**Database Project – Final Report (Parts 1 to 5)**

# Devanshu Matlawala

Database Management Systems - Professor Fernandez

George Washington University

March 11, 2016

# **Part 1**

# **Overview**

# This Database Management project focuses on the creation of a database from the conceptual model design to a fully functional database including primary and foreign key constraints, triggers for special company rules, history tracking and auditing, and indices to improve query performance. The database will be developed using the Oracle SQL Modeler and SQL Developer Applications. This database will store and manage, client and event data, and will track individual log events responsible for event booking, costing, and any other charges. This project is based on a fictional catering company. I will refer to this company as the Catering Company ABC although this is not the organization’s real name. A database will be created to store and manage ABC’s Catering projects. Information about events along with location information, menu details, and artist details are stored in the catering company’s database. Clients will request the catering company to organize different events, by giving details about the event type, number of guests, menu details, and special requirements for artists like photographer, musician, florist, and entertainer. A database will be created to store all these things including client’s information, artist’s information, and cost for catering an event. The database for Catering Company ABC will be used to decide a suitable location type for the event, through the information provided by the client (for instance, number of guests), and generate bill for the client by adding the costs given by the artists and catering costs. The proposed database will assist the company’s management for the above, and keep track of any changes made in any particular event for more efficiency.

**Purpose and Objectives**

# The Catering Company ABC’s mission is to provide the best catering service to their clients. To provide a very good service, they want to implement a database, which will store and manage their client, event, location, menu, and artist information. There are some specific conditions that the catering company wants to impose on the client requests.

# The conditions are as follows:

# A client can choose to request for an event to be held in the catering hall provided by the catering company themselves. The restriction on this is that the number of guests should be greater than 150, however, less than or equal to 200. However, this is not a compulsion and a client can choose an offsite location as well. If in case, a client chooses the catering hall for their event, but their number of guests is less than 150 or exceeds 200, then a trigger will be fired in order to change the location type of the event from ‘catering hall’ to ‘others’.

# If a client wishes to have a photographer, a florist, an entertainer, or a musician at their event, then he/she can request the catering company for the same and then the catering company will hire the required artist. However, the company will charge the client 10% more than what the artists charge the company, i.e., the artists cost.

# Any update in the event details should be tracked. There should be an audit on these changes, and a log should be created, to track the person, who did it, and the time it happened. The database will need to store all the information about the event. The data includes client’s information, the location’s information, the different types of menu, and different types of artists like photographer, florist, entertainer, and musician. We must also make sure that we have the contact numbers of all the artists and the client in case of any last minute changes. To follow the rules of the company we should include some constraints like checking the number of guests, to determine the event location. Other details on company rules, constraints, and triggers will be provided in future design documents.

**Data Modeling Tools**

Oracle SQL Developer Data Modeler was used to design the Data Flow Diagram, Entity Relationship Diagram, and the Relational Model.

**Database Software**

The database will be based on Oracle 11g software.

Oracle SQL Developer will be used to access the database.

**Additional Hardware and Software Details**

The Catering Company ABC’s project database will be hosted on a server running the Linux operating system located at the George Washington University. **For offsite access, the server will be accessed using** [**https://vpn.gwu.edu**](https://vpn.gwu.edu)**.** Oracle SQL Developer will be used to access the database.

**Part 2**

**Brief Project Goal**

A database will be created for a Catering Company ABC to store and manage their Event information, Client Information, Client Requirements, Artist Information, Location Information, and Menu Details.

**Business Direction Information**

Assumption: ABC would like the new Event management database to be developed using Oracle Database version 11g.

Problem: ABC management has come up with some conditions that they want to implement using triggers and logs in a database, to avoid any failures in the company. The new conditions as mentioned above are, their catering hall will only be used if the number of guests attending the event are between 150 and 200. They are also planning to charge the client 10% more on the artist costs. They also want to track every change made by anyone in the database so that they don’t miss anything.

Business Objective: To change the Event location from the Catering hall to Others, if the number of guests attending an event is greater than 200 or less than 150.

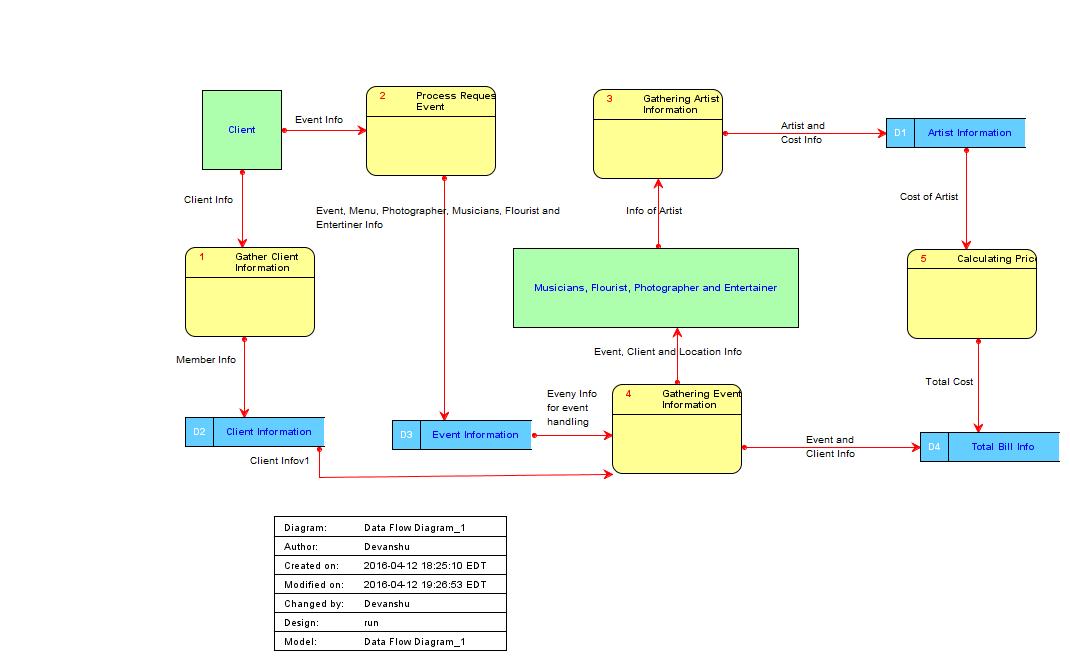
Business Objective: To make sure that they charge the client 10% more than what they are paying the artists for the event.

Key Performance Indicator: Here, the Event location should be changed from catering hall to others, if the number of guests attending an event is greater than 200 or less than 150.

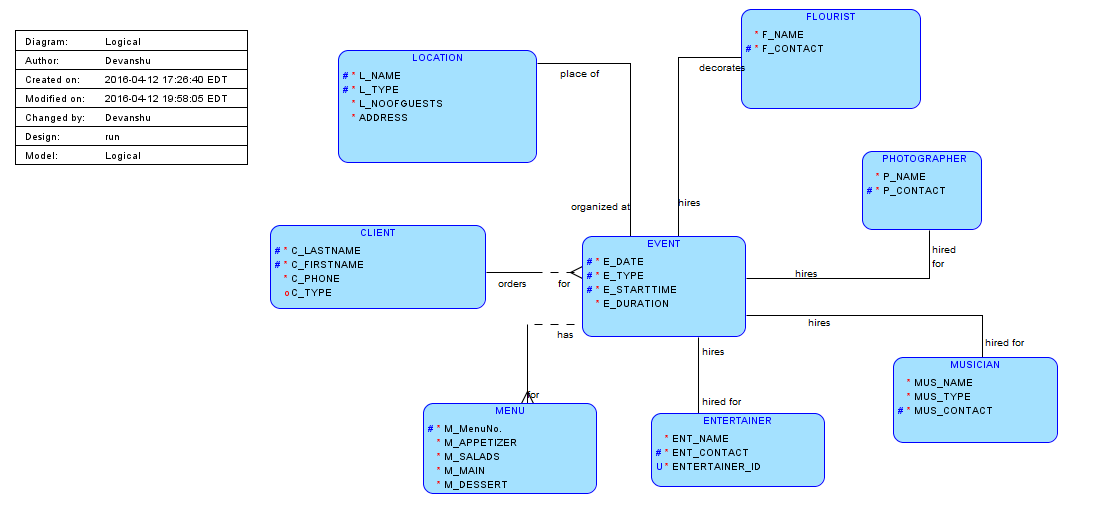
Key Performance Indicator: Every change in the event request is tracked and audited.

Critical Success Factor: The change from Catering Hall to Other is tracked, when the condition is triggered.

Critical Success Factor: The bill of the client is exactly 10% more than what the artists charge to the company.

**Data Flow Diagram (DFD)**

**Entity Relationship Diagram (ERD)**



**Part 3**

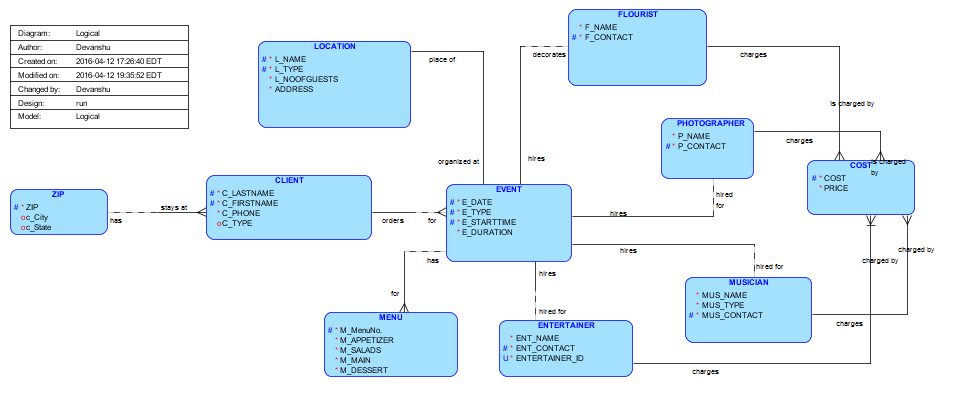
**Modification to the Original ERD**

For Part 3, I have modified the ERD that was made in part 2 by adding and changing a few attributes and relationships between entities. The modifications that I have made to the original ERD are as follows:

* The most significant change is the addition of entities like Cost and Zip. I have added these entities in order to reduce redundancy. In the original ERD, we had attribute ‘costs’ in all the artist entities. Having costs in all four entities was increasing redundancy, because two photographers can have the same cost, and then the charge applicable to the client would also be the same. Therefore, it was important to make Cost as a separate entity. As we know that ‘price’ is functionally determined by costs, we add price attribute to the Cost entity.
* Same applies for Zip, two or more clients can have the same zip. Therefore, to remove the redundancy, we separated it into another entity ‘Zip’, and as c\_City and c\_State are functionally dependent on Zip, even they move along with their determinant.
* Here, I have also paid attention to the cardinality of the relationships. An artist can choose to charge the catering company depending on the type of event and number of guests. Therefore, one artist can have more than one cost and that will form a one to many cardinality between artists (photographer, musician, entertainer, and florist) and Cost. Similar is the case with zip, because there can be more than one client residing at the same zip. Therefore, there will be a one to many cardinality from Zip to Client.
* Attention has also been given to participation of the entities in a relationship. Here we can see that, it is not necessary that every event will have a photographer, a musician, an entertainer, or a florist. There are chances that a client does not want a musician or an entertainer and might want only a photographer and a florist for his/her event. Therefore, there will be partial participation from the artist’s side to the event. However, when a client requests for an event, it is necessary to have a location and menu, because these are the main functionalities of the catering company. Therefore, there is a total participation from Event to Location and Menu.

These changes in the design and ERD modifications improve the previous design made for part 2 of the project. The current design better handles the business rules and requirements stated above in parts 1 and 2.

**Revised ERD Figure** (changes described in text)

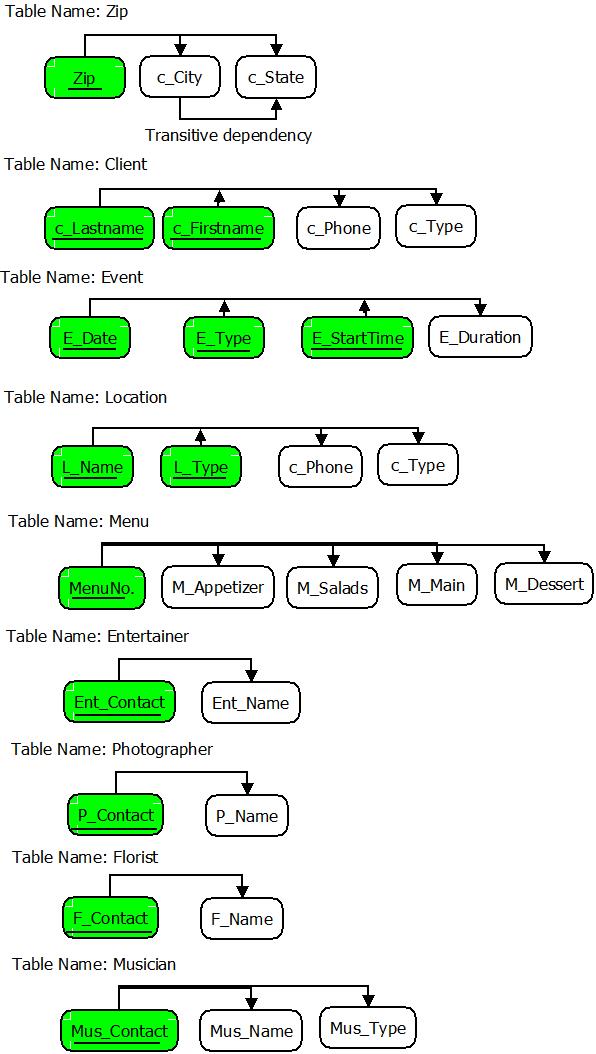


**Dependency Diagram**

The Dependency diagram below gives us an idea which attributes are dependent on which attributes. An attribute can functionally determine one or more attributes. The attribute that determines another attribute is called a determinant. Dependencies are checked in all the tables in the database. We also determine whether the dependency is partial or transitive. In this given diagram, I have underlined and shaded the box containing the primary key. Arrows above the attributes indicate desirable dependencies between attributes or those based on primary keys. Arrows below attributes indicate the less desirable dependencies and may include partial or transitive dependencies. A partial dependency occurs when a determinant is only one part of the primary key. A transitive dependency occurs when a primary key serves as a determinant only via another intermediate attribute, and only occurs when a functional dependence exists among nonprime attributes. For example, a transitive dependence exists if one attribute that is not a primary key, or not part of a primary key, determines another attribute that is also not part of a primary key. Transitive dependencies may create problems, as they create data redundancy and data anomalies. Partial dependencies are sometimes acceptable to improve performance, but tables with partial dependencies should only be used with caution because of data redundancy concerns.

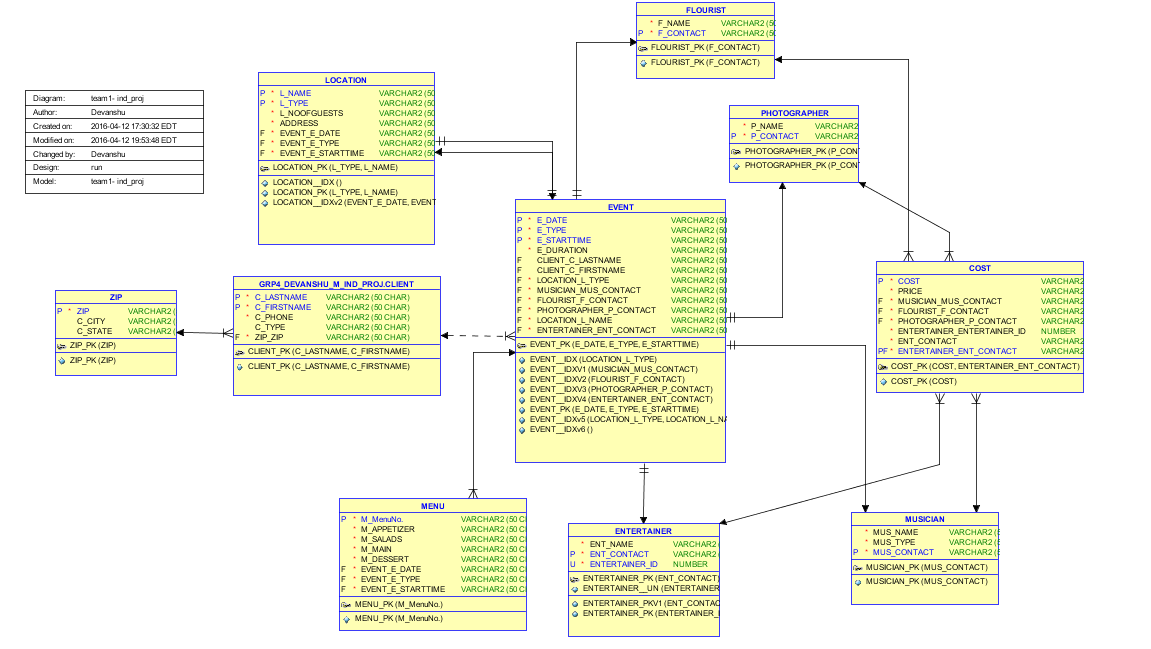
The dependency diagram below for ABC’s database in this project shows a reasonable level of normalization required for this part of the project. There is one transitive dependency in the diagram which is between c\_City and c\_State. c\_City is determined by zip and c\_State is determined by c\_City. For example, when the city Silver Spring comes in the table for the zip 20910, the state is always Maryalnd. Therefore, there is a transitive dependency. For now, there were no partial dependencies in the databases design. The transitive dependency will be resolved as we normalize the design further.

**Figure of Project Dependency Diagram:**



**Relational Diagram**

The project Entity Relationship Diagram or the Logical Diagram is translated into a relational model. The process of translating the ERD into the relational diagram is done by forward engineering. But here we have to make it manually. The process consists of converting entity relationships to relations or tables wherever entities have attributes belonging to a particular relationship. The Entity relationship diagram for ABC’s database includes only one to one and one to many relationships. The Entity Relationship Diagram did not include any many to many relationships, therefore I did not need to create any new tables with foreign keys for each participant entity in a many to many relationship. For the one to one relationships in my database project ERD, I have added a foreign key in the child table for every primary key in the owner table. For example, P\_Contact is Primary key in Photographer but a foreign key in Event. For the one to many relationships, I added a foreign key to the entity on the many side of the relationship to provide a reference to link to this entity on the one side of this relationship. For example, c\_Lastname and c\_Firstname are foreign keys in Event which is the many side, which gives the reference to entity where they are primary key, i.e., Client table which is the one side.

**Figure of Project Relational Diagram (Tables)**

**SQL DDL Script**

I used SQL Developer Data Modeler to generate the SQL DDL script. It is attached below.

***What was learned in Part 3?***

This part was important because I had to convert the Logical Data Model and Entity Relationship Diagram to a Relational Diagram and model including the tables and table structure that form the basis of the project’s relational database. The Entity Relationship Diagram is used as input to create the Relational Diagram with all the associated tables. I changed several entities, their attributes, and their relationships, as well as attribute data types. I even made sure that all the unique requirements were satisfied. Linking between the tables was done using appropriate foreign keys. These steps were very important to translate the Entity Relationship Diagram to a Relational Diagram with tables that adhere to integrity constraints based on primary and foreign keys, and unique constraint requirements. The relational diagram and tables shows the table structure of the database, with entities converted to tables, table attributes now showing their data types with length and the units, and the primary and foreign key designated as P and K, and with unique requirements indicated by an asterisk ‘\*’.

For creating a table for each entity, I selected a primary key for each entity including the single-valued attributes which hold a single value for a single entity, For example, P\_Contact in Photographer, M\_MenuNo. In Menu, etc. The changes from part 2 to part 3 helped me to learn about paying attention to redundancies that may occur in a database. Along with this I learnt about the cardinalities and participation between relationships. I modified the ERD in part 3 to take care of the total and partial participations. After this, I learnt about the functional dependencies and about partial and transitive dependencies. I verified that the generated relational model generated by SQL Modeler matched the dependencies in the Dependency Diagram. I, then confirmed that the relational model matched the Dependency Diagram contents and then proceeded to create the SQL DDL script, to create the tables and the initial database schema.

**SQL DDL script**

-- Generated by Oracle SQL Developer Data Modeler 4.1.3.901

-- at: 2016-04-09 21:35:13 EDT

-- site: Oracle Database 11g

-- type: Oracle Database 11g

CREATE TABLE Client

(

c\_Lastname VARCHAR2 (50 CHAR) NOT NULL ,

c\_Firstname VARCHAR2 (50 CHAR) NOT NULL ,

c\_Phone VARCHAR2 (50 CHAR) NOT NULL ,

c\_Type VARCHAR2 (50 CHAR) ,

Zip\_Zip VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Client ADD CONSTRAINT Client\_PK PRIMARY KEY ( c\_Lastname, c\_Firstname ) ;

CREATE TABLE Cost

(

Cost VARCHAR2 (50 CHAR) NOT NULL ,

Price VARCHAR2 (50 CHAR) NOT NULL ,

Musician\_Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Flourist\_F\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Photographer\_P\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_Entertainer\_ID NUMBER NOT NULL ,

Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_PK PRIMARY KEY ( Cost ) ;

CREATE TABLE Entertainer

(

Ent\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_ID NUMBER NOT NULL

) ;

ALTER TABLE Entertainer ADD CONSTRAINT Entertainer\_PK PRIMARY KEY ( Entertainer\_ID ) ;

ALTER TABLE Entertainer ADD CONSTRAINT Entertainer\_Ent\_Contact\_UN UNIQUE ( Ent\_Contact ) ;

CREATE TABLE Event

(

E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

E\_StartTime VARCHAR2 (50 CHAR) NOT NULL ,

E\_Duration VARCHAR2 (50 CHAR) NOT NULL ,

Client\_c\_Lastname VARCHAR2 (50 CHAR) ,

Client\_c\_Firstname VARCHAR2 (50 CHAR) ,

Location\_L\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Musician\_Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Flourist\_F\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Photographer\_P\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

L\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

CREATE UNIQUE INDEX Event\_\_IDX ON Event

(

Location\_L\_Type ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDX ON Event

(

Location\_L\_Type ASC , L\_Name ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv1 ON Event

(

Musician\_Mus\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv2 ON Event

(

Flourist\_F\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv3 ON Event

(

Photographer\_P\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv4 ON Event

(

Entertainer\_Ent\_Contact ASC

)

;

ALTER TABLE Event ADD CONSTRAINT Event\_PK PRIMARY KEY ( E\_Date, E\_Type, E\_StartTime ) ;

CREATE TABLE Flourist

(

F\_Name VARCHAR2 (50 CHAR) NOT NULL ,

F\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Flourist ADD CONSTRAINT Flourist\_PK PRIMARY KEY ( F\_Contact ) ;

CREATE TABLE Location

(

L\_Name VARCHAR2 (50 CHAR) NOT NULL ,

L\_Type VARCHAR2 (50 CHAR) NOT NULL ,

L\_NoOfGuests VARCHAR2 (50 CHAR) NOT NULL ,

Address VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_StartTime VARCHAR2 (50 CHAR) NOT NULL

) ;

CREATE UNIQUE INDEX Location\_\_IDX ON Location

(

Event\_E\_Date ASC , Event\_E\_Type ASC , Event\_E\_StartTime ASC

)

;

ALTER TABLE Location ADD CONSTRAINT Location\_PK PRIMARY KEY ( L\_Type, L\_Name ) ;

CREATE TABLE Menu

(

"M\_MenuNo." VARCHAR2 (50 CHAR) NOT NULL ,

M\_Appetizer VARCHAR2 (50 CHAR) NOT NULL ,

M\_Salads VARCHAR2 (50 CHAR) NOT NULL ,

M\_Main VARCHAR2 (50 CHAR) NOT NULL ,

M\_Dessert VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_StartTime VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Menu ADD CONSTRAINT Menu\_PK PRIMARY KEY ( "M\_MenuNo." ) ;

CREATE TABLE Musician

(

Mus\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Mus\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Musician ADD CONSTRAINT Musician\_PK PRIMARY KEY ( Mus\_Contact ) ;

CREATE TABLE Photographer

(

P\_Name VARCHAR2 (50 CHAR) NOT NULL ,

P\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Photographer ADD CONSTRAINT Photographer\_PK PRIMARY KEY ( P\_Contact ) ;

CREATE TABLE Zip

(

Zip VARCHAR2 (50 CHAR) NOT NULL ,

c\_City VARCHAR2 (50 CHAR) NOT NULL ,

c\_State VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Zip ADD CONSTRAINT Zip\_PK PRIMARY KEY ( Zip ) ;

ALTER TABLE Client ADD CONSTRAINT Client\_Zip\_FK FOREIGN KEY ( Zip\_Zip ) REFERENCES Zip ( Zip ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Entertainer\_FK FOREIGN KEY ( Ent\_Contact ) REFERENCES Entertainer ( Entertainer\_ID ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Flourist\_FK FOREIGN KEY ( Flourist\_F\_Contact ) REFERENCES Flourist ( F\_Contact ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Musician\_FK FOREIGN KEY ( Musician\_Mus\_Contact ) REFERENCES Musician ( Mus\_Contact ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Photographer\_FK FOREIGN KEY ( Photographer\_P\_Contact ) REFERENCES Photographer ( P\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Client\_FK FOREIGN KEY ( Client\_c\_Lastname, Client\_c\_Firstname ) REFERENCES Client ( c\_Lastname, c\_Firstname ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Entertainer\_FK FOREIGN KEY ( Entertainer\_Ent\_Contact ) REFERENCES Entertainer ( Ent\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Flourist\_FK FOREIGN KEY ( Flourist\_F\_Contact ) REFERENCES Flourist ( F\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Location\_FK FOREIGN KEY ( Location\_L\_Type, L\_Name ) REFERENCES Location ( L\_Type, L\_Name ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Musician\_FK FOREIGN KEY ( Musician\_Mus\_Contact ) REFERENCES Musician ( Mus\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Photographer\_FK FOREIGN KEY ( Photographer\_P\_Contact ) REFERENCES Photographer ( P\_Contact ) ;

ALTER TABLE Location ADD CONSTRAINT Location\_Event\_FK FOREIGN KEY ( Event\_E\_Date, Event\_E\_Type, Event\_E\_StartTime ) REFERENCES Event ( E\_Date, E\_Type, E\_StartTime ) ;

ALTER TABLE Menu ADD CONSTRAINT Menu\_Event\_FK FOREIGN KEY ( Event\_E\_Date, Event\_E\_Type, Event\_E\_StartTime ) REFERENCES Event ( E\_Date, E\_Type, E\_StartTime ) ;

CREATE SEQUENCE Entertainer\_Entertainer\_ID\_SEQ START WITH 1 NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER Entertainer\_Entertainer\_ID\_TRG BEFORE

INSERT ON Entertainer FOR EACH ROW WHEN (NEW.Entertainer\_ID IS NULL) BEGIN :NEW.Entertainer\_ID := Entertainer\_Entertainer\_ID\_SEQ.NEXTVAL;

END;

/

-- Oracle SQL Developer Data Modeler Summary Report:

--

-- CREATE TABLE 10

-- CREATE INDEX 7

-- ALTER TABLE 24

-- CREATE VIEW 0

-- ALTER VIEW 0

-- CREATE PACKAGE 0

-- CREATE PACKAGE BODY 0

-- CREATE PROCEDURE 0

-- CREATE FUNCTION 0

-- CREATE TRIGGER 1

-- ALTER TRIGGER 0

-- CREATE COLLECTION TYPE 0

-- CREATE STRUCTURED TYPE 0

-- CREATE STRUCTURED TYPE BODY 0

-- CREATE CLUSTER 0

-- CREATE CONTEXT 0

-- CREATE DATABASE 0

-- CREATE DIMENSION 0

-- CREATE DIRECTORY 0

-- CREATE DISK GROUP 0

-- CREATE ROLE 0

-- CREATE ROLLBACK SEGMENT 0

-- CREATE SEQUENCE 1

-- CREATE MATERIALIZED VIEW 0

-- CREATE SYNONYM 0

-- CREATE TABLESPACE 0

-- CREATE USER 0

--

-- DROP TABLESPACE 0

-- DROP DATABASE 0

--

-- REDACTION POLICY 0

--

-- ORDS DROP SCHEMA 0

-- ORDS ENABLE SCHEMA 0

-- ORDS ENABLE OBJECT 0

--

-- ERRORS 0

-- WARNINGS 0

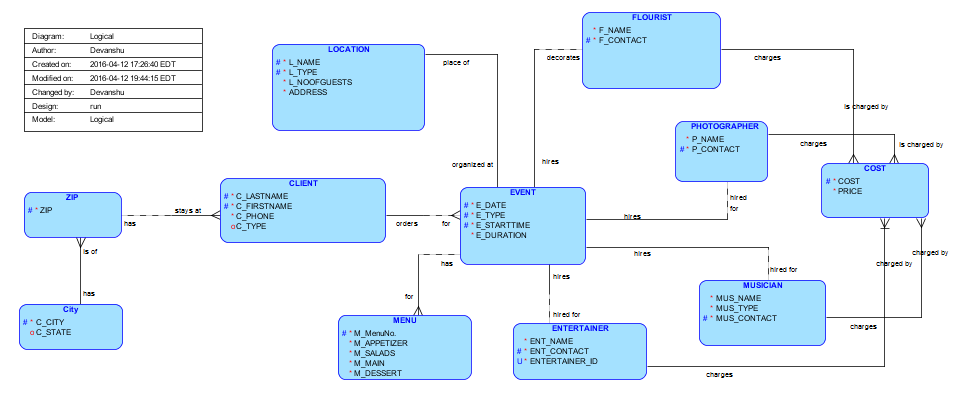
# **Part 4**

# **Design Modifications for Part 4**

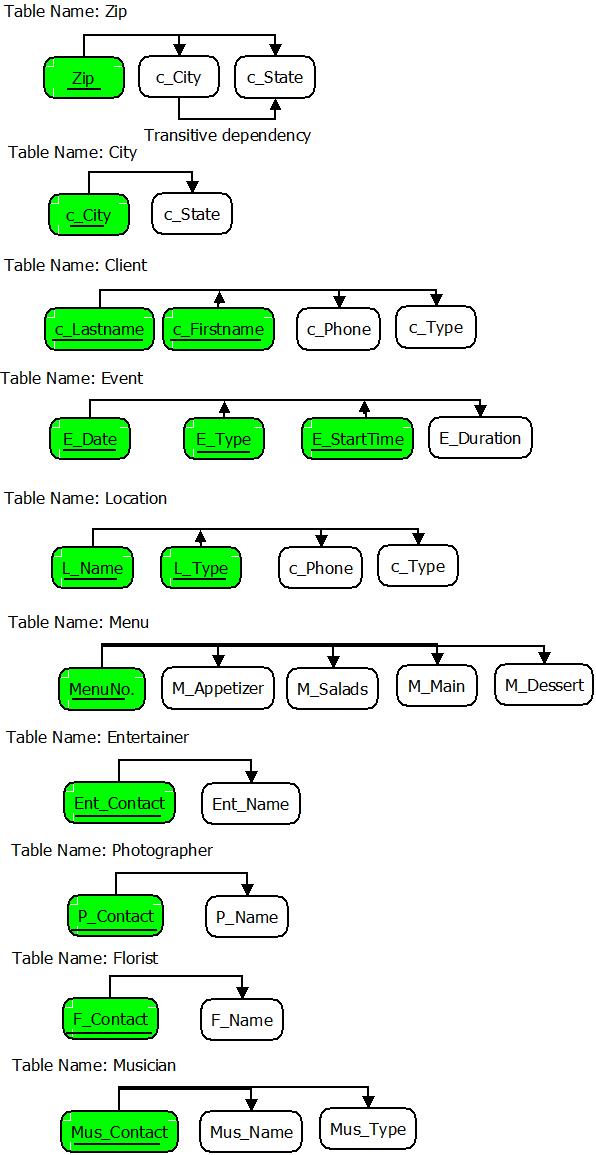
# In part 4, I took care of the anomalies, i.e., I tried to normalize all the tables in the database as much as possible. One major modification that I made from part 3 to part 4 is removing the transitive dependency. The transitive dependency from c\_City to c\_state was removed in this part. I made a separate table for City with c\_City representing the city name and c\_State representing the state name. c\_City is the primary key of the City table and foreign key in zip table. The cardinality of City to Zip is one to many, i.e., one city can have many zip codes depending on the area in the city. This removes transitive dependency and redundancies, because, now a city will be in the table only once with its respective state rather than having to repeatedly have the same city and state for many zip codes in the zip table. Another modification I did was to manually add a surrogate key to my Event table as Event\_count. The changes described above were made to the ERD, Relational Model and Diagram, and the Dependency Diagram and modified versions of these figures are shown below.

**Modified Figures (based on changes described above)**

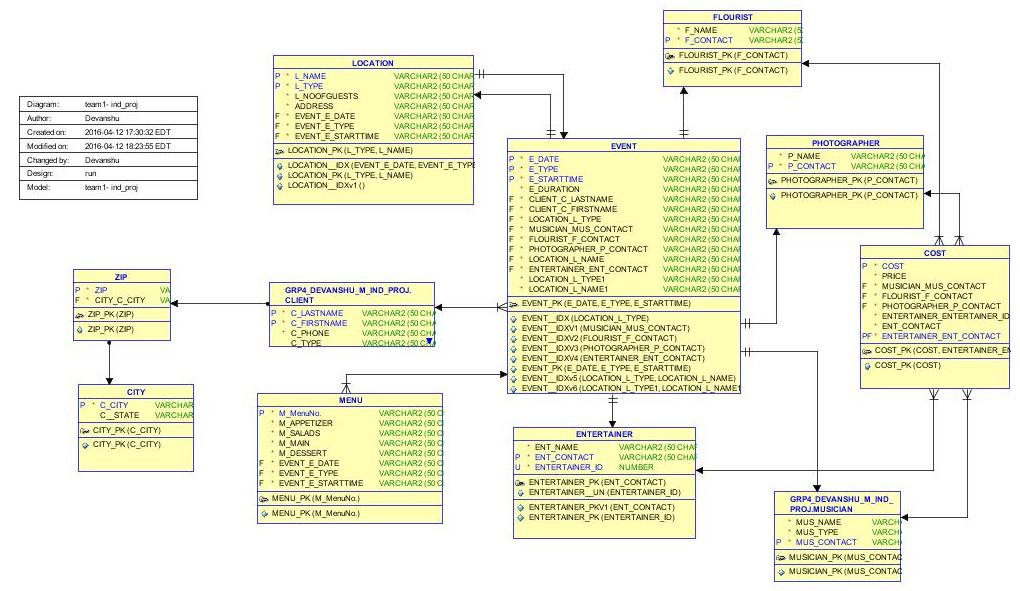
**Figure of Modified ERD**

****

**Figure of Modified Project Dependency Diagram**



**Figure of Modified Project Relational Diagram**

****

**Primary Keys, Foreign Keys, and Unique and Check Constraints**

I made sure that all tables contained a primary key (P) and that these primary keys were enforced with UNIQUE key constraints. I also created foreign keys to link table information between tables involved in relationships. Foreign keys are used to reference related attributes in a different table. Several modifications on foreign keys were made directly in the SQL code (see sql code above). Surrogate keys are generated by the system and cannot be seen by database users. I used surrogate keys in event tables. For example, Flourist\_ID, Musician\_ID, and Photo\_ID are used by the company and it is NOT implemented as surrogate keys.

Apart from this, we made sure that we keep in mind the practicality of our company database. Therefore, I have more than one attribute as the primary key in some of the tables. As mentioned in question 6.4, there can only be one event on one day, therefore, the combination of E\_Date, E\_Type, and E\_StartTime attributes is made primary key. Also L\_Name and L\_Type attributes together form the primary key of location table.

There are a lot of foreign keys in the table as seen in the relational model. The artist and location information is stored in the event table as foreign keys.

A check constraint was necessary on number of guests in order to decide the location type. However, putting a check constraint on the entire column would not solve the issue. Therefore, I created a check constraint on the number of guests by adding a trigger. When a user enters location information that violates the number of guests constraint, the location type is changed accordingly.

The trigger is as follows:

Create or replace trigger NoOfGuests

before insert on location

for each row

begin

update location

set L\_Type='Others'

where L\_NoOfGuests=201;

exception

when no\_data\_found then

DBMS\_OUTPUT.PUT\_LINE('NO DATA FOUND');

when others then

DBMS\_OUTPUT.PUT\_LINE('ERROR-'||SQLERRM);

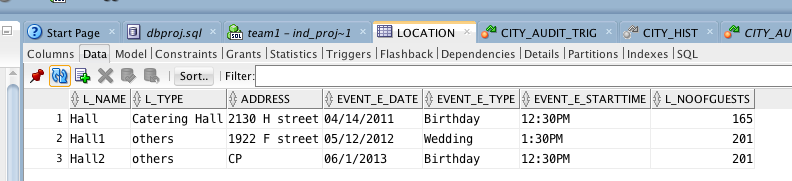
end;

/

alter trigger NoOfGuests enable;

The above trigger changes the location type to others if the number of guests are less than 150 or more than 200.

The output is as follows:



**Normalization**

A significant amount of normalization was already attained for part 3 of this project. I created a database design that was already in 1NF and 2NF. However, it was not in 3NF as there existed a transitive dependency in Zip table. I attempted to normalize the tables before and during the development of the relational model and tables for part 3. I made sure most tables were normalized up through 3NF and BCNF. In part 4, I normalized all the tables to make sure that they are in 3NF and BCNF. We separated the zip and city table to reduce redundancy. Zip contains c\_city as foreign key and City contains c\_City as primary key.

**Sequences**

I created sequences for tables using surrogate keys with values not containing any useful business information. For example, I created and used the Flourist\_Flourist\_ID\_SEQ sequence for the Flourist\_ID attribute in the Flourist table to generate Floursit\_ID values using the Oracle NEXT.VAL function. I also created and used the Musician\_Musician\_ID\_SEQ for the Music\_ID attribute to generate sequential values of Music\_ID. Along with this I created and used the Photographer\_Photo\_ID\_SEQ for the Photo\_ID attribute to generate sequential values of Photo\_ID. The trigger with the NEXT.VAL function helps to increment the value of the sequential ID.

The following sequences were added to original database, and these sequences were used to insert sequence values for the new “ID”.

Commands:

CREATE SEQUENCE Flourist\_Flourist\_ID\_SEQ START WITH 1 NOCACHE ORDER ;

ALTER TABLE Flourist

ADD Flourist\_ID NUMBER

CREATE OR REPLACE TRIGGER Flourist\_Flourist\_ID\_TRG BEFORE

INSERT ON Flourist FOR EACH ROW WHEN (NEW.Flourist\_ID IS NULL)

BEGIN :NEW.Flourist\_ID := Flourist\_Flourist\_ID\_SEQ.NEXTVAL;

END;

CREATE SEQUENCE Musician\_Musician\_ID\_SEQ START WITH 1 NOCACHE ORDER ;

ALTER TABLE Musician

ADD Musician\_ID NUMBER

CREATE OR REPLACE TRIGGER Musician\_Musician\_ID\_TRG BEFORE

INSERT ON Musician FOR EACH ROW WHEN (NEW.Musician\_ID IS NULL)

BEGIN :NEW.Musician\_ID := Musician\_Musician\_ID\_SEQ.NEXTVAL;

END;

CREATE SEQUENCE Photographer\_Photo\_ID\_SEQ START WITH 1 NOCACHE ORDER ;

ALTER TABLE Photographer

ADD Photo\_ID NUMBER

CREATE OR REPLACE TRIGGER Photographer\_Photo\_ID\_TRG BEFORE

INSERT ON Photographer FOR EACH ROW WHEN (NEW.Photo\_ID IS NULL)

BEGIN :NEW.Photo\_ID := Photographer\_Photo\_ID\_SEQ.NEXTVAL;

END;

ALTER TABLE Client

ADD CONSTRAINT client\_unique UNIQUE (c\_LastName, c\_FirstName, c\_Phone);

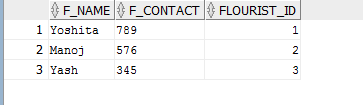
After running these sequences we tried some inserting some values:

insert into flourist values('Yoshita', '789',null);

insert into flourist values('Manoj', '576', null);

insert into flourist values('Yash', '345', null);

For these three insert commands, the value of ID is auto incremented by using the NEXT.VAL function. The output is as follows:



**Populate Tables**

I used SQL INSERT statements to populate tables in the database with data. I populated the primary database tables Client, Event, Zip, Location, Menu, and the various artists table with at least 26-30 rows of data. Some additional tables, such as Cost and City tables were also populated with data to represent the Cost and price of the artists, and the cities with their states. In cases where a surrogate key was used, I used the created SEQUENCE objects along with the NEXTVAL sequence function to generate surrogate key values like Flourist\_ID, Photo\_ID, and Musician\_ID.

insert into city values('Silver Spring', 'MD');

insert into city values('Washington', 'DC');

insert into city values('Lorton', 'VA');

insert into zip values('20910', 'Silver Spring', 'MD', 'Silver Spring');

insert into zip values('20911', 'Silver Spring', 'MD', 'Silver Spring');

insert into zip values('20037', 'Washington', 'DC', 'Washington');

insert into zip values('20938', 'Washington', 'DC', 'Washington');

insert into zip values('20079', 'Lorton', 'VA', 'Lorton');

insert into flourist values('Yoshita', '789',null);

insert into flourist values('Manoj', '576', null);

insert into flourist values('Yash', '345', null);

insert into musician values('Deadmau5', 'EDM', '123', null);

insert into musician values('Deadmau4', 'EDM', '456', null);

insert into musician values('Deadmau2', 'EDM', '789', null);

insert into photographer values('Doremon', '463', null);

insert into photographer values('Pichaku', '745', null);

insert into photographer values('Chota Bheem', '769', null);

insert into entertainer values('Anil', '193', null);

insert into entertainer values('Mukesh', '347', null);

insert into entertainer values('Dhirubhai', '764', null);

insert into client values('Matlawala', 'Devanshu', '123', 'Business', '20910');

insert into client values('Patel', 'Akash', '123', 'Business', '20910');

insert into client values('Patel', 'Maulik', '123', 'Business', '20911');

insert into client values('Patel', 'Aman', '123', 'Business', '20910');

insert into client values('Mehta', 'Akash', '123', 'Individual', '20037');

insert into event values('04/14/2011', 'Birthday', '12:30PM', '120 mins', 'Matlawala', 'Devanshu', 'Catering Hall', '123', '123', '123', 'Hall', '123');

insert into event values('05/12/2012', 'Wedding', '1:30PM', '10 hours', 'Patel', 'Akash', 'Catering Hall1', '456', '456', '456', 'Hall1', '456');

insert into event values('06/1/2013', 'Birthday', '12:30PM', '120 mins', 'Patel', 'A', 'Catering Hall2', '789', '789', '789', 'Hall2', '789');

insert into location values('Hall', 'Catering Hall', '2130 H street', '04/14/2011', 'Birthday', '12:30PM', 165);

insert into location values('Hall1', 'Catering Hall', '1922 F street', '05/12/2012', 'Wedding', '1:30PM', 201);

insert into location values('Hall2', 'Catering Hall', 'CP', '05/12/2012', 'Wedding', '1:30PM', 201);

insert into menu values('1','pav','green','bhaji','gulab','04/14/2011', 'Birthday', '12:30PM');

insert into menu values('2','bread','beans','panipuri','kulfi','05/12/2012', 'Wedding', '1:30PM');

insert into menu values('3','crispy veg','ceaser','pizza','rasgula','06/1/2013', 'Birthday', '12:30PM');

insert into cost values('1000','1100','123','789','463','193');

insert into cost values('2000','2200','456','576','745','347');

insert into cost values('3000','3300','789','345','769','746');

**SQL Queries**

select \*

from event

where E\_Date = '06/1/2013';

select distinct c\_LastName from client

where c\_type='Business';

select \* from client natural join zip;

select c\_FirstName from client

group by c\_phone;

SELECT c\_lastname, count(\*)

FROM client

GROUP BY c\_lastname;

select p\_name, count(\*)

from photographer

group by p\_name

having count(\*)>=2;

select c\_city, count(\*)

from zip

group by c\_city

having count(\*)>1;

SELECT distinct zip.zip, city.c\_city

FROM zip

INNER JOIN city

ON city.c\_city=zip.c\_city;

***WHAT WAS LEARNED in PART 4?***

In part 4, we refined the database design even more to normalize all the tables. In this, I learnt about anomalies and how to normalize a database design up to BCNF. I learnt to take care of the foreign keys while populating databases. While inserting we have to follow a chain that starts from the owner table with the primary key and runs to child table with the foreign keys. Loading data and avoiding referential integrity constraint errors required me to follow parent-child insert ordering. After this, I ran some SQL queries with JOIN, GROUP BY, and HAVING clauses. One interesting concept I learnt in part 4 was to use SEQUENCES with NEXT.VAL function and saw how they were used and implemented. SEQUENCES were used to generate surrogate key values in the database.

**Part 5**

**Modifications from previous sections**

**SURROGATE KEYS, NATURAL KEYS, UNIQUE CONSTRAINTS, AND SEQUENCE MODIFICATIONS**

I changed the primary keys for several tables. I added new surrogate key attributes (e.g. Floursit\_ID, Musician\_ID, Photo\_ID), and modified these tables to use the new attributes as primary keys (see Part 4 ‘Sequences’ sql script).

The new sequences are used when inserting values (.NEXTVAL) for the surrogate keys (e.g. Floursit\_ID, Musician\_ID, Photo\_ID). The original primary keys (e.g. c\_LastName, Zip, L\_Name, L\_Type, E\_Type) were modified to be used as natural keys. These natural keys are no longer used as primary keys but must still be unique according to the business rules explained in part 2.

**Trigger Creation and Evaluation**

Four triggers were created for the database in part 5. One trigger was created to save the old records in the Zip\_Hist table. The trigger, ZipHist, takes the old values of the rows being updated in the Zip table and saves them into the Zip\_Hist table. The Zip table stores the updated values after an update commands is executed, while the old values are stored in the Zip\_Hist table if in case the old values need to be referenced in the future.

Three additional triggers were created to record audit information for three different tables, City, Zip, and Cost. This information is stored in three audit attributes, typeof, username, and exacttime that were added to the three tables. The audit attributes, type, username, and exacttime were added to the three tables CITY, ZIP, and COST. The typeof contains whether the trigger was for an ‘insert’ or ‘update’. The username attribute contains name of users who insert and update. The exacttime attribute contains the date and time at which the insert or update took place. These new table attributes are used by the following three triggers:

COST\_AUDIT\_TRIG

CITY\_AUDIT\_TRIG

ZIP\_AUDIT\_TRIG

The final Part 5 SQL script contains both parts 4 and part 5 components, and can be run as a complete script from beginning to end to demonstrate table, constraints, sequences, and trigger creation. The script also contains insert commands to populate the database. The entire script is given in the end and included in the .sql file as well.

The audit trigger for Cost table is as follows:

create table city\_log(typeof varchar2(50), username varchar2(50), exacttime varchar(50));

create or replace TRIGGER City\_AUDIT\_TRIG

BEFORE INSERT OR UPDATE ON CITY

FOR EACH ROW

BEGIN

If INSERTING then

insert into city\_log values('Insert',user,sysdate);

end if;

if UPDATING then

insert into city\_log values('Update',user,sysdate);

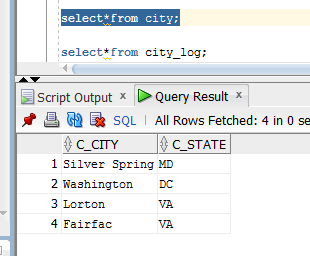
end if;

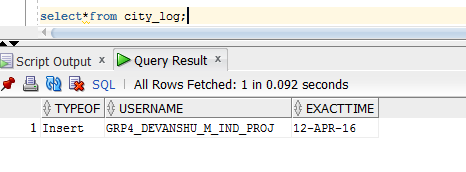
END;

insert into city values('Fairfac', 'VA');

select\*from city;

The output is as follows:





**WHAT WAS LEARNED in PART 5?**

In part 5 I learnt about creation and usage of database triggers. These triggers were used to automatically update rows in tables and also to audit information about who and where the records were inserted or updated. Triggers are useful to keep track of the changes and to make automatic changes wherever required. In our database design, we have created audit triggers that store the name of the person who inserted or updated anything in the specified tables along with the exact system date and time at which the insert or update was done. We created new tables like Zip\_Hist and City\_log to see the working of these triggers.

**SUMMARY**

# This Database Project consisted of five phases that covered everything from database designing to creating the database tables, inserting values, updating values, creating sequences, and even working with triggers. I used both Oracle SQL Data modeler along with Oralce SQL Developer to design and construct the database for our catering company. The first part included the translation of a manual hand-drawn ERD to proper logical and relational models. We engineered the logical model to automatically construct the relational model. Then we exported the DDL script from the relational model to create a working database in SQL Developer. The database created in SQL Developer included primary and foreign keys along with surrogate keys and sequences in the end.

# The objective of this project was to create a database system to store and manage events requested by the clients. I made sure to cover all aspects when deciding how to organize an event, including the location, menu, and artist requirements. I created sequences to track the number of updates and triggers to update certain tables when required and to audit changes in the system.

# The database allows the catering company’s employees to store client information along with zip and city details. The event information along with location and menu details were also stored. If the client requested for a particular artist then that requirement was also fulfilled by the catering company. The employees can also track any changes using the triggers defined in the project.

SQL script for part 4 and part 5:

-- Generated by Oracle SQL Developer Data Modeler 4.1.3.901

-- at: 2016-04-09 22:05:45 EDT

-- site: Oracle Database 11g

-- type: Oracle Database 11g

CREATE TABLE City

(

c\_City VARCHAR2 (50 CHAR) NOT NULL ,

c\_State VARCHAR2 (50 CHAR)

) ;

ALTER TABLE City ADD CONSTRAINT City\_PK PRIMARY KEY ( c\_City ) ;

CREATE TABLE Client

(

c\_Lastname VARCHAR2 (50 CHAR) NOT NULL ,

c\_Firstname VARCHAR2 (50 CHAR) NOT NULL ,

c\_Phone VARCHAR2 (50 CHAR) NOT NULL ,

c\_Type VARCHAR2 (50 CHAR) ,

Zip\_Zip VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Client ADD CONSTRAINT Client\_PK PRIMARY KEY ( c\_Lastname, c\_Firstname ) ;

CREATE TABLE Cost

(

Cost VARCHAR2 (50 CHAR) NOT NULL ,

Price VARCHAR2 (50 CHAR) NOT NULL ,

Musician\_Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Flourist\_F\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Photographer\_P\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_Entertainer\_ID NUMBER NOT NULL ,

Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_PK PRIMARY KEY ( Cost ) ;

CREATE TABLE Entertainer

(

Ent\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_ID NUMBER NOT NULL

) ;

ALTER TABLE Entertainer ADD CONSTRAINT Entertainer\_PK UNIQUE ( Entertainer\_ID ) ;

CREATE TABLE Event

(

E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

E\_StartTime VARCHAR2 (50 CHAR) NOT NULL ,

E\_Duration VARCHAR2 (50 CHAR) NOT NULL ,

Client\_c\_Lastname VARCHAR2 (50 CHAR) ,

Client\_c\_Firstname VARCHAR2 (50 CHAR) ,

Location\_L\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Musician\_Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Flourist\_F\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Photographer\_P\_Contact VARCHAR2 (50 CHAR) NOT NULL ,

Location\_L\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Entertainer\_Ent\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

CREATE UNIQUE INDEX Event\_\_IDX ON Event

(

Location\_L\_Type ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDX ON Event

(

Location\_L\_Type ASC , Location\_L\_Name ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv1 ON Event

(

Musician\_Mus\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv2 ON Event

(

Flourist\_F\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv3 ON Event

(

Photographer\_P\_Contact ASC

)

;

CREATE UNIQUE INDEX Event\_\_IDXv4 ON Event

(

Entertainer\_Ent\_Contact ASC

)

;

ALTER TABLE Event ADD CONSTRAINT Event\_PK PRIMARY KEY ( E\_Date, E\_Type, E\_StartTime ) ;

CREATE TABLE Flourist

(

F\_Name VARCHAR2 (50 CHAR) NOT NULL ,

F\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Flourist ADD CONSTRAINT Flourist\_PK PRIMARY KEY ( F\_Contact ) ;

CREATE TABLE Location

(

L\_Name VARCHAR2 (50 CHAR) NOT NULL ,

L\_Type VARCHAR2 (50 CHAR) NOT NULL ,

L\_NoOfGuests VARCHAR2 (50 CHAR) NOT NULL ,

Address VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_StartTime VARCHAR2 (50 CHAR) NOT NULL

) ;

CREATE UNIQUE INDEX Location\_\_IDX ON Location

(

Event\_E\_Date ASC , Event\_E\_Type ASC , Event\_E\_StartTime ASC

)

;

ALTER TABLE Location ADD CONSTRAINT Location\_PK PRIMARY KEY ( L\_Type, L\_Name ) ;

CREATE TABLE Menu

(

"M\_MenuNo." VARCHAR2 (50 CHAR) NOT NULL ,

M\_Appetizer VARCHAR2 (50 CHAR) NOT NULL ,

M\_Salads VARCHAR2 (50 CHAR) NOT NULL ,

M\_Main VARCHAR2 (50 CHAR) NOT NULL ,

M\_Dessert VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Date VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Event\_E\_StartTime VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Menu ADD CONSTRAINT Menu\_PK PRIMARY KEY ( "M\_MenuNo." ) ;

CREATE TABLE Musician

(

Mus\_Name VARCHAR2 (50 CHAR) NOT NULL ,

Mus\_Type VARCHAR2 (50 CHAR) NOT NULL ,

Mus\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Musician ADD CONSTRAINT Musician\_PK PRIMARY KEY ( Mus\_Contact ) ;

CREATE TABLE Photographer

(

P\_Name VARCHAR2 (50 CHAR) NOT NULL ,

P\_Contact VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Photographer ADD CONSTRAINT Photographer\_PK PRIMARY KEY ( P\_Contact ) ;

CREATE TABLE Zip

(

Zip VARCHAR2 (50 CHAR) NOT NULL ,

c\_City VARCHAR2 (50 CHAR) NOT NULL ,

c\_State VARCHAR2 (50 CHAR) NOT NULL ,

City\_c\_City VARCHAR2 (50 CHAR) NOT NULL

) ;

ALTER TABLE Zip ADD CONSTRAINT Zip\_PK PRIMARY KEY ( Zip ) ;

ALTER TABLE Client ADD CONSTRAINT Client\_Zip\_FK FOREIGN KEY ( Zip\_Zip ) REFERENCES Zip ( Zip ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Entertainer\_FK FOREIGN KEY ( Ent\_Contact ) REFERENCES Entertainer ( Entertainer\_ID ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Flourist\_FK FOREIGN KEY ( Flourist\_F\_Contact ) REFERENCES Flourist ( F\_Contact ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Musician\_FK FOREIGN KEY ( Musician\_Mus\_Contact ) REFERENCES Musician ( Mus\_Contact ) ;

ALTER TABLE Cost ADD CONSTRAINT Cost\_Photographer\_FK FOREIGN KEY ( Photographer\_P\_Contact ) REFERENCES Photographer ( P\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Client\_FK FOREIGN KEY ( Client\_c\_Lastname, Client\_c\_Firstname ) REFERENCES Client ( c\_Lastname, c\_Firstname ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Entertainer\_FK FOREIGN KEY ( Entertainer\_Ent\_Contact ) REFERENCES Entertainer ( Ent\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Flourist\_FK FOREIGN KEY ( Flourist\_F\_Contact ) REFERENCES Flourist ( F\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Location\_FK FOREIGN KEY ( Location\_L\_Type, Location\_L\_Name ) REFERENCES Location ( L\_Type, L\_Name ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Musician\_FK FOREIGN KEY ( Musician\_Mus\_Contact ) REFERENCES Musician ( Mus\_Contact ) ;

ALTER TABLE Event ADD CONSTRAINT Event\_Photographer\_FK FOREIGN KEY ( Photographer\_P\_Contact ) REFERENCES Photographer ( P\_Contact ) ;

ALTER TABLE Location ADD CONSTRAINT Location\_Event\_FK FOREIGN KEY ( Event\_E\_Date, Event\_E\_Type, Event\_E\_StartTime ) REFERENCES Event ( E\_Date, E\_Type, E\_StartTime ) ;

ALTER TABLE Menu ADD CONSTRAINT Menu\_Event\_FK FOREIGN KEY ( Event\_E\_Date, Event\_E\_Type, Event\_E\_StartTime ) REFERENCES Event ( E\_Date, E\_Type, E\_StartTime ) ;

ALTER TABLE Zip ADD CONSTRAINT Zip\_City\_FK FOREIGN KEY ( City\_c\_City ) REFERENCES City ( c\_City ) ;

CREATE SEQUENCE Entertainer\_Entertainer\_ID\_SEQ START WITH 1 NOCACHE ORDER ;

CREATE OR REPLACE TRIGGER Entertainer\_Entertainer\_ID\_TRG BEFORE

INSERT ON Entertainer FOR EACH ROW WHEN (NEW.Entertainer\_ID IS NULL) BEGIN :NEW.Entertainer\_ID := Entertainer\_Entertainer\_ID\_SEQ.NEXTVAL;

END;

/

-- Oracle SQL Developer Data Modeler Summary Report:

--

-- CREATE TABLE 11

-- CREATE INDEX 7

-- ALTER TABLE 25

-- CREATE VIEW 0

-- ALTER VIEW 0

-- CREATE PACKAGE 0

-- CREATE PACKAGE BODY 0

-- CREATE PROCEDURE 0

-- CREATE FUNCTION 0

-- CREATE TRIGGER 1

-- ALTER TRIGGER 0

-- CREATE COLLECTION TYPE 0

-- CREATE STRUCTURED TYPE 0

-- CREATE STRUCTURED TYPE BODY 0

-- CREATE CLUSTER 0

-- CREATE CONTEXT 0

-- CREATE DATABASE 0

-- CREATE DIMENSION 0

-- CREATE DIRECTORY 0

-- CREATE DISK GROUP 0

-- CREATE ROLE 0

-- CREATE ROLLBACK SEGMENT 0

-- CREATE SEQUENCE 1

-- CREATE MATERIALIZED VIEW 0

-- CREATE SYNONYM 0

-- CREATE TABLESPACE 0

-- CREATE USER 0

--

-- DROP TABLESPACE 0

-- DROP DATABASE 0

--

-- REDACTION POLICY 0

--

-- ORDS DROP SCHEMA 0

-- ORDS ENABLE SCHEMA 0

-- ORDS ENABLE OBJECT 0

--

-- ERRORS 0

-- WARNINGS 0

Triggers and other sequences code is in the respective sections.

Screenshots of the update queries mentioned in part 4 are as follows:

