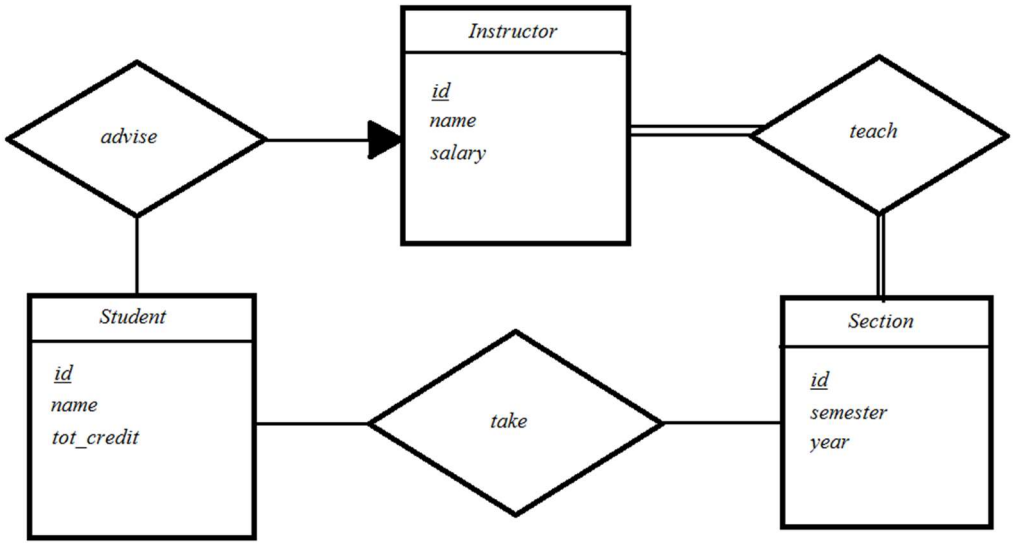
From ER Diagram, to Tables, then to SQL Scripts

Eric Lu

Here we have a ER diagram, with entities *instructor*, *student* and *section*. Instructors can teach sections, and students take them; instructors also advise students. All instructors teach all sections.



Instructor

id	*name*	*salary*
1	Einstein	80,000
2	Turing	70,000
3	Hawkings	90,000
4	Feyman	85,000

Student

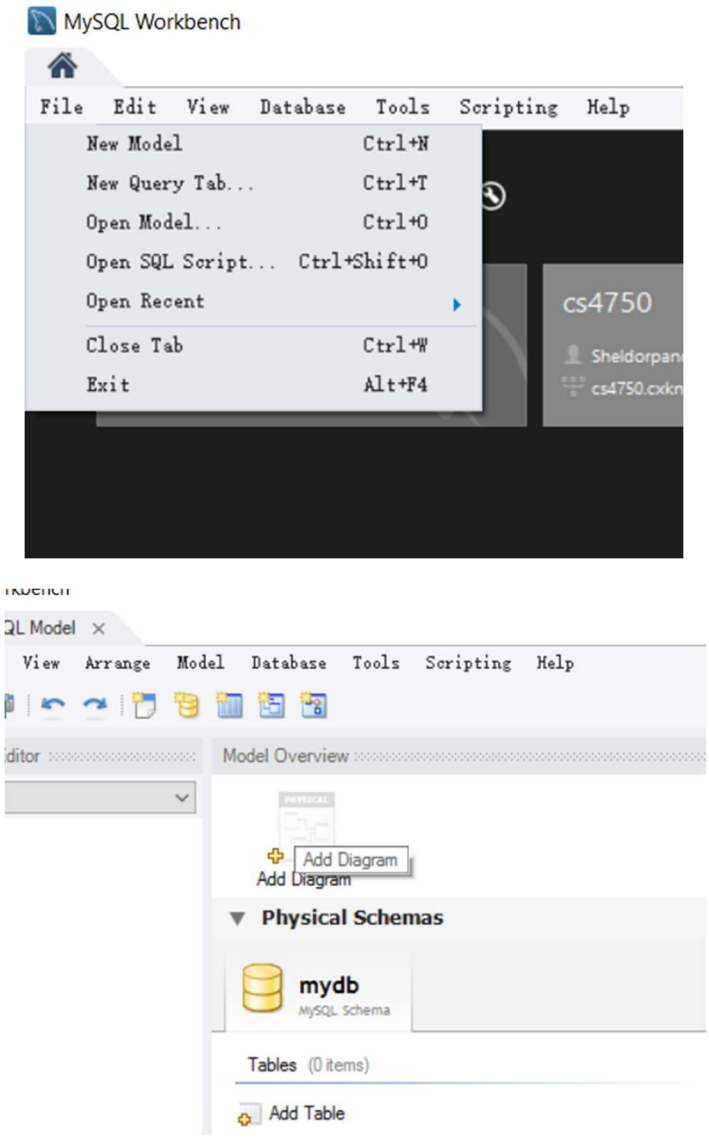
id	*name*	*tot_credit*
1	Eric	3.9
2	Tom	4.0
3	Mike	2.1
4	Anna	4.0
5	Lucy	3.8

Section

id	*semester*	*year*
1	Fall	2015
2	Spring	2016
3	Fall	2016

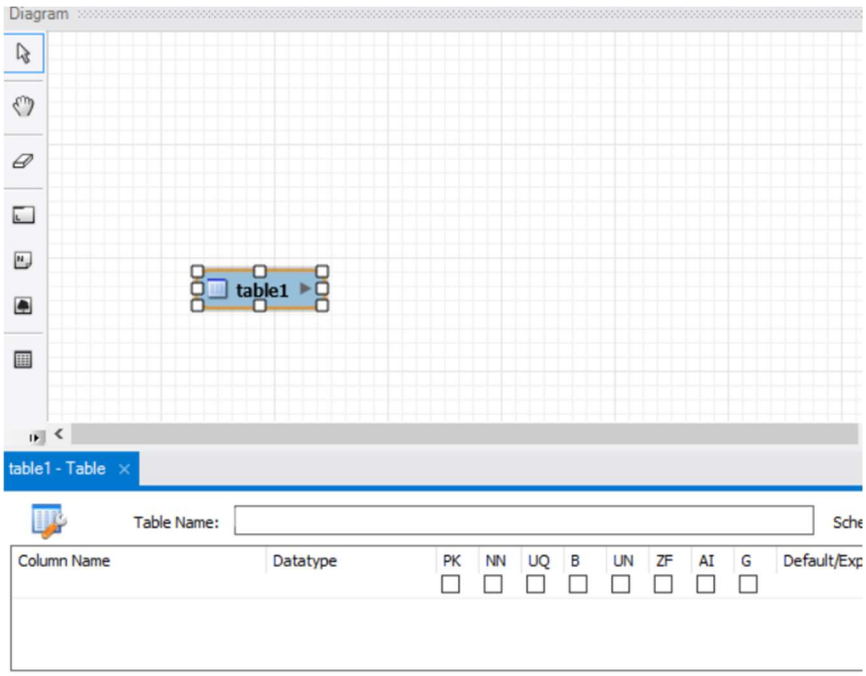
To see how a ER diagram is physically converted into tables in SQL and how it can be translated into SQL scripts, we can use MySQL workbench to draw a model. This instruction shows you how the procedure: theoretical ER-Diagram -> tables -> SQL script, can be worked out. In fact, this is a vital procedure in actual database creation, and will be extremely useful in your semester project.

Step 1: Enter MySQL Workbench, find “New Model” and double click on ”Add Diagram”.



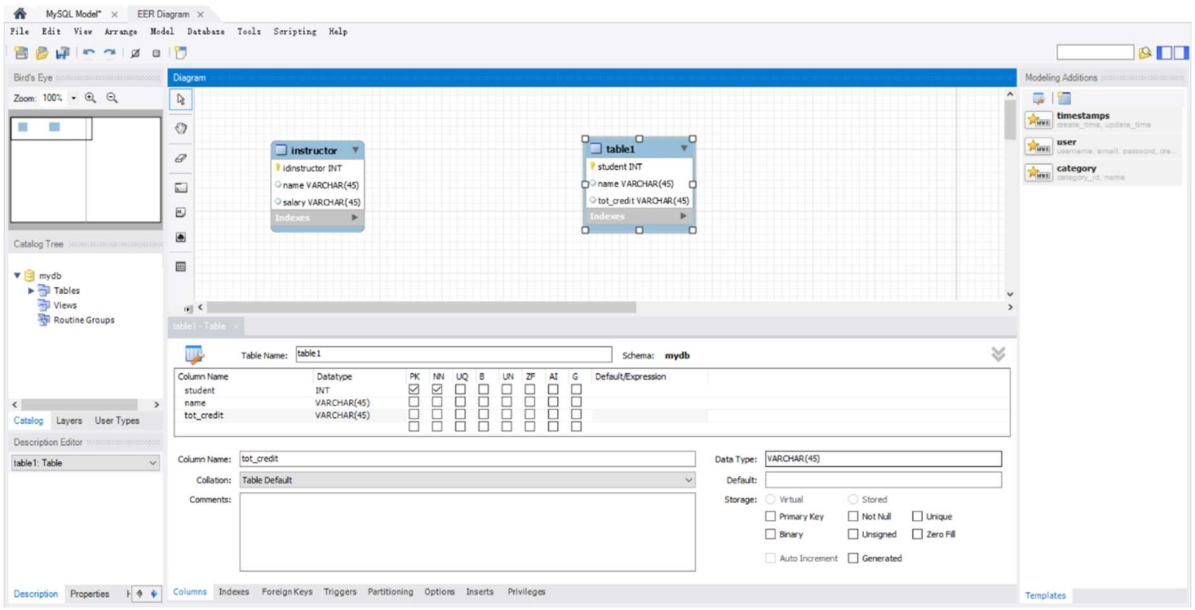
Step 2: Now we can see an empty diagram. Notice that this diagram is slightly different from the ER Diagram that we learned in class. The diagram we taught in class is a theoretical representation (what are the entities, what are the relationships, how do they associate); here, this diagram is its equivalent physical representation (what actual tables do we need to represent the entities and relationships, what attributes are inside).

Try the icons on the left side. Create a table. Notice that a table is a different concept from an entity/relationship! An entity/relationship can be represented with a table at physical level.



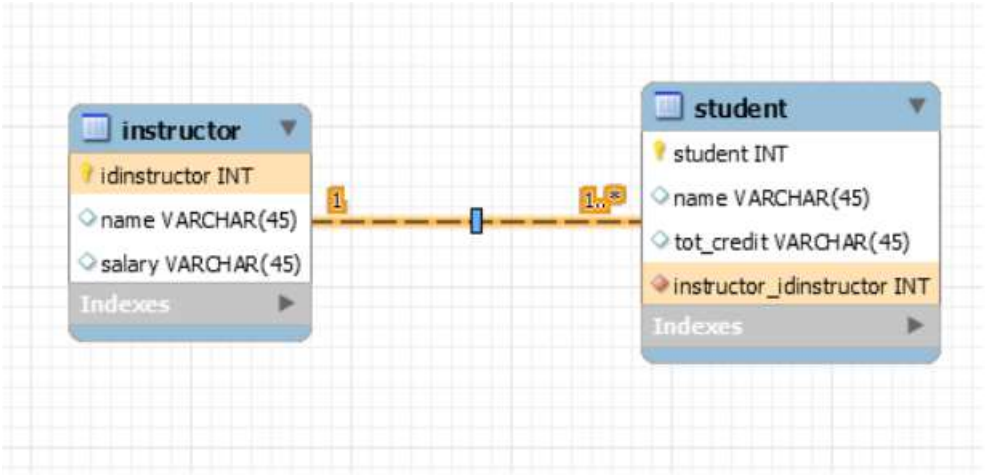
Step 3: Double click on the table and edit its attributes. “PK”, “NN” and “UQ” mean primary key, not-null and unique, respectively. You are highly encouraged to look up what the other symbols mean, and what will happen if they are selected. Also, you are highly encouraged to check out what kind of data types are there in SQL. You can change the notations of any table and relationship in Model >> Object Notation/Relationship Notation.

Here, we are going to create an *instructor* table to represent the *instructor* entity that we talked in class.



Step 4: Now we have two tables. One represents instructor and another student. We now need to establish the relationship advisor between them. Choose 1:n relationship (one to many) on the left side and connect them.

There are two types of 1:1 and 1:n relationships here: identifying and non-identifying. It will be the best if you can look it up and distinguish them. Hint: they are related to a concept named "strong and weak entities". Here, usually, we use non-identifying relationship. (After you understand what is "identifying", think: why there is no such thing as an identifying/non-identifying many-to-many relationship?)



Step 5: After we established the 1:n relationship, notice that an attribute automatically appears in the student table – a foreign key of instructor ID. The reason for this is that the most space-saving method to store an 1:n relationship is to append the primary key of the "1" side to the "n" side as a foreign key. As shown below:

Student

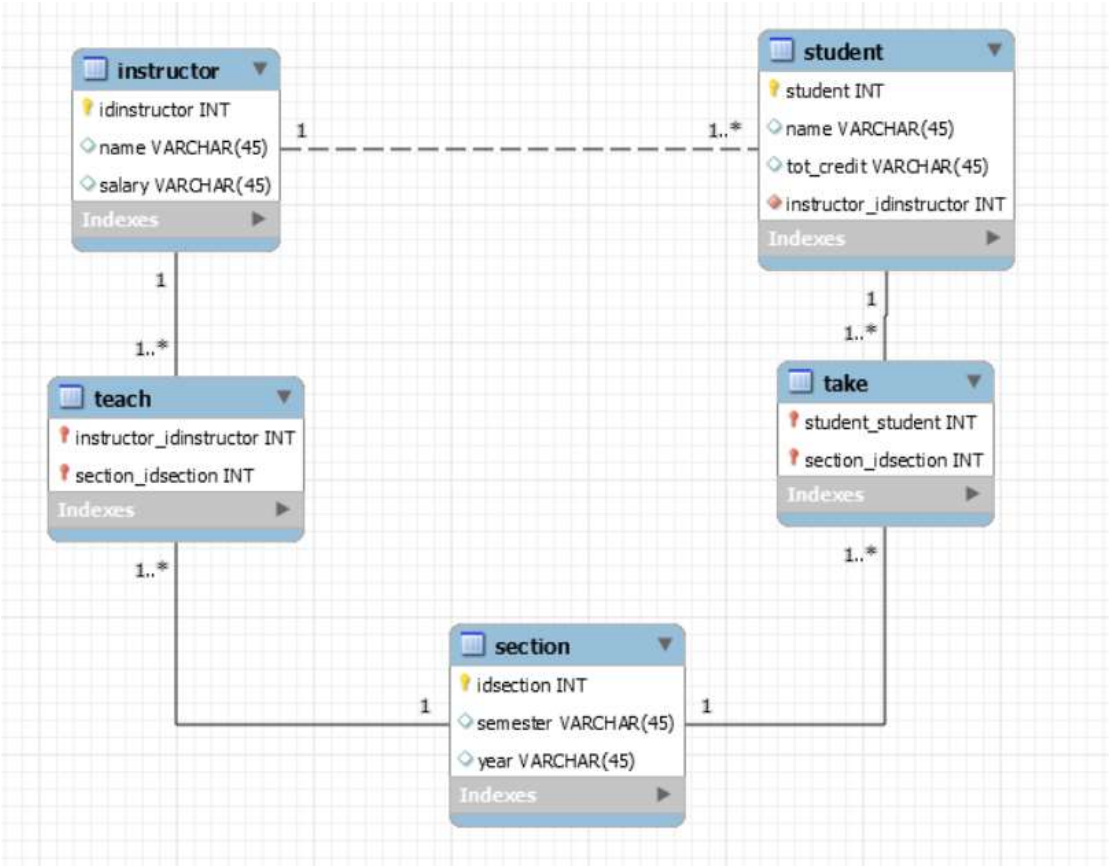
id	*name*	*tot_credit*	*instructor_id*
1	Eric	3.9	1
2	Tom	4.0	1
3	Mike	2.1	2
4	Anna	4.0	2
5	Lucy	3.8	3

We can right click on the relationship line and edit its name in both directions.

From the table above, think of this easy question: who is the instructor of each student respectively? How do you know?

Step 6: Now we are going to create another table to represent the entity *section*, and create a many-to-many relationship *teach* between *instructor* and *section*. Notice that an entire new table, holding the primary keys of *instructor* and *section*, jumps out, due to the fact that the most feasible way to store a many-to-many (n:m) relationship is to store pairs of primary keys from both sides into a new table.

Also, notice that the relationship *teach* is a total participation on both sides, although this feature cannot be shown on the graph.



Take

student_id	*section_id*
1	1
1	2
2	3
4	2
4	3
5	1
5	2
5	3

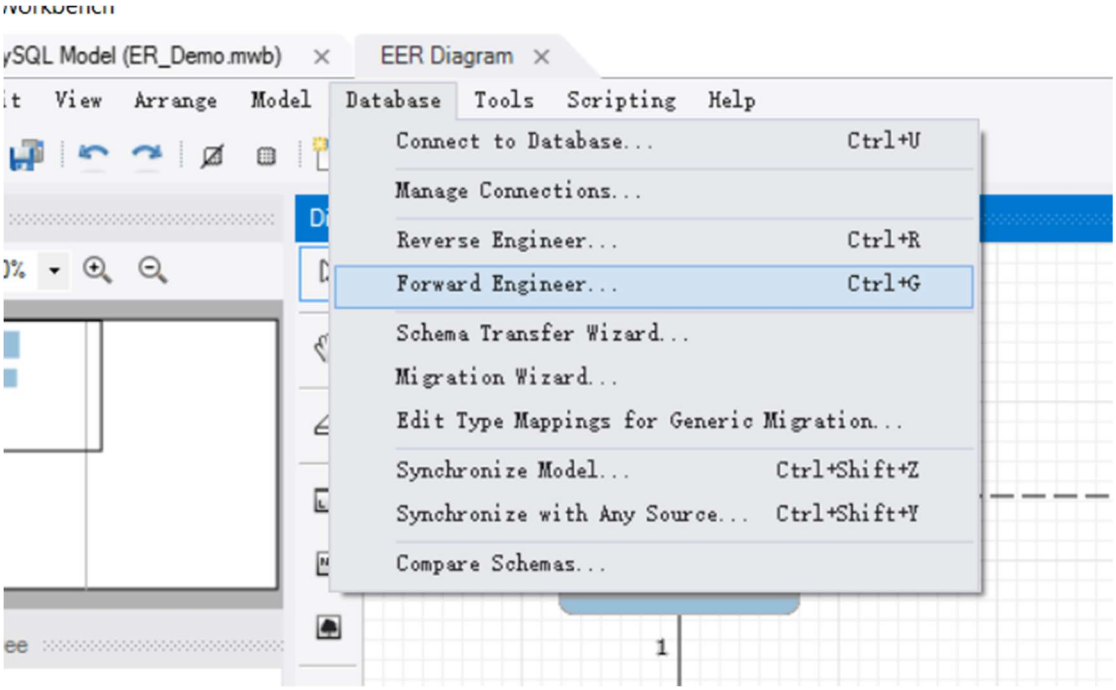
Teach

instructor_id	*section_id*
1	1
1	2
2	1
3	1
4	3

After created both *teach* and *take*, you are also encouraged to try establishing a 1:1 relationship on your own, and find out what will automatically happen and why.

Step 7: After we finish creating the diagram, we will translate it into SQL scripts. Select Database >> Forward Engineer. It will require you to choose a database in order to hold the generated tables. Usually MySQL provides a default local database, and you can use it or any other databases.

Continue to click on “next” until the script is shown. Read through the script and figure out how it relates to the theoretical ER diagram and its physical representation. Think: is the script Data Definition Language (DDL) or Data Manipulation Language (DML)?




```
19  -- -----------------------------------------------------
20  CREATE TABLE IF NOT EXISTS `mydb`.`instructor` (
21    `idinstructor` INT NOT NULL,
22    `name` VARCHAR(45) NULL,
23    `salary` VARCHAR(45) NULL,
24    PRIMARY KEY (`idinstructor`))
25  ENGINE = InnoDB;
26
27
28  -- -----------------------------------------------------
29  -- Table `mydb`.`student`
30  -- -----------------------------------------------------
31  CREATE TABLE IF NOT EXISTS `mydb`.`student` (
32    `student` INT NOT NULL,
33    `name` VARCHAR(45) NULL,
34    `tot_credit` VARCHAR(45) NULL,
35    `instructor_idinstructor` INT NOT NULL,
36    PRIMARY KEY (`student`),
37    INDEX `fk_table1_instructor_idx` (`instructor_idinstructor` ASC),
38    CONSTRAINT `fk_table1_instructor`
39      FOREIGN KEY (`instructor_idinstructor`)
40      REFERENCES `mydb`.`instructor` (`idinstructor`)
41      ON DELETE NO ACTION
42      ON UPDATE NO ACTION)
43  ENGINE = InnoDB;
44
45
46  -- -----------------------------------------------------
47  -- Table `mydb`.`section`
48  -- -----------------------------------------------------
49  CREATE TABLE IF NOT EXISTS `mydb`.`section` (
50    `idsection` INT NOT NULL,
51    `semester` VARCHAR(45) NULL,
52    `year` VARCHAR(45) NULL,
53    PRIMARY KEY (`idsection`))
```