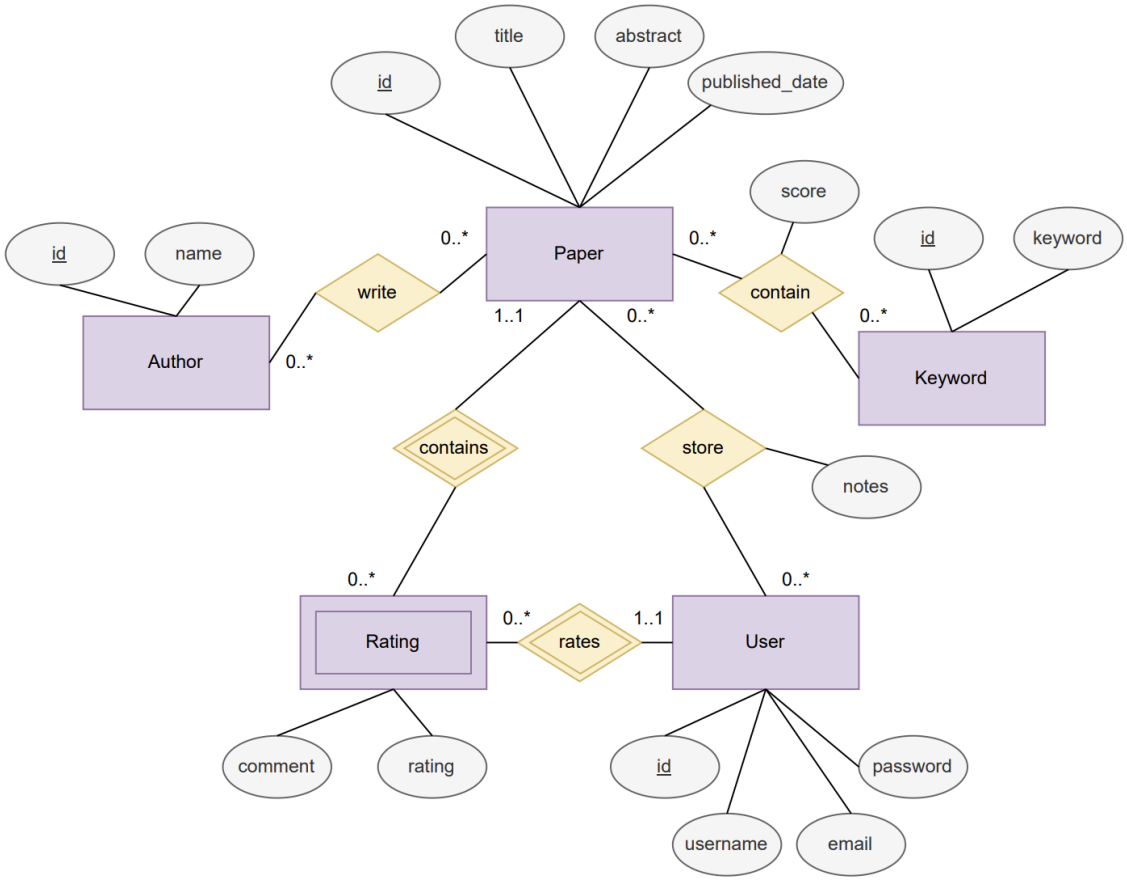
Database Design

1. ER diagram



2. Assumptions

We model the following as entities:
Author: an author represents an individual that writes papers
Paper: this entity encapsulates information about a research paper (such as its title, abstract, and published date)
Keyword: a keyword describes a topic a research paper contains (where the association of the keyword to the paper is given by a similarity score)
Rating: ratings capture comments and a numerical score left by users on research papers; this is a weak entity because it relies on the ID of the paper and the ID of the user to be uniquely identified
User: a user represents information about an account (such as an individual's username, password, and email address); a user can also store notes on papers

We make the following assumptions about the cardinality of our relationships:
Write: Authors have a many-to-many relationship with papers (i.e., multiple authors can write a paper, and a paper can be written by multiple authors).
Contain: Papers have a many-to-many relationship with keywords (i.e., a paper can contain multiple keywords, and a keyword can be associated with multiple papers).
Store: Users have a many-to-many relationship with papers (i.e., users can store multiple papers, and a paper can be stored by multiple users).
Rates: Users have a one-to-many relationship with ratings (i.e., a user can rate multiple papers, but each rating can only belong to one paper).
Contains: Papers have a one-to-many relationship with ratings (i.e., a paper can contain multiple ratings, but each rating can only belong to one paper).

3. Normalization

Based on the ER diagram, the functional dependencies are as follows:

FD = {
    User.id → { User.username, User.email, User.password },
    Keyword.id → { Keyword.keyword },
    Paper.id → { Paper.title, Paper.abstract, Paper.published_date },
    Author.id → { Author.name },
    (Paper.id, User.id) → { Rating.rating, Rating.comment, Rating.notes },
    (Paper.id, Keyword.id) → { PaperKeyword.score }
}

We can normalize the database by applying 3NF decomposition because we have either the LHS or RHS to be candidate keys for every single functional dependency in the set.

The candidate keys are found as follows:

LEFT → { User.id, Keyword.id, Paper.id, Author.id }
MIDDLE → { None }
RIGHT → {
        User.username, User.email, User.password, Keyword.keyword,
        Paper.title, Paper.abstract, Paper.published_date, Author.name,
        Rating.rating, Rating.comment, Rating.notes, PaperKeyword.score
    }
NONE → { None }

The attribute closure of the LEFT + NONE attributes includes all the attributes in the functional dependency set FD. We get (User.id, Keyword.id, Paper.id, Author.id) as the candidate key.

FD is also the minimal basis itself. [after taking the steps of (1) making sure that RHS of every functional dependency in FD is a singleton; (2) removing redundant attributes from LHS; (3) removing unnecessary functional dependencies.]

Including the candidate key, we obtain the final result of 3NF decomposition of our database schema as shown in the current relational schema in Section 4.2.

We have chosen to use 3NF instead of BCNF because of its simplicity and ease of design. Although we can achieve normalization using BCNF because the requirement (for every non-trivial functional dependency (X → Y), X must be a superkey) is fulfilled, 3NF is easier to achieve and understand compared to BCNF.

4. Logical Design (Relational Schema)

4.1 Original Schema:

```
Paper (
        id: INT [PK],
        title: VARCHAR(4096),
        abstract: VARCHAR(65536),
        published_date: VARCHAR(16)
),
Keyword (
        id: INT [PK],
        keyword: VARCHAR(255)
),
PaperKeyword (
        paperId: INT [PK] [FK to Paper.id],
        keywordId: INT [PK][FK to Keyword.id],
        score: Decimal
),
User (
        id: INT [PK],
        username: VARCHAR(255),
        email: VARCHAR(255),
        password: VARCHAR(255)
),
UserPaper (
        userId: INT [PK] [FK to User.id],
        paperId: INT [PK] [FK to Paper.id],
        notes: VARCHAR(65536)
),
Author (
        id: INT [PK],
        name: VARCHAR(255)
),
PaperAuthor (
        paperId: INT [FK to Paper.id],
        authorId: INT [FK to Author.id]
),
Rating (
        paperId: INT [PK] [FK to Paper.id],
        userId: INT [FK to User.id],
        rating: INT,
        comment: VARCHAR(65536)
)
```

4.2 Schema after Normalization:

Paper (
    id: INT [PK],
    title: VARCHAR(4096),
    abstract: VARCHAR(65536),
    published_date: VARCHAR(16)
),
Keyword (
    id: INT [PK],
    keyword: VARCHAR(255)
),
PaperKeyword (
    paperId: INT [PK] [FK to Paper.id],
    keywordId: INT [PK][FK to Keyword.id],
    score: Decimal
),
User (
    id: INT [PK],
    username: VARCHAR(255),
    email: VARCHAR(255),
    password: VARCHAR(255)
),
UserPaper (
    userId: INT [PK] [FK to User.id],
    paperId: INT [PK] [FK to Paper.id],
    notes: VARCHAR(65536),
    rating: INT,
    comment: VARCHAR(65536)
),
Author (
    id: INT [PK],
    name: VARCHAR(255)
),
PaperAuthor (
    paperId: INT [FK to Paper.id],
    authorId: INT [FK to Author.id]
)