- Relational databases are pervasive

MODEL

- Characterization or representation of some phenomenon
    - Model is a representation that enables the capture of all the relevant data required for a system to function or work.

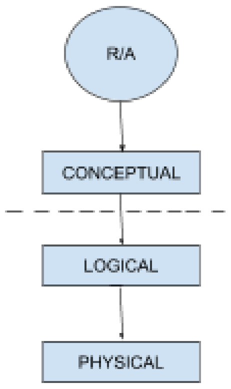
- Granularity = The lowest level of detail. At the lowest or the finest level of granularity, database stores data in data blocks.

DATA MODELING

- Data modeling is an attempt to represent real-world phenomena in order to capture relevant data and turn that data into INFORMATION.

REQUIREMENTS ANALYSIS

- The most important part of the modeling process.
- Better Analysis = Faster implementation
- Communication
- Mistakes made here are hard to fix later in the implementation phase.
- **Finding the entities, attributes, and relationships**



- The Conceptual schema is generally a pencil - to - paper model of the real - world system.
- Generally NOT implementation-specific.
- Use ERD for conceptual schema

- DASH LINES represent data mapping
- make a decision about what type of logical data model we will use to implement the system.
- The logical data model has to do with the logical schema.

- The logical schema is an implementation specific schema.
- It is composed of the descriptions of the data structures that will be employed.
- Translation from conceptual to Logical is fairly Automatic (SQL Database).

- Physical schema is a description of the lowest level implementation specific details of the data store.
- It can be machine or operating system specific
- Use MYSQL

- Modeling process involves three separate tiers.
- Each tier is independent of the tiers above.
- Changes made at higher levels means lower levels have to be re-worked.
    - This is NOT true in reverse

ENTITIES

- noun that represent generic classes
- DOES NOT have to be tangible, it can be conceptual.
- NOT Entities → System as a whole (“IU”), Individual Example (“I308”)

ATTRIBUTES

- Describe Entities
- Properties and characteristics of an entity
- Many entities has one or more attributes that uniquely identify an instance of an entity
    - EX) SSN, Student ID
        - These are called PRIMARY IDENTIFIERS
- You CANNOT have an entity without attributes
- Atomic Attribute : Simple, does not contain any meaningful smaller component.
- Composite : Comprises two or more other attributes.
- Multivalued : Many values associated with it at any one point in time. ex) PHONE

RELATIONSHIPS

- The way that two or more entities are connected together is called a relationship.
- Relationship have a cardinality that represents how many different entities participate in the relationship.

- 2 Binary
- 3 Ternary
- 4 Quarternary
- 5 Quinary
- 6 Senary
- 7 Septenary

STRONG & WEAK ENTITIES

- Weak entity is added to capture information about relationship.
- Weak entity basically describes the relationship.
- Strong Entities → Most Common, Attributes fully identify.
- Weak Entities → Not as common, Attributes do not fully identify, Identity based on relationship. EX) Administrative events.

SUPERTYPES & SUBTYPES

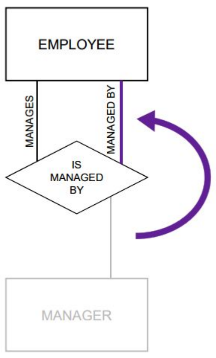
- Some entities have a common core set of attributes but differ in some other attributes.
- Sometimes a good solution is to create a **supertype** (super-entity) and **subtypes** (sub-entities).
- Subtypes inherit attributes from the supertype.
- Relationship is denoted with the ISA using a triangle.
- All common attributes are placed in the super type.
- Non-common attributes are distributed among the subtypes.

Cardinality Constraints

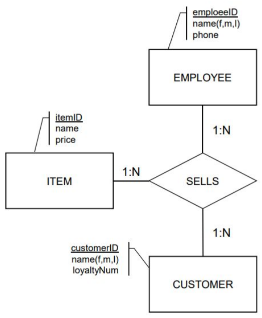
- The last step for an ERD is to add cardinality constraints.
- For every entity that participates in a relationship, we add a min and a max cardinality value to the ERD.
    - it is written as min:max
- The **minimum cardinality** represents whether the entity is required for the relationship to occur.
    - It is either 0 or 1.
    - Minimum of 0 means the entity is not necessarily required for the relationship to occur.
    - The minimums cannot be 0 if there are only two entities involved in the relationship.
- The **maximum cardinality** represents how many times a given entity is allowed to participate in relationships.
    - it is either 1 or N, where N means many
    - sometimes we use an actual number in place of N if number is known.
- weak entities always have a cardinality of 1:1 so we usually do not notate.
- Databases cannot handle discrete cardinality constraints beyond 0,1, & N.
    - This control is usually handled by the application using the database.
- If you are given a discrete cardinality, you have two options.
    - 1. Put the discrete cardinality in your ERD.
    - 2. Use 0, 1, or N’s and note the limit in the ERD Legend.
- Minimum cardinality is 0 when a relationship can occur without the entity.
- Minimum cardinality is 1 when the entity is required for the relationship to occur.

Recursive Relationships

- Often instances of an entity have a relationship with other instances of the same entity.

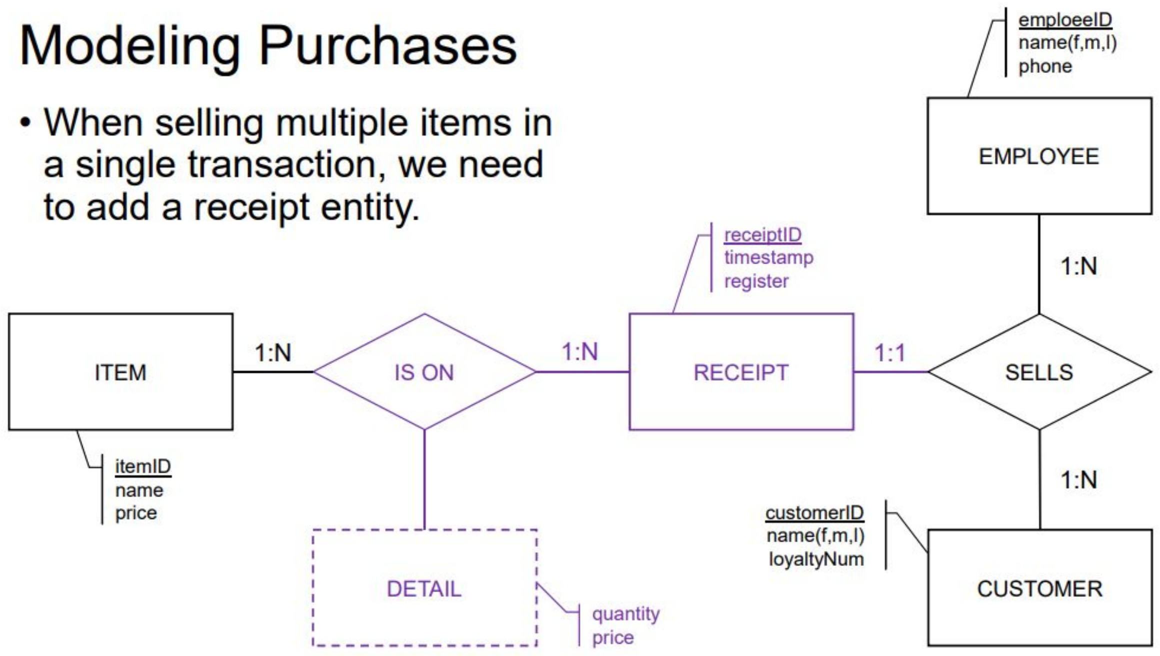
- 

Modeling Purchases



(Single)

- When selling multiple items in a single transaction, we need to add a receipt entity.



Modeling Purchases

- When selling multiple items in a single transaction, we need to add a receipt entity.

(Multiple)

Conversion, the Next Step

- While Entity-Relationship Diagrams give us a structured, detailed model of the system we are trying to understand, we still need to be able to convert that information into a more usable form.
- Relational Models are just that – the logical schema of the relations (tables) that we will eventually instantiate in our database.

Relational Model Terminology

- A **relation** is a two-dimensional table of tuples. Relations are also called tables.
- A **tuple** is a row of data made up of attributes. Tuples are also called rows.
- An **attribute** is a named characteristic of data values. Attributes are also called columns.
- Values are from a **domain**. A domain is a set of atomic values
  - Possibly infinite (integers, name)
  - Possibly finite (job title)

Conversion Rules – Relationships

- This is based on maximum cardinality in the ERD.

Strong Entities

- Directly converted to a relation

Weak Entities

- Become relations and inherit the relationship.
- gets the foreign keys from strong entities.

One-to-One

- Import a key from one entity to another as foreign key.

One-to-Many

- Parent to child

- import parent's primary key to child as foreign key.

Many-to-Many

- Can not be directly implemented with the existing entity relations
- Introduce a third intersection relation and copy primary keys from original entities.

Recursive

- Directly include another copy of the key in the relation.
- EX) manegerID→ from manages

ISA

- copy the primary key from supertype to subtypes
- supertype has relation, subtypes have relations

Multivalued Attributes

- remove the field from the original relation
- create a new relation
- add type to a new relation
- EX) employee_phone(empID→, phone, type)

Composite Attributes

- Expand the composite attribute to be the parts
- EX) nameFirst, nameLast

Order of conversion

- It is recommended to convert an ERD in the following order:
    1. Strong Entities and then their attributes
    2. Weak Entities and then their attributes
    3. Any Relationships left.

Normalization

- Goals of Normalization
    - reduce the amount of space
    - ensure the data is logically stored to avoid duplication of data
        - Duplication of data is called redundancy
    - avoid potential inconsistencies in data
        - inconsistencies often result from having redundant data
    - Most systems are normalized to Third normal Form (3NF)
    - The more you normalize, the slower your database response.
- 1NF
    - Eliminate duplicate columns in the same table.
    - Create separate tables for each group of related data.
    - Identify each row with unique columns.
- 2NF
    - Apply the 1NF rules.
    - Remove subsets of data that apply to multiple rows to a new table.
    - Create the primary/foreign relationship between these tables.
- 3NF
    - Apply the 2NF rules
    - Remove the columns that are not fully dependent upon the primary key.

When NOT to Normalize

- Joins are expensive(slow)
    - reporting systems where data is combined for faster retrieval.

- Normalization design is difficult
    - Going beyond 3NF requires a lot of theoretical skill and knowledge
- Quick design is necessary
    - Prototyping or proof of concept work
- No need to worry about inconsistency
    - Data is frozen, so there are no new updates to cause inconsistencies.