**DATA MART DESIGN**

IMAT5167 DATA WAREHOUSE DESIGN AND OLAP

ASSESSMENT I

SONIA AMARACHI OHANABA

P2559974

14th December 2020

LECTURER: PROF. YINGJIE YANG

Table of Contents

[**INTRODUCTION** 2](#_Toc58791843)

[**MAIN BODY OF REPORT** 3](#_Toc58791844)

[**DIMENSION SELECTION AND FACT IDENTIFICATION** 3](#_Toc58791845)

[**STAR SCHEMA** 4](#_Toc58791846)

[**LOGICAL RELATIONS (TABLES) AND GRANULARITY** 5](#_Toc58791847)

[**SQL FOR TABLE CREATION AND CONSTRAINTS** 5](#_Toc58791848)

[**DATA SOURCES MAPPING (USING DIAGRAMS)** 7](#_Toc58791849)

[7](#_Toc58791850)

[**SQL FOR ETL** 9](#_Toc58791851)

[CLIENT\_MT TABLE 9](#_Toc58791852)

[PRODUCTION\_MT TABLE 10](#_Toc58791853)

[TIME\_MT TABLE 11](#_Toc58791854)

[MT\_FACT TABLE 11](#_Toc58791855)

[**SQL FOR REQUIRED QUERIES (BOTH DATA MART & RELATIONAL MODEL)** 12](#_Toc58791856)

[**DATA MART SQL** 12](#_Toc58791857)

[**RELATIONAL MODEL SQL** 13](#_Toc58791858)

[**COMPARISON BETWEEN DATA MART AND RELATIONAL MODELS** 15](#_Toc58791859)

[**CONCLUSION** 15](#_Toc58791860)

[**APPENDIX** 16](#_Toc58791861)

[17](#_Toc58791862)

[Table 1 2](#_Toc57315310)

# **INTRODUCTION**

The purpose of this report is to produce a prototype ORACLE data mart for ticket sales in a company named Midlands Theatre. The company has branches of other smaller theatres found in the Midlands. They currently have a booking facility which allows their customers purchase a ticket in advance using the central booking system. This is done either via the company’s website, telephone, or an application. The booking system currently saves personal details of its customers which have been imputed into its server. However, the company wants to develop this data mart to fulfil some important analytical requirements. They are:

* Total spending in MT theatres for each client
* Names of clients who visited the theatres in the month July
* Names of all production directors with the highest total sales

The company has an ER diagram for their computerised booking system, this is made up of five (5) entities: Theatre, Production, Performance, Client, and Ticket Purchase. From the diagram, a Relational database has also been created and can be found in the table below:

|  |  |
| --- | --- |
| **Entity type** | **Contents** |
| Theatre | **Theatre#,** Name, Address, MainTelelphone |
| Production | **P#,** Title, Production Director, PlayAuthor |
| Performance | **Per#,** P#, Theatre#, pDate, pHour, pMinute, Comments |
| Client | **Client#,** Title, Name, Address, telNo, e-mail |
| TicketPurchase | **Purchase#,** Client#, Per#, PaymentMethod, Delivery Method, TotalAmount |

Table 1

In this report, the first thing that would be done is to analyse the DB design and identify the dimensions/facts for the data mart, secondly a star schema would be designed to identify corresponding Primary keys and Foreign keys. Thirdly, the granularity will be identified, and the star schema will be mapped to logical relations. SQL tables will be developed, and source data will be identified from OLTP. Fourthly, Extraction, transformation, and loading (ETL) will be implemented in the ORACLE SQL. Lastly, once all this is done, comparisons will be made to ascertain how the newly created data mart satisfies the analytic requirements of Midland Theatres.

# **MAIN BODY OF REPORT**

## **DIMENSION SELECTION AND FACT IDENTIFICATION**

The first step here is to look at the analytical requirements and deduce the relevant tables for this procedure. The first requirement is to list the total spending in MT theatres for each client. The relevant table for this would be ‘Client’ table. The ‘Client’ table would show each Client’s total spending. Hence, the first dimension.

The second requirement is to list the names of clients who visited the theatres in the month July, the relevant tables for this analysis would also be ‘Client’ as seen earlier and a new important dimension table ‘Time’ and specifically, the month, since we are required to list clients who visited only in the month of July. The attribute that will be used in the ‘time’ table is ‘month’ and would be the second dimension.

The last requirement is to list the names of all production directors with the highest total sales. The relevant table for this would be the ‘Production’ since it contains all data about production directors. Hence, this would be the third dimension.

Therefore, the selected three dimensions are as follows:

* Client\_mt for Client
* Time\_mt for Time
* Production\_mt for Production

For the facts table, Primary keys (PK) will be created and extracted from the dimensions table mentioned above and then inserted into the Facts table as Foreign keys (FK).

The following primary keys will be created, Client\_id, Time\_id, and Production\_id respectively from the Client, and Production tables. These primary keys will be attributes that will be transferred into the created Fact table. However, these will not be the only attribute of the fact table. Since two of the requirements would sales or total sales, another attribute would be added and gotten from the ops$yyang00.theatre. the name of this attribute would be Sales\_Amount as this has a very distinct relevance to two requirements: the first one which requires the list of clients and their *total spending* in MT theatres and the third one which requires the list of production directors of all productions with the *highest total sale*. The fact table would be:

MT\_FACT for the fact

## **STAR SCHEMA**

Next, a star schema will be designed in a diagrammatic form. This would be done with the facts and dimensions derived from the analysis above. The table will feature all three dimensions Client\_mt, time\_mt, and production\_mt of which three of the dimensions will originate from the originally created ER diagram. They will all be connected to a single fact table which has been named as mt\_fact, featured in the facts table will be an additional variable known as the Sales\_Amount and then the foreign keys. The star schema will illustrate the primary keys and foreign keys of the data mart in appropriate positions. The Star Schema can be illustrated below:

**Client\_mt**

**Client\_id (PK)**

Client#

Title

Name

Address

telNo

e-mail

**Production\_id (PK)**

P#

Title

ProductionDirector

PlayAuthor

**Production\_mt**

**MT\_FACT**

**Client\_id (FK)**

**Time\_id (FK)**

**Production\_id (FK)**

Sales\_Amount

**Time\_id (PK)**

Month

**Time\_mt**

The diagram above is the star schema. The next step is to identify the Primary keys and foreign keys. There are three different primary keys Client\_id, Time\_id and Production\_id each of them are found respectively on the subsidiary tables Client\_mt, Time\_mt and Production\_mt.

The foreign keys are only found in the Fact table ‘mt\_fact’ and they are primary keys derived from the subsidiary tables. The foreign keys are same as the Primary keys but only found in the mt\_fact table.

## **LOGICAL RELATIONS (TABLES) AND GRANULARITY**

The logical relations for the tables are the name of the dimension tables and fact tables, including their attributes such as primary keys and foreign keys. For the Client\_mt, there are seven (7) attributes, they include: Client\_id, Client#, title, name, address, telNo, and e-mail. For Production\_mt, there are five (5) attributes, they include: Production\_id, p#, Title, ProductionDirector, and PlayAuthor. For time\_mt, there are two (2) attributes, they include: Time\_id, and month. For the fact table which is mt\_fact, there are four (4) attributes, they include: client\_id, time\_id, production\_id, and sales\_amount.

The logical relations explained above can be simply mapped into tables in the below format:

Client\_mt (Client\_id, Client#, Title, Name, Address, TelNo, E-mail)

Production\_mt (Production\_id, p#, Title, ProductionDirector, PlayAuthor)

Time\_mt (Time\_id, Month)

Mt\_fact (Client\_id, Time\_id, Production\_id, Sales\_Amount)

In this report, considering the requirements for the tasks, the granularity for the Time table is Month because this is the smallest time span for the analytical requirements. The relevant attributes are illustrated in the Star Schema which has been explained above.

## **SQL FOR TABLE CREATION AND CONSTRAINTS**

After the completion of the logical relations, the next task is to create the tables for all dimensions and the central facts table respectively. The first table that will be created is the client\_mt table. The query that will be used in this table creation can be found below:

**CLIENT\_MT TABLE**

CREATE TABLE client\_mt(

Client\_id NUMBER (5) PRIMARY KEY,

client# NUMBER (5) NOT NULL,

client\_title VARCHAR2 (10),

client\_name VARCHAR2 (30) NOT NULL,

address VARCHAR2 (60),

telno CHAR (11),

email VARCHAR2 (50));

In the created table above, the primary key is the client\_id as was seen in the star schema, the next variable is the client# which has been assigned the attribute of a number, since it represents the clients’ unique number and has also been set to not null because of its importance. The next three variables which are the client\_title, client\_name and address have been assigned VARCHAR2 with their lengths as 10, 30 and 60 respectively. The name has been assigned not null as this a very important factor in the analysis of the requirements where we would like to identify each client’s total spending, hence it cannot be null. The telno is CHAR and with a length of 11. Lastly, the email has been set to 50 character legth and is VARCHAR2. (See Appendix 1:1)

**PRODUCTION\_MT TABLE**

CREATE TABLE production\_mt(

Production\_id NUMBER (5) PRIMARY KEY,

p# NUMBER (5) NOT NULL,

title VARCHAR2 (20) NOT NULL,

productiondirector VARCHAR2 (20),

playauthor VARCHAR2 (20));

In the created production table above, the primary key is production\_id and therefore cannot be null. However, since it is the primary key, it has automatically been set to NOT NULL. The maximum length of the variable is 5 and is a NUMBER variable. The next is the p#, this variable is a number and has been assigned a length of 5. It has also been set to NOT NULL. This is because the values in this variable cannot be NULL. The next variable is title and has been assigned VARCHAR2 with a maximum length of 20. This variable cannot also be null and has been set to NOT NULL. The last two variables are productiondirector and playauthor. They both have a maximum length of 20 and have been assigned VARCHAR2. (See Appendix 1:2)

**TIME\_MT TABLE**

CREATE TABLE time\_mt(

Time\_id NUMBER (5) PRIMARY KEY,

Month NUMBER (3) NOT NULL);

In the created table above, the primary key is time\_id and has been assigned NUMBER, this variable has been set to 5 numbers long (maximum). The last variable is the month variable and has been assigned NOT NULL, this is because the variable is very important in evaluating the requirement analysis. The length of this variable has been set to 3 because it is a date variable and will therefore follow this format, i.e 9 (for September). (See Appendix 1:3)

**MT\_FACT**

CREATE TABLE mt\_fact(

Client\_id NUMBER(5) CONSTRAINT fk33 REFERENCES Client\_mt,

Production\_id NUMBER(5) CONSTRAINT fk44 REFERENCES Production\_mt,

Time\_id NUMBER(5) CONSTRAINT fk55 REFERENCES Time\_mt,

Sales\_Amount NUMBER NOT NULL, CONSTRAINT pk88 PRIMARY KEY(Client\_id, Production\_id, Time\_id));

The fact table above is the last table that has been created. The client\_id, time\_id, and production\_id in this table are primary keys from the initially created dimensional tables. However, in this table they are foreign keys. They have a linking referential integrity known as the CONSTRAINT. The fk33, fk44, fk55 are all unique identifiers within ORACLE SQL. The REFERENCES statement draws a reference to the table where the foreign keys are located as primary keys. The last variable is the fact variable which is the sales\_amount and is CONSTRAINT to the primary keys in the subsidiary tables. It has been set to NULL as it cannot have a NULL VALUE. (See Appendix 1:4)

## **DATA SOURCES MAPPING (USING DIAGRAMS)**

Data source mapping will be shown using the original database. These are the original five tables from which the dimensional and fact tables were created. These tables serve as a direct source for the creation of the data mart. The tables are found in the ops$yyang00. Library and contain tables for Client, TicketPurchase, Theatre, Performance and Production. The data source mapping will be explained in detail using tables(diagrams). At the left hand side is the joined data source, while on the left is the data mart.

**JOINED TICKETPURCHASE & CLIENT**

Purchase#

Client#

Per#

PaymentMethod

DeliveryMethod

TotalAmount

Title

Name

Address

telNo

e-mail

**MT\_FACT**

Sales\_amount

## 

**CLIENT\_MT**

Client#

Client\_title

Client\_name

Address

telNo

e-mail

The diagram above illustrates how some elements(variables) of the fact and client table were derived. The first thing to do is to join the TicketPurchase table and Client table. This is because they both have a common key which is the Client#. This is a foreign key in the TicketPurchase table. Once this is joined, the client\_mt table will be derived from this joined table and the Total amount found in the TicketPurchase table will be used in the mt\_fact table but renamed as ‘sales\_amount’.

**EXTRACTION**

**JOINED PRODUCTION AND PERFORMANCE**

Per#

P#

PDate

PHour

PMinute

Comments

Title

ProductionDirector

PlayAuthor

**MT\_FACT**

Time\_id

Sales\_amount

**PRODUCTION\_MT**

P#

Title

ProductionDirector

PlayAuthor

**TRANSFORMATION**

**MT\_FACT**

Time\_id

Sales\_amount

**TIME\_MT**

Time\_id

month

**JOINED PRODUCTION AND PERFORMANCE**

Per#

P#

PDate

PHour

PMinute

Comments

Title

ProductionDirector

PlayAuthor

**PRODUCTION\_MT**

P#

Title

ProductionDirector

PlayAuthor

The diagram above illustrates the joining of two tables: Production and Performance. This is possible because the Performance table has a foreign key for Production (i.e p#). Once this is joined, the variables are derived from the joined table to create the Production\_mt table. At the left-hand side is the joined data source, while on the left is the data mart.

‘PDate’ variable is a very important variable because that is where the time dimension will be derived.

## **SQL FOR ETL**

After the data source mapping, the next thing to do is to populate the created tables using the data from the four tables and using ETL (Extract, transform and load). This will be done for all four created tables starting from the client\_mt, production\_mt, time\_mt and mt\_fact. Also the codes for this executions will be copied and pasted into this report.

### CLIENT\_MT TABLE

For the client\_mt table, the first thing to do is the ‘extraction’, this will be done by extracting data from the ops$yyang00.client library. The next stage is the transformation stage. For this, a sequence will be created for the primary key client\_id. The sequence will start with 1 and increase by 1 (this will be illustrated by code) and will be put into the client\_mt table by client\_seq.nextval. the first variable is client#, this does not need formatting as it is just a number. Next is the client\_title, this will be transformed into upper case and trimmed to remove extra characters from both behind and in front. The next attribute is client\_name, and the same thing that was done for the title attribute will be done for the name attribute. The next is address which is grouped into street, town and county. This will be transformed and inserted into one attribute called address in the created table. The last two variables are telno and email and will be transferred into the data mart. The next thing to do is to select all the attributes needed from ops$yyang00.client and insert them into the client\_mt table. The last part of the code involves linking the primary key in ops$yyang00.client with the foreign key ops$yyang00.ticketpurchase table. Below is the code for this:

CREATE SEQUENCE client\_seq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

The code below shows the loading part of the client ETL.

INSERT INTO client\_mt

SELECT client\_seq.nextval, client#, client\_title, client\_name, address, telno, email FROM

(SELECT DISTINCT client#, UPPER(TRIM(client\_title))client\_title, UPPER(TRIM(client\_name))client\_name, address, telno, email FROM

(SELECT client.client# client#, client.title client\_title, client.name client\_name, (client.street|| ' ' || client.town|| ' '|| client.county) AS address, client.telno telno, client.email email

FROM ops$yyang00.client, ops$yyang00.ticketpurchase

WHERE ops$yyang00.client.client# = ops$yyang00.ticketpurchase.client#));

Once this is done, sequence is created and data is loaded into the table. 3099 rows are inserted. (See Appendix 1:1)

### PRODUCTION\_MT TABLE

The next dimensional table is the production\_mt, just like in the previous tables, the first thing to do Is the extraction and source data is op$yyang00.production library. The next stage is the transformation stage. For this, a sequence will be created to get the surrogate key. The first variable is p# and is a number. Next attributes are title, productiondirector, and play author. They will be transformed into upper case and trimmed to remove extra characters from both behind and in front. The next thing to do is to select all the attributes needed from ops$yyang00.production and load them into the production\_mt table. The last part of the code involves linking the primary key in ops$yyang00.production with the foreign key in ops$yyang00.performance. Below is the code for this:

CREATE SEQUENCE p\_seq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

The code below shows the loading part of the production ETL.

INSERT INTO production\_mt

SELECT p\_seq.nextval, p#, title, productiondirector, playauthor FROM

(SELECT DISTINCT p#, UPPER(TRIM(title))title, UPPER(TRIM(productiondirector))productiondirector, UPPER(TRIM(playauthor))playauthor

FROM

(SELECT production.p# p#, production.title title, production.productiondirector productiondirector, production. Playauthor playauthor

FROM ops$yyang00.production, ops$yyang00.performance

WHERE ops$yyang00.production.p# = ops$yyang00.performance.p#));

Once this is done, sequence is created and data is loaded into the table. 78 rows have been inserted.

(See Appendix 1:2)

### TIME\_MT TABLE

The last dimensional table is the time\_mt, just like in the previous tables, the first thing to do Is the extraction and source data is op$yyang00.performance library. The next stage is the transformation stage. For this, a sequence will be created to get the surrogate key. The month attribute will be extracted from the pdate variable in op$yyang00.performance. Below is the code for this:

CREATE SEQUENCE time\_seq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

The code below shows the loading part of the time ETL.

INSERT INTO time\_mt

SELECT time\_seq.nextval, month FROM

(SELECT DISTINCT EXTRACT(month FROM pdate) month

FROM ops$yyang00.performance);

Once this is done, sequence is created and data is loaded into the table. 9 rows have been inserted. (See Appendix 1:3)

### MT\_FACT TABLE

The final table for ETL is the fact table. For the three foreign keys on this table (client\_id, production\_id and time\_id), extraction of these foreign keys will be gotten from the three tables where they are located as primary keys. The sales\_amount will be extracted from ops$yyang00.ticketpurchase where this is identified as total amount. For transformation, the foreign keys have already been transformed in the dimensional table. For the sales\_amount, the sum statement will be used and ops$yyang00.ticketpurchase ops$yyang00.performance will be used as data sources to link all the tables together. The table will finally be grouped by the primary keys of the dimensional tables. Below is the code for this:

INSERT INTO mt\_fact

SELECT client\_id, production\_id, time\_id, sales\_amount FROM

(SELECT client\_mt.client\_id, production\_mt.production\_id, time\_mt.time\_id,

SUM(totalamount) sales\_amount

FROM client\_mt, production\_mt, time\_mt, ops$yyang00.ticketpurchase, ops$yyang00.performance

WHERE Client\_mt.Client#= ops$yyang00.ticketpurchase.Client#

AND Production\_mt.P#=ops$yyang00.performance.P#

AND EXTRACT(Month FROM ops$yyang00.performance.pDate)=Time\_mt.Month

AND ops$yyang00.ticketpurchase.Per#=ops$yyang00.performance.Per#

GROUP BY Client\_mt.Client\_id, Production\_mt.production\_id, Time\_mt.Time\_id);

Once this is done, data is loaded into the table. 4792 rows have been inserted.

(See Appendix 1:4)

## **SQL FOR REQUIRED QUERIES (BOTH DATA MART & RELATIONAL MODEL)**

Now that the ETL for all the tables have been executed, the next step is to analyse all three requirements. They are:

* Total spending in MT theatres for each client
* Names of clients who visited the theatres in the month July
* Names of all production directors with the highest total sales

### **DATA MART SQL**

The first requirement is to analyse the total spending in MT theatres for each client. This will be done by using the attributes in the client\_mt table as an identification for each client and secondly the total amount will be derived from the sales\_amount attribute in the mt\_fact table. Below is the code for this:

SELECT client\_mt.client\_id, client\_mt.client\_name, SUM(sales\_amount)

FROM client\_mt, mt\_fact

WHERE client\_mt.client\_id = mt\_fact.client\_id

GROUP BY client\_mt.client\_id, client\_mt.client\_name

ORDER BY client\_mt.client\_name ASC;

After the execution of the code, the table returned 3099 rows, showing the names of each client and their total spending. Example, for Alex M12, his total sales amount is 25.5(See Appendix 1:5)

For the second requirement which is: to list the names of all clients who visited MT theatres in July, this will also be done by using the attributes in client\_mt table as an identification for each client to be listed and the month attribute in the time dimension table, specifically July. Below is the code for this:

SELECT client\_mt.client\_id, client\_mt.client\_name

FROM client\_mt, mt\_fact, time\_mt

WHERE client\_mt.client\_id = mt\_fact.client\_id

AND time\_mt.time\_id = mt\_fact.time\_id

AND time\_mt.month = 7

GROUP BY client\_mt.client\_id, client\_mt.client\_name

ORDER BY client\_mt.client\_name ASC;

After the execution of the code, the table returned 553 rows showing each client that visited each mt theatres in the month of July which is (7) (see Appendix 1:6)

For the last requirement which is: to list the production directors of all productions with the highest total sale, attributes from the production table will be used to identify the production directors and then the sales\_amount form the fact table will be used to illustrate those with the highest amount. Below is the code for this:

SELECT productiondirector

FROM (SELECT production\_mt.production\_id, production\_mt.productiondirector, SUM(sales\_amount) sales\_amount

FROM production\_mt, mt\_fact

WHERE production\_mt.production\_id = mt\_fact.production\_id

GROUP BY production\_mt.production\_id, production\_mt.productiondirector)

WHERE Sales\_amount = (SELECT MAX(sales\_amount)

FROM (SELECT production\_mt.production\_id, production\_mt.productiondirector, SUM(sales\_amount) sales\_amount

FROM production\_mt, mt\_fact

WHERE production\_mt.production\_id = mt\_fact.production\_id

GROUP BY production\_mt.production\_id, production\_mt.productiondirector));

From the output, the table returned 2 rows and it is evident that only Jethro and Adam had the highest total sales of production. (See Appendix1:7)

### **RELATIONAL MODEL SQL**

For the first requirement, the data sources will be derived directly from the source OLTP, which is the ops$yyang00.library. below is the code for the analysis of the list of each client and his total spending in MT Theatres. This is the code below:

SELECT ops$yyang00.client.name, SUM(totalamount)

FROM ops$yyang00.client, ops$yyang00.ticketpurchase

WHERE ops$yyang00.client.client# = ops$yyang00.ticketpurchase.client#

GROUP BY ops$yyang00.client.name

ORDER BY ops$yyang00.client.name;

After the execution of the code, the table returned 3099 rows, showing the names of each client and their total spending. (See Appendix 1:8). This is the same as the output from the data mart.

For the second requirement, the code below will be used to derive the analysis for all clients who visited MT theatres in *JULY* and the data for this will be derived from the source OLTP which is the ops$yyang00.library. this is the code below;

SELECT ops$yyang00.client.name, extract(month FROM ops$yyang00.Performance.pdate) AS month

FROM ops$yyang00.TicketPurchase

LEFT JOIN ops$yyang00.client ON ops$yyang00.TicketPurchase.client# = ops$yyang00.client.client#

LEFT JOIN ops$yyang00.Performance ON ops$yyang00.TicketPurchase.per# = ops$yyang00.Performance.per#

GROUP BY ops$yyang00.client.name, extract(month FROM ops$yyang00.Performance.pdate)

HAVING extract(month from ops$yyang00.Performance.pdate) = 7

ORDER BY ops$yyang00.client.name;

After the execution of the code, the table returned 553 rows, showing the names of each clients who visited the theatre only in July(7). (See Appendix 1:9). This is the same as the output from the data mart.

For the third requirement, the code below will be used to derive the analysis for all the production directors of all productions with the highest total sales. The data for this will be derived from the source OLTP which is the ops$yyang00.library.

SELECT productiondirector

FROM (SELECT p.p#, p.productiondirector, SUM(totalamount) totalamount

FROM ops$yyang00.production p, ops$yyang00.ticketpurchase tp, ops$yyang00.performance pr

WHERE PR.PER# = TP.PER#

AND P.P# = PR.P#

GROUP BY p.p#, p.productiondirector)

WHERE

totalamount = (SELECT max(totalamount)

FROM (SELECT p.p#, p.productiondirector, SUM(totalamount) totalamount

FROM ops$yyang00.production p, ops$yyang00.ticketpurchase tp, ops$yyang00.performance pr

WHERE PR.PER# = TP.PER#

AND P.P# = PR.P#

GROUP BY p.p#, p.productiondirector));

After the execution of the code, the table returned 2 rows, showing the 2 production directors with the highest production sales, Jethro and Adam. (See Appendix 1:10). This is the same as the output from the data mart.

## **COMPARISON BETWEEN DATA MART AND RELATIONAL MODELS**

**COMPARISON FOR REQUIREMENT 1:** The two queries both from the relational model and data mart produced the same output however with different SQL query commands. They both produced an output showing each Client and the corresponding sum of the ticket sales. In both tables, 3099 rows were inserted. (see Appendix 1:5 and 1:8).

**COMPARISON FOR REQUIREMENT 2:** The two queries both from the relational model and data mart produced the same output however with different SQL query commands. They both gave an output with 553 rows showing customers who visited the theatre only in the month of July. (See Appendix 1:6 and 1:9)

**COMPARISON FOR REQUIREMENT 3:** The two queries both from the relational model and data mart produced the same output however with different SQL query commands. They both gave an output with 2 rows showing the production directors with the highest total sales of production. (See Appendix 1:7 and 1:10).

Overall, it was easier to create a data mart and analyse the requirements from it, this is because the data mart narrows the requirements and does not give as much ambiguity as analysing the requirements directly from the OLTP database (unwanted data). The OLTP database query command produced the same output. however, this was achieved after so much query manipulation.

# **CONCLUSION**

In conclusion, this report shows the identification of dimensions and fact tables for the MT theatre data mart, where three dimensional tables (client\_mt, production\_mt and time\_mt) were selected and one fact table (mt\_fact) from the ops$yyang00.library. A star schema was also produced and illustrates all the foreign keys and the primary keys in the data mart. Relevant attributes of the tables were highlighted and the granularity was described. The star schema was mapped to logical relations and from this, the SQL tables were created for all four tables. (client\_mt, production\_mt, time\_mt and mt\_fact). The client\_mt table had 3099 rows, production\_mt had, time\_mt had, and the mt\_fact table had .

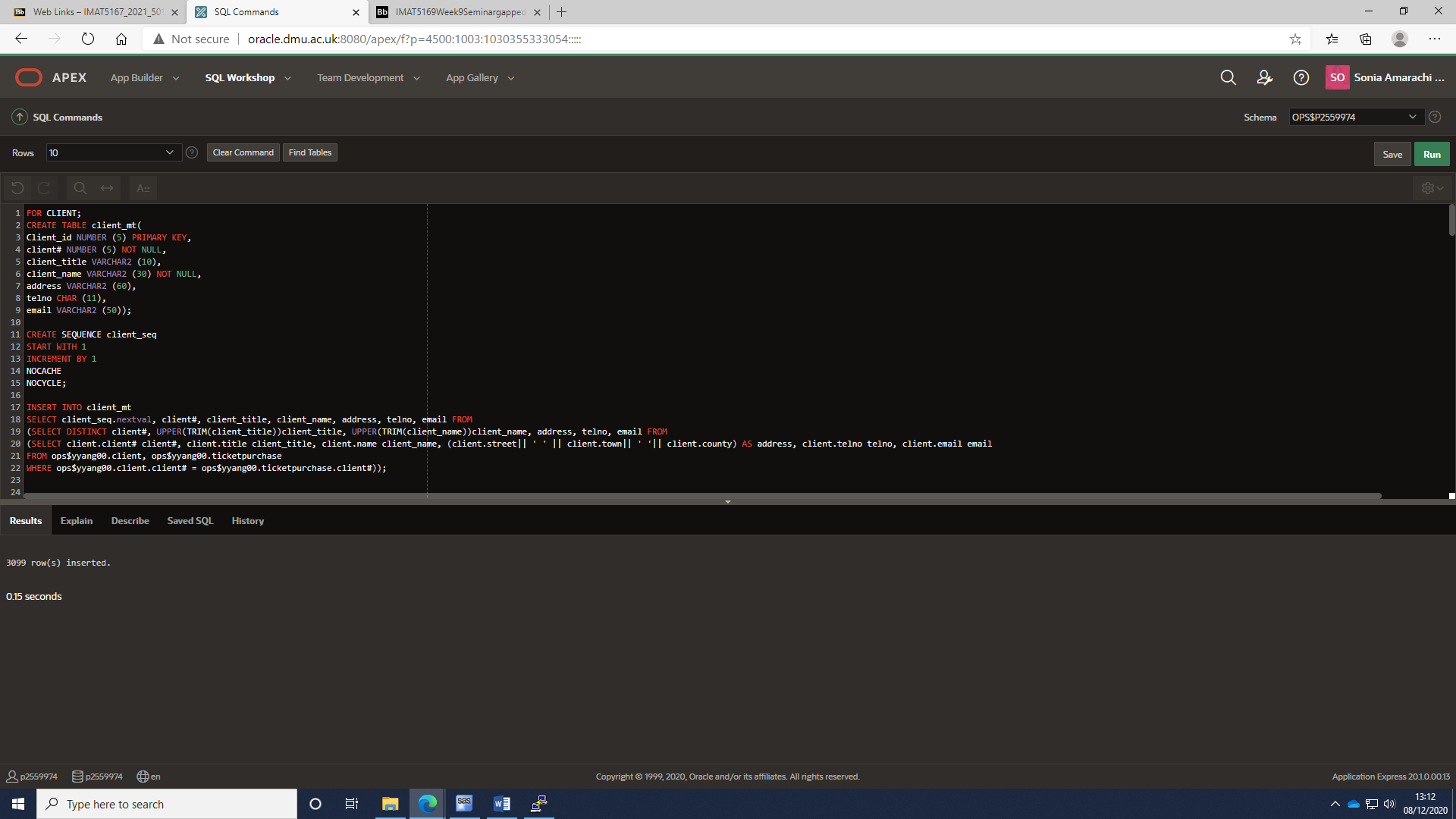
Diagrams were used to identify source data from the OLTP database which is ops$yyang00.library, from this, interconnections were made and a proper description of the development of all tables were explained. After this, the created tables were ready to be loaded with data. This was done using the insert statement in SQL.

After the creation of the four tables, the tables were ready for ETL (Extraction, Transformation and Loading). This was done firstly by creating sequences for each individual table (apart from the mt\_fact) table and loading the created tables with data.

Next, the requirements were analysed both from my created data mart and the OLTP database after which, comparisons were made. From every observation it was easier to create data marts and analyse the requirements from it, this is because the data mart narrows the requirements and does not give as much ambiguity as analysing the requirements from the OLTP database. The OLTP database produced the same output however, this was achieved after so much query manipulation.

Lastly, the created data mart satisfies the analysis of the given three requirements. With data mart, it is evident that it makes business processes a whole lot faster and allows easier access to specific information within the company rather than having to filter through a lot of data or irrelevant data with regards to the given business requirements.

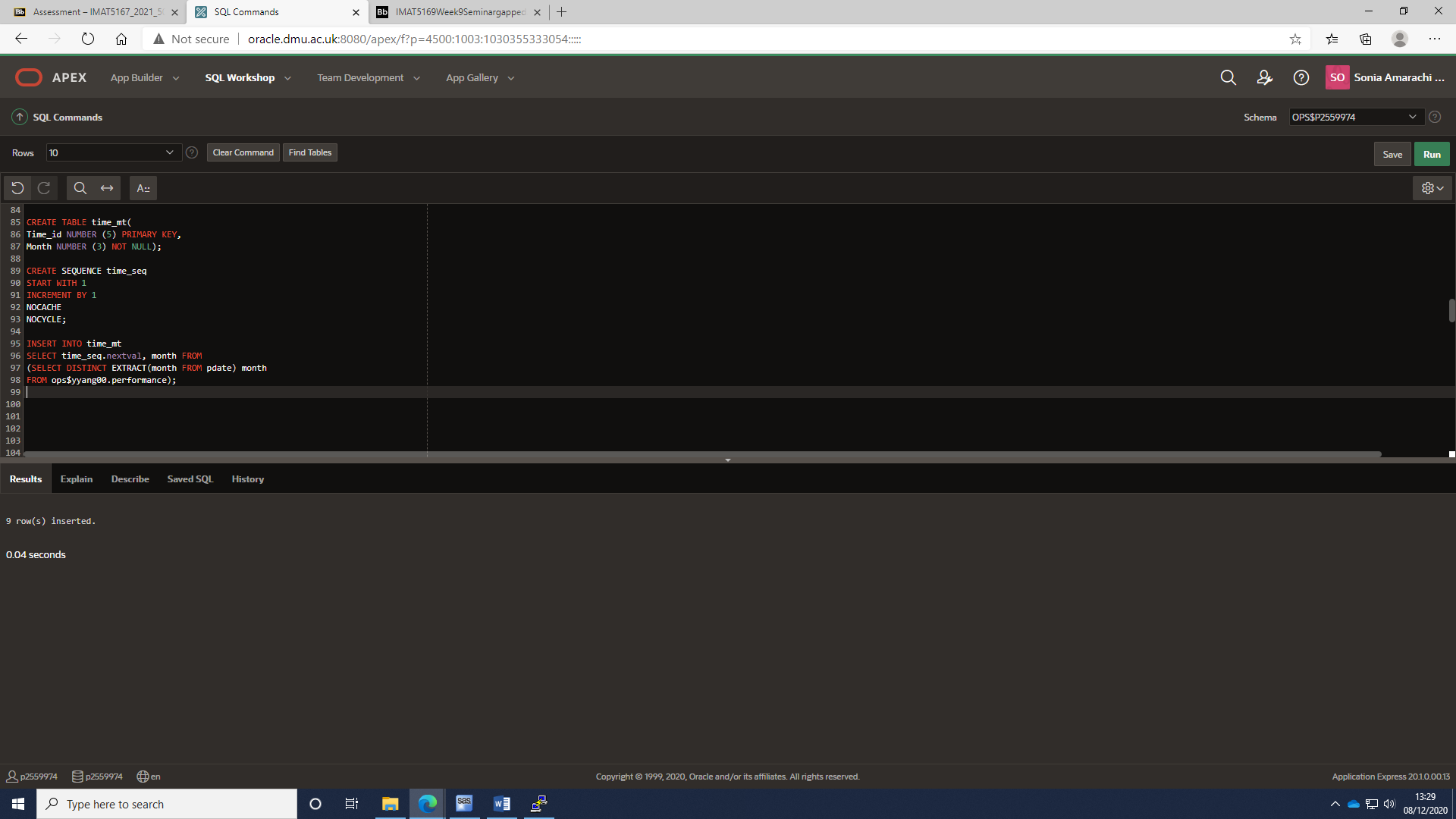
# **APPENDIX**



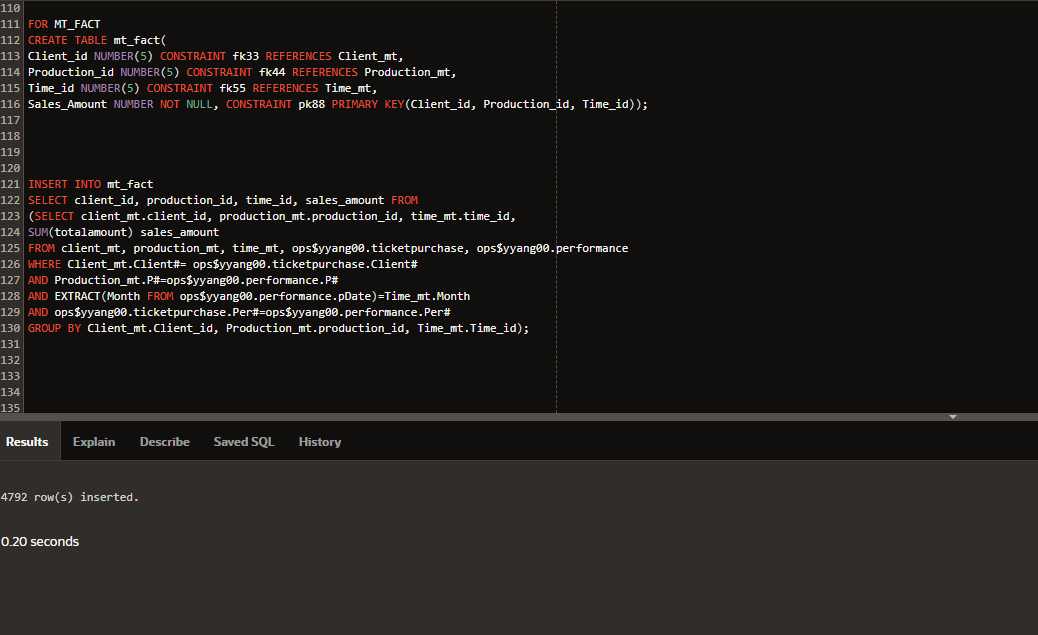
Appendix 1:1

## 

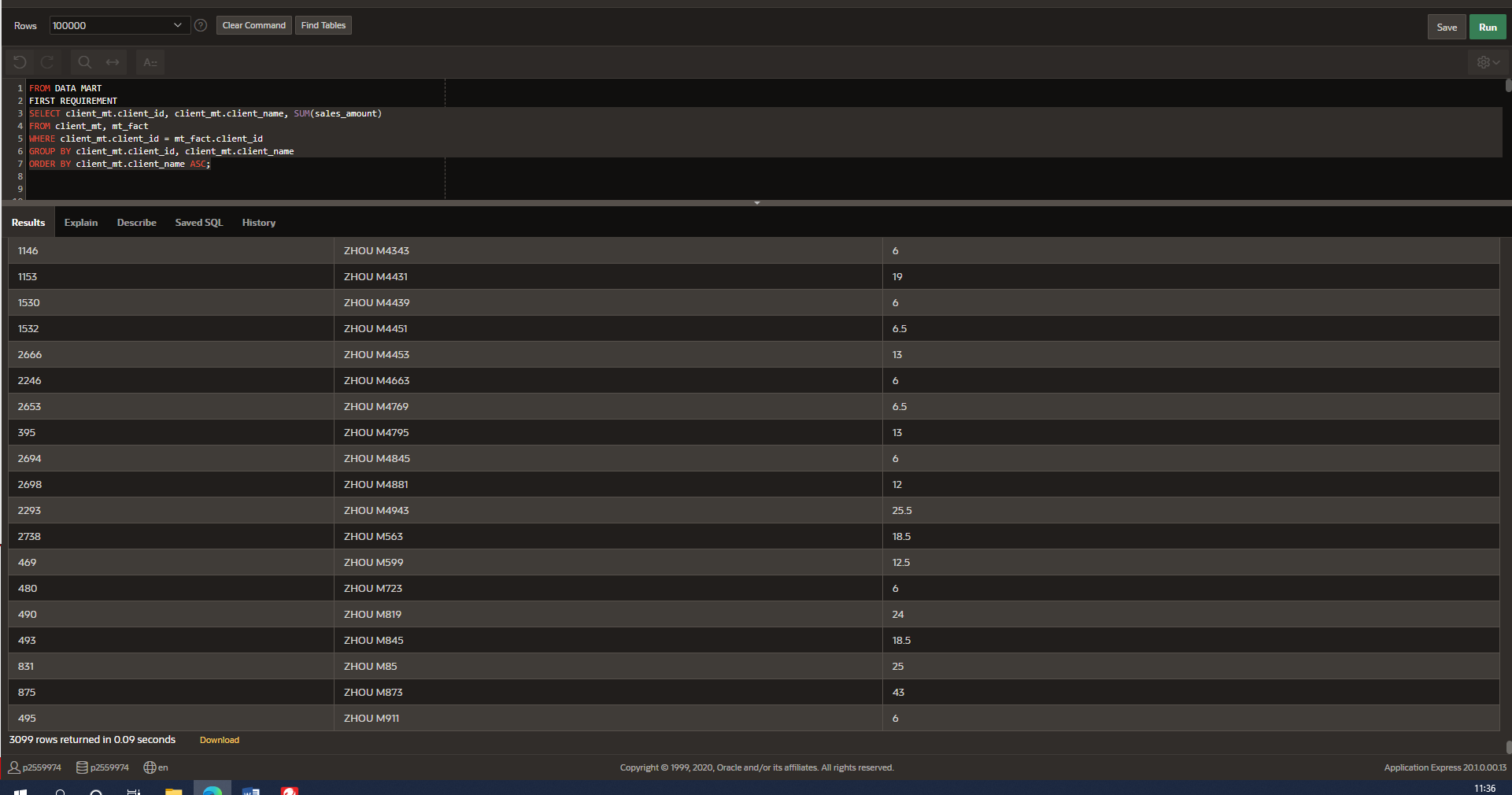
Appendix 1:2



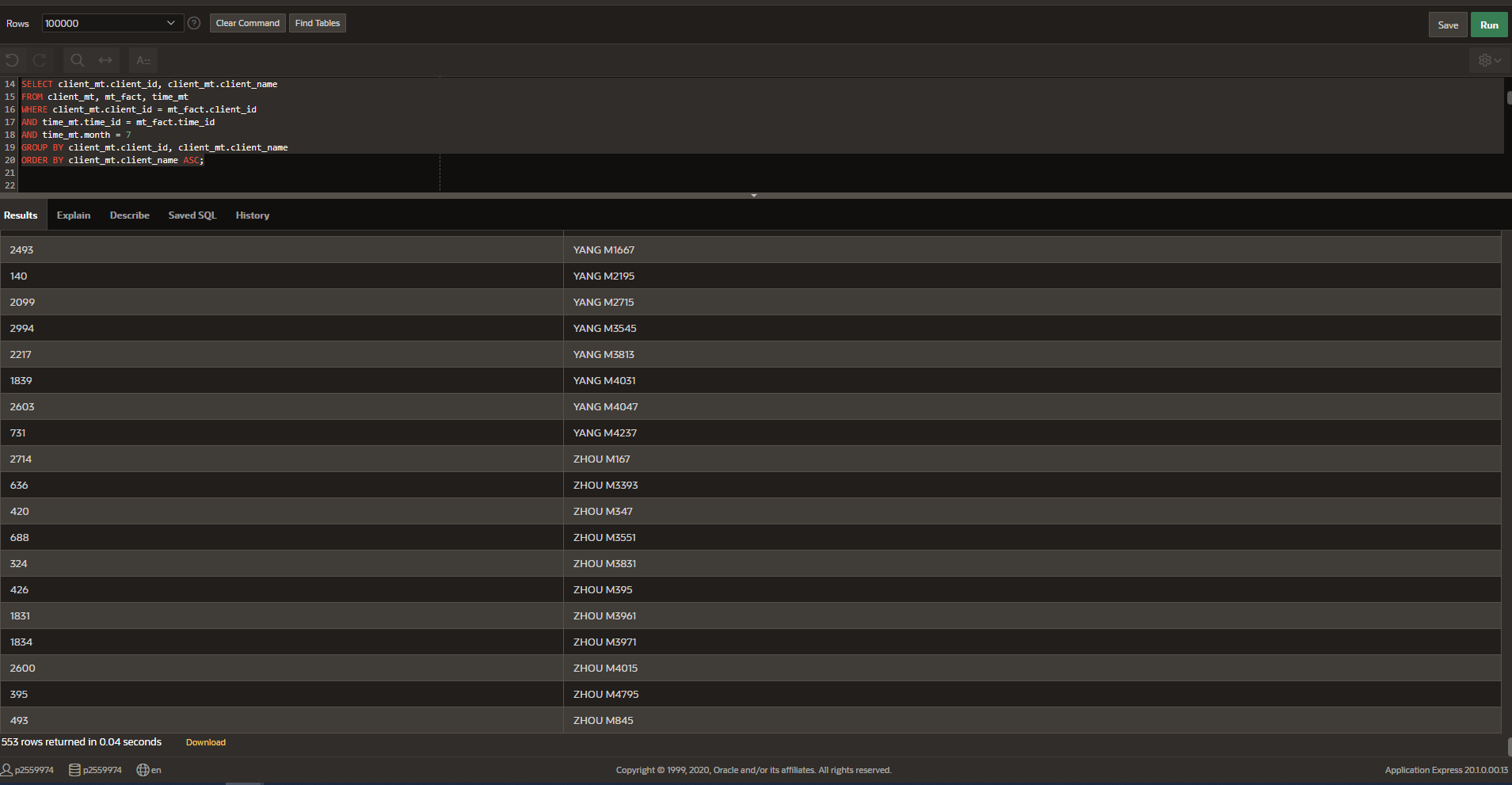
Appendix 1:3



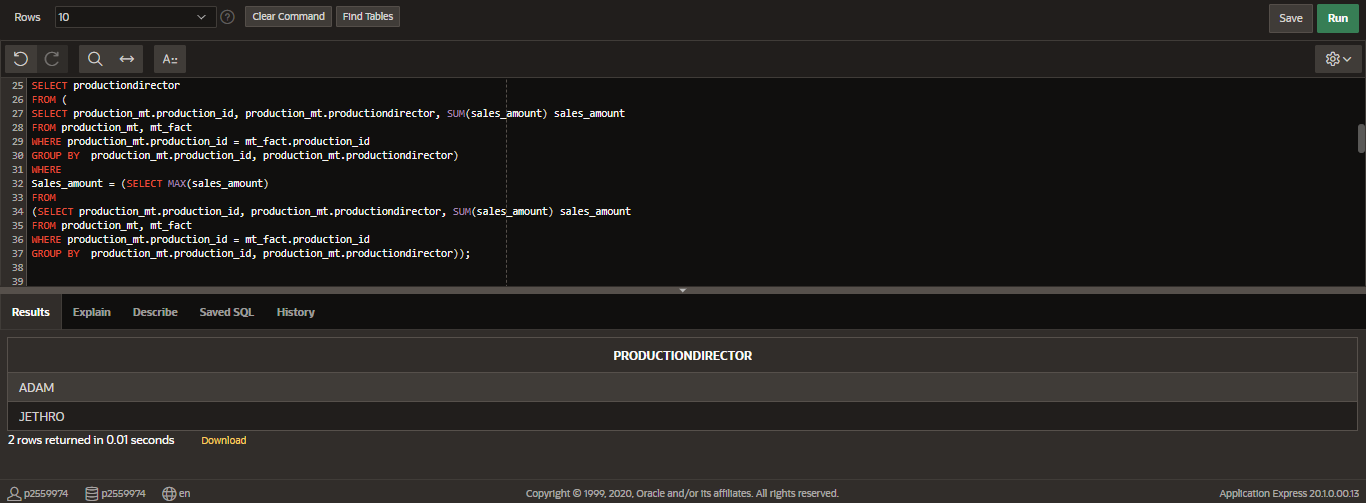
Appendix 1:4



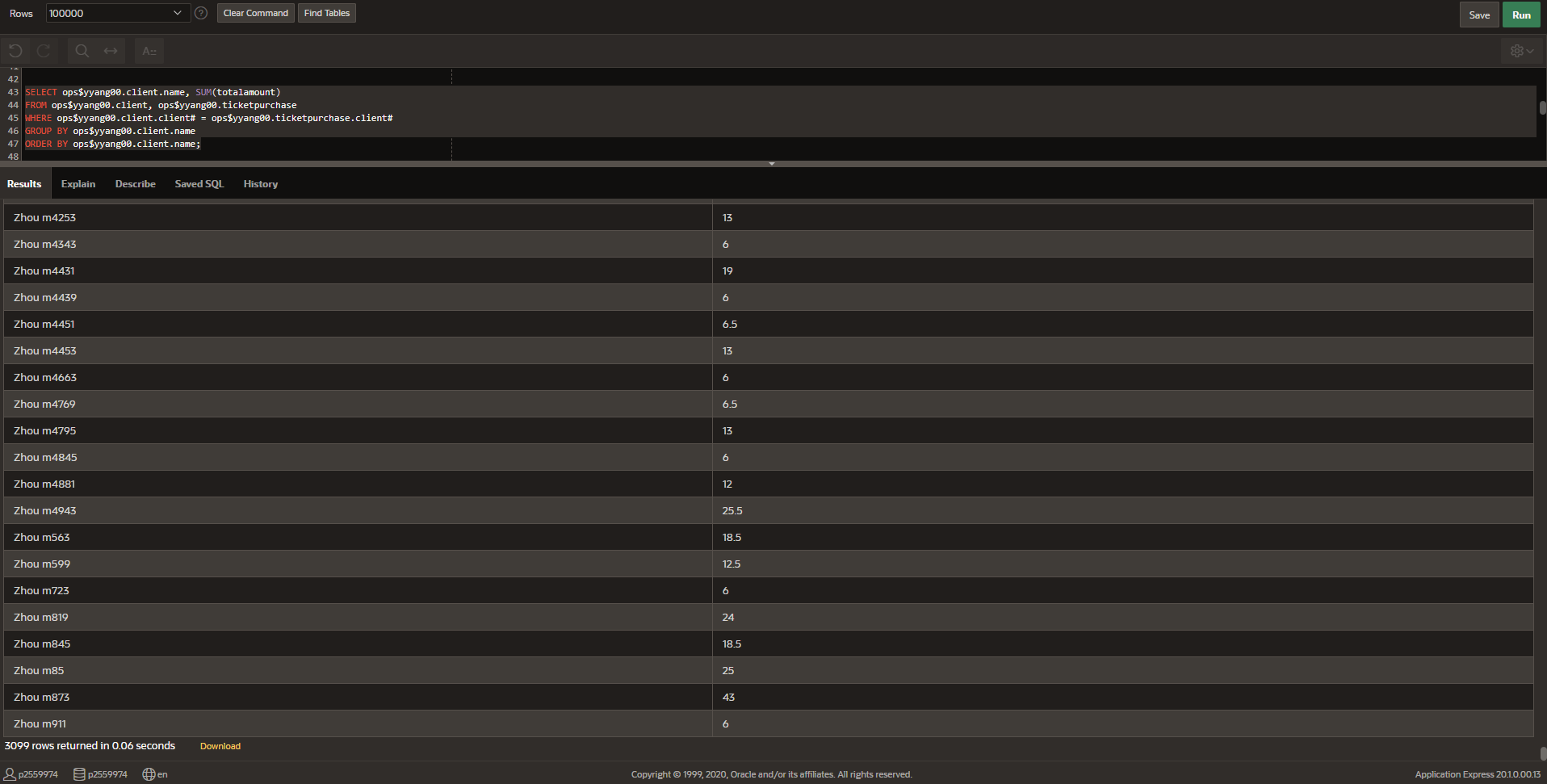
Appendix 1:5

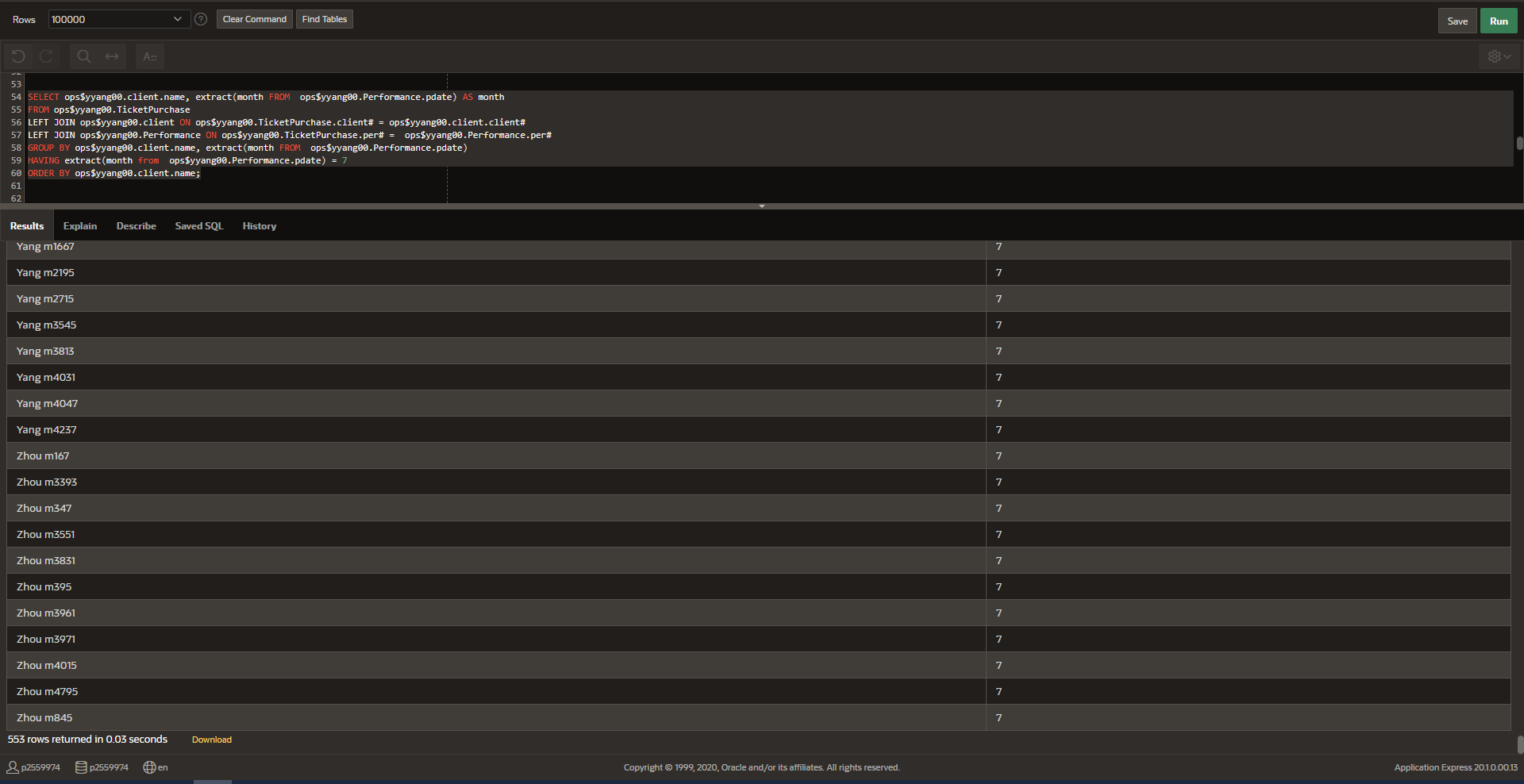


Appendix 1:6

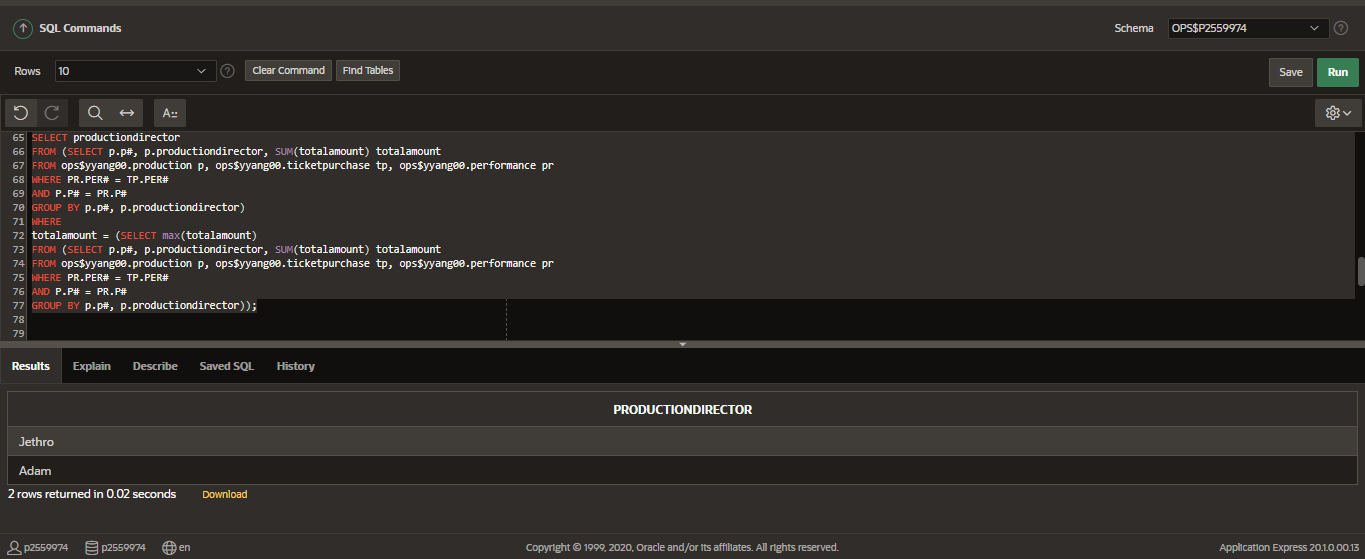


Appendix 1:7

  
Appendix 1:8



Appendix 1:9



Appendix 1:10