

Lecture 4: Entity-Relationship Model

Database and SQL: Course Notes | Brendan Shea, Ph.D. (Brendan.Shea@rctc.edu)

1 CONTENTS

2	An introduction To the Entity-Relationship Model	2
2.1	The Entity Relationship Data Model	2
2.2	Business Rules for a “Company” Database	3
2.3	Kinds of Entities.....	3
2.4	Creating Diagrams of Entity Sets.....	4
2.5	Attributes	5
2.6	Characteristic entities	6
2.7	Review Questions and Activities.....	6
3	Representing Relationships on E-R Diagrams	7
3.1	Relationship strength	7
3.2	relationship connectivity and Cardinality.....	8
3.3	RElationship degree	10
3.4	Review Questions and Activities.....	11
4	Optional Reading: Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned (by Peter Chen).....	12
4.1	Introduction.....	12
4.2	Historical Background	13
4.2.1	Competing Forces	13
4.2.2	Needs of the System Software in the Early 70’s.....	14
4.3	Fulfilling the Needs	14
4.4	Initial Reactions & Reactions in the First Five Years (1976 – 1981)	16
4.5	The Next Twenty Years (’81 –’01).....	17
4.5.1	ER Model Adopted as a Standard for Repository Systems and ANSI IRDS.	17
4.5.2	ER Model as a Driving Force for Computer-Aided Software Engineering (CASE) tools and Industry	17
4.5.3	Object-Oriented (OO) Analysis Techniques are Partially Based on the ER Concepts	17
4.5.4	Data Mining is a Way to Discover Hidden Relationships	17
4.6	In Retrospect: Another Important Factor – Chinese Culture Heritage	18
4.6.1	Chinese Culture Heritage	18
4.6.2	Ancient Egyptian Hieroglyphs	18
4.7	Lesson Learned.....	19

4.7.1	Reflections on Career choices	19
4.7.2	Implications of the Similarity and differences between the Chinese Characters and Ancient Egyptian Hieroglyphs on Software Engineering and Systems Development Methodologies	19
4.8	Conclusions	20

2 AN INTRODUCTION TO THE ENTITY-RELATIONSHIP MODELⁱ

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

1. What are the basic concepts of the **entity-relationship model (ERM)**?
2. How can **entity-relationship 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?

2.1 THE ENTITY RELATIONSHIP DATA MODEL

The **entity-relationship model (ERM)** has been used for almost 50 years. It is well suited to data modeling 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 an excellent 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. Still, they 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 modeling is based on two concepts:

- **Entities** correspond to different types of objects about which we have gathered data. These correspond to 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 and their attributes for a relational database). The ERD visualizes a database's main entities, attributes, and relationships. This allows database designers and the proposed data dictionary entries. The completed ER diagram also lets the designer communicate more precisely with those who commissioned the database design to ensure there are no errors or omissions. Finally, the completed ER diagram serves as the implementation guide for those who create the actual database.

2.2 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 code, a name, a 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, unique id number, and 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 of 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.

2.3 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 **entity types**: 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. Other tables are based on these. *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 are uniquely identified by their social security numbers, and hence are “strong” entities (that don’t depend on other entities to distinguish them). Examples of independent entities in the COMPANY database include the Customer table, Employee table or Product table.

Existent-dependent (or derived) entities depend on other entities 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 **weak** if it cannot be “distinguished” from other entities by virtue of its OWN attributes. Instead, weak entities must include other entities' attributes (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. There can be no “employee spouse” without an employee! Without a corresponding employee record, the spouse record would not exist.
- **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.

2.4 CREATING DIAGRAMS OF ENTITY SETS

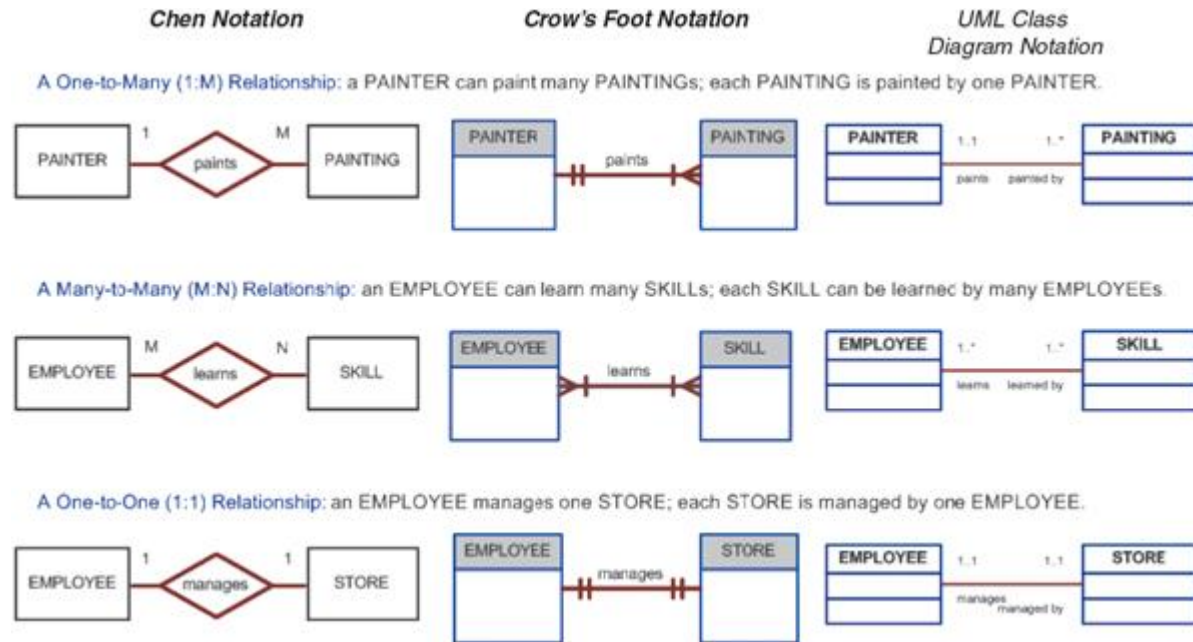
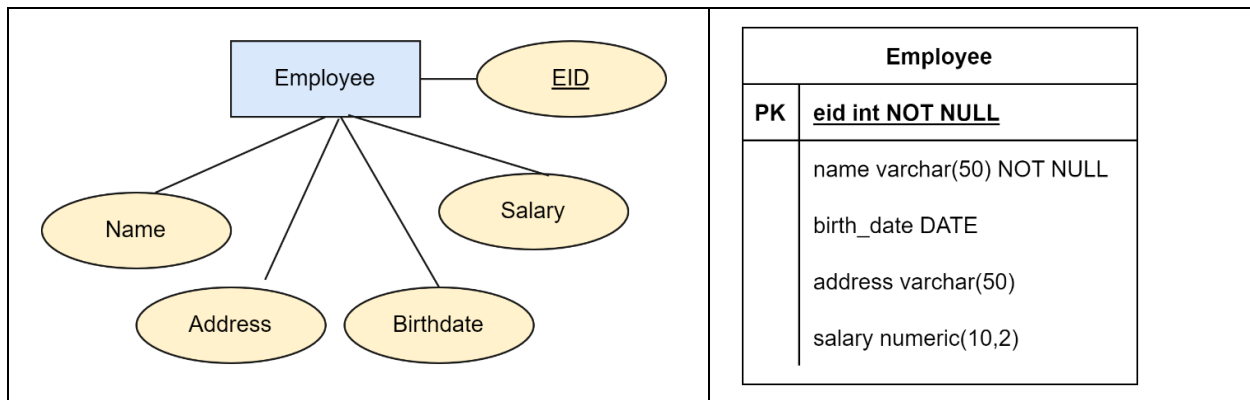


Figure 1 From Database Systems: Design, Implementation, & Management

An *entity set* is a collection of entities of an entity type at a particular time. An entity-relationship diagram (ERD) represents an entity type by a name in a box. The figure below shows two standard styles of E-R diagram—**Chen** (used mainly for “concept-level” modeling) and **Crow’s Foot** (used for “implementation-level” modeling). Another common modeling style is **Unified Modeling Language (UML)**, which can be used for both conceptual AND implementation level approaches. We’ll generally use the Chen style for the remainder of this lesson. 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
---	---

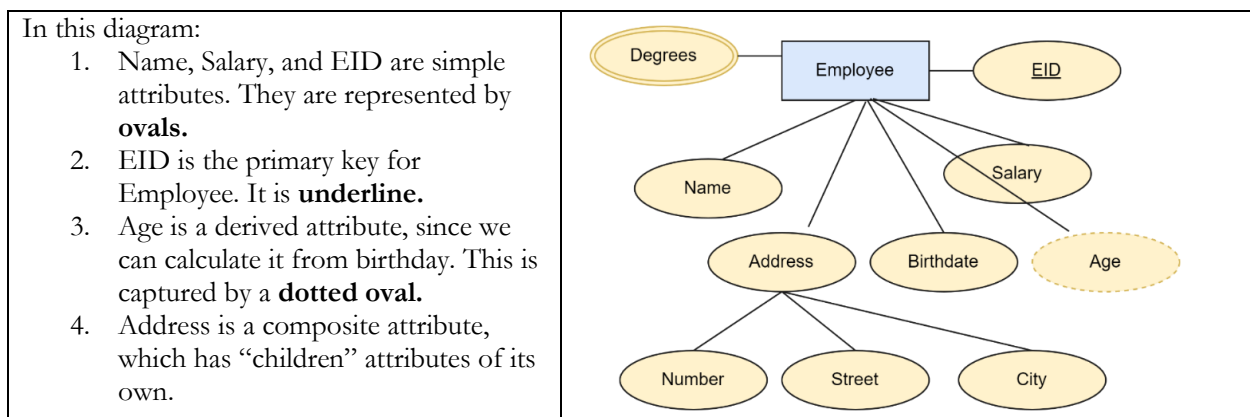


As you can see, each entity is described by a set of **attributes** (e.g., the entity Employee has attributes such as Name, Address, Birthdate (Age), and Salary). Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain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

2.5 ATTRIBUTES

Types of Attributes. There are a few types of attributes you need to be familiar with. Some are OK 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, we will discuss fixing the attributes to fit correctly into the relational model.

- **Simple (or “single-valued”) attributes** are those drawn from the atomic value domains. In the COMPANY database, an example of this would be: Name = {John} or Age = {23}. Each attribute can only take values in the acceptable **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 frequently 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):

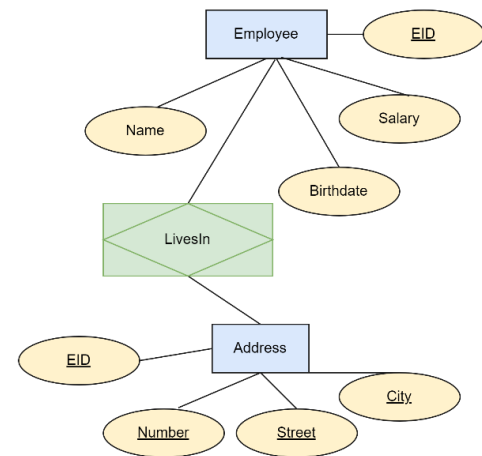


5. Degree is a multivalued attribute. This is represented by a **double-lined oval**.

2.6 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 might decide to describe the entity Employee with a characteristic entity Address (instead of treating Address as an attribute, as we've done so far).
2. They are nearly always used to represent multivalued (or **composite**) attributes. Again, Address (which contains information on street, number, city, etc.) provides an excellent example.
3. They typically have a one-to-many relationship with the entity they "describe." Each Address is associated with only ONE Employee, but an Employee might have several associated Addresses.
4. The foreign key is used to identify the characteristic entity. Here, the table for Address will need to reference the primary key of the Employee Table.
5. Options for the primary key for characteristic entities are as follows:
 - a. Use a composite of foreign key plus a qualifying column from the characteristic entity. For example: Address (EID, Number, Street, Zip).
 - b. Create a new simple primary key for the characteristic entity. For example: Address (AddressID, EID, Number, Street, Zip).



2.7 REVIEW QUESTIONS AND ACTIVITIES

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.

- Identify the primary and foreign key(s) for each table.
- Does the TRUCK table exhibit entity and referential integrity? Why or why not? Explain your answer.
- What kind of relationship exists between the TRUCK and BASE tables?
- How many entities does the TRUCK table contain ?
- Identify the TRUCK table candidate key(s).
- 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

3 REPRESENTING RELATIONSHIPS ON E-R DIAGRAMS

In this lesson, we'll be talking about the following:

- What is the difference between a **strong** and **weak** relationship?
- What are **participation** and **cardinality** constraints?
- How can **M:N** (many-to-many) relationships be converted into 1:M (one-to-many) relationships?
- What is the difference between **unary**, **binary**, and **ternary relationships**?

Relationships are the glue that holds the tables together. They are used to connect related information between tables. Relationships can be either strong or weak. They are further classified based on their **participants** (which entities are related?), their **connectivity** (is the relationship one-to-many, one-to-one, or many-to-one) and optionally their **cardinality** (a measure of the precise minimum and maximum number of entities on "each side" of the relationship). Finally, they may be classified by **degree**, or the number of entity types that are connected by the relationship.

3.1 RELATIONSHIP STRENGTH

Relationship strength is based on how the primary key of a related entity is defined.

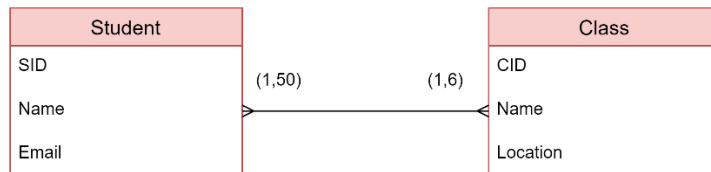
- A **weak (or non-identifying) relationship** exists if the primary key of the related entity does NOT contain a primary key component of the parent entity. In the Company database, for example, the Order is only "weakly" identified by Customer (since the CustID is not part of the primary key for Order):
 - Customer(CustID, CustName)
 - Order(OrdID, CustID, Date)
- A **strong (or identifying) relationship** exists when the primary key of the related entity contains the primary key component of the parent entity. So, for example, the Class would be strongly identified by Course:

- a. Course(CrsC ode, DeptCode, Description)
- b. Class(CrsCode, Section, ClassTime...)

3.2 RELATIONSHIP CONNECTIVITY AND CARDINALITY

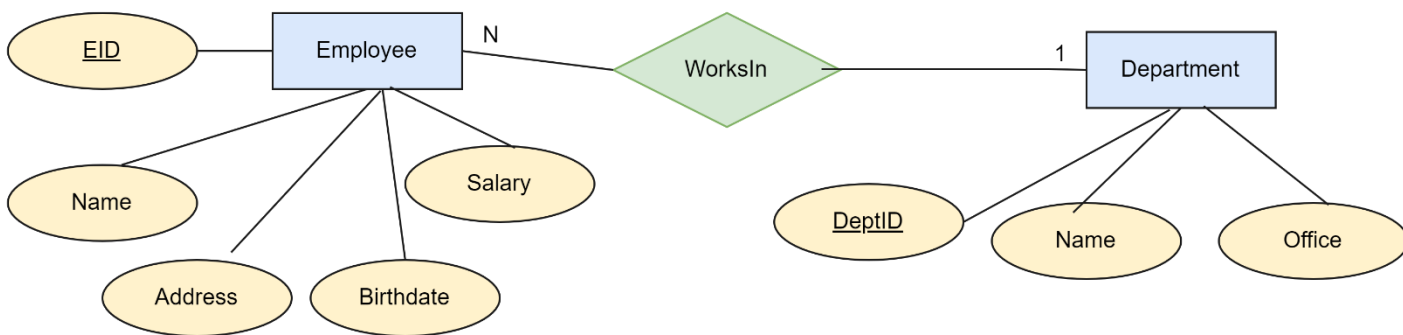
The **cardinality** of a relationship measures “how many” entities are on either side of the relationship. So, for example, if each Student must take between 1 and 6 courses, but a course may enroll between 1 and 50 students. So we have the following in Crow’s Foot notation:

Cardinality: Each class has between 1 and 50 students. Each student has between 1 and 6 classes.
Connectivity: There is many-to-many (M:N) relationship between student and class.



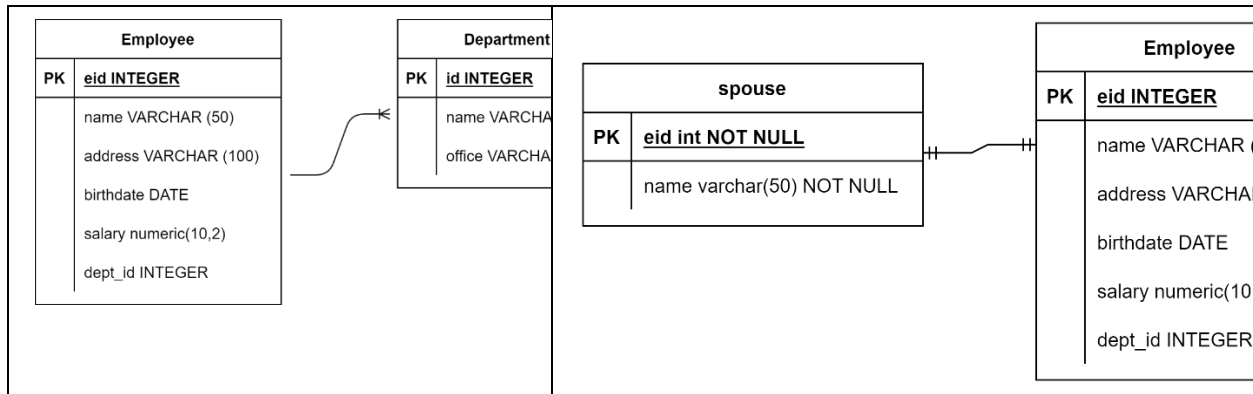
Database designers frequently exclude cardinalities from Chen and Crow’s Foot diagrams in favor of the more general **connectivity** (which is one of three values: 1:1, 1:M, or M:N). This simplification is acceptable at the E-R modeling stages because the differences between these three “types” of relationships are far more important than the individual “numbers.” With this in mind, we’ll generally be leaving the discussion of cardinality aside. However, it’s worth remembering that E-R diagrams don’t always allow us to “see” all of the cardinality constraints that will make their way into a completed relational database.

A **one-to-many (1:M or 1:N)** relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. The figure below shows the relationship of one of these employees to the department.



A **one-to-one (1:1)** relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table. An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

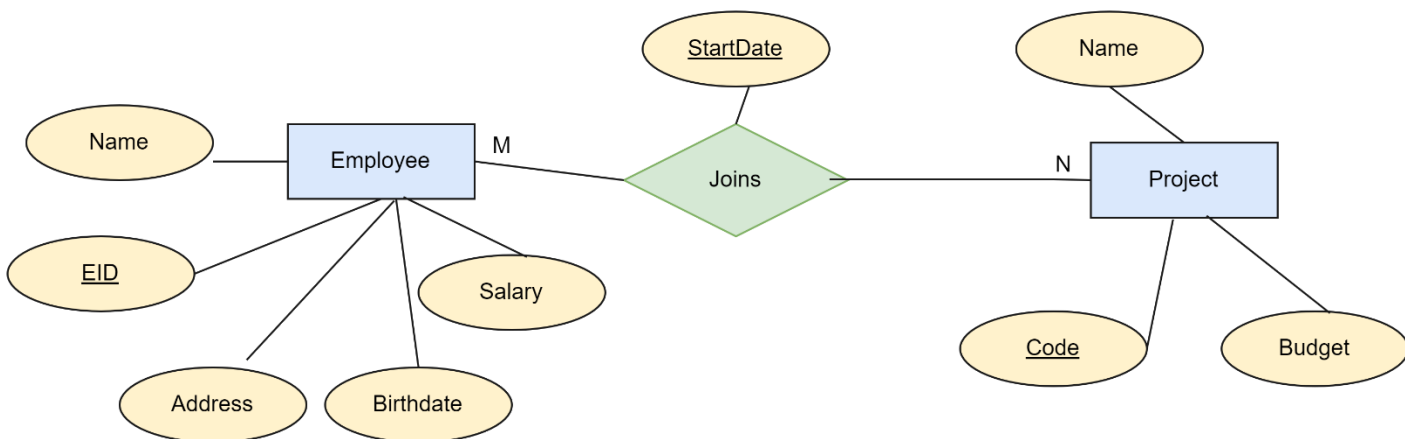
Crow’s Foot: One-to-Many Relationship	Crow’s Foot: One-to-One
---------------------------------------	-------------------------



Many-to-many (M:N) relationships can be represented within the ER model, but can NOT be implemented within relational databases. So, whenever you encounter these sorts of relationships. Instead, you'll want to do the following:

1. It can be changed into two 1:M relationships using **bridge entities**.
2. The bridge entity table (the **linking table**) must contain at least the primary keys of the original tables.
3. The linking table contains multiple occurrences of the foreign key values.
4. Additional attributes may be assigned as needed.
5. Creating a composite entity or bridge entity can avoid problems inherent in an M:N relationship. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

For example, in the following figure shows a many-to-many (written as **M:N**) relationship between Employees and Projects. One employee may be involved in many projects, and each project may have many employees. Moreover, one attribute (StartDate) is associated with the *relationship itself*, reflecting the date that each employee joined each project.



If we turned this into a relational database, we could use the following relational schema:

- EMPLOYEE(EID, Name, Address, Birthdate, Salary)
- PROJECT(Code, Name, Budget)
- JOIN(EID, Code, StartDate)

Converting M:N relationships to two 1:M relationships. To convert you need to the following:

1. For each M:N between entity types A and B, identify a relationship R that can be used to characterize this relationship.
2. Create a new relation (table) S to represent the relationship R.
3. Table S needs to contain the PKs of BOTH A and B. These together can be the PK in the S table OR these combined with another simple attribute in the new table S can be the PK. (So, in the above example, Join contains the primary keys of both employee and project AND an additional simple attribute).

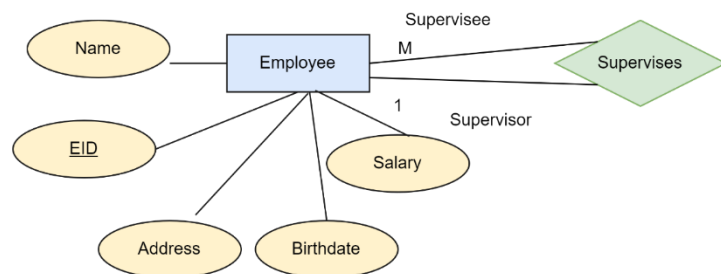
3.3 RELATIONSHIP DEGREE

Most of the relationships we have talked about so far have been **binary** relationships, which relate exactly two different types of entities. These are the “norm” with both ERM and the Relational Model. However, there are relationships with different **degrees** that relate different numbers of entity types.

A **unary relationship**, also called **recursive**, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.

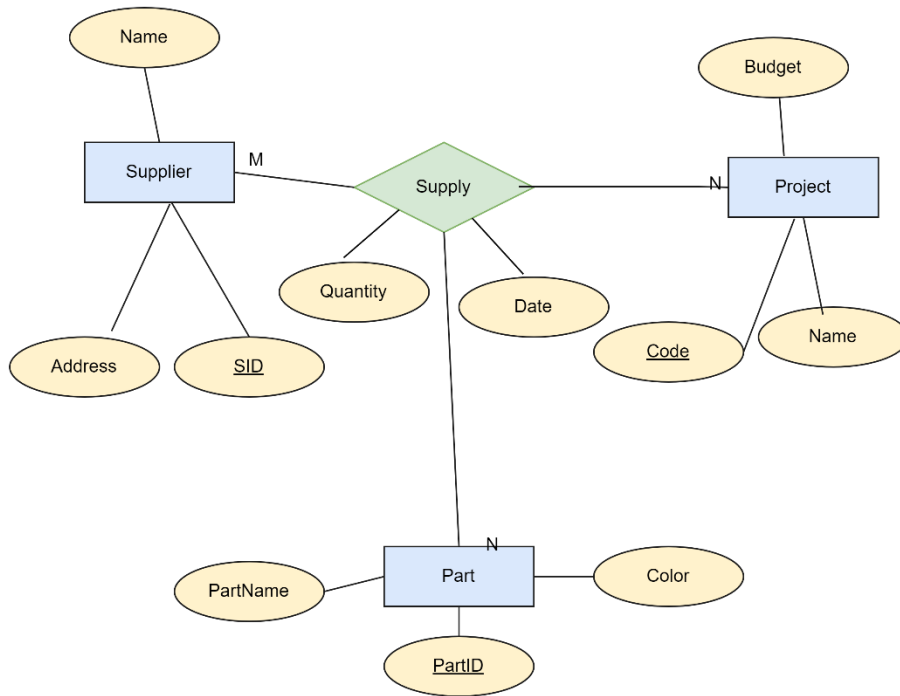
The relational scheme here is (not that each employee may have one supervisor, but that a supervisor may supervise many employees. We capture this by encoding the EID of the supervisor on the “many” side of the relationship, which in this case ends up being the entries representing the supervisees.).

- EMPLOYEE(EID, Name, Address, Birthdate, Salary, Supervisor-EID)



A **ternary relationship** is a relationship type that involves many to many relationships between three tables. It is an example of **n-ary relationship** where $n=3$ (here n = the number of different entity types that are related). The following figure provides an example of mapping a ternary relationship type. Note *n-ary*)

- For each n -ary (> 2) relationship, create a new relation to represent the relationship.
- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an n -ary relationship, all the participating entities hold a many-to-many relationship with one another.



The relational schema (which reflects the fact we need to create a SUPPLY table) is as follows:

- SUPPLIER (SID, Name, Address)
- PROJECT (Code, Name, Budget)
- PART (PartID, PartName, Color)
- SUPPLY (SID, Code, PartID, Quantity, Date)

3.4 REVIEW QUESTIONS AND ACTIVITIES

Problem 1. A manufacturing company produces products. The following product information is stored: product name, product ID and quantity on hand. These products are made up of many components. One or more suppliers can supply each component. The following component information is kept: component ID, name, description, suppliers who supply them, and products in which they are used. The business rules are as follows:

1. A supplier can exist without providing components.
2. A component does not have to be associated with a supplier.
3. A component does not have to be associated with a product. Not all components are used in products.
4. A product cannot exist without components.

Create an ERD to show how you would track this information. Show entity names, primary keys, attributes for each entity, relationships between the entities and cardinality.

Problem 2 (A challenge!). Create an ERD for a car dealership. The dealership sells new and used cars and operates a service facility. Base your design on the following business rules:

1. A salesperson may sell many cars, but only one salesperson sells each car.
2. A customer may buy many cars, but only one customer purchases each car.

3. A salesperson writes a single invoice for each car they sell.
4. A customer gets an invoice for each car they buy.
5. A customer may come in just to have their car serviced; that is, a customer need not buy a car to be classified as a customer.
6. When a customer takes one or more cars in for repair or service, one service ticket is written for each car.
7. The car dealership maintains a service history for each of the cars serviced. The service records reference the car's serial number.
8. Many mechanics can work on a car brought in for service, and each mechanic may work on many cars.
9. A car that is serviced may or may not need parts (e.g., adjusting a carburetor or cleaning a fuel injector nozzle does not require providing new parts).

Create an ERD to show how you would track this information. Show entity names, primary keys, attributes for each entity, relationships between the entities, and cardinality.

4 OPTIONAL READING: ENTITY-RELATIONSHIP MODELING: HISTORICAL EVENTS, FUTURE TRENDS, AND LESSONS LEARNED (BY PETER CHEN)¹

[Brendan's Note: The author of this piece—Peter Chen—is the original creator of the ER model. Here, he explains why/how he created the model. As he notes, there was initially quite a bit of debate over which “model” (relational or ER) was correct/best. Eventually, people figured out how to integrate the two models together, but this took a while.]



Abstract. This paper describes the historical developments of the ER model from the 70's to recent years. It starts with a discussion of the motivations and the environmental factors in the early days. Then, the paper points out the role of the ER model in the Computer-Aided Software Engineering (CASE) movement in the late 80's and early 90's. It also describes the possibility of the role of author's Chinese cultural heritage in the development of the ER model. In that context, the relationships between natural languages (including Ancient Egyptian hieroglyphs) and ER concepts are explored. Finally, the lessons learned and future directions are presented.

4.1 INTRODUCTION

Entity–Relationship (ER) modeling is an important step in information system design and software engineering. In this paper, we will describe not only the history of the development of the ER approach but also the reactions and new developments since then. In this perspective, this paper may be a little bit different from some other papers in this volume because we are not just talking about historical events that happened

¹ Peter Chen, “Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned,” in *Software Pioneers* (Springer, 2002), 296–310.

twenty or thirty years ago, we will also talk about the consequences and relevant developments in the past twenty-five years. At the end, we will talk about lessons learned during this time period. In particular, we intend to show that it is possible that one concept such as the ER concept can be applied to many different things across a long time horizon (for more than twenty-five years) in this fast-changing Information Technology area.

This paper is divided into 8 sections. Section 1 is the Introduction. In Section 2, the historical background and events happened around twenty-five years ago will be explained. For example, what happened at that time, what the competing forces were, and what triggered researchers like the author to work on this topic will be explained. Section 3 describes the initial reactions in the first five years from 1976 to 1981. For example, what the academic world and the industry viewed the ER model initially? Section 4 states the developments in the next twenty years from 1981 to 2001. In particular, the role of the ER model in the Computer-Aided Software Engineering (CASE) will be discussed. Section 5 describes a possible reason for the author to come up with the ER modeling idea, that is, the author's Chinese culture heritage. The author did not think about this particular reason until about fifteen years ago. Section 6 presents our view of the future of ER modeling. Section 7 states the lessons learned. For those of you who have similar experience in the past twenty-five years, you probably have recognized similar principles and lessons in this section. For those who just started their professional careers recently, we hope the lessons learned by the author will be helpful to those readers. Section 8 is the conclusion.

4.2 HISTORICAL BACKGROUND

In this section, we will look at the competing forces, the needs of the computer industry at that time, how the ER model was developed, and the main differences between the ER model and the relational model.

4.2.1 Competing Forces

First, Let us look at the competing forces in the computer software area at that time. What are the competing forces then? What triggered people like the author to work on this area (data models) and this particular topic (ER modeling)? In the following, we will discuss the competing forces in the industry and in the academic world in the early 70's,

Competing Forces in the industry. There were several competing data models that had been implemented as commercial products in the early 70's: the file system model, the hierarchical model (such as IBM's IMS database system), and the Network model (such as Honeywell's IDS database system). The Network model, also known as the CODASYL model, was developed by Charles Bachman, who received the ACM Turing Award in 1973. Most organizations at that time used file systems, and not too many used database systems. Some people were working on developing better data or index structures for storing and retrieving data such as the B+-tree by Bayer and McCreight [1].

Competing Forces in the Academic World. In 1970, the relational model was proposed, and it generated considerable interest in the academic community. It is correct to say that in the early 70's, most people in the academic world worked on relational model instead of other models. One of the main reasons is that many professors had a difficult time to understand the long and dry manuals of commercial database management systems, and Codd's relational model paper [2] was written in a much more concise and scientific style. For his contributions in the development of the relational model, Codd received ACM Turing Award in 1981.

Most People were working on DBMS Prototypes. Many people at that time in the academic world or in the industry worked on the implementation of database management system prototypes. Most of them were based on the relational model.

Most Academic People were investigating the definitions and algorithms for the Normal Forms of Relations. A lot of academic people worked on normalization of relations because only mathematical skills were needed to work on this subject. They could work on the improvement of existing algorithms for welldefined normal forms. Or, they could work on new normal forms. The speed of research moved very fast in the development of normal forms and can be illustrated by the following scenario. Let us say that several people were ready to publish their results on normal forms. Assuming that one person published a paper on 4th normal form and another person who had written a paper on 4th normal form but had not published it yet, the 2nd person would have changed the title of the paper from 4th normal form to 5th normal form. Then, the rest would work on the 6th normal form. This became an endless game till one day somebody wrote a paper claiming that he had an infinity-th normal form and arguing that it did not make any sense to continue this game. Most practitioners also said loudly that any relational normal form higher than 3rd or 4th won't have practical significance. As a result, the game of pursuing the next normal form finally ran out of steams.

4.2.2 Needs of the System Software in the Early 70's

The Needs of the Hardware/Software Vendors. In terms of software vendors at that time, there were urgent needs for (1) integration of various file and database formats and (2) incorporating more “data semantics” into the data models.

The Needs of the User Organizations. For user organizations such as General Motors and Citibank, there were urgent needs for (1) a unified methodology for file and database design for various file and database system available in the commercial market and (2) incorporation of more data semantics including business rules into the requirements and design specifications.

[Brendan's Note: Material cut here.]

4.3 FULFILLING THE NEEDS

How did the ER model fulfill the needs of the vendor and user organizations at that time? We will first start with the graphical representation and theoretical foundations of the ER model. Then, we will explain the significant differences between the ER model and the relational model.

The Concepts of Entity, Relationship, Types, and Roles. In Fig. 1, there are two entities; both of them are of the “Person” type. There is a relationship called, “is-married-to,” between these two persons. In this relationship, each of these two Person entities has a role. One person plays the role of “husband,” and another person plays the role of “wife.”

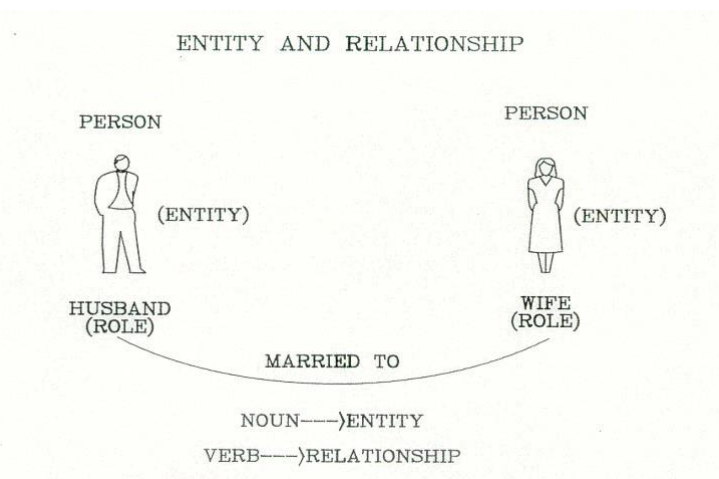


Fig. 1. The Concept of Entity and Relationship

The Entity-Relationship (ER) Diagram. One of the key techniques in ER modeling is to document the entity and relationship types in a graphical form called, Entity-Relationship (ER) diagram. Figure 2 is a typical ER diagram. The entity types such as EMP and PROJ are depicted as rectangular boxes, and the relationship types such as WORK-FOR are depicted as a diamond-shaped box. The value sets (domains) such as EMP#, NAME, and PHONE are depicted as circles, while attributes are the “mappings” from entity and relationships types to the value sets. The cardinality information of relationship is also expressed. For example, the “1” or “N” on the lines between the entity types and relationship types indicated the upper limit of the entities of that entity type participating in that relationships.

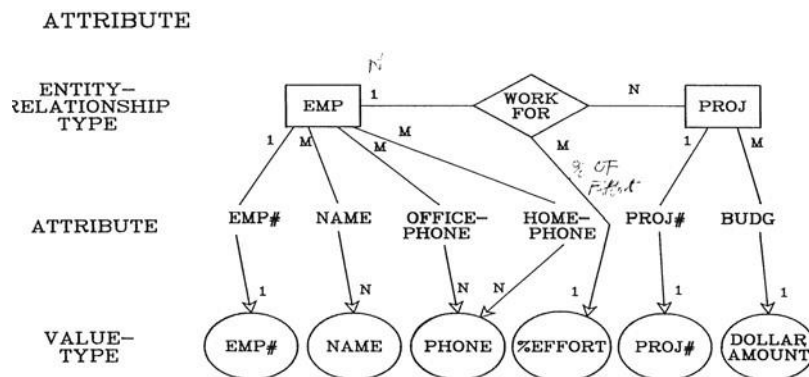


Fig. 2. An Entity-Relationship (ER) Diagram

ER Model is based on Strong Mathematical Foundations. The ER model is based on (1) Set Theory, (2) Mathematical Relations, (3) Modern Algebra, (4) Logic, and (5) Lattice Theory. A formal definition of the entity and relationship concepts can be found in Fig. 3.

SET THEORY (DEFINITIONS)

ENTITY	e
ENTITY SET	$E; e \in E$
VALUE	v
VALUE SET	$V; v \in V$
RELATIONSHIP	r
RELATIONSHIP SET	$R; r \in R$

A RELATIONSHIP SET IS DEFINED AS A
"MATHAMATICAL RELATION" ON ENTITY SETS

$$R = \{r_1, r_2, \dots, r_n\}$$

$$r_i = [e_{i1}, e_{i2}, \dots, e_{in}] | e_{i1} \in E_1, \dots, e_{in} \in E_n$$

Fig. 3. Formal Definitions of Entity and Relationship Concepts

Significant Differences between the ER model and the Relational Model. There are several differences between the ER model and the Relational Model:

ER Model uses the Mathematical Relation Construct to Express the Relationships between Entities. The relational model and the ER model both use the mathematical structure called Cartesian product. In some way, both models look the same – both use the mathematical structure that utilizes the Cartesian product of something. As can be seen in Figure 3, a relationship in the ER model is defined as an ordered tuple of “entities.” In the relational model, a Cartesian product of data “domains” is a “relation,” while in the ER model a Cartesian product of “entities” is a “relationships.” In other words, in the relational model the mathematical relation construct is used to express the “structure of data values,” while in the ER model the same construct is used to express the “structure of entities.”

ER Model Contains More Semantic Information than the Relational Model. By the original definition of relation by Codd, any table is a relation. There is very little in the semantics of what a relation is or should be. The ER model adds the semantics of data to a data structure. Several years later, Codd developed a data model called RM/T, which incorporated some of the concepts of the ER model.

ER Model has Explicit Linkage between Entities. As can be seen in Figures 2 and 4, the linkage between entities is explicit in the ER model while in the relational model is implicit. In addition, the cardinality information is explicit in the ER model, and some of the cardinality information is not captured in the relational model.

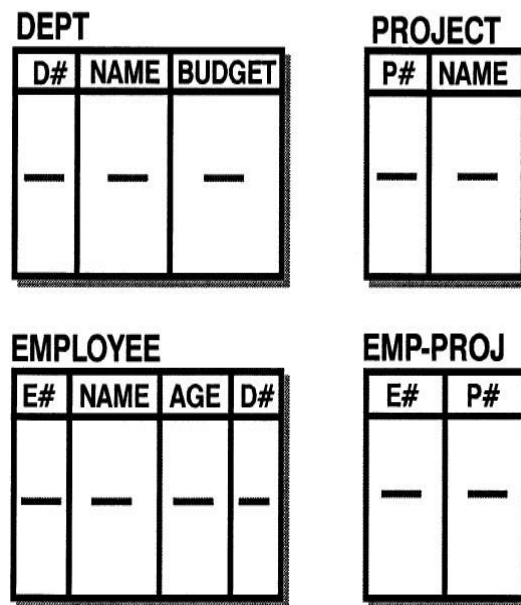


Fig. 4. Relational Model of Data

4.4 INITIAL REACTIONS & REACTIONS IN THE FIRST FIVE YEARS (1976 – 1981)

First Paper Published & Codd's Reactions. As stated before, the first ER model paper was published in 1976. Codd wrote a long letter to the editor of ACM Transaction on Database Systems criticizing the author's paper. The author was not privileged to see the letter. The editor of the Journal told the author that the letter was very long and single-spacing. In any case, Dr. Codd was not pleased with the ER model paper. Ironically, several years later, Codd proposed a new version of the relational data model called RM/T, which incorporated some concepts of the ER model. Perhaps, the first paper on the ER model was not as bad as Codd initially thought. Furthermore, in the

90's, the Codd and Date consulting group invited the author to serve as a keynote speaker (together with Codd) several times in their database symposia in London. This indicates that the acceptance of ER model was so wide spread so that initial unbelievers either became convinced or found it difficult to ignore.

Other Initial Reactions and Advices. During that time, there was a "religious war" between different camps of data models. In particular, there was a big debate between the supporters of the Relational model and that of the Network model. Suddenly, a young assistant professor wrote a paper talking about a "unified data model." In some sense, the author was a "new kid on the block" being thrown into the middle of a battle between two giants. The advice the author got at that time was: "why don't you do the research on the n-th normal form like most other researchers do? It would be much easier to get your normal form papers published." That was an example of the type of advices the author got at that time. Even though those

advice was based on good intentions and wisdom, the author did not follow that type of advice because he believed that he could make a more significant contribution to the field by continuing working on this topic (for example, [4-13]). It was a tough choice for a person just starting the career. You can imagine how much problems or attacks the author had received in the first few years after publishing the first ER paper. It was a very dangerous but a very rewarding decision the author made that not only had a significant impact on the author's career but also the daily practices of many information-modeling professionals.

[Brendan: Material cut here]

4.5 THE NEXT TWENTY YEARS ('81-'01)

4.5.1 ER Model Adopted as a Standard for Repository Systems and ANSI IRDS.

In the 80's, many vendors and user organizations recognized the need for a repository system to keep track of information resources in an organization and to serve as the focal point for planning, tracking, and monitoring the changes of hardware and software in various information systems in an organization. It turned out that the ER model was a good data model for repository systems. Around 1987, ANSI adopted the ER model as the data model for Information Resource Directory Systems (IRDS) standards. Several repository systems were implemented based on the ER model including IBM's Repository Manager for DB2 and DEC's CDD+ system.

4.5.2 ER Model as a Driving Force for Computer-Aided Software Engineering (CASE) tools and Industry

Software development has been a nightmare for many years since the 50's. In the late 80's, IBM and others recognized the needs for methodologies and tools for Computer-Aided Software Engineering (CASE). IBM proposed a software development framework and repository system called, AD Cycle and the Repository Manager that used the ER model as the data model. The author was one of the leaders who actively preached the technical approach and practical applications of CASE. In 1987, Digital Consulting Inc. (DCI) in Andover, Mass., founded by Dr. George Schussel, organized the 1st Symposium on CASE in Atlanta and invited the author to be one of the two keynote speakers. To everybody's surprise, the symposium was a huge commercial success, and DCI grew from a small company to a major force in the symposium and trade show business.

4.5.3 Object-Oriented (OO) Analysis Techniques are Partially Based on the ER Concepts

It is commonly acknowledged that one major component of the object-oriented (OO) analysis techniques are based on the ER concepts. However, the "relationship" concept in the OO analysis techniques are still hierarchy-oriented and not yet equal to the general relationship concept advocated in the ER model. It is noticeable in the past few years that the OO analysis techniques are moving toward the direction of adopting a more general relationship concept.

4.5.4 Data Mining is a Way to Discover Hidden Relationships

Many of you have heard about data mining. If you think deeply about what the data mining actually does, you will see the linkage between data mining and the ER model. What is data mining? What does the data mining really is doing? In our view, it is a discovery of "hidden relationships" between data entities. The relationships exist already, and we need to discover them and then take advantage of them. This is different from conventional database design in which the database designers identify the relationships. In data mining, algorithms instead of humans are used to discover the hidden relationships.

4.6 IN RETROSPECT: ANOTHER IMPORTANT FACTOR – CHINESE CULTURE HERITAGE

4.6.1 Chinese Culture Heritage

Many people asked the author how he got the idea of the Entity-Relationship model. After he kept on getting that kind of questions, the author thought it might be related to something that many people in Western culture may not have. After some soul searching, the author thought it could be related to his Chinese culture heritage. There are some concepts in Chinese character development and evolution that are closely related to modeling of the things in the real world.

Here is an example. Figure 5 shows the Chinese characters of “sun”, “moon, and “person”. As you can see, these characters are a close resemblance of the real world entities. Initially, many of the lines in the characters are made of curves. Because it was easier to cut straight lines on oracle bones, the curves became straight lines. Therefore, the current forms of the Chinese characters are of different shapes.







<u>Original Form</u>	<u>Current Form</u>	<u>Meaning</u>
		Sun
		Moon
		Person

Fig. 5. Chinese Characters that Represent the Real-World Entities

Chinese characters also have several principles for “composition.” For example, Figure 6 shows how two characters, SUN and MOON, are composed into a new character. How do we know the meaning of the new character? Let us first think: what does sun and moon have in common? If your answer is: both reflect lights, it is not difficult to guess the meaning of the new character is “brightness.” There are other principles of composing Chinese characters [10].

$$\text{日 (sun)} + \text{月 (moon)} = \text{明 (Bright/ Brightness by light)}$$

Fig. 6. Composition of Two Chinese Characters into a New Chinese Character

What does the Chinese character construction principles have to do with ER modeling? The answer is: both Chinese characters and the ER model are trying to model the world – trying to use graphics to represent the entities in the real world. Therefore, there should be some similarities in their constructs.

4.6.2 Ancient Egyptian Hieroglyphs

Besides Chinese characters, there are other languages have graphic characters. Ancient Egyptian language is one of them. It turns out that there are several characters in ancient Egyptian characters are virtually the same as the Chinese characters. One is “sun”, another is “mouth, and the third one is “water.” It is amazing that both the Egyptian people and the Chinese people developed very similar characters even though they were thousands of miles away and had virtually no communication at that time. Ancient Egyptian Hieroglyphs also have the concept of composition. Interested readers should refer to [11].











Hieroglyph	Meaning	Hieroglyph	Meaning
(a) 	lower arm	(f) 	man
(b) 	mouth	(g) 	woman
(c) 	viper	(h) 	sun
(d) 	owl	(i) 	house
(e) 	sieve	(j) 	water

Fig. 7. Ancient Egyptian Hieroglyphs

[Brendan: Material cut here.]

4.7 LESSON LEARNED

4.7.1 Reflections on Career choices

In the past twenty-five years, the author made some tough career choices as some of the other authors in this volume did. It is the hope of the author that our experience will be useful to some other people who just started their professional careers and are making their career choices. Here are some reflections based on the author's own experience:

Right idea, right place, right time, and belief in yourself. In order to have your idea be accepted by other people, you need not only to have the right idea but also to present them at the right place and right time. You also need “persistence.” In other words, you need to believe in yourself. This is probably the most difficult part because you have to endure some unnecessary pressures and criticisms when you are persistent on your idea and try to push it forward. Hopefully, some days in the future, you will be proved to be right. At that time, you will be happy that you have persisted.

Getting Fresh Ideas from Unconventional Places. After working on a particular area for a while, you may run out of “big” ideas. You may still have some “good” ideas to get you going, but those ideas are not “earth-breaking.” At that time, you need to look for ideas in different subject areas and to talk to new people. For example, most of us are immersed in Western culture, and learning another culture may trigger new ways of thinking. Similarly, you may look into some fields outside of information technology such as Physics, Chemistry, Biology, or Architecture to find fresh ideas. By looking at the theories, techniques, and approaches used in other fields, you may get very innovative ideas to make a breakthrough in the IT field.

4.7.2 Implications of the Similarity and differences between the Chinese Characters and Ancient Egyptian Hieroglyphs on Software Engineering and Systems Development Methodologies

As we pointed out earlier, there are several Chinese characters that are almost the same as their counterparts in ancient Egyptian hieroglyphs. What does this mean? One possible answer is that human beings think alike even though there was virtually no communication between ancient Chinese people and ancient Egyptian people. It is very likely that the way to conceptualize basic things in the real world is common to most of the

racess and cultures. As was discussed earlier, the construction and developments of other characters are different in Chinese and in Ancient Egyptian Hieroglyphs. It is valid to say that the language developments were dependent on the local environment and culture. What is the implication of the similarities and differences in character developments on the development of software engineering and information system development methodologies? The answer could be: some basic concepts and guidelines of software engineering and system development methodologies can be uniformly applied to all people in the world while some other parts of the methodologies may need to be adapted to local cultures and customs.

4.8 CONCLUSIONS

The author was very fortunate to have the opportunity to meet the right people and to be given the opportunity to develop the Entity-Relationship (ER) model at the time and environment such a model was needed. The author is very grateful to many other researchers who have continued to advance the theory of the ER approach and to many software professionals who have practiced ER modeling in their daily jobs in the past twenty-five years. We believe that the concepts of entity and relationship are very fundamental concepts in software engineering and information system development. In the future, we will see new applications of these concepts in the Web and other new frontiers of the software world.

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](https://creativecommons.org/licenses/by/3.0/). Any mistakes, are of course, my own!