CS3101 - Databases Exam

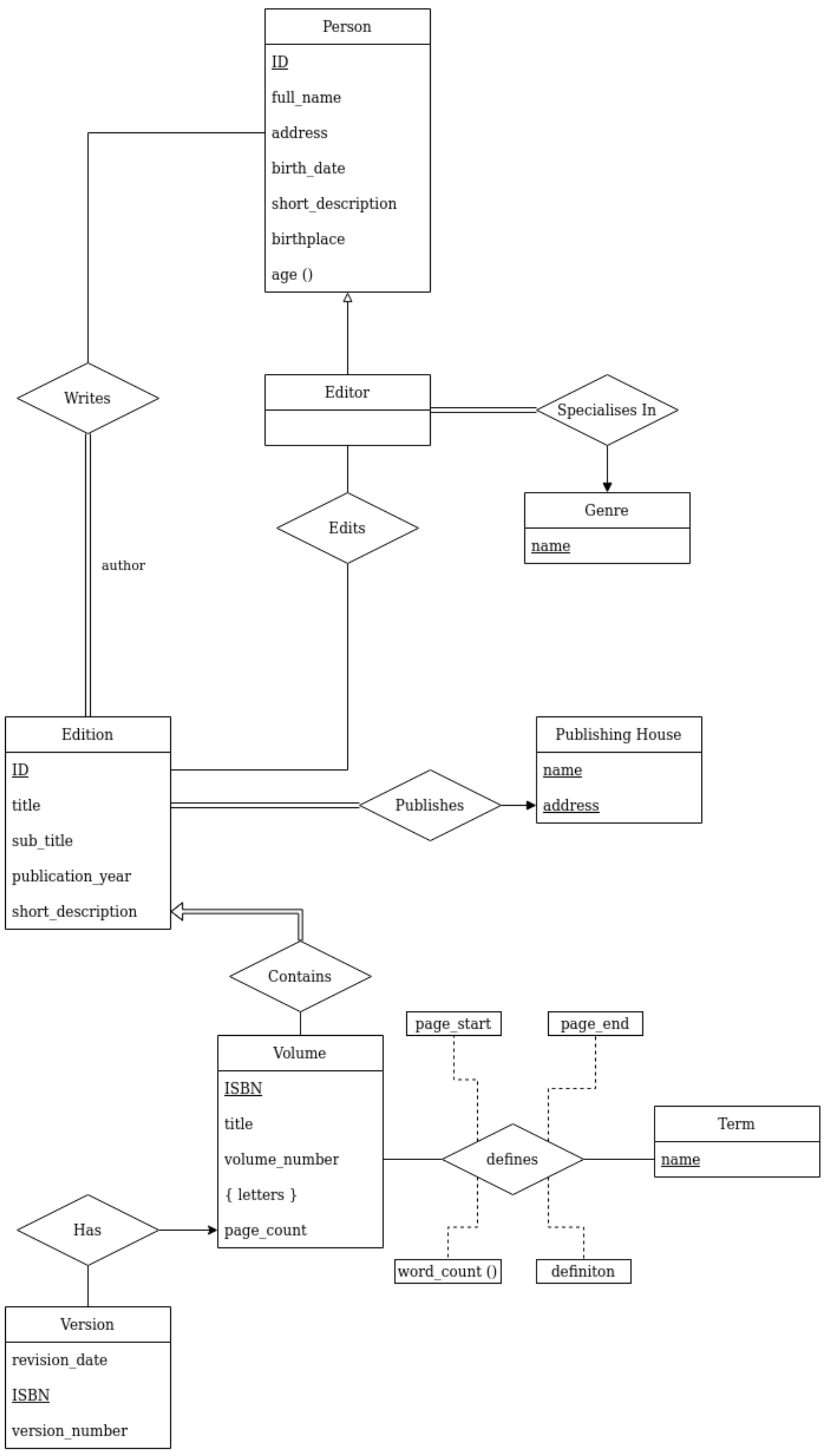
200007413

22 Apr 2023

1 TODO

1.a

For this scenario, I assumed that the ISBN are unique across all volumes, editions, and versions.


Person
ID
full_name
address
birth_date
short_description
birthplace
age ()
Writes
Editor
Specialises In
Genre
name
Edits
author
Edition
ID
title
sub_title
publication_year
short_description
Publishes
Publishing House
name
address
Contains
page_start
page_end
Volume
ISBN
title
volume_number
{ letters }
page_count
defines
Term
name
Has
word_count ()
definiton
Version
revision_date
ISBN
version_number

1.b

person(id, full_name, address, birth_date, short_description, birthplace)

editor_specialty(person_id*, genre_name)

writes(person_id*, edition_id*)

edits(person_id*, edition_id*)

edition(id, title, sub_title, publication_year, short_description)

publishing_house(name, address)

edition_publisher(publisher_name, address)

1.c

1.c.1

```
SELECT id, publication_year FROM edition
```

1.c.2

```
SELECT e.id, COUNT(v.ISBN) AS count FROM edition as e LEFT JOIN volume as v ON
v.edition_id = e.id WHERE e.publication_year < 1850 GROUP BY e.edition_id
```

1.c.3

```
SELECT start_page, end_page FROM term_definition AS td LEFT JOIN volume AS v ON
td.volume_ISBN = v.ISBN WHERE td.name = "abacus" AND exists (SELECT * FROM edititon
as e WHERE e.id = v.edition_id AND e.id = 1)
```

2

2.a

2.a.1

$$
\begin{aligned}
\text{student} = \{ \quad & \text{id} \rightarrow \text{name} \\
\} \\
\text{tutors} = \{ \quad & \text{tutor\_id} \rightarrow \text{tutor\_email} \\
\} \\
\text{practical\_grade} = \{ \quad & \text{student\_id,module\_id,practical\_number} \rightarrow \text{grade} \\
\}
\end{aligned}
$$

2.a.2

At the given exact moment, there would be a functional dependency between module_id and practical_number, as each value of module_id refers to a unique practical_number.

However, this functional dependency will not exist once a second practical for a module is added, as then one module is not guaranteed to reference a unique practical_id.

2.b

2.b.1

Yes, the *tutors* relation is in first normal form, as all it's attributes are atomic. Each tuple refers to a single student using student_id, and refers to a single staff member using tutor_id, with a single email.

2.b.2

Yes, because *tutors* is in first normal form, and each of the attributes are either candidate keys or full functionally dependent on a candidate key.

Both tutor_id and student_id are required to uniquely identify the relationship, and so are candidate keys, and then tutor_email is functionally dependent on the tutor_id.

2.b.3

Under the assumption that tutors do not have more than one email address, *tutors* is not in third normal form.

Whilst it is in second normal form, and tutor_id and student_id are part of the super-key, tutor_id itself is not a superkey, and there is a functional dependency from tutor_id to the non-candidate-key attribute tutor_email. Additionally, tutor_email-tutor_id = tutor_email, which is itself not in a candiate key.

2.b.4

This is not in Boyce-Codd normal form (BCNF) as it is not in third-normal form.

The condition for BCNF is the same as 3NF in that every functional dependency $\alpha \rightarrow \beta$ must either be trivial or be dependent on a superkey, with the added restriction that $\beta - \alpha$ being in a candidate key is not sufficient. As the tutor_id $\rightarrow$ tutor_email already doesn't meet the less strict 3NF conditions, it does not meet BCNF.

2.c

Artificial primary keys are those which do not relate to any real data, but are added to ensure that there is a unique identifier for each entry, instead of relying upon the three natural candidate keys to uniquely identify each tuple.

The advantages to using the artificial ID value is that you can create queries using just that identifier, instead of requiring that someone know the module_id, student_id, and practical_number. Practically, this could allow students to access their practical grades just with the practical_grade ID.

The downside to using the ID attribute is it will add redundant information, and prevent the schema from being in Boyce-Codd normal form. This is because each entry is already unqiuely identified using the three candidate keys, and so the ID is functionally dependent on these and visa-versa.

This redundancy would add additional space requirements for storing the table, as a new not-strictly required column is added to each tuple.

Another risk from artificial keys is with the chance for duplicate entries. If ID is set as the sole unique superkey, then more than one entry could point to the same "useful" data, but just with a new ID, practically adding duplicate data. Without artificial keys, only unique combinations of student_id,module_id, and practical_number can be added.

2.d TODO

3

3.a

A *mediumblob*/*longblob* type may be appropriate for *user.picture* for storing the raw picture data itself. This would be a large data type, and would store the picture in the database.

Another appropriate type could be a *varchar*, which would contain the URL to that user's picture.

The benefit of a blob data type is that no additional calls need to be made to retrieve the users profile picture, as the call itself returns it's raw data.

However, this means the database will be much larger than having a varchar containing a url.

Another advantage of the varchar URL method is the image can be stored at the URL as an image file. Storing as a blob ignores the file tyoe, meaning the program accessing the data must interpret the image properly. Storing the image as a file can prevent the need to interpret raw data in this way.

3.b

3.b.1

$$\Pi_{\text{username}}(\sigma_{\text{rating}>1800}(\text{user}))$$

3.b.2

$\Pi_{\text{p1\_username}}(\text{game} \bowtie \sigma_{\text{name}=\text{"Sammy Jacobs Cup"}}(\text{tournament}))$

3.c

SELECT UNIQUE p1_username FROM game WHERE EXISTS (SELECT tournament_id FROM tournament WHERE tournament_id = id AND tournament.name = "Sammy Jacobs Cup")

3.d

3.d.1

manutdlover

joker77,manudlover,tobym

alan34,alexr,bigbob joker77,kingkhalid,md216 mudlover,ohio_alice,pkg113 tobym

3.d.2

Increasing the value of N would allow for more items to be added before the tree needs to be reorganized.

An issue with this is that the graph would become more uneven, thus reducing the efficiency when searching.

3.d.3

The advantage of a hash index is that exact match queries are much faster. An example of a query which would run faster for a hash index would be:

SELECT * FROM user WHERE username = "bob"

Queries which have non-exact matching criteria would be faster for B+ indexes. For example:

SELECT * FROM user WHERE username LIKE "bo%son"