Introduction

In this project, we have gained experience in various stages of database design: schema design, data acquisition, data transformation, querying and indexing. Moreover, we have worked with large dataset and have understood the importance of efficient SQL queries.

For this project, we have utilized MySQL, Excel, and Java as the main programming tools.

1. Schema Design and Data Acquisition

1.1 E/R diagram

The ER diagram of the schema is provided below.

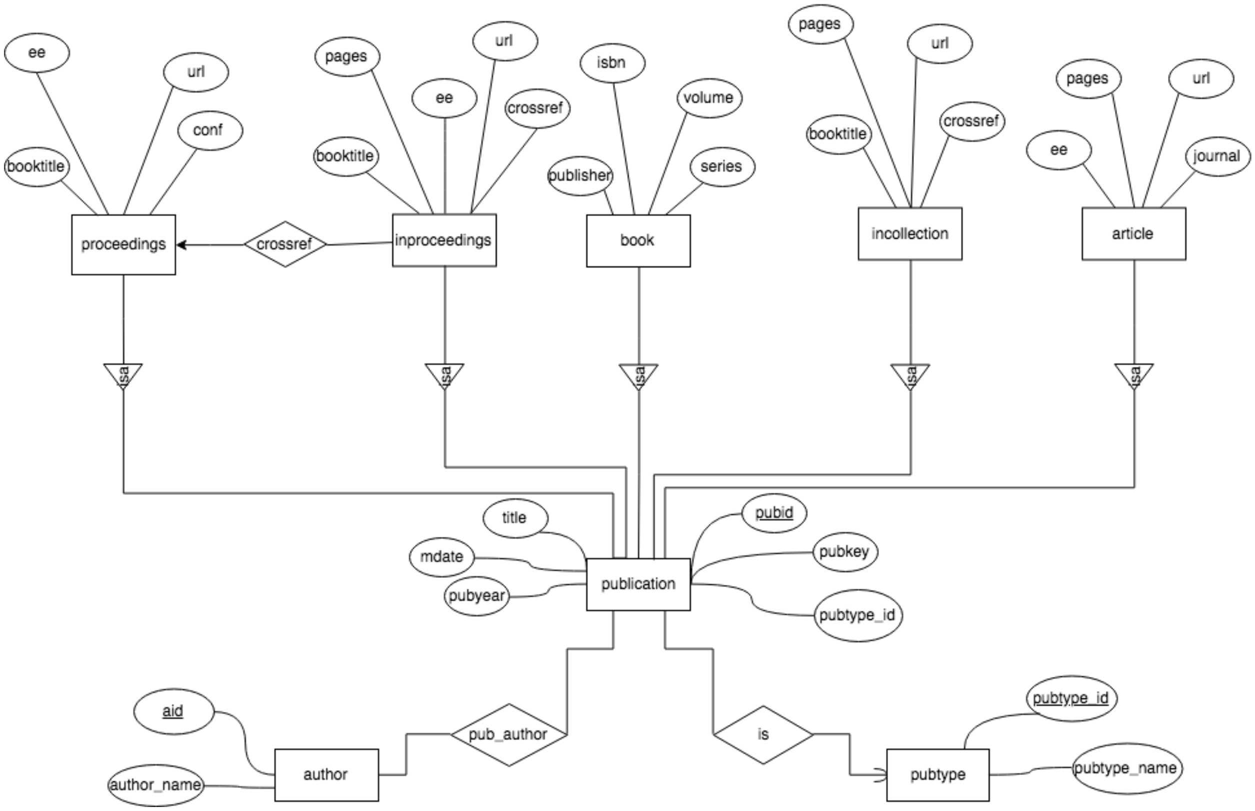


Figure 1. ER diagram

1.2 Commands for creating tables

```
-- Table structure for table article
CREATE TABLE article (
 pubid   int(11) NOT NULL,
 pages   varchar(255) DEFAULT NULL,
 ee      varchar(255) DEFAULT NULL,
 url     varchar(500) DEFAULT NULL,
 journal varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubid),
 CONSTRAINT article_publication_pubid_fk FOREIGN KEY (pubid) REFERENCES
publication (pubid)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table author
CREATE TABLE author (
 aid        int(11) NOT NULL,
 authorname varchar(255) DEFAULT NULL,
 PRIMARY KEY (aid),
 UNIQUE KEY author_authorname_uindex (authorname)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table book
CREATE TABLE book (
 pubid     int(11) NOT NULL,
 publisher varchar(255) DEFAULT NULL,
 isbn      varchar(255) DEFAULT NULL,
 series    varchar(255) DEFAULT NULL,
 volume    varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubid),
 CONSTRAINT book_publication_pubid_fk FOREIGN KEY (pubid) REFERENCES
publication (pubid)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table incollection
CREATE TABLE incollection (
 pubid     int(11) NOT NULL,
 booktitle varchar(255) DEFAULT NULL,
 pages     varchar(255) DEFAULT NULL,
 crossref  varchar(255) DEFAULT NULL,
 url       varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubid),
 CONSTRAINT incollection_publication_pubid_fk FOREIGN KEY (pubid) REFERENCES
publication (pubid)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table inproceedings
CREATE TABLE inproceedings (
 pubid          int(11) NOT NULL,
 booktitle      varchar(300) DEFAULT NULL,
 pages          varchar(255) DEFAULT NULL,
 crossref_pubid int(11)      DEFAULT NULL,
 ee             varchar(255) DEFAULT NULL,
 url            varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubid),
 KEY inproceedings_crossref_pubid_fk (crossref_pubid),
 CONSTRAINT inproceedings_crossref_pubid_fk FOREIGN KEY (crossref_pubid)
REFERENCES proceedings (pubid)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table proceedings
CREATE TABLE proceedings (
 pubid     int(11) NOT NULL,
 conf      varchar(255) DEFAULT NULL,
 ee        varchar(255) DEFAULT NULL,
 url       varchar(500) DEFAULT NULL,
 booktitle varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubid),
 CONSTRAINT proceedings_publication_pubid_fk FOREIGN KEY (pubid) REFERENCES
publication (pubid)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table pub_author
CREATE TABLE pub_author (
 pubid int(11) NOT NULL,
 aid   int(11) NOT NULL,
 PRIMARY KEY (pubid, aid),
 KEY pub_author_aid_fk (aid),
 CONSTRAINT pub_author_aid_fk FOREIGN KEY (aid) REFERENCES author (aid)
   CONSTRAINT pub_author_aid_fk FOREIGN KEY (aid) REFERENCES author (aid)
```

```
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table publication
CREATE TABLE publication (
 pubid      int(11) NOT NULL,
 pubkey     varchar(255)  DEFAULT NULL,
 pubtype_id int(11)       DEFAULT NULL,
 title      varchar(1660) DEFAULT NULL,
 mdate      date          DEFAULT NULL,
 pubyear    year(4)       DEFAULT NULL,
 PRIMARY KEY (pubid),
 KEY publication_pubyear_index (pubyear),
 KEY publication_pubtype__fk (pubtype_id),
 CONSTRAINT publication_pubtype__fk FOREIGN KEY (pubtype_id) REFERENCES
pubtype (pubtype_id)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

-- Table structure for table pubtype
CREATE TABLE pubtype (
 pubtype_id int(11) NOT NULL,
 pubname    varchar(255) DEFAULT NULL,
 PRIMARY KEY (pubtype_id)
)
 ENGINE = InnoDB
 DEFAULT CHARSET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;
```

1.3 Data Acquisition

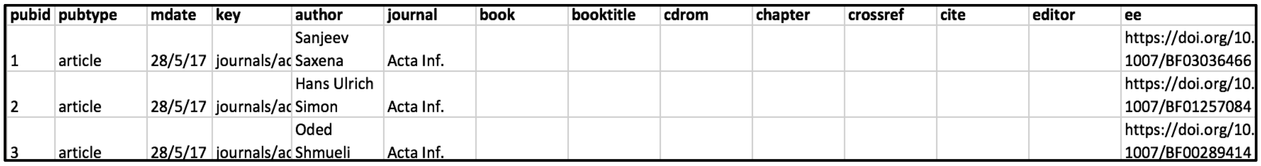
*Note: Please refer to Appendix for Java code for SAX parser.

We have created a SAX parser that is able to parse 'dblp.xml' into three files

1. publication.csv
2. author.csv
3. pub_author.csv.

1.3.1 publication.csv

This file stores all 7 subclasses of publication. Each row is a publication (pub_id = row number). The columns are all the possible fields for every publication (total 28 fields). If a publication does not have a certain field, that column is skipped.

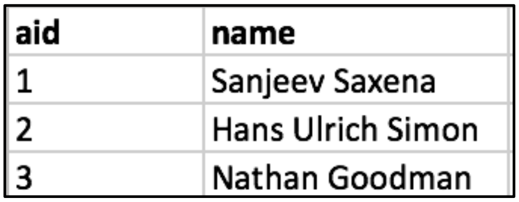


pubid	pubtype	mdate	key	author	journal	book	booktitle	cdrom	chapter	crossref	cite	editor	ee
1	article	28/5/17	journals/ac	Sanjeev Saxena	Acta Inf.								https://doi.org/10.1007/BF03036466
2	article	28/5/17	journals/ac	Hans Ulrich Simon	Acta Inf.								https://doi.org/10.1007/BF01257084
3	article	28/5/17	journals/ac	Oded Shmueli	Acta Inf.								https://doi.org/10.1007/BF00289414

Figure 2. Snapshot of 'publication.csv' file (28 columns are truncated due to lack of space)

1.3.2 author.csv

This file contains a list of authors. Each row has two columns: author_id, and author_name.



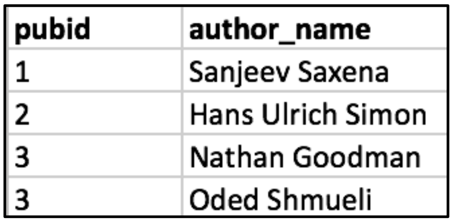
aid	name
1	Sanjeev Saxena
2	Hans Ulrich Simon
3	Nathan Goodman

Figure 3. Snapshot of 'author.csv' file

1.3.3 pub_author.csv

This file has two columns: pub_id and author_name.

It stores a many-to-many relationship between publications and authors. For example, in Fig4, pubid 3 has 2 authors: Nathan Goodman and Oded Shmueli respectively.



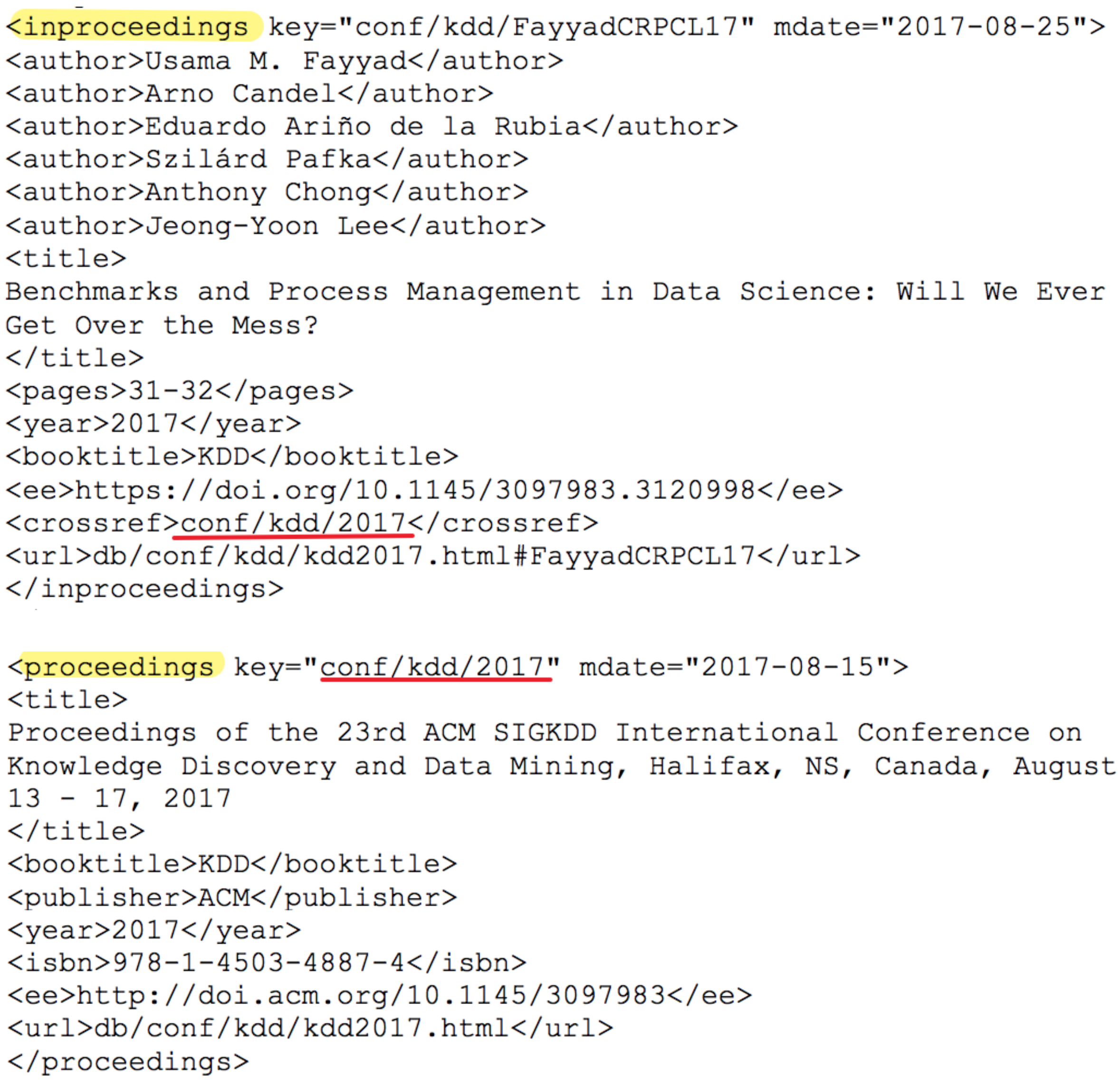
pubid	author_name
1	Sanjeev Saxena
2	Hans Ulrich Simon
3	Nathan Goodman
3	Oded Shmueli

Figure 4. Snapshot of 'pub_author.csv' file

1.3.4 Populate database tables

Process:

1. Import 3 '.csv' files into 3 temporary tables
   a. publication.csv -> publication_temp
   b. author.csv -> author_temp
   c. pub_author.csv -> pub_author_temp
2. Populate the 7 subclasses of publication by selecting necessary rows and columns from 'publication_temp' table.
   E.g:
   ```
   INSERT INTO inproceedings (pubid, booktitle, pages, crossref, ee, url)
   SELECT pub.pubid, pub.booktitle, pub.pages, pub.crossref, pub.ee,
   pub.url
   FROM publication
   WHERE pubtype = 1; //inproceeding
   ```
3. Remove duplicate authors in 'author.csv'
4. Update pub_author table by replacing author_name with author_id
5. Drop unnecessary columns in publication_temp table and keep only the columns appeared in pubSchema.
6. We noticed that inproceeding references proceeding
   E.g :



```
<inproceedings key="conf/kdd/FayyadCRPCL17" mdate="2017-08-25">
<author>Usama M. Fayyad</author>
<author>Arno Candel</author>
<author>Eduardo Ariño de la Rubia</author>
<author>Szilárd Pafka</author>
<author>Anthony Chong</author>
<author>Jeong-Yoon Lee</author>
<title>
Benchmarks and Process Management in Data Science: Will We Ever
Get Over the Mess?
</title>
<pages>31-32</pages>
<year>2017</year>
<booktitle>KDD</booktitle>
<ee>https://doi.org/10.1145/3097983.3120998</ee>
<crossref>conf/kdd/2017</crossref>
<url>db/conf/kdd/kdd2017.html#FayyadCRPCL17</url>
</inproceedings>

<proceedings key="conf/kdd/2017" mdate="2017-08-15">
<title>
Proceedings of the 23rd ACM SIGKDD International Conference on
Knowledge Discovery and Data Mining, Halifax, NS, Canada, August
13 - 17, 2017
</title>
<booktitle>KDD</booktitle>
<publisher>ACM</publisher>
<year>2017</year>
<isbn>978-1-4503-4887-4</isbn>
<ee>http://doi.acm.org/10.1145/3097983</ee>
<url>db/conf/kdd/kdd2017.html</url>
</proceedings>
```

we insert a column into inproceeding table so that it contains a crossref_pubid which stores the pubid of its corresponding proceeding.

7. Drop temporary tables

2. Queries and Optimizing Queries

We have created the required queries and ran on MySQL. The output is captured in screenshot and the execution time is recorded. Next, we prepare 2 more dataset which contains 50% and 25% of original DBLP data. Using 3 databases with different sizes, we aim to find the effect of size of database on query time.

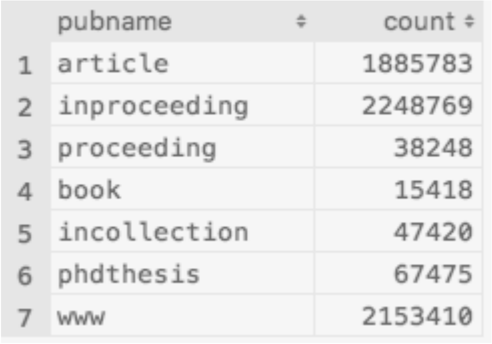
2.1 SQL queries & screen capture of results

# (1) For each type of publication, count the total number of publications of that type between 2000-2017.

Query:

```
WITH temp_count AS(
      SELECT pubtype_id, COUNT(*) AS count
      FROM publication
      GROUP BY pubtype_id
      )

SELECT pubtype.pubname, temp_count.count
FROM temp_count JOIN pubtype
WHERE temp_count.pubtype_id = pubtype.pubtype_id;
```

Result:



	pubname	count
1	article	1885783
2	inproceeding	2248769
3	proceeding	38248
4	book	15418
5	incollection	47420
6	phdthesis	67475
7	www	2153410

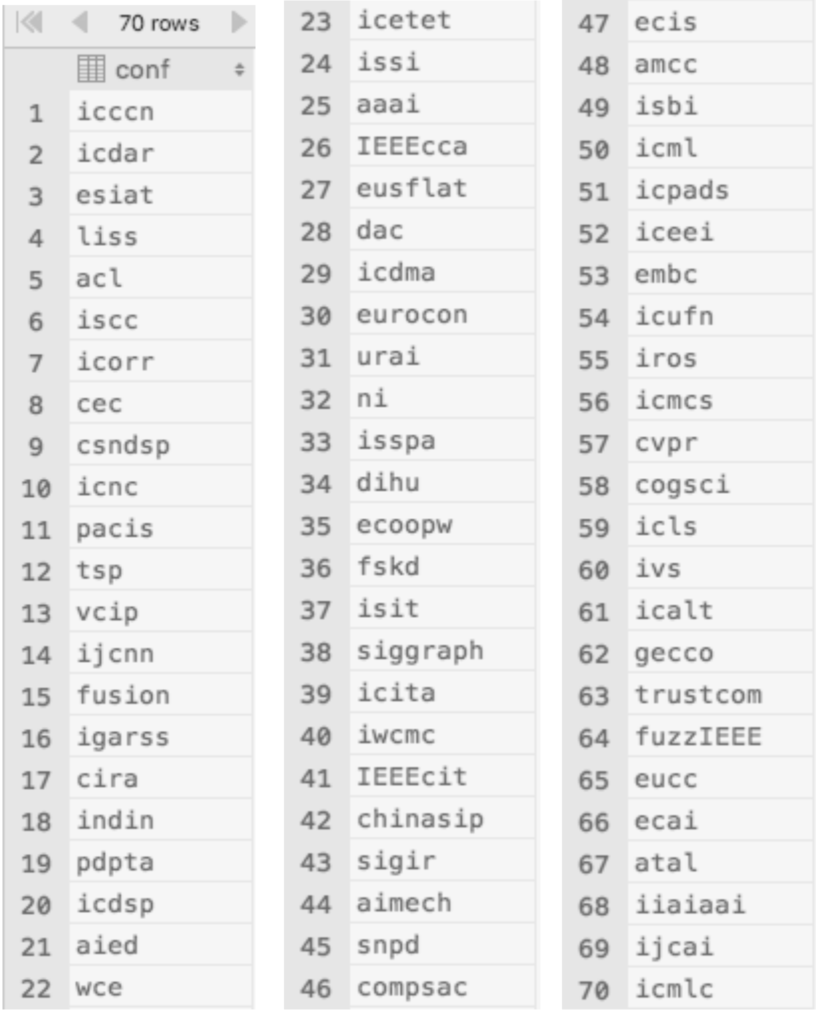
# (2) Find all the conferences that have ever published more than 200 papers in one year and are held in July.

Query:

```
WITH count_greater_than_200 AS (
       SELECT crossref_pubid AS pubid, COUNT(*) AS count
       FROM inproceedings
       GROUP BY crossref_pubid
       HAVING COUNT(*) > 200
      ),

conf_in_JULY AS (
       SELECT pubid, pubkey, title
       FROM publication
       WHERE title LIKE "%JULY%" and pubtype_id = 2
      ),

required_conf_dup AS (
      SELECT C1.pubid
      FROM count_greater_than_200 AS C1
      INNER JOIN conf_in_JULY AS C2 ON C1.pubid = C2.pubid
      )

SELECT DISTINCT P.conf
FROM proceedings AS P
INNER JOIN required_conf_dup AS R ON P.pubid = R.pubid;
```

(result table is on the following page)

Result:



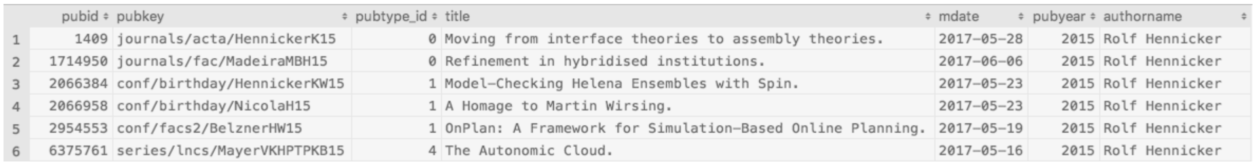
	conf
1	icccn
2	icdar
3	esiat
4	liss
5	acl
6	iscc
7	icorr
8	cec
9	csndsp
10	icnc
11	pacis
12	tsp
13	vcip
14	ijcnn
15	fusion
16	igarss
17	cira
18	indin
19	pdpta
20	icdsp
21	aied
22	wce
23	icetet
24	issi
25	aaai
26	IEEEcca
27	eusflat
28	dac
29	icdma
30	eurocon
31	urai
32	ni
33	isspa
34	dihu
35	ecoopw
36	fskd
37	isit
38	siggraph
39	icita
40	iwcmc
41	IEEEcit
42	chinasip
43	sigir
44	aimech
45	snpd
46	compsac
47	ecis
48	amcc
49	isbi
50	icml
51	icpads
52	iceei
53	embc
54	icufn
55	iros
56	icmcs
57	cvpr
58	cogsci
59	icls
60	ivs
61	icalt
62	gecco
63	trustcom
64	fuzzIEEE
65	eucc
66	ecai
67	atal
68	iiaiaai
69	ijcai
70	icmlc

# (3a) Find the publications of author = "X" (Rolf Hennicker) at year 2015.

Query:

```
WITH temp_publication AS(
      SELECT P.*, A.authorname
      FROM (pub_author AS PA
      INNER JOIN publication AS P ON PA.pubid = P.pubid)
      INNER JOIN author AS A ON PA.aid = A.aid
      )

SELECT *
FROM temp_publication
WHERE temp_publication.authorname = "Rolf Hennicker" and
temp_publication.pubyear = 2015;
```

Result:



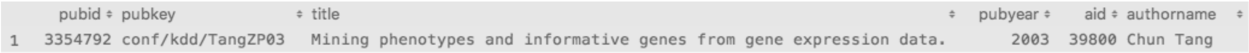
	pubid	pubkey	pubtype_id	title	mdate	pubyear	authorname
1	1409	journals/acta/HennickerK15	0	Moving from interface theories to assembly theories.	2017-05-28	2015	Rolf Hennicker
2	1714950	journals/fac/MadeiraMBH15	0	Refinement in hybridised institutions.	2017-06-06	2015	Rolf Hennicker
3	2066384	conf/birthday/HennickerKW15	1	Model-Checking Helena Ensembles with Spin.	2017-05-23	2015	Rolf Hennicker
4	2066958	conf/birthday/NicolaH15	1	A Homage to Martin Wirsing.	2017-05-23	2015	Rolf Hennicker
5	2954553	conf/facs2/BelznerHW15	1	OnPlan: A Framework for Simulation-Based Online Planning.	2017-05-19	2015	Rolf Hennicker
6	6375761	series/lncs/MayerVKHPTPKB15	4	The Autonomic Cloud.	2017-05-16	2015	Rolf Hennicker

# (3b) Find the publications of author = "X" (Chun Tang) at year "Y" (2003) at conference "Z" (KDD).

Query:

```
WITH temp_conference AS(
      SELECT *
      FROM publication
      WHERE pubkey LIKE ("conf/kdd/%") and pubyear = 2003
      )

SELECT *
FROM (temp_conference JOIN pub_author ON temp_conference.pubid =
pub_author.pubid)
JOIN author ON pub_author.aid = author.aid
WHERE author.authorname = "Chun Tang";
```

Result:



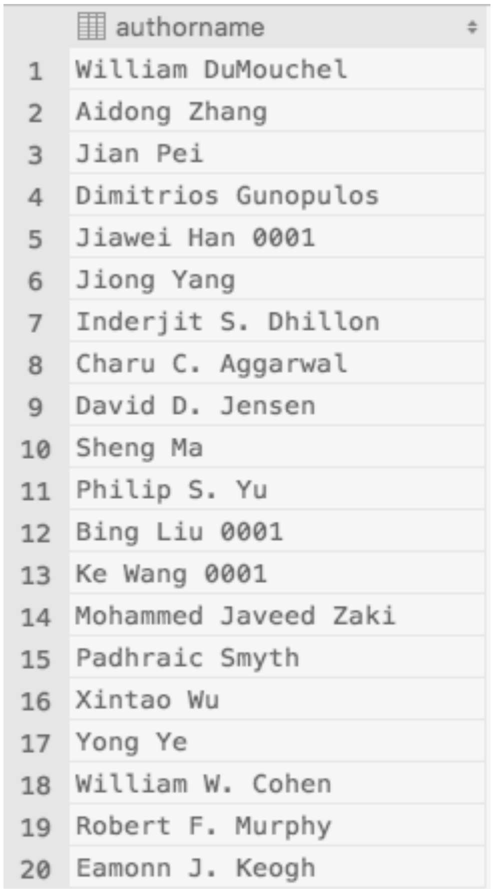
	pubid	pubkey	title	pubyear	aid	authorname
1	3354792	conf/kdd/TangZP03	Mining phenotypes and informative genes from gene expression data.	2003	39800	Chun Tang

# (3c). Find authors who published at least 2 papers at conference "Z" (KDD) at year "Y" (2003).

Query:

```
WITH temp_conference AS(
      SELECT *
      FROM publication
      WHERE pubkey LIKE ("conf/kdd/%") and pubyear = 2003
      )

SELECT author.authorname
FROM (temp_conference JOIN pub_author ON temp_conference.pubid =
pub_author.pubid)
JOIN author ON pub_author.aid = author.aid
GROUP BY author.authorname
HAVING count(*) > 1;
```

Result:



	authorname
1	William DuMouchel
2	Aidong Zhang
3	Jian Pei
4	Dimitrios Gunopulos
5	Jiawei Han 0001
6	Jiong Yang
7	Inderjit S. Dhillon
8	Charu C. Aggarwal
9	David D. Jensen
10	Sheng Ma
11	Philip S. Yu
12	Bing Liu 0001
13	Ke Wang 0001
14	Mohammed Javeed Zaki
15	Padhraic Smyth
16	Xintao Wu
17	Yong Ye
18	William W. Cohen
19	Robert F. Murphy
20	Eamonn J. Keogh

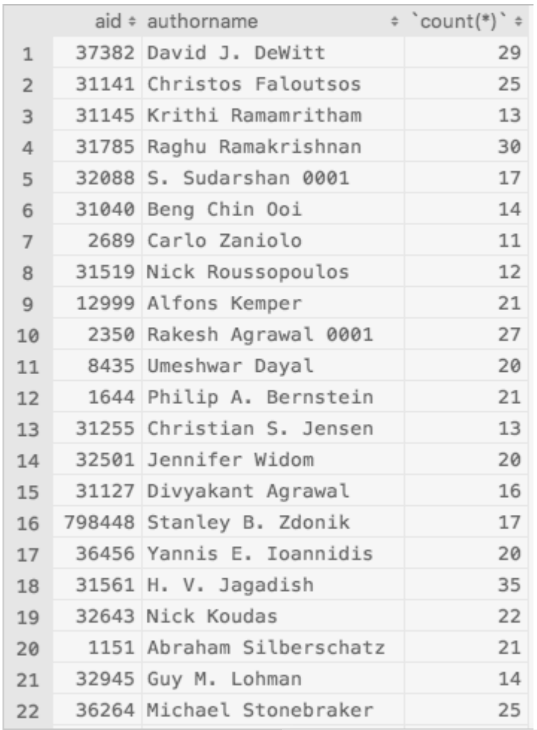
# (4a). All authors who published at least 10 PVLDB papers and published at least 10 SIGMOD papers.

Query:

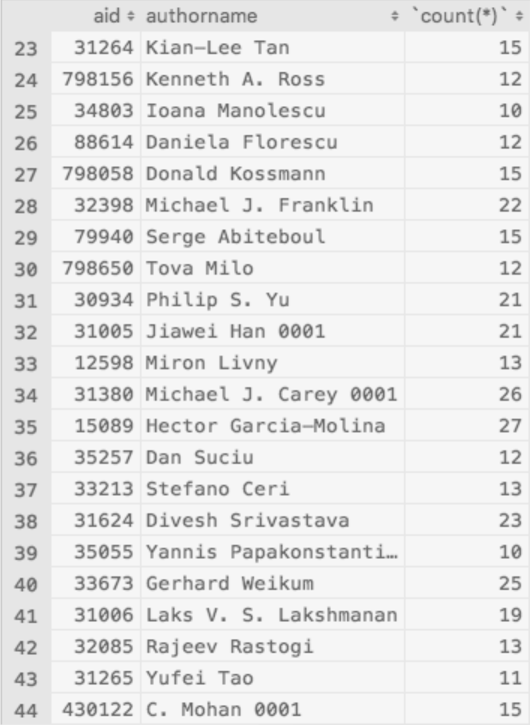
```
WITH temp_pub_author AS (
      SELECT pub_author.aid, pub_author.pubid, author.authorname
      FROM pub_author JOIN author ON pub_author.aid = author.aid
      ),

temp_pvldb AS (
      SELECT temp_pub_author.aid, temp_pub_author.authorname, count(*)
      FROM publication JOIN temp_pub_author ON publication.pubid =
      temp_pub_author.pubid
      WHERE pubkey LIKE ("conf/vldb/%")
      GROUP BY temp_pub_author.aid, temp_pub_author.authorname
      HAVING COUNT(*) > 9
      ),

 temp_sigmod AS (
       SELECT temp_pub_author.aid, temp_pub_author.authorname, count(*)
       FROM publication JOIN temp_pub_author ON publication.pubid =
      temp_pub_author.pubid
       WHERE pubkey LIKE ("conf/sigmod/%")
       GROUP BY temp_pub_author.aid, temp_pub_author.authorname
       HAVING COUNT(*) > 9
      )

SELECT *
FROM temp_pvldb
WHERE temp_pvldb.authorname IN
      (SELECT authorname
      FROM temp_sigmod);
```

(result table is on the following page)

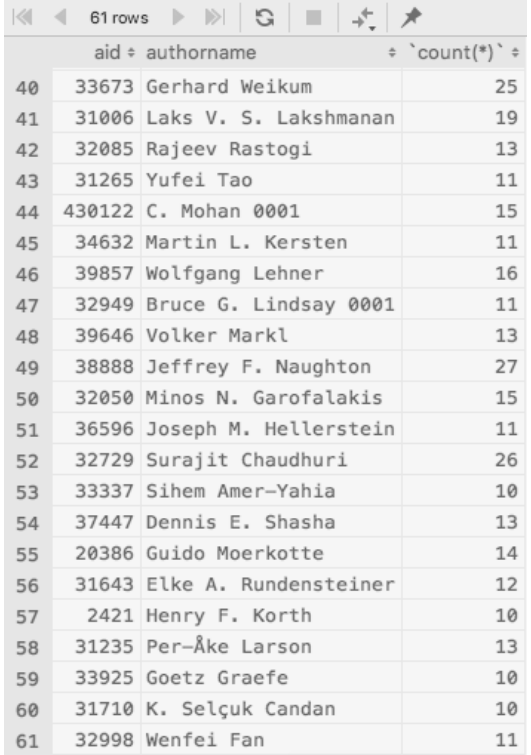
Result:



	aid	authorname	`count(*)`
1	37382	David J. DeWitt	29
2	31141	Christos Faloutsos	25
3	31145	Krithi Ramamritham	13
4	31785	Raghu Ramakrishnan	30
5	32088	S. Sudarshan 0001	17
6	31040	Beng Chin Ooi	14
7	2689	Carlo Zaniolo	11
8	31519	Nick Roussopoulos	12
9	12999	Alfons Kemper	21
10	2350	Rakesh Agrawal 0001	27
11	8435	Umeshwar Dayal	20
12	1644	Philip A. Bernstein	21
13	31255	Christian S. Jensen	13
14	32501	Jennifer Widom	20
15	31127	Divyakant Agrawal	16
16	798448	Stanley B. Zdonik	17
17	36456	Yannis E. Ioannidis	20
18	31561	H. V. Jagadish	35
19	32643	Nick Koudas	22
20	1151	Abraham Silberschatz	21
21	32945	Guy M. Lohman	14
22	36264	Michael Stonebraker	25



	aid	authorname	`count(*)`
23	31264	Kian-Lee Tan	15
24	798156	Kenneth A. Ross	12
25	34803	Ioana Manolescu	10
26	88614	Daniela Florescu	12
27	798058	Donald Kossmann	15
28	32398	Michael J. Franklin	22
29	79940	Serge Abiteboul	15
30	798650	Tova Milo	12
31	30934	Philip S. Yu	21
32	31005	Jiawei Han 0001	21
33	12598	Miron Livny	13
34	31380	Michael J. Carey 0001	26
35	15089	Hector Garcia-Molina	27
36	35257	Dan Suciu	12
37	33213	Stefano Ceri	13
38	31624	Divesh Srivastava	23
39	35055	Yannis Papakonstanti…	10
40	33673	Gerhard Weikum	25
41	31006	Laks V. S. Lakshmanan	19
42	32085	Rajeev Rastogi	13
43	31265	Yufei Tao	11
44	430122	C. Mohan 0001	15



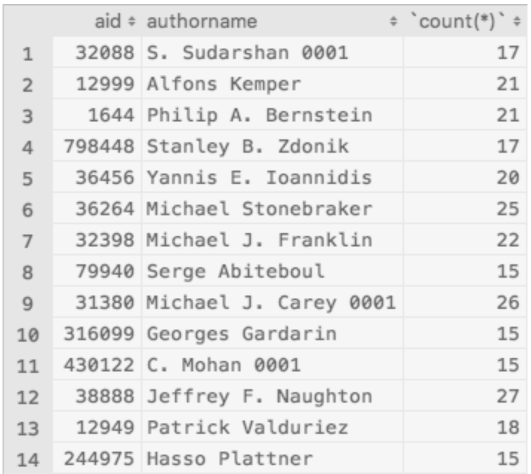
	aid	authorname	`count(*)`
40	33673	Gerhard Weikum	25
41	31006	Laks V. S. Lakshmanan	19
42	32085	Rajeev Rastogi	13
43	31265	Yufei Tao	11
44	430122	C. Mohan 0001	15
45	34632	Martin L. Kersten	11
46	39857	Wolfgang Lehner	16
47	32949	Bruce G. Lindsay 0001	11
48	39646	Volker Markl	13
49	38888	Jeffrey F. Naughton	27
50	32050	Minos N. Garofalakis	15
51	36596	Joseph M. Hellerstein	11
52	32729	Surajit Chaudhuri	26
53	33337	Sihem Amer-Yahia	10
54	37447	Dennis E. Shasha	13
55	20386	Guido Moerkotte	14
56	31643	Elke A. Rundensteiner	12
57	2421	Henry F. Korth	10
58	31235	Per-Åke Larson	13
59	33925	Goetz Graefe	10
60	31710	K. Selçuk Candan	10
61	32998	Wenfei Fan	11

# (4b). all authors who published at least 15 PVLDB papers but never published a KDD paper.

Query:

```
WITH temp_pub_author AS (
      SELECT pub_author.aid, pub_author.pubid, author.authorname
      FROM pub_author JOIN author ON pub_author.aid = author.aid
      ),

temp_pvldb AS (
      SELECT temp_pub_author.aid, temp_pub_author.authorname, count(*)
      FROM publication JOIN temp_pub_author ON publication.pubid =
      temp_pub_author.pubid
      WHERE pubkey LIKE ("conf/vldb/%")
      GROUP BY temp_pub_author.aid, temp_pub_author.authorname
      HAVING COUNT(*) > 14
      ),

 temp_kdd AS (
       SELECT DISTINCT temp_pub_author.aid, temp_pub_author.authorname
       FROM publication JOIN temp_pub_author ON publication.pubid =
      temp_pub_author.pubid
       WHERE pubkey LIKE ("conf/kdd/%")
      )

SELECT *
FROM temp_pvldb
WHERE temp_pvldb.authorname NOT IN (
      SELECT authorname
      FROM temp_kdd);
```

Result:



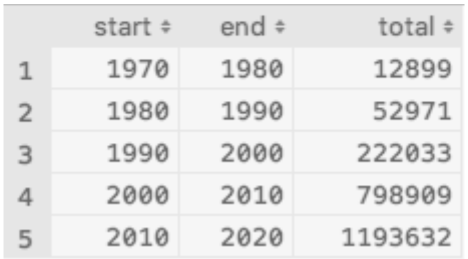
	aid	authorname	`count(*)`
1	32088	S. Sudarshan 0001	17
2	12999	Alfons Kemper	21
3	1644	Philip A. Bernstein	21
4	798448	Stanley B. Zdonik	17
5	36456	Yannis E. Ioannidis	20
6	36264	Michael Stonebraker	25
7	32398	Michael J. Franklin	22
8	79940	Serge Abiteboul	15
9	31380	Michael J. Carey 0001	26
10	316099	Georges Gardarin	15
11	430122	C. Mohan 0001	15
12	38888	Jeffrey F. Naughton	27
13	12949	Patrick Valduriez	18
14	244975	Hasso Plattner	15

# (5). For each 10 consecutive years starting from 1970, compute the total number of conference publications in DBLP in that 10 years.

Query:

```
WITH year_temp AS(
       SELECT DISTINCT P.pubyear, FLOOR((P.pubyear-1970)/10) AS decade_no
       FROM pub AS P
       WHERE P.pubyear >= 1970
       ORDER BY P.pubyear ASC
      ),

pcount AS(
       SELECT P.pubyear, count(P.pubid) AS psum
       FROM pub AS P
       WHERE P.pubkey LIKE ("conf/%") and P.pubyear >= 1970
       GROUP BY P.pubyear
      )

SELECT (decade_no * 10 + 1970) AS start, (decade_no * 10 + 1980) AS end,
SUM(pcount.psum) AS total
FROM pcount JOIN year_temp ON pcount.pubyear = year_temp.pubyear
GROUP BY decade_no
ORDER BY decade_no ASC;
```

Result:



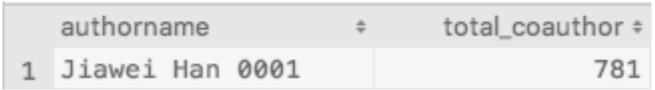
	start ÷	end ÷	total ÷
1	1970	1980	12899
2	1980	1990	52971
3	1990	2000	222033
4	2000	2010	798909
5	2010	2020	1193632

# (6). Find the most collaborative authors who published in a conference or journal whose name contains “data”.

Query:

```
WITH pub_title_contain_data AS (
      SELECT pubid, title
      FROM publication
      WHERE title LIKE "%DATA%"
      ),

pubid_author_count as (
      SELECT P1.pubid, COUNT(*) AS num_coauthor
      FROM pub_title_contain_data AS P1
       INNER JOIN pub_author AS P2
      ON P1.pubid = P2.pubid
      GROUP BY P1.pubid
      ),

aid_coauthor_count AS (
      SELECT P2.aid, P1.num_coauthor
      FROM pubid_author_count AS P1 INNER JOIN pub_author AS P2
      ON P1.pubid = P2.pubid
      ),

most_coauthor AS (
       SELECT aid, SUM(num_coauthor - 1 ) AS total_coauthor
       FROM aid_coauthor_count
       GROUP BY aid
       ORDER BY total_coauthor DESC
       LIMIT 1
      )

SELECT A2.authorname, A1.total_coauthor
FROM most_coauthor AS A1 INNER JOIN author AS A2 ON A1.aid = A2.aid;
```

Result:



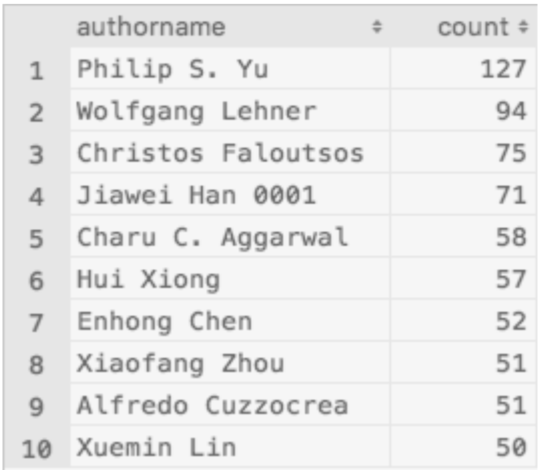
	authorname	total_coauthor
1	Jiawei Han 0001	781

# (7). Find the top 10 authors with the largest number of publications that are published in conferences and journals whose titles contain word "Data" in the last 5 years.

```
Query:
WITH conf_DATA AS (
        SELECT P.pubid
        FROM proceedings AS I
        LEFT JOIN publication AS P ON I.pubid = P.pubid
        WHERE P.title LIKE "%DATA%" and P.pubyear > 2013
        ),

inproceedings_in_conf AS (
        SELECT I.pubid AS pubid
        FROM conf_DATA AS C
        INNER JOIN inproceedings AS I ON C.pubid = I.crossref_pubid
        ),

aid_count AS (
        SELECT P.aid, COUNT(*) AS count
        FROM inproceedings_in_conf AS I
        INNER JOIN pub_author AS P ON I.pubid = P.pubid
        GROUP BY P.aid
        ORDER BY COUNT(*) DESC
        LIMIT 10
        )

SELECT A2.authorname, A1.count
FROM aid_count AS A1 JOIN author AS A2 on A1.aid = A2.aid;
```

(result table is on the following page)

Result:



	authorname	count
1	Philip S. Yu	127
2	Wolfgang Lehner	94
3	Christos Faloutsos	75
4	Jiawei Han 0001	71
5	Charu C. Aggarwal	58
6	Hui Xiong	57
7	Enhong Chen	52
8	Xiaofang Zhou	51
9	Alfredo Cuzzocrea	51
10	Xuemin Lin	50

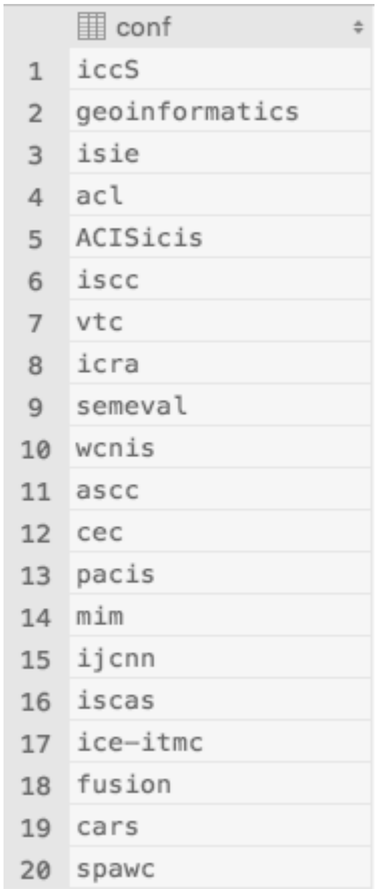
# (8). List the name of the conferences such that it has ever been held in June, and the corresponding proceedings contain more than 100 publications.

Query:

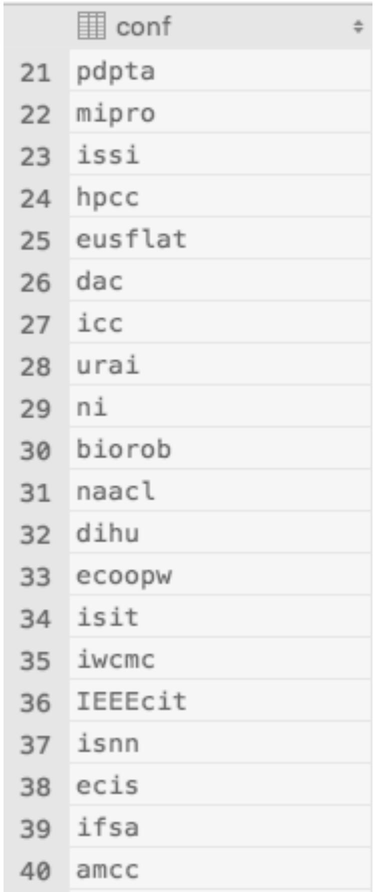
```
WITH count_greater_than_100 AS (
       SELECT crossref_pubid AS pubid, COUNT(*) AS count
       FROM inproceedings
       GROUP BY crossref_pubid
       HAVING COUNT(*) > 200
      ),

conf_in_june AS (
       SELECT pubid, pubkey, title
       FROM publication
       WHERE title LIKE "%june%" and pubtype_id = 2
      ),

required_conf_dup AS (
      SELECT C1.pubid
      FROM count_greater_than_100 AS C1
      INNER JOIN conf_in_june AS C2 ON C1.pubid = C2.pubid
      )

SELECT DISTINCT P.conf
FROM proceedings AS P INNER JOIN required_conf_dup AS R ON P.pubid = R.pubid;
```

(result table is on the following page)

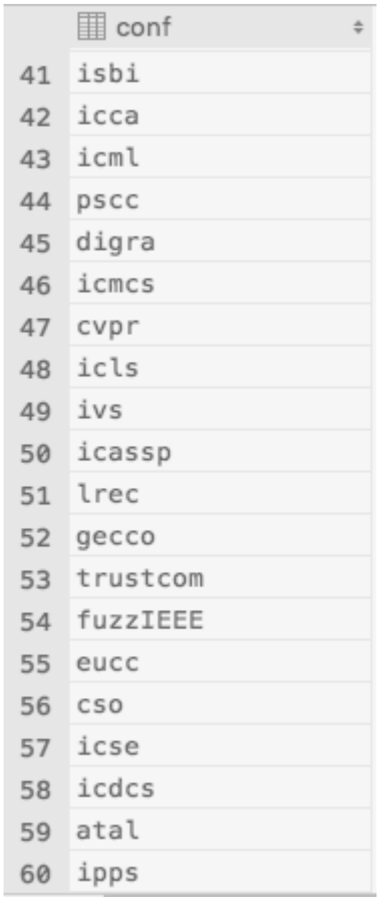
Result:



	conf
1	iccS
2	geoinformatics
3	isie
4	acl
5	ACISicis
6	iscc
7	vtc
8	icra
9	semeval
10	wcnis
11	ascc
12	cec
13	pacis
14	mim
15	ijcnn
16	iscas
17	ice-itmc
18	fusion
19	cars
20	spawc



	conf
21	pdpta
22	mipro
23	issi
24	hpcc
25	eusflat
26	dac
27	icc
28	urai
29	ni
30	biorob
31	naacl
32	dihu
33	ecoopw
34	isit
35	iwcmc
36	IEEEcit
37	isnn
38	ecis
39	ifsa
40	amcc



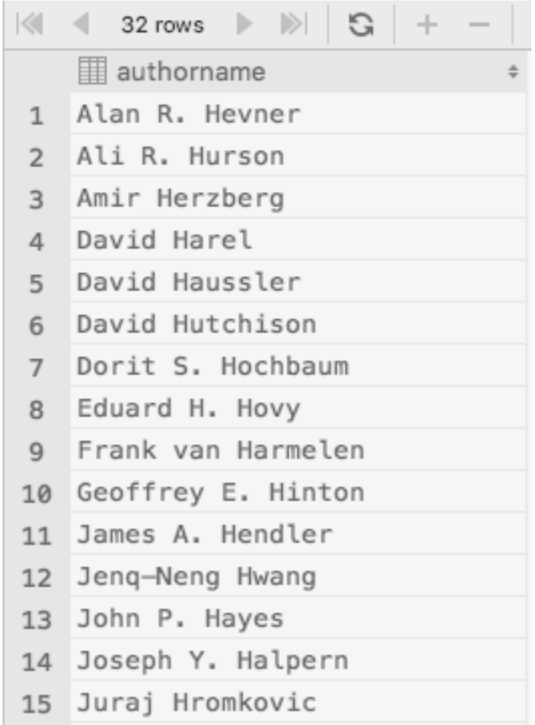
	conf
41	isbi
42	icca
43	icml
44	pscc
45	digra
46	icmcs
47	cvpr
48	icls
49	ivs
50	icassp
51	lrec
52	gecco
53	trustcom
54	fuzzIEEE
55	eucc
56	cso
57	icse
58	icdcs
59	atal
60	ipps

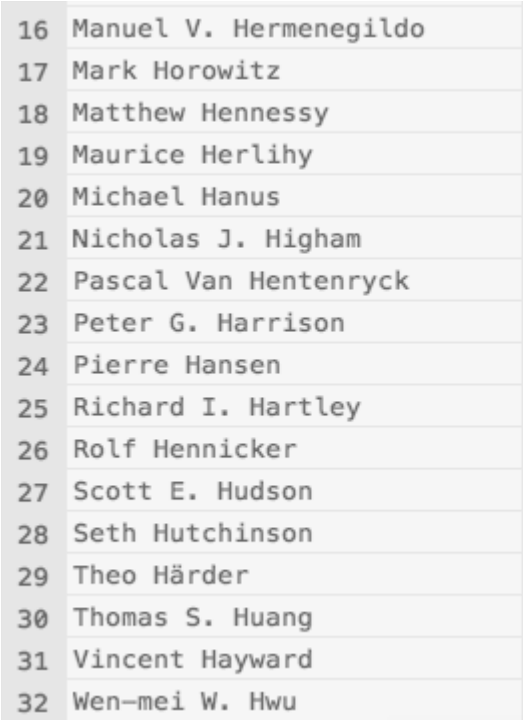
# (9a). Find authors who have published at least 1 paper every year in the last 30 years, and whose family name start with 'H'.

Query:

```
WITH pub_last_30_years AS (
	SELECT *
	FROM publication
	WHERE pubyear BETWEEN 1988 AND 2018
),

author_fname_H AS (
	SELECT *
	FROM author
	WHERE SUBSTRING_INDEX(author.authorname , ' ', -1) LIKE 'H%'
),

author_H_pub AS (
	SELECT A.authorname, P.pubid
	FROM author_fname_H AS A JOIN pub_author AS P ON A.aid = P.aid
),
author_H_pub_30 AS (
      SELECT DISTINCT A.authorname, P.pubyear
      FROM author_H_pub AS A
      INNER JOIN pub_last_30_years AS P ON A.pubid = P.pubid
)

SELECT A.authorname
FROM author_H_pub_30 AS A
GROUP BY A.authorname
HAVING COUNT(*) >30;
```

Result:





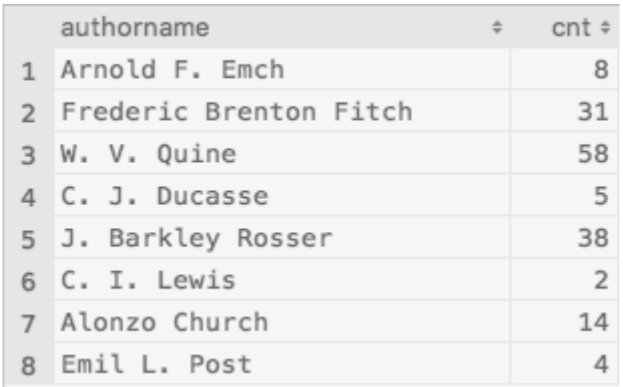
	authorname
1	Alan R. Hevner
2	Ali R. Hurson
3	Amir Herzberg
4	David Harel
5	David Haussler
6	David Hutchison
7	Dorit S. Hochbaum
8	Eduard H. Hovy
9	Frank van Harmelen
10	Geoffrey E. Hinton
11	James A. Hendler
12	Jenq-Neng Hwang
13	John P. Hayes
14	Joseph Y. Halpern
15	Juraj Hromkovic
16	Manuel V. Hermenegildo
17	Mark Horowitz
18	Matthew Hennessy
19	Maurice Herlihy
20	Michael Hanus
21	Nicholas J. Higham
22	Pascal Van Hentenryck
23	Peter G. Harrison
24	Pierre Hansen
25	Richard I. Hartley
26	Rolf Hennicker
27	Scott E. Hudson
28	Seth Hutchinson
29	Theo Härder
30	Thomas S. Huang
31	Vincent Hayward
32	Wen-mei W. Hwu

# (9b) Find the names and number of publications for authors who have the earliest publication record in DBLP.

Query:

```
WITH earliest_pub as (
      SELECT author.aid AS aid, author.authorname AS authorname
      FROM publication, pub_author, author
      WHERE publication.pubyear = (
            SELECT MIN(pubyear)
            FROM publication)
      AND publication.pubid = pub_author.pubid
      AND pub_author.aid = author.aid
)

SELECT authorname, COUNT(*) AS cnt
FROM earliest_pub as t1, pub_author
WHERE t1.aid = pub_author.aid
GROUP BY t1.aid,authorname;
```

Result:



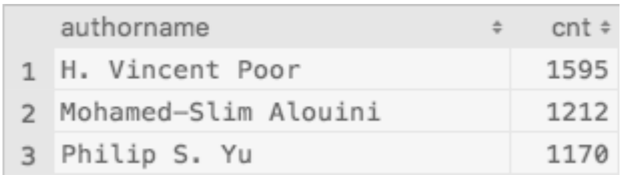
	authorname	cnt
1	Arnold F. Emch	8
2	Frederic Brenton Fitch	31
3	W. V. Quine	58
4	C. J. Ducasse	5
5	J. Barkley Rosser	38
6	C. I. Lewis	2
7	Alonzo Church	14
8	Emil L. Post	4

# (10). Find the top 3 authors with largest number of publications in DBLP.

Query:

```
WITH author_top3 AS (
      SELECT aid, COUNT(*) AS cnt
      FROM pub_author
      GROUP BY aid
      ORDER BY COUNT(*) DESC
      LIMIT 3
      )

SELECT A.authorname, author_top3.cnt
FROM author_top3 INNER JOIN author AS A ON author_top3.aid = A.aid;
```

Result:



	authorname	cnt
1	H. Vincent Poor	1595
2	Mohamed-Slim Alouini	1212
3	Philip S. Yu	1170

2.2 Figures and analysis

Table 1 records the different execution time when we ran the same queries on full DBLP database, half of DBLP publication records and quarter of data respectively.

Table 1 shows that execution time is roughly proportional to data size. When data size is halved, the average execution time is 57.77%. When data size is a quarter of the original size, the execution time became 20.15% of the original execution time. This is because the larger the data size, the more number of blocks needed to store the data, leading to a more significant disk I/O time.

We also notice that the decrease in execution time is not linear (100% -> 57.77% -> 20.15%). When the data size becomes 25%, the average drop in execution time is more than 75%. This may be because the original data size cannot fit into memory and hence two pass algorithm is required whereas smaller data size might be able to fit into memory and only single pass algorithm is required. Hence, the number of disk I/O can be decreased significantly. (e.g 3a, 3b)

Furthermore, the improvement on execution time varies for different queries. For example, when data size is halved, the execution time for query 2 is 80.6% of the original time. On the other hand, the execution time for query 6 drops to 43.5%.

Query	Full / ms	Half / ms	Quarter / ms	Half vs Full	Quarter vs Full
1	2625	1287	569	49.03%	21.68%
2	1299	1047	530	80.60%	40.80%
3a	10608	8413	112	79.31%	1.06%
3b	2374	1114	53	46.93%	2.23%
3c	1860	883	780	47.47%	41.94%
4a	3396	1630	1067	48.00%	31.42%
4b	3390	1997	737	58.91%	21.74%
5	20437	10995	4584	53.80%	22.43%
6	8016	3486	1826	43.49%	22.78%
7	6983	3489	650	49.96%	9.31%
8	1388	524	210	37.75%	15.13%
9a	25892	15632	7957	60.37%	30.73%

9b	20390	20156	961	98.85%	4.71%
10	10155	5521	1637	54.37%	16.12%
			AVERAGE	**57.77%**	**20.15%**

Table 1. Execution time for different data sizes

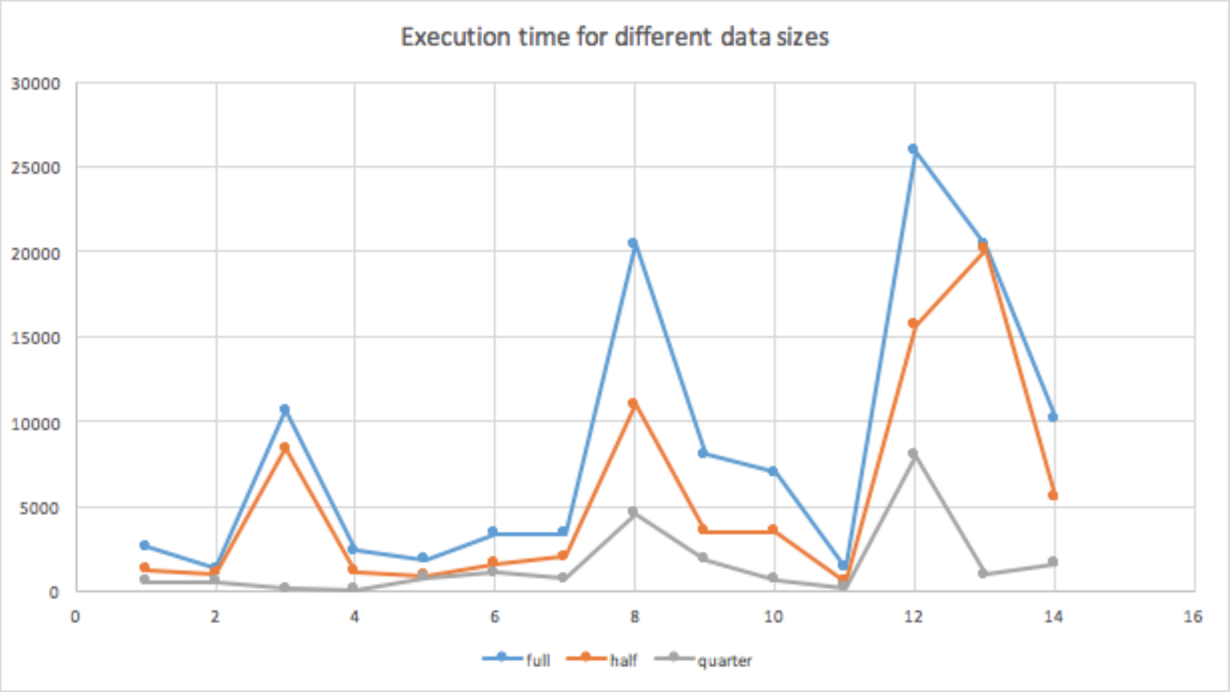


Figure 5. Execution time for different data sizes

3. Effect of Index

3.1 Create index command

```
CREATE UNIQUE INDEX author_authorname_uindex
ON author (authorname);

CREATE UNIQUE INDEX inproceedings_crossref_pubid
ON inproceedings (crossref_pubid);

CREATE UNIQUE INDEX pub_author_aid
ON pub_author (aid);

CREATE INDEX publication_pubtype_id
ON publication (pubtype_id);

CREATE INDEX publication_pubyear
ON publication (pubyear);

CREATE INDEX publication_pubkey
ON publication (pubkey);
```

3.2 Analysis on Index

Index is a file that accelerates the retrieval of records from a data file. Table 2 shows the execution time before and after adding the indices.

We noticed that the indices created in section 3.1 have different effects on different queries. For example, the decrease in execution time for query 6 is 18.73% where as that for query 9b is 99.68%. This is because query 6 has a subquery "`WHERE title LIKE "%DATA%"`" and indices does not help in LIKE operator. On the other hand, query 9 has RANGE query on `pubyear`. Hence, an index on `pubyear` will improve execution time significantly for query 9.

Since `pubid` is unique for every publication and *aid* is unique for every author. Creating index for these attributes will significantly speed up processing queries which look for certain publication or author. As for the non-unique attributes, for example, *authorname* and *pubkey*, adding indices still plays an important role to upgrade the performance. Even though the indices cannot tell the placement of certain record, they provide information about the location of record and shorten the scanning time.

We also created an index on inproceeding(crossref_pubid). This helps the performance when we want to group the inproceedings by the its proceeding pubid. This index is very useful for query 2, 7, 8 where we need to find all the inproceedings in a particular proceeding(conference).

Query	Before / ms	After / ms	Difference / ms	Change
1	2625	1423	1202	45.79%
2	1299	790	509	39.18%
3a	10608	129	10479	98.78%
3b	2374	265	2109	88.84%
3c	1860	663	1197	64.35%
4a	3396	168	3228	95.05%
4b	3390	119	3271	96.49%
5	20437	7912	12525	61.29%
6	8016	6515	1501	18.73%
7	6983	559	6424	91.99%
8	1388	690	698	50.29%
9a	25892	7879	18013	69.57%
9b	20390	66	20324	99.68%
10	10155	3575	6580	64.80%

Table 2. Execution time before and after adding the index

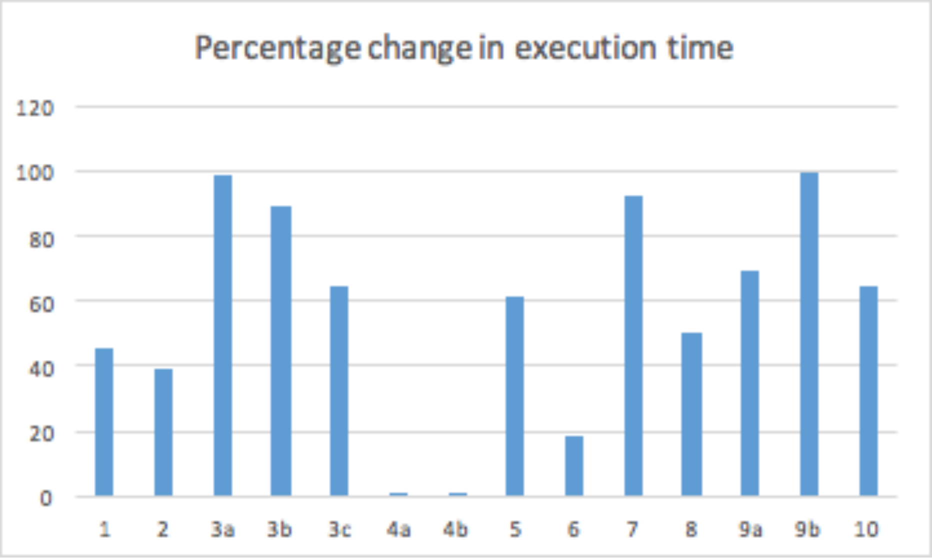


Figure 5. Percentage change in execution time

4. Effect of Cache

4.1 Analysis on effect of cache

In MySQL InnoDB storage engine, environment variable `innodb_buffer_pool_size` is used for configuring the size of buffer pool which is used for caching data and indexes in main memory.

In the experiments, we have compared the execution time for all 14 given queries with `innodb_buffer_pool_size` being 5GB, 512MB, 128MB, and 64MB, and the result has been shown in *Table 3* and *Figure 6.*

Query	**5G / ms**	**512M / ms**	**128M / ms**	**64M / ms**
1	926	940	1006	1020
2	581	713	757	719
3a	18	21	24	27
3b	25	24	25	17
3c	30	45	47	66
4a	79	140	124	115
4b	81	145	140	140
5	6204	7748	7576	6749
6	5952	6815	6975	6920
7	391	590	637	583
8	280	645	659	721
9a	21234	62264	98058	123378
9b	18	20	23	20
10	3161	3385	3674	3407

*Table 3. Execution time for different innodb_buffer_pool_size setting*s

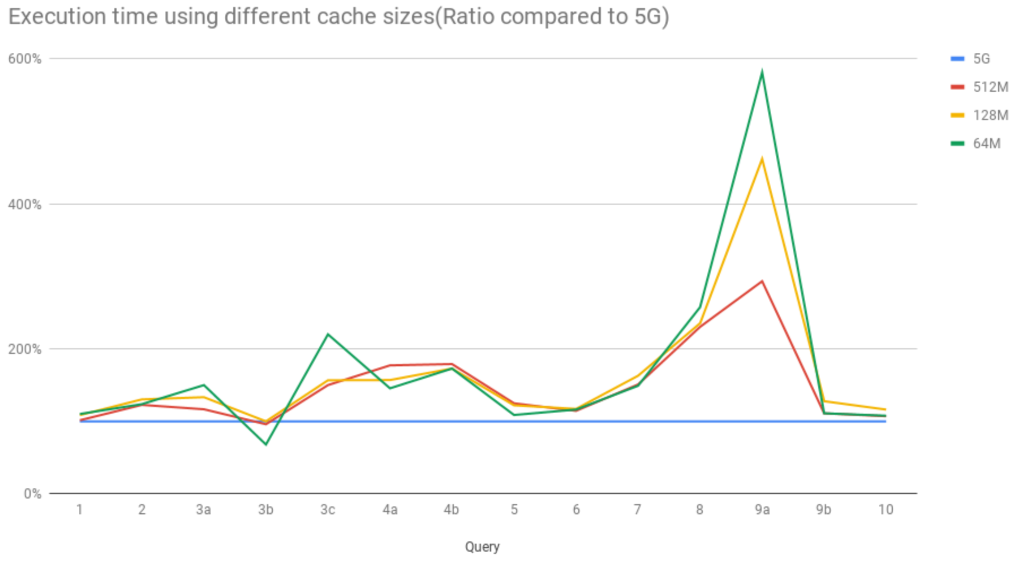


Figure 6. Percentage change in execution time

We could see that, the larger `innodb_buffer_pool_size` is, in general, the better the performance is. The reason being that is with larger pool size, more data or indexes could be cached in main memory, which leads to less disk I/O. Especially, taking query *9a* as an example, when the query is very complicated, the advantages of larger `innodb_buffer_pool_size` become more significant.

5. Appendix

5.1 Java code SAX Parser

```java
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
* This SAX parser is able to parse dblp.xml into three files: publictaion.csv, author.csv, and
pub_author.csv.
* <ol>
* <li>publication.csv</li>
* This file stores all 7 subclasses of publication. Each row is a publication (pub_id = row
number)
* The columns are all the possible fields for every publication.
* If a publication does not have a certain field, that column is skipped.
* <li>author.csv</li>
* This file contains a list of authors. Each row has two columns: author_id, and author_name
* <li>pub_author.csv</li>
* This file has two columns: pub_id and author_name.
* It stores a many-to-many relationship between publications and authors.
* </ol>
*/
public class UserHandler extends DefaultHandler {
   private List<String> mPubElements;
   private List<String> mFieldElements;
   private HashMap<String, String> mValues;
   private int mPubId = 0; //unique pubid for each publication
   private int mAuthorId = 0; //unique author_id for each author
   private String mDelimeter = "\t";
   private BufferedWriter mWritePubAuthor;
   private BufferedWriter mWriterPub;
   private BufferedWriter mWriterAuthor;
   private StringBuilder mSb;

   public UserHandler() {
       mSb = new StringBuilder();

       //list of subclasses of publications
       mPubElements = new ArrayList<>();
       mPubElements.add("article");
       mPubElements.add("inproceedings");
       mPubElements.add("proceedings");
```

```
        mPubElements.add("book");
        mPubElements.add("incollection");
        mPubElements.add("phdthesis");
        mPubElements.add("masterthesis");
        mPubElements.add("www");

        //list of all possible fields for a publication
        mFieldElements = new ArrayList<>();
        mFieldElements.add("pubid");
        mFieldElements.add("pubtype");
        mFieldElements.add("mdate");
        mFieldElements.add("pubkey");
        mFieldElements.add("booktitle");
        mFieldElements.add("address");
        mFieldElements.add("author");
        mFieldElements.add("cdrom");
        mFieldElements.add("chapter");
        mFieldElements.add("city");
        mFieldElements.add("crossref");
        mFieldElements.add("cite");
        mFieldElements.add("editor");
        mFieldElements.add("ee");
        mFieldElements.add("isbn");
        mFieldElements.add("journal");
        mFieldElements.add("month");
        mFieldElements.add("note");
        mFieldElements.add("number");
        mFieldElements.add("pages");
        mFieldElements.add("publisher");
        mFieldElements.add("publnr");
        mFieldElements.add("school");
        mFieldElements.add("series");
        mFieldElements.add("title");
        mFieldElements.add("url");
        mFieldElements.add("volume");
        mFieldElements.add("year");

        mValues = new HashMap<>();
    }
```

```
/**
   * Handles differently a publication element from a field element.
   * If it is a publication element, all attributes are retrieved.
   * If it is a field element, there is no attribute.
   * @param uri
   * @param localName
   * @param qName
   * @param attributes
   * @throws SAXException
   */
  @Override
  public void startElement(String uri,
                           String localName,
                           String qName,
                           Attributes attributes) throws SAXException {
      qName = qName.toLowerCase();
      if (mPubElements.contains(qName)) {
          String mdate = attributes.getValue("mdate");
          String key = attributes.getValue("key");
          mValues.clear();
          mPubId++;
          mValues.put("pubid", String.valueOf(mPubId));
          mValues.put("pubtype", qName);
          mValues.put("mdate", mdate);
          mValues.put("pubkey", key);

      } else if (mFieldElements.contains(qName)) {
          if (qName.equalsIgnoreCase("author"))
              mAuthorId++;
      }

  }
```

```
/**
     * Handles differently a publication element from a field element.
     * If it is the end of a publication element: insert one entry into "publication.csv"
     * If it is the end of a field element: insert into mValues which stores key-value pairs of
all fields.
     * A special case is 'author' element.
     * If it is the end element of 'author' element, we have to insert one entry into
'author.csv' and one entry into 'pub_author.csv'
     * @param uri
     * @param localName
     * @param qName
     * @throws SAXException
     */
   @Override
   public void endElement(String uri,
                             String localName, String qName) throws SAXException {

       qName = qName.toLowerCase();
       if (mPubElements.contains(qName)) {
           writeNewPub(mValues);
       } else if (mFieldElements.contains(qName)) {
           mValues.put(qName, new String(mSb));
           mSb.setLength(0);
           if (qName.equalsIgnoreCase("author")) {
                writeNewAuthor(mValues.get("author"));
                writeNewPubAuthor(mValues.get("author"));
           }

       }

   }
```

```
 /**
     * Read content between a start element and a end element.
     * Store the characters in a string builder.
     * Handles special characters such as "\"
     * @param ch
     * @param start
     * @param length
     * @throws SAXException
     */
    @Override
    public void characters(char ch[], int start, int length) throws SAXException {
        String str = new String(ch, start, length);
        mSb.append(str.replaceAll("\\\\", ""));
    }

    /**
     * Insert one entry into 'publication.csv'
     * @param values
     */
    private void writeNewPub(HashMap<String, String> values) {

        try {
            mWriterPub = new BufferedWriter(new FileWriter("publication.csv", true));
            for (String field : mFieldElements) {
                if (values.containsKey(field)) {
                    if (mFieldElements.indexOf(field) != 0)
                        mWriterPub.write(mDelimeter);
                    mWriterPub.write(values.get(field));
                } else {
                    mWriterPub.write(mDelimeter);
                }
            }
            mWriterPub.write("\n");
            mWriterPub.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

```
/**
   * Insert one entry into 'author.csv'
   * @param author
   */
  private void writeNewAuthor(String author) {
      try {
          mWriterAuthor = new BufferedWriter(new FileWriter("author.csv", true));
          mWriterAuthor.write(mAuthorId + mDelimeter + author);
          mWriterAuthor.write("\n");
          mWriterAuthor.close();
      } catch (IOException e) {
          e.printStackTrace();
      }
  }
```

```
 /**
    * Insert one entry into 'pub_author.csv'
    * @param author
    */
   private void writeNewPubAuthor(String author) {
       try {
           mWritePubAuthor = new BufferedWriter(new FileWriter("pub_author.csv", true));
           mWritePubAuthor.write(mPubId + mDelimeter + author);
           mWritePubAuthor.write("\n");
           mWritePubAuthor.close();
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```