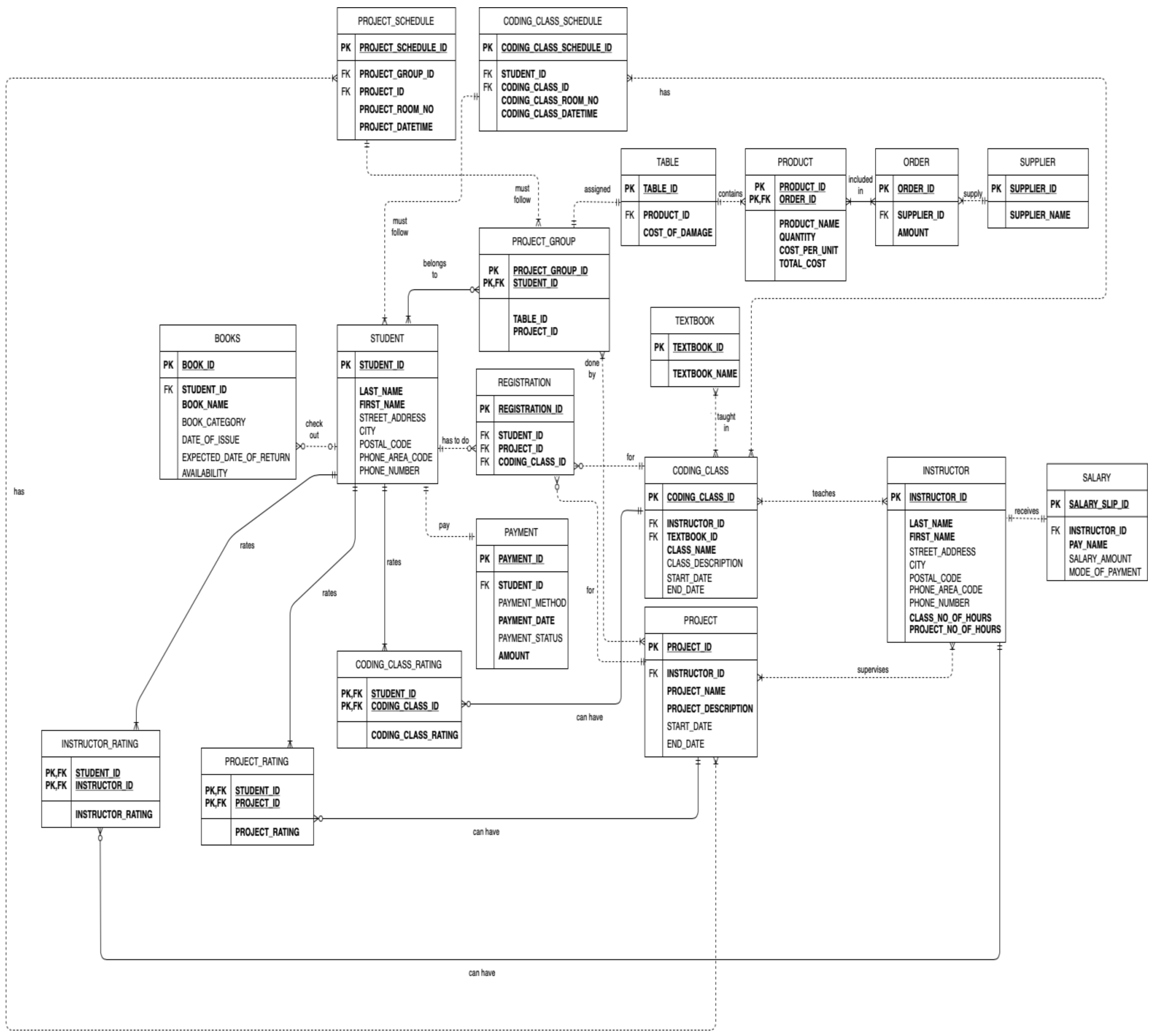
CSCI 585 – DATABASE SYSTEMS
Homework 1 (ER Modelling)

Name: Rohan Mahendra Chaudhari
Email: rmchaudh@usc.edu
USC ID: 6675-5653-85

ER DIAGRAM



ENTITIES

1. **STUDENT** entity contains attributes that are required from a student during the registration process. Here the Primary Key is the STUDENT_ID and the FIRST_NAME and LAST_NAME attributes are highlighted indicating that they are required attributes. Thus, the attributes, STUDENT_ID, FIRST_NAME and LAST_NAME have a constraint on them and cannot contain a NULL value. The rest of the attributes are optional. Further the STUDENT_ID is used as a Foreign Key in all the Relations connected to STUDENT.

2. **BOOKS** entity contains attributes that denote details about the books that a student can check out from the library. Here BOOK_ID is the Primary Key and the STUDENT_ID and BOOK_NAME is highlighted indicating that they are required attributes. Thus, the attributes, BOOK_ID, STUDENT_ID and BOOK_NAME have a constraint on them and cannot contain a NULL value. The rest of the attributes are optional. Here STUDENT_ID is used as a foreign key to maintain referential integrity with the STUDENT entity.

3. **REGISTRATION** entity contains attributes that are required to register a particular student for a class and/or project respectively. This maintains a record of all the courses and projects that a particular student register for. Here the Primary Key is the REGISTRATION_ID and the STUDENT_ID, PROJECT_ID and CODING_CLASS_ID are highlighted indicating that they are required attributes as they are foreign keys in the REGISTRATION entity and to maintain referential integrity these specific values cannot be NULL.

4. **PAYMENT** entity contains attributes required to store the payment information of a particular student towards their respective registered classes and projects. Here the Primary Key is the PAYMENT_ID. Attributes including PAYMENT_ID, STUDENT_ID, PAYMENT_DATE and AMOUNT are highlighted indicating that they are required attributes. Here STUDENT_ID is the foreign key as every payment transaction must be linked to a particular student which is a flat rate and thus the PAYMENT entity is linked to the STUDENT entity.

5. **CODING_CLASS** entity contains attributes that gives information about the classes that the students can register for. Here the Primary Key is CODING_CLASS_ID. Attributes such as CODING_CLASS_ID, TEXTBOOK_ID, INSTRUCTOR_ID and CLASS_NAME is highlighted to indicate that they are required fields and cannot contain NULL values. Here START_DATE and END_DATE are multivalued attributes as a particular course can be reopened again. Here, the assumption is that class and course mean the same thing.

6. **PROJECT** entity contains attributes that gives information about the multiple projects that a student can select from. Here the Primary Key is PROJECT_ID. Attributes such as PROJECT_ID, PROJECT_NAME, INSTRUCTOR_ID and PROJECT_DESCRIPTION is highlighted which means they are required fields and cannot contain NULL value. Here START_DATE and END_DATE are multivalued attributes as registration for a particular project can be reopened in multiple weeks.

7. **INSTRUCTOR** entity is used to store information about a particular instructor. The primary Key is INSTRUCTOR_ID and has multiple required attributes such as INSTRUCTOR_ID, FIRST_NAME, LAST_NAME, CLASS_NO_OF_HOURS and PROJECT_NO_OF_HOURS.

8. **SALARY** entity is used to store the salary an instructor receives which depending on the number of hours the instructor teaches classes and supervises project which vary. The Primary Key is SALARY_SLIP_ID with INSTRUCTOR_ID as a foreign key to reference every instructor and note down the salary of a particular instructor.

9. **TEXTBOOK** entity is used to store information about which textbooks are required for the coding class. A particular textbook can be used in different classes as well and thus to make sure the final database is easy to read, we have a separate entity for this with TEXTBOOK_ID as the Primary Key which is referenced in the CODING_CLASS entity.

10. **INTRUCTOR_RATING** entity is used to store the ratings given by the student for the instructors. The Primary Key is the composition of STUDENT_ID and INSTRUCTOR_ID. Here INSTRUCTOR_RATING is a bridge/composite entity used to represent the M:N relationship between INSTRUCTOR and STUDENT. INSTRUCTOR_RATING in this case is also a weak entity.
Here, STUDENT - INSTRUCTOR_RATING and INSTRUCTOR_RATING – INSTURCTOR are both strong relationships.

11. **PROJECT_RATING** entity is used to store the ratings given by the student for the project topic they worked on. The Primary Key is the composition of STUDENT_ID and PROJECT_ID. Here PROJECT_RATING is a bridge/composite entity used to represent the M:N relationship between PROJECT and STUDENT.
PROJECT_RATING in this case is also a weak entity. Here, STUDENT – PROJECT_RATING and PROJECR_RATING – PROJECT are both strong relationships.

12. **CODING_CLASS_RATING** entity is used to store the ratings given by the students for the classes they enroll in. The Primary Key is the composition of STUDENT_ID and CODING_CLASS_ID. Here CODING_CLASS_RATING is a bridge/composite entity used to represent the M:N relationship between CODING_CLASS and STUDENT.
CODING_CLASS_RATING in this case is also a weak entity.
Here, STUDENT – CODING_CLASS_RATING and CODING_CLASS_RATING – CODING_CLASS are both strong relationships.

13. **PROJECT_GROUP** entity is used to store the students that belong to a particular group. A max of 4 students can belong to a group as per the requirements. The Primary Key is PROJECT_GROUP_ID along with a composite primary key STUDENT_ID which is used for better understanding of the schema and to maintain entity integrity.
Here, STUDENT – PROJECT_GROUP is a strong relationship.

14. **TABLE** entity is just used to store data about which project group is allocated which table number and what parts are assigned to that particular table as per the requirement mentioned in the write up. This also stores the COST_OF_DAMAGE which denotes the cost a particular table has to incur in case of damage of products.

15. **PRODUCT** entity is used to store data about the products that are ordered from the supplier and what the cost of each product is for future reference. It also records the quantity of a particular product ordered and remaining in the inventory. Here the Primary key is PRODUCT_ID with a composite Primary key ORDER_ID. This referential integrity with the ORDER entity is used to denote which supplier produced which particular part in which order number.
    The attribute TOTAL_COST is a derived attributed that can be derived from 2 attributes in the PRODUCT entity – QUANTITY and COST_PER_UNIT.
    Here, PRODUCT – ORDER is a strong relationship.

16. **ORDER** entity is used to store the details of multiple orders placed by the organization to their multiple suppliers for product. Since products can be procured from multiple suppliers, we make use of this entity to link PRODUCT and SUPPLIER. Every row in this entity corresponds 1 order placed to a supplier. 1 order can either contain multiple quantities of the same product or contain multiple different products as well. This table is required since the organization can place multiple orders with a single supplier.

17. **SUPPLIER** entity is used to store the details of the suppliers who supply products for the various projects. Here the Primary Key is SUPPLIER_ID.

18. **PROJECT_SCHEDULE** entity is used to store the schedule such as PROJECT_ROOM_NO, PROJECT_DATETIME for a particular project and the students assigned to that project room and time. The primary Key is PROJECT_SCHEDULE_ID with PROJECT_GROUP_ID denoting project group that must follow that schedule and PROJECT_ID denoting the projects that are going to take place in a particular room and time as the foreign keys.

19. **CODING_CLASS_SCHEDULE** entity is used to store the schedule such as CODING_CLASS_ROOM_NO, CODING_CLASS_DATETIME for a particular class and the students assigned to that classroom and time. The Primary Key is CODING_CLASS_SCHEUDLE_ID with STUDENT_ID denoting students that must follow that schedule and CODING_CLASS_ID denoting the classes that are going to be taught in a particular room and time as the foreign keys.

DESIGN CHOICES

1. Here we assume that a student can register for a mix of multiple projects and classes and thus a particular REGISTRATION_ID corresponds to a student registering for 1 project or 1 class. This way a student can register for multiple classes and projects with a unique REGISTRATION_ID for each. Every student would have a fixed rate to pay irrespective of the number and type of courses and projects they register for as each course will last 1 week and the entire curriculum would last 6 weeks as given.
2. An entity REGISTRATION is considered to record what courses and/or projects a student has registered for. The assumption made is in regards with the write up that states that a "student can register for a mix of coding classes and projects" and thus a student can have multiple registrations. A particular registration can be made by a single student only and thus has a unique REGISTRATION_ID associated with every STUDENT_ID.
3. An instructor can teach multiple courses (ex: Python, Java) and supervise multiple projects. On the other hand, a course (ex: Python) can be taught by multiple instructors but must belong to different schedules i.e., different room or different datetime and the same would work for projects also.
4. A project group can contain a max of 4 students as per the write up and thus I have assumed a case where a student can work on multiple projects and thus belong to multiple project groups as per his project requirements if required and also that a particular project can be done by multiple groups.
5. As per the write up every project group is assigned a table with some products which has been taken into consideration by the relationship between PROJECT_GROUP - TABLE and TABLE - PRODUCT. Here the PROJECT_GROUP - TABLE relationship is 1:1 since a group can be assigned only 1 table and the TABLE - PRODUCT relationship is 1:M since a table can contain multiple products.
6. The organization has to place an order to its multiple suppliers for the products and thus to maintain a record of all the orders placed we use an ORDER entity. In this entity every record specifies what is the order placed with a particular supplier. An order can contain multiple products.
7. We also have 3 different ratings tables INSTRUCTOR_RATING, PROJECT_RATING and CODING_COURSE_RATING instructor, project and coding-course respectively where we record the rating a particular student gives for each of the above-mentioned categories.
8. We also have 2 schedule entities PROJECT_SCHEDULE and CODING_CLASS_SCHEDULE which maintain a schedule of the project and coding classes respectively. Here we have assumed that a class or project can belong to multiple schedules i.e., a class or project can be carried out at multiple different time slots and in different rooms and also that a particular schedule can contains multiple students and/or project groups.

RELATIONSHIPS

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY	COMMENTS
STUDENT	has to do	1:M	REGISTRATION	A particular registration corresponds to a student registering for a course or project. Thus, a student can have multiple registrations, but a particular unique registration id can only belong to a particular student.
STUDENT	rates	1:M	INSTRUCTOR_RATING	A student can have multiple instructors teaching/supervising and thus the student must give a rating for each.
STUDENT	rates	1:M	PROJECT_RATING	A student could have registered for multiple projects and thus must rate each project.
STUDENT	rates	1:M	CODING_CLASS_RATING	A student could have registered for multiple classes and thus must rate each class.
STUDENT	check out	1:M	BOOKS	A student may or may not check out a book from the library, but a book can be checked out only once at a time. A student can check out a max of 4 books at a time. Thus 'M' must be a max of 4.
STUDENT	must follow	1:M	CODING_CLASS_SCHEDULE	A student can belong to multiple schedules, but a schedule can have a student only once.
STUDENT	belongs to	M:N	PROJECT_GROUP	A student can belong to multiple project groups and a project group contains multiple students.
STUDENT	pay	1:1	PAYMENT	Every student is associated with 1 payment which is a flat rate as mentioned in the write up.
REGISTRATION	for	1:M	CODING_CLASS	1 coding class can have multiple registrations, but 1 registration id can be registered in 1 course only once.
REGISTRATION	for	1:M	PROJECT	1 project can have multiple registrations, but 1 registration id can be registered to 1 project only once.
CODING_CLASS	can have	1:M	CODING_CLASS_RATING	1 class can have multiple ratings given by many students.
PROJECT	can have	1:M	PROJECT_RATING	1 project can have multiple ratings given by many students.
PROJECT	done by	M:N	PROJECT_GROUP	1 project (ex: Raspberry Pi game building) can be carried out by multiple groups that have registered for it and 1 project group can do multiple projects.
TEXTBOOK	taught in	M:N	CODING_CLASS	1 particular textbook can be taught in multiple classes (ex: Python1, Python2) and 1 course taught by different instructors can have different textbooks (ex: Intro

				to Python, Basics of Python both for the same class Python1).
INSTRUCTOR	receives	1:1	SALARY	1 instructor can have only 1 salary.
INSTRUCTOR	can have	1:M	INSTRUCTOR_RATING	1 instructor can have multiple ratings given by multiple students.
PROJECT_GROUP	assigned	1:1	TABLE	A particular project group can be assigned only to 1 table at a time.
PROJECT_GROUP	must follow	1:M	PROJECT_SCHEDULE	A project group can belong to multiple schedules, but a schedule can have a project group only once.
TABLE	contains	1:M	PRODUCT	1 table can contain multiple products.
PRODUCT	included in	M:N	ORDER	1 order can contain multiple products and 1 product can belong to multiple orders.
SUPPLIER	supply	1:M	ORDER	1 supplier can be requested with multiple orders.
INSTRUCTOR	teaches	M:N	CODING_CLASS	An instructor can teach multiple different classes (ex: Python1, Java) and a particular class (ex: Python) can be taught by multiple professors.
INSTRUCTOR	supervises	M:N	PROJECT	An instructor can supervise multiple different projects and a particular project (ex: Raspberry Pi game building) can be supervised by multiple instructors.
CODING_CLASS	has	M:N	CODING_CLASS_SCHEDULE	A particular class (ex: Python) can be taught at different times and 1 particular schedule can contain multiple classes of Python taught by different professors as well or can also contain multiple distinct classes.
PROJECT	has	M:N	PROJECT_SCHEDULE	A particular project (ex: Raspberry Pi game building) can be conducted at different times for different set of groups. On the other hand, 1 particular schedule can contain multiple projects of the same topic happening at different rooms or can also have multiple different projects happening simultaneously.