Software systems

Xin Wang

I. Overview

A. Analysing software systems

- Aspects to consider:
  - System high-level functions
  - System nodes
  - Types of data managed and processed
  - Data movement within the system
- Usually expressed with pictures

B. Modelling data (Database)

- Data is always stored, transformed and analysed
- **Abstract Data Model** used to understand process
- **Database theory** creates the Abstract Data Model
- Database theory considers:
  - Important entities in Database
  - **Attributes** of these entities
  - **Relationships** between these entities
- **Entity modelling** formally expresses database theory
- **Database systems** implements the Abstract Data Model

C. Moving data (Network)

- Process of data moving between nodes
- **Network models** defines the type of network structure
- **Network protocol** and **API** implements the model

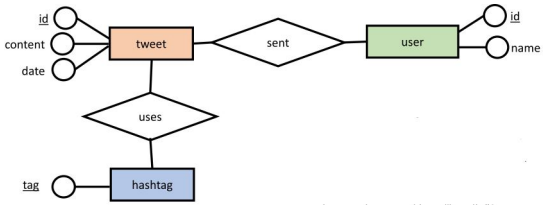
II. Entity Relation Modelling

- Creates **Entity Relationship Diagram**
- Establishing **relationships** in a given system:
  - **Entities**: Aspects within a given system
  - **Relationships**: How entities are related
  - **Attributes**: Properties of an entity or relationship
- Captures constraints and requirements on data
- Used as a guide to *implement* relations

Entity Sets	A set of distinguishable entities that all have the same set of properties (attributes). Could be physical things, events, conceptual, ... Normally correspond to nouns	Rectangle Student
Relationship	A relationship set describes how two or more entity sets are related to each other. Some times correspond to verbs : *owns, has, drives, ...* Entity sets can be involved in many relationship sets	Diamond manages
Attributes	Properties or attributes of an entity or relationship set. Underlined attributes are ***primary keys***.	Small circles id name

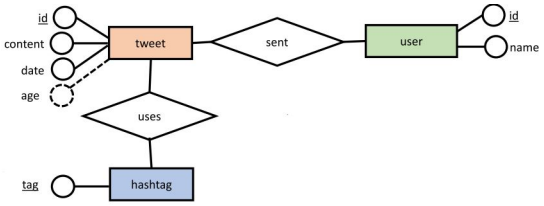
A. Primary keys

- An attribute that **uniquely identifies** an entity
- Properties:
  - There will never be two entities with the same key
  - Can contain **multiple** attributes if needed
  - Shown on ERD as underlined attributes
- Two types of primary keys:
  - **Natural keys**: Attributes from application data
  - **Surrogate keys**: *Invented* attributes

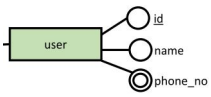


B. Complex attributes

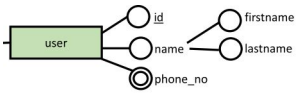
- **Computed attributes**: Calculated from other attributes



- **Multi-valued attributes**: Sets or lists of multiple values

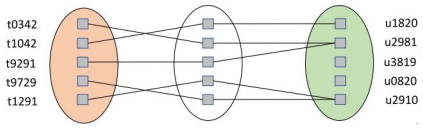


- **Composite attributes**: Properties that has sub-attributes



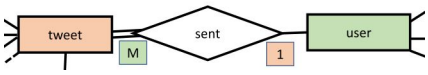
III. Relationships: Sets of relations

- Entity sets contain distinct entities
- **Relationships** contain sets of relations
- Each **relation** is a *pair of links* to an entity in the two entity sets

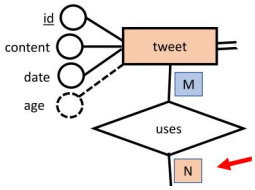


A. Relation constraints

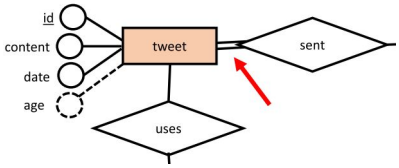
- **Cardinality constraint**: Number of times entity appears
  - One-to-one
  - One-to-many



  - Many-to-many

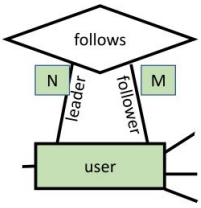


- **Total participation**: Entities **must** appear in relationships



B. Self relations

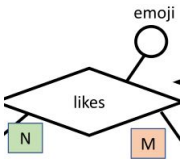
- Label the two connecting lines to show **roles**



- Cardinality constraints still apply

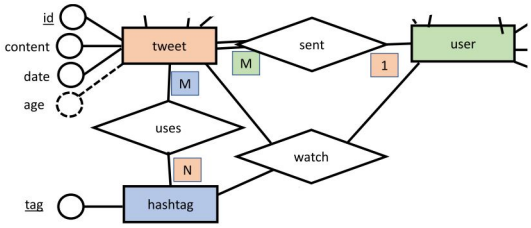
C. Relations with attributes

- Example: User can like a tweet with emojis



D. Three-way relationships

- Some relationships have more than two entity sets
- Example: User can *watch* for new retweets



IV. ERM and Relations

- Entities can be mapped into relations i.e. ERM
- ERM captures important aspects of the world
- With an ERM, work can be done on data e.g. SQL

A. Relations

ATTRIBUTES (the columns)
name:type

HEADING

BODY

title:string	year:int	length:int	genre:string
Gone with the Wind	1939	231	Drama
Star Wars	1977	124	SF
Wayne's World	1992	95	Comedy

TUPLES (the rows)

- Relation composition:
  - Relation Name
  - Heading:
    - Attributes:
      - Name
      - Type
  - Body:
    - Tuples
      - Attribute value i.e. name and value
- **Database**: Collection of relations
- **Relation Schema**: Relation name + Header
  - *movies(title:string, year:int, length:int, genre:string)*
- **Database Schema**: Collection of relation schema

B. ER diagrams → Relations → SQL

- Turning ER diagrams into concrete relations:
  - ER attributes → Relation attributes
  - ER entity → Relations
  - ER relationship sets → Relations or may disappear
- Relations are then turned into SQL

V. Structure Query Language (SQL) Introduction

- Domain specific language
- Defines, query and updates data
- Mostly portable and often performance tuning required
- Composed of **tokens**:
  - **Keywords**: CREATE, TABLE, SELECT ...etc
  - **Ordinary identifier**: *x*, *y*, *movies*
  - **Numbers**:3, 4.1, 1e-9
  - **Delimited identifiers**: "Peter, Mary"
- SQL are case-sensitive

A. Creating a table

```
CREATE TABLE movies (
    title varchar(100),
    year int,
    length int,
    genre char(16)
);
```

title:string	year:int	length:int	genre:string
Gone with the Wind	1939	231	Drama

B. Inserting data into a table

```
INSERT INTO movies
    VALUES (
    "Gone with the Wind",
    1939,
    231,
    "Drama"
);
```

title:string	year:int	length:int	genre:string
Gone with the Wind	1939	231	Drama

C. Extracting data from table

```
SELECT * from movies;
sqlite> select * from movies;

Gone with the Wind|1939|231|Drama
Star Wars|1977|124|SF
Wayne's World|1992|95|Comedy
```

title:string	year:int	length:int	genre:string
Gone with the Wind	1939	231	Drama
Star Wars	1977	124	SF
Wayne's World	1992	95	Comedy

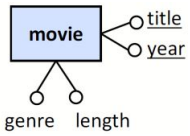
D. Extracting data from a table with filter

```
SELECT * from movies WHERE year = 1977;

Star Wars|1977|124|SF
```

VI. Mapping ERM to Relations

A. Entities and attributes

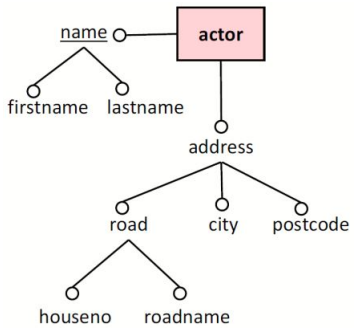
- Simple attributes:
  - Entity maps to relation with same attributes
  - Each entity in set becomes a row in relation
  - Entity primary key is relation primary key



movie(title, year, length, genre)

```
create table movie (
    title    varchar(120),
    year     int,
    length   int,
    genre    char(20),

    primary key (title, year)
)
```

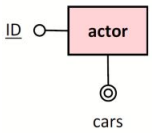
- Composite attributes:
  - Composite attributes become list of attributes



actor(firstname, lastname, houseno, roadname, city, postcode)

```
create table actor (
    firstname    varchar(30),
    lastname     varchar(30),
    houseno      int,
    roadname     varchar(30),
    city         varchar(40),
    postcode     varchar(10)

    primary key (firstname, lastname)

)
```

- Multi-valued attributes:
  - Attributes become own relation
  - Relation linked to original set with **foreign key**



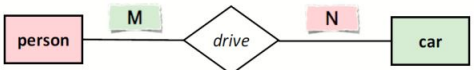
actor(ID, *otherattributes*)
actor_cars(actorID, carID, *otherattributes*)

```
create table actor_cars (
    actorID   int,
    carID     varchar(10),

    primary key (actorID, carID),
    foreign key (actorID) references actor.ID
)
```

- Derived attributes:
  - Not directly supported
  - Can be defined and used **within queries**

B. Relationship sets

- Many-to-many relationship:
  - Relation with two foreign keys
  - Entity primary key is relation primary key



person(ID, *otherattributes*)
car(regno, *otherattributes*)
drive(personID, regno,*otherattributes*)

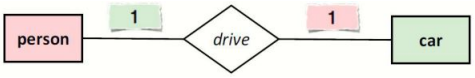
```
create table drive (
    personID  varchar(10),
    regno     varchar(12),
    primary key (personID, regno),
    foreign key (personID) references person.ID on delete cascade,
    foreign key (regno) references car.regno on delete cascade

)
```

- One-to-many relationship:
  - Primary key of one as foreign key in another

person(ID, *otherattributes*)
car(regno, personID, *otherattributes*)

```
create table car (
    regno     varchar(12),
    personID varchar(10),
    primary key (regno),
    foreign key (personID) references person.ID
)
```

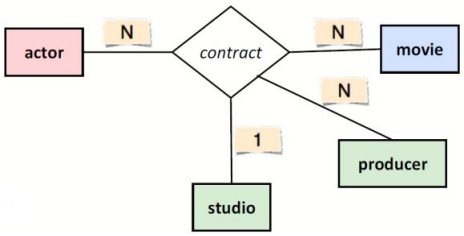
- One-to-one relationship:
  - Primary key of one as foreign key in another



person(ID, regno, *otherattributes*)
car(regno, *otherattributes*)

```
create table person (
    personID  varchar(12),
    regno     varchar(10),
    primary key (personID),
    foreign key (regno) references car.regno
)
```

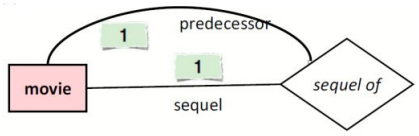
- Multi-way relationship:
  - Form relation with foreign keys to entity sets
  - Entity foreign keys form relation primary key



actor(ID, *otherattributes*)
movie(ID, *otherattributes*)
studio(ID, *otherattributes*)
producer(ID, *otherattributes*)
contract(aID, mID, sID, pID,*otherattrs*)

```
create table contract (
    actorID  int, movieID  int, studioID int, producerID int,
    primary key (actorID, movieID, producerID),
    plus foreign key declarations for actorID, movieID, studioID, producerID
)
```

- One-way relationship:
  - Each role is foreign key on entity set



movie(ID, *otherattributes*)
sequelof(originalID, sequelID, *otherattributes*)

```
create table sequelof (
    originalID  int,
    sequelID  int,
    primary key (originalID, sequelID),
    foreign key originalID references movie.ID,
    foreign key sequelID references movie.ID
)
```