CPSC 304 Project Cover Page

Milestone #: ___4___

Date: __Nov. 28, 2021___

Group Number: ____14_____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Doris Sun	41134776	d1w5a	doris.sun@queensu.ca
Raymond Zhang	86395571	s8b3b	raymondz6011@gmail.com
Michelle Kim	55441778	k3x2b	yeojin011016@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

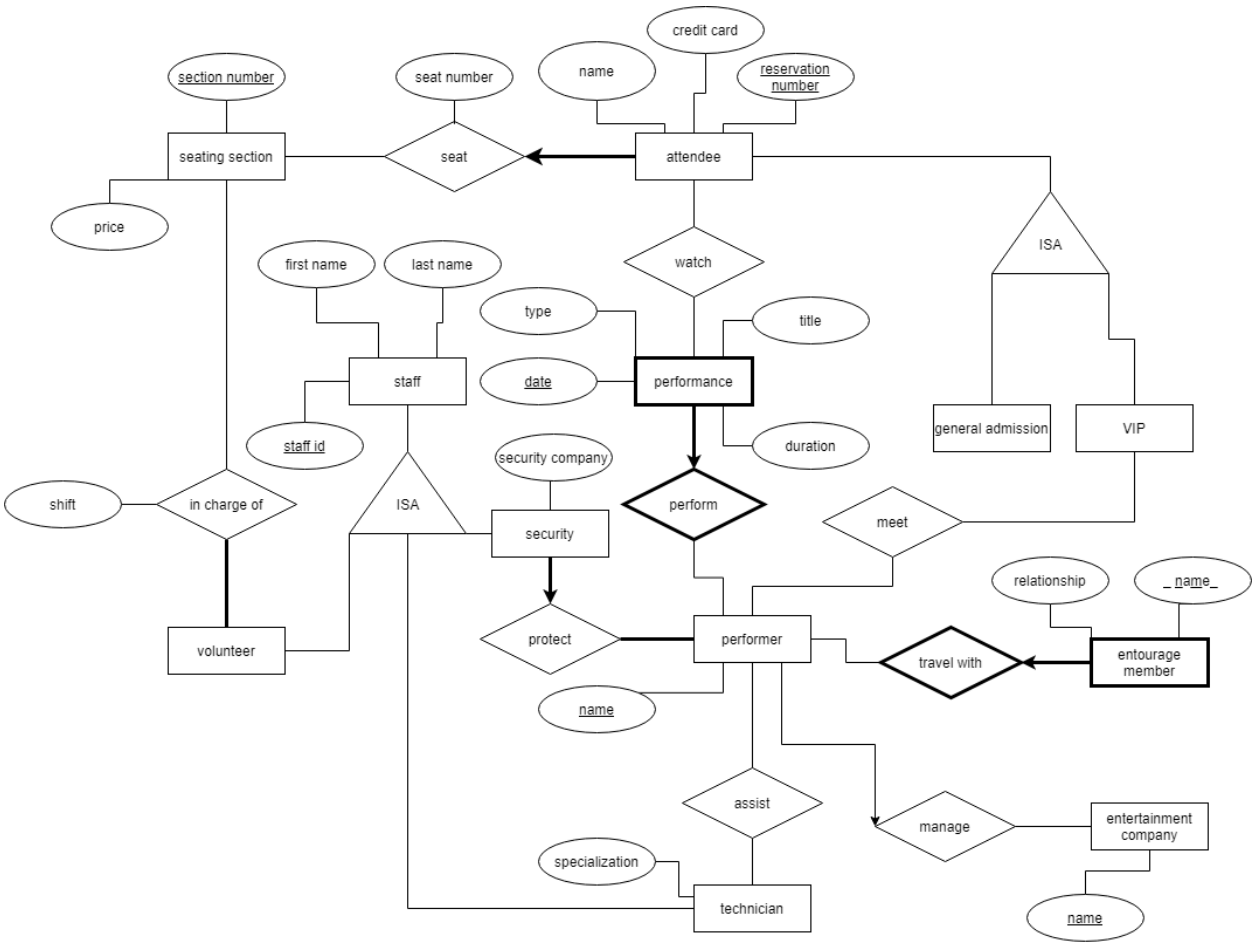
1. A short description of the final project, and what it accomplished.

   For the complete project files, please go to our github page:
   https://github.students.cs.ubc.ca/CPSC304-2021W-T1/project_d1w5a_k3x2b_s8b3b

   Our Concert Venue Simulator allows a business, such as a concert venue, stadium, arena, etc., to manage their employees, schedule performances, and issue tickets for attendees. The system lets the venue keep track of different types of employees, different kinds of attendees, as well as performers and the people who travel with them. The venue can use this system to assign work to different levels of staff, track attendance for performances, and keep a general record of every person in the venue during a given performance.

2. A description of how your final schema differed from the schema you turned in. If the final schema differed, explain why. Note that turning in a final schema that's different from what you planned is fine, we just want to know what changed and why.

   Our final schema has not changed from what was previously submitted. The schema below is the same schema from Milestone 2:



3. A list of all SQL queries used. For SQL query requirements, check the rubric listed on Canvas for Milestone 4.

   For the full runnable SQL script, please see “concert-venue.sql” on our github page. The initialization script includes the following queries:
   - Drop table [table name]
   - Create table [table name...]
   - Insert into [table name] values [...]
   - Grant select on [table name] to public

   Required queries:

   Insert operation:
   - INSERT into Staff values (:bind1, :bind2, :bind3)

   Update operation:
   - UPDATE Staff SET firstname='" . $new_fname . "' WHERE staffID='" . $id . "'
   - UPDATE Staff SET lastname='" . $new_lname . "'WHERE staffID='" . $id. "'

   Delete operation:
   - DELETE from Staff WHERE staffID='" . $id . "'

   Selection:
   - SELECT $attribute FROM $table WHERE $condition

   Projection:
   - SELECT $attributes FROM Attendee_Seat_2

   Join:
   - SELECT DISTINCT resnum, cardnum
     FROM Attendee_Seat_1
     NATURAL JOIN Watch
     WHERE pname = '$pname'

   Division:
   - SELECT *
     FROM Staff s
     WHERE NOT EXISTS (SELECT p1.pname
     FROM Performer p1
     WHERE NOT EXISTS (SELECT p2.pname
     FROM Protect p2
     WHERE p2.staffID = s.staffID
     AND p1.pname = p2.pname))

Aggregation with Group By:

- SELECT pname, count(*)
  FROM Performance_1
  GROUP BY pname

Aggregation with Having:

- SELECT DISTINCT pname, pdate
  FROM Watch w
  GROUP BY w.pname, w.pdate
  HAVING count(*) >= :bind1

Nested Aggregation with Group By:

- SELECT MIN(duration), type
  FROM Performance_2
  WHERE duration > :bind1
  GROUP BY type
  HAVING AVG(duration) > (SELECT AVG(duration)
  FROM Performance_2)

Additional queries:

- SELECT Count(*) FROM $tableName
- SELECT * FROM $tableName
- SELECT table_name from user_tables

4. Screenshots of the sample output of the queries using the GUI (for example, you can show what data is in your table before you run the query, and then show another screenshot after running the query, from some kind of GUI input like a button). You need only to include screenshots for the specified queries – if you implemented more than what was required, screenshots are not needed for those extra queries.

Insert:
Before query:

Retrieved data from table Staff:

1001	Michael	Scott
1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson

Query:

Insert

Add a new staff member to the venue's database.

staffID is a unique int. Names are case sensitive and will be saved as they're entered.

staffID: 1234

First Name: Andrew

Last Name: Bernard

Insert

After query:

Retrieved data from table Staff:

1001	Michael	Scott
1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Andrew	Bernard

Delete:
Before query: Staff table and Volunteer table (Volunteer has an ISA relationship with Staff)

Retrieved data from table Staff:

1001	Michael	Scott
1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Andrew	Bernard

Retrieved data from table Volunteer:

1001
1005
1009
1010
1011

Query:

Delete

Delete a staff member from the venue's database.

staffID: 1001

Delete

After query:

Retrieved data from table Staff:

1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Andrew	Bernard

Retrieved data from table Volunteer:

1005
1009
1010
1011

Update:
Before query:

Retrieved data from table Staff:

1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Andrew	Bernard

Query 1: Update only the first name

Update

Update a staff member's information. staffID cannot be changed.

Names are case sensitive and will be saved as they're entered.

staffID: 1234

New First Name: Andy

New Last Name:

Update

After query 1:

Retrieved data from table Staff:

1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Andy	Bernard

Query 2: Update both first name and last name

Update

Update a staff member's information. staffID cannot be changed.

Names are case sensitive and will be saved as they're entered.

staffID: 1234

New First Name: Robert

New Last Name: California

Update

After query 2:

Retrieved data from table Staff:

1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson
1234	Robert	California

Selection:
Original Staff table:

Retrieved data from table Staff:

1001	Michael	Scott
1002	Kevin	Malone
1003	Jim	Halpert
1004	Pam	Beesly
1005	Merideth	Palmer
1006	Stanley	Hudson
1007	Dwight	Schrute
1008	Angela	Martin
1009	Oscar	Martinez
1010	Phyllis	Vance
1011	Kelly	Kapur
1012	Ryan	Howard
1013	Toby	Flenderson
1014	Daryl	Philbin
1015	Jan	Levinson

Query:

Search - Selection

The values are case sensitive.

Table name: Staff
Attributes (separated by comma ','): firstname, lastname
Conditions (separated by comma ','): staffID > 1004, staffID < 10

Submit

After Query:

Retrieved data from table Staff:

Merideth	Palmer
Stanley	Hudson
Dwight	Schrute
Angela	Martin
Oscar	Martinez
Phyllis	Vance
Kelly	Kapur
Ryan	Howard
Toby	Flenderson

Projection:

Original Attendee_Seat_2 table:

Retrieved data from table Attendee_Seat_2:

Andy Dwyer	1234567890123456	100	2
April Ludgate	1324567890123456	300	3
Ron Swanson	1423567890123456	203	14
Leslie Knope	1523467890123456	101	76
Tom Haverford	1623457890123456	105	57
Jerry Gergich	1723456890123456	209	3
Ann Perkins	1823456790123456	222	35
Ben Wyatt	1923456780123456	102	32
Chris Traeger	1023456789123456	301	14
Donna Meagle	2134567890123456	109	3

Query:

Attendee Seat Table - Projection

- [ ] full name
- [x] card number
- [ ] section number
- [x] seat number

Submit

After query:

Retrieved data from table Attendee_Seat_2:

1234567890123456	2
1324567890123456	3
1423567890123456	14
1523467890123456	76
1623457890123456	57
1723456890123456	3
1823456790123456	35
1923456780123456	32
1023456789123456	14
2134567890123456	3

Join:
Original Attendee_Seat_1 and Watch tables:

Retrieved data from table Watch:

21433	Scrantonicity	30-JUN-22
21435	Duke Silver	14-FEB-22
31524	DJ Disco	31-OCT-21
35142	Ping	24-APR-22
35193	DJ Disco	31-OCT-21
41245	Ping	29-SEP-23
49532	Mouse Rat	15-AUG-24
52341	Scrantonicity	30-JUN-22
79384	Duke Silver	14-FEB-22
92448	Mouse Rat	15-AUG-24

Retrieved data from table Attendee_Seat_1:

35142	1234567890123456
49532	1324567890123456
52341	1423567890123456
21435	1523467890123456
31524	1623457890123456
79384	1723456890123456
92448	1823456790123456
41245	1923456780123456
21433	1023456789123456
35193	2134567890123456

Query:

Search by Performance - Join

Search the reservation number and card number of an attendee who's watching performance(s) by a specific performer.

Performer name: Ping

Submit

After query:

Retrieved data from table Attendee_Seat_1 and Watch:

41245	1923456780123456
35142	1234567890123456

Divide:

Before and after divide query on SqlPlus:

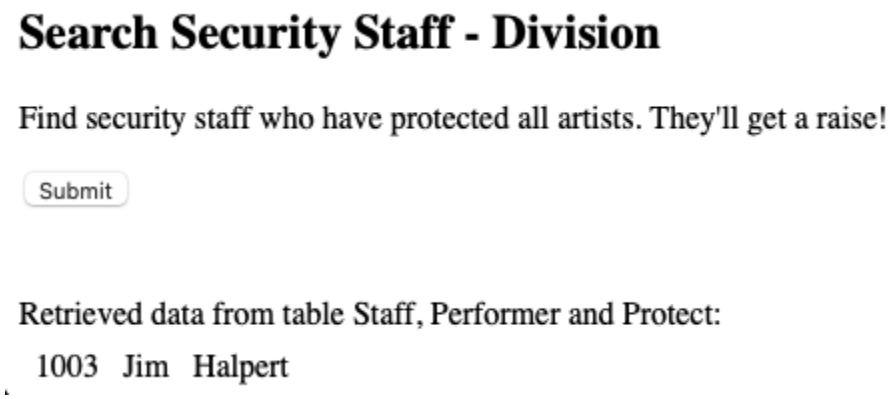
```
SQL> select *
  2  from staff s
  3  where not exists (select p1.pname
  4  from performer p1
  5  where not exists (select p2.pname
  6  from protect p2
  7  where p2.staffID = s.staffID));

   STAFFID FIRSTNAME           LASTNAME
---------- ------------------- --------------------
      1003 Jim                 Halpert
      1007 Dwight              Schrute
      1012 Ryan                Howard
      1014 Daryl               Philbin
      1015 Jan                 Levinson

SQL> select *
  2  from staff s
  3  where not exists (select p1.pname
  4  from performer p1
  5  where not exists (select p2.pname
  6  from protect p2
  7  where p2.staffID = s.staffID
  8  and p1.pname = p2.pname));

   STAFFID FIRSTNAME           LASTNAME
---------- ------------------- --------------------
      1003 Jim                 Halpert
```

After divide in our GUI:



Aggregation with Group By:

Display Performance Count by Performer

Submit Query

Display Performances with Minimum Number of Attendees

Minimum Number of Attendees:

Submit Query

Display Tables List

All tables currently in the database.

Submit Query

Number of performances by performer:

Performer	Number of performances
DJ Disco	2
Duke Silver	2
Mouse Rat	1
Ping	3
Scrantonicity	1

Aggregation with Having:

Display Performances with Specified Minimum Number of Attendees

Minimum Number of Attendees: 1

Submit

Performances:

Performance	Date
DJ Disco	31-OCT-21
Scrantonicity	30-JUN-22
Ping	29-SEP-23
Duke Silver	14-FEB-22
Ping	24-APR-22
Mouse Rat	15-AUG-24

Display Performances with Specified Minimum Number of Attendees

Minimum Number of Attendees: 2

Submit

Performances:

Performance	Date
DJ Disco	31-OCT-21
Scrantonicity	30-JUN-22
Duke Silver	14-FEB-22
Mouse Rat	15-AUG-24

Nested Aggregation with Group By:

Display Minimum Duration Performance by type, over a Length with Greater Than Average Duration

Returns the shortest performance > the specified duration (in minutes) for each performance type, for which the average duration is longer than the average duration of all the performances in the database.

Minimum Duration of Performance over: 60

Submit

Performances:

Duration	type
120	rock

Display Minimum Duration Performance by type, over a Length with Greater Than Average Duration

Returns the shortest performance > the specified duration (in minutes) for each performance type, for which the average duration is longer than the average duration of all the performances in the database.

Minimum Duration of Performance over: 150

Submit

Performances:

Duration	type
180	rock