COMP 1140: Database and SQL | Brendan Shea, PhD (Brendan.Shea@rctc.edu)

AN INTRODUCTION TO THE ENTITY-RELATIONSHIP MODEL[i]

In this lesson, we'll be answering the following questions:

1. What are the basic concepts of the **entity-relation model (ERM)?**
2. How can **entity-relation diagrams (ERDs)** help us visualize and communicate information about the ERM?
3. What does it mean for an entity to be **existent-dependent?** What is the difference between **strong** and **weak entities?**
4. How are entities and attributes depicted on E-R diagrams?

THE ENTITY RELATIONSHIP DATA MODEL

The **entity-relationship model (ERM)** has been in use for almost 50 years. It is well suited to data modelling for use with databases because it does well with both "concept-level" and "implementation-level" modeling. It is also easy to discuss and explain, which makes it a great way for database designers and administrators to *communicate* with clients, programmers, and users. ERMs are readily translated to the *relational* data model that is the basis of many modern DBMSs, but can also be translated to other sorts of models (for example, object-oriented, or NoSQL). ER models, also called an ER schema, are represented by **Entity-Relationship Diagrams (ERDs).**

ER modelling is based on two concepts:

- **Entities** correspond to different types of objects about which we have gathered data. These correspond the *tables* of the relational model. **Entity instances** correspond to the rows of the relational model.
- **Relationships**, defined as the associations or interactions between entities

Example: Professor B (entity) teaches (relationship) COMP 1140: Introduction to Database and SQL (entity).

What's the point of ERDs? A completed ER diagram provides a sort of "blueprint" for the database that we will create. It can, among other things, help determine *which* sort of DBMS to use (Relational? NoSQL? Object?). The design of an ERD must reflect an organization's operations accurately if the database is to meet that organization's data requirements, and can help guide the creation of a **relational schema** (a formal list of tables for a relational database). The ERD form the basis for a final check on whether the included entities are appropriate and sufficient, on the attributes found within those entities, and on the relationships between those entities. It is also used as a final crosscheck against the proposed data dictionary entries. The completed ER diagram also lets the designer communicate more precisely with those who commissioned the database design. Finally, the completed ER diagram serves as the implementation guide to those who create the actual database.

BUSINESS RULES FOR A "COMPANY" DATABASE

For this lesson, we will use a sample database called the COMPANY database to illustrate the basic concepts of ERM design and development. This database contains information about employees, departments and projects.

Important **business rules** that must be captured in the ERM include:

1. There are several departments in the company.
2. Each department has a unique identification, a name, location of the office and a particular employee who manages the department.
3. A department controls a number of projects, each of which has a unique name, a unique number and a budget.
4. Each employee has a name, identification number, address, salary and birthdate.
5. An employee is assigned to one department but can join in several projects.
6. We need to record the start date of the employee in each project.
7. Each employee has exactly one direct supervisor.
8. We want to keep track of the dependents and spouses for each employee.
9. Each dependent or spouse has a name, birthdate and relationship with the employee.
10. Not all employees will have dependents or spouses.

KINDS OF ENTITIES

As we've discussed in previous lessons, an *entity* is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be (1) an object with physical existence (e.g., a lecturer, a student, a car) or (2) an object with conceptual existence (e.g., a course, a job, a position). The ER model distinguishes between several e**ntity types** including independent entities, dependent entities, strong entities, weak entities, and characteristic entities. These are described below.

Existent-independent entities also referred to as **kernels, strong entities,** or **regular entities**, are the backbone of the database. They are what other tables are based on. *Kernels* have the following characteristics:

1. They are the building blocks of a database.
2. The primary key may be simple or composite.
3. The primary key is not a foreign key.
4. They do not depend on another entity for their existence.

Example: U.S. citizens is uniquely identified by their social security number, and hence are "strong" entities (that don't depend on other entities to distinguish them). In the COMPANY database, examples of an independent entity include the Customer table, Employee table or Product table.

Existent-dependent entities, also referred to as *derived entities*, depend on other tables for their meaning. These entities have the following characteristics:

1. Dependent entities are used to connect two kernels together.
2. They are said to be existence dependent on two or more tables.
3. Many-to-many relationships become associative tables with at least two foreign keys.
4. They may contain other attributes.
5. The foreign key identifies each associated table.
6. There are three options for the primary key: (a) Use a composite of foreign keys of associated tables if unique. (b) Use a composite of foreign keys and a qualifying column, or (c) Create a new simple primary key

Weak Entities. An entity is considered **weak** if it cannot be "distinguished" from other entities by virtue of its OWN attributes. Instead, weak entities must include the attributes of other entities (as foreign keys) to individuate them. In practice, this means that BOTH (1) the entity in question is existent-dependent on the parent entity (and cannot exist without it) AND (2) the primary key is derived from the primary key of the parent entity.

- **Example 1:** In a COMPANY database, Spouse is a weak entity because its primary key is dependent on the Employee table. Without a corresponding employee record, the spouse record would not exist. There can be no "employee spouse" without an employee!
- **Example 2:** A person's bank account may be identified by an AccountNumber, but this number is meaningless without being associated with a Bank's routing number.

CREATING DIAGRAMS OF ENTITY SETS

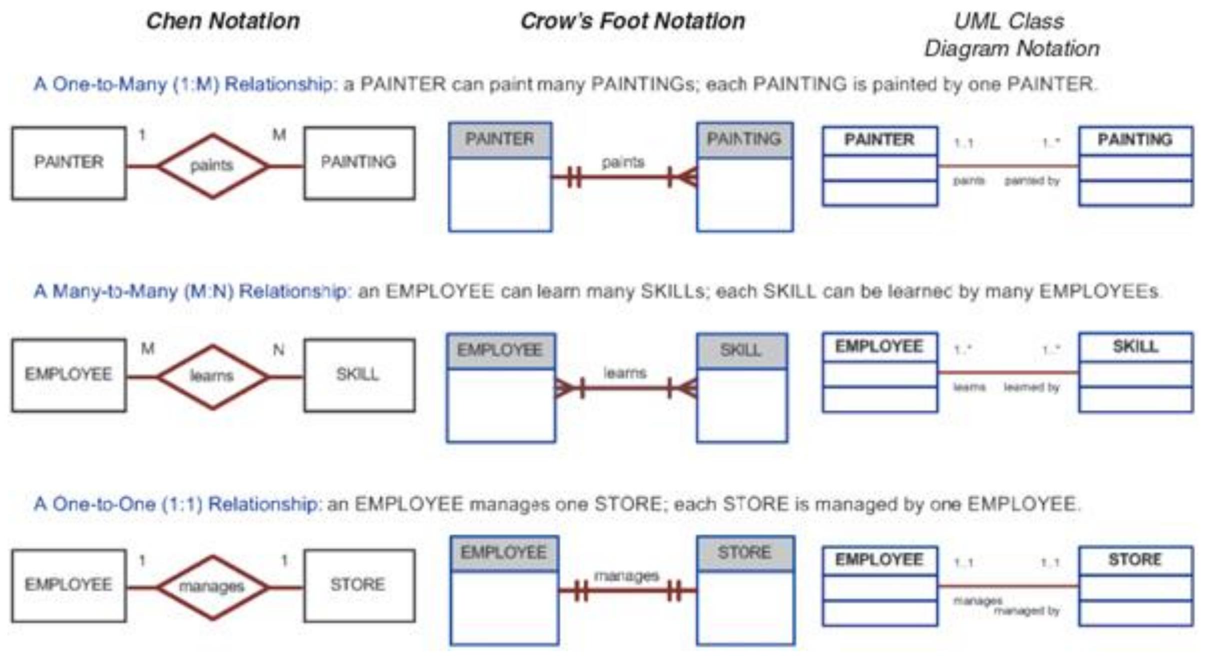
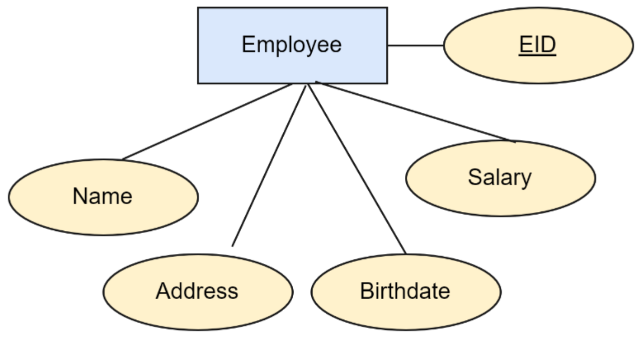
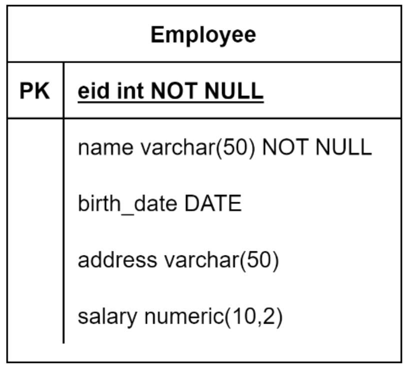

Chen Notation
Crow's Foot Notation
UML Class Diagram Notation
A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGs; each PAINTING is painted by one PAINTER.
PAINTER
1
paints
M
PAINTING
PAINTER
paints
PAINTING
PAINTER
1..1
1..*
PAINTING
paints
painted by
A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLs; each SKILL can be learned by many EMPLOYEEs.
EMPLOYEE
M
learns
N
SKILL
EMPLOYEE
learns
SKILL
EMPLOYEE
1..*
1..*
SKILL
learns
learned by
A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.
EMPLOYEE
1
manages
1
STORE
EMPLOYEE
manages
STORE
EMPLOYEE
1..1
1..1
STORE
manages
managed by

Figure 1 From Database Systems: Design, Implementation, & Management

An *entity set* is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. The figure below shows two common styles of E-R diagram—**Chen** (used mainly for "concept-level" modeling) and **Crow's Foot** (used for "implementation-level" modeling)**.** Another common style of modeling is called **Unified Modeling**

Language, which can be used for both conceptual AND implementation level approaches. For the remainder of this lesson, we'll generally be using Chen (the book often uses Crow Foot). I've made all diagrams using http://diagrams.net.

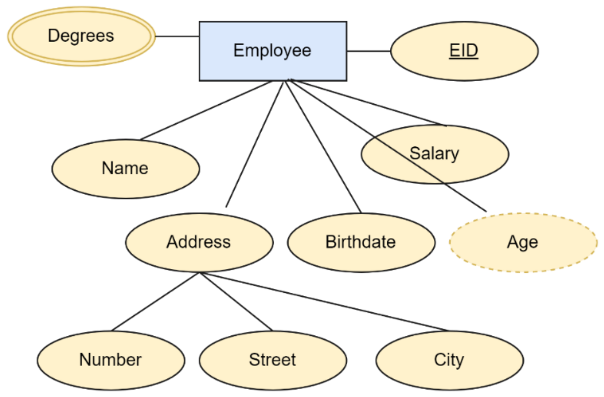
Chen Style: Box-y Entities and Oval Attributes	**Crow-Foot Style: Large entity box with attribute lines**
	 Employee PK eid int NOT NULL name varchar(50) NOT NULL birth_date DATE address varchar(50) salary numeric(10,2)

As you can see, each entity is described by a set of **attributes** (e.g., the entity Employee has attributes such as Name, Address, Birthdate (Age), and Salary. Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the Chen-style ERD.

ATTRIBUTES

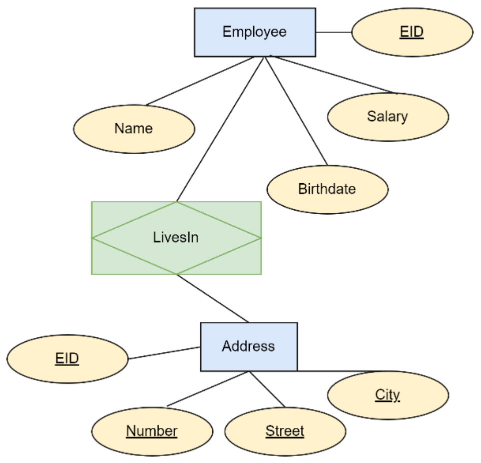
Types of Attributes. There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

- **Simple attributes** are those drawn from the atomic value domains; they are also called *single-valued attributes*. In the COMPANY database, an example of this would be: Name = {John} or Age = {23}. Each attribute can only take values in the permitted **range.**
- **Composite attributes** are those that consist of a hierarchy of attributes. Using our database example, and shown in the figure below Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}
- attribute from the COMPANY database, as seen the figure above, are the degrees of an employee, which might include MULTIPLE entries, such as AS, BS, MS, or PhD.
- **Derived attributes** are attributes that contain values calculated from other attributes. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a **stored attribute**, which is physically saved to the database. The decision to create stored attributes depends, in large part, on how often we think we'll be retrieving them (more retrieval -> better idea to store) and how often we'll be updating (more updating -> worse idea to store).
- **Multivalued attributes** can take on multiple values. So, for example, the Degrees attribute of Employee might contain multiple values (if the employee has an Associates, Bachelors, and Masters, for example):

In this diagram: 1. Name, Salary, and EID are simple attributes. They are represented by **ovals.** 2. EID is the primary key for Employee. It is **underline.** 3. Age is a derived attribute, since we can calculate it from birthday. This is captured by a **dotted oval.** 4. Address is a composite attribute, which has "children" attributes of its own. 5. Degree is a multi-valued attribute. This is represented by a **double-lined oval.**	 Degrees Employee EID Name Salary Address Birthdate Age Number Street City

CHARACTERISTIC ENTITIES

Characteristic entities provide more information about another entity type. These entities have the following characteristics:



1. They describe other entities. For example, we would like to describe the entity Employee with a characteristic entity Address. Previously, we represented this as an "attribute," but now we want to think about what it would be like to represent this an enity.
2. They represent multivalued attributes (and thus represent **composite attributes** – see address above**)**. So, an address contains multiple attributes (street, number, city). It would be pointless to have a single-value attribute as a "characteristic entity" (since we could just include it in the original table).
3. They typically have a one-to-many relationship with the entity that they "describe." That is, each address is associated with only ONE employee, but an employee might have several associated addresses.
4. The foreign key is used to further identify the characterized entity. Here, the table for Address will need to reference the primary key of the Employee Table.
5. Options for primary key are as follows:
    a. Use a composite of foreign key plus a qualifying column. For example: Address (EID, Number, Street, Zip).
    b. Create a new simple primary key. For example: Address (AddressID, EID, Number, Street, Zip).

REVIEW QUESTIONS AND ACTVITIES

Question 1: Consider the database tables at the right, which represents Plays and Directors of Plays:

a. Identify all CANDIDATE keys.
b. Identify any SUPER keys. Which one is the PRIMARY key?
c. Identify the foreign key in the PLAY table.
d. Does the PLAY table exhibit referential integrity? Why or why not?
e. Draw an ER Diagram of each of the two tables (don't worry about connecting them yet).

DIRECTOR

DIRNUM	DIRNAME	DIRDOB
100	J.Broadway	01/08/39
101	J.Namath	11/12/48
102	W.Blake	06/15/44

PLAY

PLAYNO	PLAYNAME	DIRNUM
1001	Cat on a cold bare roof	102
1002	Hold the mayo, pass the bread	101
1003	I never promised you coffee	102
1004	Silly putty goes to Texas	100
1005	See no sound, hear no sight	101
1006	Starstruck in Biloxi	102
1007	Stranger in parrot ice	101

Question 2: Consider the tables at the right, represent Trucks, along with their home bases, and type of truck.

a. Identify the primary and foreign key(s) for each table.
b. Does the TRUCK table exhibit entity and referential integrity? Why or why not? Explain your answer.
c. What kind of relationship exists between the TRUCK and BASE tables?
d. How many entities does the TRUCK table contain ?
e. Identify the TRUCK table candidate key(s).
f. Draw an ER Diagram of the three tables. Don't worry about connecting them yet.

TRUCK

TNUM	BASENUM	TYPENUM	TMILES	TBOUGHT	TSERIAL
1001	501	1	5900.2	11/08/90	aa-125
1002	502	2	64523.9	11/08/90	ac-213
1003	501	2	32116.0	09/29/91	ac-215
1004		2	3256.9	01/14/92	ac-315

BASE

BASENUM	BASECITY	BASESTATE	BASEPHON	BASEMGR
501	Dallas	TX	893-9870	J. Jones
502	New York	NY	234-7689	K. Lee

TYPE

TYPENUM	TYPEDESC
1	single box, double axle
2	tandem trailer, single axle

[i] This set of lecture notes is adapted, with substantial modification, from Adrienne Watt, "Chapter 8 The Entity Relationship Data Model," October 24, 2014, https://opentextbc.ca/dbdesign01/chapter/chapter-8-entity-relationship-model/. It is licensed under a Creative Commons Attribution License 3.0 license. Any mistakes, are of course, my own!