**Class Activity:** Print out the All 1-2-3 NF Problems (2 copies per class), which is enough for 10 groups. One Case per group (2 – 3 students). They work on their problem, go write the solution on the board. I go around and check each solution at the board.

Then they type up their solution at the end of the “Normal Forms” homework document that we’ve been doing in class and upload it to Normal Forms Homework.

This activity, plus taking the Final Team Checkup Survey and the Course Evaluation Survey, and wrapping up loose ends, will take one class meeting. (Best to do it the last class meeting of the semester).

**Case 1:** A store wants to keep track of its bicycle and bicycle parts inventory. Each bicycle has many parts and a part is for one bicycle. The person who designed this database obviously doesn’t know much about database design, since this is the table that they made. Please fix this design and show the schema after you have converted it to 3rd Normal Form:

BICYCLE(BikeID, ManufNum, ManufName, ModelNum, Color, PartID1, Desc1, Price1, PartID2, Desc2, Price2, … PartID10, Desc10, Price10, Location)

BikeID PK🡪 ManufNum, ModelNum, Color

PartID PK🡪 Desc, Price

ManufNum PK🡪 ManufName, Location

Solution:

BICYCLE(BikeID, ManufNum, ModelNum, Color)

FK ManufNum 🡪 MANUF

PART(PartID, Descr, Price, bikeID)

FK bikeID 🡪 BICYCLE

MANUF(ManufNum, ManufName, Location)

**Case 2:** Doctor has many patients, and a patient can be seen by many doctors, which gives the associative table DOCTOR-PATIENT. Furthermore, that doctor/patient can have many appointments, which are represented by the repeating set of fields in this table (each appointment has one doctor-one patient). However whoever designed this table included way too many things in it, and it needs to be corrected. Please fix this design and show the schema after you have converted it to 3rd Normal Form:

DOCTOR-PATIENT( doctorID, patientID, doctorName, doctorSpecialty, patientName, patientAddress, patientDOB, appt\_ID1, appt\_Date1, appt\_time1, appt\_ID2, appt\_Date2, appt\_time2, appt\_ID3, appt\_Date3, appt\_time 3…., appt\_ID10, appt\_Date10, appt\_time10)

These are the functional dependencies (keys):

doctorID PK🡪 doctorName, doctorSpecialty

patientID PK🡪 patientName, patientAddress, patientDOB

Appt\_ID PK 🡪 appt\_date, appt\_time

Solution:

DOCTOR(doctorID, doctorName, doctorSpecialty)

PATIENT(patientID, patientName, patientAddress, patientDOB)

DOCTOR-PATIENT( doctorID, patientID)

FK doctorID 🡪 DOCTOR

FK patientID 🡪 PATIENT

APPOINTMENT(apptID, appt\_date, appt\_time, doctorID, patientID)

FK doctorID 🡪 DOCTOR

FK patientID 🡪 PATIENT

**Case 3:** A gym is trying to keep information concerning its fitness classes. Each class has a name, such as Pilates or Zumba, and other information as shown. However, the person who designed this database obviously doesn’t know much about database design, since this is the table that they made. Please fix this design and show the schema after you have converted it to 3rd Normal Form:

FITNESS\_CLASS( FitNumber, className, classDay, classTime, InstructorNumber, InstructorCertification, InstructorFirst, InstructorLast, InstructorDOB, ClientFname1, ClientLname1, ClientPhone1, ClientFname2, ClientLname2, ClientPhone2, ClientFname3, ClientLname3, ClientPhone3, …ClientFname20, ClientLname20, ClientPhone20)

These are the functional dependencies (keys):

FitNumber PK🡪 className, classDay, classTime, InstructorNumber

InstructorNumber PK🡪 InstructorCertification, InstructorFirst, InstructorLast, InstructorDOB

ClientID PK🡪 ClientFname, ClientLName, ClientPhone

Assumptions: An instructor can teach many classes, each class has just one instructor. Each class can have up to 20 clients, but each client is only enrolled in one class.

Solution:

INSTRUCTOR(InstructorNumber, InstructorCertification, InstructorFirst, InstructorLast, InstructorDOB)

FITNESS\_CLASS( FitNumber, className, classDay, classTime, InstructorNumber)

FK InstructorNumber 🡪 INSTRUCTOR

CLIENT(ClientID, ClientFname, ClientLName, ClientPhone, FitNumber)

FK FitNumer 🡪 FITNESS\_CLASS

**CASE 4:** The Acme Jewelry Importers is a company that consists of one or more departments. Each department has a set of at most 10 employees, who have an employee number, an office number, and a phone number. Each department has one or more employees, and an employee works in just one department. A manager may manage more than one department, and each department has exactly one manager. The person who made this schema obviously doesn’t understand normal forms, because the DEPARTMENT table has a repeating group (Emp\_Id, Ename, Office\_Id, Phone\_Id), as shown below:

DEPARTMENT(Dept\_Id, Dept\_Name, ManagerID, ManagerFirst, ManagerLast, Emp\_Id1, Ename1, Office\_Id1, Phone\_Id1, Emp\_Id2, Ename2, Office\_Id2, Phone\_Id2, … Emp\_Id10, Ename10, Office\_Id10, Phone\_Id10 )

Here are the functional dependencies (keys):

**Show the table scheme after converting it to 3NF**

Solution:

MANAGER(ManagerID, ManagerFirst, ManagerLast)

DEPARTMENT(Dept\_Id, Dept\_Name, ManagerID)

FK ManagerID 🡪 MANAGER

EMPLOYEE(Emp\_Id, Ename, Office\_Id, Phone\_Id, Dept\_ID)

FK Dept\_ID 🡪 DEPARTMENT

**Case 5:** Wendy Witchita owns a business that caters Halloween parties. After taking ITEC 1001 at GGC, she feels like she knows enough about databases to create a one. She created a database that will hold information about the Halloween parties that she is hired to cater.

A client is a person who hires Wendy to cater the party. A client may cater one or more parties, but each party has exactly one client. One or more invitations are sent out for a party (up to 30), and an invitation is for just one party. She has made the following table, but it violates the rules of normalization. Please fix this design and show the schema after you have converted it to 3rd Normal Form:

PARTY(PartyID, PartySize, PartyDate, ClientID, ClientFirst, ClientLast, ClientPhone, InvitationID1, InvFirst1, InvLast1, Street1, City1, State1, Zip1, DateSent1, RSVP1, InvitationID2, InvFirst2, InvLast2, Street2, City2, State2, Zip2, DateSent2, RSVP2, …InvitationID30, InvFirst30, InvLast30, Street30, City30, State30, Zip30, DateSent30, RSVP30)

Solution:

CLIENT(ClientID, ClientFirst, ClientLast, ClientPhone)

PARTY(PartyID, PartySize, PartyDate, ClientID)

FK ClientID 🡪 CLIENT

INVITATION(InvitationID, InvFirst, InvLast, Street, City, State, Zip, DateSent, RSVP, PartyID)

FK PartyID 🡪 PARTY