Viivi Väisänen

Final project

Advanced Data Management Systems

Stage 1: Design

An important client has hired us to design a comprehensive research database that provides access to a vast collection of scholarly literature, including scientific journals, and conference proceedings. The database should be designed to facilitate the discovery and analysis of research publications across various disciplines. Publications have a title, abstract, keywords and DOI number and are published in a specific source (journal or conference proceedings) in a given year. They can also have other attributes such as volume and issue number. Moreover, publications can cite other publications. Publications are written by one or several authors, which belong to a certain affiliation and country, and for each of them we also have the total number of publications they have and their gender. Journals have a unique identifier (source) name and sometimes a fixed ISBN or ISSN number. Conference proceedings also have a unique identifier (source) and have a different title, ISBN or ISSN for every edition of the conference (often every year). Currently, the data is in a single table in a MySQL database (see **RESEARCH.sql** file on eLearn) and looks like this:

PaperID	Year	Title	Abstr...	Keywords	Source	DOI	Volume	Issue	ISSN_ISBN	Type	CitedBy	Author	AuthorID	Gender	Affiliation	Country	Npub
1086	2020	Sustainable...	The smart...	Electricity c...	Sustaina...	10.33...	12	8	20711050	Article	2514; 1914; 3...	(Ene) G.D.	574230...	Male	Department...	Romania	1
1071	2020	OLAP paral...	The adve...	compute clo...	Concurr...	10.10...	32	7	15320626	Article	2730; 684; 28...	A. B. Lima...	572124...	Male	COPPE, Un...	Brazil	1
256	2023	VAARTA: A...	Chat appli...	Advanced e...	Lecture...	10.10...	682 L...	NULL	23673370 978-9...	Confe...	1250; 576; 309	Aarti	559043...	Male	Department...	India	1
1065	2020	Data Mana...	Most data...	Distributed...	Proceedi...	10.11...	NULL	NULL	978-172816095-5	Confe...	982; 2711	Aatish C.	560221...	Female	University o...	Mauritius	1
805	2020	DBMS, No...	In modern...	DBMS; DB...	2020 Int...	10.11...	NULL	NULL	978-172819673-2	Confe...	3095; 3467; 3...	Abahussai...	572153...	Male	University o...	Bahrain	1
134	2023	Efficient Dis...	Session In...	cache; Clus...	IEEE Int...	10.11...	2023-...	NULL	15503607 978-1...	Confe...	503; 2862; 16...	Abakar K.S.	579597...	Male	Irimas, Univ...	France	1
1105	2020	Standards...	The area...	CEN 13606...	CEUR W...	NULL	2656	NULL	16130073	Confe...	1741; 2905; 2...	Abanos S.	572172...	Male	Faculty of...	Bulgaria	1
891	2021	NoSQL Co...	This articl...	Cassandra;...	Advance...	10.10...	1366...	NULL	21945357 978-3...	Confe...	329; 2910; 63...	Abbasi M.	556518...	Male	CISUC—C...	Portugal	7
846	2021	MongoDB,...	With the r...	Couchbase;...	Advance...	10.10...	1366...	NULL	21945357 978-3...	Confe...	1693; 2422; 822	Abbasi M.	556518...	Male	CISUC—C...	Portugal	7
265	2023	Database P...	The numb...	Database p...	Smart In...	10.10...	337 S...	NULL	21903018 978-9...	Confe...	994; 974; 822	Abbasi M.	556518...	Male	CISUC—C...	Portugal	7
139	2023	Performanc...	This pape...	benchmarki...	2023 2n...	10.11...	NULL	NULL	979-835030541-8	Confe...	2519; 754; 16...	Abbasi M.	556518...	Male	CISUC—C...	Portugal	7
25	2024	Performanc...	Several m...	HarperDB;...	Smart In...	10.10...	344	NULL	21903018 978-9...	Confe...	3533; 1513; 1...	Abbasi M.	556518...	Male	CISUC—C...	Portugal	7
631	2021	MetaXplor:...	Backgrou...	assignment;...	GigaSci...	10.10...	10	2	2047217X	Article	1054; 3173; 3...	Abbé M.	572220...	Male	CIRAD, UM...	France	1
740	2021	IoT web-ba...	This pape...	Amazon; A...	Internati...	10.11...	2021-...	NULL	21619646 978-1...	Confe...	3110; 1002; 81	Abdallah...	572186...	Male	Universite d...	Canada	1
645	2021	A compreh...	Nowadays...	Big Data; G...	Journal...	NULL	99	12	19928645	Article	NULL	Abdelgab...	571891...	Male	Department...	Egypt	2
305	2022	A Compreh...	Currently,...	big data; mi...	Big Data...	10.33...	6	3	25042289	Article	3029; 2376; 2...	Abdelgab...	571891...	Male	Department...	Egypt	2
1161	2020	Discovering...	NoSQL sy...	Big Data; M...	ICEIS 20...	NULL	1	NULL	978-989758423-7	Confe...	2452; 1497; 9...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
1144	2020	Reverse en...	In recent y...	Big data; C...	Lecture...	10.10...	12393...	NULL	03029743 978-3...	Confe...	NULL	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
889	2021	OCL constr...	Big data h...	Big Data; C...	Internati...	10.40...	17	1	15483924	Article	1277; 2067; 1...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
730	2021	Ingestion of...	The expo...	Big Data; D...	Internati...	NULL	3	NULL	21843228 978-9...	Confe...	1056; 2939; 3...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
728	2021	Automatic...	The NoS...	Big Data; M...	Internati...	NULL	1	NULL	21844992 978-9...	Confe...	3107; 228; 38...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
466	2022	Extraction...	The ""sch...	ATL; Logical...	Internati...	10.52...	NULL	NULL	21844348 978-9...	Confe...	3241; 3779; 2...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
416	2022	Data Ingest...	Nowadays...	Big Data; D...	Internati...	10.52...	1	NULL	21844992 978-9...	Confe...	2861; 1940; 1...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
345	2022	DLToDW: T...	Over the p...	Big Data; D...	SN Com...	10.10...	3	5	2662995X	Article	1836; 3286; 2...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
167	2023	Medical dat...	In today's...	Big Data; co...	Proceedi...	10.11...	NULL	NULL	21615322 979-8...	Confe...	1907; 2416; 2...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
145	2023	Extraction o...	The prior...	Logical sch...	SN Com...	10.10...	4	2	2662995X	Article	470; 1260; 13...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
107	2023	Conceptual...	Massive d...	conceptual...	Proceedi...	10.11...	NULL	NULL	21615322 979-8...	Confe...	2707; 3519; 3...	Abdelhedi F.	572065...	Female	CBI2-TRIM...	France	18
714	2021	Towards No...	Maritime s...	Document...	Informati...	10.31...	45	3	03505596	Article	1245	Abdelkad...	573460...	Female	Computer s...	Algeria	1
66	2024	From Funct...	The desig...	Document-o...	Commu...	10.10...	2071	NULL	18650929 978-3...	Confe...	722; 828; 347	Abdellatif T.	562923...	Male	SERCOM - ...	Tunisia	1

As you can see, this format is prone to many mistakes and not optimized at all for its use in a real system so your task is to improve it:

A. Determine the degree of normalization of the current database. If it is not at least 3NF, normalize it further until you reach this degree of normalization and implement it in SQL. Explain all steps of the process.

The current database's degree of normalization is non-normalized, because it violates the 1NF: it has columns, which store semicolon- separated values, which are Keywords and CitedBy. Comma-separated values (CSV) are problematic, because they create difficulties in querying and limitations in data integrity, indexing and searching. Also, to achieve at least the 1NF, we can't have comma-separated values in our columns.

Viivi Väisänen

There are several problems in the current database, and if it is not normalized, we will face many anomalies, some discussed below.

To normalize the database, we need to create separate tables, each containing information related to one subject. This way, we can ensure that we'll achieve 3NF.
I divided the database into these following tables:

Author-table
- <u>authorID</u>
- name
- gender
- country
- totalNumbPublications
- affiliation

Explanation:
I created this table to store data about the authors. This table stores data about the author's name, gender, country, the total number of publications, and the affiliation. By creating this table, we can avoid duplicate information about the authors. In the original one-table-database, it would be impossible to insert a new author, who does not have a paper. Also, if we would want to delete a paper, we might lose valuable data about the authors. By creating this table, these anomalies are avoided.

Paper-table
- <u>paperID</u>
- title
- abstract
- DOI
- source
- type
- year
- ISSN_ISBN
- volume

Explanation:
I created this table to store data about the publications. The table stores information about the publications title, abstract, DOI-number, source, type, year, ISSN_ISBN-number and volume. I separated the issue into a separate table to avoid null values. With this paper-table we do not have to list the same article many times, if we want to insert a new author, for example.

PaperIssues-table
- paperID
- issue

Explanation:
I separated the paper's possible issue into a separate table, because I counted how many null values Issues-column had in the one-table-database, and compared to the not null values, the number was significantly higher. The number of rows where the issue was null was 9677, and the number of not null-rows was 3568. This means that storage space has been wasted in many rows unnecessarily. To avoid null values, I created a separate table to store the issues related to a specific paper.

paperAuthors-table

- paperID
- authorID

Explanation:
I created this intermediate table about the paper's authors. This table enables associating one or more authors with a specific paper.

keyword-table

- keywordID
- keyword

Explanation:
I created a keyword-table to store data about the keywords.

paperKeyWords-table

- paperID
- keywordID

Explanation:
I created this intermediate table about the keywords that relate to a specific paper. Thanks to this table, we can avoid the CSV problem in the Keywords-column. This table enables associating zero or more keywords with a specific paper.

CitedBy-table

- paperID
- citedPaperID

Explanation:
I created this table to avoid semicolon separated values in the original database. Thanks to this table, we can associate a paper to zero or more cited papers, without CSV.

—---

Here is the implementation in SQL:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema publications
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema publications
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `publications` DEFAULT CHARACTER SET
```

```
utf8mb3 ;
USE `publications` ;

-- -----------------------------------------------------
-- Table `publications`.`author`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`author` (
  `authorID` VARCHAR(30) NOT NULL,
  `name` VARCHAR(301) NOT NULL,
  `gender` VARCHAR(6) NULL DEFAULT NULL,
  `totalNumbPublications` INT NULL DEFAULT NULL,
  `affiliation` VARCHAR(417) NULL DEFAULT NULL,
  `country` VARCHAR(31) NULL DEFAULT NULL,
  PRIMARY KEY (`authorID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `publications`.`paper`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`paper` (
  `paperID` INT NOT NULL,
  `title` VARCHAR(367) NOT NULL,
  `abstract` VARCHAR(4782) NULL DEFAULT NULL,
  `DOI` VARCHAR(52) NULL DEFAULT NULL,
  `source` VARCHAR(491) NULL DEFAULT NULL,
  `type` VARCHAR(16) NULL DEFAULT NULL,
  `ISSN_ISBN` VARCHAR(160) NULL DEFAULT NULL,
  `volume` VARCHAR(116) NULL DEFAULT NULL,
  `year` INT NOT NULL,
  PRIMARY KEY (`paperID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `publications`.`citedby`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`citedby` (
  `paperID` INT NOT NULL,
  `citedPaper` INT NOT NULL,
  PRIMARY KEY (`paperID`, `citedPaper`),
  CONSTRAINT `fk_citedBy_paper1`
    FOREIGN KEY (`paperID`)
    REFERENCES `publications`.`paper` (`paperID`),
  CONSTRAINT `fk_citedBy_paper2`
    FOREIGN KEY (`citedPaper`)
    REFERENCES `publications`.`paper` (`paperID`)
)

ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;
```

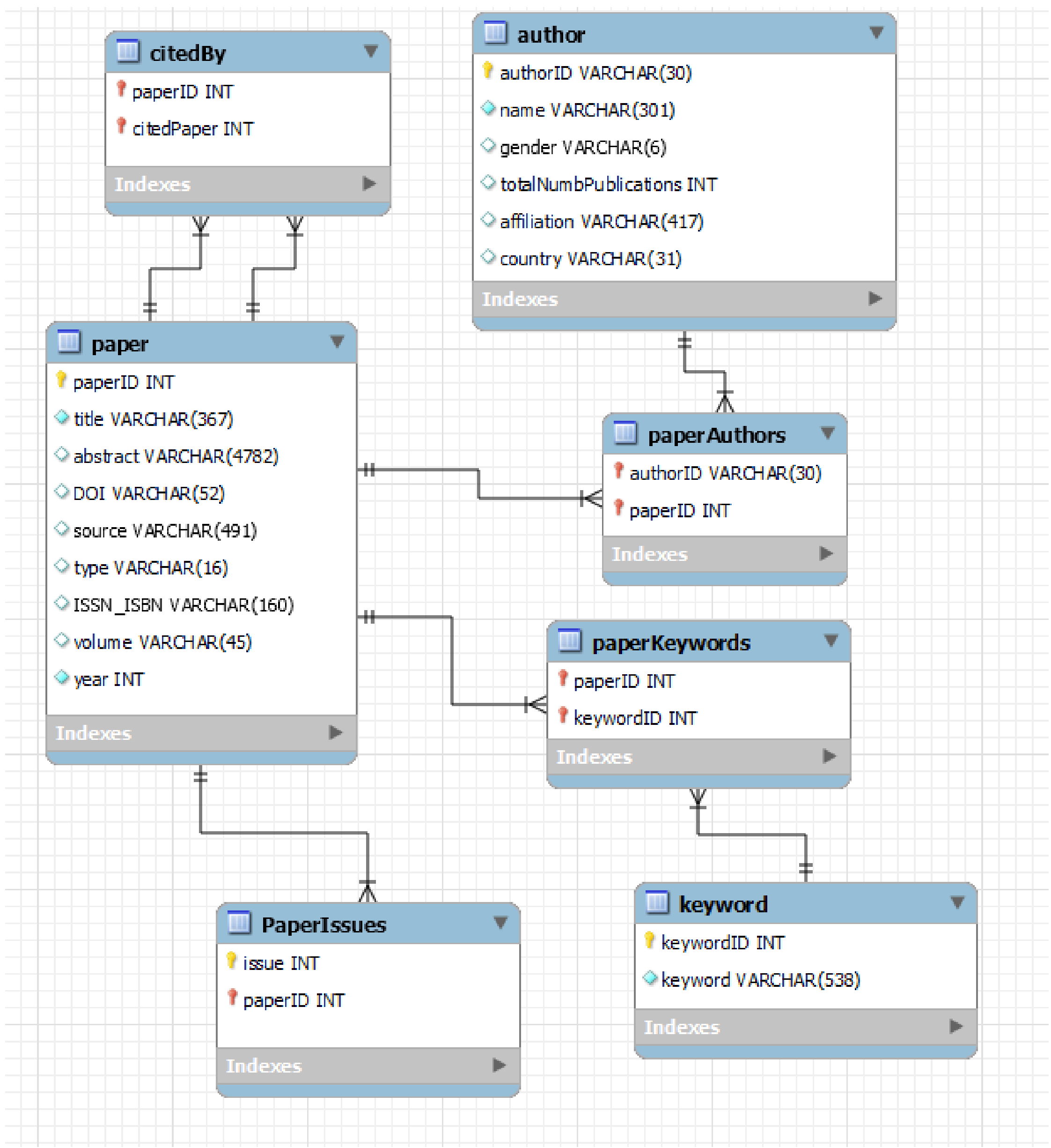
```
-- -----------------------------------------------------
-- Table `publications`.`keyword`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`keyword` (
  `keywordID` INT NOT NULL,
  `keyword` VARCHAR(538) NOT NULL,
  PRIMARY KEY (`keywordID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `publications`.`paperauthors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`paperauthors` (
  `authorID` VARCHAR(30) NOT NULL,
  `paperID` INT NOT NULL,
  PRIMARY KEY (`authorID`, `paperID`),
  INDEX `fk_paperAuthors_paper1_idx` (`paperID` ASC) VISIBLE,
  CONSTRAINT `fk_paperAuthors_author1`
    FOREIGN KEY (`authorID`)
    REFERENCES `publications`.`author` (`authorID`),
  CONSTRAINT `fk_paperAuthors_paper1`
    FOREIGN KEY (`paperID`)
    REFERENCES `publications`.`paper` (`paperID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `publications`.`paperissues`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`paperissues` (
    `issue` VARCHAR(22) NOT NULL,
  `paperID` INT NOT NULL,
  PRIMARY KEY (`issue`, `paperID`),
  INDEX `fk_PaperIssues_paper1_idx` (`paperID` ASC) VISIBLE,
  CONSTRAINT `fk_PaperIssues_paper1`
    FOREIGN KEY (`paperID`)
    REFERENCES `publications`.`paper` (`paperID`)
)

ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- -----------------------------------------------------
-- Table `publications`.`paperkeywords`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `publications`.`paperkeywords` (
  `paperID` INT NOT NULL,
  `keywordID` INT NOT NULL,
  PRIMARY KEY (`paperID`, `keywordID`),
  INDEX `fk_paperKeywords_paper1_idx` (`paperID` ASC) VISIBLE,
  INDEX `fk_paperKeywords_keyword1_idx` (`keywordID` ASC)
```

Viivi Väisänen

```
VISIBLE,
  CONSTRAINT `fk_paperKeywords_keyword1`
    FOREIGN KEY (`keywordID`)
    REFERENCES `publications`.`keyword` (`keywordID`),
  CONSTRAINT `fk_paperKeywords_paper1`
    FOREIGN KEY (`paperID`)
    REFERENCES `publications`.`paper` (`paperID`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

B. Draw the EER or physical diagram (you can use the reverse engineering feature of MySQLworkbench) of your final solution.

Viivi Väisänen



C. Transfer the data to the new schema using SQL commands. An example of how to do this can be seen here: https://www.w3schools.com/sql/sql_insert_into_select.asp

The research-schema and my publications-schema were in the same MySQL WorkBench connection. In the connection, I transferred the data to my new schema using SQL commands below.

Notes: *While parsing the data in the CitedBy column of the research table, I noticed that there are 119 cited paperIDs that are not present in the database. This resulted in me being unable to add citations to the new citedby-table because of a foreign key constraint failure. As a result, I decided to remove these 119 individual citations that were not added to the database.*

```
use research;

-- data insertion to author-table
INSERT INTO publications.author
     SELECT distinct AuthorID, Author, Gender, Npub, Affiliation,
country FROM research.research;

-- data insertion to paper-table
 INSERT INTO publications.paper
     SELECT distinct PaperID, title, abstract, DOI, source, type,
ISSN_ISBN, volume, Year FROM research.research;

-- data insertion to paperauthors-table
 INSERT INTO publications.paperauthors
     SELECT AuthorID, PaperID FROM research.research;


-- I investigated the maximum number of citations in the database
with this query (result was 19)
-- The query calculates the amount of semocolons: number of
citations is the number of semicolons + 1:
-- SELECT max((LENGTH(CitedBy) - LENGTH(REPLACE(CitedBy, '; ',
'')))) + 1 AS CitedBy_amount
-- FROM research.research;

-- Let's create a temporary table, which is as big as the length
of the longest list of citations.
-- I found out that in the original database, papers have 19
citations maximum:
create temporary table research.temp_table as (
select 1 as n
union select 2 as n
union select 3 as n
union select 4 as n
union select 5 as n
union select 6 as n
union select 7 as n
union select 8 as n
union select 9 as n
union select 10 as n
union select 11 as n
union select 12 as n
union select 13 as n
union select 14 as n
union select 15 as n
union select 16 as n
union select 17 as n
union select 18 as n
union select 19 as n
);
```

```
-- Next, let's join the temporary table to the research-table.
select r.CitedBy from research.research r
join research.temp_table t
  on char_length(r.CitedBy)
    - char_length(replace(r.CitedBy, '; ', ''))
    >= n - 1;

-- Let's create a new table for help
CREATE TABLE research.aputaulu (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    paperID INT,
    citedPaper INT
);

-- data insertion to the table we just created
insert into research.aputaulu (paperID, citedPaper)
SELECT DISTINCT
    r.paperID,
    SUBSTRING_INDEX(SUBSTRING_INDEX(r.CitedBy, '; ', n), '; ',
-1) AS Cited
FROM research.research r
JOIN research.temp_table t ON CHAR_LENGTH(r.CitedBy) -
CHAR_LENGTH(REPLACE(r.CitedBy, '; ', '')) >= n - 1
JOIN publications.paper p ON r.paperID = p.paperID;

-- let's delete all the 119 citations, that don't exist in the
research database
delete from research.aputaulu
where citedPaper not in (select paperID from research.research);

-- let's add the data to the citedby-table
insert into publications.citedby
select paperID, citedPaper from research.aputaulu;


DROP table if exists research.aputaulu;
DROP temporary table research.temp_table;




-- data insertion to paperissues-table
insert into publications.paperissues
select distinct Issue, paperID from research.research
where Issue is not null;



-- I investigated the maximum number of keywords in the database
with this query (result was 53)
-- The query calculates the amount of semocolons: number of
keywords is the number of semicolons + 1:
-- SELECT max((LENGTH(Keywords) - LENGTH(REPLACE(Keywords, '; ',
'')))) + 1 AS keyword_amount
-- FROM research.research;
```

```
-- Let's create a temporary table, which is as big as the length
of the longest list of keywords.
-- I found out that in the original database, papers have 53
keywords maximum:
create temporary table research.temp_table1 as (
select 1 as n
union select 2 as n
union select 3 as n
union select 4 as n
union select 5 as n
union select 6 as n
union select 7 as n
union select 8 as n
union select 9 as n
union select 10 as n
union select 11 as n
union select 12 as n
union select 13 as n
union select 14 as n
union select 15 as n
union select 16 as n
union select 17 as n
union select 18 as n
union select 19 as n
union select 20 as n
union select 21 as n
union select 22 as n
union select 23 as n
union select 24 as n
union select 25 as n
union select 26 as n
union select 27 as n
union select 28 as n
union select 29 as n
union select 30 as n
union select 31 as n
union select 32 as n
union select 33 as n
union select 34 as n
union select 35 as n
union select 36 as n
union select 37 as n
union select 38 as n
union select 39 as n
union select 40 as n
union select 41 as n
union select 42 as n
union select 43 as n
union select 44 as n
union select 45 as n
union select 46 as n
union select 47 as n
union select 48 as n
union select 49 as n
```

```
union select 50 as n
union select 51 as n
union select 52 as n
union select 53 as n
);


-- Next, let's join the temporary table to the research-table.
select r.Keywords from research.research r
join research.temp_table1 t
  on char_length(r.Keywords)
    - char_length(replace(r.Keywords, '; ', ''))
    >= n - 1;

-- Let’s create another table for help
CREATE TABLE research.aputaulu1 (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    keyword varchar(538)
);


-- Data insertion to the table we just created
insert into research.aputaulu1 (keyword)
SELECT DISTINCT
    SUBSTRING_INDEX(SUBSTRING_INDEX(r.Keywords, '; ', n), '; ',
-1) AS Keyword
FROM research.research r
JOIN research.temp_table1 t ON CHAR_LENGTH(r.Keywords) -
CHAR_LENGTH(REPLACE(r.Keywords, '; ', '')) >= n - 1;


-- data insertion to keyword-table
Insert into publications.keyword
select ID, keyword from research.aputaulu1;

drop table if exists research.aputaulu1;




-- let's create another table for help.
CREATE TABLE research.aputaulu5 (
    paperID INT,
    keyword varchar(538)
);


-- data insertion to aputaulu5
insert into research.aputaulu5 (paperID, keyword)
SELECT DISTINCT
paperID,
    SUBSTRING_INDEX(SUBSTRING_INDEX(r.Keywords, '; ', n), '; ',
-1) AS Keyword
FROM research.research r
JOIN research.temp_table1 t ON CHAR_LENGTH(r.Keywords) -
```

```
CHAR_LENGTH(REPLACE(r.Keywords, '; ', '')) >= n - 1;

-- data insertion to paperkeywords-table
insert into publications.paperkeywords
select a.paperID, k.keywordID
from research.aputaulu5 a
join publications.keyword k on k.keyword = a.keyword;

drop table if exists research.aputaulu5;
drop temporary table if exists research.temp_table1;
```

Stage 2: Optimization

The client is now very satisfied with your design but he is worried that it might not work great for some of the most recurring features that users will make use of in the online interface of the database:

- Search articles by terms included in the title or abstract and filter by year, source (conference proceedings title or journal title) or keyword. The search should return, for each article, the title, author names, year, keywords, source (journal or conference proceedings where it was published), and number of citations.
- Search for all the information related to a specific author including their name, all their publications (title, year and source), gender, their co-authors and the number of joint publications with each of them, the total number of citations received, and their institution and country.
- Search for the most prolific authors (by total number of publications) from a given country, gender or affiliation (or overall).

With this new information,

A. Write the SQL code needed for the required features,

Here's a SQL query, which returns, for each article the title, author names, year, keywords, source and number of citations. In the query, there is where-statement (which is commented below), and it allows to filter the results by title or abstract, and filter by year, source or keyword. Without the WHERE-clause, the query returns the results without filters.

```
SELECT p.title AS title,
--   Let's combine the authors of a paper into a single string
using GROUP_CONCAT:
    GROUP_CONCAT(DISTINCT a.name ORDER BY a.name SEPARATOR ', ')
AS authors,
    p.year AS year,
```

```
--   Let's combine the keywords of a paper into a single string
using GROUP_CONCAT:
    GROUP_CONCAT(DISTINCT k.keyword ORDER BY k.keyword SEPARATOR
', ') AS keywords,
    p.source AS source,
-- If there's no citations, the number of citations is 0:
    IFNULL(c.citation_count, 0) AS citation_count
FROM
    publications.paper p
    JOIN publications.paperauthors pa ON p.paperID = pa.paperID
    JOIN publications.author a ON pa.authorID = a.authorID
    LEFT JOIN publications.paperkeywords pk ON p.paperID =
pk.paperID
    LEFT JOIN publications.keyword k ON pk.keywordID =
k.keywordID
    LEFT JOIN (
        SELECT citedPaper, COUNT(*) AS citation_count
        FROM publications.citedby
        GROUP BY citedPaper
    ) c ON p.paperID = c.citedPaper

-- THIS PART ALLOWS TO FILTER THE RESULTS BY TERMS IN THE TITLE
OR ABSTRACT,
-- AND FILTER ALSO BY YEAR, SOURCE, OR KEYWORD.
-- WHERE
-- (p.title LIKE CONCAT('%String here%') OR p.abstract LIKE
CONCAT('%Search_string_here%'))
-- AND p.year = 0000
-- AND p.source = 'source here'
 -- AND k.keyword = 'keyword name here'
-- Let's group the results by each row so that each row
corresponds to a unique paper:
GROUP BY
    p.paperID, p.title, p.year, p.source, citation_count
ORDER BY
    p.year DESC, p.title ASC;
```

Here's a sql query, which returns all the information related to a specific author including their name, all their publications (title, year and source), gender, their co-authors and the number of joint publications with each of them, the total number of citations received, and their institution and country. This search is done with three separate select-statements. To get all the information, run all of the three selects.

```
-- This query returns a specific author's name, gender, country
```

```
and total citations.
SELECT
    a.name AS author_name,
    a.gender,
    a.affiliation,
    a.country,
    IFNULL(c.total_citations, 0) AS total_citations
FROM
    publications.author a
    LEFT JOIN (
        SELECT
            authorID,
            COUNT(c.citedPaper) AS total_citations
        FROM
            publications.paperauthors pa
            JOIN publications.citedby c ON pa.paperID =
c.citedPaper
        GROUP BY
            authorID
    ) c ON a.authorID = c.authorID
WHERE
    a.authorID = 10039885900 ; -- INSERT THE AUTHOR'S ID HERE


 -- This second query returns all publications of a specific
author, along with their title, year, source and co-authors
 SELECT
    p.title AS paper_title,
    p.year AS publication_year,
    p.source AS publication_source,
    GROUP_CONCAT(DISTINCT co_authors.name ORDER BY
co_authors.name SEPARATOR ', ') AS co_authors
FROM
    publications.paper p
    JOIN publications.paperauthors pa ON p.paperID = pa.paperID
    JOIN publications.author a ON pa.authorID = a.authorID
    LEFT JOIN publications.paperauthors co_pa ON p.paperID =
co_pa.paperID AND co_pa.authorID != a.authorID
    LEFT JOIN publications.author co_authors ON co_pa.authorID =
co_authors.authorID
WHERE
    a.authorID = 10039885900 -- INSERT THE AUTHOR'S ID HERE
GROUP BY
    p.title, p.year, p.source;



-- This third query retrieves all co-authors of a specific author
and number of joint publications with each of them
SELECT
    co_authors.name AS co_author_name,
    COUNT(DISTINCT p.paperID) AS joint_publications_count
FROM
    publications.paperauthors pa
    JOIN publications.author a ON pa.authorID = a.authorID
```

```
    JOIN publications.paper p ON pa.paperID = p.paperID
    LEFT JOIN publications.paperauthors co_pa ON p.paperID =
co_pa.paperID AND co_pa.authorID != a.authorID
    LEFT JOIN publications.author co_authors ON co_pa.authorID =
co_authors.authorID
WHERE
    a.authorID = 10039885900  -- INSERT THE AUTHOR'S ID HERE
GROUP BY
    co_authors.name;
```

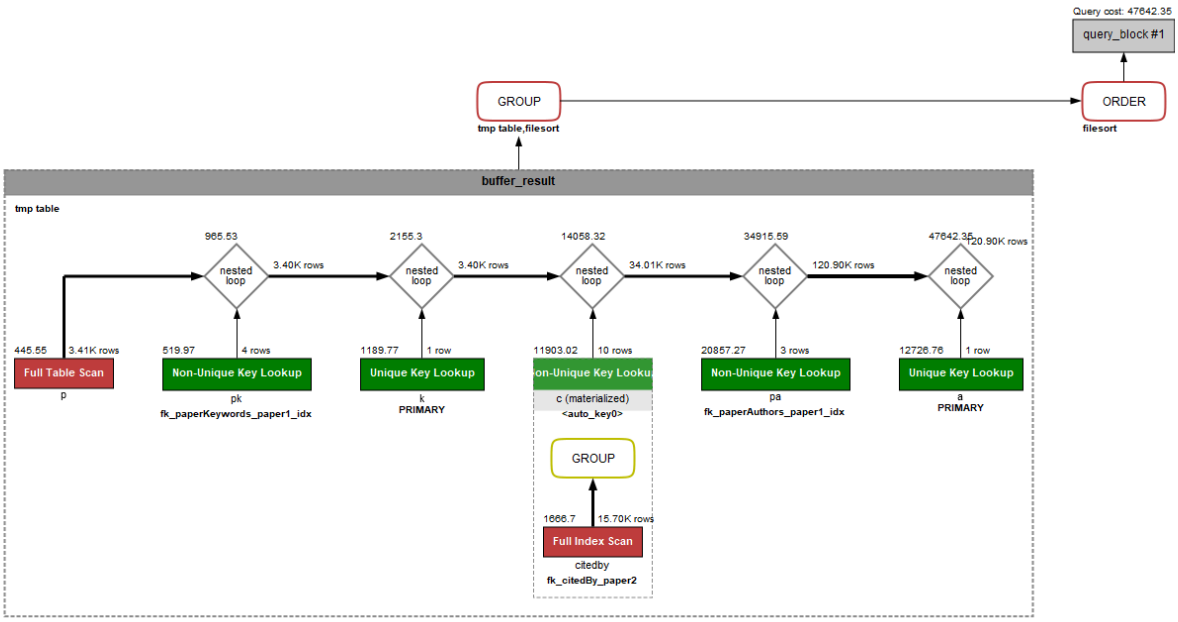
This query searches the 10 most prolific authors by total number of publications. In the query, you can also search by country, gender of affiliation name by removing the comments from the where-clauses

```
-- This query searches 10 most prolific authors by total number
of publications
-- If you want to search by author's country, gender or
affiliation name, uncomment where-clauses
select
    authorID,
    name as author_name,
    totalNumbPublications as total_publications_count
from
    publications.author

-- IF YOU WANT TO SEARCH BY COUNTRY, UNCOMMENT THIS:
-- where country = 'Brazil' -- insert the country here

-- IF YOU WANT TO SEARCH BY GENDER, UNCOMMENT THIS:
-- where gender = 'Female' -- insert gender here ('Female' or
'Male')

-- IF YOU WANT TO SEARCH BY AFFILIATION, UNCOMMENT THIS:
-- where affiliation like 'insert here%' -- insert affiliation
name here
ORDER BY
     total_publications_count DESC
LIMIT 10;  -- The number of rows that are shown
```

B. Identify possible bottlenecks and ways to optimize the performance. Document all of the things you try.

In this section, I will attempt to identify the possible bottlenecks in the queries from section A and suggest and try ideas on how the queries could be optimized by modifying the database structure.

Identifying possible bottlenecks in the queries is possible by studying MySQL’s execution plan.
For example, the first query’s execution plan here below, when results are filtered by ‘concurrency’ included in the title:



Total cost: 47642.35

Query’s performance can be hindered by bottlenecks, for example full table or index scans, hash joins and nested loops. Full scans are very costly for large tables, and require much CPU resources. As can be seen in the plan, the first query does full table scan on paper-table, and full index scan on citedby-column (red coloured boxes). These can be possibly reduced by creating indexes that match the query condition.

Denormalizing the database would optimize the performance in the searches. The searches in section A use GROUP_CONCAT to combine multiple rows into one string, separated by comma. This could be avoided by storing keywords in the paper-table, in the same way as in the original database’s Keywords and CitedBy column. This could also be done to paper’s authors, since the queries use CROUP_CONCAT to combine paper’s authors into a single string. This would improve the performance, but we should also consider the risks of denormalization. If we want to store comma-separated values, no indexes are going to work right in those columns, also comma-separated values create difficulties in querying, although it might improve the performance of certain queries.

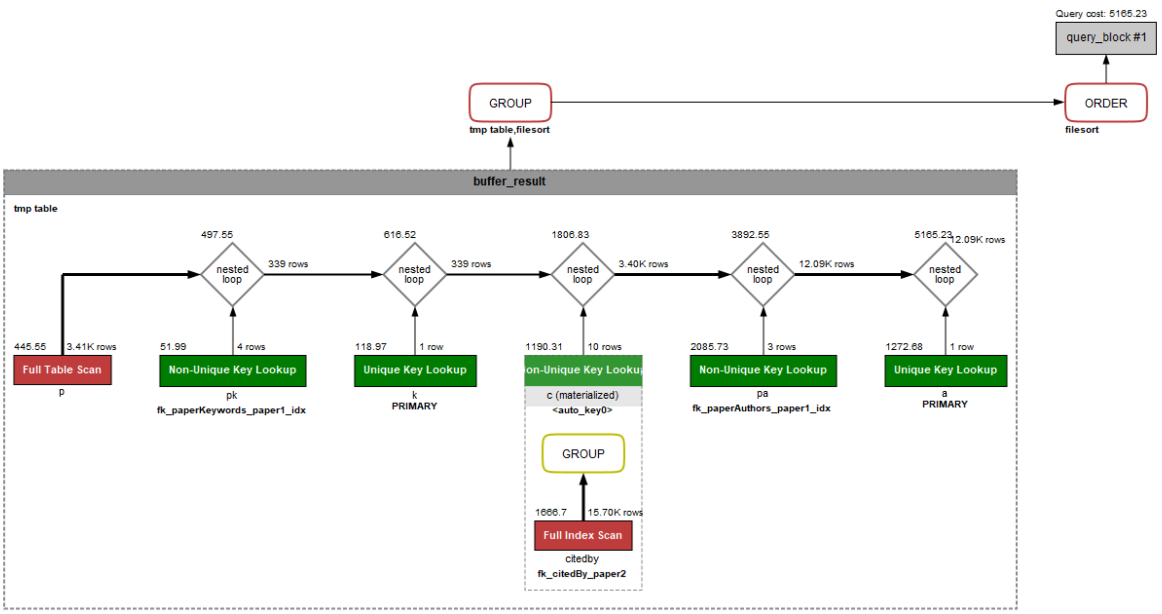
We can also optimize the performance by denormalizing the database structure to be able to reduce the number of joins in the queries, since each join increases the total cost. This is possible by creating columns to a table that already exist in another table, combining multiple tables into one, or creating summary tables.

OPTIMIZING THE FIRST SEARCH

The first search of section A searches articles by terms included in the title or abstract and filter by year, source or keyword. The search returns, for each article, the title, author names, year, keywords, source (journal or conference proceedings where it was published), and number of citations. The execution plan can be seen above.

I is possible to improve

Let's run the same query, but filter results on titles that start with C, and are published in year 2023:

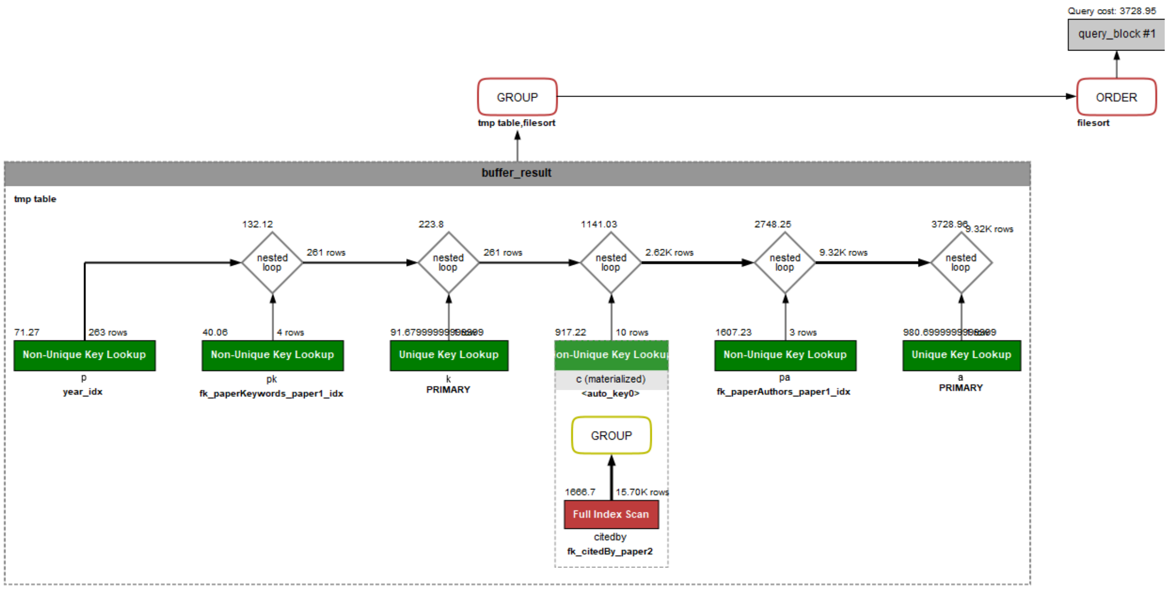


Total cost 5165.23

As seen in the execution plan above, the query does full table scan on table paper and full index scan on citedby table. When searching by year, we can possibly avoid full table scan on paper table by creating an index on column year.

MODIFICATIONS

I created an index on column year in table paper. (you can see this in SQL in section C)
Let's study the query plan after creating the index.
The results are filtered again by year (2023) and title (starting with c):



Total cost: 3728.95

When compared to the previous execution plan (before creating the index), we can see that the total cost has decreased from 5165.23 to 3728.95. With the index we avoided full table scan on table paper, because the query is able to use the index created on year. So, when searched by a certain year, index made on column year does optimize the query's execution. This is a great example of how much a well-chosen index can optimize a query.

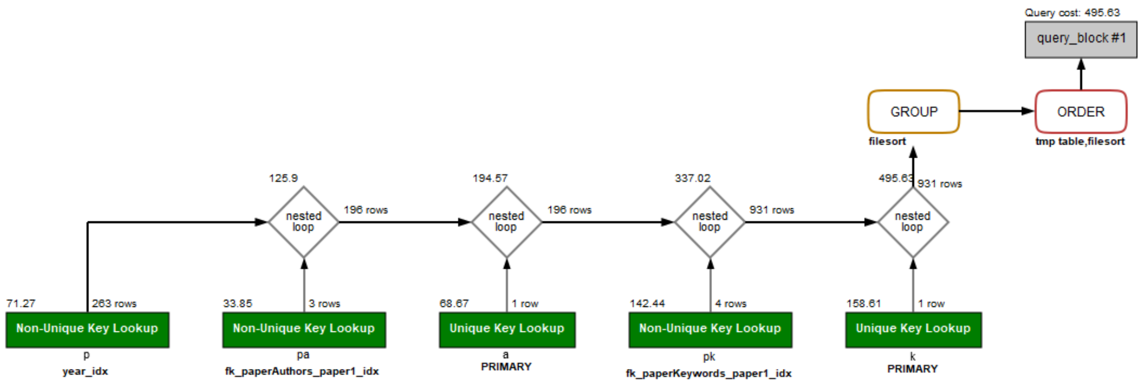
It is possible to optimize the query's execution further by modifying the database structure in a way that we can avoid as many joins as possible.

First, let's try to create a new column citation_count in table paper, so we don't have to calculate citations or join to the citedby table. Full index scan on citedby table seems to be the last possible bottleneck of the query (red box in the plan above). (you can see the column's creation in SQL in section C)

Let's modify the query to utilize the new column.
Here is the modified query and it's execution plan (results are filtered again with year 2023, and country 'Brazil':

```
SELECT p.title AS title,
--   Let's combine the authors of a paper into a single string
using GROUP_CONCAT:
    GROUP_CONCAT(DISTINCT a.name ORDER BY a.name SEPARATOR ', ')
AS authors,
```

```
    p.year AS year,
--   Let's combine the keywords of a paper into a single string
using GROUP_CONCAT:
    GROUP_CONCAT(DISTINCT k.keyword ORDER BY k.keyword SEPARATOR
', ') AS keywords,
    p.source AS source,
    p.citation_count
FROM
    publications.paper p
    JOIN publications.paperauthors pa ON p.paperID = pa.paperID
    JOIN publications.author a ON pa.authorID = a.authorID
    LEFT JOIN publications.paperkeywords pk ON p.paperID =
pk.paperID
    LEFT JOIN publications.keyword k ON pk.keywordID =
k.keywordID


-- THIS PART ALLOWS TO FILTER THE RESULTS BY TERMS IN THE TITLE
OR ABSTRACT,
-- AND FILTER ALSO BY YEAR, SOURCE, OR KEYWORD.
 WHERE
 (p.title LIKE CONCAT('C%') OR p.abstract LIKE
CONCAT('%Search_string_here%'))
AND p.year = 2023
-- AND p.source = 'International%'
-- AND k.keyword = 'keyword_here'

-- Let's group the results by each row so that each row
corresponds to a unique paper:
GROUP BY
    p.paperID, p.title, p.year, p.source, citation_count
ORDER BY
    p.year DESC, p.title ASC;
```

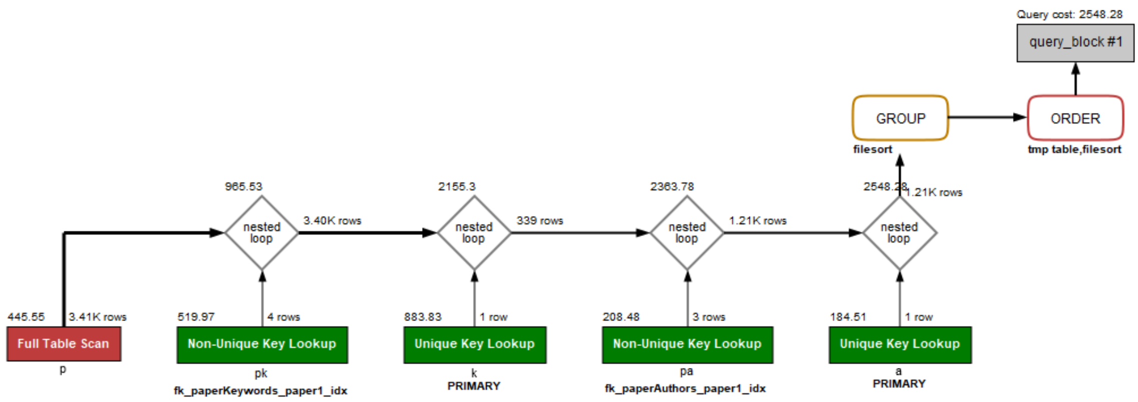


Total cost: 495.63

When comparing the plan to the previous plan before creating the new column, we can see that the query's total cost has decreased from 3728.95 to just 495.63, which is a significant drop in cost. This proves that the full index scan on table citedby was a bottleneck, which hindered the performance. This bottleneck was addressed by adding a new column to the paper table, which also allowed us to avoid one join which can be seen in the execution plan as one fewer diamond compared to the plan before the column was created.

Let's study the execution plan without the filter year.

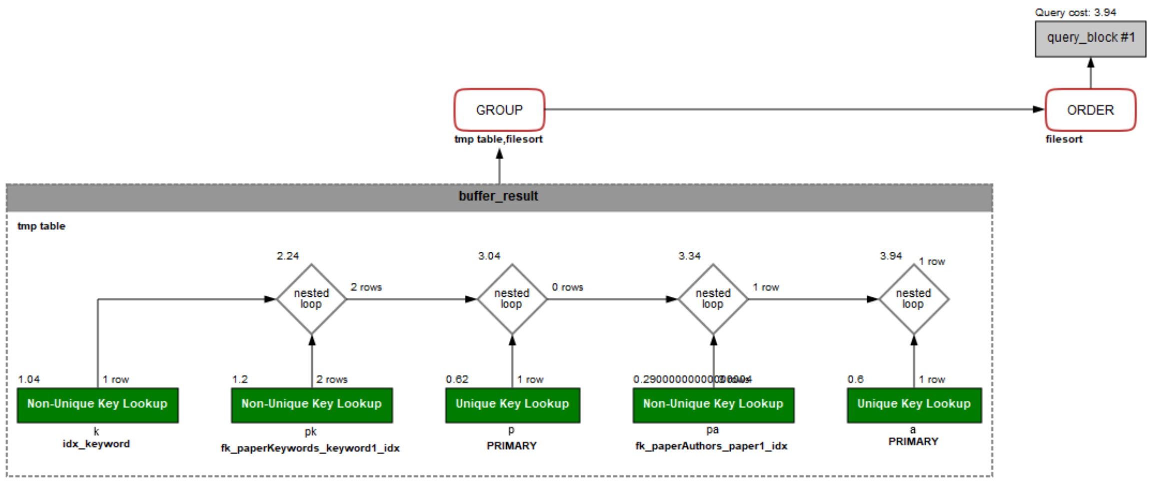
The results are filtered by title (starting with C), and by keyword 'Database':



Full cost: 2548.28

As can be seen in the execution plan, the index created on column year obviously doesn't work wonders, when we are not having year in the where-clause.

Let's try to create an index on keyword, and then study the execution plan. (see the creation of an index in section C.)
The results are filtered by title (starting with C), and by keyword 'Database.':



Total cost: 3.94

When comparing the plan to the previous plan before creating the index, we can see that the query's total cost has decreased significantly: from 2548.28 to just 3.94. This proves that the full table scan on table keyword was a bottleneck, which hindered the performance when searched by keyword. This was fixed with a single appropriate index. Now, the query uses non-unique key lookup on table keyword, and it is able to retrieve data without any full table scans.

SUMMARY

By creating two indexes and adding one new column, I have significantly optimized the first query's execution from section A. Creating an index on the year column optimized queries when searching by year. Creating an index on the keyword column in the keyword table reduced query costs when searching by keyword. The new citation_count column in the 'paper' table improved overall query performance, as it helped avoid one join and a full index scan on the 'citedby' table.

Here's a summarized comparison of total costs, showing that each change I made positively impacted performance, especially when searching for results with a specific keyword or year:

<u>Searching by title (starting with c) and year (2023)</u>

- Total cost before index on year was made: **5165.26**
- Total cost after index on year was made: **3728.95**

<u>Searching by title (starting with c) and year (2023) (Index already created on year)</u>

- Total cost before column was made: **3728**
- Total cost after column was made: **495.63**

<u>Searching by title (starting with c) and keyword ('Database')</u>

- Total cost before index on keyword was made: **2548**
- Total cost after index on keyword was made: **3.95**

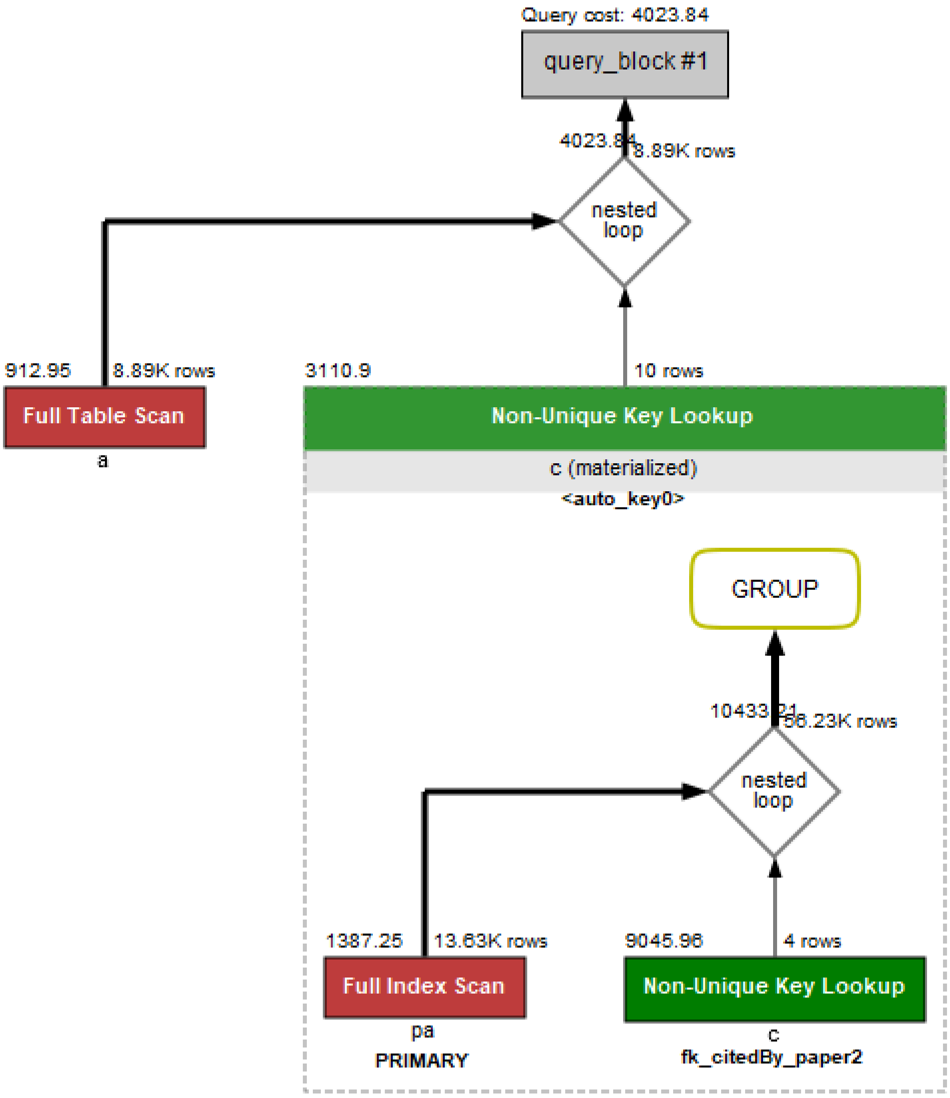
OPTIMIZING THE SECOND SEARCH

The A-section's second search searches certain author's co-parents from the paperauthors-table, and also counts the number of joint publications with each of them. Also, the search counts the number of received citations of a certain author. This query's performance could be improved by creating a summary table about the author's co-authors, and the number of joint publications with each of them. Also adding the number of received citations as a column to the author-table reduces the amount of calculation that the search has to go through.

Viivi Väisänen

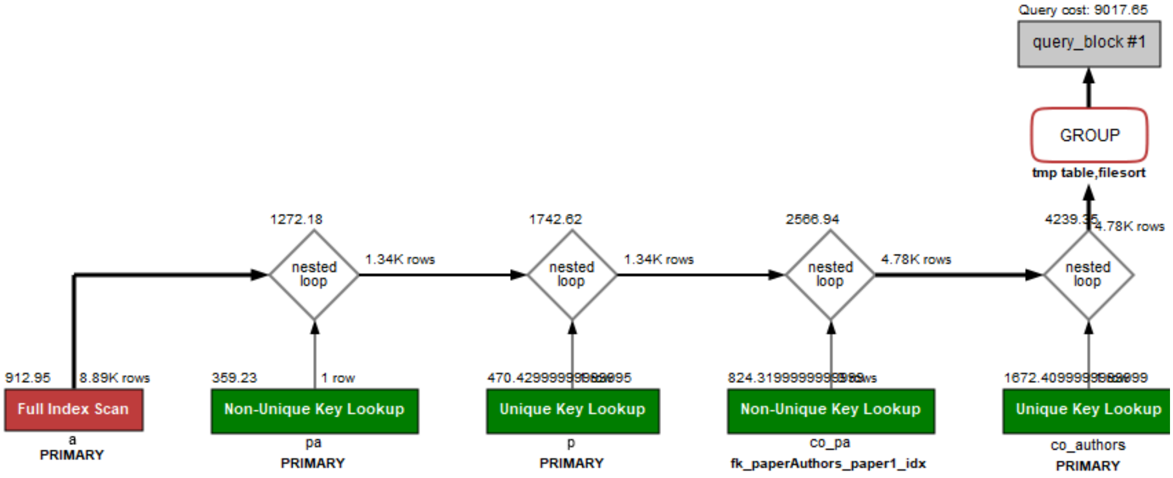
Let’s study section A’s second search execution, taking notes about the total costs. The search is conducted by three different SELECTs.

Here’s the first SELECT’s execution plan, which returns a specific author’s name, gender, country and total citations, filtered by authorID 10039885900:



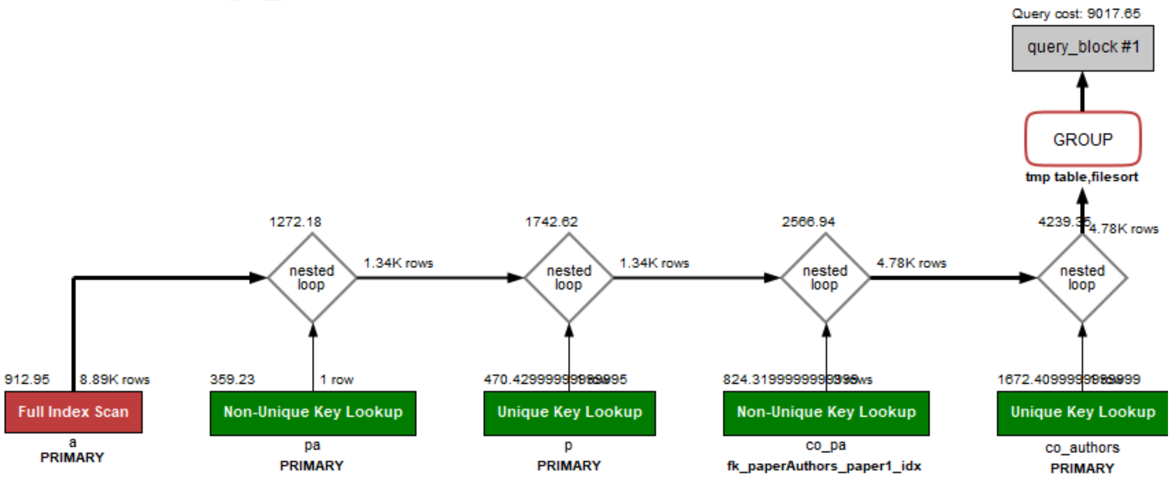
Total cost is 4023.84

The second SELECT returns all publications of a specific author, along with their title, year, source and co-authors. Here's the execution plan of it, filtered by authorID 10039885900:



Total cost is 9017.65

The third SELECT returns all co-authors of a specific author, and number of joint publications with each of them. There's the execution plan of it, filtered by authorID 10039885900:



Total cost is 9017.65

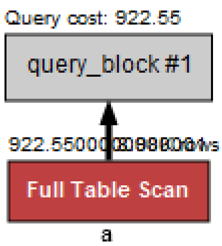
MODIFICATIONS:

I created a new summary table co_parent_summary, added a new column recieved_citations_count to the table author and transferred all the data to the new table and column. (you can see these in SQL in the next section [C].)

Let's change the queries and then study the execution plans to see if the executions have changed to less costly.

I added a new column to the author-table - that's why we don't need JOINs to return a specific author's number of total citations.
Here's the new version of the **first SELECT** and it's execution plan:

```
-- This query returns a specific author's name, gender, country
and total citations.
SELECT
     a.name AS author_name,
     a.gender,
     a.affiliation,
     a.country,
     a.recieved_citations_count
FROM
     publications.author a
     where
     a.authorID = 10039885900 ; -- INSERT THE AUTHOR'S ID HERE
```



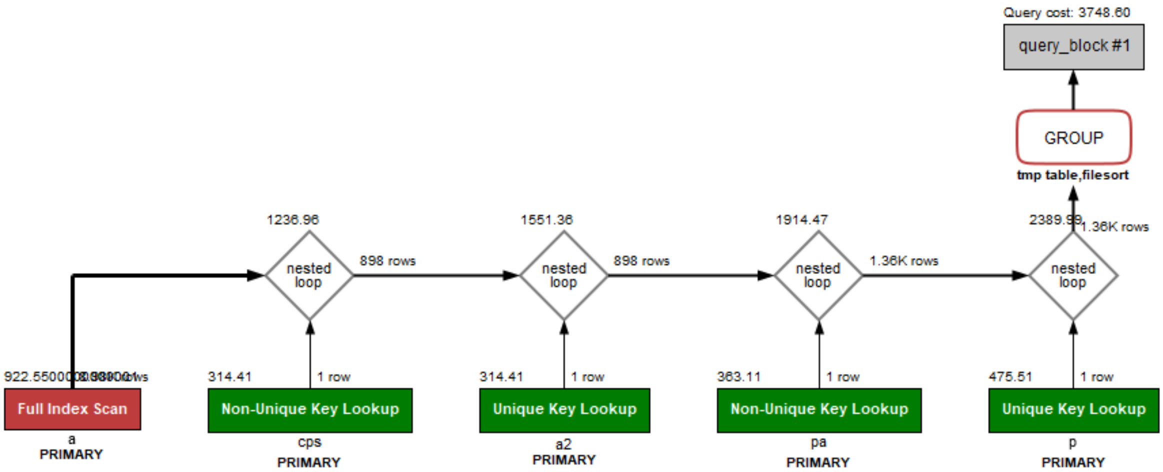
Total cost: 922.55

Compared to the previous plan before the modifications, the cost has decreased significantly: from 4023.84 to 922.5. This is all thanks to the new column added to the author-table. The query does not have to calculate the number of received citations or join tables anymore, and it only does one full table scan to get the wanted results, which results in lower cost.

Viivi Väisänen

I created a new co_parent_summary table, let's change the second SELECT to utilize the new table for finding co-authors.
Here's a new version of the **second SELECT** and it's execution plan:

```
-- This query returns all publications of a specific author, along
with their title, year, source and co-authors
SELECT
    p.title AS paper_title,
    p.year AS publication_year,
    p.source AS publication_source,
    GROUP_CONCAT(DISTINCT a2.name ORDER BY a2.name SEPARATOR ',
') AS co_authors
FROM
    publications.paper p
JOIN
    publications.paperauthors pa ON p.paperID = pa.paperID
JOIN
    publications.author a ON pa.authorID = a.authorID
LEFT JOIN
    publications.co_parent_summary cps ON cps.authorID =
a.authorID
LEFT JOIN
    publications.author a2 ON cps.co_authorID = a2.authorID
WHERE
    a.authorID = 10039885900 -- Insert the author's ID here
GROUP BY
    p.title, p.year, p.source;
```



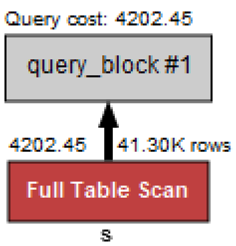
Total cost: 3748.60

Compared to the previous execution plan before the modifications, the query's cost has dropped from 9017.65 to 3748.60. The change in the query is that it searches the co-authors from table co_parent_summary, when in the previous version it searched co-authors by joining paperauthors-table to itself. Thanks to the summary table, the query does not have to do as much work to search for the co-authors, and that is why the total cost has dropped.

Viivi Väisänen

Here's a new version of the **third SELECT** and it's execution plan:

```
-- This query retrieves all co-authors of a specific author and
number of joint publications with each of them
SELECT
    co_authorID,
    joint_publications
FROM
    publications.co_parent_summary s
WHERE
    authorID = 10039885900; -- INSERT THE AUTHOR'S ID HERE
```



Total cost: 4202.45

Compared to the previous plan before the modifications, the cost has decreased from 9017.65 to 4202.45. This is thanks to the new summary table co_parent_summary. Because of the new table, we avoided all JOINs and that is why the query has to do only one full table scan to get the results.

SUMMARY

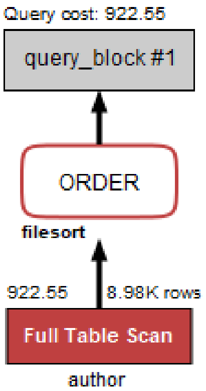
By creating a co_parent_summary table and adding a new column recieved_citations_count to the actor table, we significantly reduced the total cost of the query execution for the second search in task A.

The cost before denormalizing: 4023.84 + 9017.65 + 9017.65 = 22059.14
The cost after denormalizing: 922.55 + 3748.60 + 4202.45 = 8873.6

This proves that the modifications I made optimize the second search queries in task A.

OPTIMIZING THE THIRD SEARCH

The third search of section A searches the 10 most prolific authors by total number of publications, filtered by country, gender or affiliation. Let's take note of the query's total cost before modifications. Results are filtered by country 'Brazil':

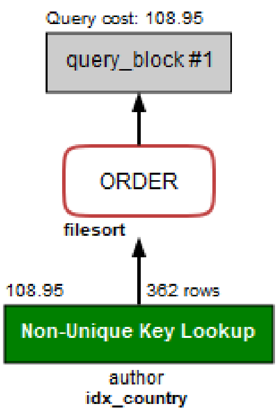


Total cost: 922.55

MODIFICATIONS

I created indexes in the author table on columns that appear in where-caluses; country and gender. These indexes are made to avoid full table scan on table author. (you can see the creation of the indexes in SQL in the next section [C].) I also tried to create an index on affiliation, because results are searched based on that too. The index did not affect the execution cost, because the query was not able to find the index. This was because there are so little duplicate values in the affiliation column: no indexes can help!

After creating the indexes, the same query's execution plan looks like this (results are filtered again by country 'Brazil'):

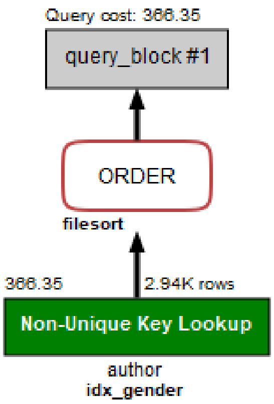


Total cost: 108.95

When compared to the previous plan, we can see that we did in fact avoid full table scan, thanks to the created indexes. The query is able to use the index made on the column country. The query does a non-unique key lookup to find the results from the table author, filtered by country. Also, the total cost has dropped from 922.55 to just 108.95.

Let's filter the results by gender too, taking notes about the query costs.

Filtered by gender 'Female':



Total cost: 366.35

Compared to the execution plan before creating the indexes, we can see that the total cost has dropped from 922.55 to 366.35. The reason for decreased cost is, that the query does not have to do a full table scan on the table author, because it is able to use the index made on the column gender.
When compared to the previous query, which searches based on country, the total cost is higher. This is because the gender column contains only two distinct values, whereas the country column has a smaller number of repeated values.

C. Perform the necessary modifications to your original design and/or implementation to optimize for these features. If needed, write the necessary schema modifications/indices/triggers in SQL to implement your solution.

```
use publications;

-- new index
create index year_idx on publications.paper(year);

-- New column citation_count in table paper
ALTER TABLE publications.paper
ADD COLUMN citation_count INT DEFAULT 0;

-- Data insertion to the citation_count column:
UPDATE publications.paper p
LEFT JOIN (
    SELECT citedPaper, COUNT(*) AS citation_count
    FROM publications.citedby
```

```
    GROUP BY citedPaper
) c ON p.paperID = c.citedPaper
SET p.citation_count = IFNULL(c.citation_count, 0);



-- new index on keyword in keyword table
create index idx_keyword on publications.keyword(keyword);


-- The new summary-table co_parent_summary about co-parents:
CREATE TABLE IF NOT EXISTS `publications`.`co_parent_summary` (
    `authorID` VARCHAR(30) NOT NULL,
    `co_authorID` VARCHAR(30) NOT NULL,
    `joint_publications` INT NOT NULL,
    PRIMARY KEY (`authorID`, `co_authorID`),
    CONSTRAINT `fk_co_parent_summary_author1`
        FOREIGN KEY (`authorID`)
        REFERENCES `publications`.`author` (`authorID`),
    CONSTRAINT `fk_co_parent_summary_author2`
        FOREIGN KEY (`co_authorID`)
        REFERENCES `publications`.`author` (`authorID`)
)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb3;


-- New column called recieved_citations_count to the author-table:
ALTER TABLE `publications`.`author`
ADD COLUMN `recieved_citations_count` INT NULL DEFAULT 0;


-- Data insertion to the co_parent_summary:
INSERT INTO `publications`.`co_parent_summary` (authorID,
co_authorID, joint_publications)
SELECT
    pa.authorID,
    co_pa.authorID as co_authorID,
    -- let's count how many times the author and
    -- co-author is occuring in the paperauthors-table:
    COUNT(*) as joint_publications
FROM
    `publications`.`paperauthors` pa
    -- Let's join paperauthors with itself (self-join). Let's add
condition, which rules that authorID and co-author
    -- can't be the same ID (AND pa.authorID != co_pa.authorID)
    JOIN `publications`.`paperauthors` co_pa ON pa.paperID =
co_pa.paperID AND pa.authorID != co_pa.authorID
group by
    pa.authorID, co_pa.authorID;

-- Data insertion to the column recieved_citations_count:
UPDATE publications.author a
JOIN (
```

```
    SELECT
        pa.authorID,
        COUNT(c.citedPaper) AS total_citations
    FROM
        publications.paperauthors pa
    JOIN
        publications.citedby c ON pa.paperID = c.citedPaper
    GROUP BY
        pa.authorID
) cit ON a.authorID = cit.authorID
SET a.recieved_citations_count = cit.total_citations;




-- New indexes
CREATE INDEX idx_country ON publications.author(country);
CREATE INDEX idx_gender ON publications.author(gender);




-- BELOW ARE THE NECESSARY TRIGGERS THAT KEEP THE DATA UP TO DATE


-- AFTER INSERT trigger, which keeps the author table's
totalNumbPublications column up to date
-- After new paper is inserted, the column in author table is
updated.
delimiter //

CREATE TRIGGER paper_insert
     AFTER INSERT ON publications.paper
    FOR EACH ROW
BEGIN
     UPDATE publications.author
    SET totalNumbPublications = totalNumbPublications + 1
    WHERE author.authorID = NEW.author.authorID;
END //

delimiter ;


-- This AFTER DELETE trigger keeps the author table's
totalNumbPublications column up to date
-- After a paper is deleted, the column in author table is
updated.
delimiter //

CREATE TRIGGER paper_delete
     AFTER DELETE ON publications.paper
    FOR EACH ROW
BEGIN
```

```
    UPDATE publications.author
    SET totalNumbPublications = totalNumbPublications -1
    WHERE author.authorID = OLD.author.authorID;
END //

delimiter ;


-- This AFTER DELETE trigger keeps the author table's
totalNumbPublications column up to date
-- If an author is deleted from a paper's authors, the
totalNumbPublications column is updated
DELIMITER //

CREATE TRIGGER paper_author_delete
     AFTER DELETE ON publications.paperAuthors
     FOR EACH ROW
BEGIN
    UPDATE author
    SET totalNumbPublications = totalNumbPublications - 1
    WHERE authorID = OLD.authorID;
END//

DELIMITER ;




-- This AFTER INSERT trigger keeps the author table's column
recieved_citations up to date, when
-- new citation is inserted to the citedby table
DELIMITER //

CREATE TRIGGER citedby_insert
     AFTER INSERT ON publications.citedby
     FOR EACH ROW
BEGIN
    UPDATE publications.author
    SET received_citations = received_citations + 1
    WHERE authorID = (SELECT authorID FROM publications.paper
WHERE paper.paperID = NEW.citedby.citedPaper);
END//

DELIMITER ;



-- This AFTER DELETE trigger keeps the author table's column
recieved_citations up to date, when
-- citation is deleted from the citedby table
DELIMITER //

CREATE TRIGGER citedby_delete
AFTER DELETE ON publications.citedby
```

```
FOR EACH ROW
BEGIN
    UPDATE publications.author
    SET received_citations = received_citations - 1
    WHERE authorID = (SELECT authorID FROM publications.paper
WHERE paper.paperID = OLD.citedby.citedPaper);
END//

DELIMITER ;



-- This AFTER UPDATE trigger keeps the data up to date, when data
in citedby table is updated
DELIMITER //

CREATE TRIGGER citedby_update
AFTER UPDATE ON publications.citedby
FOR EACH ROW
BEGIN
    -- Decreases the citation count for the old cited paper's
author
    UPDATE publications.author
    SET received_citations = received_citations - 1
    WHERE author.authorID = (SELECT authorID FROM
publications.paper WHERE paper.paperID =
OLD.citedby.citedPaperID);

    -- Increases the citation count for the new cited paper's
author
    UPDATE publications.author
    SET received_citations = received_citations + 1
    WHERE author.authorID = (SELECT authorID FROM
publications.paper WHERE paper.paperID =
NEW.citedby.citedPaperID);
END//

DELIMITER ;
```

Stage 3: Deployment

Before putting our database in production, we need to create a specific database user for the web application that will make use of our database. *For this purpose, provide the SQL code to create a new user named 'web' with password 'research', and give this user only reading privileges to the database.*

```
CREATE USER 'web'@'localhost' IDENTIFIED BY 'research';
GRANT SELECT ON publications.* TO 'web'@'localhost';
-- Now the user has SELECT privilege on all of the tables in
```

```
schema publications.
FLUSH PRIVILEGES;
```

Stage 4: Data analysis and visualization

The client is interested in generating a report from the data stored in the database. Namely, they are interested in the following:

A. The evolution of the number of articles per year

This SQL query retrieves distinct years and counts the number of articles for each year:

```
select distinct p.year, count(pa.paperID) as article_count
     from publications.paper p
     join publications.paper pa on p.year = pa.year
where p.type = 'Article' and pa.type = 'Article'
group by p.year
order by year asc;
```

year	article_count
2010	16
2011	25
2012	100
2013	961
2014	2916
2015	4624
2016	12321
2017	13689
2018	10609
2019	17689
2020	15129
2021	9801
2022	9025
2023	12544
2024	900

B. The number of articles per country per year

The original database did not have data about the article's countries. In the single-table-database, the ‘´country' column was information related to the author.

C. The number of articles per author per year, for the top 20 most productive authors

This query returns the top 20 most productive authors and displays the number of articles they have authored each year:

```
SELECT
    author.authorID,
    author.name AS author_name,
    -- column for each year's number of articles per year
    SUM(CASE WHEN p.year = 2011 AND p.type = 'Article' THEN 1
ELSE 0 END) AS articles_2011,
```

```
    SUM(CASE WHEN p.year = 2012 AND p.type = 'Article' THEN 1
ELSE 0 END) AS articles_2012,
    SUM(CASE WHEN p.year = 2013 AND p.type = 'Article' THEN 1
ELSE 0 END) AS articles_2013,
    SUM(CASE WHEN p.year = 2014 AND p.type = 'Article' THEN 1
ELSE 0 END) AS articles_2014,
    SUM(CASE WHEN p.year = 2015 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2015,
    SUM(CASE WHEN p.year = 2016 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2016,
    SUM(CASE WHEN p.year = 2017 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2017,
    SUM(CASE WHEN p.year = 2018 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2018,
    SUM(CASE WHEN p.year = 2019 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2019,
    SUM(CASE WHEN p.year = 2020 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2020,
    SUM(CASE WHEN p.year = 2021 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2021,
    SUM(CASE WHEN p.year = 2022 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2022,
    SUM(CASE WHEN p.year = 2023 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2023,
    SUM(CASE WHEN p.year = 2024 AND p.type = 'Article'  THEN 1
ELSE 0 END) AS articles_2024,
    COUNT(DISTINCT p.paperID) AS total_publications_count
FROM
    publications.author
JOIN
    publications.paperAuthors pa ON author.authorID = pa.authorID
JOIN
    publications.paper p ON pa.paperID = p.paperID
GROUP BY
    author.authorID,
    author.name
ORDER BY
    total_publications_count DESC
LIMIT 20; -- limits the rows by 20 --> we get 20 most productive
authors
```

authorID	author_name	articles_2011	articles_2012	articles_2013	articles_2014	articles_2015	articles_2016	articles_2017	articles_2018	articles_2019	articles_2020	articles_2021	articles_2022	articles_2023	articles_2024	total_publications_count
23037278500	Störl U.	0	0	0	0	0	0	0	0	0	0	0	1	0	0	32
6601959236	Holanda M.	0	0	0	0	1	0	2	1	1	0	0	0	0	0	31
16317498000	Scherzinger S.	0	0	0	0	0	0	0	0	0	0	1	0	0	0	30
16039653400	Klettke M.	0	0	0	0	0	0	0	0	0	0	0	1	0	1	27
8847095400	Bernardino J.	0	0	0	0	4	0	1	0	0	1	0	1	3	0	24
6507596657	Zurfluh G.	0	0	0	0	0	0	0	1	0	0	1	1	1	0	22
15125412100	Molina J.G.	0	0	0	0	1	0	0	0	1	0	1	1	1	1	21
57202521929	Joosen W.	0	0	0	0	0	0	1	2	1	1	2	0	0	1	21
6505857335	Teste O.	0	0	0	0	0	0	0	0	1	1	0	0	0	0	20
6602113319	Gargouri F.	0	0	0	0	0	0	0	1	0	0	1	0	2	0	19
57221114969	Sevilla Ruiz D.	0	0	0	0	0	0	0	0	1	0	1	1	1	0	19
57206513787	Abdelhedi F.	0	0	0	0	0	0	0	1	0	0	1	1	1	0	18
36761221300	Ma K.	0	0	1	2	2	1	2	0	0	0	0	0	0	0	17
23388937400	Mello R.D.S.	0	0	0	0	0	1	0	0	1	2	0	1	0	0	17
56006924900	Rafique A.	0	0	0	0	0	0	1	2	1	0	1	0	0	0	16
22735445100	Van Landuyt D.	0	0	0	0	0	0	1	2	1	0	1	0	0	1	16
57192639518	Brahim A.A.	0	0	0	0	0	0	0	1	0	0	1	0	0	0	15
12645423500	Villari M.	0	0	0	1	0	0	0	1	1	1	0	0	0	0	15
55800889500	Carey M.J.	0	0	0	0	0	0	0	0	0	1	0	0	1	0	15
56954758100	Konstantinou I.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15

D. The distribution in the number of citations per gender

This query calculates the total number of citations received by both genders:

```
SELECT distinct
    gender,
    SUM(recieved_citations_count) AS total_citations
FROM
    author
GROUP BY
    gender;
```

gender	total_citations
Female	21145
Male	44153

E. The number of citations in contrast with the number of authors in a paper

```
SELECT
    author_count,
    round(AVG(citation_count), 2) AS average_citations
FROM
     (
       SELECT
           pa.paperID,
             COUNT(DISTINCT cited.paperID) AS citation_count,
             COUNT(DISTINCT pa.authorID) AS author_count
       FROM
             paperauthors pa
       LEFT JOIN
           citedby cited ON pa.paperID = cited.paperID
       GROUP BY
           pa.paperID
    ) AS subquery
GROUP BY
  author_count
order by author_count asc;
```

author_count	average_citations
1	0.88
2	0.90
3	0.91
4	0.90
5	0.91
6	0.89
7	0.92
8	0.91
9	0.90
10	0.75
11	1.00
12	1.00
13	1.00
14	1.00
16	1.00
17	1.00
18	1.00
20	1.00
21	0.67
23	1.00
52	1.00

For each of the previous analyses decide which is the most suitable visual representation, and write the SQL queries to extract the necessary data to generate such visualizations.

Stage 6: Anonymization

The client is thinking of releasing some of their private data about researchers' salaries as open data for scientific purposes after anonymization. The dataset looks like this:

Author	Affiliation	Country	Npub	Gender	Salary
Molina J.G.	Faculty of Computer Science, University of Murcia, Murcia	Spain	21	Female	25000
Klettke M.	Faculty of Computer Science and Data Science, University of Regensburg, Regensburg	Germany	27	Male	110000
Ruiz D.S.	Faculty of Computer Science, University of Murcia, Murcia	Spain	1	Male	71000
Chillon A.H.	Faculty of Computer Science, University of Murcia, Murcia	Spain	11	Male	44000
Sethy P.K.	Guru Ghasidas Vishwavidyalaya, Department of Ece, C.G, Bilaspur	India	1	Male	89000
Somasundar A.V.S.S.	Sagi Rama Krishnam Raju (S.R.K.R) Engineering College (A), Department of Information Technology, A. P, Bhimavaram, 534204	India	1	Male	35000
Kumari Behera S.	Vssut Burla, Department of Cse, Odisha	India	1	Male	74000

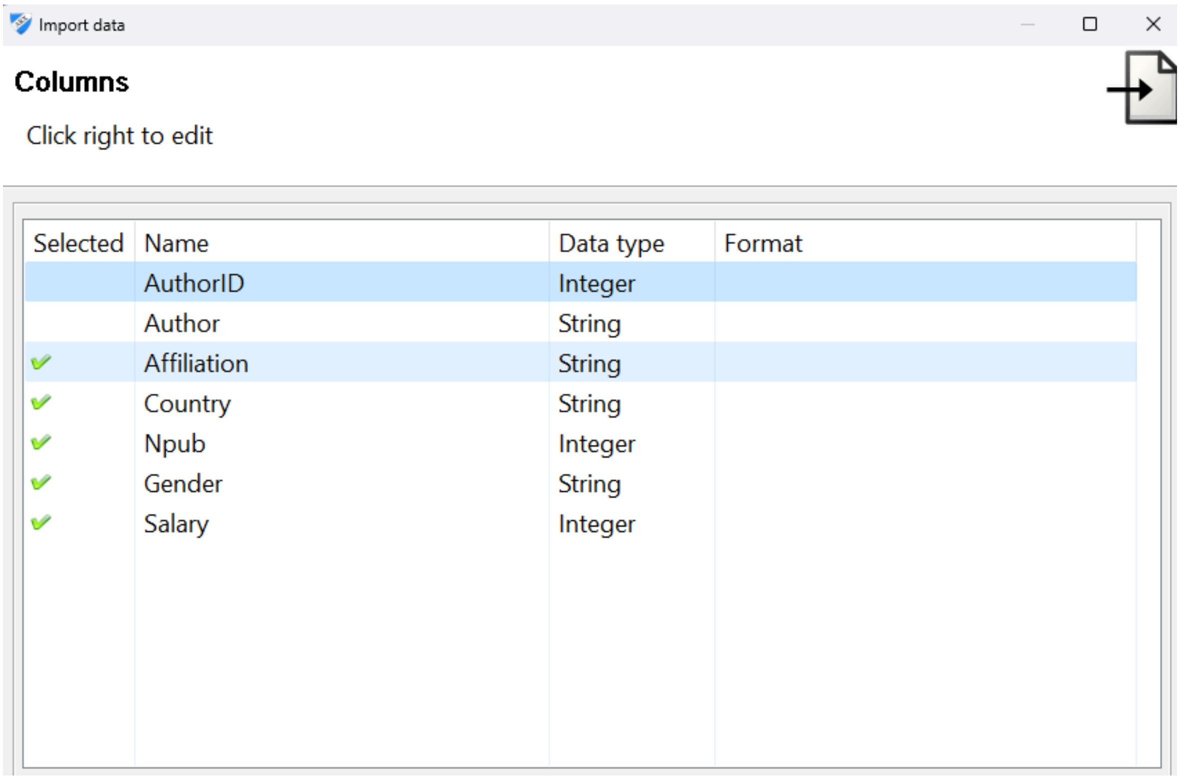
The client is worried that even after anonymization, the fields on the dataset that are public in the scientific database (affiliation, country, number of publications [Npub] and Gender) would allow to identify the researchers and therefore find out their salary.
To avoid this, use the ARX tool (or similar) and apply the necessary anonymization techniques to ensure K-anonimity where K=10 and l-diversity where l= 3 while keeping most of the usefulness of the dataset. Document the anonymization techniques that you have tried and the final decision, and paste a screenshot of the dataset in which these techniques can be seen and the anonymization report from ARX.

To achieve k-autonomity and l-diversity, we have to first identify quasi-identifiers and sensitive attributes in our dataset. For the analysis, the authorID and the author's name are not significant pieces of information, and they can be used to identify a specific author too easily. That's why we won't be choosing those attributes in the ARX tool, when we import the data.

- Quasi-identifiers: affiliation, country, Npub, gender
- Sensitive data: salary

My ideas to achieve k=10 k-anonimity and l-3 l-diversity:
- Change countries as continents: Europe, Africa, Oceania, South-America or America.
- Change the nPub into ranges. For example 1-5, 5-10, 10-15, 15-20…
- Change Affiliation to ‘University’, or replace with *.

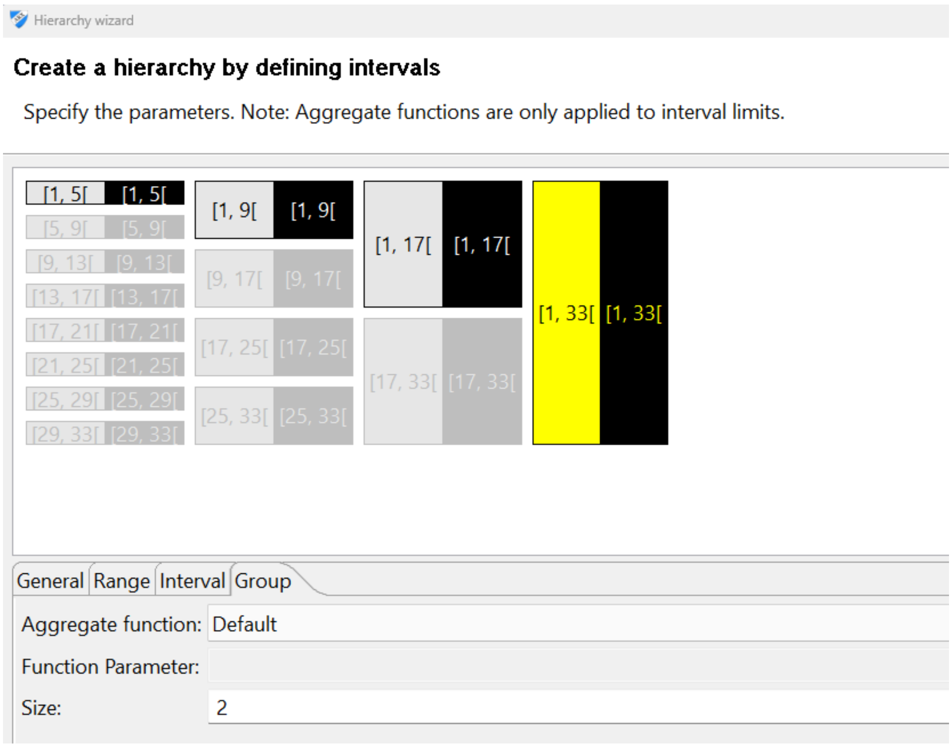
Let’s import the dataset to the ARX Anonymization tool, and select the columns that are needed for analyzing the dataset. I didn’t include AuthorID and Author’s name.:



After importing the dataset, I set attributes affiliation, country, Npub and gender to quasi-identifiers, and salary as sensitive data.

I started creating hierarchies to the attributes with the ARX tool to the quasi-identifiers. I started from column Npub, and created hierarchy by defining intervals:

Viivi Väisänen



I also changed all of the affiliations to constant value 'University.'

Review the hierarchy

Overview of groups and values

#Groups
4738
1

Table

Level-0	Level-1
'Politehnica' Univ...	University
'Politehnica' Univ...	University
(FORMAS Resear...	University
(FORMAS Resear...	University
3D GIS Research ...	University
5/411 Mannarthi...	University
ADD\Fleetenergi...	University
AG Heterogene I...	University
AGH University o...	University
AGH University o...	University

Next, I changed all countries to their continents. This was the most difficult phase. I created a hierarchy using ordering, and in order to change each country's name to its respective continent, I needed to have the countries listed so that, for example, the European countries came first, then the Asian countries, then Africa, and so on. I chose the ordinal order, from which I got a CSV file of the distinct countries present in the dataset. After that, I opened the list in a text editor and manually organized the countries. Then I imported the CSV file into the ARX tool and grouped the countries correctly. I counted the number of countries in each continent from the file and created the correct groups for each continent in order.

Review the hierarchy

Overview of groups and values

#Groups

#Groups
116
6
1

Table

Level-0	Level-1	Level-2
Malta	Europe	*
Netherlands	Europe	*
Norway	Europe	*
Poland	Europe	*
Portugal	Europe	*
Slovakia	Europe	*
Slovenia	Europe	*
Spain	Europe	*
Sweden	Europe	*
Switzerland	Europe	*
Turkey	Europe	*
Ukraine	Europe	*
United Kingdom	Europe	*
Bahrain	Asia	*
Bangladesh	Asia	*
Brunei Darussalam	Asia	*
China	Asia	*
Hong Kong	Asia	*
India	Asia	*
Indonesia	Asia	*
Iran	Asia	*
Iraq	Asia	*
Japan	Asia	*

Here's ARX-tool's analyze about **original** dataset's quasi-identifiers:

Quasi-identifier	Distincti...	Separati...
Salary	1.01235%	98.62757%
Country	1.29047%	93.88514%
Affiliation	52.70887%	99.97864%
Npub, Gender	0.47836%	64.54693%
Gender, Salary	1.64646%	99.1846%
Country, Gender	2.32506%	96.59275%
Country, Npub	4.51663%	95.91211%
Npub, Salary	5.69585%	99.11934%
Country, Salary	32.10591%	99.91695%
Affiliation, Country	52.77561%	99.97889%
Affiliation, Npub	59.61731%	99.98299%
Affiliation, Gender	66.47013%	99.98786%
Affiliation, Salary	98.78741%	99.99973%
Country, Npub, Gender	6.95294%	97.71478%
Npub, Gender, Salary	7.7428%	99.4747%
Country, Gender, Salary	40.8833%	99.9506%
Country, Npub, Salary	45.48893%	99.9442%
Affiliation, Country, Npub	59.67293%	99.98319%
Affiliation, Country, Gender	66.55913%	99.98798%
Affiliation, Npub, Gender	71.95461%	99.9904%
Affiliation, Country, Salary	98.79853%	99.99973%
Affiliation, Npub, Salary	98.99878%	99.99977%
Affiliation, Gender, Salary	99.29914%	99.99984%
Country, Npub, Gender, Salary	53.88809%	99.96653%
Affiliation, Country, Npub, Gender	72.02136%	99.99048%
Affiliation, Country, Npub, Salary	99.0099%	99.99978%
Affiliation, Country, Gender, Salary	99.31027%	99.99985%
Affiliation, Npub, Gender, Salary	99.38814%	99.99986%
Affiliation, Country, Npub, Gender, Salary	99.39927%	99.99987%

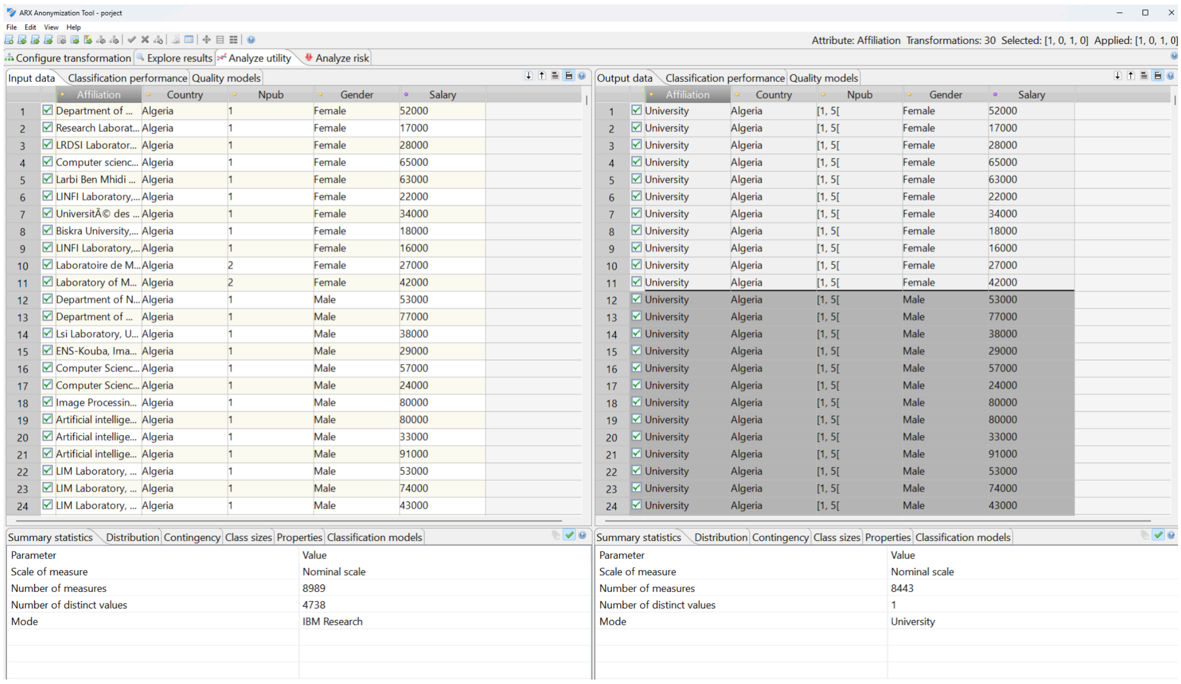
As seen in the picture below, the Affiliation attribute on quasi-identifier, which appears in many combinations of quasi-identifiers, can be used to identify almost all individuals in the dataset. Just by affiliation, we can identify 52.70..% of persons in the dataset, which was the reason I changed all of the affiliations to just 'University'

Viivi Väisänen

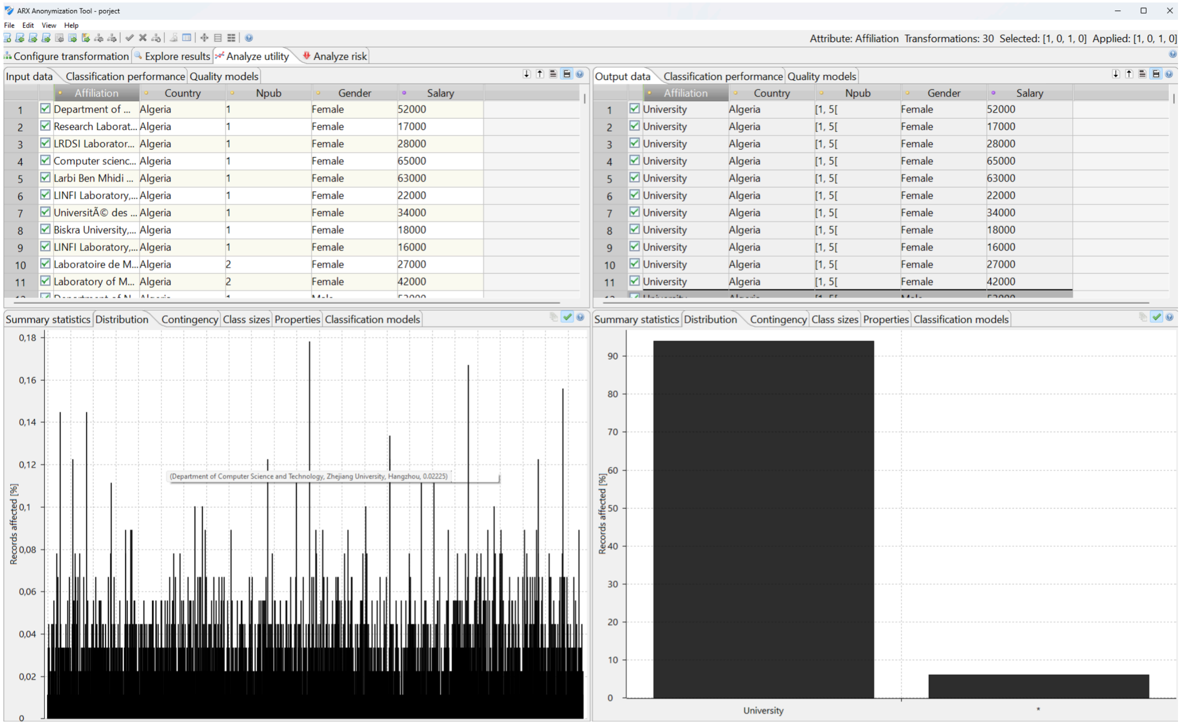
Before anonymising, I created two privacy models: 10-Anonymity (k=10), and l-diversity (l=3) on attribute Salary.

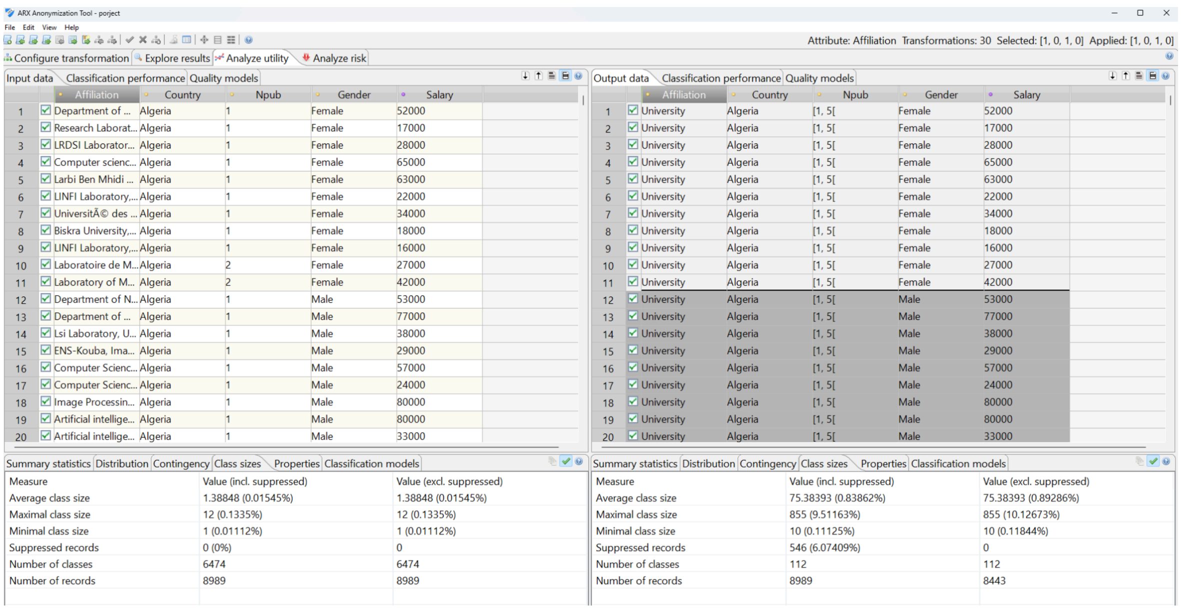
Anonymization results

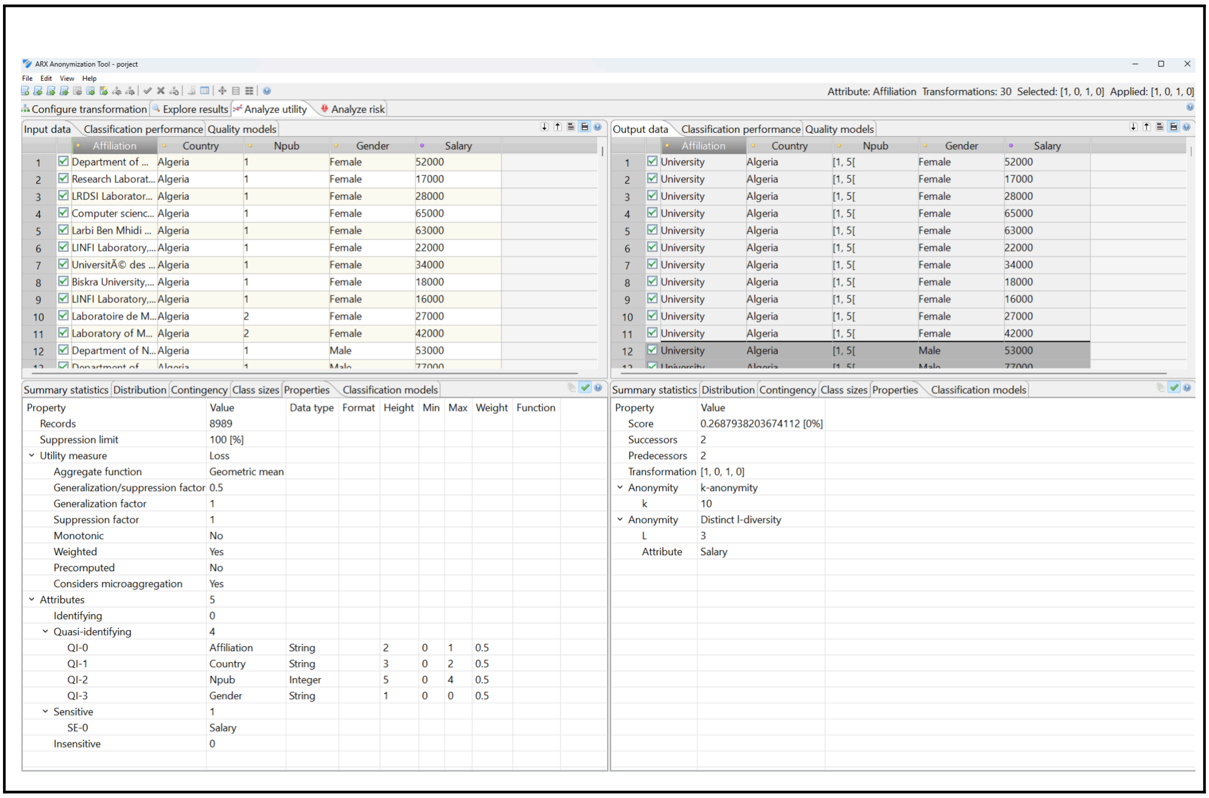
Here are screenshots of the results after the anonymization process.
The tool did not end up using the hierarchy I created for the country attribute, probably because it was no longer necessary after generalizing the Affiliation to 'University.'



Viivi Väisänen







Extra (for the students aiming for 5 ECTS)

Given the requirements of our database, discuss in detail whether a NoSQL solution would be more suitable and, if so, explain which one and why. Regardless of which one you have chosen, define how the schema would look like in MongoDB and repeat the queries for stage 2 in MongoDB.

Submission

Clone or download this document and reply in the provided blanks (use all the space you need). Submit the filled-in document on eLearn. It is enough that one person in the group submits it but do not forget to add the names of all the group members in the header of the document.