# 1 Introduction

**Database design**

- How to design a DB schema
- Will use E-R model
- Translate it to relational DB

→ Issues in designing a DB schema. If using `E-R` model, as we will, then how to identify entities and their relations.

  – For e.g., if we have information about `emp_name, dob, salary, title, dep_name` then maybe we can leave it as one entity, `employee`; but if we add `dep_id, dep_budget`, then we can separate another entity out, `department`.
  – Another issue: how to translate the ER design into relation terms (schemas + constraints)

**DB Application Design Process**

- Design the DB schema
- Design programs for access/update
- Design security scheme
- Some issues:
  – *functional* requirements
  – Bad design: redundancy, incompleteness
  – Choose between (good) design choices

→ Here, we are talking about a complete software system that uses a DB at its heart

→ Program design: kind of language (driver issues), platform on which to develop, the frequency at which they can access (does this mean we should create more indexes?), what kind of n/w connection do we have

→ Security is not an after-thought; need to think of user account previliges at an early stage (no. of users, who needs to look at/insert/update what relations/data)

→ Redundancy: for e.g. we don't want to repeat the department name & budget for each employee in the employee relation.

→ Incompleteness: difficult to enter some information. For e.g., if customer name & address are stored in the account relation (with acc_id as primary key), then impossible to enter a customer's information who doesn't have an account (even if there's no primary key issue, will be entering null values).

→ Choosing between design choices: for e.g., the *sale* of a *product* by a *customer*–should it be represented as a relation b/w the two entities or should it be a new entity related to the other two.

# 2 The Entity-Relationship Model

**ER-model**

- Represents enterprise data & its *semantic*
- Conceptually model real world data meaning & relationships
- Models *entity sets*, *relationship sets*, *attributes*

⇒ **Semantic**: the meaning of the data should be represented as well (attribute and entity names, relationship definitions)

⇒ **Relationships** in ER are not the same as relations in relational algebra.

**Entity Sets**

- An "object" distinguishable from others
- Has properties that *define* it
- A set of properties uniquely *identifies* it
- Concrete or abstract
- **Entity Set:** set of similar entities (same properties)
- Individual entities are called *extensions* of the set
- Not always disjoint sets

⇒ An entity ('Othello') may be defined by the properties `101, 'Othello', 1.` `101` uniquely identifies the entity

⇒ Concrete (person, book) or abstract (loan, holiday)

⇒ So `Book` is an entity set as its elements (books) have the same properties

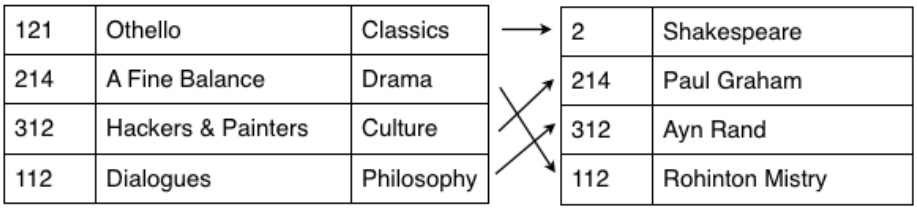
⇒ Not always disjoint entity sets: an entity may belong to both *customer* and *employer* sets

⇒ In terms of relational model, a single tuple is an entity, and a table is an entity set

**Attributes**

- Entity is represented by **attributes**
  - properties shared by each entity in a set
- An entity has **values** for attributes

⇒ Part of the design is selecting which properties to include as attributes (may or may not drop phone no., state, country, zip information for a customer)



| | | |
|---|---|---|
| 121 | Othello | Classics |
| 214 | A Fine Balance | Drama |
| 312 | Hackers & Painters | Culture |
| 112 | Dialogues | Philosophy |

| | |
|---|---|
| 2 | Shakespeare |
| 214 | Paul Graham |
| 312 | Ayn Rand |
| 112 | Rohinton Mistry |

Figure 1: Relationship Set `written-by`

**Relationship Sets**

- **Relationship**: Association among several (usually 2) entities
  - **Othello** $\xrightarrow{relationship}$ **Shakespeare**
- **Relationship Set**: relationships of same type
- Formally, a mathematical relation on $n \geq 2$ entity sets
  - If $E_1, E_2, ...., E_n$ are entity sets, then relationship set $R$ is a subset of

$$(e_1, e_2, ...., e_n) | e_1 \in E_1, e_2 \in E_2, ...., e_n \in E_n$$
$$\text{where, } (e_1, e_2, ..., e_n) \text{ is a relationship}$$

- $E_1, E_2, ...$ **participate** in $R$
- If $n = 2$, **binary** relationship set

⇒ **Relationship**: between entities; **Relationship Set**: between entity sets

⇒ Usually between two entity sets.

**Roles**

- The function represented by the relationship
- Usually implicit, since entity sets are distinct
  - Useful in **recursive** relationship set
  - For e.g., for `book` entity set, `references`
    - role(references) = (source, reference)
- **Descriptive attributes** for a relationship

⇒ If entities are distinct, it is easy to discern what the role is. For e.g., a relationship set between entity sets `book` and `author` is understood to be `written-by`

⇒ `references` is a relationship-set whose role needs to be said explicitly: a pair of (source, reference) entities taht both belong to the entity set `book`

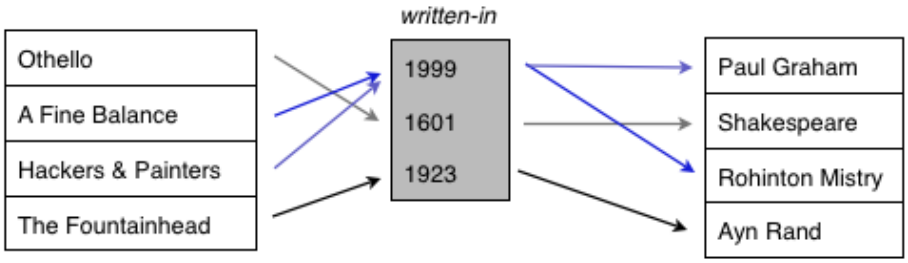



Figure 2: `written-in`: attribute of relationship set `written-by`

**Attributes**

- Takes values from **domain** or **value set**
- Entity = (attribute, value)
- *Composite* & *multi–valued* attributes
  - `name` ≡ `first_name, middle_name, last_name`
  - `phone_no` can be multi–valued
- grouping makes modeling cleaner
- *Derived* attributes
  - `total_books_written` = count from the `books` entity set
- `Null` values

⇒ Domain for `book_name` attribute may be all strings of a certain length

⇒ The set is of pairs, one for each attribute

⇒ Grouping: both in composite and multi–valued attributes

⇒ An attribute takes null values when entity does not have a value for it. = N/A, missing

# 3 Constraints

- Deals with the kind of limitations/constraints one can put on entities in an ER model.

**Mapping cardinalities**

- Entity set relationship mapping cardinalities
- Usually for binary relationships
  - One-to-one
  - One-to-many
  - Many-to-one
  - Many-to-many

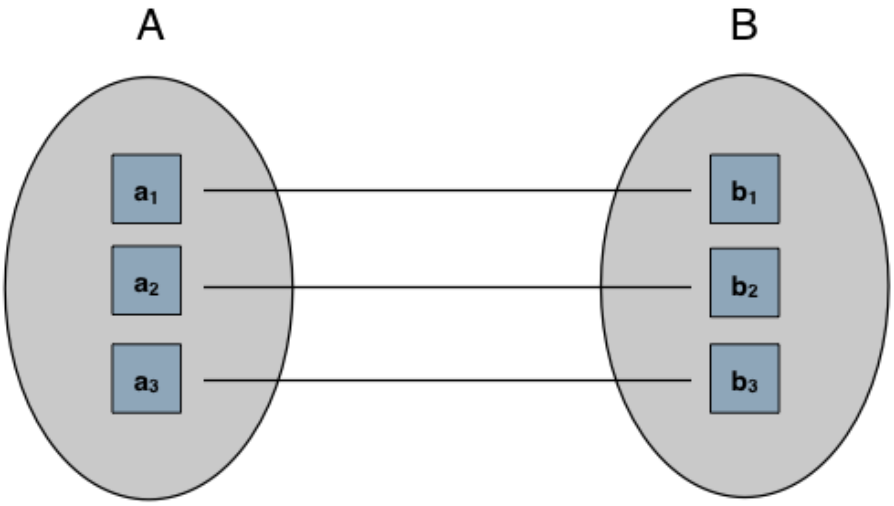



Figure 3: One-to-one mapping

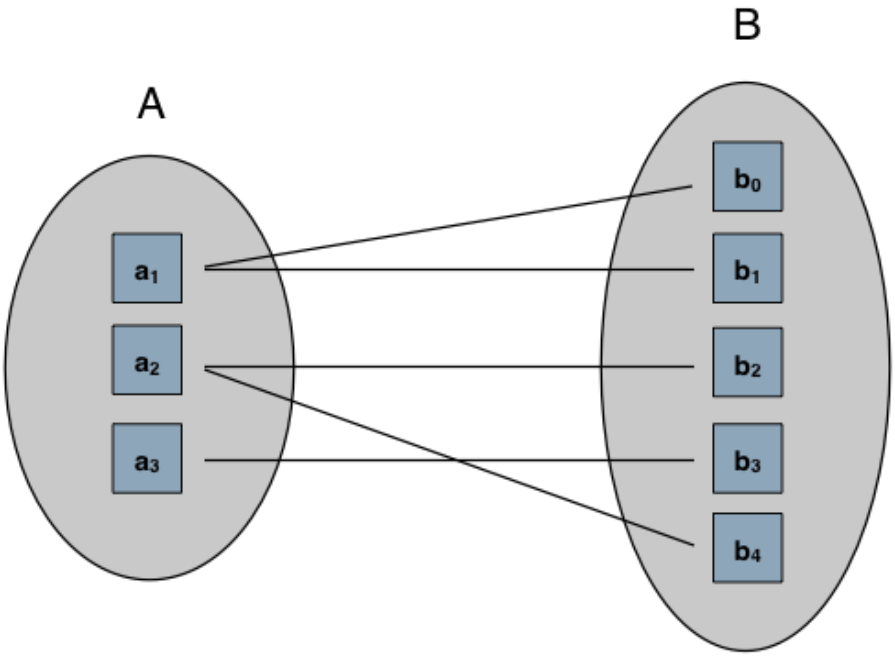



Figure 4: One-to-many mapping

- How many entities from one set can be related to an entity in another set.
- One-to-one: `customer` - `loan`
- One-to-many: `author` - `book`

**Keys**

- **Super keys**: set of attributes that uniquely identify
- **Candidate keys**: minimal super keys
- Primary key: a candidate key that is chosen
  - unique, rarely changed
- **Super-keys**: set of attributes that uniquely identify an entity in an entity set
  - (`book_id`, `book_name`)
- Candidate keys: no subset is a super key
- generated id's are good choices except in mergers.

**Keys for relationships**

- If $E_1$ and $E_2$ are related by $R$, then

  $$primary\_key(E_1) \cup primary\_key(E_2) \cup a_1, a_2, \ldots, a_m$$

  is the super-key for $R$
- Primary key for $R$ depends on mapping cardinality:
  - many-to-many: $primary\_key(E_1) \cup primary\_key(E_2)$
  - one-to-many: $primary\_key(E_2)$
  - many-to-one: $primary\_key(E_1)$
  - one-to-one: $primary\_key(E_1)$ or $primary\_key(E_2)$
- Just like primary keys identify unique entities, need a mechanism for identifying relationships
- For primary keys, the entity set which has more ambiguity, its primary key is chosen as the key for the relationship.