ISM6218 - ADVANCED DATABASE MANAGEMENT SYSTEMS
GROUP PROJECT

Group 12
GLOBAL RESTAURANT SEARCH SYSTEM

GROUP MEMBERS -

1. Sachin Kant Misra
2. Prashant Bhowmik
3. Harshita Srivastava
4. Jagpreet Singh Sethi
5. Nitesh Dubey

Table of Contents

1 ABSTRACT 3

2 LOGICAL DATABASE DESIGN 4

2.1 Assumption 4
2.2 TABLE USER_DATA 5
2.3 TABLE RESTAURANT 7
2.4 TABLE CUISINE 8
2.5 TABLE MENU_ITEMS 9
2.6 TABLE AMBIENCE 10
2.7 TABLE LOCATION 11
2.8 TABLE RATING 12

3 PHYSICAL DATABASE DESIGN 13

3.1 INDEXING STRATEGIES 14
3.2 CAPACITY PLANNING 14
3.3 ARCHITECTURAL ISSUES: 15
3.4 ISSUES WITH DISTRIBUTED DATABASE 16

4 DATA GENERATION AND LOADING - 17

4.1 DATA LOADING 17
Ambience Table Structure 17
Cuisine Table Structure 17
Location Table Structure 18
Menu_Items Table Structure 18
Rating Table Structure 18
Restaurant Table Structure 19
User_Data Table Structure 20
Issues faced during Data loading 24

5 PERFORMANCE TUNING 26

5.1 SIMPLE INDEXING 26
Query 1 26
Query 2 28
Query 3 29
Extracting Restaurant_ID using IN operator with Index 30
5.2 FUNCTION BASED INDEXING 30
5.3 MATERIALIZED VIEW 32
5.4 SELECTIVITY 32
5.5 DATABASE PARTITIONING 34

6 QUERYING 39

7 DBA Scripts 43

8 Database Security Policy 54

9 Online Transaction Processing 55

1 ABSTRACT

Searching for the best cuisine in the town has always been a challenge especially when you visit a foreign land. We plan to make a global restaurant search system for the food lovers so that they can explore new restaurants, bars, night clubs etc based on cuisine type, location, budget, delivery and other operations. The motivation behind such a system is to have the best option to search for and discover great places to eat.

We plan to make a system wherein restaurant search will be based on -

1) **Location:** Details such as country, city and area code can be used to search restaurant in a particular area.

2) **Rating and Review**: Users can search the restaurant based on the review & ratings - 5 being the best rated and 1 being the least.

3) **Budget**: Estimated amount for 2 people when they eat at a restaurant. This will give an idea on how expensive is the restaurant relatively and provides a filter to the customers on the basis of budget.

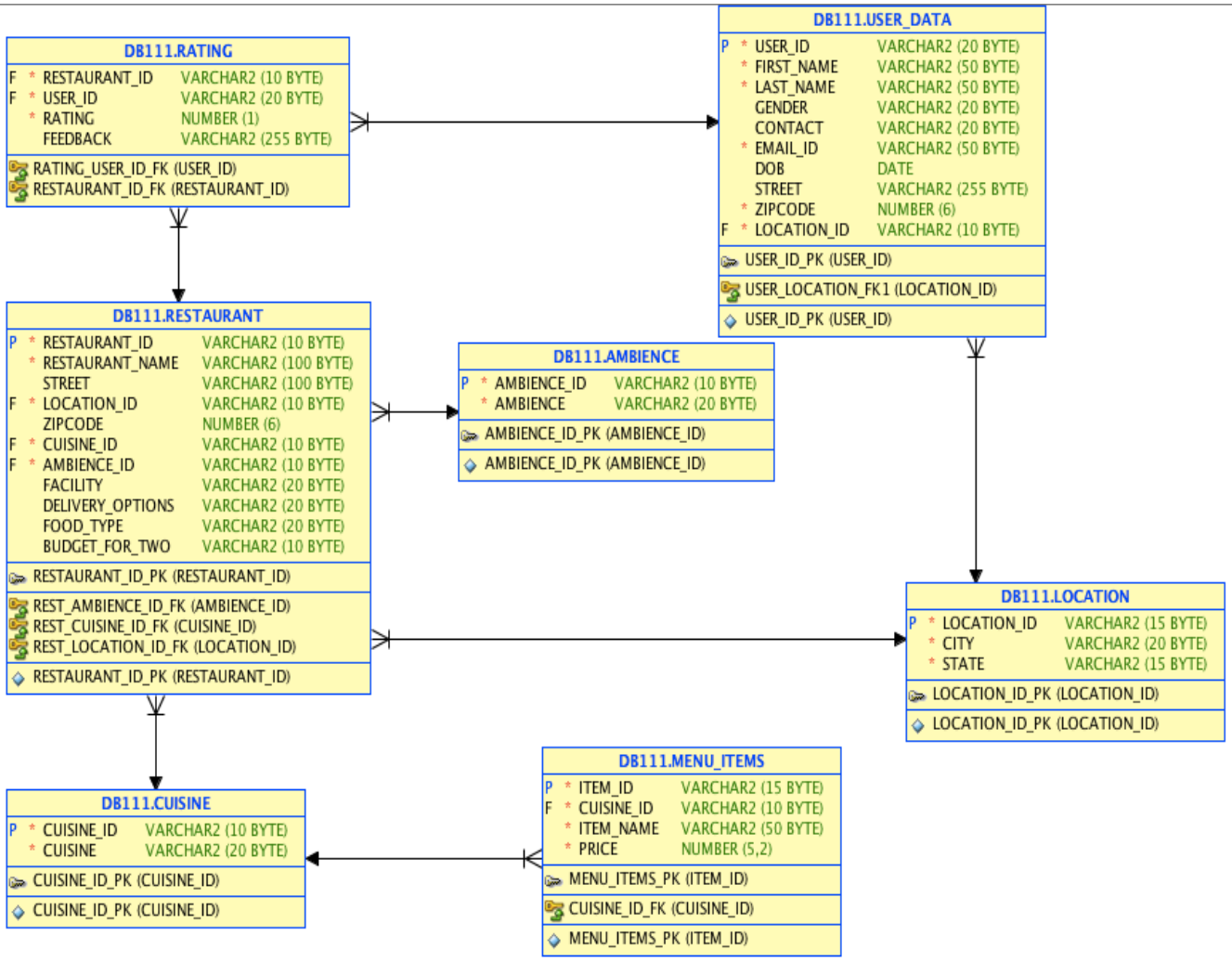
4) **Cuisine**: Each restaurant will be categorized and subcategorized based on the cuisine type – Chinese, Mexican, North Indian, American, Italian, Chinese etc

5) **Features**: Each restaurant will be classified on features such as night-life, discotheque, live music, Outdoor seating etc which may be available at the particular eating joint.

Our system provides a next level of user satisfaction and friendliness as it is completely customer centric and they may select a place to eat based on their preferences and mood without ever going wrong about the place!

2 LOGICAL DATABASE DESIGN

Entity-relationship diagrams (ERDs) for our system is as below.



2.1 Assumption

1. We have assumed that the customer will search the restaurant using our website, select the restaurant and then the request will be redirected to the restaurant.
2. The customer can view the menu, menu sections and the menu items and place the order. But the payment will be taken care of by the individual restaurants.

3. The customers will be able to filter the restaurant based on its ambience like pub, family restaurant, romantic candle night dinner etc
4. The user will be able to provide ratings and reviews for the restaurants listed in our system.
5. Each restaurant has a variety of options for cuisines offered. The menu items are segregated as per the cuisine type.

2.2 TABLE USER_DATA

The table user_data consists of data for the restaurant users. It contains fields such as user_id, first name, last name, gender, contact, email_id, date of birth and street etc.

DDL for Table USER_DATA

```
CREATE TABLE "DB111"."USER_DATA"
(    "USER_ID" VARCHAR2(20 BYTE),
     "FIRST_NAME" VARCHAR2(50 BYTE),
     "LAST_NAME" VARCHAR2(50 BYTE),
     "GENDER" VARCHAR2(20 BYTE),
     "CONTACT" VARCHAR2(20 BYTE),
     "EMAIL_ID" VARCHAR2(50 BYTE),
     "DOB" DATE,
     "STREET" VARCHAR2(255 BYTE),
     "ZIPCODE" NUMBER(6,0),
     "LOCATION_ID" VARCHAR2(10 BYTE)
 ) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

DDL for Index USER_ID_PK

```
CREATE UNIQUE INDEX "DB111"."USER_ID_PK" ON "DB111"."USER_DATA" ("USER_ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Constraints for Table USER_DATA

```
ALTER TABLE "DB111"."USER_DATA" MODIFY ("USER_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" MODIFY ("FIRST_NAME" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" MODIFY ("LAST_NAME" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" MODIFY ("EMAIL_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" MODIFY ("LOCATION_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" MODIFY ("ZIPCODE" NOT NULL ENABLE);
ALTER TABLE "DB111"."USER_DATA" ADD CONSTRAINT "USER_ID_PK" PRIMARY KEY
("USER_ID") USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Ref Constraints for Table USER_DATA

```
ALTER TABLE "DB111"."USER_DATA" ADD CONSTRAINT "USER_LOCATION_FK1" FOREIGN
KEY ("LOCATION_ID") REFERENCES "DB111"."LOCATION" ("LOCATION_ID") ENABLE;
```

2.3 TABLE RESTAURANT

This table contains all the information about the restaurants such as Restaurant Name, address, zip code etc

DDL for Table RESTAURANT

```
CREATE TABLE "DB111"."RESTAURANT"
  (       "RESTAURANT_ID" VARCHAR2(10 BYTE),
          "RESTAURANT_NAME" VARCHAR2(100 BYTE),
          "STREET" VARCHAR2(100 BYTE),
          "LOCATION_ID" VARCHAR2(10 BYTE),
          "ZIPCODE" NUMBER(6,0),
          "CUISINE_ID" VARCHAR2(10 BYTE),
          "AMBIENCE_ID" VARCHAR2(10 BYTE),
          "FACILITY" VARCHAR2(20 BYTE),
          "DELIVERY_OPTIONS" VARCHAR2(20 BYTE),
          "FOOD_TYPE" VARCHAR2(20 BYTE),
          "BUDGET_FOR_TWO" VARCHAR2(10 BYTE)
  ) SEGMENT CREATION IMMEDIATE
 PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE "STUDENTS" ;
```

DDL for Index RESTAURANT_ID_PK

```
  CREATE UNIQUE INDEX "DB111"."RESTAURANT_ID_PK" ON "DB111"."RESTAURANT" ("RES-
TAURANT_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS" ;
```

Constraints for Table RESTAURANT

```
ALTER TABLE "DB111"."RESTAURANT" MODIFY ("RESTAURANT_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RESTAURANT" MODIFY ("RESTAURANT_NAME" NOT NULL ENABLE);
ALTER TABLE "DB111"."RESTAURANT" MODIFY ("LOCATION_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RESTAURANT" MODIFY ("CUISINE_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RESTAURANT" MODIFY ("AMBIENCE_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RESTAURANT" ADD CONSTRAINT "RESTAURANT_ID_PK" PRIMARY
KEY ("RESTAURANT_ID")
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE "STUDENTS"  ENABLE;
```

Ref Constraints for Table RESTAURANT

```
ALTER TABLE "DB111"."RESTAURANT" ADD CONSTRAINT "REST_AMBIENCE_ID_FK" FOREIGN
KEY ("AMBIENCE_ID") REFERENCES "DB111"."AMBIENCE" ("AMBIENCE_ID") ENABLE;
ALTER TABLE "DB111"."RESTAURANT" ADD CONSTRAINT "REST_CUISINE_ID_FK" FOREIGN
KEY ("CUISINE_ID") REFERENCES "DB111"."CUISINE" ("CUISINE_ID") ENABLE;
ALTER TABLE "DB111"."RESTAURANT" ADD CONSTRAINT "REST_LOCATION_ID_FK" FOREIGN
KEY ("LOCATION_ID") REFERENCES "DB111"."LOCATION" ("LOCATION_ID") ENABLE;
```

2.4 TABLE CUISINE

This table contains the information about the type of cuisines available in the various restaurants e.g. Chinese, American etc.

DDL for Table CUISINE

```
CREATE TABLE "DB111"."CUISINE"
 (      "CUISINE_ID" VARCHAR2(10 BYTE),
        "CUISINE" VARCHAR2(20 BYTE)
 ) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE "STUDENTS" ;
```

DDL for Index CUISINE_ID_PK

```
CREATE UNIQUE INDEX "DB111"."CUISINE_ID_PK" ON "DB111"."CUISINE" ("CUISINE_ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Constraints for Table CUISINE

```
ALTER TABLE "DB111"."CUISINE" MODIFY ("CUISINE_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."CUISINE" MODIFY ("CUISINE" NOT NULL ENABLE);
ALTER TABLE "DB111"."CUISINE" ADD CONSTRAINT "CUISINE_ID_PK" PRIMARY KEY ("CUI-
SINE_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

2.5 TABLE MENU_ITEMS

This table contains the different options in the menu of the restaurant. It contains fields such as Item name, price etc

DDL for Table MENU_ITEMS

```
CREATE TABLE "DB111"."MENU_ITEMS"
 (      "ITEM_ID" VARCHAR2(255 BYTE),
        "CUISINE_ID" VARCHAR2(255 BYTE),
        "ITEM_NAME" VARCHAR2(255 BYTE),
        "PRICE" NUMBER(5,2)
 ) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

DDL for Index MENU_ITEMS_PK

```
CREATE UNIQUE INDEX "DB111"."MENU_ITEMS_PK" ON "DB111"."MENU_ITEMS" ("ITEM_ID")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

Constraints for Table MENU_ITEMS

```
ALTER TABLE "DB111"."MENU_ITEMS" MODIFY ("ITEM_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."MENU_ITEMS" MODIFY ("CUISINE_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."MENU_ITEMS" MODIFY ("ITEM_NAME" NOT NULL ENABLE);
ALTER TABLE "DB111"."MENU_ITEMS" MODIFY ("PRICE" NOT NULL ENABLE);
ALTER TABLE "DB111"."MENU_ITEMS" ADD CONSTRAINT "MENU_ITEMS_PK" PRIMARY KEY
("ITEM_ID")
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS"  ENABLE;
```

Ref Constraints for Table MENU_ITEMS

```
ALTER TABLE "DB111"."MENU_ITEMS" ADD CONSTRAINT "CUISINE_ID_FK" FOREIGN KEY
("CUISINE_ID")
        REFERENCES "DB111"."CUISINE" ("CUISINE_ID") ENABLE;
```

2.6 TABLE AMBIENCE

This tables consists of the values for the ambience of the restaurant based on which the restaurant can be searched on the global restaurant search system.

DDL for Table AMBIENCE

```
CREATE TABLE "DB111"."AMBIENCE"
(       "AMBIENCE_ID" VARCHAR2(10 BYTE),
        "AMBIENCE" VARCHAR2(20 BYTE)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE "STUDENTS" ;
```

DDL for Index AMBIENCE_ID_PK

```
  CREATE UNIQUE INDEX "DB111"."AMBIENCE_ID_PK" ON "DB111"."AMBIENCE" ("AMBI-
ENCE_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS" ;
```

Constraints for Table AMBIENCE

```
  ALTER TABLE "DB111"."AMBIENCE" MODIFY ("AMBIENCE_ID" NOT NULL ENABLE);
  ALTER TABLE "DB111"."AMBIENCE" MODIFY ("AMBIENCE" NOT NULL ENABLE);
  ALTER TABLE "DB111"."AMBIENCE" ADD CONSTRAINT "AMBIENCE_ID_PK" PRIMARY KEY
("AMBIENCE_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS"  ENABLE;
```

2.7 TABLE LOCATION

Description: Location data structure contains information about State and City. We have taken sample data from three States – New York, California and Florida spread across 23 Cities.

DDL for Table LOCATION

```
  CREATE TABLE "DB111"."LOCATION"
   (      "LOCATION_ID" VARCHAR2(15 BYTE),
          "CITY" VARCHAR2(20 BYTE),
          "STATE" VARCHAR2(15 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS" ;
```

DDL for Index LOCATION_ID_PK

```
  CREATE UNIQUE INDEX "DB111"."LOCATION_ID_PK" ON "DB111"."LOCATION" ("LOCA-
TION_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS" ;
```

Constraints for Table LOCATION

```
  ALTER TABLE "DB111"."LOCATION" MODIFY ("LOCATION_ID" NOT NULL ENABLE);
  ALTER TABLE "DB111"."LOCATION" MODIFY ("CITY" NOT NULL ENABLE);
  ALTER TABLE "DB111"."LOCATION" MODIFY ("STATE" NOT NULL ENABLE);
  ALTER TABLE "DB111"."LOCATION" ADD CONSTRAINT "LOCATION_ID_PK" PRIMARY KEY
("LOCATION_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS"  ENABLE;
```

2.8 TABLE RATING

Description: Rating Data structure contains information about restaurant rating and it is mapped with User_ID. Feedback data_type will have customer feedback for the restaurant.

DDL for Table RATING

```
  CREATE TABLE "DB111"."RATING"
   (      "RESTAURANT_ID" VARCHAR2(10 BYTE),
          "USER_ID" VARCHAR2(20 BYTE),
          "RATING" NUMBER(2,0),
          "FEEDBACK" VARCHAR2(255 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "STUDENTS" ;
```

Constraints for Table RATING

```
ALTER TABLE "DB111"."RATING" MODIFY ("RESTAURANT_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RATING" MODIFY ("USER_ID" NOT NULL ENABLE);
ALTER TABLE "DB111"."RATING" MODIFY ("RATING" NOT NULL ENABLE);
```

Ref Constraints for Table RATING

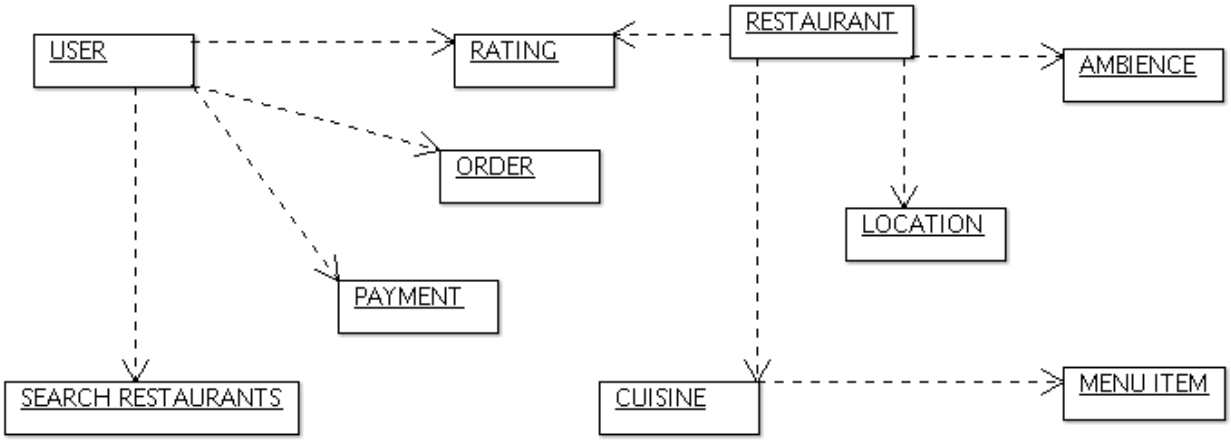
```
ALTER TABLE "DB111"."RATING" ADD CONSTRAINT "RATING_USER_ID_FK" FOREIGN KEY
("USER_ID")
        REFERENCES "DB111"."USER_DATA" ("USER_ID") ENABLE;
ALTER TABLE "DB111"."RATING" ADD CONSTRAINT "RESTAURANT_ID_FK" FOREIGN KEY
("RESTAURANT_ID")
        REFERENCES "DB111"."RESTAURANT" ("RESTAURANT_ID") ENABLE;
```

3 PHYSICAL DATABASE DESIGN

The objective of physical database design is to use the knowledge gained from analyzing of the database system to improve the efficiency of the overall system. The efficiency of the database is determined by the requirements of the users and the database design. There are many different tools that the database designer may use when implementing the physical database.

Indexing increases the speed of accessing the data by storing special data structures which can be processed very quickly. Caching uses part of the computer's main memory to store part of the database. Main memory is very quick to access. De-normalization is a process of changing the structure of the relations to improve the performance of queries and updates.

When a database relation becomes very large, searching for information in it can be a very slow process. Database management systems allow users to create indexes to speed up certain types of query.



3.1 INDEXING STRATEGIES

- Index primary keys - Primary keys are unique and are often used to access individual tuples in a relation. Primary keys are used to join two relations.
- Index foreign keys - Foreign keys are used to join two relations together. An index will improve the speed of joins between relations.
- Sorting data is a very slow process. Indexes are sorted when they are built. Sorting data using an index simply means reading the index.
- Do not index attributes with few values
- Indexes work best when accessing individual tuples. Attributes with few values will have many duplicates.
- Do not index every attribute. Indexes are stored on disc. Unnecessary indexes waste space.

3.2 CAPACITY PLANNING

Capacity planning acknowledges that the business requirements on the system may increase, and forecasts how much resource must be added to the database system to ensure that the user experience continues uninterrupted.

Typically, the resources you'll add may be CPU power, memory, storage, or network capacity, or more likely a combination of these, depending on how demands are predicted to grow. This

makes capacity planning different to performance tuning, because the latter is a reactive process. You tweak the system's parameters to ensure that it does not dip below pre-set levels based on current performance requirements. You plan capacity to ensure that it does not fall below performance levels in the future.

To properly perform capacity planning a cooperative effort must be undertaken between the system administrators, database administrators and network administrators.

Step 1: Size the Oracle database

Step 2: Determine Number and Type of Users

Step 3: Determine Hardware Requirements to Meet Required Response Times and Support User Load

Step 4: Determine Backup Hardware to Support Required Uptime Requirements

The DBA_TEMP_FREE_SPACE dictionary view contains information about space usage for each temporary tablespace. The information includes the space allocated and the free space. You can query this view for these statistics using the following command.

3.3 ARCHITECTURAL ISSUES:

Architecture Bottlenecks mainly consists of scaling bottlenecks and they are formed due to the following two issues:

- **Centralized component:** A component in application architecture which cannot be scaled out adds an upper limit on the number of requests that entire architecture or request pipeline can handle.
- **High latency component:** A slow component in request pipeline puts lower limit on the response time of the application. Usual solution to fix this issue is to make high latency components into background jobs or executing them asynchronously with queuing.

3.4 ISSUES WITH DISTRIBUTED DATABASE

- **Complexity** — DBAs may have to do extra work to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- **Security** — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (for example, by encrypting the network links between remote sites).
- **Difficult to maintain integrity** — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- **Inexperience** — distributed databases are difficult to work with, and in such a young field there is not much readily available experience in "proper" practice.
- **Lack of standards** — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
- **Database design more complex** — In addition to traditional database design challenges, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.

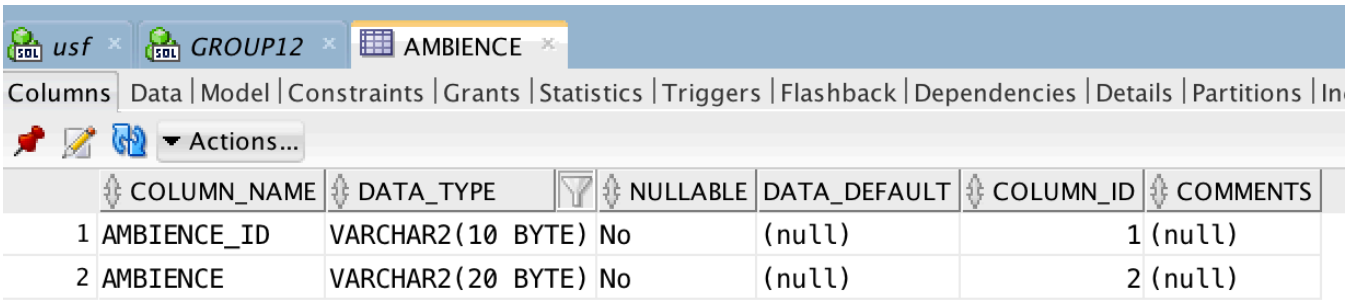
4 DATA GENERATION AND LOADING

As part of this project, we have created all the data and uploaded it to Group 12 created in the database. We have used a real time data for our GLOBAL RESTAURANT SEARCH SYSTEM tailored according to the requirement of our system. The data was identified and cleaned as per the requirements of the system.

4.1 DATA LOADING

The structure of our seven tables User, Cuisine, Location, Ambience, Rating, menu_items and restaurant are as under -

Ambience Table Structure


usf | GROUP12 | AMBIENCE

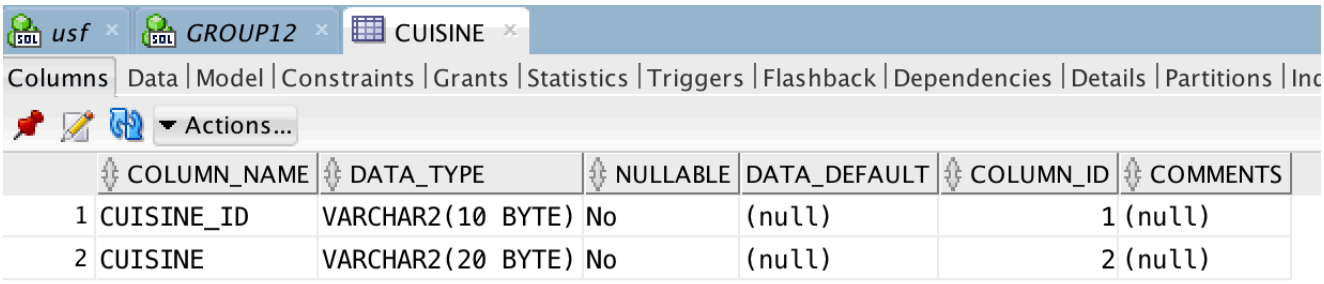
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | In

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	AMBIENCE_ID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	AMBIENCE	VARCHAR2(20 BYTE)	No	(null)	2	(null)

Description: Ambience data structure contains information about Restaurant Ambience like Bar, Night Life, Lounge, Outdoor sitting and live music.

Cuisine Table Structure


usf | GROUP12 | CUISINE

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

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CUISINE_ID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	CUISINE	VARCHAR2(20 BYTE)	No	(null)	2	(null)

Description: Cuisine Data Structure contains cuisine information like Thai, American, Indian etc and it is mapped with Restaurant ID and Menu ID

Location Table Structure

usf | GROUP12 | LOCATION

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	LOCATION_ID	VARCHAR2(15 BYTE)	No	(null)	1	(null)
2	CITY	VARCHAR2(20 BYTE)	No	(null)	2	(null)
3	STATE	VARCHAR2(15 BYTE)	No	(null)	3	(null)

Description: Location data structure contains information about State and City. We have taken sample data from three States – New York, California and Florida spread across 23 Cities.

Menu_Items Table Structure

usf | GROUP12 | MENU_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	ITEM_ID	VARCHAR2(15 BYTE)	No	(null)	1	(null)
2	CUISINE_ID	VARCHAR2(10 BYTE)	No	(null)	2	(null)
3	ITEM_NAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	PRICE	NUMBER(5,2)	No	(null)	4	(null)

Description: Menu_Items data structure contains information about Item Name and Price

Rating Table Structure

usf | GROUP12 | RATING

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	RESTAURANT_ID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	USER_ID	VARCHAR2(20 BYTE)	No	(null)	2	(null)
3	RATING	NUMBER(1,0)	No	(null)	3	(null)
4	FEEDBACK	VARCHAR2(255 BYTE)	Yes	(null)	4	(null)

Description: Rating Data structure contains information about restaurant rating and it is mapped with User_ID. Feedback data_type will have customer feedback for the restaurant.

Restaurant Table Structure

usf | GROUP12 | RESTAURANT

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	RESTAURANT_ID	VARCHAR2(10 BYTE)	No	(null)	1	(null)
2	RESTAURANT_NAME	VARCHAR2(100 BYTE)	No	(null)	2	(null)
3	STREET	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)
4	LOCATION_ID	VARCHAR2(10 BYTE)	No	(null)	4	(null)
5	ZIPCODE	NUMBER(6,0)	Yes	(null)	5	(null)
6	CUISINE_ID	VARCHAR2(10 BYTE)	No	(null)	6	(null)
7	AMBIENCE_ID	VARCHAR2(10 BYTE)	No	(null)	7	(null)
8	FACILITY	VARCHAR2(20 BYTE)	Yes	(null)	8	(null)
9	DELIVERY_OPTIONS	VARCHAR2(20 BYTE)	Yes	(null)	9	(null)
10	FOOD_TYPE	VARCHAR2(20 BYTE)	Yes	(null)	10	(null)
11	BUDGET_FOR_TWO	VARCHAR2(10 BYTE)	Yes	(null)	11	(null)

Description: Restaurant data table contains information about restaurant details like ambience, facilities, food_type, delivery options etc.

User_Data Table Structure

usf | GROUP12 | USER_DATA

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(20 BYTE)	No	(null)	1	(null)
2	FIRST_NAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	LAST_NAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	GENDER	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
5	CONTACT	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)
6	EMAIL_ID	VARCHAR2(50 BYTE)	No	(null)	6	(null)
7	DOB	DATE	Yes	(null)	7	(null)
8	STREET	VARCHAR2(255 BYTE)	Yes	(null)	8	(null)
9	ZIPCODE	NUMBER(6,0)	No	(null)	9	(null)
10	LOCATION_ID	VARCHAR2(10 BYTE)	No	(null)	10	(null)

Description: User_data table structure contains information about user's first name, last name, gender, contact number etc.

Number Of Records In The Tables

The user_data table contains 8214 rows.

```
SELECT COUNT(*) FROM USER_DATA;
```

Script Output | Explain Plan | Query

SQL | All Rows Fetched: 1 in 0.05

	COUNT(*)
1	8214

The ambience table contains 11 rows.

```
SELECT COUNT(*) FROM AMBIENCE;
```

Query Result

SQL | All Rows Fetched: 1 in 0.(

	COUNT(*)
1	11

The CUISINE table contains 11 rows.

```
SELECT COUNT(*) FROM CUISINE;
```

Query Result

SQL | All Rows Fetched: 1 in (

	COUNT(*)
1	11

The location table contains 23 rows.

```
SELECT COUNT(*) FROM LOCATION;
```

Query Result

SQL | All Rows Fetched: 1 in 0.0

	COUNT(*)
1	23

The MENU_ITEMS table contains 91 rows.

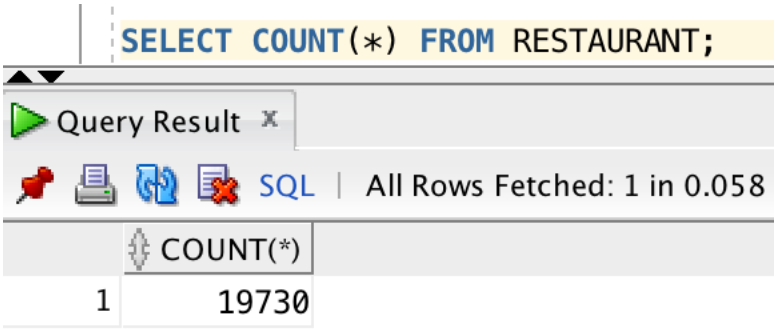
```
SELECT COUNT(*) FROM MENU_ITEMS;
```

Query Result

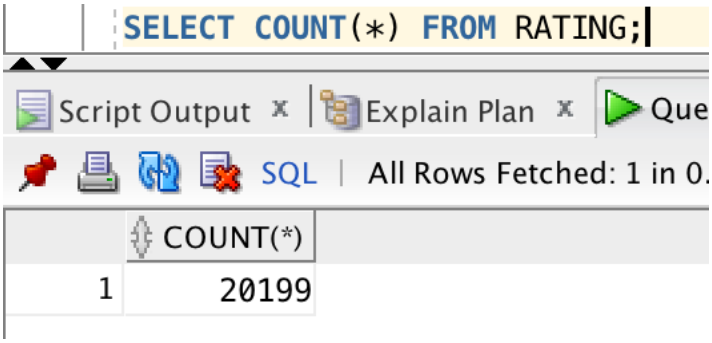
SQL | All Rows Fetched: 1 in 0.054 s

	COUNT(*)
1	91

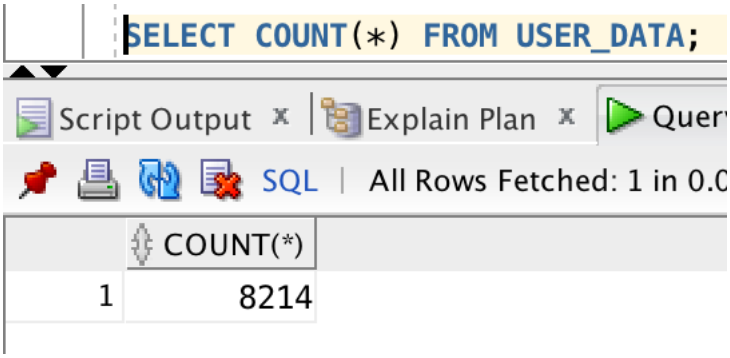
The RESTAURANT table contains 19730 rows.



The RATING table contains 20199 rows.



The USER_DATA table contains 8214 rows.

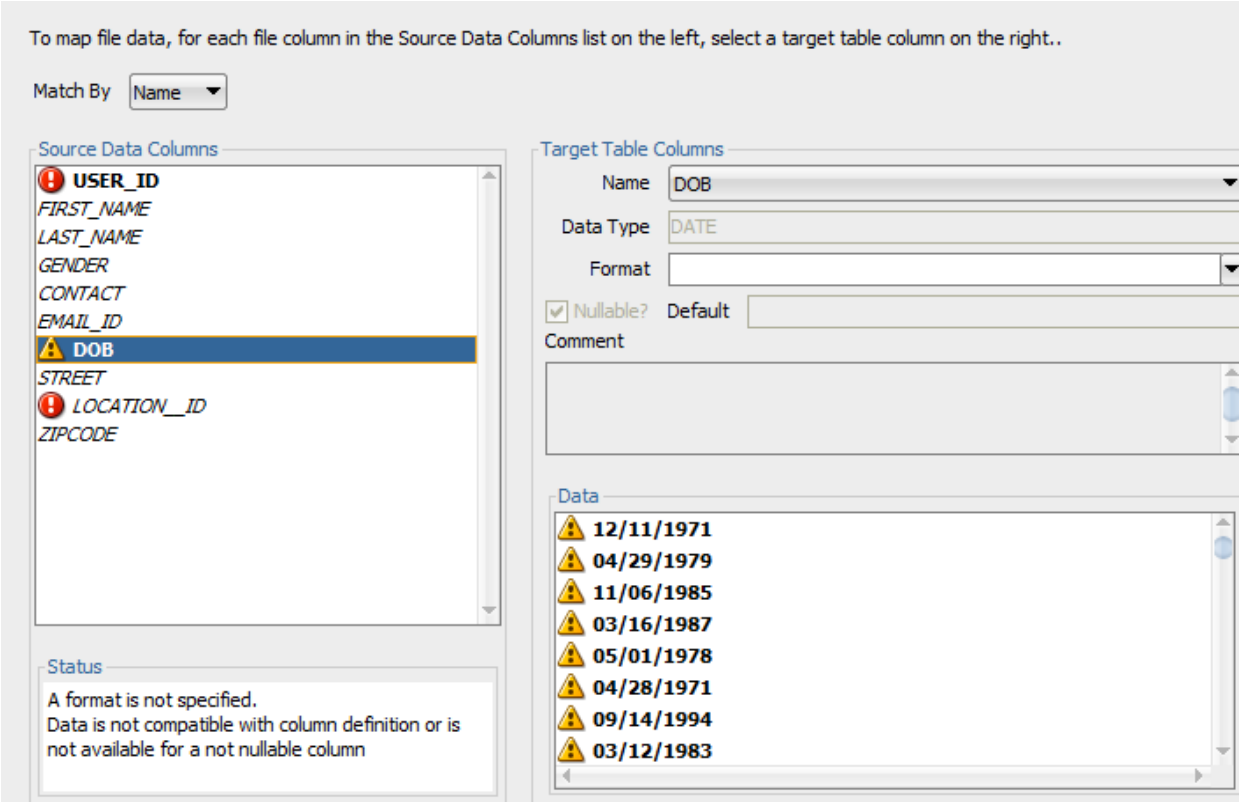


Total Count Of The Rows Inserted Into The Database Are As Under -

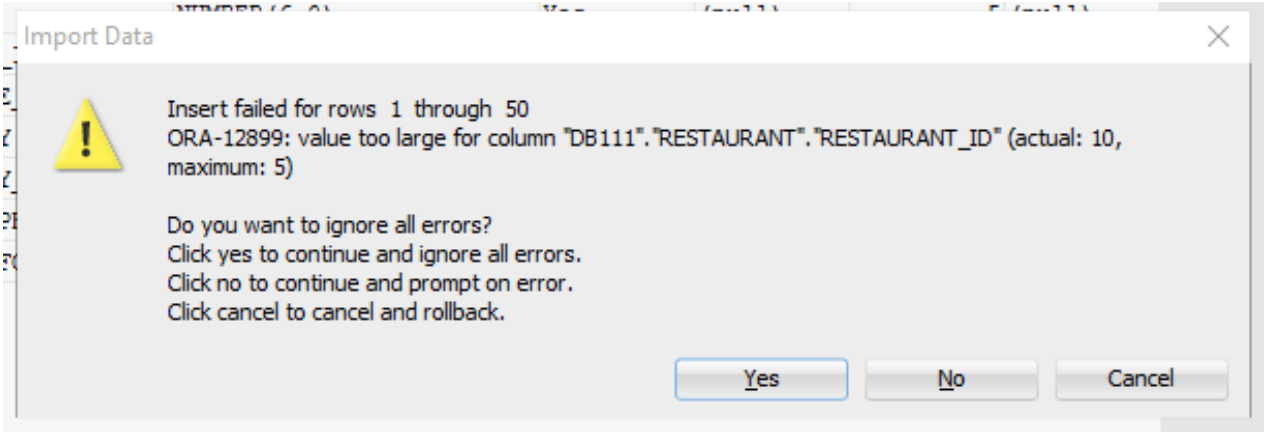
Table Name	Count
AMBIENCE	11
CUISINE	11
LOCATION	23
MENU_ITEMS	91
RATING	20199
RESTAURANT	19730
USER_DATA	8214

Issues faced during Data loading

1. We used CSV file to import data into the user table. While performing the operation we faced issues with the date format. The format in file was MM/DD/YYYY, which was not supported by oracle sql developer. SO we had to change the current format of the date in oracle sql developer. We assigned the preferred date format by going to Tools-> Preferences -> Database -> NLS.



2. We faced issues with table size space while entering 20000 rows in restaurant table. So while executing the query we were interrupted at 7300 rows and data up to that point was fed to the database. We analyzed the issue and came to know that we were running out of space as we had assigned size for varchar2 as 255 bytes. So we had to reduce the assigned size which helped us to free some data into the table spaces.



3. While entering user data, we faced an error of not enough space to assign variables. This occurred due to the fact that we had assigned varchar2 size for userID as 10 bytes. But as the userID was 10 digits so it was not getting enough allocated space. After increasing the size for variable to 20 bytes we were able to proceed with the data loading.

5 PERFORMANCE TUNING

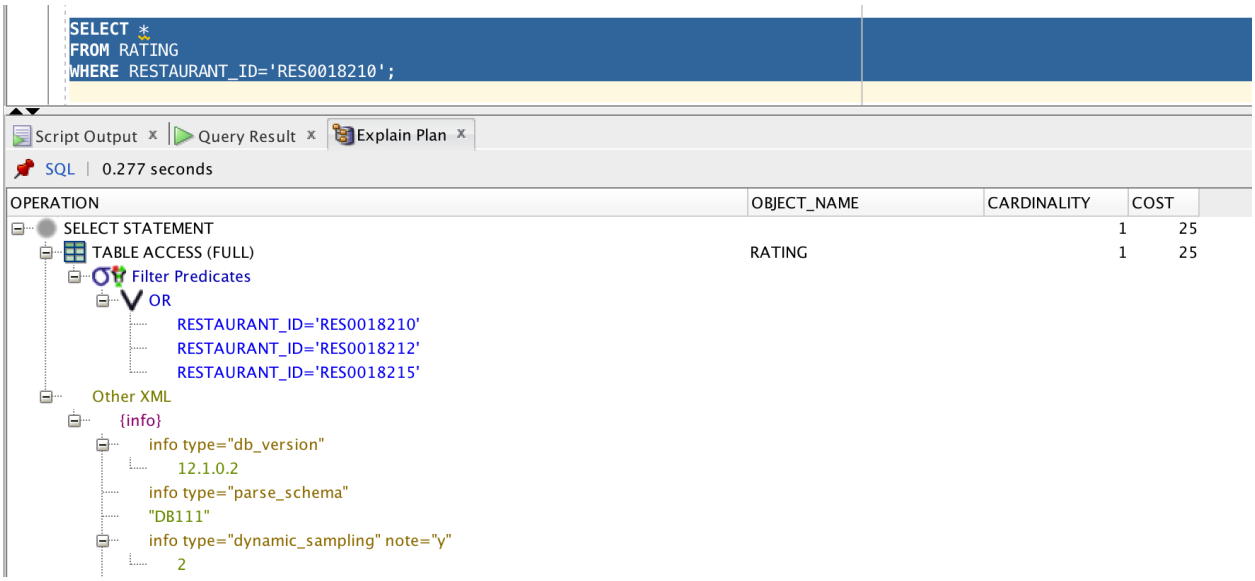
This section covers some of the experiments run as part of the project related to performance tuning.

5.1 SIMPLE INDEXING

Query 1

Simple query is executed to extract information about the Restaurant ID. This operation has been performed with and without index.

Retrieving Restaurant_ID without Index



Retrieving Restaurant_ID with Index

```
CREATE INDEX IDX_RATING_COL_REST_ID ON RATING(RESTAURANT_ID);
SELECT *
FROM RATING
WHERE RESTAURANT_ID='RES0018210';
```

Script Output | Query Result | Explain Plan

SQL | 0.236 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST	
SELECT STATEMENT			2	3
TABLE ACCESS (BY INDEX ROWID BATCHED)	RATING		2	3
INDEX (RANGE SCAN)	IDX_RATING_COL_REST_ID		2	1
Access Predicates				
RESTAURANT_ID='RES0018210'				
Other XML				
{info}				
info type="db_version"				
12.1.0.2				
info type="parse_schema"				
"DB111"				
info type="dynamic_sampling" note="y"				
2				
info type="plan_hash_full"				
1186417100				

Observation:

Type	Cost	Operation
Using '='		
Without Index	25	Full Table scan has happened
With Index	3	Index has been used to directly identify the matching value

Query 2

Easy query has been used to fetch the range of Restaurant_ID. This operation again has been performed using with and without index.

Range of Restaurant_ID extracted without Index

```
SELECT *
FROM RATING
WHERE RESTAURANT_ID BETWEEN 'RES0018210' AND 'RES0018215';
```

Script Output | Query Result | Explain Plan

SQL | 0.222 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		6	25
TABLE ACCESS (FULL)	RATING	6	25
Filter Predicates			
AND			
RESTAURANT_ID>='RES0018210'			
RESTAURANT_ID<='RES0018215'			
Other XML			
{info}			
info type="db_version"			
12.1.0.2			
info type="parse_schema"			
"DB111"			
info type="dynamic_sampling" note="y"			
2			
info type="plan_hash_full"			

Range of Restaurant_ID extracted with Index

```
CREATE INDEX IDX_RATING_COL_REST_ID ON RATING(RESTAURANT_ID);
SELECT *
FROM RATING
WHERE RESTAURANT_ID BETWEEN 'RES0018210' AND 'RES0018215';
```

Script Output | Query Result | Explain Plan

SQL | 0.245 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7	9
TABLE ACCESS (BY INDEX ROWID BATCHED)	RATING	7	9
INDEX (RANGE SCAN)	IDX_RATING_COL_REST_ID	7	2
Access Predicates			
AND			
RESTAURANT_ID>='RES0018210'			
RESTAURANT_ID<='RES0018215'			
Other XML			
{info}			
info type="db_version"			
12.1.0.2			
info type="parse_schema"			
"DB111"			
info type="dynamic_sampling" note="y"			
2			

Observation:

Type	Cost	Operation
Using 'BETWEEN'		
Without Index	25	Full Table scan has happened
With Index	9	Index has been used and only Range scan has happened.

Query 3

Query has been executed to extract Restaurant_ID using IN operator. This operation has been performed using with and without Index.

Extracting Restaurant_ID using IN operator without Index

```
SELECT *
FROM RATING
WHERE RESTAURANT_ID IN('RES0018210','RES0018212','RES0018215');
```

Script Output | Query Result | Explain Plan

SQL | 0.272 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	25
TABLE ACCESS (FULL)	RATING	1	25
Filter Predicates			
OR			
RESTAURANT_ID='RES0018210'			
RESTAURANT_ID='RES0018212'			
RESTAURANT_ID='RES0018215'			
Other XML			
{info}			
info type="db_version"			
12.1.0.2			
info type="parse_schema"			
"DB111"			
info type="dynamic_sampling" note="y"			
2			

Extracting Restaurant_ID using IN operator with Index

```
CREATE INDEX IDX_RATING_COL_REST_ID ON RATING(RESTAURANT_ID);
SELECT *
FROM RATING
WHERE RESTAURANT_ID IN('RES0018210','RES0018212','RES0018215');
```

Script Output | Query Result | Explain Plan

SQL | 0.188 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	24
INLIST ITERATOR			
TABLE ACCESS (BY INDEX ROWID BATCHED)	RATING	1	24
INDEX (RANGE SCAN)	IDX_RATING_COL_REST_ID	79	4
Access Predicates			

Other XML

```
{info}
    info type="db_version"
        12.1.0.2
    info type="parse_schema"
    "DB111"
    info type="dynamic_sampling" note="y"
        2
    info type="plan_hash_full"
        3154004439
```

Observation:

Type	Cost	Operation
Using 'IN'		
Without Index	25	Full Table scan has happened
With Index	24	Index has been used and only Range scan has happened.

5.2 FUNCTION BASED INDEXING

Objective: We are trying to find out how much will be the cost of eating for two people in restaurants having '008' in their location_id. For this we are executing a query with and without index illustrating function based indexing.

A function-based index computes the value of an expression that involves one or more columns and stores it in the index. It improves the performance of queries that use the index expression and increases the number of situations where the database can perform an index range scan (resulting in typically a fast response time) instead of a full index scan.

Without Index

```
SELECT BUDGET_FOR_TWO
FROM RESTAURANT
WHERE SUBSTR(LOCATION_ID,6,3)='008';
```

Script Output | Query Result | Explain Plan

SQL | 0.373 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		73	102
TABLE ACCESS (FULL)	RESTAURANT	73	102
Filter Predicates			
SUBSTR(LOCATION_ID,6,3)='008'			
Other XML			

With Index

```
CREATE INDEX IDX_RESTAURANT_COL_LOC_ID ON RESTAURANT(SUBSTR(LOCATION_ID,6,3),LOCATION_ID);
SELECT BUDGET_FOR_TWO
FROM RESTAURANT
WHERE SUBSTR(LOCATION_ID,6,3)='008';
```

Script Output | Query Result | Explain Plan

SQL | 0.358 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		73	29
TABLE ACCESS (BY INDEX ROWID BATCHED)	RESTAURANT	73	29
INDEX (RANGE SCAN)	IDX_RESTAURANT_COL_LOC...	29	2
Access Predicates			
SUBSTR(LOCATION_ID,6,3)='008'			
Other XML			

Observation:

Type	Cost	Operation
Without Index	102	Full Table scan has happened
With Index	29	Index has been used and only Range scan has happened.

5.3 MATERIALIZED VIEW

Materialized views are generally used to pre-compute and store aggregated data. This increases data availability by providing local access to the target data and when combined with mass deployment and data subsetting (both of which also reduce network loads), greatly enhance the performance and reliability of the database.

Query - CREATE MATERIALIZED VIEW MVF AS

SELECT * FROM Restaurant;

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST
SELECT STATEMENT			102		
TABLE ACCESS (FULL)	RESTAURANT	7300	102	4	

The data is already pre-computed, the response time and the cost of execution are brought down considerably when implementing Materialized Views.

5.4 SELECTIVITY

a) 10 Rows

```
SELECT *
FROM RESTAURANT
WHERE RESTAURANT_ID <='RES0000010';
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.333 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	3
TABLE ACCESS (BY INDEX ROWID BATCHED)	RESTAURANT	1	3
INDEX (RANGE SCAN)	RESTAURANT_ID_PK	1	2
Access Predicates			
RESTAURANT_ID<='RES0000010'			

b) 99 Rows

```
SELECT *
FROM RESTAURANT
WHERE RESTAURANT_ID <='RES0000100';
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.476 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		5	3
TABLE ACCESS (BY INDEX ROWID BATCHED)	RESTAURANT	5	3
INDEX (RANGE SCAN)	RESTAURANT_ID_PK	5	2
Access Predicates			
RESTAURANT_ID<='RES0000100'			

c) 992 Rows

```
SELECT *
FROM RESTAURANT
WHERE RESTAURANT_ID <='RES0001000';
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.361 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1042	21
TABLE ACCESS (BY INDEX ROWID BATCHED)	RESTAURANT	1042	21
INDEX (RANGE SCAN)	RESTAURANT_ID_PK	1042	4
Access Predicates			
RESTAURANT_ID<='RES0001000'			

d) 4963 Rows

```
SELECT *
FROM RESTAURANT
WHERE RESTAURANT_ID <='RES0005000';
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.366 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		5207	101
TABLE ACCESS (BY INDEX ROWID BATCHED)	RESTAURANT	5207	101
INDEX (RANGE SCAN)	RESTAURANT_ID_PK	5207	16
Access Predicates			
RESTAURANT_ID<='RES0005000'			

e) 7927 Rows

```
SELECT *
FROM RESTAURANT
WHERE RESTAURANT_ID <='RES0008000';
```

Script Output | Query Result | Query Result 1 | Explain Plan

SQL | 0.353 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7300	102
TABLE ACCESS (FULL)	RESTAURANT	7300	102
Filter Predicates			
RESTAURANT_ID<='RES0008000'			

Observation:

Query Fetched	Type Scan	Cardinality
WHERE RESTAURANT_ID <='RES0000010';	RANGE SCAN	1
WHERE RESTAURANT_ID <='RES0000100';	RANGE SCAN	5
WHERE RESTAURANT_ID <='RES0001000';	RANGE SCAN	1042
WHERE RESTAURANT_ID <='RES0005000';	RANGE SCAN	5207
WHERE RESTAURANT_ID <='RES0008000';	FULL TABLE SCAN	7300

It is observed that at the cardinality of 5207 which fetches 4963 rows, type of scan switches from RANGE SCAN to FULL TABLE SCAN.

5.5 DATABASE PARTITIONING

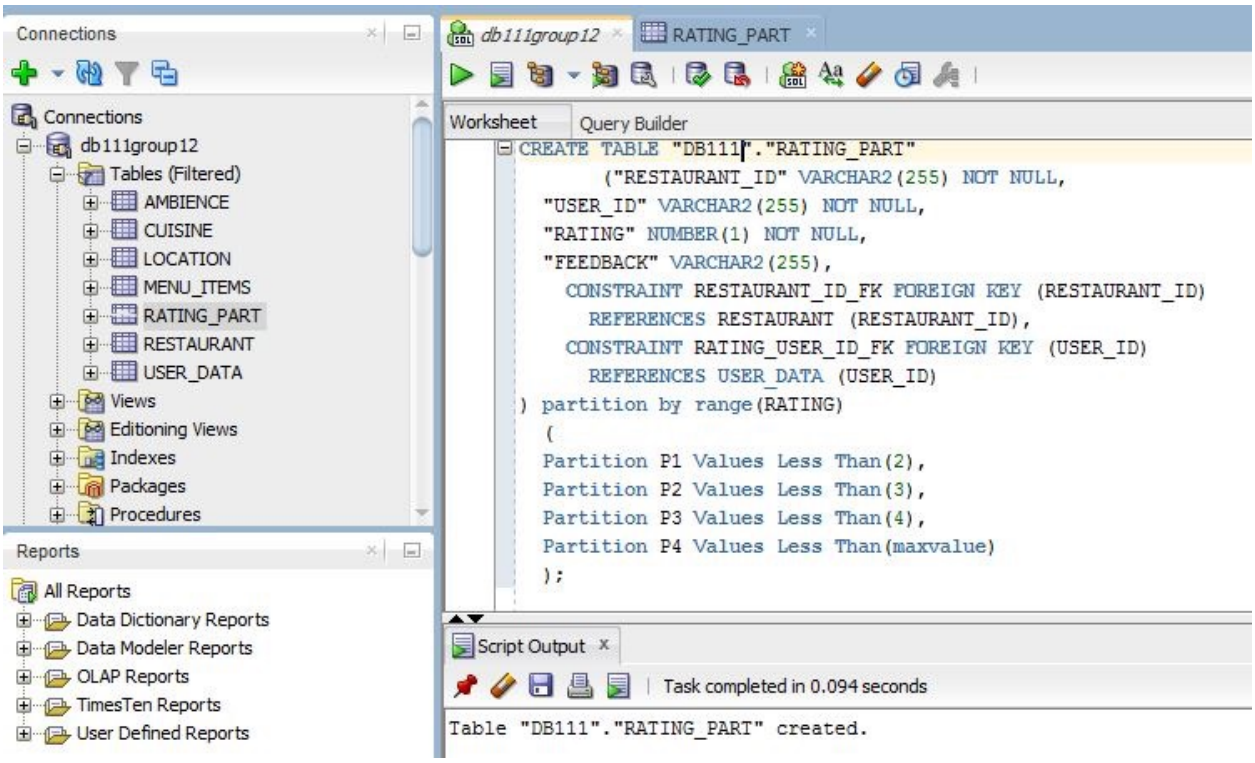
The purpose of the database partitioning in this database is to segregate the cells in a table based on the Utilization Rate.

The **partitioning** can be done by either building separate smaller databases (each with its own tables, indices, and transaction logs), or by splitting selected elements, for example just one table.

Horizontal **partitioning** involves putting different rows into different tables. Low utilized cells or sites, often called LUTs are given utmost priority and special improvement plans are designed to bring up the utilization rate. The cells are divided into different categories and each cell has a value every day and is fit into different partitions as per the utilization rate.

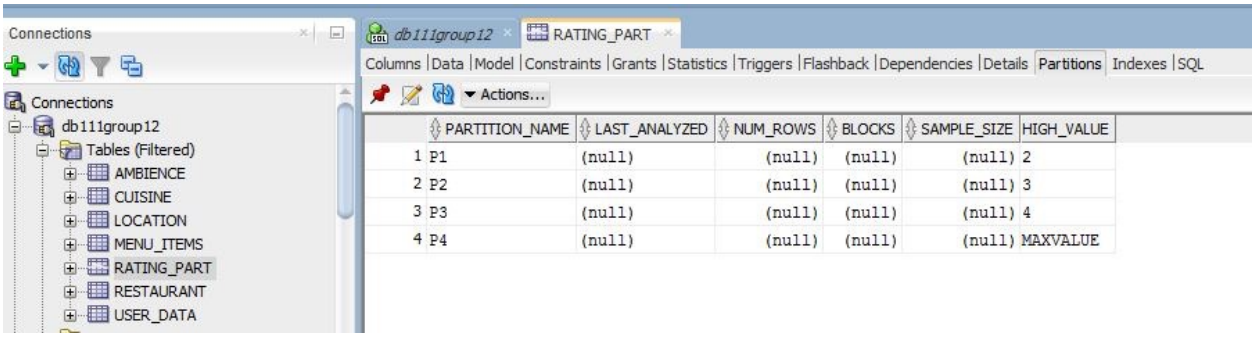
5.5.1 RANGE PARTITIONING

Below is the query to create partition by Range:

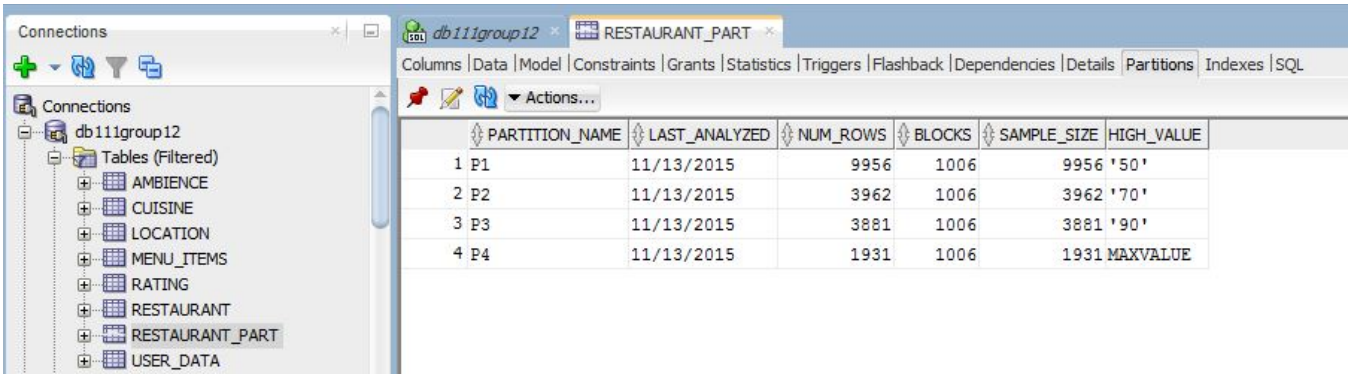


The table is divided into 4 partitions based on the Rating of the restaurants given by the users.

Following table shows there is a partition with no data row in a table:

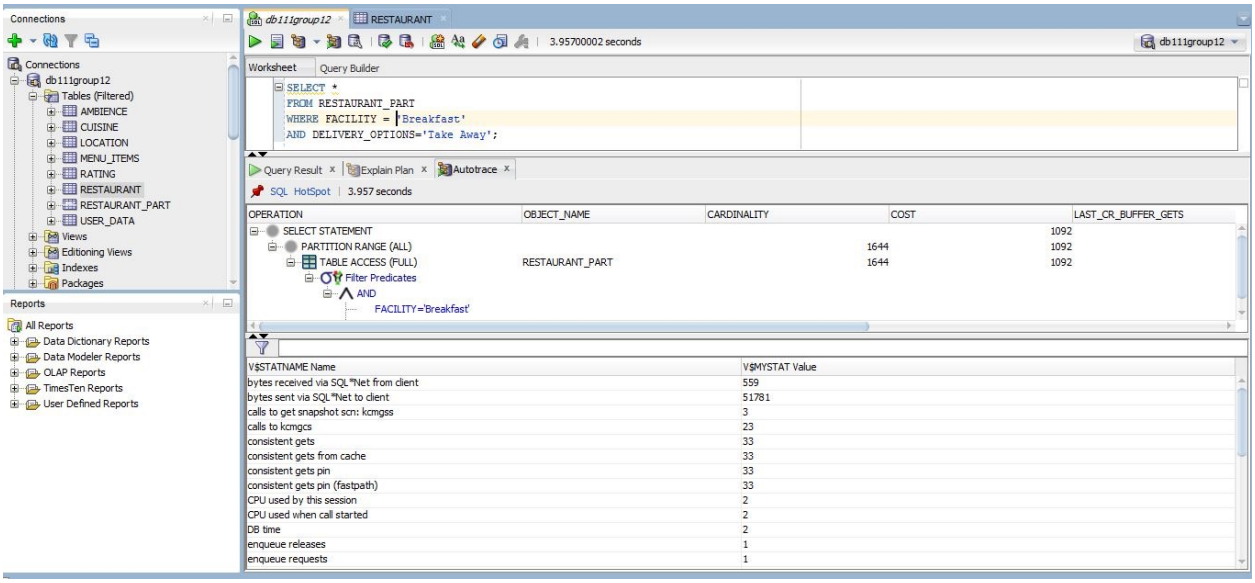


Following table shows different block size and sample size used in the partition while data row insertion from the table restaurant of size 19730



	PARTITION_NAME	LAST_ANALYZED	NUM_ROWS	BLOCKS	SAMPLE_SIZE	HIGH_VALUE
1	P1	11/13/2015	9956	1006	9956	'50'
2	P2	11/13/2015	3962	1006	3962	'70'
3	P3	11/13/2015	3881	1006	3881	'90'
4	P4	11/13/2015	1931	1006	1931	MAXVALUE

Following image illustrates the query path and auto trace with the cost involved for the query on the partitioned table.

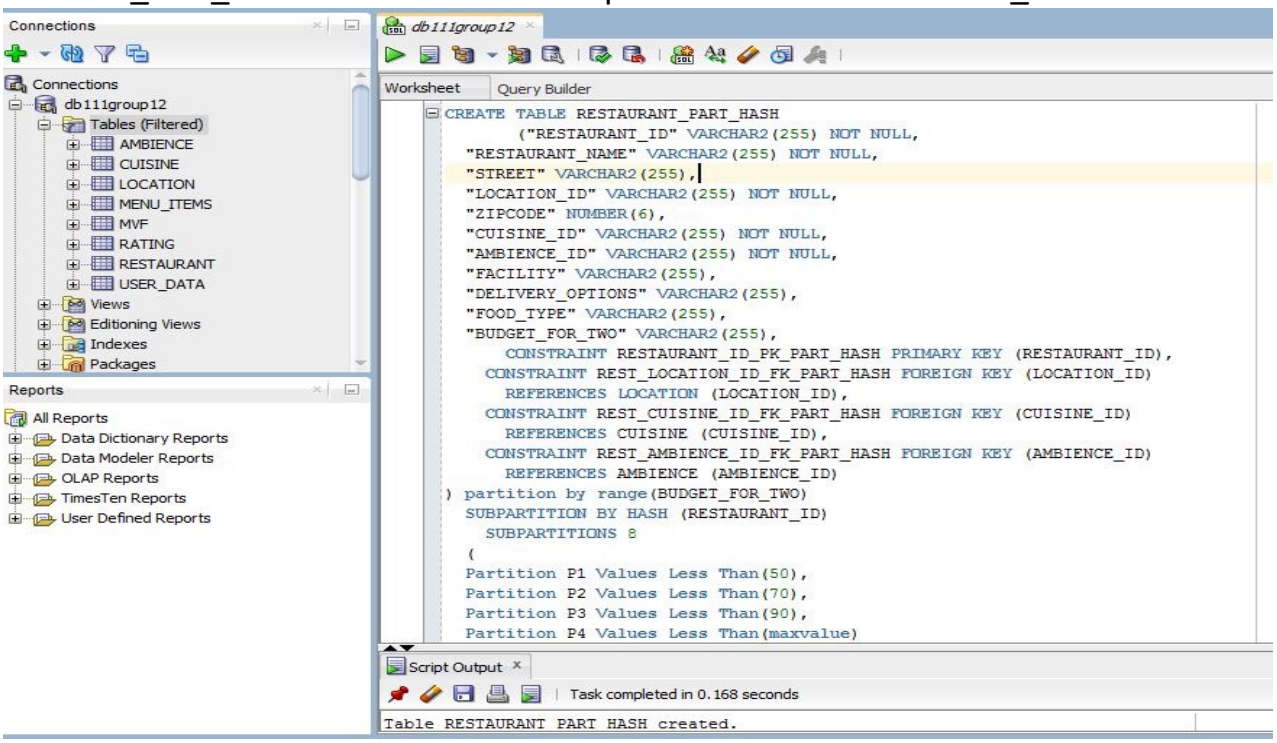


OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT				1092
PARTITION RANGE (ALL)			1644	1092
TABLE ACCESS (FULL)	RESTAURANT_PART		1644	1092
Filter Predicates				
AND				
FACILITY='Breakfast'				

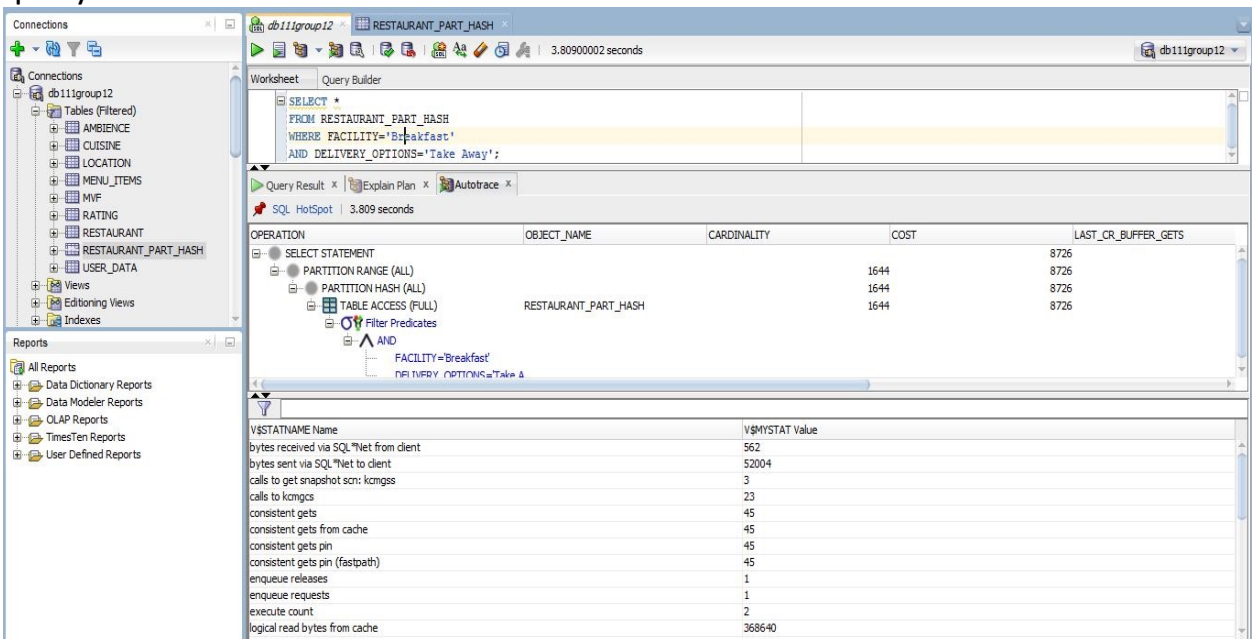
V$STATNAME Name	V$MYSTAT Value
bytes received via SQL*Net from client	559
bytes sent via SQL*Net to client	51781
calls to get snapshot scn: kcmgss	3
calls to kcmgcs	23
consistent gets	33
consistent gets from cache	33
consistent gets pin	33
consistent gets pin (fastpath)	33
CPU used by this session	2
CPU used when call started	2
DB time	2
enqueue releases	1
enqueue requests	1

5.5.2 HASH PARTITIONING

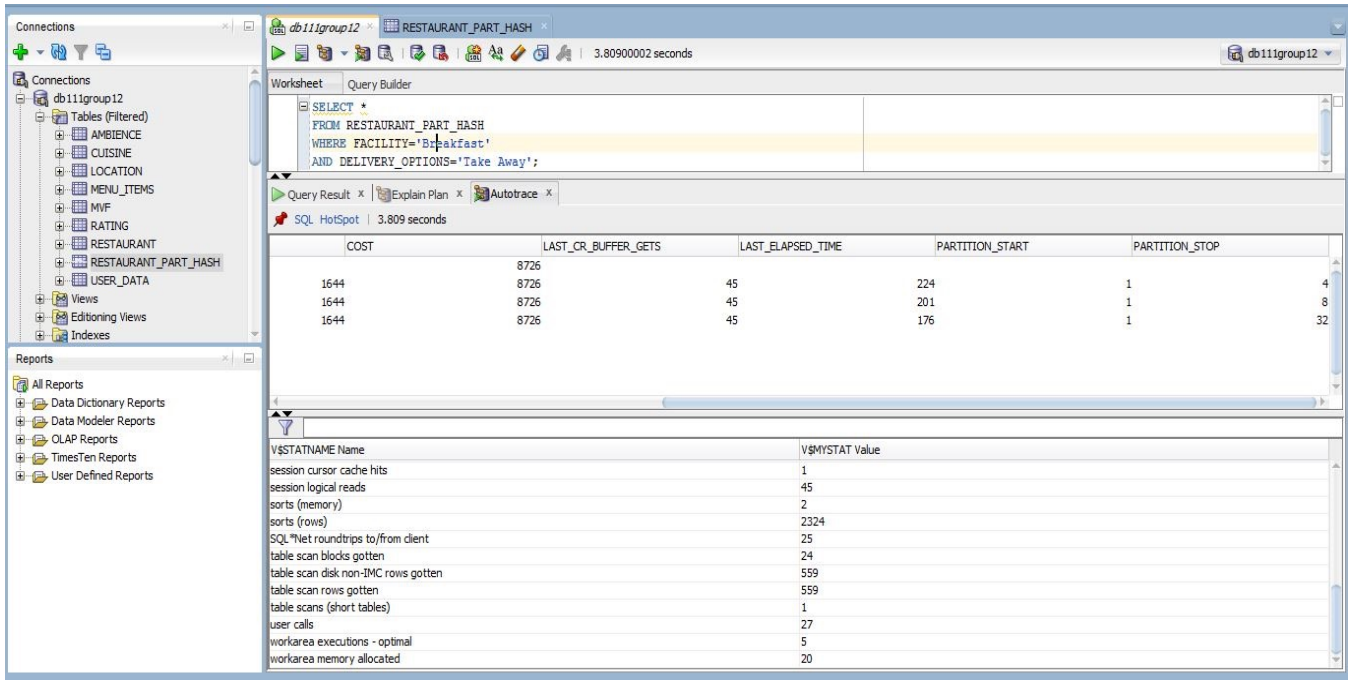
Below image illustrates that the table is divided into 4 major partitions based on the BUDGET_FOR_TWO column with 8 sub-partitions on RESTAURANT_ID:



Following image illustrates the query path and auto trace with the cost involved for the query



Following illustrates Partitioning traces for the table based on the query fired.



6 QUERYING

1) Find restaurants in Tampa which plays Live Music and serves BBQ food.

Worksheet | Query Builder

```
SELECT
RESTAURANT_NAME
FROM
RESTAURANT
INNER JOIN LOCATION
ON RESTAURANT.LOCATION_ID=LOCATION.LOCATION_ID
INNER JOIN AMBIENCE
ON RESTAURANT.AMBIENCE_ID=AMBIENCE.AMBIENCE_ID
INNER JOIN CUISINE
ON RESTAURANT.CUISINE_ID=CUISINE.CUISINE_ID
WHERE LOCATION.CITY='Tampa'
AND
AMBIENCE.AMBIENCE='Live Music'
AND
CUISINE.CUISINE='BBQ'
ORDER BY RESTAURANT.RESTAURANT_NAME;
```

Query Result

SQL | All Rows Fetched: 4 in 0.094 seconds

	RESTAURANT_NAME
1	EMPIRE TACO
2	GRIT N GLORY
3	JULIETTE
4	ZZ CLAM BAR

2) Find the maximum and minimum price of each cuisines that are served by restaurants.

```
SELECT CUISINE.CUISINE,MAX(PRICE),MIN(PRICE)
FROM  MENU_ITEMS
INNER JOIN CUISINE
ON MENU_ITEMS.CUISINE_ID=CUISINE.CUISINE_ID
GROUP BY CUISINE.CUISINE;
```

Query Result

SQL | All Rows Fetched: 11 in 0.079 seconds

	CUISINE	MAX(PRICE)	MIN(PRICE)
1	Italian	19	11
2	Chinese	17	5
3	Seafood	19	5
4	American	20	5
5	BBQ	19	7
6	Indian	20	8
7	Thai	14	5
8	Japanese	20	6
9	Afghan	18	5
10	Continental	19	5
11	Mexican	18	7

3) Find name and Address of restaurant in LOS Angeles that serve Italian food priced below $15 and have delivery options available.

Worksheet | Query Builder

```
SELECT
RESTAURANT_NAME, STREET
FROM
RESTAURANT
INNER JOIN LOCATION
ON RESTAURANT.LOCATION_ID=LOCATION.LOCATION_ID
WHERE RESTAURANT.CUISINE_ID IN
(SELECT CUISINE.CUISINE_ID
FROM CUISINE
INNER JOIN MENU_ITEMS
ON MENU_ITEMS.CUISINE_ID=CUISINE.CUISINE_ID
WHERE CUISINE='Italian' AND PRICE < 15)
AND LOCATION.CITY='Los Angeles' AND RESTAURANT.DELIVERY_OPTIONS='Delivery'
ORDER BY RESTAURANT_NAME;
```

Query Result

SQL | All Rows Fetched: 26 in 0.093 seconds

	RESTAURANT_NAME	STREET
1	Abumi Sushi	207 EAST 26 STREET
2	BEYOND SUSHI	227229 E 14TH ST
3	BUONA NOTTE RISTORANTE	120 MULBERRY STREET
4	BURGER KING	734 BROADWAY
5	CAFE ANGELIQUE	317 BLEECKER STREET
6	CANA Y CAFE BAR RESTAURANT	856 KNICKERBOCKER AVENUE
7	CENTRAL CAFE	16 VANDERBILT AVENUE
8	FOO AN CHINESE FOOD	22318 SOUTH CONDUIT AVENUE
9	GENES @ BARNEYS	660 MADISON AVENUE
10	GREEN OLIVE KOSHER PIZZA & FALAFEL	271-11 UNION TURNPIKE

4. Find cafe in Los Angeles that are rated 5.

```
SELECT
RESTAURANT_NAME, AVG(RATING.RATING)
FROM
RESTAURANT
INNER JOIN RATING
ON RESTAURANT.RESTAURANT_ID=RATING.RESTAURANT_ID
INNER JOIN LOCATION
ON RESTAURANT.LOCATION_ID=LOCATION.LOCATION_ID
WHERE LOCATION.CITY='Los Angeles' AND RESTAURANT.FOOD_TYPE='Cafe'
GROUP BY RATING.RESTAURANT_ID, RESTAURANT_NAME
HAVING AVG(RATING)=5;
```

Query Result

SQL | All Rows Fetched: 17 in 0.105 seconds

	RESTAURANT_NAME	AVG(RATING.RATING)
1	ENJOY MY BAGELS	5
2	BUNCH OF BAGELS	5
3	NEWS BAR	5
4	DUNKIN' DONUTS	5
5	STARBUCKS COFFEE	5
6	TU SABOR LATINO	5
7	PARSONS DELI & GROCERY	5
8	GOOD WORLD KTV	5
9	THE HAVEN	5
10	BCD TOFU HOUSE	5
11	EL MANGU SABROSO RESTAURANT	5
12	DOS CAMINOS	5

7 DBA Scripts

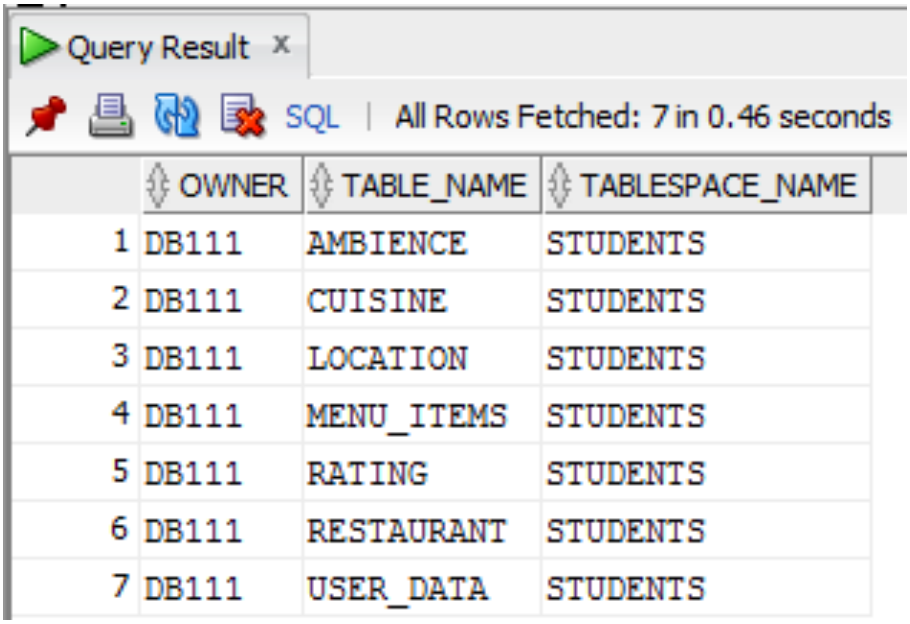
Database Administration scripts are used to manage and monitor the database, and also for status checks and privilege handling. We have listed some of the useful DBA scripts which we found as interesting as well as useful.

1. Display all Tables in the Database

This is a basic script which displays all tables present in the database, along with its owner and Tablespace Name. The below query removes all Oracle Admin owned system tables so as to filter out the Manually created tables by different users in the DB. The scripts could be modified to search for particular tables or owners.

```
SELECT
OWNER,
TABLE_NAME,
TABLESPACE_NAME
FROM
DBA_ALL_TABLES
WHERE
OWNER = 'DB111'
ORDER BY OWNER, TABLE_NAME, TABLESPACE_NAME;
```

Result:



	OWNER	TABLE_NAME	TABLESPACE_NAME
1	DB111	AMBIENCE	STUDENTS
2	DB111	CUISINE	STUDENTS
3	DB111	LOCATION	STUDENTS
4	DB111	MENU_ITEMS	STUDENTS
5	DB111	RATING	STUDENTS
6	DB111	RESTAURANT	STUDENTS
7	DB111	USER_DATA	STUDENTS

2. Active Users and Sessions Details:

This below query lists out all Users and Sessions in the Database by accessing Session and Process system tables in the server. This helps the DBA to keep track of the Active users in the DB and monitor the same in a security/reporting perspective.

```
SELECT
NVL(SESS.USERNAME, '(ORACLE)') AS USERNAME,
SESS.OSUSER,
SESS.SID,
SESS.SERIAL#,
PRO.SPID,
SESS.STATUS,
SESS.SERVICE_NAME,
SESS.MACHINE,
SESS.PROGRAM,
TO_CHAR(SESS.LOGON_TIME,'DD-MON-YYYY HH24:MI:SS') AS LOGIN_TIME
FROM V$SESSION SESS,
V$PROCESS PRO
WHERE SESS.PADDR = PRO.ADDR
ORDER BY SESS.USERNAME, SESS.OSUSER;
```

Result(as on 13th Nov 2015, 5:55 pm)

Query Result

SQL | All Rows Fetched: 69 in 0.193 seconds

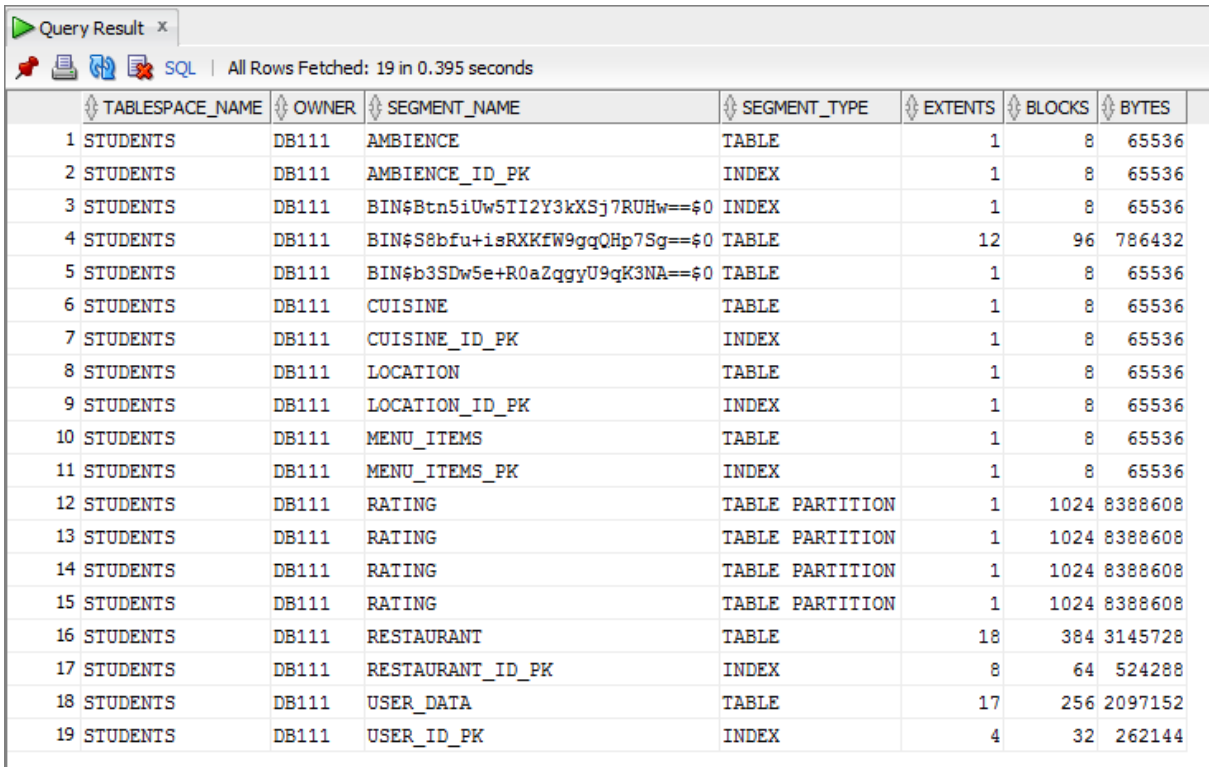
	USERNAME	OSUSER	SID	SERIAL#	SPID	STATUS	SERVICE_NAME	MACHINE	PROGRAM	LOGIN_TIME
1	DB106	Chintan	76	6330	240	INACTIVE	SYS$USERS	Chintan	SQL Developer	13-NOV-2015 17:43:34
2	DB106	nitin	60	63770	4012	INACTIVE	SYS$USERS	nitin-pc	SQL Developer	13-NOV-2015 16:28:11
3	DB107	Bhavyank	49	37603	3136	INACTIVE	SYS$USERS	Bhavyank	SQL Developer	13-NOV-2015 16:54:14
4	DB107	Md. Ihtesham	80	17557	2788	INACTIVE	SYS$USERS	Lenovo-PC	SQL Developer	13-NOV-2015 17:22:43
5	DB107	RITHIK	62	31226	3788	INACTIVE	SYS$USERS	RITHIK-PC	SQL Developer	13-NOV-2015 17:15:59
6	DB111	Sachin Kant Misra	1	35808	1388	INACTIVE	SYS$USERS	SKM-VAIO	SQL Developer	13-NOV-2015 15:26:15
7	DB111	harshitasrivastava	47	41062	4040	INACTIVE	SYS$USERS	Harshitas-Air.fios-router.home	SQL Developer	13-NOV-2015 16:57:59
8	DB111	hp	84	3526	2884	ACTIVE	SYS$USERS	hp-HP	SQL Developer	13-NOV-2015 17:41:17
9	DB111	hp	81	32253	1136	INACTIVE	SYS$USERS	hp-HP	SQL Developer	13-NOV-2015 17:17:08
10	DB111	jagpreet	56	12691	3880	INACTIVE	SYS$USERS	Jagpreets-MacBook-Pro.local	SQL Developer	13-NOV-2015 15:04:50
11	DB111	lenovo	72	42750	1604	INACTIVE	SYS$USERS	MeanMachine	SQL Developer	13-NOV-2015 15:07:25
12	DB112	AkshayKumar	55	1052	800	INACTIVE	SYS$USERS	WINDOWS-5DSAO4P	SQL Developer	13-NOV-2015 17:44:50
13	DB112	AkshayKumar	68	44892	3036	INACTIVE	SYS$USERS	WINDOWS-5DSAO4P	SQL Developer	13-NOV-2015 15:15:41
14	DB112	Manohar	89	25919	2780	INACTIVE	SYS$USERS	Mano	SQL Developer	13-NOV-2015 15:48:53
15	DB112	Rohit	44	59943	2632	INACTIVE	SYS$USERS	Rohit	SQL Developer	13-NOV-2015 15:29:03
16	DB112	Saikrishna B	61	10212	2012	INACTIVE	SYS$USERS	Saikrishna	SQL Developer	13-NOV-2015 15:12:18
17	DB112	gvred	78	63694	3440	INACTIVE	SYS$USERS	DESKTOP-UMI44NM	SQL Developer	13-NOV-2015 15:19:22
18	DB113	Deepen	75	55556	4064	INACTIVE	SYS$USERS	Deepen-PC	SQL Developer	13-NOV-2015 17:47:30
19	DB113	preet	50	28126	2212	INACTIVE	SYS$USERS	DESKTOP-U5SDGGK	SQL Developer	13-NOV-2015 12:41:00
20	DB113	shivani	70	60453	1828	INACTIVE	SYS$USERS	WINDOWS-KSE64OV	ORACLE.EXE (P004)	13-NOV-2015 17:01:43
21	DB113	shivani	77	43494	2948	INACTIVE	SYS$USERS	WINDOWS-KSE64OV	ORACLE.EXE (P003)	13-NOV-2015 17:01:43
22	DB113	shivani	59	38785	2904	INACTIVE	SYS$USERS	WINDOWS-KSE64OV	ORACLE.EXE (P001)	13-NOV-2015 17:01:43

3. Table Space Usage:

The below query displays the Tablespace Name, Owner, Table details, Type of Segment (Table/Index), Extents, Number of DB Blocks in the segment and Number of bytes in the segment in the DB. This query can be used to track the size and blocks of data for each table. Below query has been filtered for only ADB owners.

```
SELECT
TABLESPACE_NAME,
OWNER,
SEGMENT_NAME,
SEGMENT_TYPE,
EXTENTS,
BLOCKS,
BYTES
FROM DBA_SEGMENTS
WHERE OWNER LIKE 'DB111'
ORDER BY OWNER, SEGMENT_NAME;
```

Result:


Query Result

SQL | All Rows Fetched: 19 in 0.395 seconds

	TABLESPACE_NAME	OWNER	SEGMENT_NAME	SEGMENT_TYPE	EXTENTS	BLOCKS	BYTES
1	STUDENTS	DB111	AMBIENCE	TABLE	1	8	65536
2	STUDENTS	DB111	AMBIENCE_ID_PK	INDEX	1	8	65536
3	STUDENTS	DB111	BIN$Btn5iUw5TI2Y3kXSj7RUHw==$0	INDEX	1	8	65536
4	STUDENTS	DB111	BIN$S8bfu+isRXKfW9gqQHp7Sg==$0	TABLE	12	96	786432
5	STUDENTS	DB111	BIN$b3SDw5e+R0aZqgyU9qK3NA==$0	TABLE	1	8	65536
6	STUDENTS	DB111	CUISINE	TABLE	1	8	65536
7	STUDENTS	DB111	CUISINE_ID_PK	INDEX	1	8	65536
8	STUDENTS	DB111	LOCATION	TABLE	1	8	65536
9	STUDENTS	DB111	LOCATION_ID_PK	INDEX	1	8	65536
10	STUDENTS	DB111	MENU_ITEMS	TABLE	1	8	65536
11	STUDENTS	DB111	MENU_ITEMS_PK	INDEX	1	8	65536
12	STUDENTS	DB111	RATING	TABLE PARTITION	1	1024	8388608
13	STUDENTS	DB111	RATING	TABLE PARTITION	1	1024	8388608
14	STUDENTS	DB111	RATING	TABLE PARTITION	1	1024	8388608
15	STUDENTS	DB111	RATING	TABLE PARTITION	1	1024	8388608
16	STUDENTS	DB111	RESTAURANT	TABLE	18	384	3145728
17	STUDENTS	DB111	RESTAURANT_ID_PK	INDEX	8	64	524288
18	STUDENTS	DB111	USER_DATA	TABLE	17	256	2097152
19	STUDENTS	DB111	USER_ID_PK	INDEX	4	32	262144

4. Program CPU Performance:

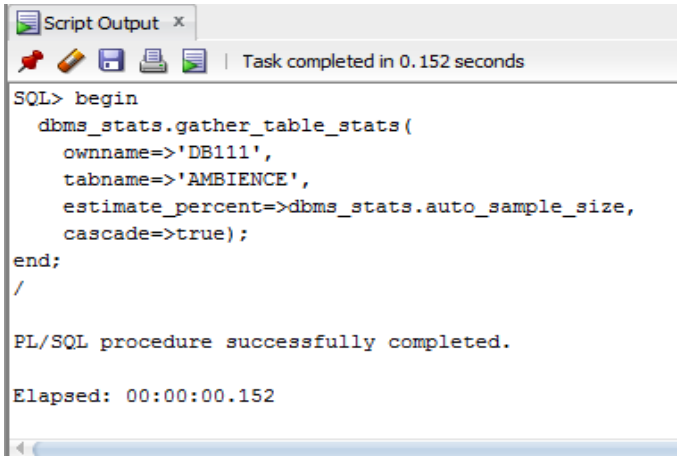
This script can be used by the DBA to monitor CPU performance. The query lists out top 10 programs which uses maximum CPU, aiding the DBA to monitor program performance.

```
SELECT ROWNUM AS RANK,
 TEMP.*
FROM
 (SELECT V.SID,
 PROGRAM,
 ROUND(V.VALUE / (100 * 60),2) CPUMINS
 FROM V$STATNAME S,
 V$SESSTAT V,
 V$SESSION SESS
 WHERE S.NAME = 'CPU USED BY THIS SESSION'
 AND SESS.SID = V.SID
 AND V.STATISTIC#=S.STATISTIC#
 AND V.VALUE >0
 ORDER BY V.VALUE DESC
 ) TEMP
WHERE ROWNUM <=10;
```

5. Gather Table Stats:

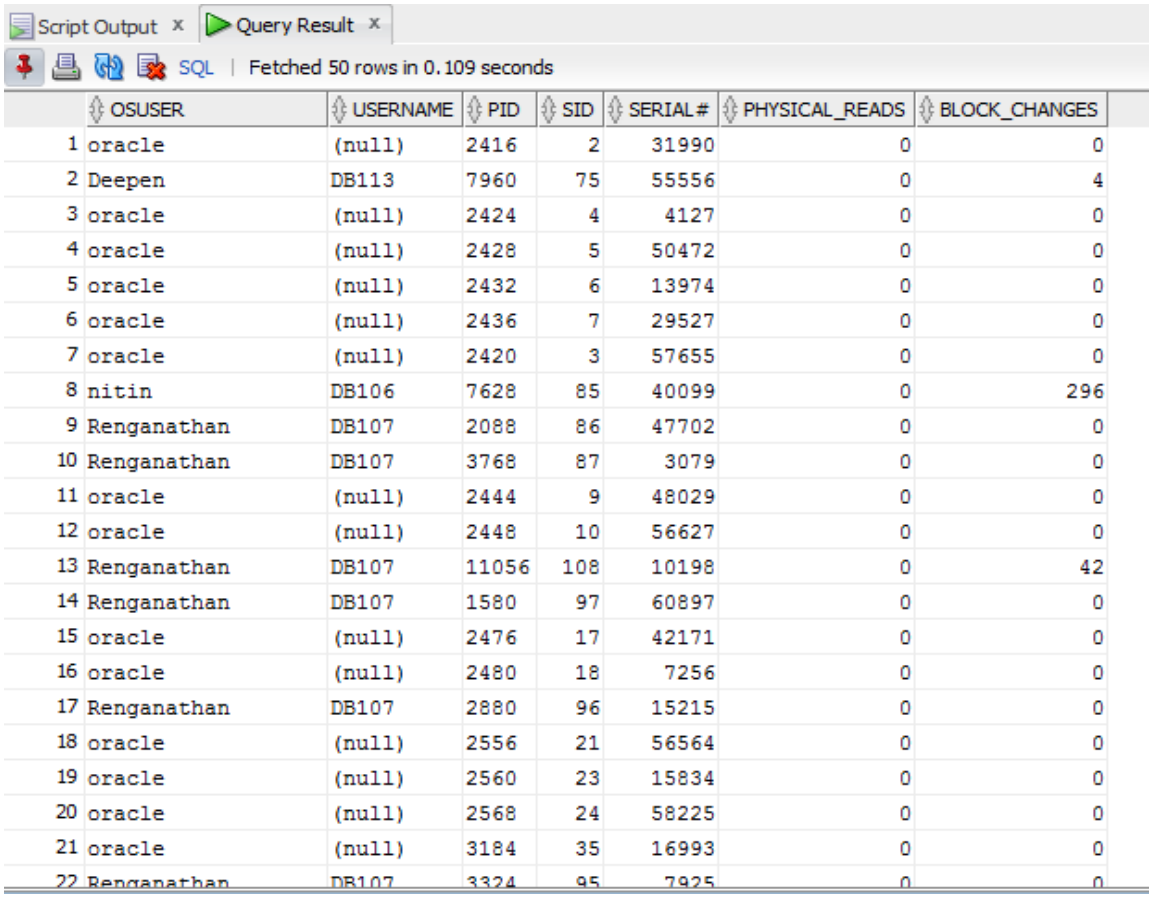
The below script is used to gather table stats. Similar queries can be used to gather schema stats as well.

```
begin
  dbms_stats.gather_table_stats(
    ownname=>'DB111',
    tabname=>'AMBIENCE',
    estimate_percent=>dbms_stats.auto_sam-
ple_size,
    cascade=>true);
end;
/
```



6. *Monitoring sessions with high physical reads:* The below mentioned query is used to monitor sessions with high physical reads and can help the DBA to optimize the DB performance based on this data.

```
set linesize 120
col osuser format a10
col username format 10
select
OSUSER osuser,
username,
PROCESS pid,
ses.SID sid,
SERIAL#,
PHYSICAL_READS,
BLOCK_CHANGES
from v$session ses,
v$sess_io sio
where ses.SID = sio.SID
order by PHYSICAL_READS;
```



Script Output | Query Result

SQL | Fetched 50 rows in 0.109 seconds

	OSUSER	USERNAME	PID	SID	SERIAL#	PHYSICAL_READS	BLOCK_CHANGES
1	oracle	(null)	2416	2	31990	0	0
2	Deepen	DB113	7960	75	55556	0	4
3	oracle	(null)	2424	4	4127	0	0
4	oracle	(null)	2428	5	50472	0	0
5	oracle	(null)	2432	6	13974	0	0
6	oracle	(null)	2436	7	29527	0	0
7	oracle	(null)	2420	3	57655	0	0
8	nitin	DB106	7628	85	40099	0	296
9	Renganathan	DB107	2088	86	47702	0	0
10	Renganathan	DB107	3768	87	3079	0	0
11	oracle	(null)	2444	9	48029	0	0
12	oracle	(null)	2448	10	56627	0	0
13	Renganathan	DB107	11056	108	10198	0	42
14	Renganathan	DB107	1580	97	60897	0	0
15	oracle	(null)	2476	17	42171	0	0
16	oracle	(null)	2480	18	7256	0	0
17	Renganathan	DB107	2880	96	15215	0	0
18	oracle	(null)	2556	21	56564	0	0
19	oracle	(null)	2560	23	15834	0	0
20	oracle	(null)	2568	24	58225	0	0
21	oracle	(null)	3184	35	16993	0	0
22	Renganathan	DB107	3324	95	7925	0	0

7. *Monitor datafiles with highest I/O activity:*

```
select * from (
select name,phyrds, phywrts,readtim,writetim
from v$filestat a, v$datafile b
where a.file#=b.file#
order by readtim desc) where rownum <6;
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 5 in 0.147 seconds

	NAME	PHYRDS	PHYWRTS	READTIM	WRITETIM
1	D:\APP\ORACLE\ORADATA\CDB9\SYSAUX01.DBF	1399590	401418	716563	372411
2	D:\APP\ORACLE\ORADATA\CDB9\SYSTEM01.DBF	655271	108284	286313	37501
3	D:\APP\ORACLE\ORADATA\CDB9\STUDENTS05.DBF	387990	567613	89123	214395
4	D:\APP\ORACLE\ORADATA\CDB9\USERS02.DBF	50444	8866	36247	6749
5	D:\APP\ORACLE\ORADATA\CDB9\USERS01.DBF	47408	10872	27377	12113

8. *Index data query:*

The below query retrieves all the indexes in a database as per the table owner or in a specified table. We have considered only the tables which we are working on. The script helps the database administrator to see all the indexes on a particular table and drop them if not required to improve the performance.

```
SELECT * FROM all_indexes a
WHERE a.owner = 'DB111';
--- AND TABLE_NAME LIKE '%RATING%'; -----optional
```

SQL | All Rows Fetched: 8 in 0.374 seconds

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH	TABLESPACE_N
1	DB111	RESTAURANT_ID_PK_PART_HASH	NORMAL	DB111	RESTAURANT_PART_HASH	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
2	DB111	SYS_C_SNAP$_61RESTAURANT_ID_P	NORMAL	DB111	MVF	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
3	DB111	USER_ID_PK	NORMAL	DB111	USER_DATA	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
4	DB111	RESTAURANT_ID_PK	NORMAL	DB111	RESTAURANT	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
5	DB111	LOCATION_ID_PK	NORMAL	DB111	LOCATION	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
6	DB111	CUISINE_ID_PK	NORMAL	DB111	CUISINE	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
7	DB111	AMBIENCE_ID_PK	NORMAL	DB111	AMBIENCE	TABLE	UNIQUE	DISABLED	(null)	STUDENTS
8	DB111	MENU_ITEMS_PK	NORMAL	DB111	MENU_ITEMS	TABLE	UNIQUE	DISABLED	(null)	STUDENTS

Similarly, the below query can be used to delete all the indexes on a specified table by the administrator at a single go instead of removing each and every index separately.

```
SET SERVEROUTPUT ON
SET LINESIZE 100
SET VERIFY OFF
SET FEEDBACK OFF
PROMPT
DECLARE
CURSOR cu_idx IS
SELECT * FROM all_indexes a
WHERE a.table_name = ‘RATING’
AND a.owner = 'DB111';
BEGIN
DBMS_Output.Disable; DBMS_Output.Enable(1000000);
DBMS_Output.Put_Line('PROMPT');
DBMS_Output.Put_Line('PROMPT Droping Indexes on ' || Upper('&&1'));
FOR cur_rec IN cu_idx LOOP
DBMS_Output.Put_Line('DROP INDEX ' || Lower(cur_rec.index_name) || ';');
END LOOP; END;
```

9. *Dynamic memory monitoring:*

This script provides information about dynamic memory components. This helps the Administrator to understand the statistics of the memory components and can plan on how to overcome the challenges if they are reaching the maximum memory.

```
SELECT component,
ROUND(current_size/1024/1204) AS CURRENTSIZE_MB,
ROUND(min_size/1024/1204) AS MINSIZE_MB,
ROUND(max_size/1024/1204) AS MAXSIZE_MB
FROM v$memory_dynamic_components
WHERE current_size != 0
ORDER BY component;
```

Result:

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

SQL | All Rows Fetched: 7 in 0.108 seconds

	COMPONENT	CURRENTSIZE_MB	MINSIZE_MB	MAXSIZE_MB
1	DEFAULT buffer cache	286	286	612
2	PGA Target	490	490	490
3	SGA Target	912	912	912
4	Shared IO Pool	41	41	41
5	java pool	14	14	14
6	large pool	27	27	122
7	shared pool	531	204	531

10. *Performance indicators:*

Displays several performance indicators and comments on the value.

```
SET SERVEROUTPUT ON
SET LINESIZE 1000
SET FEEDBACK OFF

SELECT *
FROM   v$database;
PROMPT

DECLARE
  v_value  NUMBER;

  FUNCTION Format(p_value  IN  NUMBER)
    RETURN VARCHAR2 IS
  BEGIN
    RETURN LPad(To_Char(Round(p_value,2),'990.00') || '%',8,' ') || '  ';
  END;

BEGIN
```

Dictionary Cache Hit Ratio

```
SELECT (1 - (Sum(getmisses)/(Sum(gets) + Sum(getmisses)))) * 100
  INTO   v_value
  FROM   v$rowcache;

  DBMS_Output.Put('Dictionary Cache Hit Ratio       : ' || Format(v_value));
  IF v_value < 90 THEN
    DBMS_Output.Put_Line('Increase SHARED_POOL_SIZE parameter to bring value above
90%');
  ELSE
    DBMS_Output.Put_Line('Value Acceptable.');
  END IF;
```

Library Cache Hit Ratio

```
SELECT (1 -(Sum(reloads)/(Sum(pins) + Sum(reloads)))) * 100
       INTO   v_value
       FROM   v$librarycache;

       DBMS_Output.Put('Library Cache Hit Ratio          : ' || Format(v_value));
       IF v_value < 99 THEN
         DBMS_Output.Put_Line('Increase SHARED_POOL_SIZE parameter to bring value above
      99%');
       ELSE
         DBMS_Output.Put_Line('Value Acceptable.');
       END IF;
```

DB Block Buffer Cache Hit Ratio

```
      SELECT (1 - (phys.value / (db.value + cons.value))) * 100
       INTO   v_value
       FROM   v$sysstat phys,
              v$sysstat db,
              v$sysstat cons
       WHERE  phys.name  = 'physical reads'
       AND    db.name    = 'db block gets'
       AND    cons.name  = 'consistent gets';

       DBMS_Output.Put('DB Block Buffer Cache Hit Ratio  : ' || Format(v_value));
       IF v_value < 89 THEN
         DBMS_Output.Put_Line('Increase DB_BLOCK_BUFFERS parameter to bring value above
      89%');
       ELSE
         DBMS_Output.Put_Line('Value Acceptable.');
       END IF;
```

Latch Hit Ratio

```
SELECT (1 - (Sum(misses) / Sum(gets))) * 100
 INTO   v_value
 FROM   v$latch;

 DBMS_Output.Put('Latch Hit Ratio              : ' || Format(v_value));
 IF v_value < 98 THEN
  DBMS_Output.Put_Line('Increase number of latches to bring the value above 98%');
 ELSE
  DBMS_Output.Put_Line('Value acceptable.');
 END IF;
```

Disk Sort Ratio

```
SELECT (disk.value/mem.value) * 100
 INTO   v_value
 FROM   v$sysstat disk,
        v$sysstat mem
 WHERE  disk.name = 'sorts (disk)'
 AND    mem.name  = 'sorts (memory)';

 DBMS_Output.Put('Disk Sort Ratio              : ' || Format(v_value));
 IF v_value > 5 THEN
  DBMS_Output.Put_Line('Increase SORT_AREA_SIZE parameter to bring value below 5%');
 ELSE
  DBMS_Output.Put_Line('Value Acceptable.');
 END IF;
```

Rollback Segment Waits

```
SELECT (Sum(waits) / Sum(gets)) * 100
 INTO   v_value
 FROM   v$rollstat;

 DBMS_Output.Put('Rollback Segment Waits          : ' || Format(v_value));
 IF v_value > 5 THEN
  DBMS_Output.Put_Line('Increase number of Rollback Segments to bring the value below 5%');
 ELSE
  DBMS_Output.Put_Line('Value acceptable.');
 END IF;
```

Dispatcher Workload

```
SELECT NVL((Sum(busy) / (Sum(busy) + Sum(idle))) * 100,0)
 INTO   v_value
 FROM   v$dispatcher;

 DBMS_Output.Put('Dispatcher Workload             : ' || Format(v_value));
 IF v_value > 50 THEN
  DBMS_Output.Put_Line('Increase MTS_DISPATCHERS to bring the value below 50%');
 ELSE
  DBMS_Output.Put_Line('Value acceptable.');
 END IF;

END;
/

PROMPT

SET FEEDBACK ON
```

Result:

```
Elapsed: 00:00:00.188
Dictionary Cache Hit Ratio       :   99.77%  Value Acceptable.
Library Cache Hit Ratio          :   99.09%  Value Acceptable.
DB Block Buffer Cache Hit Ratio  :   98.56%  Value Acceptable.
Latch Hit Ratio                  :   99.94%  Value acceptable.
Disk Sort Ratio                  :    0.00%  Value Acceptable.
Rollback Segment Waits           :    0.01%  Value acceptable.
Dispatcher Workload              :    0.00%  Value acceptable.
```

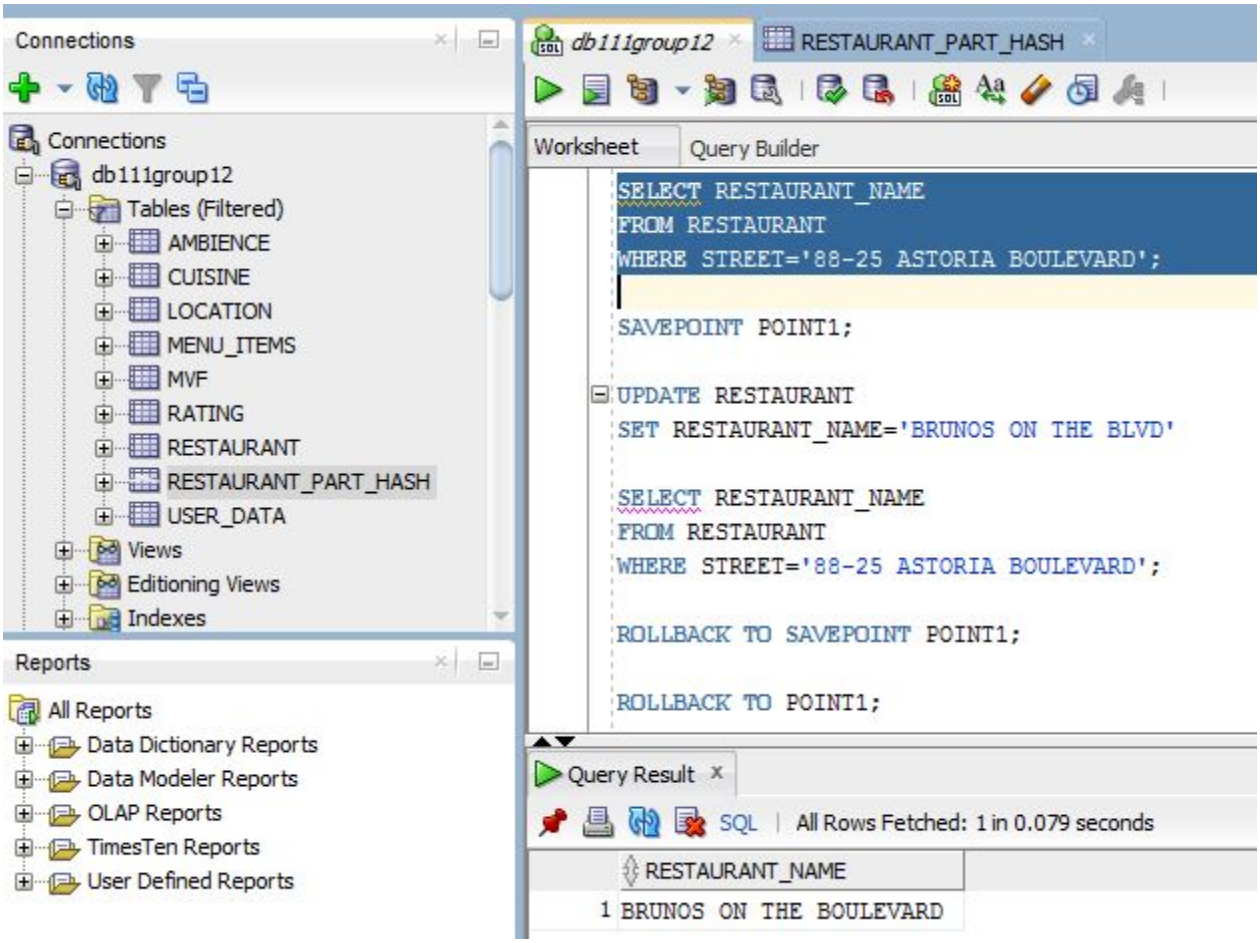
8 Database Security Policy

Our DBAs will adopt the following security policy to keep our database secure and well-protected -

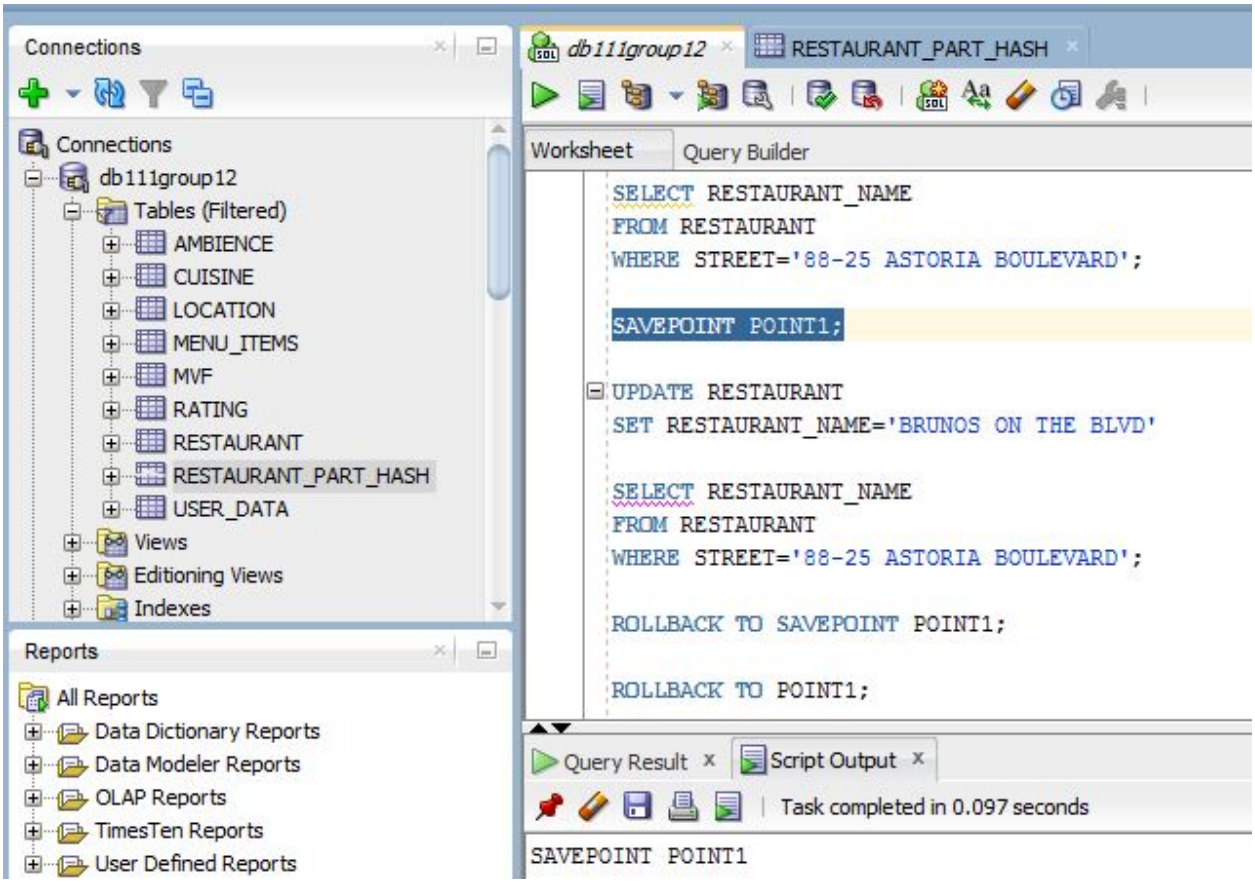
1. Encrypting Stored Files and Backups
2. Separating the Database and Web Servers
3. Using a Firewall
4. Lower system and database privileges; increase database security with Role Based Access Control
5. Minimizing the Use of 3rd Party Apps
6. Secure Your Data with an Effective Disaster Recovery Plan

9 Online Transaction Processing

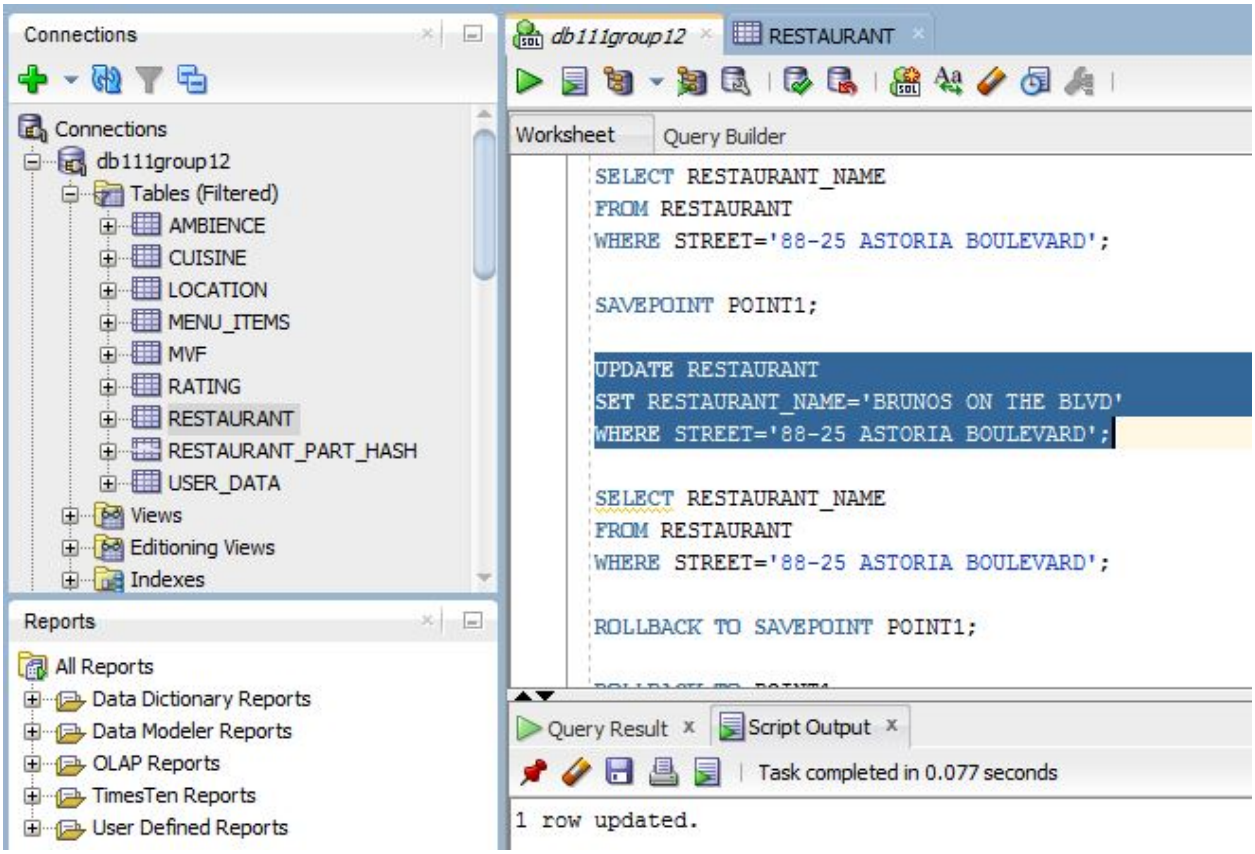
Below set of queries has been executed to showcase the SAVEPOINT and ROLLBACK features of the database integrity during online transaction processing.



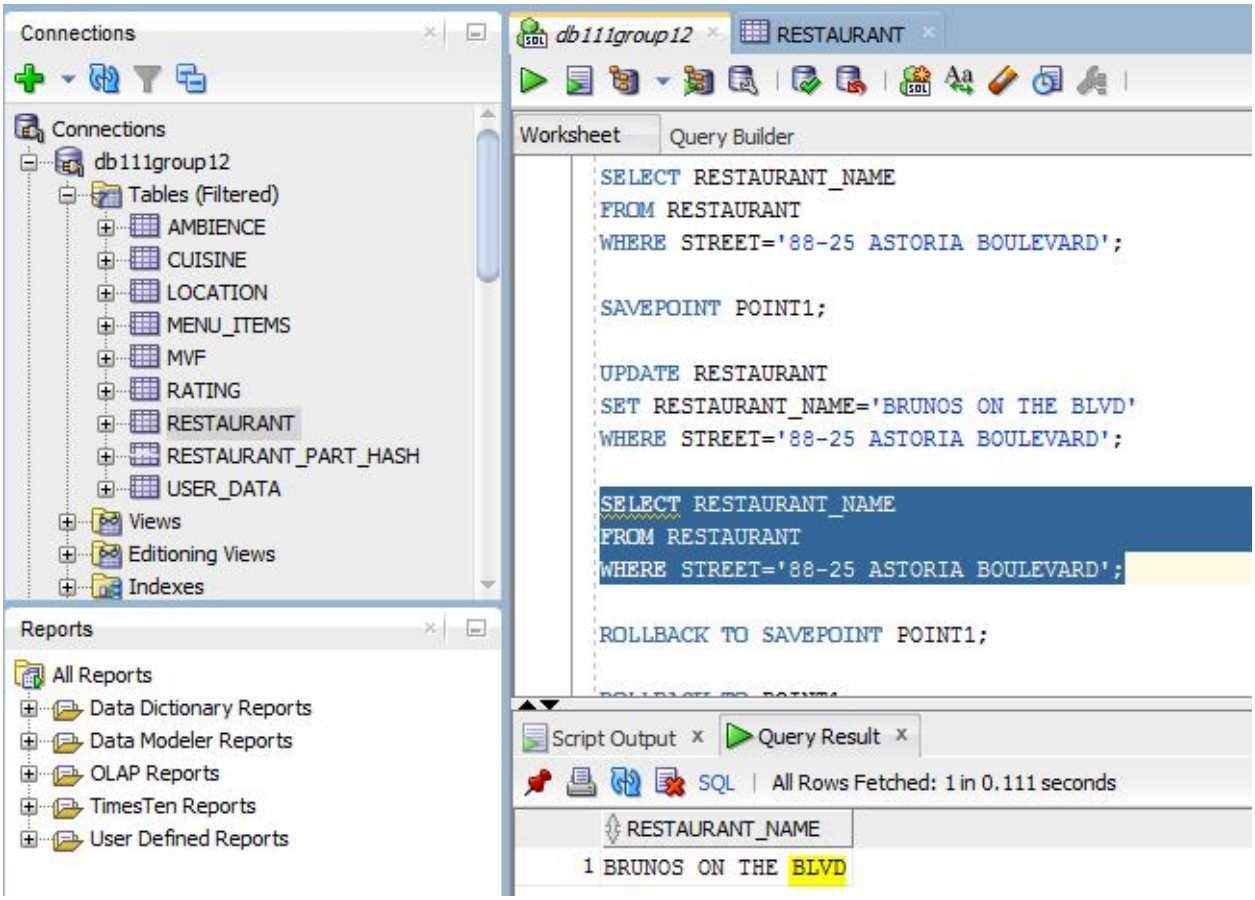
Taking a SAVEPOINT for the current status of RESTAURANT table.



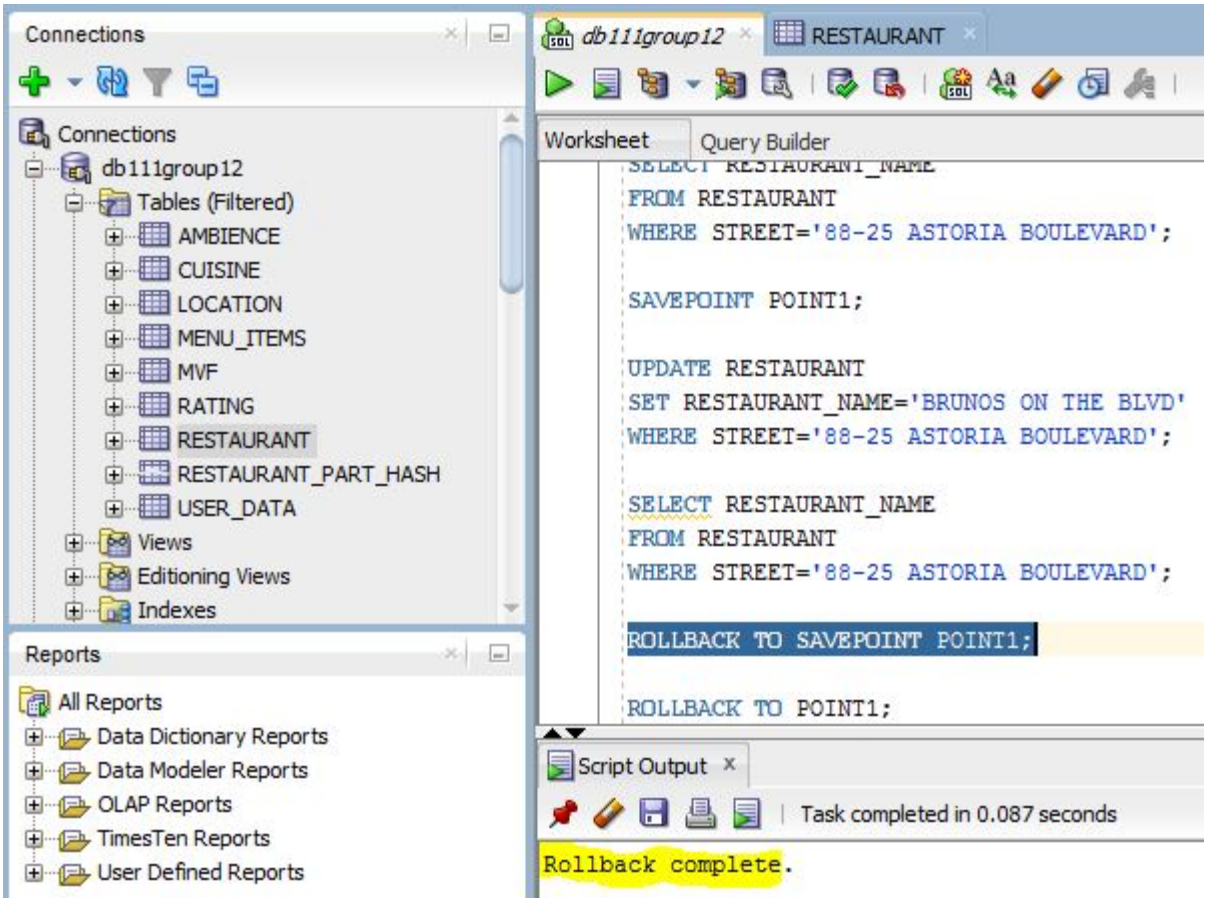
Updating the RESTAURANT_NAME of the RESTAURANT table for a particular STREET address.



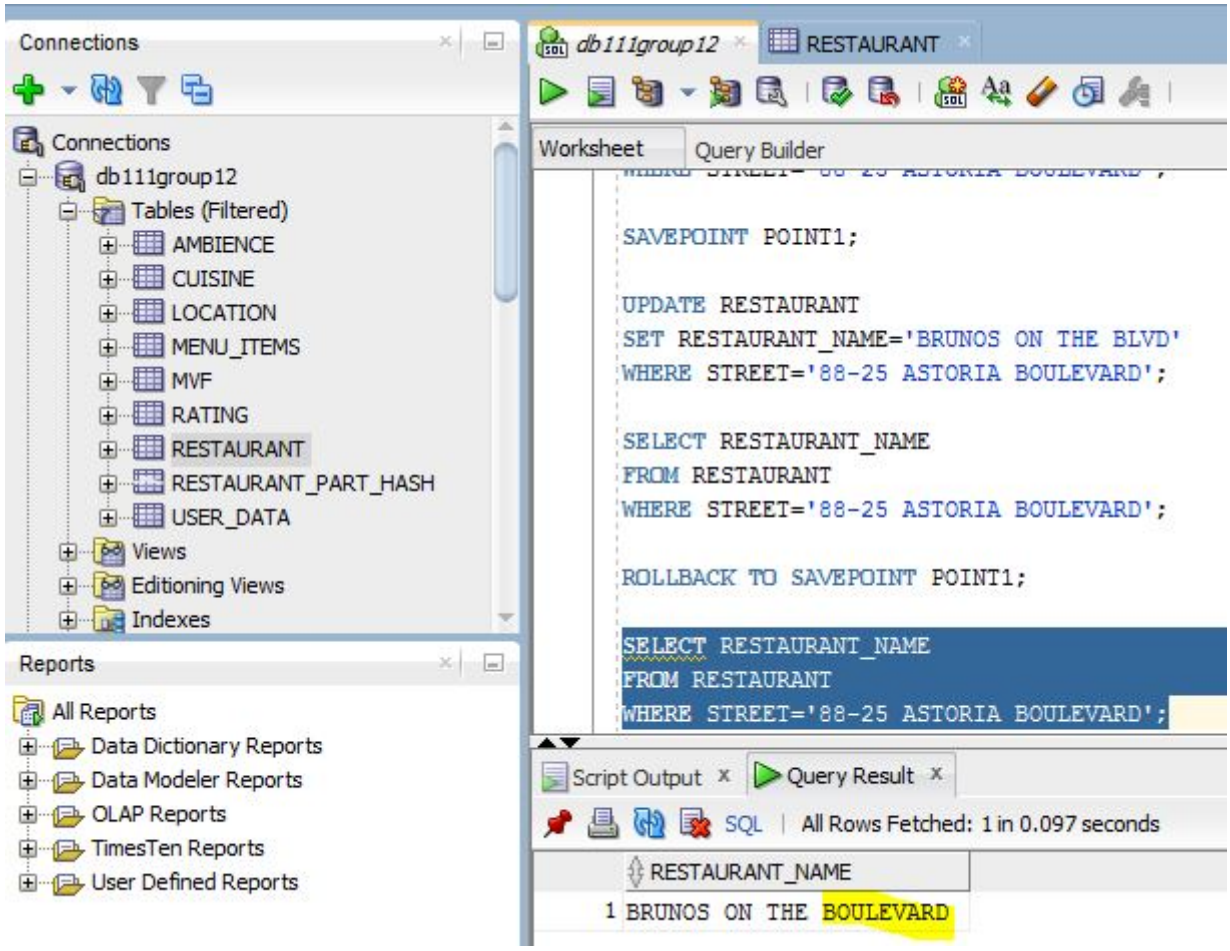
Displays the changed name (BLVD) of the RESTAURANT.



Rolling back the changes to the SAVEPOINT POINT1.



Name of the restaurant stored back to its original state (BOULEVARD)



Filename: ADBMSGroupProject.docx
Folder: /Users/jagpreet/Library/Containers/com.microsoft.Word/Data/Documents
Template: /Users/jagpreet/Library/Group Containers/UBF8T346G9.Office/User Content.localized/Templates.localized/Normal.dotm
Title:
Subject:
Author: Jagpreet Sethi
Keywords:
Comments:
Creation Date: 11/13/15 11:05 PM
Change Number: 2
Last Saved On: 11/13/15 11:05 PM
Last Saved By: Jagpreet Sethi
Total Editing Time: 1 Minute
Last Printed On: 11/13/15 11:05 PM
As of Last Complete Printing
Number of Pages: 60
Number of Words: 6,040 (approx.)
Number of Characters: 34,428 (approx.)