CSC423 Database Systems

Project: Design, development and implementation of a relational database

Due: Deadline date @ 11:59 PM EST

A relational database should be developed following the subsequent steps for the case study described in this PDF:

1. Develop a conceptual data model reflecting the following requirements: (11/02/21)

a. Identify the main entity types.

- Entities:
  - Department
  - Event
  - Student
  - Major

b. Identify the main relationship types between the entity types identified in "a".
c. Determine the multiplicity constraints for each relationship identified in "b".

Entity 1	Relationship	Entity 2	Participation	Cardinality	Multiplicity	Type of rel.
Department Event	Hosts HostedBy	Event Department	0 1	* *	0..* 1..*	*:*
Student Major	Declares DeclaredBy	Major Student	1 1	* *	1..* 1..*	*:*
Student Event	Attends AttendedBy	Event Student	1 1	* *	1..* 1..*	*:*
Department Major	Offers OfferedBy	Major Department	1 1	* 1	1..* 1..1	1:*
Faculty Department	WorksAt Has	Department Faculty	1 1	1 *	1..1 1..*	1:*

d. Identify attributes and associate them with entity or relationship types.

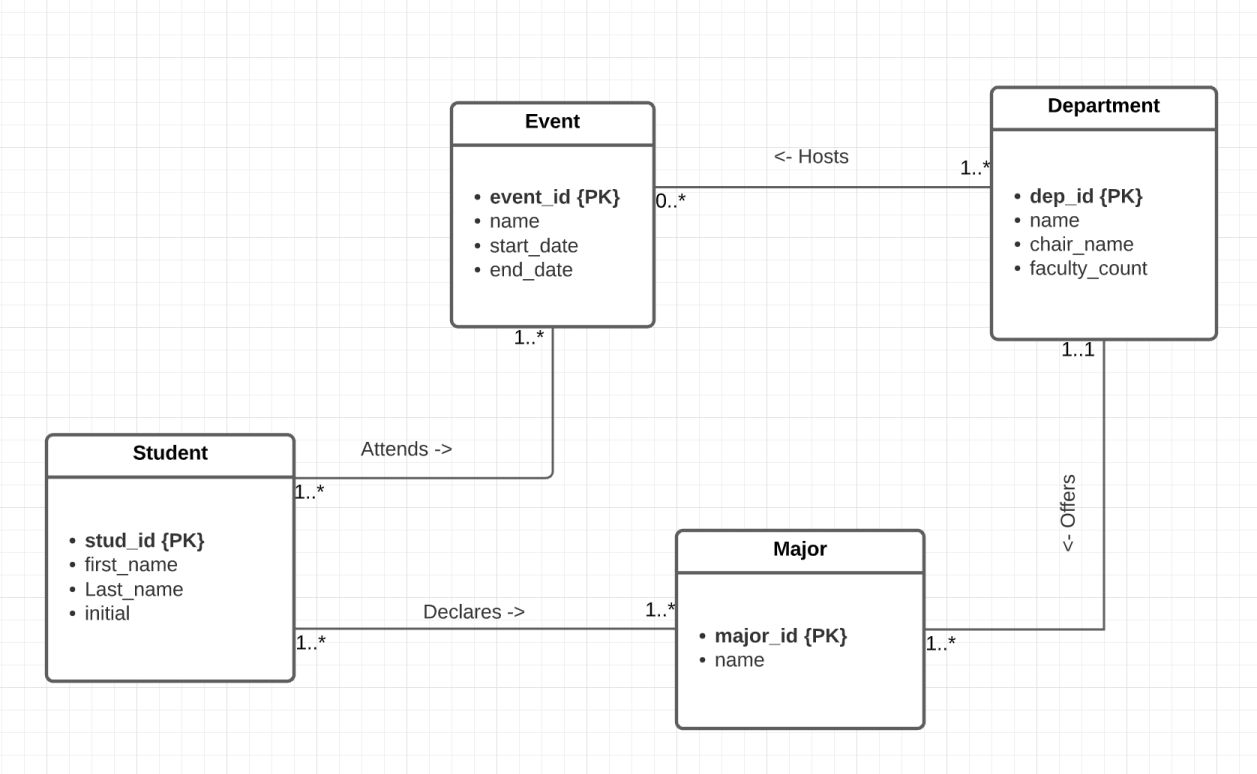
- Attributes for each entity:
  - **Department**
    - **Dep_id {PK}**
    - name
    - chair_name
    - faculty_count
  - **Event**
    - **Event_id {PK}**
    - name
    - Start_date
    - End_date
  - **Student**
    - **Stud_id {PK}**
    - first_name
    - Last_name
    - initial
  - **Major**
    - **Major_id {PK}**

- name

e. Determine candidate and primary key attributes for each (strong) entity type.
Department -> **dep_id {PK}** , **name**, chair_name, faculty_count
Event -> **event_id {PK}**, **name**, start_date, end_date
Student -> **stud_id {PK}**, first_name, last_name, initial
Major -> **major_id {PK}**, **name**

Alternate keys are indicated in BOLD.

f. Generate the E-R diagram for the conceptual level (no FKs as attributes).



2. Develop a logical data model based on the following requirements:
(11/19/21)

a. Derive relations from the conceptual model.

Department(dep_id, name, chair_name, faculty_count)
Primary Key dep_id
Alternate Key name

Student(stud_id, first_name, last_name, initial)
Primary Key stud_id

Event(event_id, name, start_date, end_date)
Primary Key event_id
Alternate Key name

Major(major_id, name, dep_id)
Primary Key major_id
Alternate Key name
Foreign Key dep_id **references** Department(dep_id)

MajorRecord(major_id, stud_id)
Primary Key major_id, stud_id
Foreign Key major_id **references** Major(major_id)
Foreign Key stud_id **references** Student(stud_id)

EventHost(event_id, dep_id)
Primary Key event_id, dep_id
Foreign Key event_id **references** Event(event_id)
Foreign Key dep_id **references** Department(dep_id)

EventAttendance(event_id, stud_id)
Primary Key event_id, stud_id
Foreign Key event_id **references** Event(event_id)
Foreign Key stud_id **references** Student(stud_id)

b. Validate the logical model using normalization to 3NF.

Functional dependencies:
Dep_id {PK} → name, chair_name, faculty_count
Stud_id {PK} → first_name, last_name, initial
Event_id {PK} → name, start_date, end_date
major_id {PK} → name, dep_id
These are all of the functional dependencies and none are partial or transitive. Therefore the relations are in 1NF, 2NF, and 3NF.

c. Validate the logical model against user transactions.

5 Examples of user transactions:

- List the details of events attended by a named student
    - EventAttendance contains records of student attendees for each event and the details of the student can be found through the Student entity. The following path can be used: **Student -> EventAttendance -> Event**

- List the majors offered by each department
    - The Offers relationship between the Department and Major entities can be used to generate this list.

- List the amount of students in each department
    - The path that can be used to generate this list is from **MajorRecord -> Major**. Because the Major entity has details of what department it belongs to.

- List the events hosted by a named department
    - The following path can be used, **Department -> EventHost -> Event**

- List the host for a named event, or hosts if there are multiple
    - The following path can be used, **Event -> EventHost -> Department**

d. Define integrity constraints:

- Primary key constraints.
    - All primary keys are assumed to be NOT NULL
- Referential integrity/Foreign key constraints.
    - Check the ON UPDATE / ON DELETE actions defined under each relation below
    - Generally, a foreign key value must exist in the parent relation
- Alternate key constraints (if any).
    - Check Alternate key constraints defined below
- General constraints (if any).
    - These are in the Check clauses

Department(dep_id, name, chair_name, faculty_count)
Primary Key dep_id
Alternate Key name
CHECK name **LIKE** 'Department%'
CHECK faculty_count > 0
CHECK chair_name **IS NOT NULL**

Student(stud_id, first_name, last_name, initial)
Primary Key stud_id
CHECK first_name **IS NOT NULL**
CHECK last_name **IS NOT NULL**
CHECK initial **IS NOT NULL**
CHECK initial **LIKE** '__%' (two blank spaces means it must be longer than one character)

Event(event_id, name, start_date, end_date)
Primary Key event_id
Alternate Key name
CHECK start_date **IS NOT NULL**
CHECK end_date **IS NOT NULL**
CHECK start_date **<** end_date
CHECK {current_date} **<** start_date (assume current_date represent today's date and is updated at the time of insertions)
CHECK {current_date} **<** end_date

Major(major_id, name, dep_id)
Primary Key major_id (this is the major code)
Alternate Key name

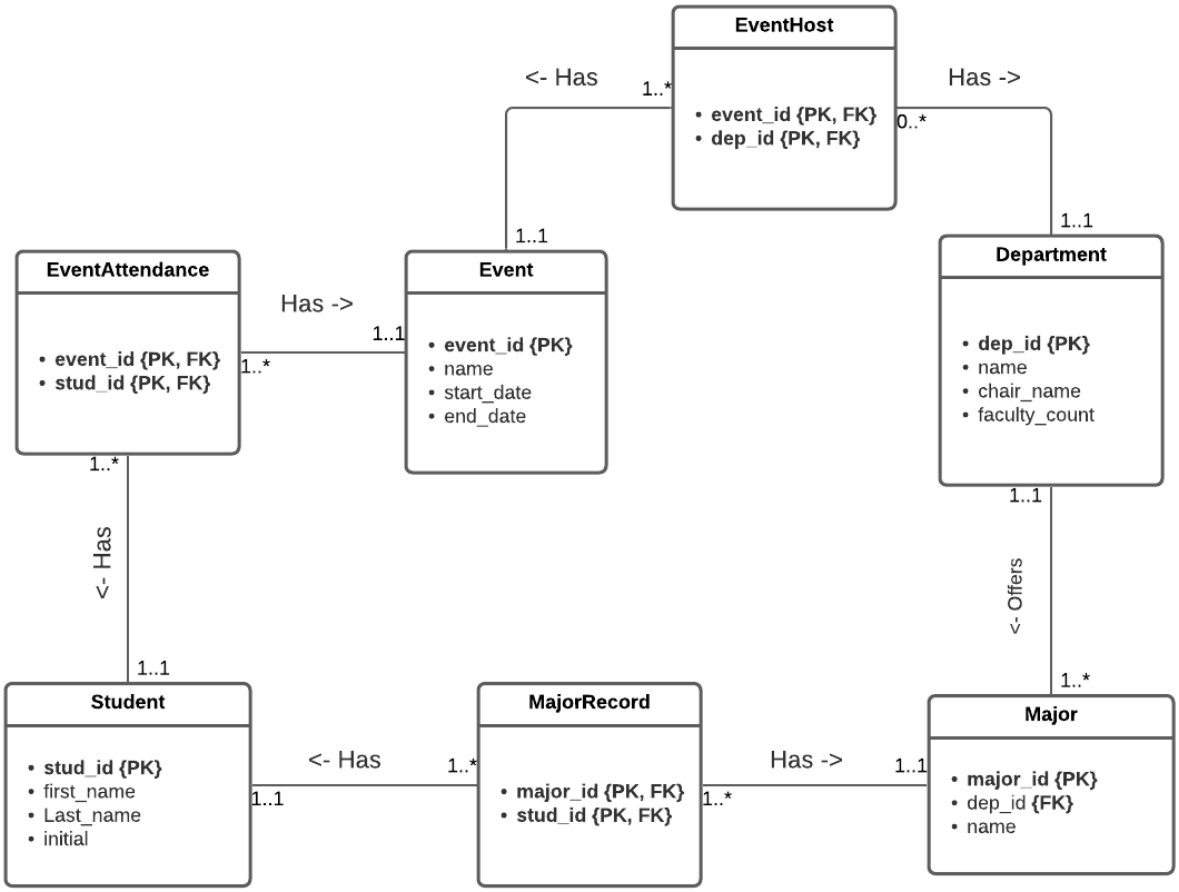
Foreign Key dep_id **references** Department(dep_id)
ON DELETE SET NULL
ON UPDATE CASCADE
CHECK major_id **LIKE** '___' (three blank spaces means it must be 3 characters)

MajorRecord(major_id, stud_id)
Primary Key major_id, stud_id
Foreign Key major_id **references** Major(major_id)
ON DELETE CASCADE
ON UPDATE CASCADE
Foreign Key stud_id **references** Student(stud_id)
ON DELETE CASCADE
ON UPDATE CASCADE

EventHost(event_id, dep_id)
Primary Key event_id, dep_id
Foreign Key event_id **references** Event(event_id)
ON DELETE CASCADE
ON UPDATE CASCADE
Foreign Key dep_id **references** Department(dep_id)
ON DELETE CASCADE
ON UPDATE CASCADE

EventAttendance(event_id, stud_id)
Primary Key event_id, stud_id
Foreign Key event_id **references** Event(event_id)
ON DELETE CASCADE
ON UPDATE CASCADE
Foreign Key stud_id **references** Student(stud_id)
ON DELETE CASCADE
ON UPDATE CASCADE

e. Generate the E-R diagram for the logical level (contains FKs as attributes).



3. Translate the logical data model for the Oracle Enterprise DBMS. (12/09/21)

a. Develop SQL code to create the entire database schema, reflecting the constraints identified in previous steps.

```
CREATE TABLE Department(
    dep_id INTEGER PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    chair_name VARCHAR(255),
    faculty_count INT,
    CHECK (faculty_count > 0),
    CHECK (name LIKE 'Department %')
);
```

```
CREATE TABLE Student(
    stud_id INTEGER PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    initial VARCHAR(255) NOT NULL,
    CHECK (initial LIKE '__%')
);
CREATE TABLE Event(
    event_id INTEGER PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    CHECK (start_date < end_date),
    CHECK ('2021-12-05' < start_date),
    CHECK ('2021-12-05' < end_date)
);

CREATE TABLE Major(
    major_id VARCHAR(8) PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    dep_id INTEGER,
    FOREIGN KEY(dep_id) REFERENCES Department(dep_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    CHECK (major_id LIKE '___')
);



CREATE TABLE MajorRecord(
```

```
    major_id VARCHAR(8),
    stud_id INTEGER,
    PRIMARY KEY (major_id, stud_id),
    FOREIGN KEY(major_id) REFERENCES Major(major_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(stud_id) REFERENCES Student(stud_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE EventHost(
    event_id INTEGER,
    dep_id INTEGER,
    PRIMARY KEY(event_id, dep_id),
    FOREIGN KEY(event_id) REFERENCES Event(event_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY(dep_id) REFERENCES Department(dep_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE EventAttendance(
    event_id INTEGER,
    stud_id INTEGER,
    PRIMARY KEY (event_id, stud_id),
    FOREIGN KEY(event_id) REFERENCES Event(event_id),
    FOREIGN KEY(stud_id) REFERENCES Student(stud_id)
);
```

b. Create at least 5 tuples for each relation in your database.

```
INSERT INTO Department(name, chair_name, faculty_count)
        VALUES
            ('Department of Biology', 'John Doe', 100),
            ('Department of Computer Science', 'Dark Bark', 205),
            ('Department of Engineering', 'Eng Ineer', 50),
            ('Department of Mathematics', 'Math Metician', 14),
            ('Department of Chemistry', 'Walter White', 235);

INSERT INTO Major(major_id, name, dep_id)
        VALUES
            ('BIO', 'Biology', 1),
            ('CSC', 'Computer Science', 2),
            ('ECE', 'Electrical Engineering', 3),
            ('MTH', 'Mathematics', 4),
            ('CHM', 'Chemistry', 5);

INSERT INTO Student(first_name, last_name, initial)
        VALUES
            ('Jin', 'Curia', 'JC'),
            ('Red', 'Blue', 'RB'),
            ('Green', 'Black', 'GB'),
            ('John', 'Johnson', 'JJ'),
            ('Rat', 'Ratson', 'RR');

INSERT INTO MajorRecord(major_id, stud_id)
        VALUES
            ('BIO', 1),
            ('CSC', 2),
            ('CSC', 3),
            ('ECE', 4),
            ('MTH', 5);
```

```
INSERT INTO Event(name, start_date, end_date)
      VALUES
            ('Christmas Party', '2021-12-25', '2021-12-26'),
            ('Halloween Party', '2022-11-29', '2022-11-30'),
            ('February Jam', '2022-02-04', '2022-02-05'),
            ('June Jam', '2022-06-01', '2022-06-02'),
            ('August Party', '2022-08-01', '2022-08-02');

INSERT INTO EventHost(event_id, dep_id)
        VALUES
            (1,1),
            (2,1),
            (3,3),
            (4,4),
            (5,5);

INSERT INTO EventAttendance(event_id, stud_id)
        VALUES
            (1,1),
            (2,2),
            (3,3),
            (4,4),
            (5,5);
```

Resultant tables:

Department:

	dep_id	name	chair_name	faculty_count
0	1	Department of Biology	John Doe	100
1	2	Department of Computer Science	Dark Bark	205
2	3	Department of Engineering	Eng Ineer	50
3	4	Department of Mathematics	Math Metician	14
4	5	Department of Chemistry	Walter White	235

Student:

	stud_id	first_name	last_name	initial
0	1	Jin	Curia	JC
1	2	Red	Blue	RB
2	3	Green	Black	GB
3	4	John	Johnson	JJ
4	5	Rat	Ratson	RR

Major:

	major_id	name	dep_id
0	BIO	Biology	1
1	CSC	Computer Science	2
2	ECE	Electrical Engineering	3
3	MTH	Mathematics	4
4	CHM	Chemistry	5

Event:

	event_id	name	start_date	end_date
0	1	Christmas Party	2021-12-25	2021-12-26
1	2	Halloween Party	2022-11-29	2022-11-30
2	3	February Jam	2022-02-04	2022-02-05
3	4	June Jam	2022-06-01	2022-06-02
4	5	August Party	2022-08-01	2022-08-02

MajorRecord, EventHost, EventAttendance:

	major_id	stud_id
0	BIO	1
1	CSC	2
2	CSC	3
3	ECE	4
4	MTH	5

	event_id	dep_id
0	1	1
1	2	1
2	3	3
3	4	4
4	5	5

	event_id	stud_id
0	1	1
1	2	2
2	3	3
3	4	4
4	5	5

c. Develop 5 SQL queries using embedded SQL (see Python tutorial).

```
    1. List records of EventAttendance but include
student names and event names.

To achieve this, use the EventAttendance relationship
To join that table with Student and Major relations.

    2. List records of declared majors, showing the
student name and major name

Join MajorRecord with Student and Major relations.

    3. Event names and the name of their hosts.
Join EventRecord with Event and Department relations.

    4. List the names of Majors and the department names
they belong to.

Join Major and Department relations.

    5. List students and departments they belong to.

Join Student and Major with MajorRecord, then
Join Major and Department relations.
```

```
    queries = [
        """
            SELECT Ev.name AS event_name,
Stud.first_name, Stud.last_name
            FROM Event Ev, Student Stud, EventAttendance
```

```
Att
            WHERE Ev.event_id = Att.event_id
            AND Stud.stud_id = Att.stud_id;
        """,
        """
            SELECT Stud.first_name, Stud.last_name,
Maj.name AS major_name
            FROM Student Stud, Major Maj, MajorRecord
Rec
            WHERE Stud.stud_id = Rec.stud_id
                AND Maj.major_id = Rec.major_id;
        """,
        """
            SELECT Ev.name AS event_name, Host.name AS
host_name
            FROM Event Ev, Department Host, EventHost
Rec
            WHERE Ev.event_id = Rec.event_id
            AND Host.dep_id = Rec.dep_id;
        """,
        """
            SELECT Maj.major_id AS major_code, Maj.name
AS major_name, Dep.name AS dep_name
            FROM Major Maj, Department Dep
            WHERE Maj.dep_id = Dep.dep_id;
        """,
        """
            SELECT Stud.first_name, Stud.last_name,
Dep.name AS dep_name
            FROM Student Stud, Department Dep, Major
Maj, MajorRecord Rec
            WHERE Stud.stud_id = Rec.stud_id
```

```
            AND Maj.major_id = Rec.major_id
            AND Maj.dep_id = Dep.dep_id;
    """]
```

Resultant Tables:

	event_name	first_name	last_name
0	Christmas Party	Jin	Curia
1	Halloween Party	Red	Blue
2	February Jam	Green	Black
3	June Jam	John	Johnson
4	August Party	Rat	Ratson

	first_name	last_name	major_name
0	Jin	Curia	Biology
1	Red	Blue	Computer Science
2	Green	Black	Computer Science
3	John	Johnson	Electrical Engineering
4	Rat	Ratson	Mathematics

	event_name	host_name
0	Christmas Party	Department of Biology
1	Halloween Party	Department of Biology
2	February Jam	Department of Engineering
3	June Jam	Department of Mathematics
4	August Party	Department of Chemistry

	major_code	major_name	dep_name
0	BIO	Biology	Department of Biology
1	CSC	Computer Science	Department of Computer Science
2	ECE	Electrical Engineering	Department of Engineering
3	MTH	Mathematics	Department of Mathematics
4	CHM	Chemistry	Department of Chemistry

	first_name	last_name	dep_name
0	Jin	Curia	Department of Biology
1	Red	Blue	Department of Computer Science
2	Green	Black	Department of Computer Science
3	John	Johnson	Department of Engineering
4	Rat	Ratson	Department of Mathematics

d. Upload all the code and documentation to GitHub.
https://github.com/c0mpassion/csc423-DB-project

Reports: A report will be created for *each deadline* including detailed documentation of each of the steps, the results obtained at each stage, test data, sample output and conclusion. For example, the ER diagrams for the conceptual and logical models must be included in
each of the reports. All assumptions made in the design must be clearly stated. Screenshots of the contents of the database created in step 3 must be included in the report. GitHub: The code generated during step 3 (SQL statements + program) must be uploaded to a GitHub repository. The link to the repository must be provided in the last report. The GitHub repository must also include all the documentation (i.e., reports) generated in the three steps.

Case Study: Redwood University

A university has requested the design and implementation of a database to store its data. The university encompasses multiple departments, each of which has a chair. ~~The university does not want to store particular information regarding the chair, rather information pertaining to the department name and chair name~~, as well as the number of faculty members the department has. ~~Department names must always start with~~

~~*Department*~~.

The university has numerous students and each of them has declared at least one major. Additionally, the name and initials of a student are stored. ~~Initials must be more than one character long.~~ ~~For each major, the university wants to store the major name, the department it is associated with, and a code~~. For example, 'Biology' is associated with department 3 (i.e., the Department of Biology) and has the code 'BIO'. Major codes must be three characters. Majors can be declared by one or more students. A major references one department, however a department offers one or more majors.

Each department has the possibility of hosting events, and an event can be (collaboratively) hosted by one or more departments. In addition to the event name, the university would like to store the start and end dates of the event. ~~As it is logical, an event cannot end before the start date. Information pertaining to events are stored ahead of time, therefore at the time of insertion an event cannot be a past date or the current date.~~ Students must attend one or more events, and each event will comprise one or more students.