CPSC 304

2015 Summer Term 1

Project Part __3: Project Documentation _

Group Name: ______Camputer Scientists_____

Group Members:

Name	**Student Number**	**Unix ID**	**Email Address**
Euan Chow	47437116	j2z7	chow_ec@hotmail.com
Yen-Yu (Eric) Lai	25920109	k2r7	yyleric@yahoo.ca
Kaitlyn Melton	19320126	v7x8	kaitlynmelton@alumni.ubc.ca
Dylan Otruba	40441115	m4c8	dotruba@gmail.com

1. Project Accomplishments

The domain modeled for this project is the registration for a summer camp, specifically the registration information associated with summer camp activities for both campers and counsellors.

As per the schema and specifications, the database contains and models information about registration (payment, time, etc), campers (assigned cabin, activities signed up, etc.), counsellors (activities being taught, cabin supervised, etc.), and facilities (address, available cabins, etc.). Each camper can complete multiple registrations (at least one registration is required) for a variety of camps and sessions, can be assigned to cabins, and can sign up for multiple activities (registered campers must sign up for at least one activity). A payment is required for each registration. Campers are identified through their name and phone number.

Camps are associated with a week and can hire counsellor(s), who will supervise cabins and lead activities. Each cabin requires a counsellor as a supervisor, but not all counsellors have to be supervisors.

All the information in the database is stored on the Oracle server and can be accessed via a Java-based User Interface.

2. Changes in Final Schema

Some changes were made to the final schema, as we discovered additional things. M
The changes we made are outlined below, most of which were either adjusting naming of variables or adding constraints. Changes to naming were made mostly for consistency in naming conventions and across schema, while we originally failed to include certain constraints that needed to hold in the database. Certain references to foreign keys needed not null constraints for logical consistency in the database - a camp cannot be created without a facility or a type, certain elements of registration are required for the registration to be valid.

The one structural change we made was changing the attributes of “session”, changing it from having a start date and an end date to having a name (essentially giving the index of the week) and description, that included the start and end date as a string. We realized it made the most sense for the sessions to be hardcoded into the database, and we did not want anyone to have the functionality to add or remove sessions. Because the start date and end date were strictly for information purposes, it was more convenient and logical to store these as a string in a description.

Camp
Camp(name, fid, type, max_capacity)

- Foreign Key:
  - fid references Facility
  - type references TypeFee

Changes
Attribute name change: facility_ID → fid
fid: +NOT NULL
type: +NOT NULL

TypeFee
TypeFee(type, fee)

Changes
None

Facility
Facility(id, name, address, phone_num)

Changes
Attribute name change: phone# → phone_num

CampSession
CampSession(id, name, description)

Changes

Table name change: Session → CampSession
Remove attribute: start_date
Remove attribute: end_date
Added attribute: name
Added attribute: description

Registration

Registration(conf_num, sid, camp_name, camper_id, cabin_id, counsellor_id, is_paid)

- Foreign Key:
    - sid references CampSession
    - camp_name references Camp
    - camper_id references Camper
    - cabin_id references Cabin
    - counsellor_id references Counsellor

Changes

Attribute name change: confirmation# → conf_num
Attribute name change: session_ID → sid
Attribute name change: camper_ID → camper_id
Attribute name change: cabin_ID → cabin_id
Attribute name change: counsellor_ID → counsellor_id
sid: +NOT NULL
camp_name: +NOT NULL
camper_id: +NOT NULL

Counsellor

Counsellor(id, name, camp_name, cabin_id)

- Foreign Key:
    - cabin_id reference Cabin
    - camp_name reference Camp

Changes

Added attribute: camp_name

Activity

Activity(name, supplies, description)

Changes

None

Camper

Camper(camper_id, name, phone_num, address, email)

Changes

Attribute name change: phone# → phone_num

Cabin

Cabin(id, num, fid)

- Foreign Key:
    - fid references Facility

Changes

Attribute name change: cabin_ID → id
Attribute name change: number → num
Attribute name change: facility_ID → fid

CampOffers

CampOffers(camp_name, activity_name)

- Foreign Key:
    - camp_name references Camp
    - activity_name references Activity

Changes

None

3. SQL Queries

Campers will have access to 6.1~6.8
Administrators will have access to 6.9~6.15, 6.20, 6.21
Instructors will have access to 6.16~6.19, 6.20, 6.21

See CamperQueries.java, AdminQueries.java, CounsellorQueries.java for more details

6.1 Complete Registration - CamperQueries.java
Input: *name, address, phone, email*

Inserts new camper's name, address, phone number and email into Campers database

```
INSERT INTO Camper(id, name, phone_num, address, email)
VALUES (camper_counter.nextval, name, phone, address, email

SELECT camper_counter.currval
FROM Camper
```

Output: Generated *camperID*

Input: *camperID, sessionID, campName*

Inserts Generated camperID and the sessionID and camp name to be assigned into Registration database

```
INSERT INTO Registration(conf_num, sid, camp_name, camper_id, is_paid)
VALUES (registration_counter.nextval, sessionID, campName, camperID, 0)

SELECT registration_counter.currval
FROM Registration
```

Output: Generated *confirmationNo*

6.2 Make Payment - CamperQueries.java
Input: *confNo*

For Registration record with inputted confNo, sets is_paid flag to Paid.

```
SELECT is_paid
FROM Registration
WHERE conf_num = confNo

UPDATE Registration
SET is_paid = 1
```

```
WHERE conf_num = confNo
```

6.3 Search Activity by Camp - CamperQueries.java
Input: *campName*

Takes campName, searches in CampOffers table for all activities offered by that camp, and returns that list of activities.

```
SELECT activity_name
FROM CampOffers
WHERE camp_name = campName
```

Output: List of *activity_name*s

6.4 Search Camps by Activity - CamperQueries.java
Input: List of *activities*[n]

Takes list of activities, searches CampOffers table for camps that offer all activities in that list, then returns the found list of camp_names.

```
SELECT c1.name
FROM Camp c1
WHERE NOT EXISTS ((SELECT a.name
                   FROM Activity a
                   WHERE a.name = activities[1] OR a.name = activities[2] OR …
                 OR a.name = activities[n])
                   MINUS
                       (SELECT activity_name
                        FROM CampOffers c2
                       WHERE c1.name = c2.camp_name))
```

Output: List of *camp_name*s

6.5 Search Activities by Supply - **REMOVED**

6.6 Search for Session - **REMOVED**

NEW 6.6 Get all Sessions

Returns all sessions in CampSession table

```
SELECT *
```

```
FROM CampSession
```

Output: List of *session_name*s

6.7 Cancel Registration - CamperQueries.java

Input: *confNo*

Searches Registration for record with that confNo, then deletes that record.

```
DELETE FROM Registration
WHERE conf_num = confNo
```

6.8 Change Session - CamperQueries.java

Input: *confNo, sessionID*

For Registration record with that confNo, changes session to new sessionID

```
UPDATE Registration
SET sid = sessionID
WHERE conf_num = confNo
```

6.9 Assign Cabin Supervisor - AdminQueries.java

Input: *cabinID, counsellor_id*

Checks Counsellor, Cabin and Camp tables; if counsellor with counsellor_id is working at the same camp as the cabin with CabinID, assigns that cabin to that counsellor in Counsellor table

```
SELECT fid
FROM Cabin
WHERE id = cabinID AND fid IN (SELECT ca.fid
                               FROM Camp ca, Counsellor co
                               WHERE co.camp_name = ca.name AND co.id =
                          counsellor_id)
```

```
UPDATE Counsellor
SET cabin_id = cabinID
WHERE id = counsellor_id
```

6.10 Set Works At - AdminQueries.java

Input: *counsellor_id, camp_name*

Checks Counsellor table; if counsellor with counsellor_id is not assigned to any camp, assigns the camp with camp_name to that counsellor

```
SELECT camp_name
FROM Counsellor
WHERE id = counsellor_id AND camp_name IS NULL

UPDATE Counsellor
SET camp_name = camp_name
WHERE id = counsellor_id
```

6.11 Assign Registration to Counsellor - AdminQueries.java

Input: *counsellor_id, confirmNo*

Checks Registration for the record with confirmNo; if that record has no assigned counsellor, assigns that counsellor with counsellor_id to that registration record.

```
SELECT R.conf_num, R.camp_name, R.counsellor_id, C.id
FROM Counsellor C, Registration R
WHERE R.camp_name = C.camp_name AND R.conf_num = confirmNo

UPDATE Registration
SET counsellor_id = counsellor_id
WHERE conf_num = confirmNo
```

6.12 Create Session - **REMOVED**

NEW 6.12 Delete Camper

Input: *camperID*

Looks in Camper table for camper with camperID, then deletes that camper from table if it exists.

```
SELECT id
FROM Camper
WHERE id = camperID

DELETE FROM Camper
WHERE id = camperID
```

6.13 Check Registration Payment - AdminQueries.java

Input: *camp_name*

Looks in Registration table and searches for all campers attending camp_name that have not paid yet, and returns a list of phone numbers for those campers via the Camper table.

```
SELECT C.phone_num
FROM Camper C, Registration R
WHERE C.id = R.camper_id AND R.camp_name = camp_name AND R.is_paid = 0"
```

Output: List of *phone_number*s

6.14 Multiple Camps - AdminQueries.java

NOTE: Implemented, but not used in current application.

Input: List of *camp_name*s[m], List of *camperID*s[n]

Parses Registration table and filters records in table for all inputted camperIDs. Then parses the resulting group to search for records with any of the inputted camp_names. Finally, groups campers by camperID and selects campers that have registered for more than one of the given camps.

```
CREATE VIEW campersFilter AS (SELECT *
                              FROM Registration
                              WHERE camper_id = camperID[1] OR camper_id =
                         camperID[2] OR … OR camper_id = camperID[n]

SELECT camper_ID, COUNT(*)
FROM campersFilter
WHERE camp_name = camp_name[1] OR camp_name[2] OR … OR camp_name[m]
GROUP BY camper_id
HAVING COUNT(*) > 1
```

Output: List of *camperID*s

6.15 Check Counsellor Role - AdminQueries.java

Looks in Counsellor table for all counsellors that are not working at any camp, and returns that list of counsellor_ids.

```
SELECT *
FROM Counsellor
WHERE camp_name IS NULL
```

Output: List of *counsellor_id*s

6.16 Check Campers to Supervise - CounsellorQueries.java

Input: *counsellor_ID*

Looks in Registration table, and collects the camper_IDs from all records that have the inputted *counsellor_ID* as a supervisor, or is assigned a cabin that the inputted counsellor is supervising (looked at via Cabin table). Then, the collected camper_IDs are joined with the camper names via the Campers table, and the list of camper_IDs and camper names is returned.

```
SELECT c.id, c.name
FROM Camper c, Registration r
WHERE c.id = r.camper_id AND
        (r.counsellor_id = counsellor_ID OR r.cabin_id = (SELECT cabin_id
                                                          FROM Counsellor
                                                          WHERE id = counsellor_ID))
```

Output: List of *camperID*s and *names*

6.17 Assign Camper Cabin - CounsellorQueries.java

Input: *confNo*

From Cabins table, get a list of all cabins that are present, along with the number of people in each cabin. Then using the confNo, finds the record in Registration table with that number, gets the camp_name for that record, and finds the associated facility via the Camp table. Then, for all cabins in that facility, looks for the cabin with the least amount of people in it, gets that cabin's cabinID, and assigns that id to the confNo's record in Registration table.

```
CREATE VIEW cabinCount AS (SELECT fid, id AS cabin, count(camper_id) AS
num_campers
                                  FROM Cabin LEFT OUTER JOIN Registration
                                      ON Cabin.id = Registration.cabin_id
                                  GROUP BY id, fid

UPDATE Registration
SET cabin_id = (SELECT fc.cabin
                FROM cabinCount fc, Camp c1, Registration r
                WHERE r.conf_num = ? and r.camp_name = c1.name and c1.fid = fc.fid
        and fc.num_campers <= ALL (SELECT num_campers
                                   FROM cabinCount cc
```

```
                                    WHERE r.camp_name = c1.name and c1.fid =
                                cc.fid)) "
    WHERE conf_num = confNo

    SELECT cabin_id
    FROM Registration
    WHERE conf_num = confNo
```

Output: *cabinID*

6.18 Offer Activity - CounsellorQueries.java

Input: *actName, description, supplies*

Checks Activity table, and if there is no activity with actName, adds said activity and all its attributes into the table.

```
INSERT INTO Activity
VALUES (actName, description, supplies)
```

Input: *camp_name, activityName*

Updates CampOffers table so that the inputted camp now offers the inputted activity.

```
INSERT INTO CampOffers
VALUES (camp_name, activityName)
```

6.19 Check Registered for Session - CounsellorQueries.java

Input: *camp_name, sessionID*

Checks Registration table, and returns all camperIDs attending the inputted camp during the inputted session. Then, those IDs are joined with the campers' names via Camper table, and that list of ids and names is returned.

```
SELECT c.id, c.name
FROM Registration r, Camper c
WHERE r.camper_id = c.id AND r.sid = sessionID AND r.camp_name = camp_name
```

Output: List of *camperID*s and *names*

6.20 Multiple Sessions - AdminQueries.java, CounsellorQueries.java

Looks in Registration table and searches for all campers that have registered for more than one camp, then matches camperIDs with names in Campers table and returns both ids and names.

```
SELECT C.id, C.name, COUNT(R.sid) AS session_count
FROM Registration R, Camper C
WHERE R.camper_id = C.id
GROUP BY C.id, C.name
HAVING COUNT(R.sid) > 1
```

Output: List of <u>*camperID*</u>s and <u>*name*</u>s

<u>6.21 Check Campers by Activity - **REMOVED**</u>

<u>6.22 Return Camp Statistics - AdminQueries.java</u>

Aggregates Registration and CampFees table, and returns the following:

- COUNT of campers for each camp
- AVERAGE COUNT of campers for each camp
- MIN COUNT of campers registered for a camp
- MAX COUNT of campers registered for a camp
- MIN fee for any camp
- MAX fee for any camp

```
SELECT camp_name, Count(camper_id)
FROM Registration
GROUP BY camp_name

SELECT AVG(COUNT(camper_id))
FROM Registration
GROUP BY camp_name

SELECT MIN(COUNT(camper_id))
FROM Registration
GROUP BY camp_name

SELECT MAX(COUNT(camper_id))
FROM Registration
GROUP BY camp_name

SELECT *
FROM Typefee

SELECT MIN(fee)
FROM TypeFee
```

```
SELECT MAX(fee)
FROM TypeFee
```

Output: Statistics summary as above