GYM 460 Design and Analysis

Jake Bode & Francisco Gonzalez

Due: December 5, 2023

Conceptual database design

Design rationale:

While designing the conceptual design for the relational model of GYM 460, per the entity-relationship model, we began by identifying a few vital entities and some relationships between them that would be important to assess in storing information for a fitness center/gym as described in this scenario. Important entities:

Member, Class, Course, Package, Trainer, Rental Item, Membership Level, Transaction

From here, we determined that outside of these entities and their identifying attributes, the remaining data that we wanted to store in the database would be able to be stored in relations representing the relationships between existing entities. These relationships and the entities they utilize are briefed in the following table and shown in more detail in the ER diagram:

Relationship Name	**Entities**	
Membership	Member	MembershipLevel
MemberClass	Member	Class
Offering	Course	Class
Teaching	Trainer	Class
CoursePackage	Course	Package
Log	Member	Transaction
RentalLog	RentalItem	Member
MemberPackage	Member	Package

We determined that we would include three weak entity sets within our relational model, MemberClass, which offers the classes that each member is taking; CoursePackage, which describes which courses are in which package(s); and MemberPackage, which gives which packages have been purchased by each member and the transaction ID that corresponded with each purchase.

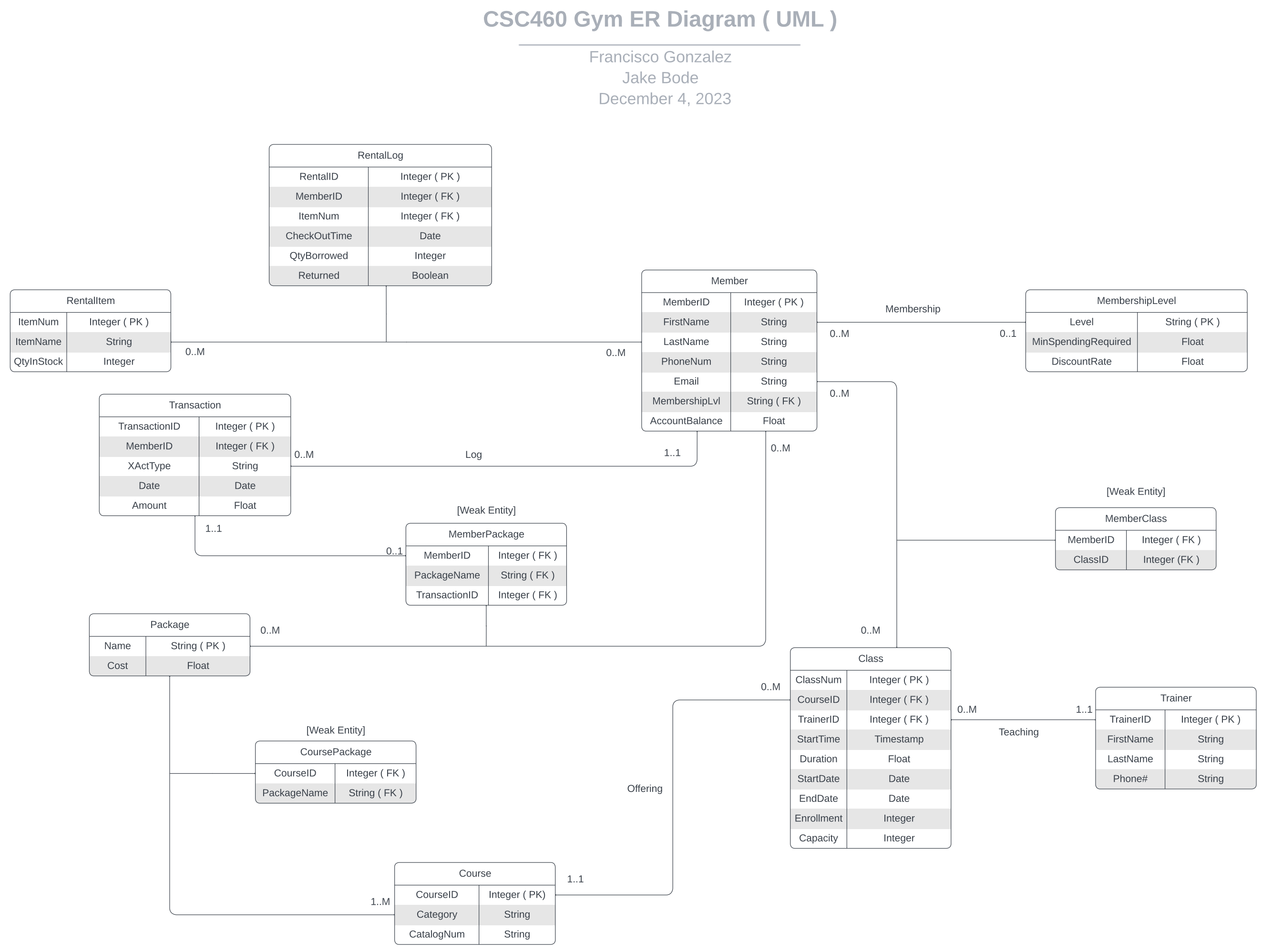
Other details:

A few non-essential attributes within the Member entity set that we determined to be useful to store here, and which will need to be computed after transactions for that member, are Membership Level, which can be determined using the minimum requirements for each level in the MembershipLevel entity set, and the AccountBalance field, which is straightforward to determine.

We have stored Package and Course as separate relations to reduce redundancy with the cost of a package, and since courses cannot be purchased individually, so the cost is an attribute of the package. Additionally, since a course can belong to multiple packages and packages can include a variable number of courses, we included the relationship between these entities as a relationship relation, CoursePackage.

Finally, determining whether the inputted new member information, class offering information, and class scheduling for trainers, all make sense in the scope of the gym, we will use Java pre-processing to avoid more complicated Oracle operations.

Following is our ER diagram, which gives more detail, including the attributes that each entity set will retain in our logical schema.



Logical Database Design

Following is a diagram showing the conceptual database design described in the ER diagram as a relational schema (PKs are underlined, FKs are bolded):

Member	MemberID	FName	LName	PhoneNum	Email	**MemLvl**	AcctBalance
MembershipLevel	LevelName	MinSpendingReq	DiscountRate				
MemberClass	**MemberID**	**ClassNum**					
Class	ClassNum	**CourseID**	**TrainerID**	StartTime	Duration	StartDate	EndDate
	Enrollment	Capacity					
Trainer	TrainerID	FName	LName	PhoneNum			
Course	CourseID	Category	CatalogNum				
Package	PackageName	Cost					
CoursePackage	**CourseID**	**PackageName**					
MemberPackage	**MemberID**	**PackageName**	**TransactionID**				
Transaction	TransactionID	**MemberID**	XactType	XactDate	Amount		
RentalItem	ItemNum	ItemName	QtyInStock				
RentalLog	RentalID	**MemberID**	**ItemNum**	OutTime	Quantity	Returned	

Normalization Analysis

Now, we will show that our schema adheres to third normal form or BCNF. In listing all the functional dependencies for each relation, trivial FDs will be ignored, as they do not contribute to whether the relation is in 3NF or BCNF. Additionally, we found it sufficient in the analysis to consider FDs with multiple attributes in the second set of the FDs since any further FD created with decomposition will adhere to the same properties as these FDs.

Entity Sets:

Member

FDs: $\{MemberID\} \rightarrow \{FName, LName, Phone\#, Email, MemLvl, AcctBalance\}$
The ID functionally determines all the other attributes of a member, and no other relevant functional relationship exists within this relationship. Since MemberID is a superkey of the relation, this relation is in BCNF.

MembershipLevel

FDs: $\{LevelName\} \rightarrow \{MinSpendingReq, DiscountRate\}$
Each membership level's name is unique, and since there may be multiple levels which have the same minimum spending requirements, but different discount rates, as the gym may determine that a certain membership level is inactive, but they want to keep the history of this information with the database. Therefore, since the name of a membership level is a candidate key of the relation, it is in BCNF.

MemberClass

No non-trivial FDs exist in this relation—neither attribute determines the other, and both make up the compound key. This relationship relation is in BCNF.

Class

FDs: $\{ClassNum\} \rightarrow \{CourseID, TrainerID, StartTime, Duration, StartDate, EndDate, Enrollment, Capacity\}$
None of the attributes in the right-hand set of attributes functionally determine any of the other attributes in the relation. The only potential one is the CourseID determining the Capacity, but different offerings of a course may allow for different numbers of members to be in the class. This FD, and thus the entire relation, adheres to BCNF, as the ClassNum is a superkey (further, a candidate key).

Trainer

FDs: $\{TrainerID\} \rightarrow \{FName, LName, Phone\#\}$
The trainer's identifying number functionally determines the trainer's name and phone number. No other FDs exist in this relation, so the relation is in BCNF.

Course

FDs: $\{CourseID\} \rightarrow \{Category, CatalogNum\}, \{Category, CatalogNum\} \rightarrow \{CourseID\}$
The course ID and the Category/Catalog Number combination are both candidate keys for this relation. Then, since in the first FD, CourseID is a candidate key, and in the second FD, {Category, CatalogNum} is a prime attribute set, as it is a CK itself, the relation is in 3NF.

Package

FDs: $\{PackageName\} \rightarrow \{Cost\}$
This relation simply gives the cost associated with each package, which is not necessarily unique. PackageName as the primary key is clearly a superkey, so this relation is in BCNF.

CoursePackage

No non-trivial FDs exist in this relation, so the relation must be in BCNF.

MemberPackage

FDs: $\{MemberID, PackageName\} \rightarrow \{TransactionID\}$,
$\{TransactionID\} \rightarrow \{MemberID, PackageName\}$
Both sets of attributes in these FDs are candidate keys, so both superkeys. Therefore, this relation is in BCNF.

Transaction

FDs: $\{TransactionID\} \rightarrow \{MemberID, Type, Date, Amount\}$
None of the attributes in the right-hand side of this FD functionally determine the transaction ID or the other attributes, since in practice, separate transactions of the same type can occur at the same time. Since the transaction ID is a candidate key, the FD meets the requirements for this relation to be in BCNF.

RentalItem

FDs: $\{ItemNum\} \rightarrow \{ItemName, QtyInStock\}, \{ItemName\} \rightarrow \{ItemNum\}$
Supposing that two separate rentable items cannot have the same name, item name is a candidate key for this relation. Then since the item number is the primary key, the left-hand sets in both FDs are candidate keys, and thus superkeys, so the relation is in BCNF.

RentalLog

FDs: $\{RentalID\} \rightarrow \{MemberID, ItemNum, OutTime, Quantity, Returned\}$
This is the only FD in this relation, since none of the attributes in the right-hand set determine the rental transaction or any of the other properties associated with the transaction. Thus, since RentalID is a CK, it is a superkey, so the relation is in BCNF.

We see that following this analysis, all the relations within this relational schema are in at least third normal form (most are also in Boyce-Codd Normal Form).

Query Description

Our self-designed query:

> *Given a member's ID, which items has this member checked out from the rental center that have not been returned yet?*

With this query, we will take a member ID as input from the program interface. From here, we will return the name of the items that have not been returned and the quantity of each of those items that have been borrowed by that member. The query uses data from the Member, RentalItem, and RentalLog relations.

Within the context of this gym, the rental center can use this query to answer gym members' questions about whether they have remembered to return all rental items. Additionally, when a member returns to the gym, this query can be run for that member to make sure that no charges need to be added to the member's account for lost rental items, which is not done by our program but can be done manually by the gym rental center.