Database Management Systems

(COP 5725)

Spring 2015

Instructor: Dr. Markus Schneider

Project phase III:

Movie Database Management system

FirstView

Group 22 Members:
Chen, Chen
Du, Xin
Tu, Daoyu
Zhang, Wenchao

Feb. 25th, 2015

Part I Project Recapitulation:

The aim of our proposed project is to design and develop a movie database management system—*FirstView* as described in the previous phases of this project.

The motivation of this project is to establish a platform that involves both the public customers and cinema administrator users. By using the system, people shall get access to acknowledge the movies information according their preferences, while the cinema shall be able to conduct business interactions with customers as well as be acquainted with their state of operation.

The main functions to be achieved are listed as follows:

A. Queries with respects to movies and cinemas;

B. Tickets purchase according to customers' preference;

C. Ticket and schedule management with financial analysis of cinema.

In phase I, a careful requirement analyses with respect to the expected functionalities of the system was conducted. Three different roles involved are identified respectively. In phase II, the overall *conceptual database design* was described. Through an Entity-Relationship (ER) diagram, we illustrated the important entity sets and relationship sets along with their attributes. In addition, the *user interface design* was performed to devise a set of web pages that convey the promised functionalities of the proposed system.

During this phase period, a transformation algorithm is applied in the ER diagram and the database schema is produced as discussed in class. The corresponding SQL's data definition language (DDL) commands with additional explanation are implemented in Oracle offered by CISE server. Furthermore, several sample data manipulation language (DML) commands are provided to illustrate the realization sample of expected queries.

Part II ER Diagram

In this section, an ER schema diagram for the movie database system designed in last phase is presented to depict and describe data as entities, relationships and attributes in Figure 1. The descriptions of the ER diagram are as follows:

- There are two types of users: customers and cinemas. Each user can be uniquely identified by an ID. Also a user needs to provide its username and password. Email address can be provided as optional information of the user.
- Customer users can choose to provide one or more phone numbers in the system.
- Customer needs to provide purchase methods if he or she wishes to make a purchase. The purchase method information requires the card number as a key, card category, name on card, security code and expiration date.
- Review including a rating scaled from 0 to 5 and comments can be added by customers to movies or cinemas. A timestamp is generated once the review is made.
- Each cinema must have its name and location including state, city, street and zip code. A cinema administrator can add description, contacts and transports information.
- Each cinema consists of projection halls to display the movies. Hall ID is not enough to distinguish the projection halls without cinema's user ID. For each hall, column number and row number of seats need to be provided.
- A unique movie ID is assigned to each movie. Among the movie information, it contains its title, year, length, category, Motion Picture Association of America (MPAA) rating as well as its writer, director, casts, production corporation and introduction. Other than the movie ID, a movie can also be distinguished by the combination of its title with year. Several main actors may be included in the cast attribute. One movie is not limited to be categorized in one category. An "*if_on_show*" indicator is attached to check if the movie can be currently found in any cinema.
- Tickets applying on certain movies must be released by cinemas. Each ticket can be uniquely identifiable by its ticket ID. Moreover, a ticket contains the following information: location consisting of cinema name, Hall ID and seat number, time period, price and discount. An "*if_sold*" indicator checks the availability of the ticket.
- Customer can make purchase of tickets. Once a purchase is made, the card number applied is recorded and a purchase timestamp is generated simultaneously.

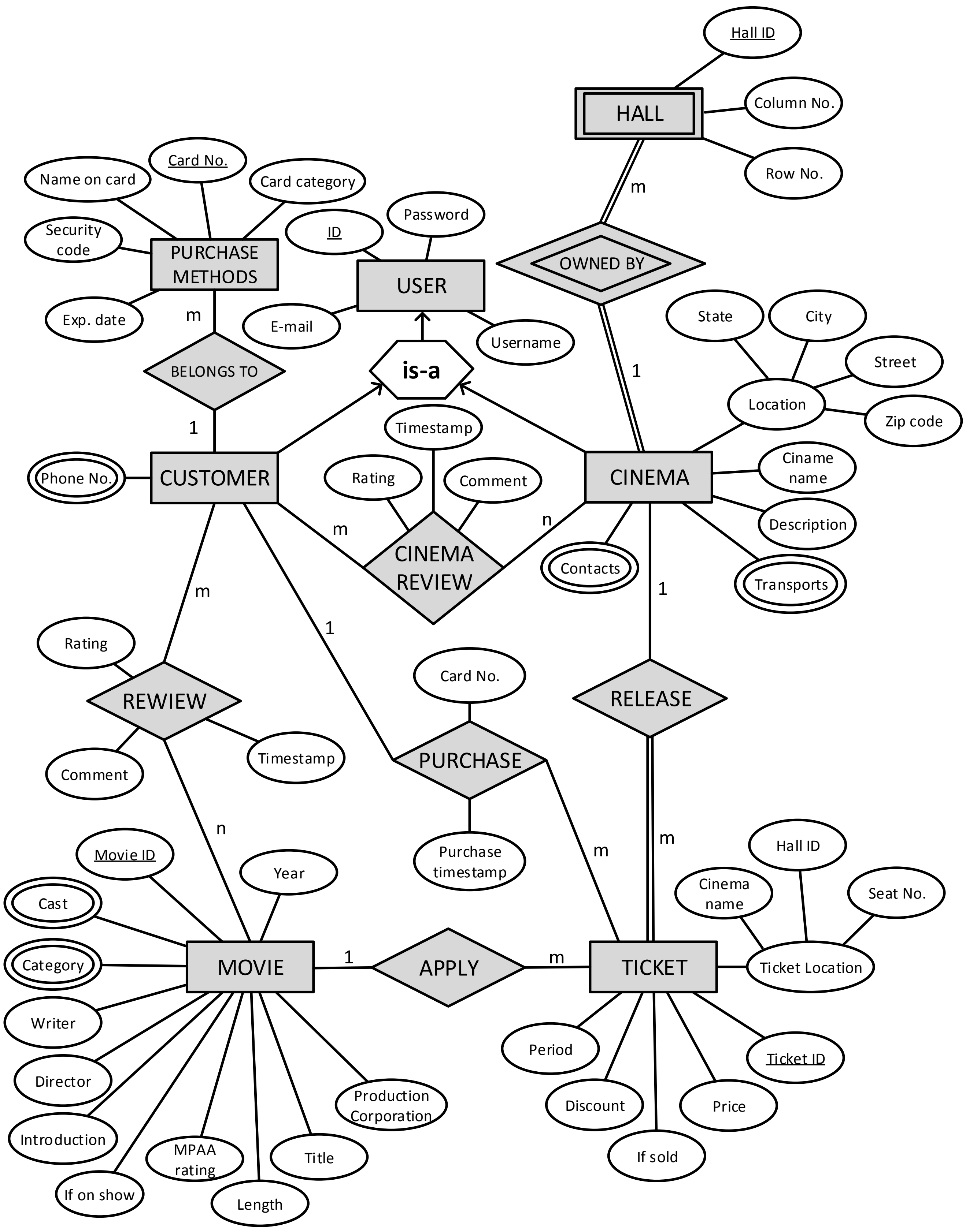


Figure 1. *ER schema diagram of the movie database management system*

In the ER diagram, the primary key is clearly marked with underline for each entity set. For each relationship identified, the cardinalities (1:1, 1:m or m:n) are stated on the entities participating in this relationship.

Part III Data Schema Transformation with DDL Commands

In this section, we performed a transformation from ER diagram to the relevant data schema. For each entity set and relation set, their attributes and specified domains are described. The transformed data schemas with the DDL commands are listed as below

1. USER (ID : integer, Username : varchar, password : varchar, E-mail : varchar);

PRIMARY KEY: ID.

SQL-DDL Command:

```
CREATE TABLE USERS
           ( UserID int NOT NULL,
             Username varchar (40) NOT NULL,
             Password varchar (30),
             Email varchar (40),
             primary key (UserID) );
```

Explanation:

In this command, a table USER is created. UserID is set as primary key which is not allowed to be null. The type of password is varchar with 30 chars at most. Similarly, the type of Email is also varchar with 40 chars at most. CUSTOMER and CINEMA sharing attributes such as username, password and Email are generalized as USER.

2. CUSTOMER (ID : integer);

PRIMARY KEY: ID.

SQL-DDL Command:

```
CREATE TABLE CUSTOMER
          ( Cust_ID int NOT NULL,
            primary key(Cust_ID) );
```

Explanation:

In this command, a table CUSTOMER is created. Cust_ID is set as primary key which is not allowed to be null. CUSTOMER is considered as one category of USER.

3. CUSTOMER_PHONE (<u>Phone No.</u> : varchar, <u>Cust ID</u> : integer);

PRIMARY KEY: Phone_No., Cust_ID;

FOREIGN KEY: Cust_ID referencing **CUSTOMER**;

Note: Multivalued Attribute of CUSTOMER.

SQL-DDL Command:

```
CREATE TABLE CUSTOMER_PHONE
        ( Phone_No varchar(40) NOT NULL,
          Cust_ID int NOT NULL,
          primary key(Phone_No,Cust_ID),
          foreign key(Cust_ID) references CUSTOMER(Cust_ID));
```

Explanation:

Since a customer may have more than one phone number, the phone number is set as a multivalued attribute. Herein, a table CUSTOMER_PHONE is created which sets Phone_No as primary key with a non-null value. Cust_ID is a foreign key referencing to CUSTOMER (Cust_ID) which identifies a specific customer.

4. PURCHACE_METHODS (<u>Card No.</u> : varchar, Card category : varchar, Name on Card : varchar, Security code : varchar, Exp. date: date, <u>Cust ID</u> : integer) ;

PRIMARY KEY: Card No., Cust_ID;

FOREIGN KEY: Cust_ID referencing **CUSTOMER**.

SQL-DDL Command:

```
CREATE TABLE PURCHAE_METHODS
          ( Card_No varchar (40) NOT NULL,
            Card_category varchar (40) NOT NULL,
            Name_On_Card varchar (40) NOT NULL,
            Security_code varchar (40) NOT NULL,
            Exp_Date date NOT NULL,
            Cust_ID int NOT NULL,
            primary key(Card_No,Cust_ID),
            foreign  key(Cust_ID)  references  CUSTOMER(Cust_ID));
```

Explanation:

In this command, a table PURCHAE_METHODS is created with Card_No and Cust_ID as primary key. Cust_ID is a foreign key referencing to CUSTOMER (Cust_ID). Other attributes including card category, name on card, security code, expiration date are necessary to save a tuple.

5. CINEMA (ID : integer, Cinema name : varchar, Location(State: varchar, City : varchar, Street : varchar, Zip code: integer), Description : varchar)

PRIMARY KEY: ID.

SQL-DDL Command:

```
CREATE TABLE CINEMA
     (Cine_ID int NOT NULL,
      Cinema_name varchar(40) NOT NULL,
      State varchar(40) NOT NULL,
      City varchar(40) NOT NULL,
      Street varchar(40) NOT NULL,
      Zip_Code int NOT NULL,
      Description varchar(1000),
      primary key(Cine_ID));
```

Explanation:

In this command, a table CINEMA is created with Cine_ID as primary key being not null. Cust_ID is a foreign key referencing to CUSTOMER (Cust_ID). In order to register for a cinema administrator user, the address information consisting of state, city, street, zip code is needed. CINEMA is considered as the other category of USER.

6. HALL (Cine ID : integer, Hall ID : integer, Column No. : integer, Row No. integer)

PRIMARY KEY: Cine_ID, Hall_ID;

FOREIGN KEY: Cine_ID referencing **CINEMA**.

SQL-DDL Command:

```
CREATE TABLE HALL
         (Cine_ID int NOT NULL,
          Hall_ID int NOT NULL,
          Column_No integer NOT NULL,
          Row_No int NOT NULL,
          primary key(Cine_ID, Hall_ID),
          foreign key(Cine_ID) references CINEMA(Cine_ID));
```

Explanation:

In this command, a table is created for HALL. Cine_ID and Hall_ID are set as primary key which are required to be not null. Cine_ID is the foreign key referencing to CINEMA (Cine_ID). To provide complete information of a hall, a cinema user needs to specify the number of column and row for the seats within.

7. **CINEMA_CONTACT** (Contacts : varchar, Cine ID : varchar)

PRIMARY KEY: Contacts, Cine_ID;

FOREIGN KEY: Cine_ID referencing **CINEMA**;

Note: Multivalued Attribute of CINEMA.

SQL-DDL Command:

```
CREATE TABLE CINEMA_CONTACT
         ( Contacts varchar(40) NOT NULL,
           Cine_ID int NOT NULL,
           primary key(Contacts,Cine_ID),
           foreign key(Cine_ID) references CINEMA(Cine_ID));
```

Explanation:

In this command, a table is created for CINEMA_CONTACT. Contacts and Cine_ID combined forms primary key CINEMA_CONTACT. Cine_ID is a foreign key referencing to CINEMA (Cine_ID) which is used to identify a specific cinema.

8. CINEMA_TRANS (Transports : varchar, Cine ID : varchar)

PRIMARY KEY: Transports, Cine_ID;

FOREIGN KEY: Cine_ID referencing **CINEMA**;

Note: Multivalued Attribute of CINEMA.

SQL-DDL Command:

```
CREATE TABLE CINEMA_TRANS
        ( Transport varchar(40) NOT NULL,
          Cine_ID int NOT NULL,
          primary key(Transport,Cine_ID),
          foreign key(Cine_ID) references CINEMA(Cine_ID));
```

Explanation:

For there may be several ways to get to a certain cinema, transportation information of cinema is set as a multivalued attribute. In this command, we create a table for CINEMA_TRANS. Transport and Cine_ID are both primary key being both not null. Cine_ID is a foreign key referencing to CINEMA (Cine_ID) used to identify a specific cinema.

9. MOVIE (Movie ID : integer, Title : varchar, Year : date, Rated : varchar, Length : integer, If on show : integer, Director : varchar, Writer : varchar, Production Corporation : varchar, Introduction : varchar)

PRIMARY KEY: Movie_ID;

ALTERNATE KEY: Title, Year.

SQL-DDL Command:

```
CREATE TABLE MOVIE
    ( Movie_ID int NOT NULL,
      Title varchar(40) NOT NULL,
      Year int NOT NULL,
      MPAA_rating varchar(20),
      Length  int,
      If_On_Show int,
      Director varchar(40),
      Writer varchar(40),
      Prod_Corp  varchar(40),
      Introduction  varchar(1000),
      primary key(Movie_ID));
```

Explanation:

In this command, a table MOVIE is created with Moive_ID as primary key. It is assumed that the combination of movie's title and release year is also sufficient to identify a movie, thus considered as alternate key.

10. MOVIE_CAST (Cast : varchar, Movie ID : integer)

PRIMARY KEY: Casts, Movie_ID;

FOREIGN KEY: Movie_ID referencing **MOVIE**;

Note: Multivalued Attribute of Movie.

SQL-DDL Command:

```
CREATE TABLE MOVIE_CAST
    ( Cast  varchar(40) NOT NULL,
      Movie_ID   int NOT NULL,
      primary key (Cast, Movie_ID),
      foreign key (Movie_ID) references MOVIE(Movie_ID));
```

Explanation:

For that movie cast needs to be a multivalued attribute, a corresponding new table is established. Cast with Movie_ID is set as primary key. Movie_ID is a foreign key referencing to MOVIE (movie_Id) identifying a specific movie.

11. MOVIE_CATEGORY (Category : varchar, Movie ID : integer)

PRIMARY KEY: Category, Movie_ID;

FOREIGN KEY: Movie_ID referencing **MOVIE**;

Note: Multivalued Attribute of Movie.

SQL-DDL Command:

```
CREATE TABLE MOVIE_CATEGORY
   ( Category   varchar(20) NOT NULL,
     Movie_ID   int NOT NULL,
     primary key (Category, Movie_ID),
     foreign key (Movie_ID) references MOVIE(Movie_ID));
```

Explanation:

By the same token, since a single movie can be categorized into different categories, a table is created for movie category.

12. MOVIE_REVIEW (Cust ID : integer, Movie ID : integer, Timestamp : date, Rating : integer, Comment : varchar)

PRIMARY KEY: Cust ID, Movie ID;

FOREIGN KEY: Movie_ID referencing **MOVIE**;

Cust_ID referencing **CUSTOMER**;

Note: Cardinality m:n relationship.

SQL-DDL Command:

```
CREATE TABLE MOVIE_REVIEW
   ( Cust_ID    int NOT NULL,
     Movie_ID   int NOT NULL,
     Review_timestamp date,
     Rating  int,
     Comments   varchar(1000),
     primary key (Cust_ID, Movie_ID),
     foreign key (Movie_ID) references MOVIE(Movie_ID),
     foreign key (Cust_ID) references CUSTOMER(Cust_ID));
```

Explanation:

Since the cardinality between customer making reviews on movies is m:n, a table of is needed to be established. In this way, we create a MOVIE_REVIEW table. Cust_ID with Movie_ID is the primary key, in which Movie_ID is a foreign key referencing to MOVIE (Movie_ID) and Cust_ID is a foreign key referencing to CUSTOMER (Cust_ID).

13. TICKET (Ticket ID : string, Movie ID : integer, Cine ID : integer, Cust ID : integer, Ticket location(Cinema name : varchar, Hall No.: varchar, Seat No.: varchar), Price : numeric, Discount : float, Period : interval, Purchase timestamp : date, Card No. : varchar, If sold : integer)

PRIMARY KEY: Ticket ID.

SQL-DDL Command:

```
CREATE TABLE TICKET
    ( Ticket_ID  varchar(10) NOT NULL,
      Movie_ID   int NOT NULL,
      Cine_ID    int NOT NULL,
      Cust_ID    int NOT NULL,
      HallNo  varchar(10) NOT NULL,
      SeatNo  varchar(10) NOT NULL,
      Price   numeric(5,2) NOT NULL,
      Discount   numeric(2,2) NOT NULL,
      Period     int NOT NULL,
      Purchase_Time  date NOT NULL,
      Card_No    varchar(30) NOT NULL,
      If_Sold    int NOT NULL,
      primary key (Ticket_ID),
      foreign key (Movie_ID) references MOVIE(Movie_ID),
      foreign key (Cust_ID) references CUSTOMER(Cust_ID),
      foreign key (Cine_ID) references CINEMA(Cine_ID));
```

Explanation:

In this command, a table TICKET is created in which Ticket_ID as primary key is used to uniquely identify a movie ticket. To better form ticket information, Movie_ID, Cust_ID and Cine_ID are referred to as foreign keys and other information is included such as seat location, ticket price, if ticket has been sold and so forth.

14. **CINEMA_REVIEW** (<u>Cine_ID</u> : integer, <u>Cust_ID</u> : integer, Timestamp : date, Rating : integer, Comment : varchar)

PRIMARY KEY: Cine_ID, Cust_ID;

FOREIGN KEY: Cine_ID referencing **CINEMA**;

Cust_ID referencing **CUSTOMER**;

Note: Cardinality m:n relationship.

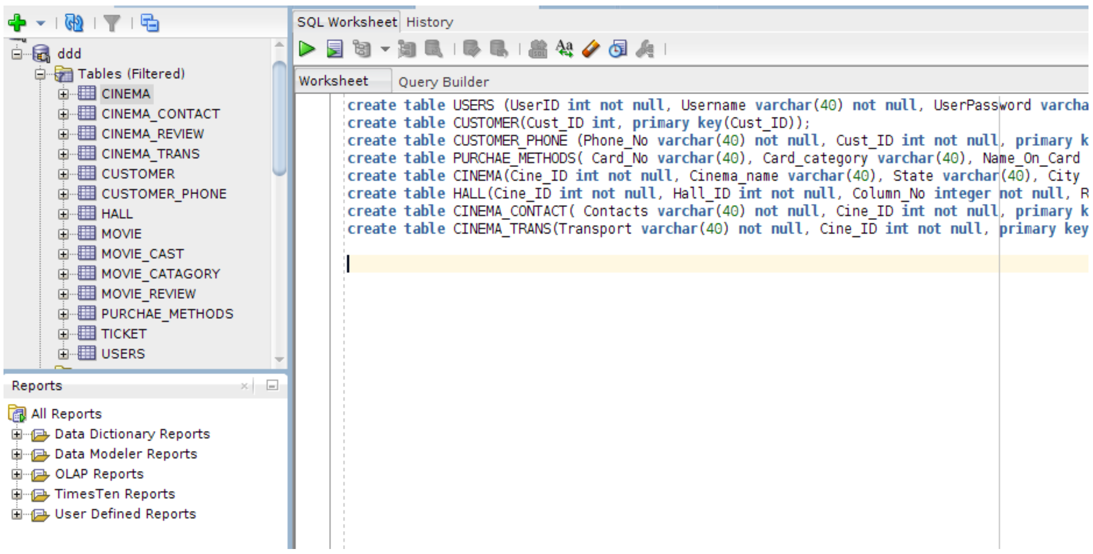
SQL-DDL Command:

```
CREATE TABLE CINEMA_REVIEW
    ( Cine_ID int NOT NULL,
      Cust_ID int NOT NULL,
      Review_timestamp  date,
      Rating  int,
      Comments   varchar(1000),
      primary key (Cine_ID, Cust_ID),
      foreign key (Cine_ID) references CINEMA(Cine_ID),
      foreign key (Cust_ID) references CUSTOMER(Cust_ID));
```

Explanation:

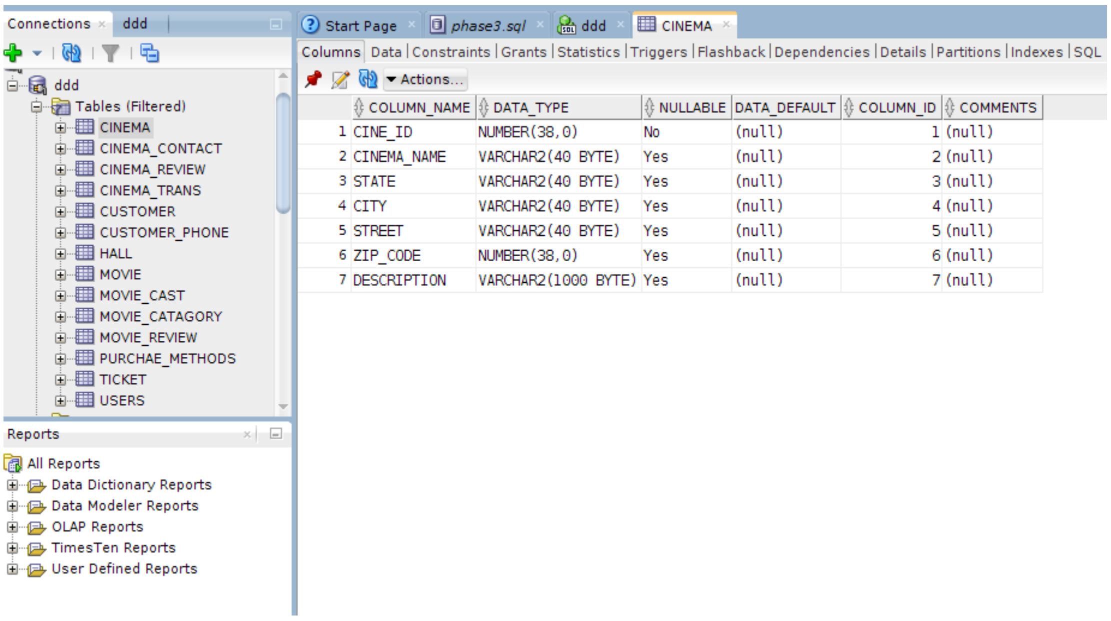
Similar to the movie review table, another cinema review table is created due to the connecting relationship having a cardinality of m:n. Cine_ID with Cust_ID is the primary key, in which Cine _ID is a foreign key referencing to CINEMA (Cust _ID) and Cust _ID is a foreign key referencing to CUSTOMER (Cust_ID).

Furthermore, sample SQL's (DDL) commands are implemented in Oracle offered by CISE server. In the worksheet, the tables are created shown in the snapshot below.



```
create table USERS (UserID int not null, Username varchar(40) not null, UserPassword varcha
create table CUSTOMER(Cust_ID int, primary key(Cust_ID));
create table CUSTOMER_PHONE (Phone_No varchar(40) not null, Cust_ID int not null, primary k
create table PURCHAE_METHODS( Card_No varchar(40), Card_category varchar(40), Name_On_Card
create table CINEMA(Cine_ID int not null, Cinema_name varchar(40), State varchar(40), City
create table HALL(Cine_ID int not null, Hall_ID int not null, Column_No integer not null, R
create table CINEMA_CONTACT( Contacts varchar(40) not null, Cine_ID int not null, primary k
create table CINEMA_TRANS(Transport varchar(40) not null, Cine_ID int not null, primary key
```

As an illustration, the CINEMA table is checked after the implementation of DDL in Oracle.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	CINE_ID	NUMBER(38,0)	No	(null)	1	(null)
2	CINEMA_NAME	VARCHAR2(40 BYTE)	Yes	(null)	2	(null)
3	STATE	VARCHAR2(40 BYTE)	Yes	(null)	3	(null)
4	CITY	VARCHAR2(40 BYTE)	Yes	(null)	4	(null)
5	STREET	VARCHAR2(40 BYTE)	Yes	(null)	5	(null)
6	ZIP_CODE	NUMBER(38,0)	Yes	(null)	6	(null)
7	DESCRIPTION	VARCHAR2(1000 BYTE)	Yes	(null)	7	(null)

Part III Sample Queries DML

In this section, several sample data manipulation language (DML) commands are provided to illustrate the realization sample of expected queries.

Sample Situation 1:

Assuming that a customer user, Feynman, wishes to search for a movie related to "Harry Potter", she types key word '' Harry Potter '' in the search bar. The corresponding DML for this situation is as follows:

```
SELECT Title,Year
FROM MOVIE
WHERE Title like ' % Harry Potter % '
```

Sample Situation 2:

Assume a customer user needs to revise his or her phone contact in the account information. More precisely, Boltzmann, a customer user having a customer user ID 1234, wishes to change his cell phone from # (352)-281-0000 to # (352)-281-0001 . The corresponding DML for this situation gives as follows:

```
UPDATE CUSTOMER_PHONE
SET Phone_No. = ' 3522810001'
WHERE Phone_No. = '3522810000' AND Cust ID = '1234'
```

Sample Situation 3:

The cinema administrator of Gator Cinema wishes to acknowledge the number of tickets of movie "Bird Man" identified by the movie ID 2353 provided in its cinema. The corresponding DML for this situation gives as follows:

```
SELECT count(*)
FROM TICKET AS T JOIN MOVIE AS M
ON T. MOVIE ID= M. MOVIE ID
WHERE M. Title=' Bird Man ' AND M. Movie_ID='2353' AND T.
Cinema_Name='Gator Cinema'
```

Sample Situation 4:

The cinema administrator of Gator Cinema wishes to edit ticket proce of movie "The theory of everything" identified by the movie ID 2354 to 15.00 USD. The corresponding DML for this situation gives as follows:

```
UPDATE Ticket
SET Price='15.00'
WHERE Title=' The theory of everything ' AND Movie_ID='2354'
AND Cinema_name='Gator cinema'
```