LE/EECS 3421 - Course Notes

Sample Version

Completed Edition:

1. The Relational Model of Data
2. The Entity/Relationship Model
3. Design Theory for Relational Databases
    a) Functional Dependencies
    b) Design of Relational Database Schemas
    c) Decomposition
    d) Third Normal Form
    e) Practice Problems
4. An Algebraic Query Language
    a) Relational Algebra Review
    b) Set Operations on Relation
    c) Projection
    d) Selection
    e) Cartesian Product
    f) Natural Join
    g) Theta Join
    h) Combining Operations to form Queries
    i) Naming and Renaming
    j) Relationships Among Operations
    k) Practice Problems
5. The Database Language SQL
    a) Simple Queries in SQL
    b) Queries Involving More than One Relation
    c) Subqueries
    d) Full-Relation Operations
    e) Database Modifications
    f) Transactions in SQL
    g) Additional Practice

Preface

This is a document that particularly pertains to reviewing and practicing the core concepts of an introductory level introduction to database management systems course. The main source of notes will be referenced from, "Database Systems: The Complete book (2nd)" with additional YouTube videos.

In addition to this, the language of choice for the course is SQL. Specifically, our framework is the open-source software, PostgreSQL.

1) The Relational Model of Data

An Overview of Data Models

What is a Data Model?

A data model is a notation for describing data or information. The description generally consists of three parts:

1. Structure of the Data
2. Operations on the Data
    a. Queries
    b. Modifications
3. Constraints on the Data

Important Data Models

1. **Relational Model (Including Object-Relational Extensions)**
    a. The main focus of this particular course.
    b. Present in all commercial database management systems.
    c. Based on storing and handling tables
    d. Operations {Relational Algebra}
2. **Semi structured-Data Model (Including XML and Related Standards)**
    a. An added feature on most relational database management systems, appearing in a number of other contexts as well.
    b. Represents trees or graphs, rather than tables or arrays.
    c. Principle Representation = XML
        i. A way to represent data by hierarchically nested tagged elements.
    d. Operations {Elementary Tree Traversal(s)…}

3. **Object-Relational Model**
    a. Values can have structure, rather than being elementary types.
    b. Relations can have associated methods

The Strength of the Relational Model

1. Provides a simple, limited approach to structuring data, yet is reasonably versatile, so anything can be modeled.
2. Provides a limited, yet useful, collection of operations on data.

The, "limitations" allow us to implement languages such as SQL, which enable programmers to express their wishes at a very high level. Implementation becomes much easier, where a few lines of SQL can accomplish the work of hundreds of lines of C.

Basics of the Relational Model

The relational model gives us a single way to represent data: as a two-dimensional table called a relation. The rows each represent a separate entry while the columns represent a property of the table. In the relational model, a database consists of one or more relations.

In this section, we shall introduce the most important terminology regarding relations.

1. **Attributes**
    a. The columns of a relation are named by attributes. They appear at the top of the columns.
    b. Represents the meaning of the entries in the column below.
2. **Schemas**
    a. We show the schema for the relation with the relation's name followed by a parenthesized list of attributes.
        i. The attributes in a relation are a set; We treat the above, "lists" ordering to be the standard order whenever we display the relation or any of its rows.
    b. The set of schemas for the relations of a database is called a relational database schema, or just a database schema.

    Illustration: **Movies(title, year, length, genre)**
3. **Tuples**
    a. Represents the rows of a relation, other than the header row, containing the attribute names.
    b. Has one component for each attribute of the relation.
    c. We use commas to separate components, and use parentheses to surround the tuple when representing it in isolation from a relation.

    Illustration: **(Gone with the Wind, 1939, 231, drama)**
4. **Domains**
    a. The relational model requires that each component of each tuple be atomic; that is, it must be of some elementary type such as an integer or string.
    b. The domain is a necessary requirement, associated with each attribute of a relation, specifying a particular elementary type.
        i. It is possible to include the domain, or data type, for each attribute in a relation schema. We shall do so by appending a colon and a type after attributes.

    Illustration: **Movies(title:string, year:integer, length:integer, genre:string)**

Equivalent Representations of a Relation

Relations are sets of tuples. As a result, the order in which the tuples of a relation are presented is immaterial, leading to many equivalent representations.

Moreover, we can reorder the attributes of a relation as we choose, without changing the relation. Each tuple has its components permuted in the same way as the attributes are permuted.

Relation Instances

We shall call a set of tuples for a given relation an instance of that relation. Relations are not static; they must necessarily change over time. For instance, we expect to insert, remove, or edit tuples.

In addition, the schema of a relation may change as well, although this case is far less common. There are situations where we may want to add or delete attributes. Schema changes can be very expensive in commercial systems, because millions of tuples need to be rewritten to add or delete components.

Keys of Relations

Key constraints are fundamental to the relational model.

A set of attributes forms a key for a relation if we do not allow two tuples in a relation instance to have the same values in all the attributes of the key. This statement must pertain to all possible instances of the relation. We indicate the attribute(s) that form a key for a relation by underlining the key attributes.

Illustration: In the preceding Movies relation, we may choose the key to be formed by the title and year attributes. Thus, we could represent the necessary changes in the relational schema below via:

Movies(title, year, length, genre)

Many real-world databases use artificial keys, doubting that it is safe to make any assumptions about the values of attributes outside their control. Employee ID's are an example of such a key.

Defining a Relation Schema in SQL

SQL ("Sequel") is the principal language used to describe and manipulate relational databases. There are two aspects to SQL:

1. Data-Definition
    a. The sublanguage for declaring database schemas
2. Data-Manipulation
    a. The sublanguage for querying databases and for modifying the database.

In this section, we shall begin a discussion of the data-definition portion of SQL.

Relations in SQL

SQL makes a distinction between three kinds of relations:

1. **Stored Relations (Tables)**
    a. Typically, what we deal with ordinarily; a relation that exists in the database and that can be modified by changing its tuples, as well as queried.
    b. The SQL **CREATE TABLE** statement declares the schema for a stored relation.
        i. It gives a name for the table, its attributes, and their data types.
        ii. It allows us to declare a key, or even several keys, for a relation.
2. **Views**
    a. Relations defined by a computation.
    b. Not stored, but are constructed, in whole or in part, when needed.
3. **Temporary Tables**
    a. Constructed by the SQL language processor when it performs its job of executing queries and data modifications.
        i. These relations are then thrown away and not stored.

Data Types

Recall that all attributes must have a data type. The following is a list of primitive data types that are supported by SQL systems:

1. **Character Strings**
    a. Fixed Length
        i. CHAR(N)
    b. Varying Length
        i. VARCHAR(N)

    The difference between the two is implementation-dependent.

    CHAR implies that the short strings are padded to make n characters, while VARCHAR implies that an end marker or string-length is used.

2. **Bit Strings**
    a. Fixed Length
        i. BIT(N)
    b. Varying Length
        i. BIT VARYING(N)

    These strings are analogous to fixed and varying-length character strings, but their values are strings of bits rather than characters.

3. **Boolean**
    a. Denotes an attribute whose value is logical. The possible values of such an attribute are:
        i. TRUE
        ii. FALSE
        iii. UNKOWN

4. **Integers**
    a. **INT / INTEGER**
        i. Denotes a typical integer value
    b. **SHORTINT**
        i. Also denotes integers, but the number of bits permitted may be less, depending on the implementation (as with the types int and short int in C.)
5. **Floating-Point**
    a. **FLOAT / REAL**
        i. Represents the typical floating-point numbers.
    b. **DOUBLE PRECISION**
        i. Represents the typical floating-point numbers to a higher precision.
    c. **DECIMAL (n, d)**
        i. Represents real numbers with a fixed decimal point.
        ii. This method allows values that consist of N decimal digits, with decimal point assumed to be D positions from the right.
    d. **NUMERIC**
        i. A synonym for DECIMAL, although there are possible implementation-dependent differences.
6. **Dates**
    a. **DATE 'YYYY-MM-DD'**
        i. Single-digit months and days are padded with a leading 0.
        ii. May be coerced to a string type and vice-versa if it makes sense.
7. **Times**
    b. **TIME 'HH:MM:SS.___'**
        i. Uses the 24-hour clock.
        ii. ___ represents the decimal point to the said, "seconds."
        iii. May be coerced to a string type and vice-versa if it makes sense.

Simple Table Declarations

Recall that the simplest form of declaration of a relation schema consists of the keyword **CREATE TABLE** followed by the name of the relation and a parenthesized, comma-separated list of the attribute names and their types.

Illustration

Using the preceding Movies illustration, we can create a simple relation as follows:

```
create table Movies(
    title char(100),
    year int,
    length int,
    genre char(10),
    studioName char(30),
    producerID int
);
```

Modifying Relation Schemas

1. Deletion
    a. **DROP TABLE R;**
        i. Relation R is no longer part of the database schema, and we can no longer access any of its tuples.
2. Alter
    a. **ALTER TABLE R * _____;**
        i. $* \in \{ADD, DROP\}$
        ii. ___ represents an attribute name, or instantiation depending on the alter.
    b. **Note:** In the case where we're adding to an existing table, the default value assigned to all pre-existing tuples would be NULL.

Illustration

Given the preceding relation Movies, we could write an alter statement such as:

ALTER TABLE Movies ADD RATING INT;

Default Values

When we create or modify tuples, we sometimes do not have values for all components. In general, in any place we declare an attribute and its data type, we may add the keyword **DEFAULT** and an appropriate value.

Such a value is either NULL, or a constant.

Illustration

```
10  create table moviestar(
11      name char(30),
12      address varchar(255),
13      gender char(1) default '?',
14      birthdate date default null
15  );
```

Similarly, another way to use a DEFAULT keyword is in an ALTER statement as follows:

ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT 'unlisted';

Declaring Keys

There are two ways to declare an attribute or set of attributes to be a key in the **CREATE TABLE** statement that defines a stored relation.

1. Declare a singular attribute to be a key when that attribute is listed in the relation schema.
2. Declare a list of items declared in the schema an additional declaration that says a particular attribute or set of attributes forms the key.

    Note: We must use the second method in the case where the key consists of more than one attribute.

There are two declarations that may be used to indicate keyness:

1. PRIMARY KEY
2. UNIQUE

The effect of declaring a set of attributes *S* to be a key for relation *R* either using PRIMARY KEY or UNIQUE is the following:

- Two tuples in *R* cannot agree on all of the attributes in set *S*, unless one of them is NULL. Any attempt to insert or update a tuple that violates this rule causes the DBMS to reject the action that caused the violation.

In addition, if PRIMARY KEY is used, then attributes in *S* are not allowed to have NULL as a value for their components.

NULL is permitted if and only if the set *S* is declared UNIQUE.

Illustration

```sql
create table Movies(
    title char(100),
    year int,
    length int,
    genre char(10),
    studioName char(30),
    producerID int,
    primary key (title, year)
);

create table moviestar(
    name char(30) primary key,
    address varchar(255),
    gender char(1) default '?',
    birthdate date default null
);
```

Additional Practice

1. Given the following relations that constitute part of a banking database, indicate the following:

acctNo	type	balance
12345	savings	12000
23456	checking	1000
34567	savings	25

firstName	lastName	idNo	account
Robbie	Banks	901-222	12345
Lena	Hand	805-333	12345
Lena	Hand	805-333	23456

a. The attributes of each relation.

The attributes for the accounts relation are:

1) **acctNo**
2) **type**
3) **balance**

The attributes for the customers relation are:

1) **firstName**
2) **lastName**
3) **idNo**
4) **account**

b. The tuples of each relation.

The tuples for the accounts relation are:

$$\{(12345, savings, 12000), (23456, checkings, 1000), (34567, savings, 25)\}$$

The tuples for the customers relation are:

$$\{(Robbie, Banks, 901-222, 12345), (Lena, Hand, 805-333, 12345), (Lena, Hand, 805-333, 23456)\}$$

c. The relation schema for each relation.

The relation schema for the accounts relation is: Accounts(acctNo, type, balance)

The relation schema for the customer relation is: Customer(lastName, firstName, idNo, Account

d. The database schema.

The database schema would be:

{

Accounts(acctNo, type, balance),
Customer(lastName, firstName, idNo, Account)

}

e. A suitable domain for each attribute.

The domain for the accounts relation is: Accounts(acctNo:integer, type:string, balance:integer)

The domain for the customer relation is: Customer(lastName:string, firstName:string, idNo:integer, Account:integer)

f. Another equivalent way to represent each relation.

The relations could equivalently be represented by:

1) **Changing the ordering of the rows (tuples).**
2) **Changing the ordering of the columns (attributes) in relation to the tuples as well.**

g. How many different ways (considering orders of tuples and attributes) are there to represent a relation instance if that instance has:

1) Three attributes and three tuples.

$$\#(\boldsymbol{Representations}) = \mathbf{3}! \times \mathbf{3}! = \mathbf{36}$$

2) Four attributes and five tuples.

$$\#(\boldsymbol{Representations}) = \mathbf{4}! \times \mathbf{5}! = \mathbf{2880}$$

3) n attributes and m tuples?

$$\#(\boldsymbol{Representations}) = \boldsymbol{n}! \times \boldsymbol{m}!$$

2. In this exercise we introduce one of our running examples of a relational database schema. The database schema consists of four relations, whose schema are:

a. Product (maker, model, type)
b. PC (model, speed, ram, hd, price)
c. Laptop (model, speed, ram, hd, screen, price)
d. Printer (model, color, type, price)

The `Product` relation gives the manufacturer, model number and type (PC, laptop, or printer) of various products. We assume for convenience that model numbers are unique over all manufacturers and product types; that assumption is not realistic, and a real database would include a code for the manufacturer as part of the model number. The `PC` relation gives for each model number that is a PC the speed (of the processor, in gigahertz), the amount of RAM (in megabytes), the size of the hard disk (in gigabytes), and the price. The `Laptop` relation is similar, except that the screen size (in inches) is also included. The `Printer` relation records for each printer model whether the printer produces color output (true, if so), the process type (laser or ink-jet, typically), and the price.

Write the following declarations:

a. A suitable schema for relation Product

Product (maker, model, type)

b. A suitable schema for relation PC

PC (model, speed, ram, hd, price)

c. A suitable schema for relation Laptop

Laptop (model, speed, ram, hd, screen, price)

d. A suitable schema for relation Printer

Printer (model, color, type, price)

e. An alteration to your printer schema for (d) to delete the attribute color.

ALTER TABLE DROP COLOR;

f. An alteration to your Laptop schema from © to add the attribute od (optical-disk type, e.g., cd or dvd). Let the default value for this attribute be 'none' if the laptop does not have an optical disk.

ALTER TABLE LAPTOP ADD od VARCHAR(25) DEFAULT 'none';

2) The Entity/Relationship Model

In the entity-relationship model (or E/R model), the structure of data is represented graphically as an **entity-relationship diagram**. It uses the three following principal element types:

1. **Entity Sets**
    a. An **entity** is an abstract object of some sort, and a collection of similar entities forms an **entity set.**

    In terms of OOP principles, we could make the following logical equivalences:
        1) $Entity \equiv Object$
        2) $Entity\ Set\ \equiv Class$

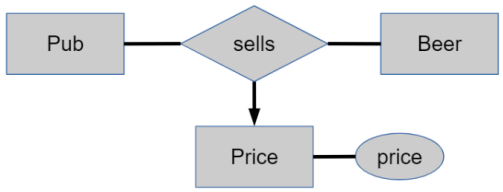
    b. Represented by rectangles in the corresponding entity-relationship diagram.

2. **Attributes**
    a. Resemble properties of the entities in the set. They are of primitive types, such as strings, integers, or reals.
    b. Represented by ovals in the corresponding entity-relationship diagram.
3. **Relationships**
    a. Represents connections among two or more entity sets.
    b. Represented by diamonds in the corresponding entity-relationship diagram.
    c. In general, the value of an E/R relationship can be thought of as a relationship set of tuples whose components are the entities participating in the relationship.

Entity-Relationship Diagrams

Edges connect an entity set to its attributes, and also connect a relationship to its entity sets.

Illustration



Instances of an E/R Diagram

E/R diagrams are a notation for describing schemas of databases.

For each entity set, the database instance will have a particular finite set of entities. Each of these entities has particular values for each attribute.

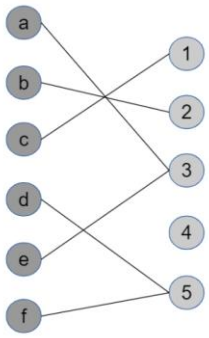
A relationship R that connects n entity sets $E_1, E_2, \ldots, E_n$ may be imagined to have an, "instance" that consists of a finite set of tuples $(e_1, e_2, \ldots, e_n)$ where each e_i is chosen from the entities that are in the current instance of entity set E_i. We regard each of these tuples as, "connected" by relationship R. This set of tuples is called the relationship set for R.

Multiplicity of Binary E/R Relationships

It is common for there to be a restriction on the, "multiplicity" of a relationship. Suppose R is a relationship connecting entity sets E and F.

Then:

1. **Many One**
    a. R is many-one from E to F if each member of E can be connected by R to **at most one** member of F.
    Note: An entity of the second set can be connected to **zero, one, or many** entities of the first set.

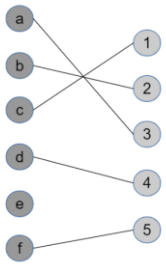


i.e., A drinker has a favourite beer.

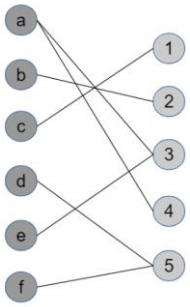
2. **One-One**
    a. R is one-one if it's:
        i. Many-One from E to F
        ii. Many-One from F to E

    In such a relationship, an entity of either entity set can be connected to **at most one** entity of the other set.

    i.e., A manufacturer has a best-selling beer.



3. **Many-Many**
    a. If *R* is neither many-one from E to F or from F to E, then we say that it's many-many. That is, an entity of either set can be connected to many entities of the other set.



i.e., Bars sell beers
i.e., A drinker likes beers

Representing Multiplicity

1. **Edges**
    a. Represents the "many" relationships.
2. **Arrows**
    a. Sharp
        i. Represents the, "at most one" relationship. That is, 0 or 1.
    b. Rounded
        i. Represents the, "exactly one" relationship.

Illustration

Figure 4.3: A one-one relationship

We assume that a president can run only one studio, and a studio has only one president, so this relationship is one-one as indicated by the arrows.

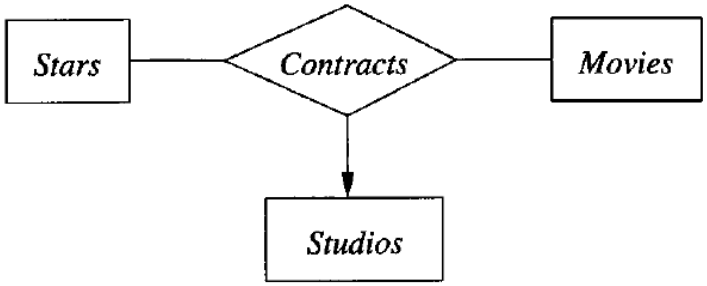
However, a studio might not have a president at some particular time, so the arrow from Runs to Presidents truly means, "at most one."

In addition to this, a president may not run a studio at a particular time, so the arrow from Runs to Studios truly means, "at most one."

Multiway Relationships

The E/R model makes it convenient to define relationships involving more than two entity sets. In practice, it's rare, but they are occasionally necessary to reflect the true state of affairs.

Illustration

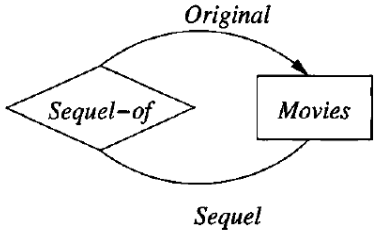


- For a particular star and movie, there is only one studio with which the star has contracted for that movie.
- A studio may contract with several stars for a movie.
- A star may contract with one studio for more than one movie.

Roles in Relationships

In the case where an entity set appears two or more times in a single relationship, we must give each connecting edge a different role. This is because each line represents a different role that the entity set plays in the relationship.

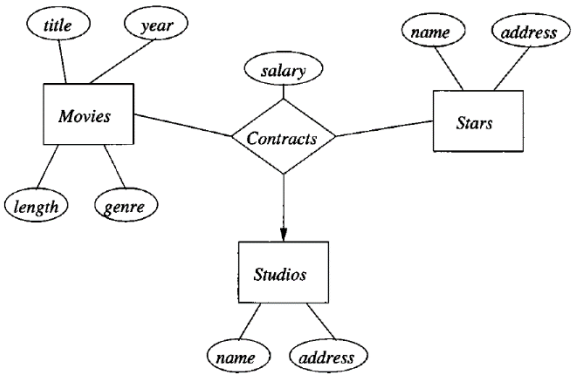
Illustration



We assume that a movie may have many sequels, but for each sequel, there is only one original movie. Thus, the relationship is many-one from Sequel movies to original movies.

Attributes on Relationships

Sometimes it is convenient, or even essential, to associate attributes with a relationship, rather than with any one of the entity-sets that the relationship connects. For example, consider the following salary attribute associated with the relationship connecting entity sets Stars, Movies, and Studios.



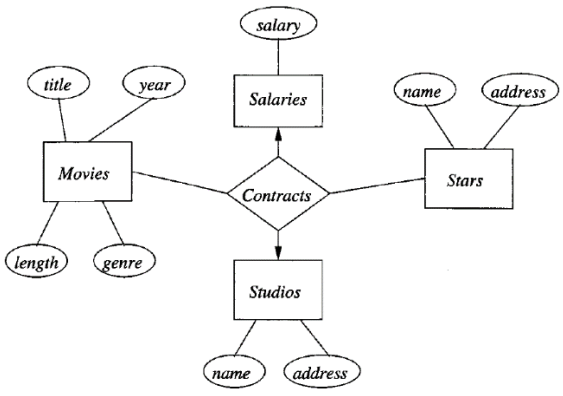
In general, we may place one or more attributes on any relationship The values of these attributes are functionally determined by the subset of the entire tuple in the relationship set for that relation. For example, the salary in the preceding example is really determined by the movie and star entities, since the studio entity is itself determined by the movie entity.

Removing Attributes from Relationships

It is never necessary to place attributes on relationships. We can instead invent a new entity set, whose entities have the attributes ascribed to the relationship. We then include this entity set in the relationship.

In general, when we do a conversion from attributes on a relationship to an additional entity set, we place an arrow into that entity set.

The proceeding illustration below illustrates this process. Notice that there is an arrow into the Salaries entity set. It's appropriate, since we know that the salary is determined by all the other entity sets involved in the relationship.



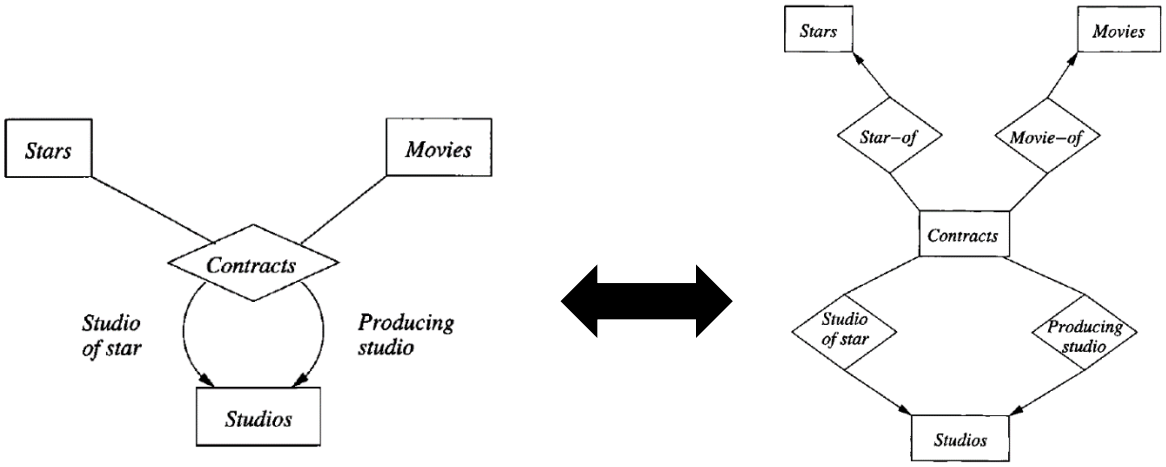
Converting Multiway Relationships to Binary

Any relationship connecting more than two entity sets can be converted to a collection of binary, many-one relationships. To do so, introduce a new entity set whose entities we may think of as tuples of the relationship set for the multiway relationship. We may call this entity set a connecting entity set.

We then introduce many-one relationships from the connecting entity set to each of the entity sets that provides components of tuples in the original multiway relationship.

If an entity set plays more than one role, then it is the target of one relationship for each role.

Illustration



Subclasses in the E/R Model

Often, an entity set contains certain entities that have special properties not associated with all members of the set. If so, we find it useful to define certain special-case entity sets, or subclasses, each with its own special attributes and/or relationships.

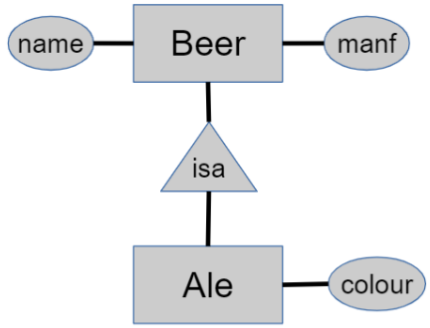
We connect an entity set to its subclasses using a relationship called, "*isa"*. Such a relationship is represented using a triangle, whereby the superclass is connected at the top, while the subclass at the said bottom.

Every *"isa"* relationship is one-one, although we shall not draw the two arrows that are associated with other one-one relationships.

We shall limit isa-structures to trees, in which there is one root entity set that is the most general, with progressively more specialized entity sets extending below the root in a tree.

All subclasses inherit the attributes and participate in whatever relationships its ancestors partake in. In addition to this, only the root entity set has a key, and it must serve as the key for all entities in the hierarchy. Lastly, E/R entities have representatives in all subclasses to which they belong. That is, if an entity *e* is represented in a subclass, then it must be represented in the superclass (and recursively up the tree.)

Subclass – Illustration



Design Principles

1. **Faithfulness**
    a. The design should be faithful to the specifications of the application. That is, entity sets and their attributes should reflect reality.
2. **Avoiding Redundancy**
    a. We should be careful to say everything once only.
    Side Effects of Redundancy Include:
        i. Additional Storage Taken Up
        ii. Update-Anomaly Potential
3. **Simplicity Counts**
    a. Avoid introducing more elements into your design than is absolutely necessary
    Side Effects Include:
        i. Wastes Space
        ii. Encourages Errors
4. **Choosing the Right Relationships**
    a. Adding to our design every possible relationship is not often a good idea.
    Side Effects Include:
        i. Redundancy
        ii. Update Anomalies
        iii. Deletion Anomalies
5. **Picking the Right Kind of Element**
    a. In general, an attribute is simpler to implement than either an entity set or a relationship. However, making everything an attribute will usually get us into trouble.
    Side Effects of Picking the Wrong Element Include:
        i. Repetition of Information
        ii. Update Anomaly
        iii. Deletion Anomaly

When to Choose an Attribute Over an Entity Set

Suppose E is an entity set. Here are conditions that E must obey in order for us to replace it by an attribute or attributes of several other entity sets:

1. All relationships in which E is involved must have arrows entering it.
2. The only key for E is all its attributes
3. No relationships involve E more than once.

Constraints in the E/R Model

Keys in the E/R Model

Recall: A key for an entity set E is a set K of one or more attributes such that, given any two distinct entities e_1 and e_2 in E, e_1 and e_2 cannot have identical values for each of the attributes in the key K.

Specifics

1. Every entity set must have a key, although in some cases – isa-hierarchies and "weak" entity sets, the key actually belongs to another entity set.
2. There can be more than one possible key for an entity set. However, it is customary to pick one key as the, "primary key", and to act as if that were the only key.
3. When an entity set is involved in an isa-hierarchy, we require that the root entity set have all the attributes needed for a key, and that the key for each entity is found from its component in the root entity set, regardless of how many entity-sets in the hierarchy have components for the entity.

Representing Keys in the E/R Model

We shall underline the attributes belonging to a key for an entity set. Note that we only underline the primary key.

Referential Integrity

These kinds of constraints say that a value appearing in one context must also appear in another. For instance, let us consider the many-one, rounded arrow relationship properties.

The rounded arrow notation in E/R diagrams is able to indicate whether a relationship is expected to support referential integrity in one or more directions.

Degree Constraints

In the E/R model, we can attach a bounding number to the edges that connect a relationship to an entity set, indicating limits on the number of entities that can be connected to any one entity of the related entity set.

In the following illustration, a movie entity cannot be connected by the relationship, "Stars-In" to more than 10-star entities.

Weak Entity Sets

A weak entity set is an entity set whose key is composed of some, or all attributes which belong to a differing entity sets. Weak entity sets are sometimes caused by hierarchies unrelated to the "isa" relationship.

Requirements for Weak Entity Sets

If E is a weak entity set, then its key consists of:

1. Zero or more of its own attributes.
2. Key attributes from entity sets that are reached by certain **many-one** relationships from E to other entity sets. These many-one relationships are called **supporting relationships** for E, and the entity sets reached from E are **supporting entity sets.**

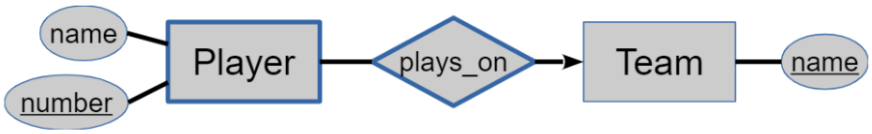
Requirements for Supporting Relationships

1. Relationship R must be a **binary, many-one relationship** from E to F.
2. R must have **referential integrity** from E to F. That is, for every E entity, there must be exactly one existing F entity related to it by R.
3. The attributes that F supplies for the key of E must be key attributes of F.
4. If F is a weak entity set itself, then some or all of the key attributes of F supplied to E will be key attributes of one or more entity sets G to which F is connected by a supporting relationship.
5. If there are several different supporting relationships from E to the same entity set F, then each relationship is used to supply a copy of the key attributes of F to help form the key of E.

Weak Entity Set Notation

1. If an entity set is weak, it will be shown as a rectangle with a double border.
2. If a relationship is supporting, it will be shown as a diamond with a double border, supporting the referential integrity constraint.

Weak Entity Set Notation - Illustration



From E/R Diagrams to Relational Designs

From (Non-Weak) Entity Sets to Relations

For each non-weak entity set, we shall create a relation of the same name and with the same set of attributes. This relation will not have any indication of the relationships in which the entity set participates.

From (Non-Supporting) Relationships to Relations

1. For each entity set involved in relationship R, we take its key attribute(s) as part of the schema of the relation.

   Note: If one entity set is involved several times in a relationship, in different roles, then its key attributes each appear as many times as there are roles. We must rename the attributes to avoid name duplication.

2. If the relationship has attributes, then these are also attributes of relation R.

Combining Relations

Sometimes the relations that we get from converting entity sets and relationships to relations are not the best possible choice of relations for the given data.

One common situation occurs when there is an entity set E with a **many-one** relationship R from E to F. Due to the fact that R is many-one, all these attributes are functionally determined by the key for E, and we can combine them into one relation with a schema consisting of:

1. All attributes of E.
2. The key attributes of F, which are converted to attributes of our new relation.
3. Any attributes belonging to relationship R.

Whether or not we choose to combine relations in this manner is a matter of judgement. The advantages of doing this involve greater efficiency in our queries involving attributes of one relation, rather than using many relations to do the same said task.

Handling Weak Entity Sets

1. The relation for the weak entity set W itself must include the attributes of W, but also the key attributes of the supporting entity set(s).
2. The relation for any relationship in which the weak entity set W appears must use as a key for W, all of its key attributes, including those of other entity sets that contribute to W's key.
3. A supporting relationship R, from the weak entity set W to a supporting entity set need not be converted to a relation at all, unless it has an attribute.

Converting Subclass Structures to Relations

For the purposes of maintaining brevity of the notes, we will only cover the E/R viewpoint method of converting subclass structures to relations.

For each entity set E in the hierarchy, create a relation that includes the key attributes from the root and any attributes belonging to E.

Keys and Foreign Keys

Foreign Keys

Foreign keys are used to maintain and implement the referential-integrity constraints. These foreign key constraints assert that a value appearing in one relation must also appear in the primary-key component(s) of another relation.

They are represented in the schema with an underline and appending star at the said end of the attribute label.

Declaring Foreign-Key Constraints

In SQL, we may declare an attribute, or attribute(s) of one relation to be a foreign key, referencing some attribute(s) of a second relation. The implication of this declaration is two-fold:

1. The referenced attribute(s) of the second relation must be declared UNIQUE, or the PRIMARY KEY for their relation.
2. Values of the foreign key appearing in the first relation must also appear in the referenced attributes of some tuple.

Single Attribute Declaration

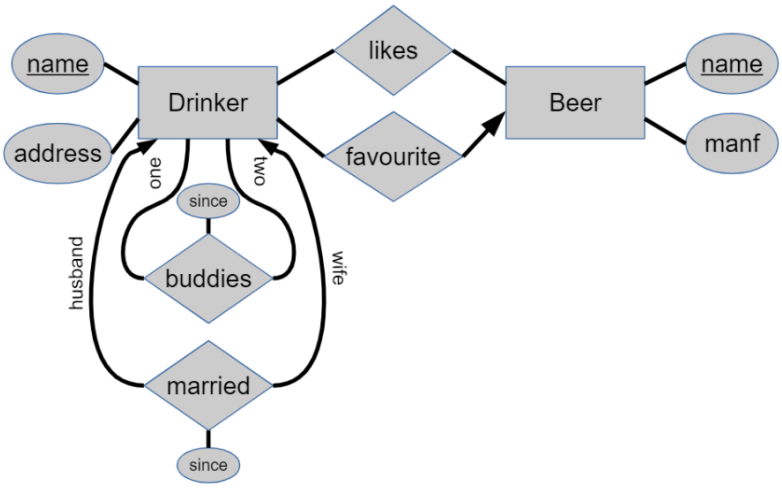
$ATTRIBUTE\ REFERENCES\ TABLE(Attribute)$

Multi-Attribute Line Declaration

$FOREIGN\ KEY\ \ (ATTRIBUTES)REFERENCES\ TABLE(ATTRIBUTES)$

Additional Practice

1. Give the schemas for the corresponding relationships of the following E/R diagram.



The following schemas are presented as:

married(**husband***, **wife***, since)
buddies(**drinker1***, **drinker2***, since)
likes(**drinker***, **beer***)
favourite(**drinker***, **beer***)

2. From the first question, provide the schema for the combined relation consisting of relation Drinkers and Favorite.

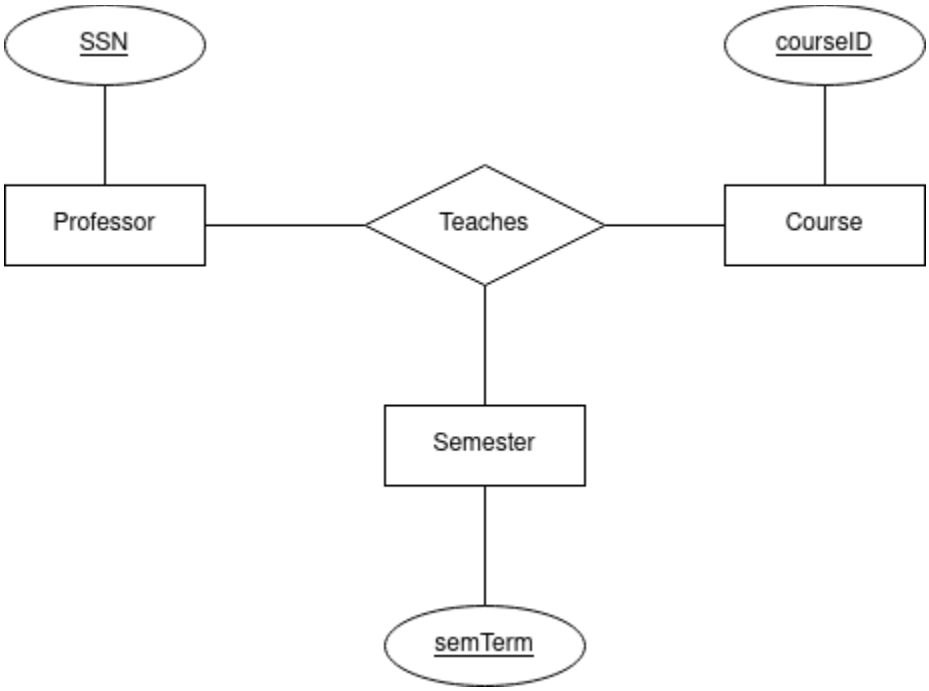
The combined relation of Drinker and Favourite will produce the following relational schema:

Drinker(**name**, address, beer)

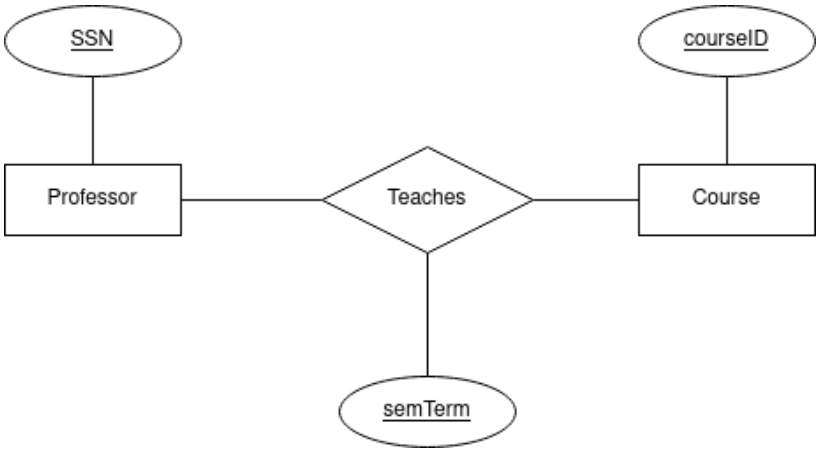
3. A University database contains information about professors (identified by social security number, or SSN) and courses (identified by courseid). Professors teach courses, each of the following situations concerns the, “Teaches” relationship set.

   For each situation, draw an ER diagram that describes it (assuming that no further constraints hold.)

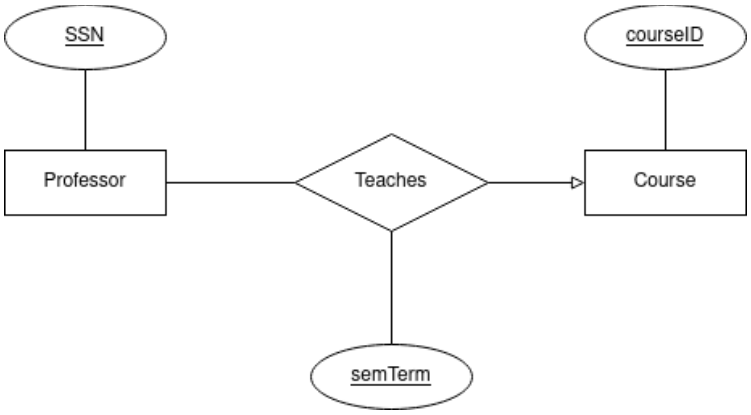
   a. Professors can teach the same course in several semesters, and each offering must be recorded.



   b. Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies in all subsequent questions.)

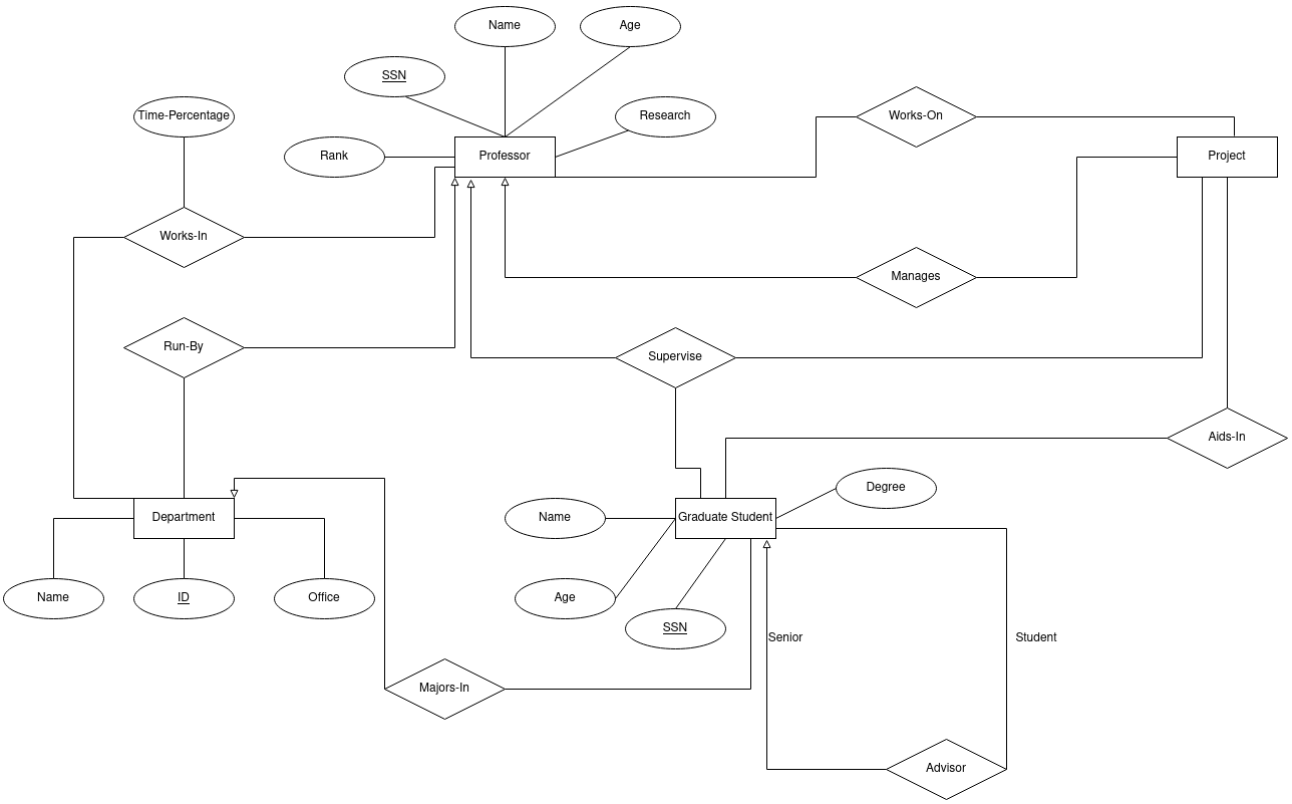


c. Every Professor teaches exactly one course (no more, no less.)

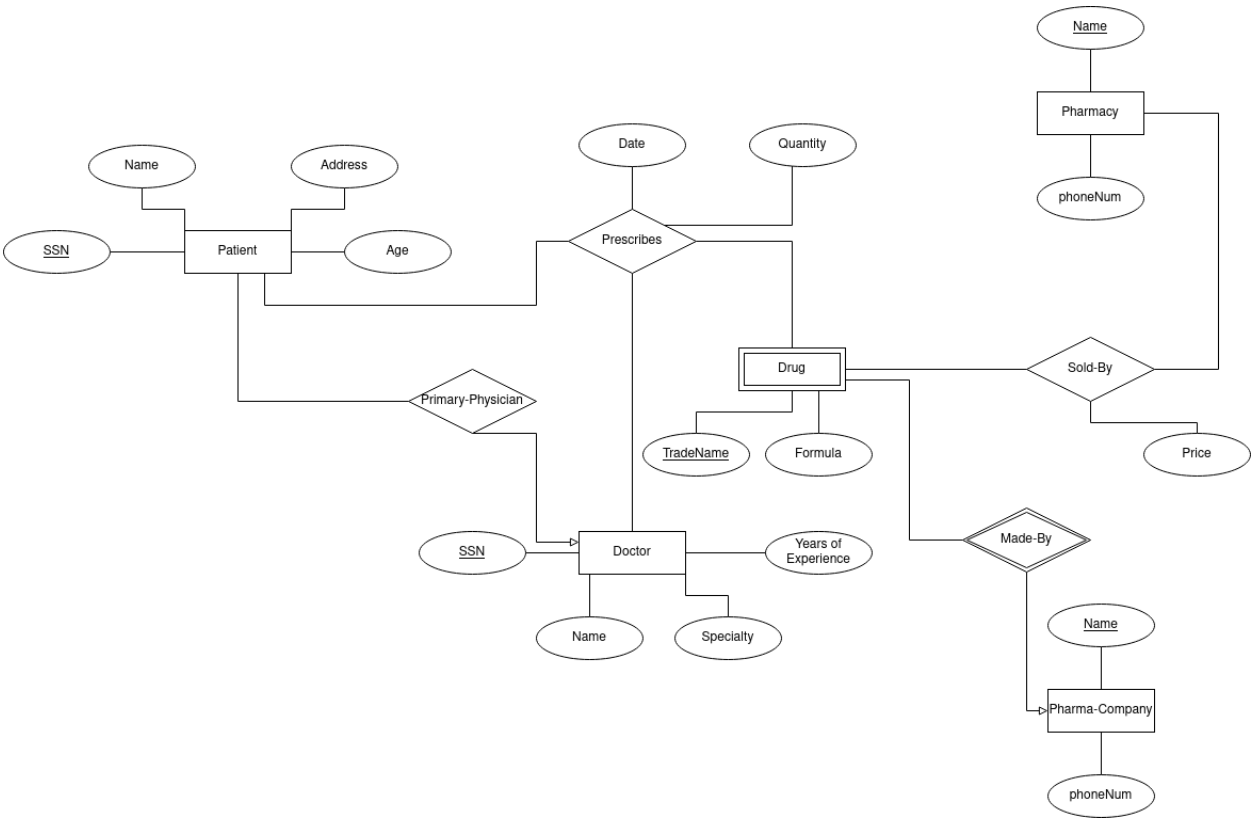


4. Consider the following information about a University Database:
    a. Professors have an SSN, name, age, rank, and research speciality.
    b. Graduate Students have an SSN, name, age, and degree program (e.g., M.S. or Ph.D.).
    c. Each project is managed by one professor (Known as the projects principal investigator.)
    d. Professors can manage and/or work on multiple projects.
    e. Each project is worked on by one or more graduate students (known as the projects research assistants.)
    f. When graduate students work on a project, a professor must supervise their work on the project.
    g. Graduate students can work on multiple projects, in which case they will have a potentially different supervisor for each one.
    h. Departments have a department number, name, and main office.
    i. Departments have a professor (known as the chairman) who runs the department.
    j. Professors work in one or more departments, and for each department that they work in, a time percentage is associated with their job.
    k. Graduate students have one major department in which they are working on their degree.
    l. Each Graduate student has another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.)

Draw an ER diagram that descries the theoretical database.



5. The Prescriptions R-X chains of pharmacies has offered to give you a free lifetime supply of medicines if you design its database. Given the rising costs of healthcare, you agree. Here's the information that you gather:
    a. Patients are identified by an SSN, and their name, address, and age must be recorded.
    b. Doctors are identified by an SSN, and their name, speciality, and years of experience must be recorded.
    c. Each pharmaceutical company is identified by name, and has a phone number.
    d. For each drug, the trade name and formula must be recorded. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely from among the products of that company. If a pharmaceutical company is deleted, you need not keep track of its products any longer.
    e. Each pharmacy has a name, address, and phone number.
    f. Every patient has a primary physician.
    g. Every doctor has at least one patient.
    h. Each pharmacy sells several drugs and has a price for each. A drug could be sold at several pharmacies, ad the price could vary from one pharmacy to another.
    i. Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient can obtain prescriptions from several doctors.
    j. Each prescription has a date and quantity associated with it.



6. York University has decided to consolidate the functionality of three small overlapping database systems which support applications for
   1) teaching (e.g. instructor assignment and evaluation), for
   2) registration (e.g. online course status, waiting lists), and for
   3) student records (e.g. transcript generation).

   The resulting new system will support the following enterprise description:

   Professors and graduate teaching assistants (GTAs) are assigned as a team to administer the sections of each class being offered in a semester. At the end of the semester, they get a "team rating" (professors and GTAs together get one rating per section). To support the assignment of professors to sections, a record is kept of which class each professor can teach. Classes can have one or more prerequisite classes. Students can take several sections each semester, and receive a grade for taking each section. Students may end up waiting for some sections, and receive a "rank" (determining the order they will be admitted if other students drop). However, no more than 10 students can wait on a class at the same time. Note that GTAs are students, however they differ in that they have a salary. All people (e.g. students, professors) are uniquely identified by their social security number. All classes are identified by department name (e.g. "EECS") and course number (e.g. "3421"). Sections of classes are distinguished by their section number (e.g. "N").

   Part 1 - Given this functional description of the business processes at YU:

   Draw an ER-diagram for the database, identifying the following:

   (i) all the entity sets,

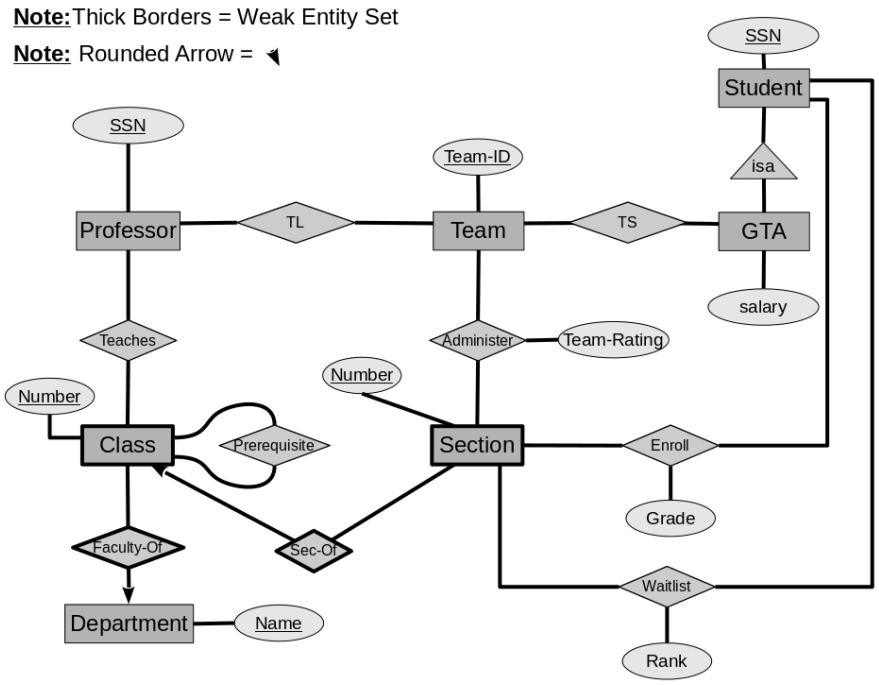
   (ii) all the relationship sets and their cardinalities (key constraints, i.e. "many to many", "one to one", etc.), and

   (iii) the key for each entity set (and weak entity set, if any). You can invent your own attribute(s) for the entity sets (in addition to any mentioned).

Database Schema

1. **Entity Sets**
   a. Professors(SSN)
   b. Team(ID)
   c. Students(SSN)
   d. GTAs(SSN, Salary)
   e. Class(Department, CourseNum)
2. **Weak Entity Sets**
   a. Section(Number, classNo*, DeptName*)
3. **Relationships**
   a. CanTeach(SSN*, Department*, CourseNum*)
   b. On_Team1(SSN*, TeamID*)
   c. Admin(Rating, SSN*, SecID*, Class-Dep*, Class-CN*)
   d. On_Team2(TeamID*, SSN*)
   e. Wait_For(Rank, Sec_ID*, Class-Dep*, Class-CN*, SSN*)
   f. Take(Grade, Sec_ID*, Class-Dep*, Class-CN, SSN*)
   g. Req_Of(ClassDep*, ClassCN, PRCD*, PRCN*)

E/R Model



Part 2 - Implement the preceding E/R schema using PostgreSQL.

```
CREATE TABLE Professors(
  name    VARCHAR(30),
  SSN     CHAR(9),
  PRIMARY KEY(SSN)
);
CREATE TABLE Team(
  TeamID  INTEGER,
  PRIMARY KEY(TeamID)
);
CREATE TABLE Students(
  name    VARCHAR(30),
  SSN     CHAR(9),
  PRIMARY KEY(SSN)
);
CREATE TABLE GTAs(
  name    VARCHAR(30),
  SSN     CHAR(9),
  salary  FLOAT,
  PRIMARY KEY(SSN),
  FOREIGN KEY (SSN) REFERENCES Students(SSN)
);
CREATE TABLE Class(
  Department      VARCHAR(25),
  CourseNumber    INTEGER,
  PRIMARY KEY(Department, CourseNumber)
);
```

```
CREATE TABLE Section(
  SectionID       CHAR(1),
  Department      VARCHAR(25) NOT NULL,
  CourseNumber    INTEGER NOT NULL,
  PRIMARY KEY(SectionID, Department, CourseNumber),
  FOREIGN KEY (Department, CourseNumber) REFERENCES Class(Department, CourseNumber)
);
CREATE TABLE CanTeach(
  SSN             CHAR(9),
  Department      VARCHAR(25),
  CourseNumber    INTEGER,
  PRIMARY KEY(SSN, Department, CourseNumber),
  FOREIGN KEY (SSN) REFERENCES Professors(SSN),
  FOREIGN KEY (Department, CourseNumber) REFERENCES Class(Department, CourseNumber)
);
CREATE TABLE On_Team1(
  SSN     CHAR(9),
  TeamID  INTEGER,
  PRIMARY KEY(SSN, TeamID),
  FOREIGN KEY (SSN) REFERENCES Professors(SSN),
  FOREIGN KEY (TeamID) REFERENCES Team(TeamID)
);
```

```
CREATE TABLE Admin(
  Rating  INTEGER,
  TeamID  INTEGER,
  SectionID CHAR(1),
  Department VARCHAR(25) NOT NULL,
  CourseNumber INTEGER NOT NULL,
  PRIMARY KEY(TeamID, SectionID, Department, CourseNumber),
  FOREIGN KEY (TeamID) REFERENCES Team(TeamID),
  FOREIGN KEY (SectionID, Department, CourseNumber) REFERENCES Section(SectionID, Department, CourseNumber)
);
CREATE TABLE On_Team2(
  TeamID  INTEGER,
  SSN     CHAR(9),
  PRIMARY KEY(TeamID, SSN),
  FOREIGN KEY (TeamID) REFERENCES Team(TeamID),
  FOREIGN KEY (SSN) REFERENCES GTAs(SSN)
);
CREATE TABLE Wait_For(
  Waitlist_Rank   INTEGER,
  SectionID       CHAR(1),
  Department      VARCHAR(25) NOT NULL,
  CourseNumber    INTEGER NOT NULL,
  SSN             CHAR(9),
  PRIMARY KEY(SectionID, Department, CourseNumber, SSN),
  FOREIGN KEY (SectionID, Department, CourseNumber) REFERENCES Section(SectionID, Department, CourseNumber),
  FOREIGN KEY (SSN) REFERENCES Students(SSN)
);
```

```
CREATE TABLE Take(
  Grade          VARCHAR(6),
  SectionID      CHAR(1),
  Department     VARCHAR(25) NOT NULL,
  CourseNumber   INTEGER NOT NULL,
  SSN            CHAR(9),
  FOREIGN KEY (SectionID, Department, CourseNumber) REFERENCES Section(SectionID, Department, CourseNumber),
  FOREIGN KEY (SSN) REFERENCES Students(SSN)
);
CREATE Table Req_Of(
  PR_Department          VARCHAR(25),
  PR_CourseNumber        INTEGER,
  R_Department           VARCHAR(25),
  R_CourseNumber         INTEGER,
  PRIMARY KEY(PR_Department, PR_CourseNumber, R_Department, R_CourseNumber),
  FOREIGN KEY (PR_Department, PR_CourseNumber) REFERENCES Class(Department, CourseNumber),
  FOREIGN KEY (R_Department, R_CourseNumber) REFERENCES Class(Department, CourseNumber)
);
```

Part 3 – Provide an SQL program that drops the generated tables in an appropriate order, avoiding any run-time errors.

```
drop table Admin;
drop table CanTeach;
drop table On_Team1;
drop table On_Team2;
drop table Wait_For;
drop table Take;
drop table Req_Of;
drop table Professors;
drop table Team;
drop table GTAs;
drop table Students;
drop table Section;
drop table Class;
```

Part 4 – Populate your relations with appropriate data.

```
insert into Professors (name, SSN) values
    ('Jarek Gryz', '111111111'),
    ('Paul Skoufranis', '222222222');
insert into Students (name, SSN) values
    ('John Malkovich', '333333333'),
    ('Steven Chen', '444444444'),
    ('Oscar Whiley', '555555555'),
    ('Emma Whyte', '666666666');
insert into gtas (name, SSN, salary) values
    ('Oscar Whiley', '555555555', 38.81),
    ('Emma Whyte', '666666666', 27.00);
insert into team (teamid) values
    (1),
    (2),
    (3),
    (4);
insert into class (department, coursenumber) values
    ('EECS', 3421),
    ('EECS', 3101),
    ('MATH', 1300),
    ('MATH', 1310);
insert into section (sectionid, department, coursenumber) values
    ('A', 'EECS', 3421 ),
    ('B', 'EECS', 3101),
    ('C', 'MATH', 1300),
    ('D', 'MATH', 1310);
```

```
insert into on_team1 (SSN, TeamID) values
    ('111111111', 1),
    ('111111111', 2),
    ('222222222', 3),
    ('222222222', 4);
insert into on_team2 (SSN, TeamID) values
    ('555555555', 1),
    ('555555555', 2),
    ('666666666', 3),
    ('666666666', 4);
insert into admin (rating, teamid, sectionid, department, coursenumber) values
    (9, 1, 'A', 'EECS', 3421),
    (7, 2, 'B', 'EECS', 3101),
    (7, 3, 'C', 'MATH', 1300),
    (10, 4, 'D', 'MATH', 1310);
insert into wait_for (waitlist_rank, sectionid, department, coursenumber, ssn) values
    (1, 'A', 'EECS', 3421, '444444444'),
    (9, 'D', 'MATH', 1310, '444444444');
insert into take (grade, sectionid, department, coursenumber, ssn) values
    ('A+', 'A', 'EECS', 3421, '333333333'),
    ('A+', 'B', 'EECS', 3101, '333333333'),
    ('B+', 'C', 'MATH', 1300, '333333333'),
    ('B+', 'D', 'MATH', 1310, '333333333'),
    ('A+', 'A', 'EECS', 3421, '444444444'),
    ('B+', 'B', 'EECS', 3101, '444444444');
insert into req_of (pr_department, pr_coursenumber, r_department, r_coursenumber) values
    ('MATH', 1300, 'MATH', 1310);
```

For access to the **remaining notes**, please contact me over LinkedIn or GitHub.