CS 480  
Database Systems  
Fall 2019

**Midterm #2: Tuesday, October 29th**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ LAST 4 DIGITS OF ID: \_\_\_\_\_\_\_\_\_\_\_\_

# Short Answer (10 points)

1. In the attached Olympics database, you would like the public to be able to look at the up to date list of events, but you don’t want to have to maintain an entirely separate database. Describe how you might resolve this issue. (3 points)

CREATE Role – public user

CREATE View - Events

GRANT SELECT to public user ON Events

1. When setting up a database, sometimes we set up primary keys and foreign keys. Why do we use each of them? (3 points)  
     
   Primary Keys – Guarantees uniqueness – identifies a specific row of the table – fast lookup

Foreign Keys – To rejoin two tables that were split apart by the database design – guarantee existence of referred to entity

1. If you have a foreign key reference to a row in another table, and a delete command is executed on that row, what effects can occur? List each possible effect, and the command in the used to set up that reaction. (2 points)

RESTRICT – Prevent the delete from occurring – NO ACTION  
CASCADE – also delete the row from this table  
SET NULL – set the values of the foreign key attributes to NULL

SET DEFAULT – set the values of the foreign key attributes to the defaults for those columns

1. There are 4 primary operations that we can perform on the data **within** a table in SQL. What are these four operations, and what SQL keywords do we invoke to perform these operations? (2 points)

CRUD

CREATE – READ – UPDATE - DESTROY

SELECT

DELETE

INSERT

UPDATE

# SQL Schema Definition (30 points)

On the handout there are tables showing a limited selection from a database. Write a sequence of SQL statements to do the following based on the schema used by those tables:

1. Create a database named Olympics, and begin using that Database. (3 points)

DELETE DATABASE IF EXISTS Olympics  
  
CREATE DATABASE Olympics;

USE Olympics;  
  
  
  
Each table should contain a primary key, and fields with appropriate data types. Do not actually populate the table with the data from the attached handout.

1. Write a sequence of SQL statements to build the table Events. (6 points)

CREATE TABLE Events

(EventID INT AUTO\_INCREMENT,

EventName VARCHAR(15) UNIQUE,

EventArena INT,

ScoringType VARCHAR(7),

PRIMARY KEY (EventID)

);

1. Write a sequence of SQL statements to build the table Participants. (6 points)

create table Participants

(AthleteID int,

EventID int,

Medal varchar(6),

primary key (AthleteID,EventID),

foreign key (AthleteID) references Athletes(AthleteID)

on delete cascade,

foreign key (EventID) references Events(EventID)

on delete cascade

);

1. Add the athlete Flash Frost competing for Canada to your Olympics Database. (3 points)

//Assume that ID is auto incremented  
INSERT INTO Athletes VALUES (NULL, ‘Flash’, ‘Frost’, ‘Canada’)

INSERT INTO Athletes(Athlete\_FName, Athlete\_LName, Country) VALUES (‘Flash’, ‘Frost’, ‘Canada’)

INSERT INTO Athletes VALUES (999, ‘Flash’, ‘Frost’, ‘Canada’)

1. Enroll the athlete Flash Frost in the Slalom event. Do not make assumptions about IDs. (3 points)  
     
   INSERT INTO Participants(AthleteID, EventID)

(SELECT AthleteID, EventID

FROM Athletes, Events

WHERE Athlete\_FName=’Flash’ AND Athlete\_LName=’Frost’ AND EventName=’Slalom’);

(SELECT AthleteID, EventID INTO @A, @E

FROM Athletes, Events

WHERE Athlete\_FName=’Flash’ AND Athlete\_LName=’Frost’ AND EventName=’Slalom’);

(SELECT AthleteID INTO @A

FROM Athletes

WHERE Athlete\_FName=’Flash’ AND Athlete\_LName=’Frost’);

(SELECT EventID INTO @E

FROM Events

WHERE EventName=’Slalom’);

INSERT INTO Participants(AthleteID, EventID) (@A, @E)

1. Record a time of 1:23 for the first run of the slalom event by Flash Frost. (3 points)

INSERT INTO TimedRun

(SELECT AthleteID, EventID, ‘1:23.11’, 1

FROM Athletes, Events

WHERE Athlete\_FName=’Flash’ AND Athlete\_LName=’Frost’ AND EventName=’Slalom’);  
  
INSERT INTO TimedRun (@A, @E, ‘1:23’, 1)

1. Upon judicial review, Flash Frost was found to have failed to pass one of the gates on their first run and should have received a time penalty. Update their first run of the Slalom event to have a time of 1:33. (3 points)  
     
     
   UPDATE TimedRUN

SET Time = 1:33  
WHERE AthleteID = @A AND EventID=@E AND RunNumber = 2

UPDATE TimedRUN

SET Time = 1:33  
WHERE AthleteID = (SELECT AthleteID

FROM Athletes

WHERE Athlete\_FName=’Flash’ AND Athlete\_LName=’Frost’)

AND EventID=(SELECT EventID

FROM Events

WHERE EventName=’Slalom’)

AND RunNumber = 2

1. Further debates have led to the run being struck from the record and the event will be restarted. Remove the Slalom event and all associated runs from the database. (3 points)

-- IF ON DELETE CASCADE is defined

DELETE FROM Events WHERE EventName = ‘Slalom’

-- ELSE  
(SELECT EventID INTO @E

FROM Events

WHERE EventName=’Slalom’);

DELETE FROM TimedRuns WHERE EventID=@E;  
 DELETE FROM Participants WHERE EventID=@E;  
 DELETE FROM Events WHERE EventID=@E;

# Interpreting SQL queries (18 points)

1. What is the result of the following query based on the Database SimpleDB tables? Show the exact answer; include attribute names. (6 points)

SELECT Table2.Artist, Table1.Author

FROM Table1

NATURAL JOIN Table2

ORDER BY Table2.Artist DESC;

|  |  |
| --- | --- |
| Artist | Author |

1. What is the result of the following query based on the Database SimpleDB tables? Show the exact answer; include attribute names. (6 points)

SELECT Table1.Publisher, Length, Table1.Author

FROM Table1

INNER JOIN Table2 ON Table1.Publisher = Table2.Publisher

ORDER BY Length ASC;

|  |  |  |
| --- | --- | --- |
| Publisher | Length | Author |
| Media House | 3:47 | Shel Silverstein |
| Media House | 3:47 | Patrick Rothfuss |
| Nisdey | 4:11 | George R. R. Martin |

1. What is the result of the following query based on the Database Olympics tables? Show the exact answer; include attribute names. (6 points)

SELECT Athletes.AthleteFName, Participants.EventID

FROM Athletes  
LEFT JOIN Participants ON Athletes.AthleteID = Participants.AthleteID

ORDER BY Atheletes.AthleteFName

|  |  |
| --- | --- |
| **AthleteFName** | **EventID** |
| Emily | NULL |
| Flash | 4 |
| George | 5 |
| Jane | 2 |
| Jane | 4 |
| Rosie | NULL |
| Viktor | 1 |

# Writing SQL queries (42 points)

For the rest of the exam use the Olympics Schema defined on the attached worksheet. These queries should be written relative to the schema defined, not the specific data that is shown as a sample.

1. List the number of Events at the 2022 Winter Olympics. This is the number of events in the Olympics database, a new database is created for each year. (3 points)  
     
   SELECT COUNT(\*) FROM Events;
2. For each event, list the name of the event and the number of Athletes participating in that event. (6 points)

SELECT EventName, COUNT(\*)

FROM Events LEFT JOIN Participants USING (EventID)  
GROUP BY EventName

1. List the names of each event along with the first and last name for the gold medalist for that event. If an event has no gold medalist, leave those fields as NULL. (6 points)  
     
     
   SELECT EventName, Athlete\_FName, Athlete\_LName

FROM Events LEFT JOIN (SELECT EventID, Athlete\_FName, Athlete\_LName

FROM Participants NATURAL JOIN Athletes

-- JOIN ON Participants.Athlete\_ID = Athletes.Athlete\_ID

-- JOIN USING (Athlete\_ID

WHERE Medal=’Gold’) T

ON Events.EventID = T.EventID

1. Suppose the SQL variable @A contains an athleteID.  
   Write a query getting the times of all runs athlete @A has participated in. (6 points)  
     
   SELECT Time

FROM TimedRuns

WHERE AthleteID = @A

1. Write a query returning for each country the name of the country, and the number of gold, silver, and bronze medals earned by athletes competing for that country. Your result should contain 4 columns. Order the results from most to fewest gold medals. (6 points)  
     
   SELECT Country, COUNT(Medal) Medals

FROM Athletes JOIN Participants USING (Athlete\_ID)  
WHERE Medal IS NOT NULL

GROUP BY Country;

SELECT Country, GoldMedals, SilverMedals, BronzeMedals

FROM

(SELECT Country, COUNT(\*) GoldMedals

FROM Athletes JOIN Participants USING (Athlete\_ID)  
WHERE Medal = ’Gold’

GROUP BY Country) AS Golds

,

(SELECT Country, COUNT(\*) SilverMedals

FROM Athletes JOIN Participants USING (Athlete\_ID)  
WHERE Medal = ‘Silver’

GROUP BY Country) AS Silvers

,

(SELECT Country, COUNT(\*) BronzeMedals

FROM Athletes JOIN Participants USING (Athlete\_ID)  
WHERE Medal = ‘Bronze’

GROUP BY Country) AS Bronzes

WHERE Bronzes.Country = Silvers.Country AND Silvers.Country = Golds.Country

WITH Countries AS

(SELECT DISTINCT Country  
FROM Athletes)  
  
  
  
SELECT Country, SUM(Medal=’Gold’) Golds, SUM(Medal=’Silver’) Silvers, SUM(Medal=’Bronze’) Bronzes

FROM Athletes JOIN Participants USING (AthleteID)  
GROUP BY Country

1. Create a view by the name TopAthletes containing the names of the top 5 events by number of gold medals earned by athletes participating in that event. (6 points)

CREATE VIEW TopAthletes AS

(SELECT EventName -- , COUNT(\*) NumberOfGoldMedals

FROM Participants NATURAL JOIN Events

WHERE Medal = ‘Gold’

GROUP BY EventName

ORDER BY COUNT(\*) DESC

LIMIT 5);

1. Suppose the SQL variable @E contains an eventID.  
   Write a query returning the first and last names, alongside the average time among their runs of athletes participating in event @E. (3 points)  
     
     
   SELECT Athlete\_FName, Athlete\_LName, AVG(Time)

FROM Athletes NATURAL JOIN TimedRun NATURAL JOIN Events  
WHERE EventID = @E  
GROUP BY Athlete\_FName, Athlete\_LName -- AthleteID

SELECT Athlete\_FName, Athlete\_LName, AVGTIME  
FROM Athletes LEFT JOIN

(SELECT AthleteID, AVG(Time) AVGTIME

FROM Athletes NATURAL JOIN TimedRun NATURAL JOIN Events  
WHERE EventID = @E  
GROUP BY AthleteID) T USING (AthleteID)

1. Write a query returning the average of the average time per athlete (get the average time for each athlete, then take the average of those values) of competitors for the event with id @E. This result should be a single value. (6 points)  
     
   SELECT AVG(AVGTIME)

FROM  
(SELECT Athlete\_FName, Athlete\_LName, AVG(Time) AVGTIME

FROM Athletes NATURAL JOIN TimedRun NATURAL JOIN Events  
WHERE EventID = @E  
GROUP BY Athlete\_FName, Athlete\_LName – AthleteID) T

**EXTRA CREDIT**: Write SQL to construct an additional table relevant to the Olympics database and populate it with three rows of data (made up is expected)

**Database Olympics**

**Athletes**

|  |  |  |  |
| --- | --- | --- | --- |
| **AthleteID** | **AthleteFName** | **AthleteLName** | **Country** |
| 111 | Jane | Artielo | Belgium |
| 241 | Flash | Gordan | USA |
| 367 | Rosie | Beestinger | China |
| 440 | George | Lucas | USA |
| 501 | Viktor | Krum | Russia |
| 715 | Emily | Sweeney | Brazil |

**Participants**

|  |  |  |
| --- | --- | --- |
| **AthleteID** | **EventID** | **Medal** |
| 111 | 4 | NULL |
| 241 | 4 | Bronze |
| 111 | 2 | Gold |
| 440 | 5 | Silver |
| 501 | 1 | NULL |

**Events**

|  |  |  |  |
| --- | --- | --- | --- |
| **EventID** | **EventName** | **EventArena** | **ScoringType** |
| 1 | Slalom | 1 | Time |
| 2 | Long Jump | 1 | Score |
| 3 | Luge | 4 | Time |
| 4 | Figure Skating | 3 | Score |
| 5 | Curling | 2 | Points |

**TimedRun**

|  |  |  |  |
| --- | --- | --- | --- |
| **AthleteID** | **EventID** | **Time** | **RunNumber** |
| 501 | 1 | 1:01 | 1 |
| 501 | 3 | 1:32 | 1 |
| 367 | 1 | 1:15 | 1 |
| 241 | 3 | 1:52 | 1 |
| 241 | 3 | DQ | 2 |
| 241 | 3 | 1:41 | 3 |

**Database SimpleDB**

**Table1**

|  |  |  |
| --- | --- | --- |
| Publisher | Title | Author |
| Infoviz | The Silmarillion | J. R. R. Tolkien |
| Media House | The Giving Tree | Shel Silverstein |
| Media House | The Wise Man’s Fear | Patrick Rothfuss |
| Nisdey | A Song of Fire and Ice | George R. R. Martin |

**Table 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Publisher | Artist | Length | Title |
| Betamax Plus | Katrina and the Waves | 3:16 | Walking on Sunshine |
| Betamax Plus | The Beatles | 2:38 | Yellow Submarine |
| Nisdey | Bon Jovi | 4:11 | Livin’ on a Prayer |
| Media House | Kristen Anderson-Lopez and Robert Lopez | 3:47 | Let It Go |
| Disc Media | Beyoncé | 3:52 | Sorry |