# Homework#2

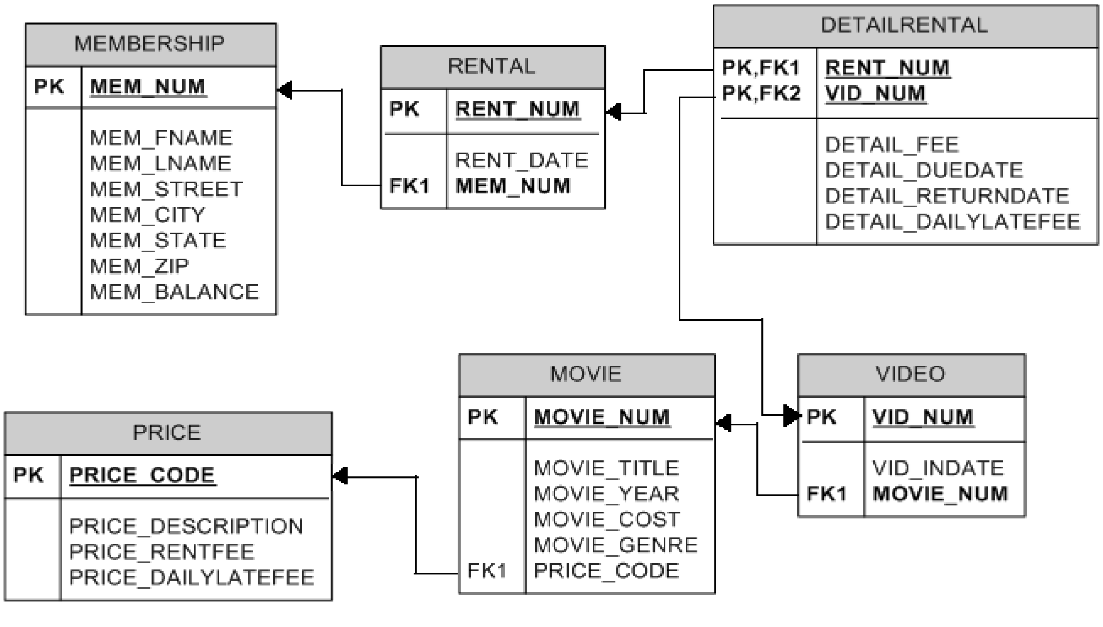
ACS575 Database Systems, Spring 2024

**Due: Feburary 23**

- For this homework submission, prepare two files:
  (1) YourLastName_YourFirstName_ACS575_HW2. doc (or .pdf) and
  (2) YourLastName_YourFirstName_ACS575_HW2_Output.doc (or .pdf)
  In the first file, **shows your answers, including SQL statements for each question**.
  In the second file, **present the execution results of your SQL statements. This can be the direct output or a clear screenshot of the results.**
- Label your answer clearly with the corresponding question number such as Part I Q1, Q2, etc.

## Part I. Table Creation

1. Construct tables using the provided relational schemas. **Incorporate all the specified integrity constraints like domain constraints and referential integrity constraints during table creation.** Submit the SQL statements you employed for this creation.



* In VIDOE relation, VID_INDATE refers to the date when the video was added to the store's inventory or collections.

Table: PRICE

| Column | Data Type and Length | Constraint |
|---|---|---|
| PRICE_CODE | NUMBER(2,0) | PK |
| PRICE_DESCRIPTION | VARCHAR2(20) | NOT NULL |
| PRICE_RENTFEE | NUMBER(5,2) | CHECK (PRICE_RENTFEE >= 0) |
| PRICE_DAILYLATEFEE | NUMBER(5,2) | CHECK (PRICE_DAILYLATEFEE >= 0) |

Table: MOVIE

| Column | Data Type and Length | Constraint |
|---|---|---|
| MOVIE_NUM | NUMBER(8,0) | PK |
| MOVIE_TITLE | VARCHAR2(75) | NOT NULL |
| MOVIE_YEAR | NUMBER(4,0) | CHECK (MOVIE_YEAR > 1900) |
| MOIVE_COST | NUMBER(5,2) | |
| MOVIE_GENRE | VARCHAR2(50) | |
| PRICE_CODE | NUMBER(2,0) | FK  references PRICE |

Table: VIDEO

| Column | Data Type and Length | Constraint |
|---|---|---|
| VID_NUM | NUMBER(8,0) | PK |
| VID_INDATE | DATE | |
| MOVIE_NUM | NUMBER(8,0) | FK references MOVIE |

Table: MEMBERSHIP

| Column | Data Type and Length | Constraint |
|---|---|---|
| MEM_NUM | NUMBER(8,0) | PK |
| MEM_FNAME | VARCHAR2(30) | NOT NULL |
| MEM_LNAME | VARCHAR2(30) | NOT NULL |
| MEM_STREET | VARCHAR2(120) | |
| MEM_CITY | VARCHAR2(50) | |
| MEM_STATE | CHAR(2) | |
| MEM_ZIP | CHAR(5) | |
| MEM_BALANCE | NUMBER(10,2) | |

Table: RENTAL

| Column | Data Type and Length | Constraint |
|---|---|---|
| RENT_NUM | NUMBER(8,0) | PK |
| RENT_DATE | DATE | DEFAULT SYSDATE |
| MEM_NUM | NUMBER(8,0) | FK references MEMBERSHIP |

Table: DETAILRENTAL

| Column | Data Type and Length | Constraint |
|---|---|---|
| RENT_NUM | NUMBER(8,0) | PK, FK1 references RENT |
| VID_NUM | NUMBER(8,0) | PK, FK2 references VIDEO |
| DETAIL_FEE | NUMBER(5,2) | |
| DETAIL_DUEDATE | DATE | |
| DETAIL_RETURNDATE | DATE | |
| DETAIL_DAILYLATEFEE | NUMBER(5,2) | |

2. Input data into the tables using the provided **EliteVideoData.sql** file.

## Part II. SQL Query

For each of the following queries, write a SQL statement and provide the execution result. **If there are specific instructions provided in the question, ensure your answer adheres to them.**

*Selection and Projection*

**1.** Retrieve the movie number, title, cost, and genre of movies that are priced under 50 and belong to either the “ACTION” or “COMEDY” genres. Order the results in ascending order based on the genre.

(**NOTE**: Use the IN operator, even though it can be achieved using the OR conjunction.)

**2.** Retrieve the movie title, year, and cost of all movies with a title that includes the word “hope” in any case variation, such as “Hope”, “HOPE”, “hOpe” . Ensure your query is case-insensitive to capture all variations.

(**NOTE**: For this query, utilize a single-block query structure.)

**3.** Provide the last name, house number, street name and city of all members located in Tennessee (TN). The results should display the house number and street name as separate columns, as shown below.

```
 LAST NAME       HOUSE NO       STREET NAME        CITY
-------------------  ------------------  ------------------------  -----------------
```

*Aggregation*

**4**. Display the genre name along with the total count of movies for each genre. Sort the results with the genre having the highest movie count at the top and the one with the least count at the bottom.

```
 Movie Genre        Number of Movies
-----------------     ---------------------------
```

**5.** Display the video number and its rental frequency for all videos that have been rented a minimum of 2 times.

**6.** Identify the year in which the highest number of movies were added to the database. Provide both the year and the corresponding movie count.

(**NOTE**: You might need to sort the count of movies per year and select the top entry. Use ROWNUM in Oracle, or LIMIT in other RDBMS. )

*Nested queries*

**7.** List the membership number, first name, last name, and zip code of all members who have not rented any videos.

(**NOTE**: Use a nested query structure for this task, even though a JOIN could also achieve the same result.)

**8.** Identify all members who have rented more than 2 videos. Display their membership numbers in the results.

**9.** Identify movies that have never been rented. Display their movie titles in the result.

(**NOTE**: Consider using a subquery structure combined a join operation)

*Functions and nested queries*

**10.** Display the minimum, maximum and average balances of memberships that have made rentals. Round the average balance to two decimal places.

(**NOTE**: Consider using a nested query approach with the EXISTS operator, even though a join operation could achieve the same result. )

```
  Minimum Balance        Maximum Balance       Average Balance
------------------------  ------------------------  ----------------------
```

*Join I*

**11.** List the movie title, genre, price description, and rental fee of all movies.

(**NOTE**: Utilize the join operator for this query)

**12.** Calculate the total number of videos available for each movie title.

```
MOVIE_TITLE          TOTAL VIDEOS
----------------------   -------------------------
```

**13**. List the full names of members (formatted as first name followed by |last name), the movie titles they've rented, and the due dates associated with each rental. Order the results by the members' full names.

```
NAME     MOVIE_TITLE      DETAIL_DUEDATE
---------    ---------------------    ----------------------------
```

**14**. Display the rental number, rental date, video number, movie title, due date, and return date for <u>all videos returned past their due date</u>. Order the query result by rental number followed by movie title**.**

(**NOTE**: Construct this using a single-block query with JOIN. Avoid using nested query structures for this implementation)

*Join II (Outer join)*

**15.** Retrieve the membership number, first name, last name, and the total number of rentals. <u>Include members who haven't rented any videos</u>, displaying their rental count as 0. Order the results by total rentals in descending order.

**16**. For each movie, determine if it has been rented out, and if so, provide the name of the last member who rented it and the corresponding rental date.

(**NOTE**: Utilize outer joins to ensure all movies are included, and group the results to display only the latest rental information for each movie and apply MAX function. For movies that haven't been rented, the latest rental information will appear as null).

```
MOVIE_TITLE     MEM_FNAME       MEM_LNAME      LAST_RENTAL_DATE
--------------------    --------------------      --------------------     --------------------------
```

*Set operators*

**17.** List the membership number, first name, and last name of members who have rented both movies "Richard Goodhope" and "Beatnik Fever". These movies may have been rented at different times with separate rental numbers.

(**NOTE**: Implement this query using a set operator)

## Part III. View

The video store wants a unified view of all the movies rented in the current year, with details including movie title, genre, rental fee, rental date, due date, return date, and member name. They want this information easily accessible for reporting and analysis.

**1.** Create a view named with **YearlyMovieRentals**) to facilitate this requirement. The view should list out the following columns: movie_title, movie_genre, rental_fee (sourced from detail_fee from the appropriate table), detail_duedate, detail_returndate, mem_fname, and lname.
(**NOTE**: Extract the current year information from the system date.)

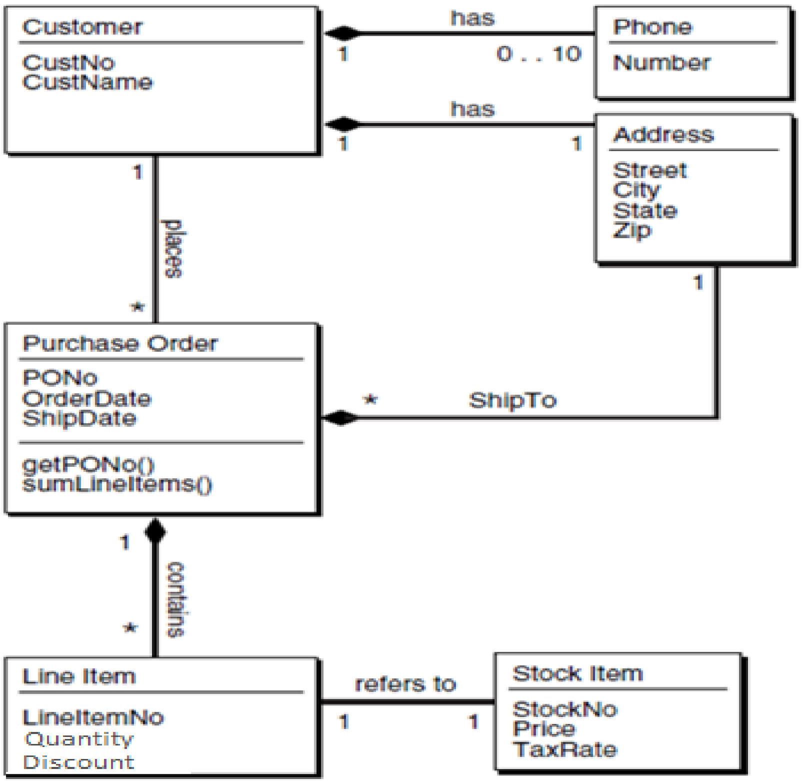
**2.** List all movies rented by members with the last name 'Knight' ('KNIGHT') this year from the YearlyMovieRentals view.

**3**. Which members have rented movies of the genre 'Action' ('ACTION') this year? Display the distinct first and last names of these members from the YearlyMovieRentals view.

## Part IV. Object-Relational Database and SQL3

The purpose of Part III is familiar with key concepts of ORDBMS and extended SQL (SQL-3 /SQL1999).

The figure below shows the UML class diagram of a purchase order application.
(Reference: https://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjxmp.htm#CHDGGDIF)



- Instead of breaking up addresses or multiple phone numbers into unrelated columns in relational tables, we define types to represent an entire address and an entire list of phone numbers.
- Similarly, we use nested tables to keep line items with their purchase orders instead of storing them separately.
- The main entities—customers, stock, and purchase orders—become object types.
- Object references are used to express some of the relationships among them.
- Collection types—varrays and nested tables—are used to model multi-valued attributes.

**For each question below, a SQL template(s) is provided. (i) Complete the SQL statement(s) required for each question and (ii) Demonstrate that your SQL statements function correctly by submitting output (e.g., a copy of the result or a screenshot).**

**For hands-on practice, the SQL syntax is tailored for Oracle SQL. Should you choose to use a different ORDBMS, modifications are necessary to ensure compatibility.**

1. **Defining Types:** Create object types below using CREATE TYPE statements.

**(1)** `Address_objtyp` type

```
CREATE TYPE ________________________(
   Street VARCHAR2(200),
   City VARCHAR2(200),
   State CHAR(2),
   Zip VARCHAR2(20)
);
```

**(2)** `PhoneList_vartyp` type

/* Any instance of type `PhoneList_vartyp` is a varray of up to 10 telephone numbers, each represented by a data item of type `VARCHAR2` with length 20 . */

```
CREATE TYPE ____________ AS VARRAY(10) _____________;
```

**(3)** `Customer_objtyp` type

`/* Address_obj` attribute has an `Address_objtyp` object, and `PhoneList_var` attribute has a `PhoneList_vartyp` object. */

```
CREATE TYPE ________________________(
    CustNo NUMBER,
    CustName VARCHAR2(200),
    Address_obj _____________,
    PhoneList_var ____________
) NOT FINAL;
```

**(4)** StockItem_objtyp type

/* This LineItem_objtyp type has three numeric attributes. */

```
CREATE TYPE ______________________ (
  StockNo NUMBER,
  Price   ______,
  TaxRate ______
);
```

**(5)** LineItem_objtyp type

/* This LineItem_objtyp type models the line item entity, and includes an object reference to the corresponding stock object. All other attributes are numeri attributes.*/

```
CREATE TYPE _____________________ (
  LineItemNo ______,
  Stock_ref ________________,
  Quantity ______,
  Discount ______
);
```

**(6)** LineItemList_ntabtyp type

/* Instance of LineItemList_ntabtyp type is a nested table object, each row of which contains an object of type LineItem_objtyp. */

```
CREATE TYPE _____________ AS TABLE OF _____________ ;
```

**(7)** PurchaseOrder_objtyp type

/* Attribute Cust_ref is a REF to Customer_objtyp, attribute LineItemList_ntab is a nested table of type LineItemList_ntabtyp, and attribute ShipToAddr_obj has an Address_objtyp object. */

```
CREATE TYPE _________________ AUTHID CURRENT_USER AS OBJECT (
    PONo NUMBER,
    Cust_ref _________________,
    OrderDate DATE,
    ShipDate DATE,
    LineItemList_ntab ________________,
    ShipToAddr_obj _______________,
    MAP MEMBER FUNCTION
      getPONo RETURN NUMBER,
    MEMBER FUNCTION
      sumLineItems RETURN NUMBER
 );
```

**2**. **Method Definitions:** Create the twomember functions getPONo and sumLineItems of type PurchaseOrder_objtyp.

```
CREATE OR REPLACE TYPE BODY PurchaseOrder_objtyp
AS
```

/* The getPONo method simply returns the value of the PONo attribute – namely, the purchase order number –of whatever instance of the type PurchaseOrder_objtyp that calls the method.*/

```
MAP MEMBER FUNCTION getPONo RETURN NUMBER
IS
  BEGIN
     ____________;
  END;
```

/* The sumLineItems method is to return the sum of the values of the line items of its associated PurchaseOrder_objtyp object. */

```
MEMBER FUNCTION sumLIneItems RETURN NUMBER
IS
     i INTEGER;
     StockVal StockItem_objtyp;
     Total NUMBER := 0;
  BEGIN
     FOR i in 1 .. SELF.LineItemList_ntab.COUNT LOOP
       UTL_REF.SELECT_OBJECT
              (LineItemList_ntab(i).Stock_ref, StockVal);
       Total :=_____
              + SELF.LineItemList_ntab(i).______ * StockVal.____;
     END LOOP;
     RETURN total;
  END;
END;
/
```

**3. Creating Object Tables:** Creating an object type and creating a table are distinct steps. Defining a type merely establishes a logical structure without allocating storage. Proceed to create the tables below, each based on its corresponding object type.

**(1)** Customer_objtab table

/* The Customer_objtab table is used to hold objects of type Customer_objtyp. CustNo is the primary key of the table. */

```
CREATE TABLE ___________ OF ___________ (______ PRIMARY KEY)
    OBJECT IDENTIFIER IS PRIMARY KEY;
```

**(2)** Stock_objtab table

/* The Stock_objtab table is for StockItem_objtyp objects. StockNo is the primary key of the table. */

```
CREATE TABLE ____________ OF _____________ (______ PRIMARY KEY)
OBJECT IDENTIFIER IS PRIMARY KEY;
```

**(3)** PurchaseOrder_objtab table

/* The PurchaseOrder_objtab table is for PurchaseOrder_objtyp objects. PONo is the primary key and Cust_ref is a foreign key which references Customer_objtab table. */

```
CREATE TABLE PurchaseOrder_objtab OF ___________________ (
   PRIMARY KEY (PONo),
   FOREIGN KEY (Cust_ref) ________ _____________)
   OBJECT IDENTIFIER IS PRIMARY KEY
   NESTED TABLE LineItemList_ntab STORE AS PoLine_ntab (
   (PRIMARY KEY(NESTED_TABLE_ID, LineItemNo))
   ORGANIZATION INDEX COMPRESS)
   RETURN AS LOCATOR
;
ALTER TABLE PoLine_ntab
ADD (SCOPE FOR (Stock_ref) IS stock_objtab) ;
```

## 4. Inserting Values.

**(1)** Inserting values in Stock_objtab

```
INSERT INTO Stock_objtab VALUES(1004, 150.00, 0.05) ;
```

Also insert three more records <1011, 200.00, 0.07>, <1534, 100.00, 0.05> and <1535, 300.00, 0.04> into Stock_objtab

**(2)** Inserting values in Customer_objtab

```
INSERT INTO Customer_objtab
VALUES (
     1, 'Kaye Pitcher',
     Address_objtyp('2101 E. Coliseum Blvd', 'Fort Wayne', 'IN',
                    '46805'),PhoneList_vartyp('260-481-6803')
) ;
```

Also insert two more record,

```
      <2, 'Jane Smith',
           Address_objtyp('456 Maple Ave', 'Lincoln', 'NE', '68502'),
           PhoneList_vartyp('609-555-9012')>
      <3, 'John Nike',
           Address_objtyp('323 College Drive', 'Edison', 'NJ', '08820'),
           PhoneList_vartyp('609-555-1212','201-555-1212')>
```

**(3)** Inserting the order values of customer number “1” with PONo “5001” in `PurchaseOrder_objtab`

```
 INSERT INTO PurchaseOrder_objtab
   SELECT 5001, REF(C), '01-Jan-24', '08-Jan-24', LineItemList_ntabtyp(),
          NULL
   FROM Customer_objtab C
   WHERE _____________;
```

And inserting two order items of the purchase order “5001” in the nested table `LineItemList_ntab` using the following insert statements.

```
INSERT INTO TABLE (
   SELECT P.LineItemList_ntab
   FROM PurchaseOrder_objtab P
   WHERE P.PONo = 5001
)
SELECT 01, REF(S), 12, 10
FROM Stock_objtab S
WHERE S.StockNo = 1534 ;

INSERT INTO TABLE (
   SELECT P.LineItemList_ntab
   FROM PurchaseOrder_objtab P
   WHERE P.PONo = 5001
)
SELECT 02, REF(S), 1, 0
FROM Stock_objtab S
WHERE S.StockNo = 1004;
```

**(4)** Similarly, inserting the order values of customer number “2” with PONo “5002” in `PurchaseOrder_objtab`

```
 INSERT INTO PurchaseOrder_objtab
   SELECT 5002, REF(C), '02-Feb-24','09-Feb-24', LineItemList_ntabtyp(),
          NULL
   FROM Customer_objtab C
   WHERE _____________;
```

And inserting one order item of the purchase order “5001” in the nested table `LineItemList_ntab` using the following insert statement.

```
INSERT INTO TABLE (
   SELECT P.LineItemList_ntab
   FROM PurchaseOrder_objtab P
   WHERE P.PONo = 5002
)
SELECT 01, REF(S), 3, 5
FROM Stock_objtab S
WHERE S.StockNo = 1535;
```

## 5. Querying

**(1)** Query all purchase order numbers.

```
_____________
______________________;
```

**(2)** Query customer and line item data for purchase order 5001.

```
SELECT DEREF(p.Cust_ref), p.ShipToAddr_obj, p.PONo,
        p.OrderDate, LineItemList_ntab
FROM ______________
WHERE ______________;
```

**(3)** Query total value of each purchase order using a member function, `sumLineItems()`.

```
SELECT p.PONo, ____________
FROM ____________________ ;
```

**(4)** Query the stock items with the highest tax rate.

```
SELECT *
FROM Stock_objtab
WHERE ____________________ ;
```

**(5)** Query all customers who have made a purchase order after Feburary 01, 2024 .

```
SELECT c.*
FROM Customer_objtab c, PurchaseOrder_objtab p
WHERE __________________________________________ ;
```