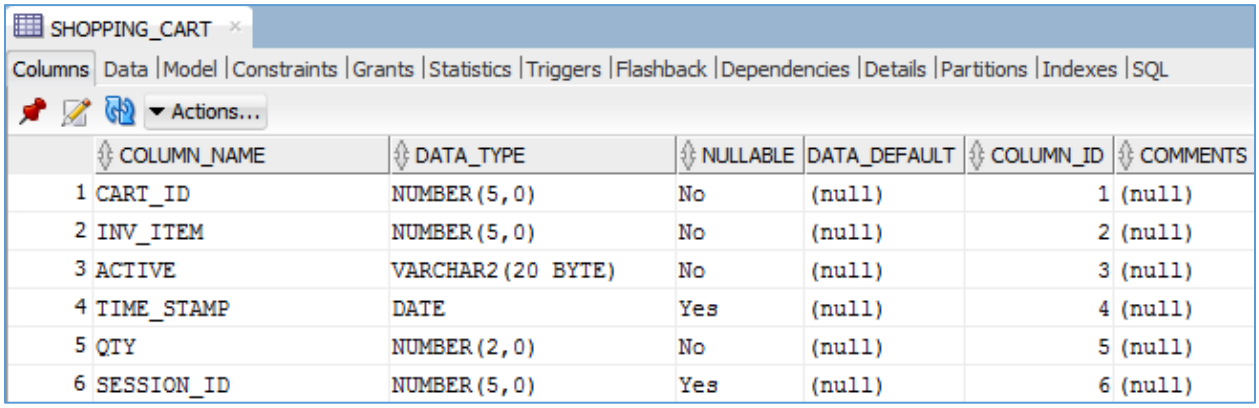
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CC_NUMBER	NUMBER(16,0)	No	(null)	1	(null)
2	CC_HOLDER_NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	CC_EXPIRY_DATE	DATE	No	(null)	3	(null)
4	ACC_NUMBER	NUMBER(6,0)	No	(null)	4	(null)
5	ADDRESS_NUMBER	NUMBER(6,0)	No	(null)	5	(null)

G. SHOPPING_CART

```
CREATE TABLE "DB215"."SHOPPING_CART"
(       "CART_ID" NUMBER(5,0) NOT NULL ENABLE,
        "INV_ITEM" NUMBER(5,0) NOT NULL ENABLE,
        "ACTIVE" VARCHAR2(20 BYTE) NOT NULL ENABLE,
        "TIME_STAMP" DATE,
        "QTY" NUMBER(2,0) NOT NULL ENABLE,
        "SESSION_ID" NUMBER(5,0),
         CONSTRAINT "SHOPPING_CART_PK" PRIMARY KEY ("CART_ID")
         CONSTRAINT "INVENTORY_ITEM_FK1" FOREIGN KEY ("INV_ITEM")
                REFERENCES "DB215"."INVENTORY_ITEM" ("INV_ITEM") ENABLE,
        CONSTRAINT "SESSION_ID_FK1" FOREIGN KEY ("SESSION_ID")
                REFERENCES "DB215"."USER_SESSION" ("SESSION_ID") ENABLE        );
```



SHOPPING_CART

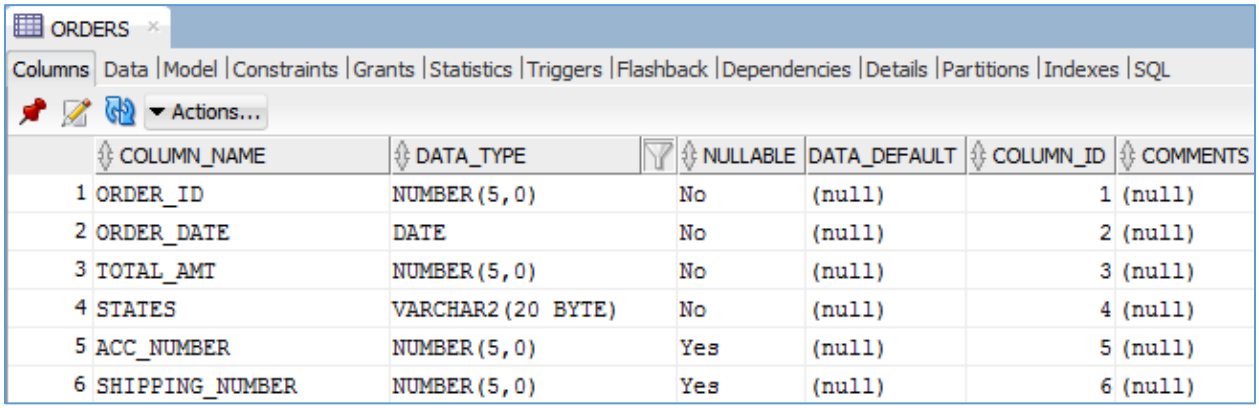
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CART_ID	NUMBER(5,0)	No	(null)	1	(null)
2	INV_ITEM	NUMBER(5,0)	No	(null)	2	(null)
3	ACTIVE	VARCHAR2(20 BYTE)	No	(null)	3	(null)
4	TIME_STAMP	DATE	Yes	(null)	4	(null)
5	QTY	NUMBER(2,0)	No	(null)	5	(null)
6	SESSION_ID	NUMBER(5,0)	Yes	(null)	6	(null)

H. ORDERS

```
CREATE TABLE "DB215"."ORDERS"
(       "ORDER_ID" NUMBER(5,0) NOT NULL ENABLE,
        "ORDER_DATE" DATE NOT NULL ENABLE,
        "TOTAL_AMT" NUMBER(5,0) NOT NULL ENABLE,
        "STATES" VARCHAR2(20 BYTE) NOT NULL ENABLE,
        "ACC_NUMBER" NUMBER(5,0),
        "SHIPPING_NUMBER" NUMBER(5,0),
         CONSTRAINT "ORDERS_PK" PRIMARY KEY ("ORDER_ID")
         CONSTRAINT "ACC_NUMBER_FK1" FOREIGN KEY ("ACC_NUMBER")
                REFERENCES "DB215"."USER_ACCOUNT" ("ACC_NUMBER") ENABLE,
        CONSTRAINT "SHIPPING_NUMBER_FK2" FOREIGN KEY ("SHIPPING_NUMBER")
                REFERENCES "DB215"."SHIPPING" ("SHIPPING_NUMBER") ENABLE    );
```



ORDERS

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(5,0)	No	(null)	1	(null)
2	ORDER_DATE	DATE	No	(null)	2	(null)
3	TOTAL_AMT	NUMBER(5,0)	No	(null)	3	(null)
4	STATES	VARCHAR2(20 BYTE)	No	(null)	4	(null)
5	ACC_NUMBER	NUMBER(5,0)	Yes	(null)	5	(null)
6	SHIPPING_NUMBER	NUMBER(5,0)	Yes	(null)	6	(null)

I. PAYMENT

```
CREATE TABLE "DB215"."PAYMENT"
(    "PAYMENT_ID" NUMBER(5,0) NOT NULL ENABLE,
     "ORDER_ID" NUMBER(5,0) NOT NULL ENABLE,
     "CC_NUMBER" NUMBER(16,0) NOT NULL ENABLE,
     "AMOUNT" NUMBER(5,0),
     "STATES" VARCHAR2(20 BYTE),
     "TIME_STAMP" DATE,
      CONSTRAINT "PAYMENT_PK" PRIMARY KEY ("PAYMENT_ID")
      CONSTRAINT "PAYMENT_ORDERS_FK1" FOREIGN KEY ("ORDER_ID")
          REFERENCES "DB215"."ORDERS" ("ORDER_ID") ENABLE,
      CONSTRAINT "PAYMENT_CREDIT_CARD_FK2" FOREIGN KEY ("CC_NUMBER")
          REFERENCES "DB215"."CREDIT_CARD" ("CC_NUMBER") ENABLE    );
```

PAYMENT

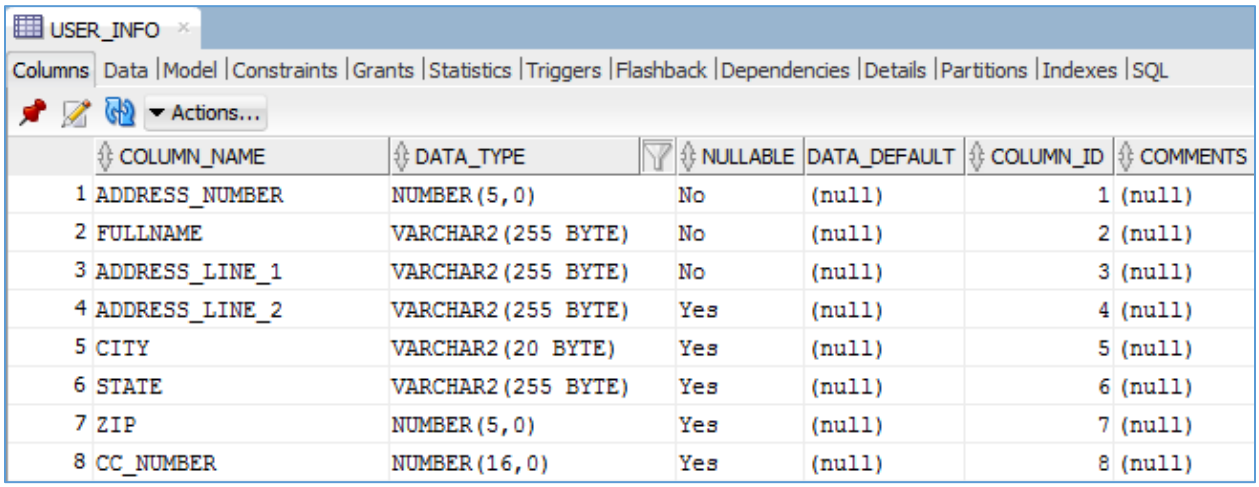
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	PAYMENT_ID	NUMBER(5,0)	No	(null)	1	(null)
2	ORDER_ID	NUMBER(5,0)	No	(null)	2	(null)
3	CC_NUMBER	NUMBER(16,0)	No	(null)	3	(null)
4	AMOUNT	NUMBER(5,0)	Yes	(null)	4	(null)
5	STATES	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)
6	TIME_STAMP	DATE	Yes	(null)	6	(null)

J. USER_INFO

```
CREATE TABLE "DB215"."USER_INFO"
(       "ADDRESS_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
        "FULLNAME" VARCHAR2(255 BYTE) NOT NULL ENABLE,
        "ADDRESS_LINE_1" VARCHAR2(255 BYTE) NOT NULL ENABLE,
        "ADDRESS_LINE_2" VARCHAR2(255 BYTE),
        "CITY" VARCHAR2(20 BYTE),
        "STATE" VARCHAR2(255 BYTE),
        "ZIP" NUMBER(5,0),
        "CC_NUMBER" NUMBER(16,0),
         CONSTRAINT "ADDRESS_NUMBER_PK" PRIMARY KEY ("ADDRESS_NUMBER")
         CONSTRAINT "CC_NUMBER_FK1" FOREIGN KEY ("CC_NUMBER")
                 REFERENCES "DB215"."CREDIT_CARD" ("CC_NUMBER") ENABLE       );
```



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ADDRESS_NUMBER	NUMBER(5,0)	No	(null)	1	(null)
2	FULLNAME	VARCHAR2(255 BYTE)	No	(null)	2	(null)
3	ADDRESS_LINE_1	VARCHAR2(255 BYTE)	No	(null)	3	(null)
4	ADDRESS_LINE_2	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)
5	CITY	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)
6	STATE	VARCHAR2(255 BYTE)	Yes	(null)	6	(null)
7	ZIP	NUMBER(5,0)	Yes	(null)	7	(null)
8	CC_NUMBER	NUMBER(16,0)	Yes	(null)	8	(null)

K. SHIPPING

```
CREATE TABLE "DB215"."SHIPPING"
(    "SHIPPING_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
     "ORDER_ID" NUMBER(5,0) NOT NULL ENABLE,
     "INVOICE_NUMBER" NUMBER(5,0) NOT NULL ENABLE,
     "SHIPPING_METHOD" VARCHAR2(255 BYTE) NOT NULL ENABLE,
     "SHIPPING_CHARGE" NUMBER(5,0) NOT NULL ENABLE,
     "ZIP" NUMBER(5,0),
     "SHIPPING_DATE" DATE NOT NULL ENABLE,
     "ADDRESS_NUMBER" NUMBER(5,0),
      CONSTRAINT "SHIPPING_NUMBER_PK" PRIMARY KEY ("SHIPPING_NUMBER")
      CONSTRAINT "SHIPPING_ORDER_ID_FK" FOREIGN KEY ("ORDER_ID")
          REFERENCES "DB215"."ORDERS" ("ORDER_ID") ENABLE,
      CONSTRAINT "SHIP_INVOICE_NUMBER_FK" FOREIGN KEY ("INVOICE_NUMBER")
          REFERENCES "DB215"."INVOICE" ("INVOICE_NUMBER") ENABLE,
      CONSTRAINT "ADDRESS_NUMBER_FK2" FOREIGN KEY ("ADDRESS_NUMBER")
          REFERENCES "DB215"."USER_INFO" ("ADDRESS_NUMBER") ENABLE );
```

SHIPPING

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	SHIPPING_NUMBER	NUMBER(5,0)	No	(null)	1	(null)
2	ORDER_ID	NUMBER(5,0)	No	(null)	2	(null)
3	INVOICE_NUMBER	NUMBER(5,0)	No	(null)	3	(null)
4	SHIPPING_METHOD	VARCHAR2(255 BYTE)	No	(null)	4	(null)
5	SHIPPING_CHARGE	NUMBER(5,0)	No	(null)	5	(null)
6	ZIP	NUMBER(5,0)	Yes	(null)	6	(null)
7	SHIPPING_DATE	DATE	No	(null)	7	(null)
8	ADDRESS_NUMBER	NUMBER(5,0)	Yes	(null)	8	(null)

Section 1.2: Data Generation and Loading

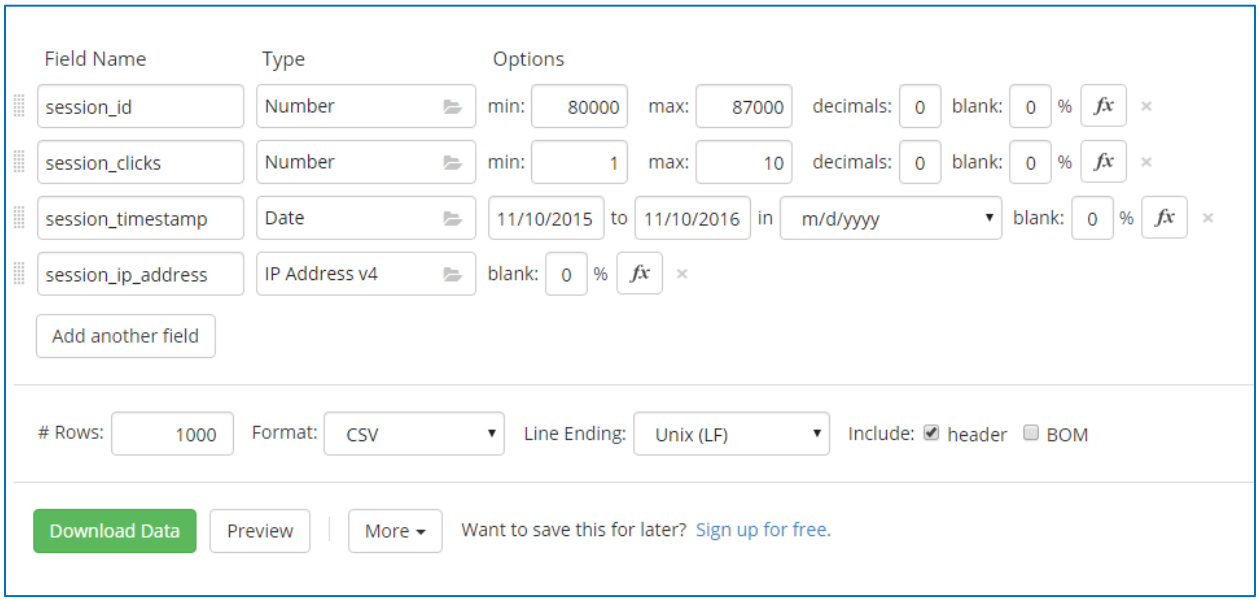
A. Data Generation

- The database design has 11 tables and the data is generate dfor all of them.
- The user account table includes user_id, password, number of visits, total transaction amount.
- Shipping information can be tracked form the shipping table.
- Payment information can be tracked from the payment table.
- The table user_session has 1000 rows rest all the tables 5000 rows each.

The external data is created using:

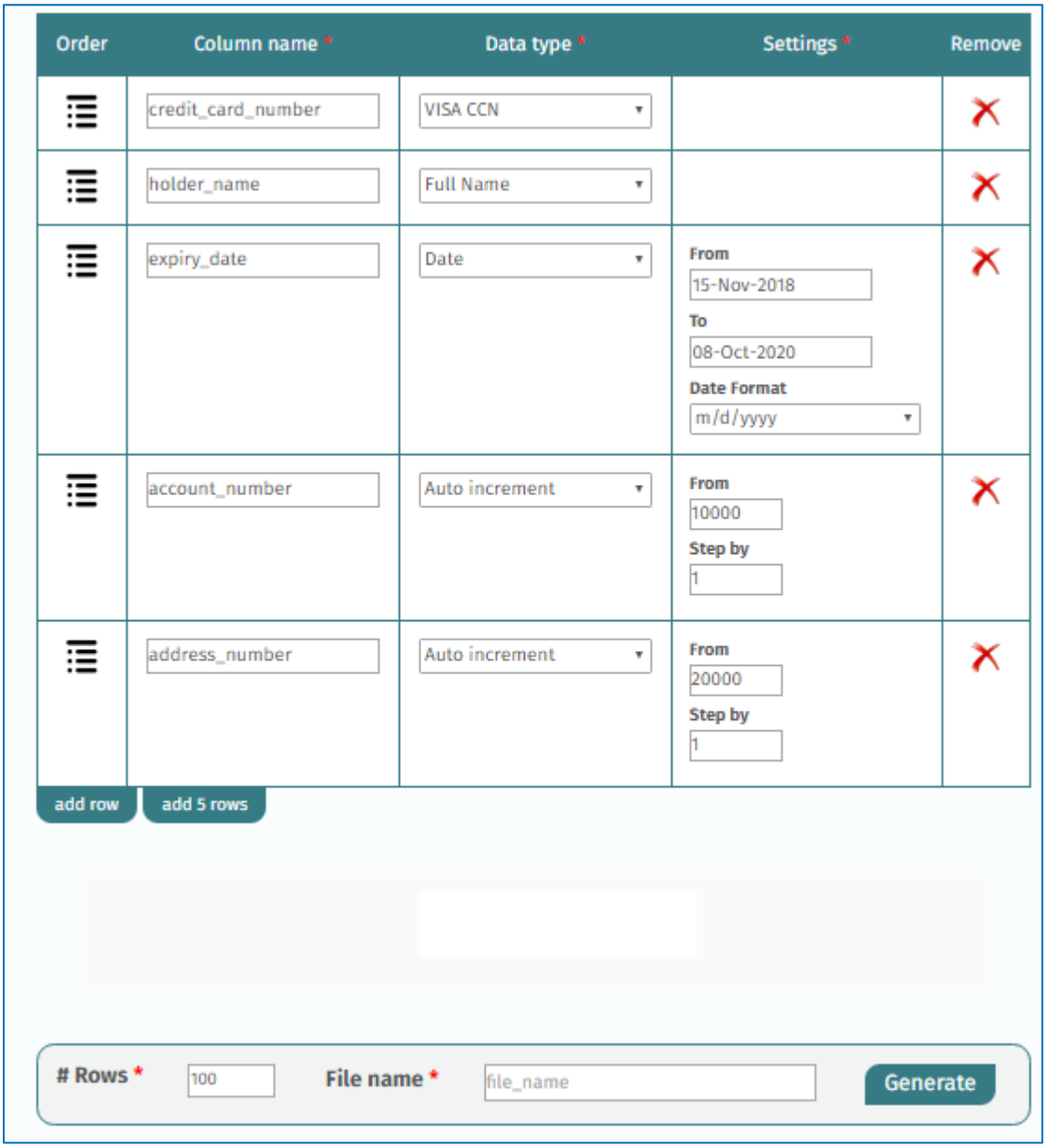
https://www.mockaroo.com

http://www.yandataellan.com



The user_session data has been created using the online generator as shown in the above snapshot. We can select the data type for each attribute value as specified in the data base design and download the csv format using the download button. Here the session_id is created as random number ranging from 80000 to 87000, session_clicks as number ranging from 1 to 10, session time ranging from 11/10/2015 to 11/10/2016 and session_ip_address as ip address v4. Using this we can generate 1000 rows.

Below is the screenshot for creating tables with 5000 rows each.



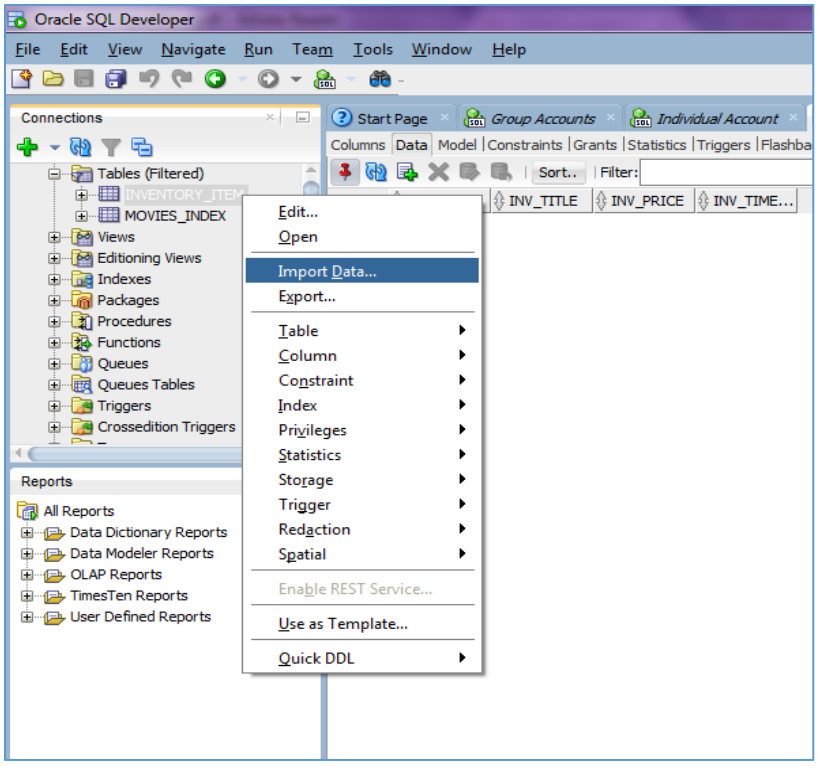
B. Data Loading

All the required data is initially generated using the external data generator and is stored in the spreadsheets. Data is then imported to SQL developer using the import feature.

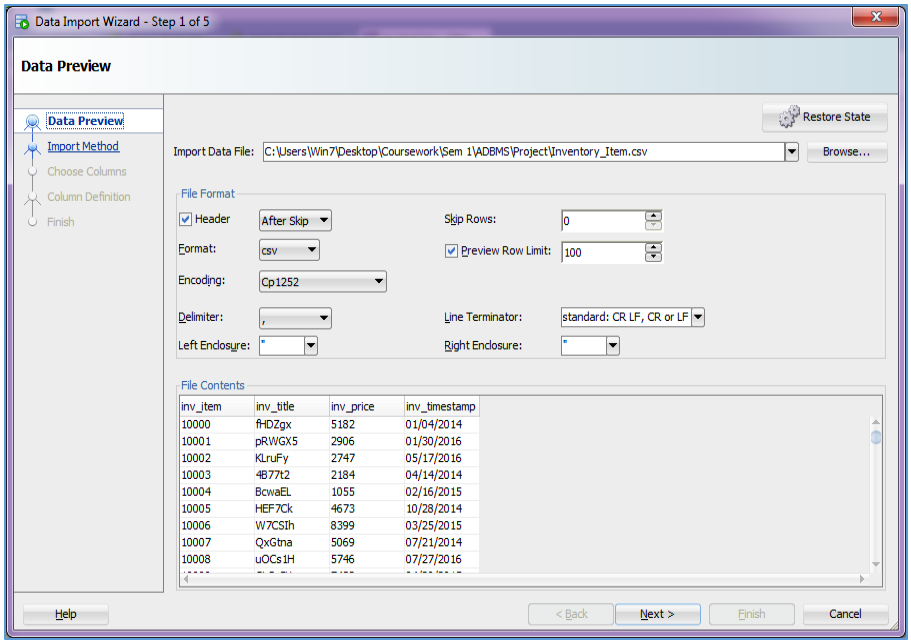
Following are steps to import data to table:

1. Right click on the table and select “import data”.
2. Select the spread sheet containing data.
3. Select the format and spread sheet in the xls file (select headers).
4. Import method is insert.
5. Add all the columns required from file.
6. Map all the columns in spread sheet to columns in the database.
7. Verify parameters before import.
8. If the verification is successful, click finish to import the data.

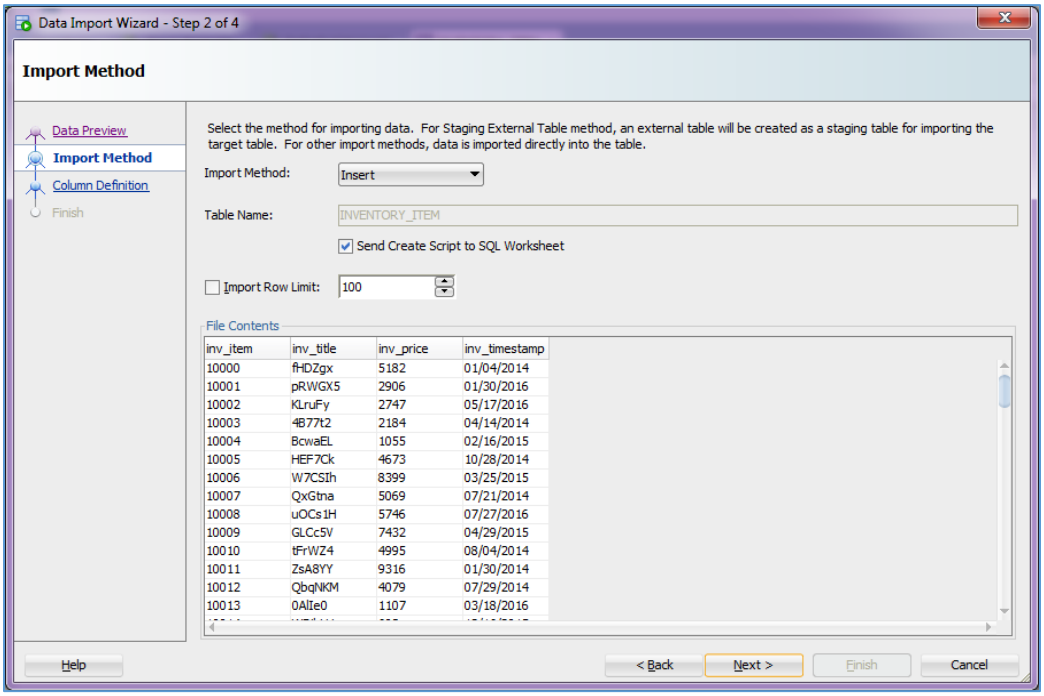
Below images shows the data import wizard and the way the data in imported into the INVENTORY_ITEM table.



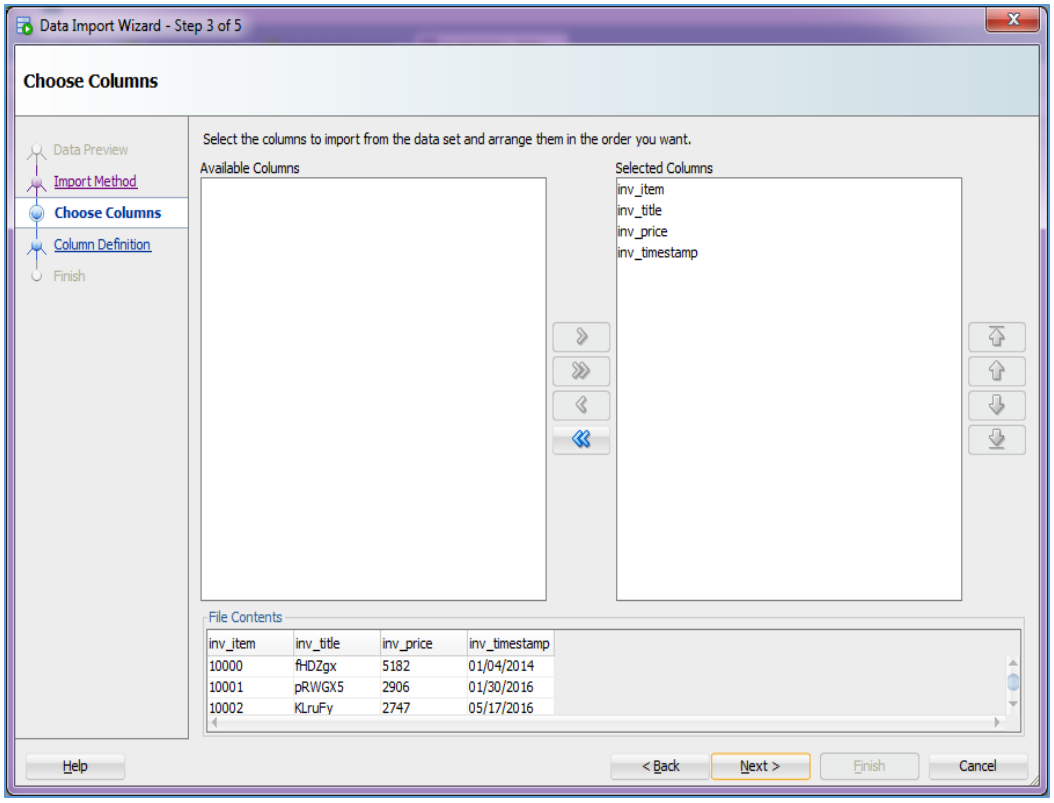
Right click on the table and select “import data”



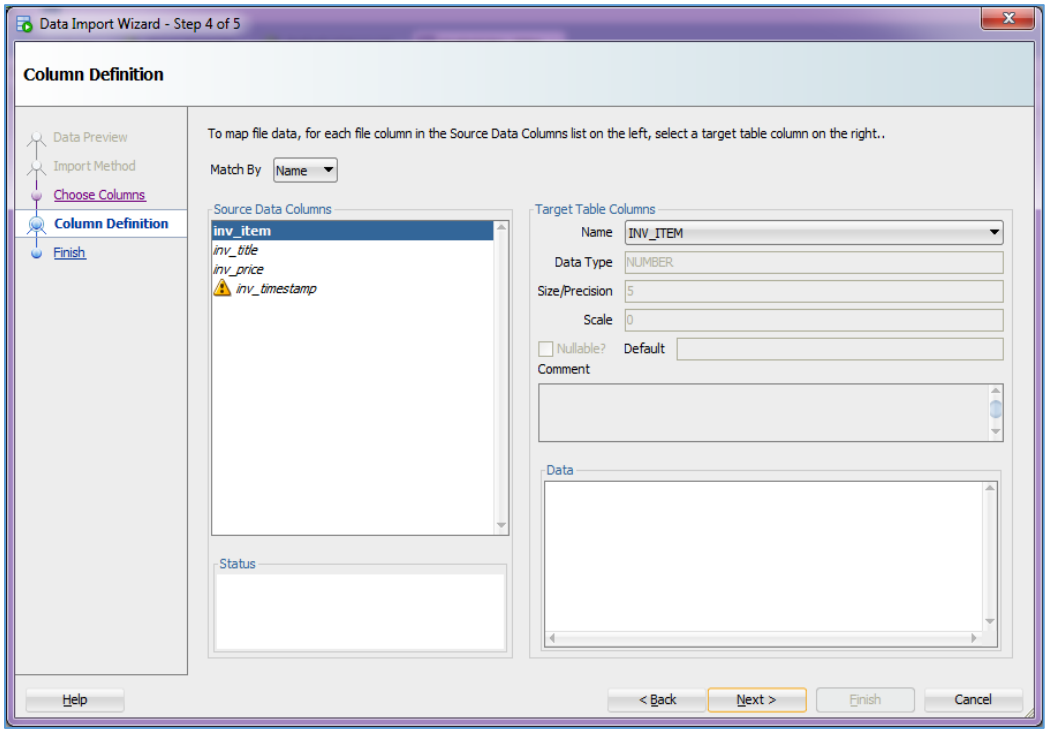
Data Import Wizard



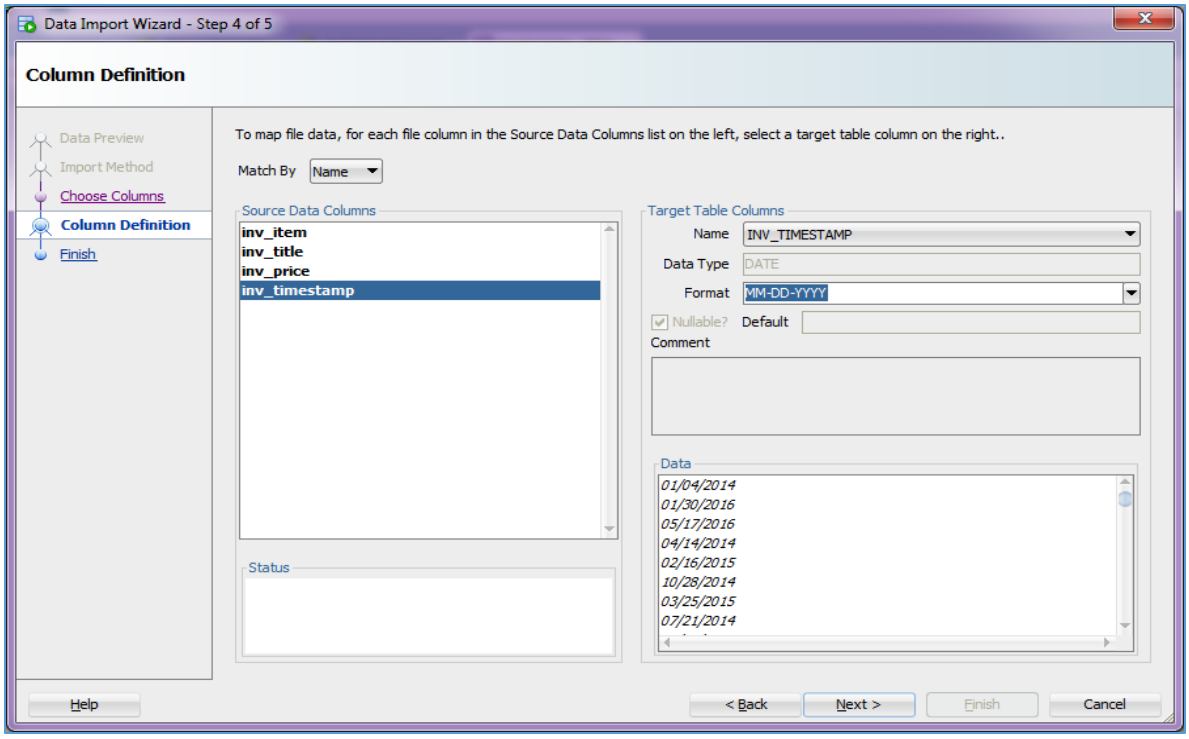
Data Preview in Data Import Wizard



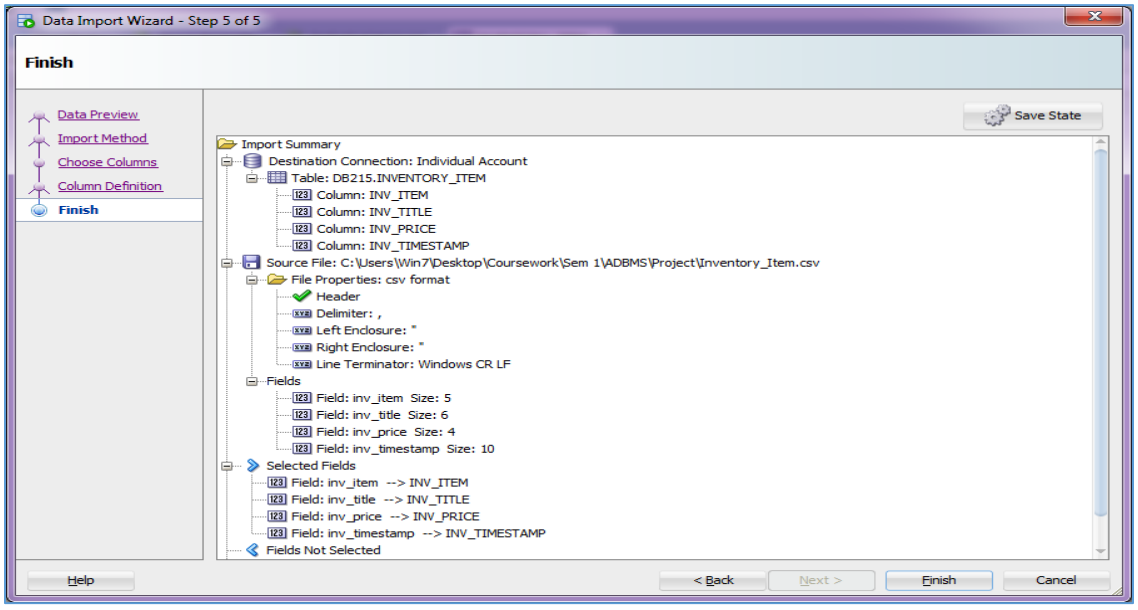
Choose Column in Data import Wizard



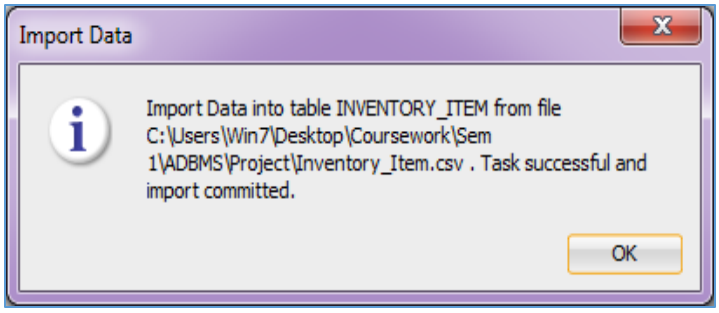
Column Definition in Data import Wizard



Column Definition for Inventory_Item Table in Data import Wizard



Final Step in Data Import Wizard



Data imported successful

Start Page | INVENTORY_ITEM | Individual Account

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashbad

Sort.. | Filter:

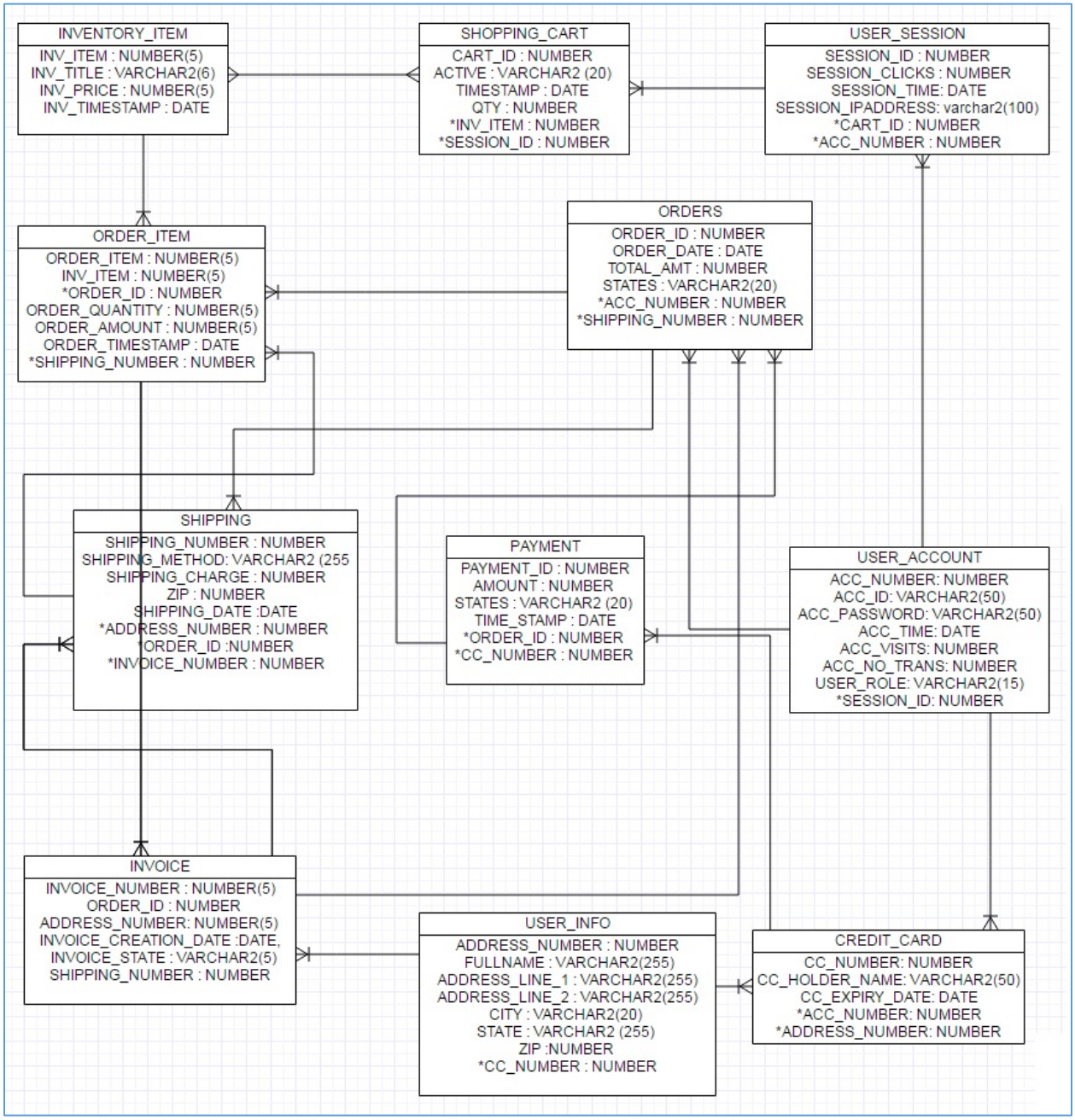
	INV_ITEM	INV_TITLE	INV_PRICE	INV_TIMESTAMP
1	11000	ju1vK3	8550	28-MAY-14
2	11001	6j53df	4855	06-JAN-16
3	11002	nq3ak5	1885	11-FEB-14
4	11003	ms6wAN	7377	24-NOV-14
5	11004	4i4er6	2972	05-APR-14
6	11005	eaKmmN	7965	29-NOV-15
7	11006	PLtpTb	3195	15-JUL-15
8	11007	8hYifI	6357	28-APR-14
9	11008	gT4Ehp	1865	15-JUN-14
10	11009	MNbTcj	7234	07-AUG-16
11	11010	WmVY0C	1452	01-APR-14
12	11011	2kVTVw	9485	22-DEC-14
13	11012	vVwFtL	775	07-DEC-15
14	11013	c4kixV	5418	21-AUG-14
15	11014	KhATDo	2711	03-APR-14

Data in the Inventory_Item Table

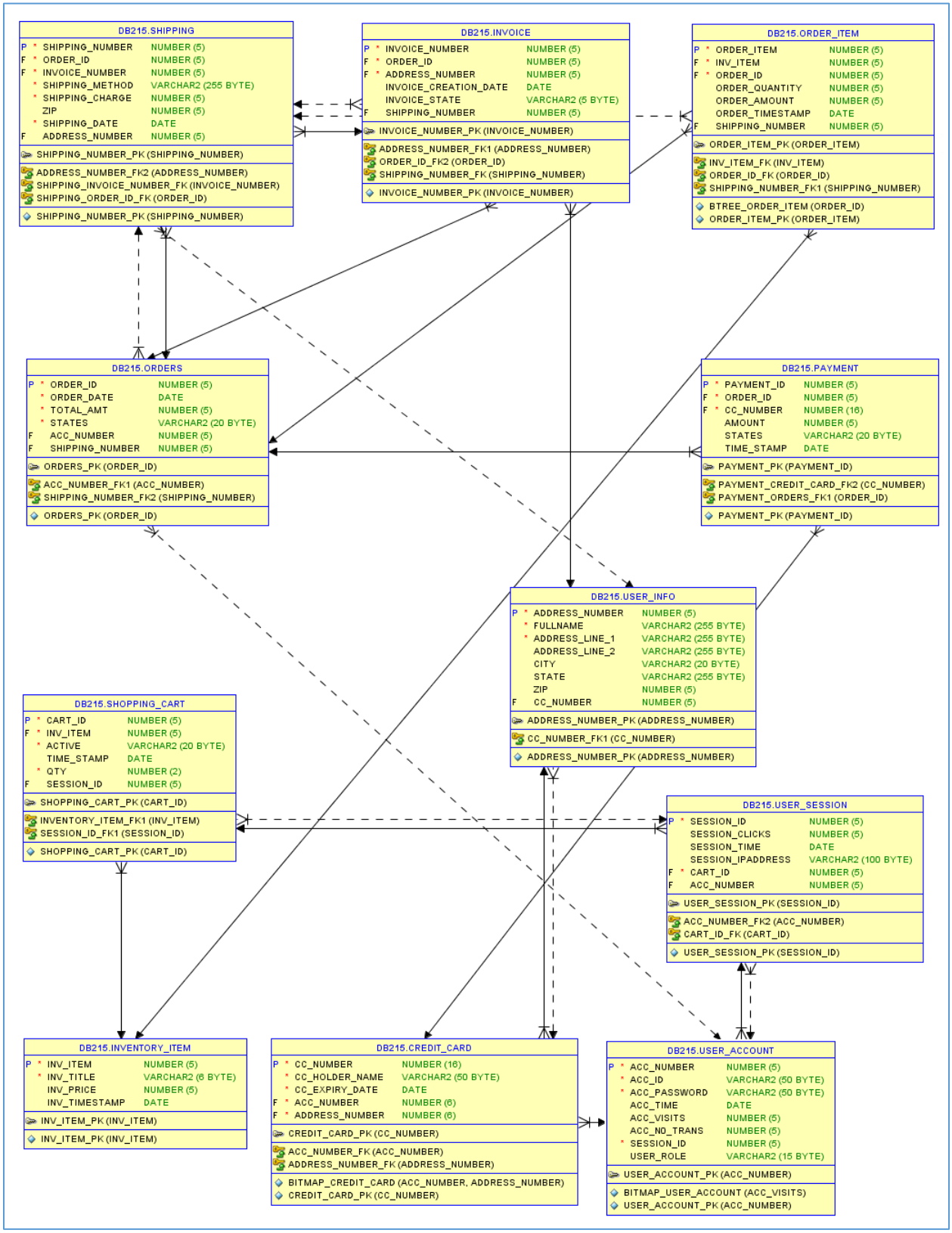
The number of record in each table is shown below:

Table	Number of Tuples
INVENTORY_ITEM	5000
ORDER_ITEM	5000
INVOICE	5000
USER_SESSION	1000
USER_ACCOUNT	5000
CREDIT_CARD	5000
SHOPPING_CART	5000
ORDERS	5000
PAYMENT	5000
USER_INFO	5000
SHIPPING	5000

Section 1.3: Logical Database Design



Section 1.4: Physical Database Design



Part 2: Query Writing

Section 2.1: SQL Queries

Section 2.2: Database Programming / Stored Procedures

Stored procedures in DBMS are group of SQL statements integrated with a programming block, which can be named and stored in backend. Stored procedures can be shared and invoked by the ir name from any program.

Below stored procedure is created to insert new items in inventory. The procedure performs check on USER_ROLE value from table USER_ACCOUNT before inserting record into the table. If the user is 'Admin', only then the insertion is allowed. Procedure will come out of the loop if the role is other than 'Admin'. In other words, every time Administrator tries to insert new record in INVENTORY_ITEM table, procedure 'UpdateInventory' is invoked.

```
CREATE OR REPLACE Procedure UpdateInventory
   ( user_id IN number, title_in IN VARCHAR2, price IN NUMBER)

IS

   urole VARCHAR2(15);

   CURSOR c1 IS
       SELECT user_role
       FROM user_account
       WHERE acc_number = user_id;

BEGIN

   OPEN c1;
   FETCH c1 INTO urole;

   WHILE c1 = 'Admin'
   LOOP
       INSERT INTO INVENTORY_ITEM
       ( inv_title, inv_price, inv_timestamp)
       VALUES
       ( title_in, price_in, sysdate);
   END LOOP;

   CLOSE c1;

   END;
```

Part 3: Performance Tuning

Section 3.1: Indexing

Indexing is an essential part of performance tuning in database design. It helps improving the performance of query. We have used 2 types of indexing techniques while designing database - B-Tree and Bitmap Indexing.

Before we begin explicitely creating indexes on tables, let us first take a look at current status of indexes in our database.

The below query will extract the index structure from database schema.

```
SELECT ui.table_name,
ui.index_name,
TO_CHAR((ui.distinct_keys / ui.num_rows) * 100, '999.99') selectivity,
ui.distinct_keys,
ui.num_rows,
ui.index_type
FROM user_indexes ui
WHERE ui.num_rows > 0
ORDER BY ui.distinct_keys / ui.num_rows;
```

As we can see, currently we have only primary key index type which was assigned while creating tables. We will focus on tables those are densly populated and try to take a measurable stance to improve the query retrival time to some extent.

	TABLE_NAME	INDEX_NAME	SELECTIVITY	DISTINCT_KEYS	NUM_ROWS	INDEX_TYPE
1	ADDRESS	ADDRESS_NUMBER_PK	100.00	4510	4510	NORMAL
2	CREDIT_CARD	CREDIT_CARD_PK	100.00	4510	4510	NORMAL
3	INVOICE	INVOICE_NUMBER_PK	100.00	4510	4510	NORMAL
4	INVENTORY_ITEM	INV_ITEM_PK	100.00	5000	5000	NORMAL
5	ORDERS	ORDERS_PK	100.00	5000	5000	NORMAL
6	USER_SESSION	USER_SESSION_PK	100.00	1000	1000	NORMAL
7	PAYMENT	PAYMENT_PK	100.00	4510	4510	NORMAL
8	SHIPPING	SHIPPING_NUMBER_PK	100.00	4221	4221	NORMAL
9	SHOPPING_CART	SHOPPING_CART_PK	100.00	5000	5000	NORMAL
10	USER_ACCOUNT	USER_ACCOUNT_PK	100.00	5000	5000	NORMAL
11	ORDER_ITEM	ORDER_ITEM_PK	100.00	5000	5000	NORMAL

B-Tree Index

Let us execute the below complex query without creating any explicit index on tables.

```
SELECT
  ORD.ORDER_ITEM, ORD.ORDER_QUANTITY, ORD.ORDER_AMOUNT, ORD.ORDER_TIMESTAMP,
  INV.INVOICE_CREATION_DATE, COUNT(*) AS "TOTAL ROWS"
FROM ORDER_ITEM ORD INNER JOIN INVOICE INV
ON (ORD.ORDER_ID = INV.ORDER_ID)
WHERE ORD.ORDER_QUANTITY > 668
GROUP BY ORD.ORDER_TIMESTAMP, ORD.ORDER_ITEM,
  ORD.ORDER_QUANTITY, ORD.ORDER_AMOUNT, INV.INVOICE_CREATION_DATE
HAVING COUNT(*) = 1
ORDER BY ORD.ORDER_AMOUNT;
```

As we can see, the query execution time is 0.062 seconds and the cost of query is 21.

Query Result

SQL | Fetched 50 rows in 0.146 seconds

	ORDER_ITEM	ORDER_QUANTITY	ORDER_AMOUNT	ORDER_TIMESTAMP	INVOICE_CREATION_DATE	TOTAL ROWS
1	20378	830	1001	17-OCT-16	02-APR-15	1
2	20378	830	1001	17-OCT-16	04-JUL-15	1
3	20378	830	1001	17-OCT-16	17-JUL-15	1
4	20378	830	1001	17-OCT-16	29-JAN-16	1
5	20378	830	1001	17-OCT-16	29-MAR-16	1
6	20311	853	1007	06-OCT-16	15-APR-15	1
7	20311	853	1007	06-OCT-16	17-AUG-15	1
8	20311	853	1007	06-OCT-16	01-DEC-15	1
9	22818	966	1008	06-FEB-15	08-JAN-15	1
10	22818	966	1008	06-FEB-15	02-MAR-15	1
11	22818	966	1008	06-FEB-15	14-JUN-15	1
12	22818	966	1008	06-FEB-15	02-DEC-15	1
13	22818	966	1008	06-FEB-15	02-JUL-16	1
14	20713	779	1014	02-OCT-15	07-SEP-15	1

Query Result | Autotrace

SQL HotSpot | 3.473 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			21
FILTER			
Filter Predicates			
COUNT(*)=1			
SORT (GROUP BY)		84	21
HASH JOIN		8384	20
Access Predicates			
ORD.ORDER_ID=INV.ORDER_ID			
TABLE ACCESS (FULL)	ORDER_ITEM	1677	11
Filter Predicates			
ORD.ORDER_QUANTITY>668			
TABLE ACCESS (FULL)	INVOICE	5000	9

We decided to create a B-Tree index on table ORDER_ITEM and on column ORDER_ID.

```
CREATE INDEX btree_order_item ON ORDER_ITEM (ORDER_ID);
```

Post index creation, we executed the same query again. We could see that this time the query took only 0.072 seconds to fetch data and query cost is reduced to 13.

Script Output | Query Result | Autotrace

SQL | Fetched 50 rows in 0.072 seconds

	ORDER_ITEM	ORDER_QUANTITY	ORDER_AMOUNT	ORDER_TIMESTAMP	INVOICE_CREATION_DATE	TOTAL ROWS
1	20378	830	1001	17-OCT-16	02-APR-15	1
2	20378	830	1001	17-OCT-16	04-JUL-15	1
3	20378	830	1001	17-OCT-16	17-JUL-15	1
4	20378	830	1001	17-OCT-16	29-JAN-16	1
5	20378	830	1001	17-OCT-16	29-MAR-16	1
6	20311	853	1007	06-OCT-16	15-APR-15	1
7	20311	853	1007	06-OCT-16	17-AUG-15	1
8	20311	853	1007	06-OCT-16	01-DEC-15	1
9	22818	966	1008	06-FEB-15	08-JAN-15	1
10	22818	966	1008	06-FEB-15	02-MAR-15	1
11	22818	966	1008	06-FEB-15	14-JUN-15	1
12	22818	966	1008	06-FEB-15	02-DEC-15	1
13	22818	966	1008	06-FEB-15	02-JUL-16	1
14	20713	779	1014	02-OCT-15	07-SEP-15	1

Query Result | V$SQL_PLAN.SQL_ID=gvym1cpx045d0 | Script Output | Autotrace

SQL HotSpot | 1.788 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME
SELECT STATEMENT			13		
FILTER				62	2396
Filter Predicates					
COUNT(*)=1					
SORT (GROUP BY)		16	13	62	2368
HASH JOIN		1677	12	62	1430
Access Predicates					
ORD.ORDER_ID=INV.ORDER_ID					
TABLE ACCESS (FULL)	ORDER_ITEM	1677	6	31	275
Filter Predicates					
ORD.ORDER_QUANTITY>66					
TABLE ACCESS (FULL)	INVOICE	4510	6	31	181

B-trees are the most widely used index type and a ubiquitous structure in computer science. Let us now create Bitmap index on comparatively densed table and observe the outcome.

Bitmap Index

Bitmap index structure uses bit-vector concept to indiacte which of the values occur in a row.

Below query is executed without creating any index on tables used.

```
SELECT UA.ACC_NUMBER, UA.ACC_VISITS,
AD.FULLNAME, AD.CITY, CC.CC_EXPIRY_DATE
FROM USER_ACCOUNT UA INNER JOIN CREDIT_CARD CC
  ON (UA.ACC_NUMBER = CC.ACC_NUMBER)
  INNER JOIN ADDRESS AD
  ON (AD.ADDRESS_NUMBER = CC.ADDRESS_NUMBER)
WHERE UA.ACC_VISITS > 4000
GROUP BY UA.ACC_VISITS, UA.ACC_NUMBER, AD.FULLNAME, AD.CITY, CC.CC_EXPIRY_DATE
ORDER BY UA.ACC_VISITS DESC;
```

As we can see, the query execution took 2.21 seconds with the cost of query as 31.

Script Output | Autotrace | Query Result

SQL | Fetched 50 rows in 2.21 seconds

	ACC_NUMBER	ACC_VISITS	FULLNAME	CITY	CC_EXPIRY_DATE
1	91511	9999	TRUE	GLENBURN	28-JAN-19
2	93873	9996	TRUE	LEVERETT	10-FEB-19
3	93294	9994	FALSE	BEDFORD	28-FEB-19
4	92638	9993	TRUE	PITTSFIELD	28-FEB-19
5	90311	9989	FALSE	DUNNELLON	09-APR-19
6	90311	9989	FALSE	DUNNELLON	03-SEP-19
7	90311	9989	FALSE	DUNNELLON	06-JAN-20
8	90311	9989	FALSE	DUNNELLON	15-JUN-20
9	92691	9988	TRUE	STONEWALL	12-OCT-19
10	93139	9974	TRUE	HOUSTON	02-DEC-18
11	93453	9973	TRUE	SHIP BOTTOM	31-MAY-20
12	90160	9970	FALSE	HOUSTON	02-JUL-19
13	90160	9970	FALSE	HOUSTON	14-AUG-19
14	90160	9970	FALSE	HOUSTON	04-OCT-19

SQL HotSpot | 2.235 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			31
PX COORDINATOR			
PX SEND (QC (ORDER))	:TQ10005	3000	31
SORT (GROUP BY)		3000	31
PX RECEIVE		3000	31
PX SEND (RANGE)	:TQ10004	3000	31
HASH (GROUP BY)		3000	31
HASH JOIN		3000	30
Access Predicates			
AD.ADDRESS_NUMBER=CC.ADDRESS_NUMBER			
PX RECEIVE		3000	15
PX SEND (HYBRID HASH)	:TQ10002	3000	15
STATISTICS COLLECTOR			

We have created bitmap index on CREDIT_CARD table as below.

```
CREATE BITMAP INDEX BITMAP_CREDIT_CARD
    ON CREDIT_CARD (ACC_NUMBER, ADDRESS_NUMBER);
```

As we can see, the query execution time reduced to 0.21 seconds whereas query cost has reduced to 21.

Script Output | Autotrace | Query Result

SQL | Fetched 50 rows in 0.21 seconds

	ACC_NUMBER	ACC_VISITS	FULLNAME	CITY	CC_EXPIRY_DATE
1	91511	9999	TRUE	GLENBURN	28-JAN-19
2	93873	9996	TRUE	LEVERETT	10-FEB-19
3	93294	9994	FALSE	BEDFORD	28-FEB-19
4	92638	9993	TRUE	PITTSFIELD	28-FEB-19
5	90311	9989	FALSE	DUNNELLON	09-APR-19
6	90311	9989	FALSE	DUNNELLON	03-SEP-19
7	90311	9989	FALSE	DUNNELLON	06-JAN-20
8	90311	9989	FALSE	DUNNELLON	15-JUN-20
9	92691	9988	TRUE	STONEWALL	12-OCT-19
10	93139	9974	TRUE	HOUSTON	02-DEC-18
11	93453	9973	TRUE	SHIP BOTTOM	31-MAY-20
12	90160	9970	FALSE	HOUSTON	02-JUL-19
13	90160	9970	FALSE	HOUSTON	14-AUG-19
14	90160	9970	FALSE	HOUSTON	04-OCT-19

SQL HotSpot | 1.64 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			21
PX COORDINATOR			
PX SEND (QC (ORDER))	:TQ10005	3000	21
SORT (GROUP BY)		3000	21
PX RECEIVE		3000	21
PX SEND (RANGE)	:TQ10004	3000	21
HASH (GROUP BY)		3000	21
HASH JOIN		3000	20
Access Predicates			
AD.ADDRESS_NUMBER=CC.ADDRESS_NUMBER			
PX RECEIVE		3000	10
PX SEND (HYBRID HASH)	:TQ10002	3000	10
STATISTICS COLLECTOR			
HASH JOIN (BUFFERED)		3000	10
Access Predicates			
UA.ACC_NUMBER=CC.ACC_NUMBER			
PX RECEIVE		3000	4
PX SEND (HYBRID HASH)	:TQ10000	3000	4
STATISTICS COLLECTOR			
PX BLOCK (ITERATOR)		3000	4
TABLE ACCESS (FULL)	USER_ACCOUNT	3000	4
Access Predicates			

Section 3.2: Parallelism

Parallel processing is used to speed up datbase operations. We can apply parallelism to a single query to attain high speed. Parallelism is a natural fit for relational database environment.

Step 1: Run the query

```
select acc_number,acc_password,acc_visits,session_id
from user_account;
```

Script Output | Query Result

SQL | Fetched 50 rows in 0.393 seconds

	ACC_NUMBER	ACC_PASSWORD	ACC_VISITS	SESSION_ID
1	92641	F4EN7	5840	80726
2	92642	DlAUa	6079	80727
3	92643	GN2rD	2769	80728
4	92644	55vkL	9296	80729
5	92645	Oosi4	2710	80730
6	92646	Z4fw6	1380	80731
7	92647	RZGWz	5956	80732

Query Result | Explain Plan

SQL | 0.557 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		5000	15
TABLE ACCESS (FULL)	USER_ACCOUNT	5000	15
Other XML			
{info}			
info type="db_version"			
12.1.0.2			
info type="parse_schema"			

Step 2: Alter the table to use parallelism.

```
ALTER TABLE user_Account PARALLEL (DEGREE 4);
```

Step 3: Run the query again and look at the new execution plan.

Script Output | Query Result

SQL | Fetched 50 rows in 0.093 seconds

	ACC_NUMBER	ACC_PASSWORD	ACC_VISITS	SESSION_ID
1	92641	F4EN7	5840	80726
2	92642	DlAUa	6079	80727
3	92643	GN2rD	2769	80728
4	92644	55vkL	9296	80729
5	92645	Oosi4	2710	80730
6	92646	Z4fw6	1380	80731
7	92647	RZGWz	5956	80732

Script Output | Query Result | Explain Plan

SQL | 2.806 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		5000	4
PX COORDINATOR			
PX SEND (QC (RANDOM))	:TQ10000	5000	4
PX BLOCK (ITERATOR)		5000	4
TABLE ACCESS (FULL)	USER_ACCOUNT	5000	4
Other XML			
{info}			

Part 4: Other Topics

Section 4.1: DBA scripts

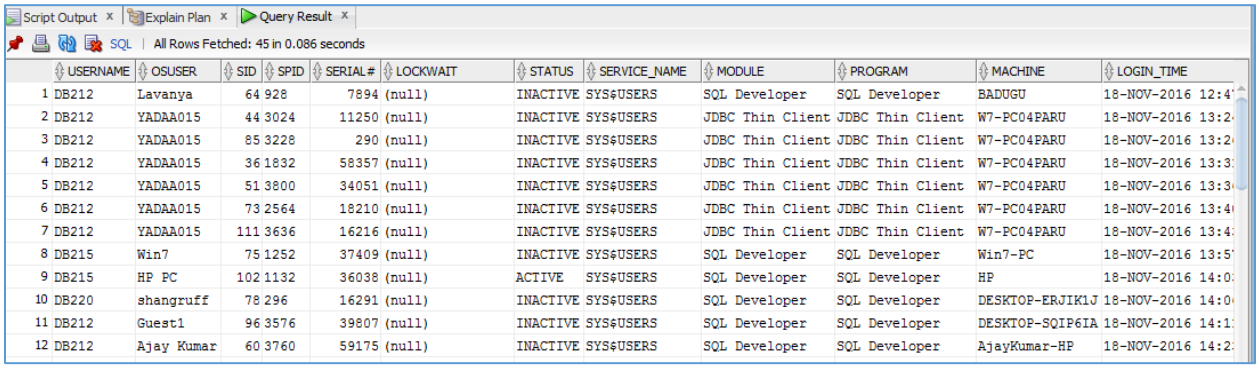
DBA scripts are an excellent ways to query data dictionary in order to better understand what's happening inside the database engine.

Below are few types of scripts an administrator can use in his routine checks.

A. Script 1

The below query will give the active sessions of the database. This is useful to monitor the database and to ensure the security of database.

```
SELECT NVL(s.username, '(oracle)') AS username,
s.osuser, s.sid,
p.spid, s.serial#,
s.lockwait, s.status,
s.service_name, s.module,
s.program,s.machine,
TO_CHAR(s.logon_Time,'DD-MON-YYYY HH24:MI:SS') AS login_time
FROM v$session s,
v$process p
WHERE s.paddr = p.addr
ORDER BY login_time;
```



Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 45 in 0.086 seconds

	USERNAME	OSUSER	SID	SPID	SERIAL#	LOCKWAIT	STATUS	SERVICE_NAME	MODULE	PROGRAM	MACHINE	LOGIN_TIME
1	DB212	Lavanya	64	928	7894	(null)	INACTIVE	SYS$USERS	SQL Developer	SQL Developer	BADUGU	18-NOV-2016 12:4
2	DB212	YADAA015	44	3024	11250	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:2
3	DB212	YADAA015	85	3228	290	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:2
4	DB212	YADAA015	36	1832	58357	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:3
5	DB212	YADAA015	51	3800	34051	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:3
6	DB212	YADAA015	73	2564	18210	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:4
7	DB212	YADAA015	111	3636	16216	(null)	INACTIVE	SYS$USERS	JDBC Thin Client	JDBC Thin Client	W7-PC04PARU	18-NOV-2016 13:4
8	DB215	Win7	75	1252	37409	(null)	INACTIVE	SYS$USERS	SQL Developer	SQL Developer	Win7-PC	18-NOV-2016 13:5
9	DB215	HP PC	102	1132	36038	(null)	ACTIVE	SYS$USERS	SQL Developer	SQL Developer	HP	18-NOV-2016 14:0
10	DB220	shangruff	78	296	16291	(null)	INACTIVE	SYS$USERS	SQL Developer	SQL Developer	DESKTOP-ERJIK1J	18-NOV-2016 14:0
11	DB212	Guest1	96	3576	39807	(null)	INACTIVE	SYS$USERS	SQL Developer	SQL Developer	DESKTOP-SQIP6IA	18-NOV-2016 14:1
12	DB212	Ajay Kumar	60	3760	59175	(null)	INACTIVE	SYS$USERS	SQL Developer	SQL Developer	AjayKumar-HP	18-NOV-2016 14:2

B. Script 2

The below script displays the free space per data file. This is useful for monitoring the datafiles when the storage is limited and this would help assimilate new datafiles.

```
Select df.tablespace_name,
df.file_name,
 df.bytes/1024/1024 total_size,
 nvl(fr.bytes/1024/1024,0) free_space,
 ((df.bytes-nvl(fr.bytes,0))/df.bytes)*100 pct_used
from (select sum(bytes) bytes,
 file_id
 from dba_free_space
 group by file_id) fr,
 dba_data_files df
where df.file_id = fr.file_id(+)
order by 1, df.file_id;
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 16 in 22.938 seconds

	TABLESPACE_NAME	FILE_NAME	TOTAL_SIZE	FREE_SPACE	PCT_USED
1	COLORS	D:\APP\ORACLE\ORADATA\CDB9\COLORS01.DBF	412	329.75	19.96359223300970873786407766990291262136
2	COLORS	D:\APP\ORACLE\ORADATA\CDB9\COLORS02.DBF	806	643	20.22332506203473945409429280397022332506
3	COLORS	D:\APP\ORACLE\ORADATA\CDB9\COLORS03.DBF	337	317.0625	5.916172106824925816023738872403560830 86
4	EXAMPLE	D:\APP\ORACLE\ORADATA\CDB9\EXAMPLE01.DBF	1260.625	41.625	96.698066435299950421417947446703024293 51
5	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS01.DBF	1024	324.3125	68.328857421875
6	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS02.DBF	1024	292.625	71.42333984375
7	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS03.DBF	1024	276.25	73.0224609375
8	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS04.DBF	1024	265.125	74.10888671875
9	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS05.DBF	1024	59.25	94.2138671875
10	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS06.DBF	1024	389.625	61.95068359375
11	STUDENTS	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS07.DBF	1024	444.625	56.57958984375
12	SYSAUX	D:\APP\ORACLE\ORADATA\CDB9\SYSAUX01.DBF	1890	110.625	94.14682539682539682539682539682539682 54
13	SYSTEM	D:\APP\ORACLE\ORADATA\CDB9\SYSTEM01.DBF	900	8.6875	99.03472222222222222222222222222222222 222

C. Script 3

It shows the background proccess that are running currently in database instance.

```
SELECT *
FROM v$bgprocess
WHERE PADDR <> '00'
ORDER BY DESCRIPTION;
```

Script Output x | Explain Plan x | Query Result x

SQL | All Rows Fetched: 22 in 0.141 seconds

	PADDR	PSERIAL#	NAME	DESCRIPTION	ERROR	CON_ID
1	000007FF65A57348	1	AQPC	AQ Process Coord	0	0
2	000007FF65A49108	1	DBRM	DataBase Resource Manager	0	0
3	000007FF65A57E98	2	CJQ0	Job Queue Coordinator	0	0
4	000007FF65A4F6D8	1	LREG	Listener Registration	0	0
5	000007FF65A50D78	1	MMON	Manageability Monitor Process	0	0
6	000007FF65A518C8	1	MMNL	Manageability Monitor Process 2	0	0
7	000007FF65A47A68	1	MMAN	Memory Manager	0	0
8	000007FF65A50228	1	PXMN	PX Monitor	0	0
9	000007FF65A4C998	1	LGWR	Redo etc.	0	0
10	000007FF65A55158	1	SMCO	Space Manager Process	0	0
11	000007FF65A4E038	1	SMON	System Monitor Process	0	0

D. Script 4

This script displays the blocks and data size of users.

```
SELECT OWNER,
SUM(BLOCKS) TOTALBLOCKS,
SUM(BYTES/(1024*1024)) TOTALMB
FROM DBA_SEGMENTS
GROUP BY OWNER
ORDER BY totalblocks;
```

Script Output x | Explain Plan x | Query

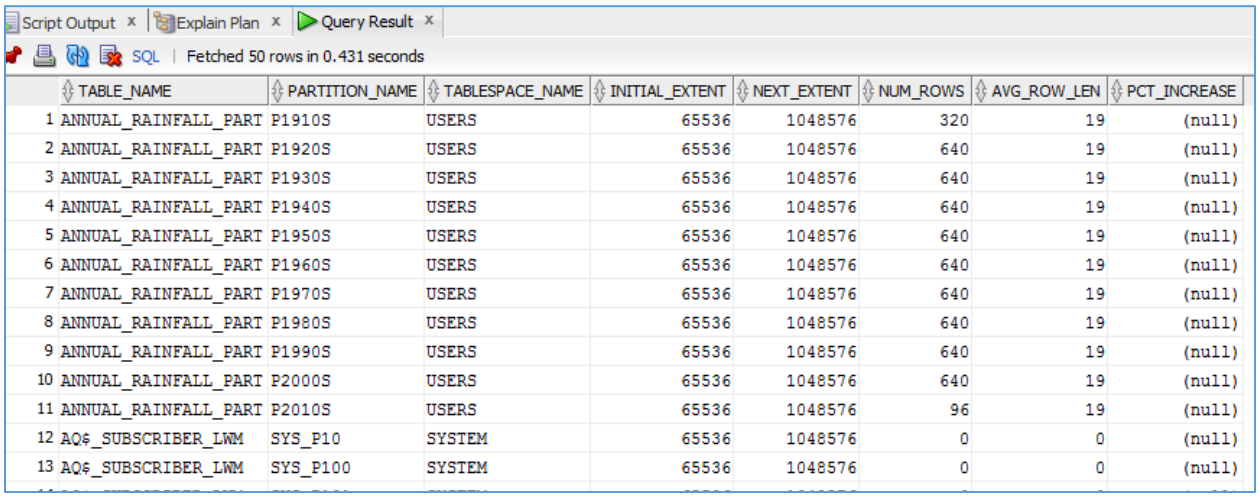
SQL | Fetched 50 rows in 3.875

	OWNER	TOTALBLOCKS	TOTALMB
1	DB143	8	0.0625
2	DB150	8	0.0625
3	DB199	8	0.0625
4	DB167	8	0.0625
5	DB139	8	0.0625
6	MED	8	0.0625
7	DB237	8	0.0625
8	DB136	8	0.0625
9	DB239	8	0.0625
10	DB273	16	0.125
11	DB140	16	0.125
12	DB297	16	0.125

E. Script 5

The below script displays the partitioning information. Partitioning helps in increasing the availability of mission critical databse. Critical tables or indexes are divided into partitions to reduce maintenance windows or recovery times or failures.

```
SELECT p.table_name,
p.partition_name,
p.tablespace_name,
p.initial_extent,
p.next_extent,
p.num_rows,
p.avg_row_len,
p.pct_increase
FROM dba_tab_partitions p
ORDER BY p.table_name, p.partition_name;
```


Script Output | Explain Plan | Query Result

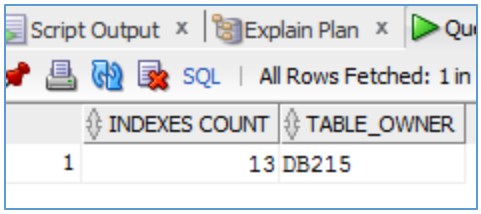
SQL | Fetched 50 rows in 0.431 seconds

	TABLE_NAME	PARTITION_NAME	TABLESPACE_NAME	INITIAL_EXTENT	NEXT_EXTENT	NUM_ROWS	AVG_ROW_LEN	PCT_INCREASE
1	ANNUAL_RAINFALL_PART	P1910S	USERS	65536	1048576	320	19	(null)
2	ANNUAL_RAINFALL_PART	P1920S	USERS	65536	1048576	640	19	(null)
3	ANNUAL_RAINFALL_PART	P1930S	USERS	65536	1048576	640	19	(null)
4	ANNUAL_RAINFALL_PART	P1940S	USERS	65536	1048576	640	19	(null)
5	ANNUAL_RAINFALL_PART	P1950S	USERS	65536	1048576	640	19	(null)
6	ANNUAL_RAINFALL_PART	P1960S	USERS	65536	1048576	640	19	(null)
7	ANNUAL_RAINFALL_PART	P1970S	USERS	65536	1048576	640	19	(null)
8	ANNUAL_RAINFALL_PART	P1980S	USERS	65536	1048576	640	19	(null)
9	ANNUAL_RAINFALL_PART	P1990S	USERS	65536	1048576	640	19	(null)
10	ANNUAL_RAINFALL_PART	P2000S	USERS	65536	1048576	640	19	(null)
11	ANNUAL_RAINFALL_PART	P2010S	USERS	65536	1048576	96	19	(null)
12	AQ$_SUBSCRIBER_LWM	SYS_P10	SYSTEM	65536	1048576	0	0	(null)
13	AQ$_SUBSCRIBER_LWM	SYS_P100	SYSTEM	65536	1048576	0	0	(null)

F. Script 6

This query displays the count fro the number of indexes for a particular user.

```
SELECT COUNT(INDEX_NAME) AS "INDEXES COUNT",
TABLE_OWNER
FROM DBA_INDEXES
GROUP BY TABLE_OWNER
HAVING TABLE_OWNER LIKE '%DB215%';
```

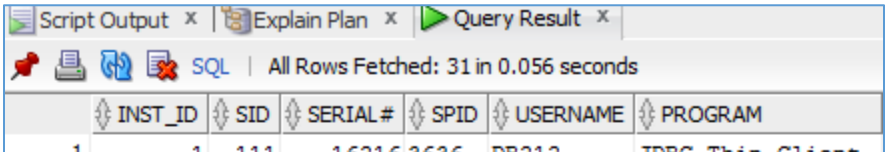


	INDEXES COUNT	TABLE_OWNER
1	13	DB215

G. Script 7

This script is to kill session. Before killing the session we have to identify the session that needs to be killed, if we kill the wrong session this can be very destructive. If a session belonging to a background process is killed this would cause an instance crash. To identify the session offending:

```
SELECT s.inst_id,
s.sid,
s.serial#,
p.spid,
s.username,
s.program
FROM   gv$session s
JOIN gv$process p ON p.addr = s.paddr AND p.inst_id = s.inst_id
WHERE  s.type != 'BACKGROUND';
```



	INST_ID	SID	SERIAL#	SPID	USERNAME	PROGRAM
1	1	111	16216	3636	DB212	JDBC Thin Client
2	1	68	22731	2708	DB215	ORACLE.EXE (P000)
3	1	70	60597	2712	DB215	ORACLE.EXE (P001)
4	1	38	59478	2716	DB215	ORACLE.EXE (P002)
5	1	105	56937	2720	DB215	ORACLE.EXE (P003)
6	1	102	36038	1132	DB215	SQL Developer
7	1	75	37409	1252	DB215	SQL Developer
8	1	60	59175	3760	DB212	SQL Developer
9	1	64	7894	928	DB212	SQL Developer
10	1	69	34281	1992	DB212	JDBC Thin Client
11	1	44	11250	3024	DB212	JDBC Thin Client
12	1	85	290	3228	DB212	JDBC Thin Client
13	1	73	18210	2564	DB212	JDBC Thin Client

Section 4.2: Database Security

Data security will be applied to ensure that no unauthorized user can edit any critical data from the USER ACCOUNT, PAYMENT, INVENTORY ITEM, ORDER ITEM, ORDER DETAILS, SHIPPING, INVOICE, CREDIT CARD, SHOPPING CART, ADDRESS and USER SESSION tables.

Roles will be assigned to all the players, security will be given to the users based on their roles. Sensitive and confidential data has to be encrypted.

Encryption helps in preventing unauthorized access to both static as well as dynamic data even if the data is copied, backed up or stored offline.

Data access is controlled rather than than database access.

Data redaction strategies are another useful way of controlling access to data. It masks data at query time, so that the rules are aplied on the data being progressed.

User authentication is a way to control database access in Oracle DBMS and other database systems. The three components are: users that must authenticate to access the database, system and object privileges to further control access and finally roles that can be used to organize groups of privileges.

Privileges that are only necessary are given to other users.

Without roles, granting privileges directly to individual and separate users would create a confusing and unwieldy network of access rights.

A person with admin_role as 'Administrator' is only authorized to update all the information from all the tables.

Granting authentication for roles:

Grant All on USER ACCOUNT for 'Administrator';

Grant All on PAYMENT for 'Administrator';

Grant All on INVENTORY ITEM for 'Administrator';

Grant All on ORDER ITEM for 'Administrator';

Grant All on ORDER DETAILS for 'Administrator';

Grant All on SHIPPING for 'Administrator';

Grant All on INVOICE for 'Administrator';

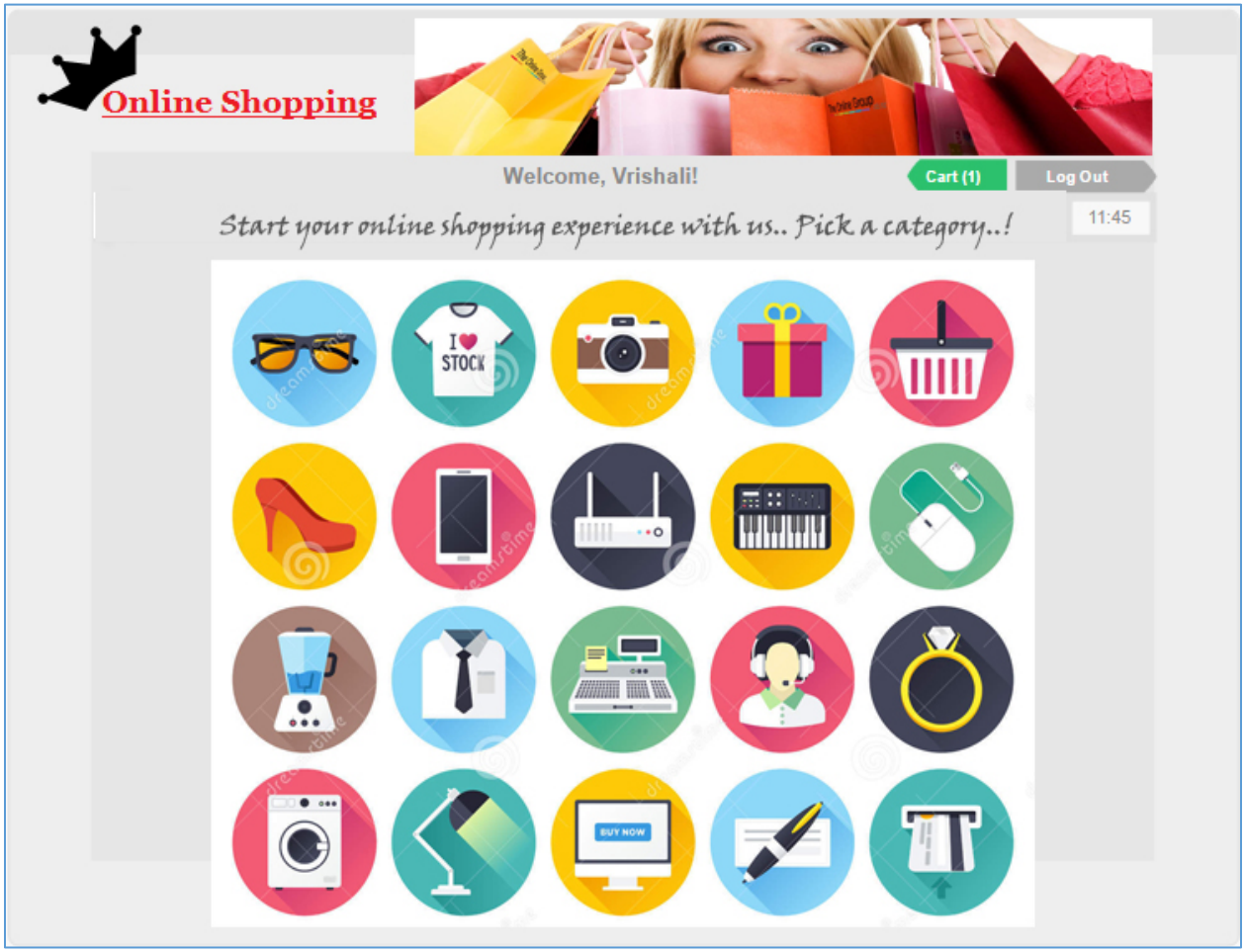
Grant All on CREDIT CARD for 'Administrator';

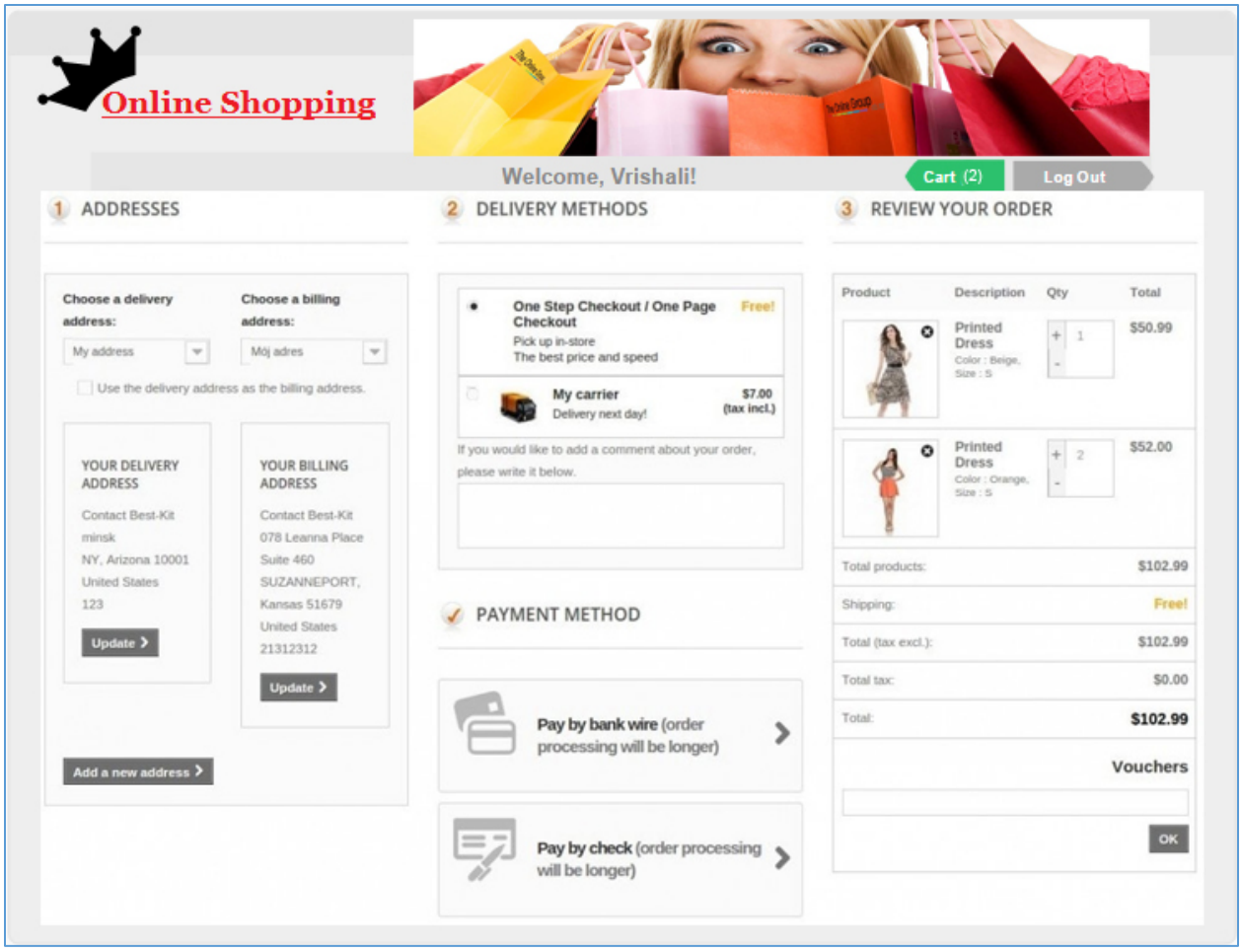
Grant All on SHOPPING CART for 'Administrator';

Grant All on ADDRESS for 'Administrator';

Grant All on USER SESSION for 'Administrator'.

Section 4.3: Interface Design




Online Shopping
Welcome, Vrishali!
Cart (2)
Log Out
1 ADDRESSES
Choose a delivery address:
My address
Choose a billing address:
Mój adres
Use the delivery address as the billing address.
YOUR DELIVERY ADDRESS
Contact Best-Kit
minsk
NY, Arizona 10001
United States
123
Update
YOUR BILLING ADDRESS
Contact Best-Kit
078 Leanna Place
Suite 460
SUZANNEPORT,
Kansas 51679
United States
21312312
Update
Add a new address
2 DELIVERY METHODS
One Step Checkout / One Page Checkout
Free!
Pick up in-store
The best price and speed
My carrier
$7.00 (tax incl.)
Delivery next day!
If you would like to add a comment about your order, please write it below.
PAYMENT METHOD
Pay by bank wire (order processing will be longer)
Pay by check (order processing will be longer)
3 REVIEW YOUR ORDER
Product
Description
Qty
Total
Printed Dress
Color : Beige, Size : S
1
$50.99
Printed Dress
Color : Orange, Size : S
2
$52.00
Total products:
$102.99
Shipping:
Free!
Total (tax excl.):
$102.99
Total tax:
$0.00
Total:
$102.99
Vouchers
OK