COMP 421 Study guide

Francis Piché

January 24, 2019

Contents

Part I
Preliminaries

1 Disclaimer

These notes are curated from Joseph D'silva COMP421 lectures at McGill University. They are for study purposes only. They are not to be used for monetary gain.

Part II
Introduction to Databases

2 Data Storage

In operating systems, file systems are used for persistent storage. This is insufficient because:

- Only provides a basic API
- Information may not be structured.
- Only linear seek possible (slow)
- May have data loss in case of power outages etc
- Can't have concurrent access to files

3 Relational Model and Data Definition Language

Entity Relationship Model is a language that describes the data determined through requirement analysis. This is very similar to UML, but is not the same.

3.1 Requirement Analysis

When designing a database, we must first identify which data needs to be stored, and how it will be used. (Which operations need to be executed on the data).

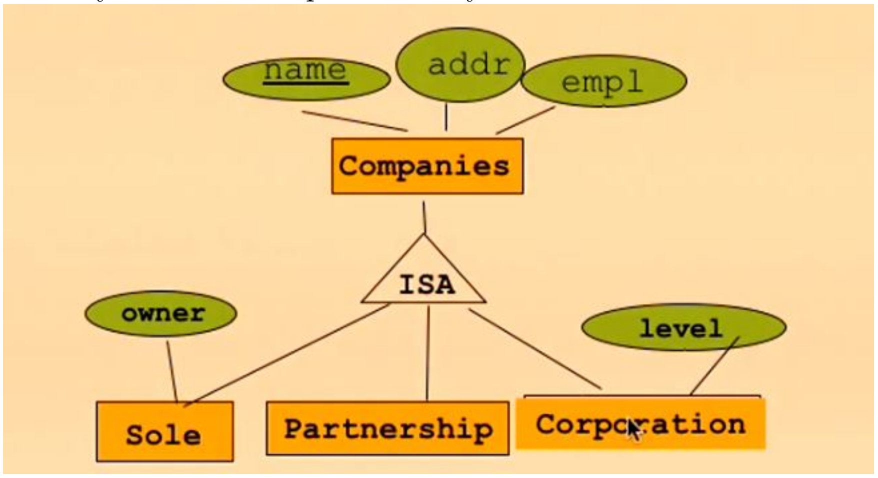
For example, if we were designing a database for Minerva at McGill, we would need to think about which entities are relevant to be stored. In OOP, these would be the classes. Things like the students, addresses, fees, courses etc are all relevant information. We would also need to think about the types of operations that must be possible by our database. Things like adding students, changing addresses, assessing fees etc.

In an OOP setting, we would break down the problem by identifying classes and relationships between them. For example, we might have Student and Instructor classes, with a parent class Person, since the Student and Instructor share some attributes (name, age...). We also have relationships between classes such as a Student and Transcript. A Transcript does not exist unless assigned to a Student. We need ways to model all of these relationships in a database setting rather than an OOP setting.

3.2 ER

An *entity* is a real world object which contains a set of attributes. A collection of instantiated entities would be an *entity set*. An entity set must have a *key*. This is an attribute that is underlined, and MUST be unique. There can be two attributes underlined, in which case both COMBINED must be unique. Individually they need not be unique.

ISA (”is a”) hierarchy is similar to subclasses in OOP. This is represented as a triangle. The key is still in the parent entity set.



There is a subtle point here in that unlike OOP, the attributes of the parent are not part of the child entity set (to avoid duplication, and keep things consistent).

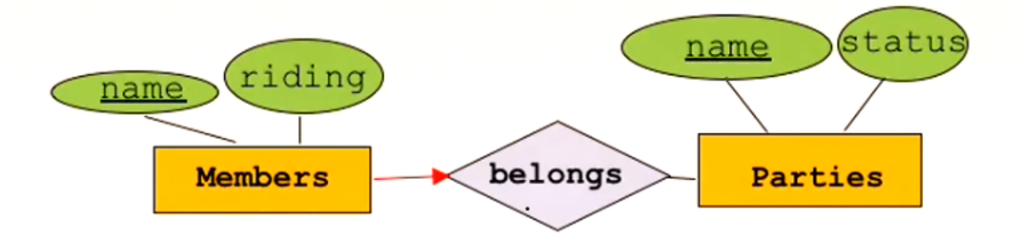
If there are overlaps (a company can be both a partnership and Corporation), we must put a note explaining the situation. We assume that the parent is not covering all possible subtypes. (There may exist a company which is not a Sole, Partnership or Corporation).

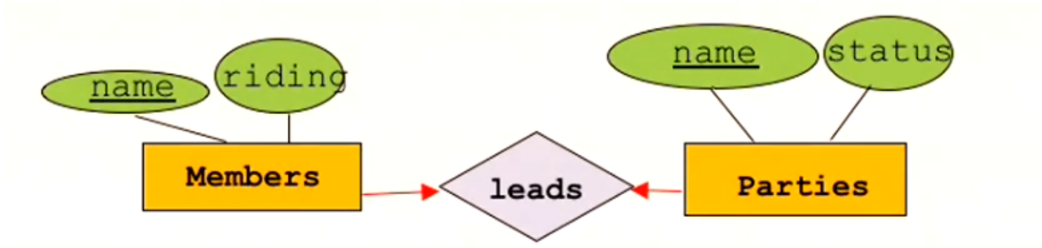
3.3 Relationships

A relationship is an association among two or more entities. This can be one to one, many-to-one, many-to-many. A relationship set is a collection of similar relationships.

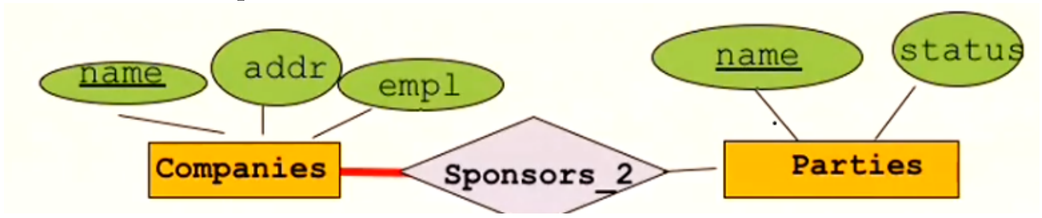
There cannot be duplicate association instances. For example, if a Company donates 10$ to a Party, (the association being the sponsorship), then there cannot be another 10$ donation with the same Company and Party. The original must be updated.

There are constraints as well. We can add an arrow, or two, to constrain to a one-to-many or one-to-one (respectively).





Participation constraints are when we must have at least one entity set is participating in the relationship.

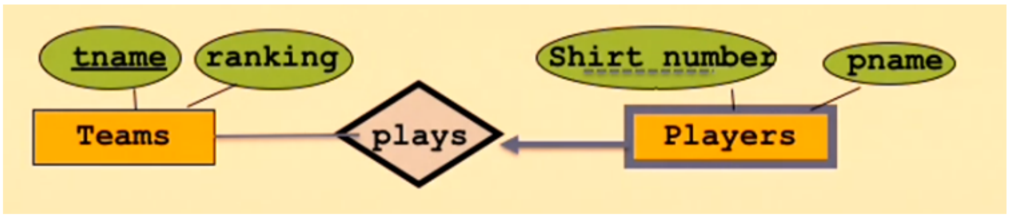


which is shown by a bold line. This can be combined with a key constraint to show a relationship in which exactly one entity set is related.

Ternary relationships are relationships involving 3 entity sets. Keep in mind that a ternary relationship database entries MUST include all 3 entities. If you have more than a ternary relationship (n-ary), it's probably an indication of bad design.

3.4 Weak Entities

Weak Entities are entities which do not have a natural attribute which can be used as a key, but there is another entity set with which we can identify the entities. For example, suppose we have Teams and Players. The teams have names, Players have names and shirt numbers. Here, the Team name may be unique for a league, but not overall, and a shirt number may be unique within a team, but not for the league. However, we can use the Team name as a primary key, with the Player shirt number as a partial key to identify a player. Note that this implies that a Player cannot exist without belonging to a Team.

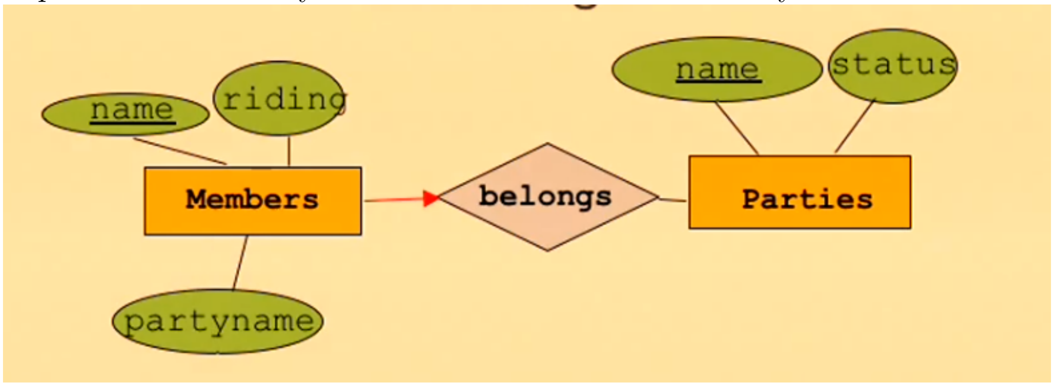


We often create artificial keys to keep track of information. For example student numbers. However, you should never use an artificial key as a partial key. If you're going to make one, make sure it's unique.

We use weak entities when there is no global authority that allows creating unique identifiers.

3.5 Avoiding Redundancies

We should avoid redundancies by not storing the same information more than once. This is important since it may allow for data to become out of sync.



suppose someone changed the partyname but not the name of the associated party. The data would be inconsistent.

3.6 Entity Set or Attribute

When can an entity set simply be an attribute? If we only care about the key of an entity set, and we only have a one to many relationship, we can represent it instead as an attribute. The one-to-many is important since we can't have a list as an attribute value. For example, if we have Movies, with the attribute Category (rather that have a relationship between Movie and Category), we can't have a list of Categories if the Movie belongs to several Categories. This would require leaving it as a relationship rather than an attribute.

4 Relational Model

This is the most common model, and is closer to actual implementation. Used by iBM DB2, PostgreSQL, MySQL, SQLite etc.

There are other types of models, which we will either ignore or look at later.

A relational database is nothing more than a set of relations. A relation consists of a schema and instance. A schema defines the name of a relation, its attributes, and the domain/type of each attribute. `Students(sid: int, name: string, login: string, faculty: string, major: string)` Ie: relation == table. Not to be confused with relationship in ER modeling (different things!). The instances are actual tuples (rows) in the table.

We cannot have identical instances (duplicate rows), at least one attribute value must be different. We also cannot assume anything about the order of the rows. The column order also doesn't matter.

4.1 DDL and DML

DDL (data definition language) defines the schema of a database. (What the tables are called, their attributes) This only gives structure, no real data.

DML (data manipulation language) manipulates the data. (deals with instances of the relations). For example, insert, update delete, query etc.

Part III
SQL

SQL is used as a standard to define DDL and DML. There are many dialects of it in varius DB's.

SQL has data types.

- CHAR (fixed length string)
- VARCHAR (variable length string)
- INT
- SHORTINT

5 Simple Queries

6 Advanced Queries

7 Constraints

8 Application Programming

Part IV DB Internals

9 Buffer Management

10 Indexing

11 Query Execution

12 Query Optimization

13 KV Stores

14 Map Reduce

15 Transactions

16 Concurrency Control

17 New Trendy Stuff