The Technical University of Denmark

Database Systems, 02170

Researcher Database

Authors:
s146996 - Christian D. Glissov

Supervisors:
Anne Elisabeth Haxthausen

June 4, 2021, Kongens Lyngby

DTU Compute
Department of Applied Mathematics and Computer Science

Table of Contents

1 About the database

From the exam handout the following assumptions of the database to be created is shown below.

- An **institution** has a worldwide unique name (iname) and belongs to a single country (given by its unique name country). An institution may (optionally) have researchers employed and host projects.
- A **researcher** has a worldwide unique id (rid) and a name (rname). A researcher is employed in a single institution. Information about freelance researchers (not employed at an institution) should not be included in this database. A researcher may (optionally) participate in projects and author articles.
- An **article** has a worldwide unique id (aid), a name (aname), and a publication year (year). Furthermore, an article has one or several of the researchers as authors.
- A **project** has a worldwide unique name (pname), and has a single host institution. A project may have researchers as participants, but may also be without participants.

Please note that throughout the report I aimed to be as consistent as possible with making tables/relations/entity sets written in bold and attributes in cursive. Tables and attributes use lower case naming as I find using upper case notation to be confusing when coding.

2 Conceptual Design

> Show an Entity-Relationship (E-R) Diagram for the domain of the database using the Textbook's Adapted UML Notation. Remember to show (1) strong and weak entity sets with their names, attributes and primary keys, and (2) relationship sets with their names, attributes (if any), cardinalities, and total/partial participation.

Below the Entity-Relationship diagram of the research database can be seen.

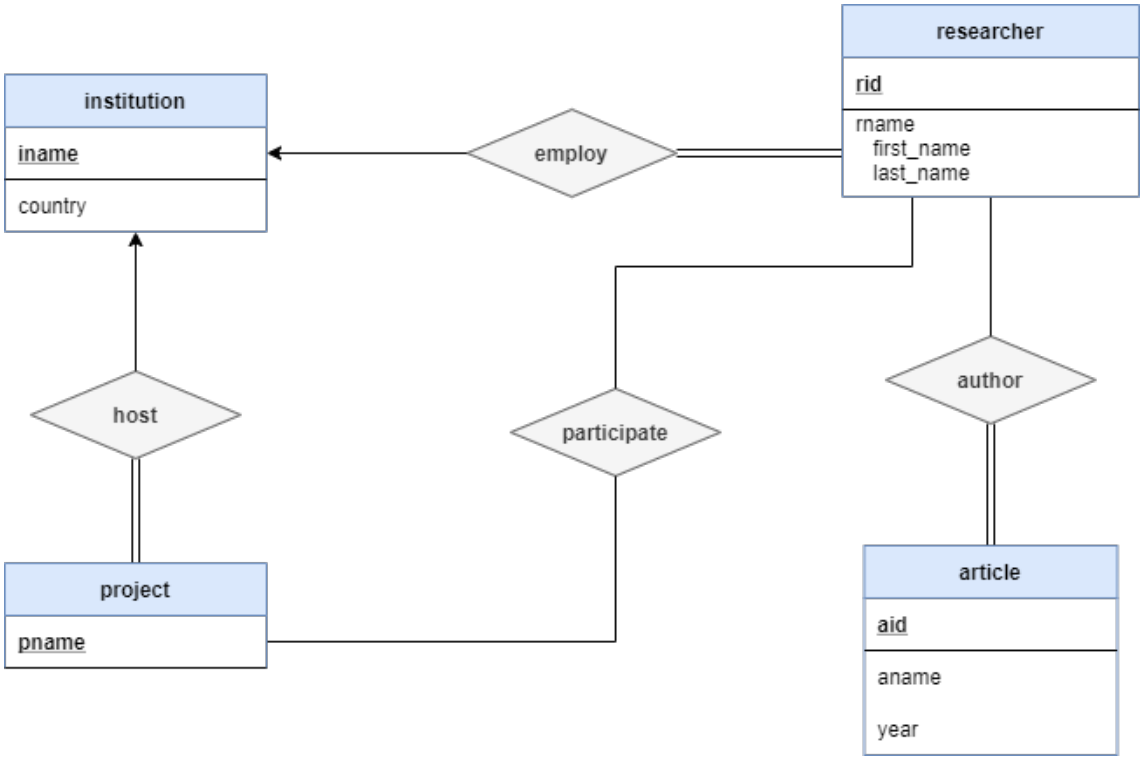

Figure 1: E-R diagram of the research database, *ResearchDatabase*

Discuss any choices made, e.g., why the cardinalities and participation constraints are chosen as they are etc.

I will now discus the choices made in the conceptual design for each entity set.

- **institution**: The entity set **institution** consists of two attritubes, *iname* and *country*. Where *iname* acts as the primary key, as it is worldwide unique. As an institution may or may not employ researchers and host projects, there is a partial participation between **institution** and the relationship **employ** and **host**.

- **researcher**: The entity set **researcher** has two attributes, a unique id acting as a primary key, *rid*, and a name, *rname*. In this case I chose *rname* to be a composite value consisting of a first and last name to help with searching for the author. First name is necessary as a researcher might have the same last name, but a different first name. Freelance researches will not be included in the database and this means the participation between the relationship **employ** and **researcher** is total. Furthermore there is a partial participation between the relationships **participate** and **author** as a researcher might not participate in a project or author an article.

- **article**: The entity set **article** has three attributes a unique id, *aid*, acting as

a primary key, a name, *aname* and a publication year, *year*. An article must have a researcher as an author, hence the total participation between **author** and **article**.

- **project**: Finally, the entity set **project** has a single attribute, a worldwide unique name, *pname*, acting as a primary key. A project does not need to have researchers as participants, this implies the partial participation between **project** and **participate**, but it does need an institution to host it, hence the total participation between **host** and **project**.

Now the cardinalities will be discussed.

- **institution** to **project**: As an institution can host many projects, but a project may only be hosted by a single institution, the relationship set between **institution** and **project** is One-to-Many.
- **institution** to **researcher**: As an institution can employ optionally many researchers, but each researcher is only employed by a single institution, the relationship set between **institution** and **researcher** is One-to-Many.
- **researcher** to **project**: As a researcher can participate in many projects and a project can be participated by many researchers the relationship set between **project** and **researcher** is Many-to-Many.
- **researcher** to **article**: A researcher can author one or several articles and an article can have one or several of the researchers as authors the relationship set between **researcher** and **article** is Many-to-Many.

3 Logical Design

Convert your conceptual design into a logical design. You must follow the method described in the course slides and book (Chapter 7): Show relation schemas including specification of foreign keys and referential actions. Show also the corresponding database schema diagram in the Text book notation.

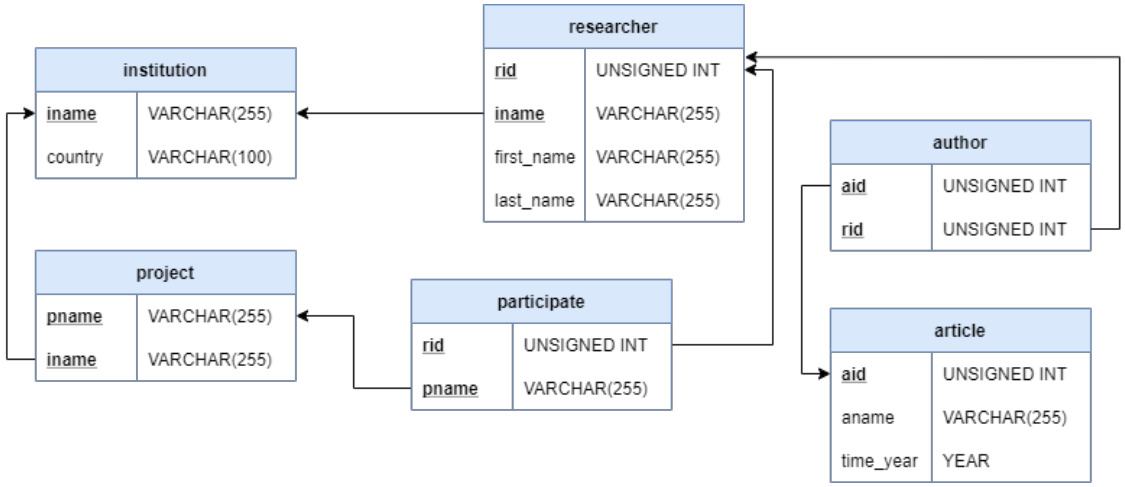

Figure 2: Database schema diagram of *ResearchDatabase*

institution(**iname**, country)

researcher(**rid**, **iname**, first_name, last_name)
FOREIGN KEY **iname** REFERENCES institution(**iname**)
ON UPDATE CASCADE

project(**pname**, **iname**)
FOREIGN KEY **iname** REFERENCES institution(**iname**)
ON UPDATE CASCADE

article(**aid**, aname, year)

participate(**pname**, **rid**)
FOREIGN KEY **pname** REFERENCES project(**pname**)
ON DELETE CASCADE
ON UPDATE CASCADE
FOREIGN KEY **rid** REFERENCES researcher(**rid**)
ON UPDATE CASCADE

author(**aid**, **rid**)
FOREIGN KEY **aid** REFERENCES article(**aid**)
ON DELETE CASCADE
ON UPDATE CASCADE
FOREIGN KEY **rid** REFERENCES researcher(**rid**)
ON UPDATE CASCADE

Figure 3: Relation schemes of *ResearchDatabase* including foreign key references and referential actions

Discuss any choices made.

- **The database schema diagram**: The conceptual design from Figure 1 has been depicted in the database schema diagram seen in Figure 2. Primary keys are shown underlined. Foreign keys are also underlined, but to differentiate arrows are used to indicate foreign-key constraints from the foreign-key attributes of the referencing relation to the primary key of the referenced relation. The foreign key *iname* is not required to satisfy uniqueness in the **researcher** and **project** relations, but I chose it as a foreign key to satisfy

relational-integrity, assuring that a researcher or project is assigned an institution that is in the database. I also included attribute data-types to the right of the attributes. 255 character limits are mainly used as it maximises the 8-bit count usage, without requiring another byte above the 255 characters. One could argue that id's such as *rid* can contain characters and are probably quite unique with choice of characters depending on the institution, but for simplicity these are chosen to only be unsigned integers with no value limit and unsigned to not allow negative values of the ids. The relationships **participate** and **author** from the conceptual design are junction relations in the database schema diagram due to the Many-to-Many relation. To convert it to a relation schema two One-to-Many entities are used creating the **participate** and **author** relations defined by foreign key attributes from the primary keys of **researcher**, **article** and **project**. For the Many-to-One relationships a new schema is avoided by adding *iname* to **researcher** and **project**. Finally, the attribute *year* from Figure 1 is renamed to *time_year* to not conflict with MySQL syntax.

- **The relation schema**: From Figure 1 we depict the relational schema. Primary keys and foreign keys are shown in bold and underlined. With foreign keys having a referential action assigned as well. Strong entity sets become a relation schema with the same attributes and primary key for the entity set. For weak entities a relation schema will additionally include the primary key of the identifying entity set, but also a foreign key attribute to satisfy uniqueness in conjunction with the primary key. The foreign key attribute will have referential actions to satisfy referential-integrity. For **researcher** and **project** cascade on update is chosen, this allows the relations containing the foreign key *iname* to update the name in case it is is changed. I assume institutes can not be deleted, one reason is because if an institute is deleted the researchers might be deleted hence an article may have no valid authors, not satisfying the total participation of articles. Another reason is if we delete an institute and set the foreign keys of the deleted values to null, then researchers might end up not being assigned an institute, once again not satisfying the assumptions. The referential actions for **participate** are cascaded on delete and update for the project. If the project name changes for *pname*, we want to update it in **participate** and likewise, as it is optional for researchers to have project, we can delete projects from **project** and cascade it to **participate**. For **rid** we do not allow deletion as **article** need to satisfy the total participation, we only allow update of the researchers id, which has to be worldwide unique based on the assumptions.

4 Normalization

> For each relation schema in your logical design, state its normal form and explain what makes it be in the given normal form.

Please notice, that the same reasoning applies to a lot of the relations when normalising, so I will omit an explanation for each of the relations as it would be very cumbersome.

4.1 First Normal Form (1NF)

For the first normal form the following must be satisfied

1. Each cell in the table has a single value.
2. There are no duplicate rows.
3. All cells in a column have values from the same domain.
4. Each column has a unique attribute name.
5. Order of the rows and columns must not have any significance.

Let us take the relation **researcher** as an example. There are no multi-valued attributes, they are all atomic. There is also no duplicate rows, each *rid* is worldwide unique. There are no values from different domains within a column. Finally, the order of the values do not matter. Hence **researcher** is in 1NF. The same reasoning can be made for the other relation schemes. Each relation schema is therefore 1NF.

4.2 Second Normal Form (2NF)

For the relation to be in 2NF the following must be satisfied

1. In order for a relation schema to be in Second Normal Form it must first be 1NF.
2. Each non primary key attribute must depend on the entire primary key.
3. In the special case there is only one attribute in the primary key or that the primary key consists of all attributes in the relation and it satisfies (1), then the relation schema is also 2NF.

All relations satisfy 1NF. As an example let us look at **researcher** again. Here we only have one candidate key *rid*, as neither *first_name*, *last_name* or *iname* depends on each other. Two researchers might have the same first or last name and institute does not define the name of the researchers and vice versa. As there is only one candidate key the functional dependence is defined as

$$\{rid\} \rightarrow \{first\_name, last\_name, iname\}$$

With *rid* as the determinant and the other attributes as the dependants. As there is also only one primary key in **project, institution, article**, and all non-primary key attributes only depend on the whole primary key, the relations satisfy 2NF. For **participate** and **author** the foreign keys form the primary key, as they do not consist of more attributes than the foreign keys they satisfy the 3. special condition. All relations are therefore 2NF.

4.3 Third Normal Form (3NF)

For the relation to be in 3NF the following must be satisfied

1. In order for a relation schema to be in Third Normal Form it must first be 2NF.

2. It must satisfy, that each non-primary key attribute must depend directly on the entire primary key. This implies that no non-primary attribute can depend on some other non-primary attribute or alternatively on parts of the key.

3. Each non-primary key attribute must depend directly on the entire primary key. It must not depend transitively via other attributes.

The first step is satisfied, all relations are 2NF. There are only one candidate key for the relations. It is not possible to write up a non-trivial transitively dependency for any of the tables, hence all relations are in 3NF. The relations are dependent on the entire primary key. An example can be shown using the **researcher** relation, let us say the following functional dependency is given

$$rid \rightarrow \{first\_name, last\_name\}, \quad \{first\_name, last\_name\} \rightarrow iname$$

But this dependency is illegal, as there might be two tuples with the same first and last name but with a different institute, other FDs can be tried, but the result is the same, all the attributes are dependent on the entire primary key *rid*.

4.4 Boyce-Codd Normal Form (BCNF)

For the relation to be in BCNF the following must be satisfied

1. In order for a relation schema to be in Boyce-Codd Normal Form it must first be 3NF.

2. For each non-trivial left irreducible functional dependency, the determinant is a candidate key. If there is only one candidate key, then 3NF and BCNF are the same.

Once again, we only have one candidate key for each of the relations and all of the relations are in BCNF. As an example we will look at the relation **researcher** again. Here the only determinant as candidate key is *rid*, clearly it is also left-irreducible. For **participate** and **author** which contain two foreign keys, only one candidate key satisfy uniqueness, the composite of the two foreign keys, hence also BCNF.

> If some tables are not in at least 3NF: Then check whether this is due to some problems in your conceptual model or its conversion to a logical design. If so go back and fix the problems, otherwise normalise the tables directly to 3NF.

All tables are in at least 3NF.

5 Implementation

> Create a MariaDB database with tables implementing the logical design (as achieved after possible revisions in task 3) using SQL statements CREATE DATABASE and CREATE TABLE.

We now implement the database from Figure 2. The code can be seen in the attached script ResearchDatabase.sql. Otherwise a snippet of the code can be seen below:

```
DROP DATABASE IF EXISTS ResearchDatabase;
CREATE DATABASE ResearchDatabase;
USE ResearchDatabase;
```

We also assure to drop the tables if they exist, this is omitted here in the report, but can be found in the script. An example of creating a table is seen here, this is for the **institution** table:

```
CREATE TABLE institution
  (iname       VARCHAR(255) NOT NULL,
```

```
  country     VARCHAR(100) NOT NULL,
  PRIMARY KEY(iname)
);
```

6 Indexing

> Which primary and secondary indices will automatically be generated by MariaDB for your tables?

MariaDB will automatically generate a primary index for each of the primary keys and a secondary index for each of the foreign keys. Each primary and sequential index in MariaDB for this database is ordered and dense. This can be seen from the output of the following pseudo-code:

```
SHOW INDEX FROM table_name;
```

All tables have a cardinality equal to the number of elements, meaning a search key is assigned to each tuple. For foreign keys with duplicate values the search key is not unique. The collation indicates that all the indices are sorted in an ascending order. For the **institution** table the primary key is used as index, *iname*, the same applies to **project** with *pname*, **article** with *aid* and **researcher** with *rid* as they are all primary keys. The foreign key *iname* in **project** and **researcher** uses a non-unique index with the column values, it allows for more efficient foreign key checks as it removes the necessity of scanning the entire table. Finally, **author** and **participate** uses a unique composite search key consisting of the two foreign keys in each table, *aid, rid* and *pname, rid* respectively. It also uses another index, as there is no index to reuse, a non-composite and non-unique index is also created for one of the foreign keys, in this case the attribute *rid* is chosen. To organise the indices a BTREE index file organisation is used to improve efficiency of different queries.

7 Database Instance

> Populate the tables with data using SQL **INSERT** statements.

The statements can be seen in ResearchDatabase.sql.

> List instances of all tables using SQL **SELECT * FROM** table statements and show the output in the report.

Below the code and output can be seen. The code can also be found in RDB-SCRIPT.SQL.

```
SELECT * FROM institution;
```

iname	country
Århus University	Denmark
CALTECH	USA
DTU	Denmark
ITU	Denmark
KU	Denmark
Oxford University	England
University of Amsterdam	Netherlands

```
SELECT * FROM project;
```

pname	iname
Chinese Military Research	DTU
Master Thesis	DTU
Sustainable Development Goals	DTU
Confidential Project	KU
Project Enigma	Oxford University

```
SELECT * FROM article;
```

aid	aname	time_year
1	Recurrent Flow Networks: A Recurrent Latent V...	2019
2	The RAISE Specification Language	1992
3	Missing importance-weighted autoencoders with...	2020
4	Auto-Encoding Variational Bayes	2014

```
SELECT * FROM researcher;
```

rid	iname	first_name	last_name
1	DTU	Christian	Glissov
2	DTU	Anna	Lia
3	DTU	Anne	Haxthausen
4	Oxford University	Stephen	Hawking
5	Oxford University	Alan	Turing
6	University of Amsterdam	Diederik	Kingma
7	Århus University	Erik	Eriksen

```
SELECT * FROM author;
```

aid	rid
1	1
2	3
3	1
3	5
4	6
4	7

```
SELECT * FROM participate;
```

pname	rid
Confidential Project	4
Confidential Project	5
Project Enigma	5
Sustainable Development Goals	1
Sustainable Development Goals	2
Sustainable Development Goals	5

8 SQL Data Queries

State an SQL statement which returns a table that for each project has a row with its name (pname) and the number of distinct articles that are authored by researchers participating in that project.

To do this we first get the values from project. We then left outer join the participate values with the selected values. This gives us the corresponding *pname* and the researcher on the given *pname*, if no researcher is on the project, then it is simply returning a null value. We now do another join with the **author** table. This will allow us to get the article ID the corresponding researcher is assigned to, if a researcher is assigned no articles or there is no researcher on the project, the value is null. Finally, we can group by *pname* and count the distinct values, this will give us the total number of distinct articles written by the researchers participating in a particular project. The DISTINCT is probably not necessary as the same researcher can not be on the same project or article twice, however, I still included it for readability.

```
SELECT pname, COUNT(DISTINCT aid) as NDistinctArticles
FROM project
NATURAL LEFT OUTER JOIN participate
NATURAL LEFT OUTER JOIN author
GROUP BY pname;
```

Show also the result of the query for the database instance made above.

The query can be seen below

pname	NDistinctArticles
Chinese Military Research	0
Confidential Project	1
Master Thesis	0
Project Enigma	1
Sustainable Development Goals	2

Comparing with the database instances we see there is three researchers on the *Sustainable Development Goals* (SDG) project, but only researcher 5 and 1 are assigned as author on 1 article each. The query then give the correct output of 2 for the SDG project. No researcher is assigned the *Chinese Military Research* project, therefore 0 distinct articles. Same reasoning is applied for the other projects.

9 SQL Table Modifications

Define an SQL function and give an illustrative example of its use in a query. Show also the result of the query for the database instance made above.

The function below returns the productivity output of the researchers of an institute. The productivity output is defined as the sum of the total number of articles and projects attended by the institutes researchers. Let us take "Oxford University" as an example, "Stephen Hawking" and "Alan Turing" are assigned "Oxford University" as their institute (also, yes I am aware Turing did not go to Oxford University). Turing is assigned 3 projects and 1 article and Hawking is assigned 1 project, the productivity output of the students on Oxford University is then the sum of these, so 5. Let us break down how it works. The input is the name of the institute. We declare the output as *iOutput*, we then create the function. First we find all projects participated and articles authored by each researcher, we combine this by taking the union. We then only look at values of the given institute using the WHERE statement and group by the research ids while aggregating the *pnames* as it contains both *pname* and *aid* due to the union. It will then count each non-null value and finally we sum the values. If any null values exist these are simply set to 0 using the COALESCE statement.

```
1 DROP FUNCTION IF EXISTS getOutput;
2 DELIMITER //
```

```
CREATE FUNCTION getOutput(institute VARCHAR(255)) RETURNS INT
BEGIN
    DECLARE iOutput INT;

    SELECT COALESCE(SUM(counts),0) INTO iOutput
    FROM (
        SELECT rid, iname, COUNT(pname) as counts
        FROM (
          (SELECT * FROM researcher NATURAL JOIN participate)
          UNION
          (SELECT * FROM researcher NATURAL JOIN author)) AS t1
        WHERE iname=institute GROUP BY rid
    ) AS t2;

  RETURN iOutput;
END; //
DELIMITER ;
```

A query can now be made, here we give each institute as input to the function made above.

```
SELECT iname, getOutput(iname) AS institute_Output FROM institution
    ORDER BY institute_Output DESC;
```

The output can be seen in Figure 4.

institute	productivity_Output
DTU	5
Oxford University	5
Århus University	1
University of Amsterdam	1
CALTECH	0
ITU	0
KU	0

Figure 4: Output of the function GETOUTPUT

We see DTU and Oxford University are the institutes with the most productive researchers.

10 SQL Programming

Show examples of how to do table modifications using the SQL commands UPDATE and DELETE. You should make one example for each of the two commands. List the queries and show the contents of the changed tables before and after executing the commands.

First I show the update command. We set the institution name "DTU" to "New DTU"

```
1    UPDATE institution SET iname="New DTU" WHERE iname = "DTU";
```

Tables before the update is seen below. First table is **institution**, second is **project** and third is **researcher**:

iname	country
Århus University	Denmark
CALTECH	USA
DTU	Denmark
ITU	Denmark
KU	Denmark
Oxford University	England
University of Amsterdam	Netherlands

pname	iname
Chinese Military Research	DTU
Master Thesis	DTU
Sustainable Development Goals	DTU
Confidential Project	KU
Project Enigma	Oxford University

rid	iname	first_name	last_name
1	DTU	Christian	Glissov
2	DTU	Anna	Lia
3	DTU	Anne	Haxthausen
4	Oxford University	Stephen	Hawking
5	Oxford University	Alan	Turing
6	University of Amsterdam	Diederik	Kingma
7	Århus University	Erik	Eriksen

Tables after the update is seen below:

iname	country
Århus University	Denmark
CALTECH	USA
ITU	Denmark
KU	Denmark
New DTU	Denmark
Oxford University	England
University of Amsterdam	Netherlands

pname	iname
Confidential Project	KU
Chinese Military Research	New DTU
Master Thesis	New DTU
Sustainable Development Goals	New DTU
Project Enigma	Oxford University

rid	iname	first_name	last_name
1	New DTU	Christian	Glissov
2	New DTU	Anna	Lia
3	New DTU	Anne	Haxthausen
4	Oxford University	Stephen	Hawking
5	Oxford University	Alan	Turing
6	University of Amsterdam	Diederik	Kingma
7	Århus University	Erik	Eriksen

Clearly, it can be seen that "DTU" is updated to "New DTU". The foreign keys of *iname* and the referential action makes sure the update is cascaded into the other tables.

Now I show the delete command. I update the institution name "New DTU" to "DTU" again, then i delete the project "Project Enigma".

```
1    DELETE FROM project WHERE pname = "Project Enigma";
```

First tables before the delete. First table is **project** and the second is **participation**:

pname	iname
Chinese Military Research	DTU
Master Thesis	DTU
Sustainable Development Goals	DTU
Confidential Project	KU
Project Enigma	Oxford University

pname	rid
Confidential Project	4
Confidential Project	5
Project Enigma	5
Sustainable Development Goals	1
Sustainable Development Goals	2
Sustainable Development Goals	5

Then the tables after the delete:

pname	iname
Chinese Military Research	DTU
Master Thesis	DTU
Sustainable Development Goals	DTU
Confidential Project	KU

pname	rid
Confidential Project	4
Confidential Project	5
Sustainable Development Goals	1
Sustainable Development Goals	2
Sustainable Development Goals	5

It can be seen that "Project Enigma" has been deleted based on the referential action.