Assignment 6 ***Assigned: Feb 25***
Due: Apr. 1

Database definition and querying

Objectives: 1 Understand the translation of an entity-relationship conceptual schema into table definitions for a relational database.

2 Understand how a relational database can be queried, especially, how data from several tables can be combined in an answer.

Tasks: 1 Define a table in Microsoft Access and enter some data.

2 Run some SQL queries in a MS Access database.

Deliverables: Printouts of the Table T5COFInstructor (showing the data you entered), of the definitions of Queries 4a - c/d, and of the query results..

This assignment is primarily a tutorial, Tasks 1 and 2 are at the end.

1 Introduction. The process of designing a relational database

Before you can start using a relational database management system (DBMS), such as Microsoft Access, you need to understand the process of designing a database.

Database design starts with developing an entity-relationship conceptual schema. This schema specifies the types of data to be covered in the data base, but it does not say how the data should be stored. In a relational database data are stored in **tables**. A table has **rows and columns**. Any given cell can have only one value in it. We thus need to transform the E-R conceptual schema into table definitions. Then we can enter data and query the database.

Sample table: T1CourseOffering

COF	Course	Semester	Room	TimeSlot	Limit	Enrolled
COF01	FDST101	1979SP	HBL1125	13	35	23
COF02	FDST257	1979SP	F1101	05	50	45
COF03	FDST663	1979SP	F1150	08	15	15
COF04	FDST101	1979SM	F0113	15	40	31

Tables are very simple yet very flexible data structures that are easy to manipulate. An **object-oriented database** can have more complex data structures but is harder to manipulate.

2 Developing the conceptual schema and defining tables

Conceptual schema for a university database

Entity types	Relationship types			
Course offering (ID: string, COF...)				
Course (ID: string, AAAA999)	Course	<*offered as*>	Course offering	(1:N)
Semester (ID: str 9999[SP, SM, F]	Course offering	<*takes place in* >	Semester	(N:1)
Room (ID: string)	Course offering	<*meets in*>	Room	(N:1)
Time slot (ID: number [1 .. 20])	Course offering	<*scheduled in*>	Time slot	(N:1)
Count number	Course offering	<*has limit*>	Count number	(N:1)
	Course offering	<*has enrolled*>	Count number	(N:1)
Text	Course	<*has title*>	Text	(1:1)
Term	Course	<*deals with*>	Subject	(N:1)
Subject (ID: string S...)	Subject	<*designated by*>	Term	(1:1)
Person (ID: string)	(Course offering, Person)	<*has grade*>	Grade	n/a
Grade (ID: number [0,1,2,3,4]	Course offering	<*has instructor*>	Person	(N:N)

Explanation of symbols (by way of examples)

(1:1) A Subject is <*designated by*> exactly one (1)Term.
A Term <*designates*> exactly one (1) subject (in this DB).

(N:1) A Course offering <*meets in*> exactly one (1) Room.
A Room may <*serve as meeting place*> for many (N) Course offerings.

(1:N) A Course may be <*offered as*> many (N) Course offerings.
A Course offering <*is offering of*> exactly one (1) Course.

(N:N) A Course offering may <*has instructor*> many (N) Persons.
A Person may <*be instructor for*> many (N) Courses.

Note: These symbols are meaningful only for binary relationships.
"Many" in this context means "more than one".

From entity-relationship conceptual schema to tables in a relational database

A table has rows and columns. Any given cell can have only one value in it. We could make a table for every relationship type, resulting in 11 tables. However, it is more efficient to keep the number of tables small by combining several relationship types into one table where possible. Here are the rules; as we apply these rules, you will see the rationale behind them.

Table definition rules

1 A multi-way relationship (3 or more) needs its own table, one row per statement.

This leaves binary relationships as candidates for combining into one table.

2 A N:N relationship needs its own table, one row per statement.

3 Formulate all remaining binary relationships so they are (N:1) or (1:1). All relationships with the same entity type on the left hand side can be combined into one table.

Applying these rules to the relationship types in the example:

By Rule 1, we select the three-way relationship *<has grade>* and make a table for it. This is easy: To each *<has grade>* statement corresponds a row in the table. Columns 1, 2, 3 correspond to the argument positions 1, 2, 3 in the relationship.

Table T4COFStudentGrade

(COF01, Tarr, L.) *<has grade>* 4
(COF01, Lund, M.) *<has grade>* 2

COF	Student	Grade
COF01	Tarr, L.	4
COF01	Lund, M.	2

All remaining relationships are binary. So, by Rule 2, we select *<has instructor>* since it is N:N (a course offering can have more than one instructor, an instructor can teach more than one course offering). Again, to each *<has instructor>* statement corresponds one row in the table; the table has just two columns. (See next page)

The remaining relationships are (1:1), (N:1), or (1:N); we can apply Rule 3 to them.

Course *<offered as>* Course offering (1:N) can be turned around to

Course offering *<is offering of>* Course (N:1)

The relationships can then be grouped into three blocks (starting at the top):
Block 1 in which all relationships start with Course offering
Block 2 in which all relationships start with Course
Block 3 in which all relationships start with Subject
Each block can be represented by a single table (see next page).

Conceptual schema for a university database (repeated)

Entity types	**Relationship types**			
Course offering (ID: string, COF...)	(First relationship turned around)			
Course (ID: string, AAAA999)	**Course offering**	***<is offering of>***	**Course**	**(N:1)**
Semester (ID: str 9999[SP, SM, F]	Course offering	*<takes place in >*	Semester	(N:1)
Room (ID: string)	Course offering	*<meets in>*	Room	(N:1)
Time slot (ID: number [1 .. 20])	Course offering	*<scheduled in>*	Time slot	(N:1)
Count number	Course offering	*<has limit>*	Count number	(N:1)
	Course offering	*<has enrolled>*	Count number	(N:1)
Text	Course	*<has title>*	Text	(1:1)
Term	Course	*<deals with>*	Subject	(N:1)
Subject (ID: string S...)	Subject	*<designated by>*	Term	(1:1)
Person (ID: string)	(Course offering, Person)	*<has grade>*	Grade	n/a
Grade (ID: number [0,1,2,3,4]	Course offering	*<has instructor>*	Person	(N:N)

Explanation of symbols (by way of examples)

(1:1) A Subject is *<designated by>* exactly one (1)Term.
A Term *<designates>* exactly one (1) subject.

(N:1) A Course offering *<meets in>* exactly one (1) Room.
A Room may *<serve as meeting place>* for many (N) Course offerings.

(1:N) A Course may be *<offered as>* many (N) Course offerings.
A Course offering *<is offering of>* exactly one (1) Course.

(N:N) A Course offering may *<has instructor>* several (N) Persons.
A Person may *<be instructor for>* several (N) Courses.

Note: These symbols are meaningful only for binary relationships.
"Many" in this context means "more than one".

Table T5COFInstructor (for relationship <*has instructor*>, which is (N:N)

COF01 <*has instructor*> Kahn, L.
COF02 <*has instructor*> Kahn, L.
COF03 <*has instructor*> Simms, B.
COF03 <*has instructor*> Zog, H.

COF	Instructor
COF01	Kahn, L.
COF02	Kahn, L.
COF03	Simms, B.
COF03	Zog, H.

Note that COF03 needs two lines in Table T5.

The next three tables each represent a block of (N:1) relationships. The first block consists of six relationships, all starting with Course offering. We could express a group of statements formed with these relationships through six 2-column tables, but we can also express them as **one** table with 7 columns. Each row corresponds to a Course offering value. The column 1 takes the Course offering ID; columns 2 - 7 each take the right-hand value of one of the statements:

Table T1CourseOffering

COF01 <*is offering of*> FDST101
COF01 <*takes place in* > 1979Sp
COF01 <*meets in*> HBL1125
COF01 <*scheduled in*> 13
COF01 <*has limit*> 35
COF01 <*has enrolled*> 23

COF	Course	Semester	Room	TimeSlot	Limit	Enrolled
COF01	FDST101	1979SP	HBL1125	13	35	23

Thus, COF01 <*belongs to*> FDST101, COF01 <*takes place in*> 1979SP, etc. Put differently, each column is defined by a relationship type.

Why not add the relationship

Course offering <*has instructor*> Person (N:N)

as an 8th column to this table? (Hint: How would you handle COF03)

The two relationship types in the next block can also be represented as one three-column table:

Table T2Course

FDST101	<*has title*>	Introduction to food processing	
FDST101	<*deals with*>		S12

Course	**Title**	**Notation**
FDST101	Introduction to food processing	S12

Each row corresponds to a course

Each of the other relationships needs its own table as follows (see explanation with Table T4):

Table T3Subject

S12 <*designated by*> Food processing

Subject	**Term**
S12	Food processing

Note that Tables T4 and T5 look just like Tables T1- T3. All tables are handled by the DBMS in the same way. But the interpretation of the tables is quite different:

Each row in a table can be seen as an instance of a frame. (The columns define the slots.) Tables T4 and T5 correspond to *minimal frames* (as defined in Lecture 4): Each frame represents one statement, each slot one argument of the relationship with which the statement is formed. Table 1 corresponds to an *extended frame*: Each row represents a block of binary statements, all starting with the same entity (the *focal entity*), each slot/column represents a binary relationship to the focal entity. In many contexts, the terminology is *file*, *record*, *field*. In object-oriented databases, the terms are *object* and *variable*. The following table shows the correspondences:

Table	File		
Row	Record	Frame	Object
Column	Field	Slot	Variable

Complexity increases from left to right: A field in a record can be *repeating* (have multiple occurrences), frame slots and object variables can have whole frames or complex objects as values (as in nesting boxes within boxes). Furthermore, a frame slot can have *procedural attachments*, and an object can have associated procedures, called *methods*, that process the data defined by the object's variables.

Complete tables with data

Table T1CourseOffering

COF	Course	Semester	Room	TimeSlot	Limit	Enrolled
COF01	FDST101	1979SP	HBL1125	13	35	23
COF02	FDST257	1979SP	F1101	05	50	45
COF03	FDST663	1979SP	F1150	08	15	15
COF04	FDST101	1979SM	F0113	15	40	31
COF05	FDST101	1979F	HBL1125	03	35	34
COF06	FDST257	1979F	F0112	15	15	12
COF07	CMSC620	1979F	HBL4115	20	25	18
COF08	CMSC424	1979F	HBL0109	05	60	45
COF09	CMSC420	1979F	HBL0103	14	15	13
COF10	FDST663	1980SP	F1150	04	15	12
COF11	CMSC424	1980SP	HBL0109	04	60	47
COF12	FDST101	1980SP	HBL1125	09	35	33
COF13	CMSC824	1980SP	HBL0109	07	20	15

Table T2Course

Course	Title	Subject
FDST101	Introduction to food processing	S12
FDST663	Seminar in meat canning	S17
FDST257	Vegetable pickling	S13
CMSC424	Database design	S19
CMSC620	Problem solving methods in artificial intelligence	S20
CMSC420	Data structure	S18
CMSC824	Relational database design	S19

Table T3Subject

Subject	Term
S12	Food processing
S13	Vegetable pickling
S17	Meat canning
S18	Data structure
S19	Database management
S20	Artificial intelligence

Table T4COFStudentGrade

COF	Student	Grade
COF01	Tarr, L.	4
COF01	Lund, M.	2
COF01	Kolb, T.	3
COF01	Doe, V.	0
COF02	Doe, J.	4
COF02	Smith, R.	4
COF03	Clay, S.	3
COF03	North, A.	3
COF03	Zipf, E.	1
COF04	Manet, J.	0
COF04	Kim, A.	4
COF04	Phillip, N.	3
COF05	Sprotto, L.	2
COF05	Jones, R.	4
COF06	Doe, V.	4
COF06	Jones, R.	4
COF06	Zipf, E.	3
COF07	Wang, L.	4
COF07	Meyer, P.	3
COF07	Gonzalez, A.	3
COF08	Gonzalez, A.	4
COF08	Hsiao, T.	2
COF08	Dellum, T.	3
COF08	Bush, M.	4
COF09	McCall, H.	4
COF09	Andreotti, S.	3
COF09	Yeltsin, B.	1
COF09	Sun, Y.	3
COF10	Tarr, L.	3
COF10	Doe, J.	4
COF10	Kolb, T.	3
COF11	McCall, H.	4
COF11	Yeltsin, B.	4
COF11	Chu, W.	3
COF12	Simon, R.	4
COF12	Gold, D.	3
COF12	Darrell, F.	1
COF12	Kovak, J.	3
COF12	David, J.	3
COF13	Gonzalez, A.	3
COF13	Hsiao, T.	3
COF13	Andreotti, S.	4
COF13	Sun, Y.	4

Table T5COFInstructor

COF	Instructor
COF01	Kahn, L.
COF02	Kahn, L.
COF03	Simms, B.
COF03	Zog, H.
COF04	Clay, S.
COF05	Kahn, L.
COF06	Simms, B.
COF07	Charniak, E.
COF07	Winston, P.

COF	Instructor
COF08	Date, C.
COF09	Minker, J.
COF10	Simms, B.
COF11	Minker, J.
COF12	Clay, S.
COF13	Codd, E.
COF13	Date, C.

3 Using Microsoft Access

3.1 Tutorial

All tables except T5COFInstructor are already defined in a database called University (on the distributed diskette) and "populated" (filled) with data. So you will first try running some queries, both predefined and created by you. You can open the University database from drive a: or copy it to your hard disk for faster operation.

Start MS Access.

Be sure the radio button *Open an existing file* is turned on.

If *University.mdb* is not already on the menu, click on *More files*, navigate to the directory where the database is stored. Double-click on *University.mdb*. A small database window opens; observe the navigation bar at the left.

Access opens with a list of tables. Double-click on *T1CourseOffering*. Examine the table, then close it. Right-click on the table; in the pop-up menu, click on *Design View* and examine the design window, then close it. Open the other tables and examine them.

Note: The main advantage of a database management system (DBMS) is that it can display data in any combination and format the user requires; it can present many views on the data. (This is a general principle of using computers for providing information. The driving force behind XML is structuring information so that it can be displayed in many ways, reused, "repurposed"). In the tutorial, you will display data using existing queries. We will start with a query for data from a single table and move on to queries that combine data from several tables, which is where the real power lies. One further point: In Access, queries can be shown in *Design view* or in *SQL* (Structured Query Language); you will look at queries both ways, but mostly in SQL.

In the navigation bar, click on *Queries*. Then double-click on *Query1a*; this will *run* the query and extract and format data from one or more tables as specified in the query. You will see a different display of data from table T1: Only some columns of data are displayed, the columns are in a different order, and the rows are sorted by course number.

Now, right-click on *Query1a*; in the pop-up menu, click on *Design View*. In the design view you can see how the query is specified in a format that approaches WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et). To see the SQL presentation, in the top menu bar, click on *View*, in the drop-down menu click on *SQL View*. You should see this (not as nicely formatted):

```
SELECT     T1CourseOffering.Course, T1CourseOffering.COF,
           T1CourseOffering.Semester, T1CourseOffering.Room
FROM       T1CourseOffering
ORDER BY   T1CourseOffering.Course;
```

This should be self-explanatory. In this example only, SQL keywords are bold.

SELECT	is followed by the columns (fields) we want to show,
FROM	introduces the table(s) from which data are do be displayed
ORDER BY	specifies the *sort key(s)* for sorting the rows displayed

Check out Query 1b the same way. In SQL view you see (the line added to Query 1a is bolded):

```
SELECT    T1CourseOffering.Course, T1CourseOffering.COF,
          T1CourseOffering.Semester, T1CourseOffering.Room
FROM      T1CourseOffering
WHERE     T1CourseOffering.Course) >= "FDST"
ORDER BY  T1CourseOffering.Course;
```

Query 1b adds a selection condition so that only selected rows are displayed. The SQL keyword is **WHERE**.

It would be nice to see the course titles in the display. But Table T1CourseOfferings has only the course number. We could use the course number to access Table T2Course and get the course title from there. A relational database supports just this kind of combination of data from several tables. Check out Query 2a. (To see the full display, maximize the window.)

Query 2a (in SQL view, minus the extraneous [] MS Access puts in)

Note: From now on, the queries will be just in SQL because it is easier to deal with combination of data from several tables. All SQL queries are given here so you need not look the up in MS Access.

The part added to Query 1a is bolded. It is the matching condition that selects the correct row from Table T2 so we get the correct table.

```
SELECT    T1CourseOffering.Course, T2Course.Title, T1CourseOffering.COF,
          T1CourseOffering.Semester, T1CourseOffering.Room
FROM      T1CourseOffering, T2Course
WHERE     T2Course.Course = T1CourseOffering.Course
ORDER BY  T1CourseOffering.Course;
```

Note 1: This query corresponds to a two-step search in a graphical representation of the data: From Course offering to Course (based on Table T1), from Course to Title (based on Table T2)

Note 2: We could have added another column to Table T1CourseOffering. How often would the title for FDST01 appear in the revised table? What would this do to storage space and, more importantly, input effort and error possibilities?

Query 2b

```
SELECT    T1CourseOffering.Course, T2Course.Title, T1CourseOffering.COF,
          T1CourseOffering.Semester, T1CourseOffering.Room
FROM      T1CourseOffering, T2Course
WHERE     T2Course.Course=T1CourseOffering.Course
          AND T1CourseOffering.Course) >= "FDST"
ORDER BY  T1CourseOffering.Course;
```

We will now build, step by step, a query to produce transcripts. That means, we will need to show grades by student, so we will start with a query on Table T4COFStudentGrade. Check out the data display for each query. You can see the SQL form here.

Query 3a

```
SELECT    T4COFStudentGrade.Student, T4COFStudentGrade.COF,
          T4COFStudentGrade.Grade
FROM      T4COFStudentGrade
ORDER BY  T4COFStudentGrade.Student;
```

A transcript should show the course number and the semester. We can get these pieces of information from Table T1CourseOffering, matching on COF:

Query 3b (additions to Query 3a bolded)

```
SELECT    T4COFStudentGrade.Student, T4COFStudentGrade.COF,
          T1CourseOffering.Course, T1CourseOffering.Semester,
          T4COFStudentGrade.Grade
FROM      T4COFStudentGrade, T1CourseOffering
WHERE     T1CourseOffering.COF=T4COFStudentGrade.COF
ORDER BY  T4COFStudentGrade.Student;
```

Finally, the transcript should contain the course title, which we can get from Table T2Course, matching on Course (a three-step search/navigation). And there are three smaller things to fix:

(1) The course offering number (COF) is not needed in the transcript, so we take it out from the list of fields following SELECT. (But is still plays a vital role in linking the tables.)

(2) The courses on a transcript should appear by semester, and within semester in course number order, so we add a second and third sort key after ORDER BY.

(3) The semester column should appear after the student, before the course number, so we rearrange the order of the fields after SELECT.

Here is the final result:

Query 3c (additions to Query 3b are bolded)

```
SELECT    T4COFStudentGrade.Student, T1CourseOffering.Semester,
          T1CourseOffering.Course, T2Course.Title,
          T4COFStudentGrade.Grade
FROM      T4COFStudentGrade, T1CourseOffering, T2Course
WHERE     T1CourseOffering.COF=T4COFStudentGrade.COF AND
          T2Course.Course=T1CourseOffering.Course
ORDER BY  T4COFStudentGrade.Student, T1CourseOffering.Semester,
          T1CourseOffering.Course;
```

b

3.2 Your tasks

Task 1. Define and populate a table

Define Table T5COFInstructor (see Complete tables with data at the end of Section 2)
In the Navigation bar to the left, double click on *Tables*
Double click on *Create table in Design view*
In the window that opens, enter a line for each of the two fields/columns.
When you are done, click the x in the upper right hand corner, answer *Yes* to save, in the box that opens enter the table name *T5COFInstructor,* answer *No* to primary key.
Double click on the new table and enter data in the window that comes up.
Hint: To speed up data entry, copy COF, then paste it every time you need it, using the shortcut key Ctrl-V.

Task 2. Define some queries

Query 4a: An alphabetical list of instructors with the course offerings they teach.
Query 4b: Add a column for course number to the display.
Query 4c: Add course titles, omit the course offering from the display.
Extra challenge
Query 4d: Instructors and the subjects they teach (as seen from the subjects of their courses).

Note: To print deliverables

Printing query definition: In design view you can see the SQL definition. Just copy and paste into a document.

Table content and query results can be printed directly.

Note: If you ever need to print table definitions, here is how to do it:

Tools > Analyze > Documenter.

Select the type of object (table, query, etc), check the specific objects you want to document, and click OK.

You will see a possibly lengthy display which can be printed.