Data Modeling and Databases: DBMS Project

Qiang Qu

Alexey Chernyshov, Vladislav Kravchenko, Murad Magomedov

Contents

1 Phase 1. Design and Implement Relational Model. 3
1.1 Introducing . . . 3
1.2 ER-Model . . . 3
1.3 Functionality . . . 4

2 Phase 2. Web-based User Interface. 5
2.1 Introducing . . . 5
2.2 Setup Instructions . . . 5
2.3 Functionality . . . 5
2.4 Site map . . . 6
2.5 Snapshots of the Interface . . . 7

3 Phase 3. DBMS 9
3.1 Introducing . . . 9
3.2 Setup Instructions . . . 9
3.3 Functionality . . . 9
3.4 Disk Page Organisation . . . 11
3.5 Application architecture . . . 12
3.6 Insights . . . 13

1 Phase 1. Design and Implement Relational Model.

1.1 Introducing

There are many repositories containing research article information, but not all of them provide interfaces to access this information. Thus we decided to use DBLP. One of its interfaces is (https://aminer.org/billboard/citation), which provides such additional information as cites and abstracts for articles. This information further will be used in ranking methods.

1.2 ER-Model

After studying the documentation we agreed on next entities for our ER-model:

- Article
- Author
- Keyword

We will extract Keywords from the abstracts in order to search for related articles. We designed this ER-Model:

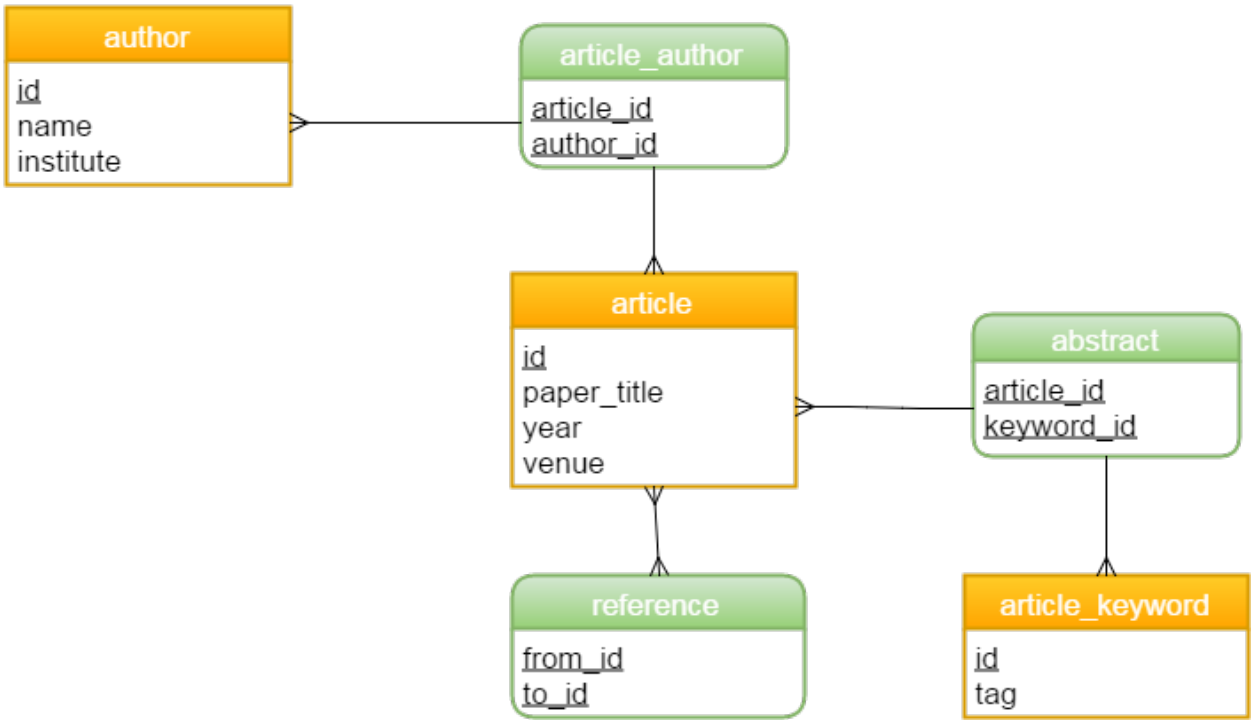

Figure 1: ER-Model diagram.

Our ER-Model was transformed into Relations:

- article: {[id: INTEGER, paper_title: STRING, venue: STRING, year: INTEGER]}
- author: {[id: INTEGER, name: STRING, institute: STRING]}
- keyword: {[id: INTEGER, tag: STRING]}
- article_author: {[id_article: INTEGER, id_author: INTEGER]}
- article_keword: {[id_article: INTEGER, id_keyword: INTEGER]}
- reference: {[from_id: INTEGER, to_id: INTEGER]}

1.3 Functionality

We designed the relations in an existing database management system (DBMS) by creating physical tables and their relationships. For our DBMS we have chosen the PostgreSQL. We imported 1.632.442 publications that we parsed using a Python script from an existing publication repository (DBLP). In addition, we created the SQL-query files that perform such operations:

- Select
- Update
- Delete
- Insert

You can search the publication based on author name, publication year, venue(conference/journal name), title, keyword, institution.
To search for the related articles you can use Keywords and References (to
from).
You can sort the publications based on such ranking methods:

- based on the number of times the Authors were cited. (In case of a new publication with no references to, but with popular authors)
- based on the number of times the Article was cited. (Popular publications)

SQL-query files can be found in attachment.

2 Phase 2. Web-based User Interface.

2.1 Introducing

According to Project Phase2 requirements we had to develop a web-based graphical user interface on top of physical database that we had created at Phase 1.
After considering different high-level programming languages we decided to use Python because of its clean, straightforward syntax and great support for building web apps. And as it was suggested by instructors on piazza.com we used PostgreSQL for the database management system and Tornado web framework for the web-based GUI.

2.2 Setup Instructions

1. Install Python: https://www.python.org/download/releases/3.4.0/
2. Install PostgreSQL: http://www.postgresql.org/download/
3. Import dump: execute in cmd in folder:

```
".../PostgreSQL/9.4/bin/ psql -U 'input your database username here' -f

'path to dump file' 'name of your database'
```

4. Execute query to add auth table:

```
...\src\sql\auth\create_auth.sql
```

5. Configure Settings.py - set your username and password
6. Install Tornado: execute in cmd in folder

```
C:\Python34\Lib\site-packages\easy_install.py tornado
```

7. Run in cmd webserver.py 8. Go to http://localhost:8000

2.3 Functionality

According to the Phase 2 requirements only authorized/registered users are allowed to use the interface that we have created. The table with usernames and passwords can be stored in another database but in our case we decided to store it at the main database, even it is not logically connected with it. Authorized/registered users can carry out all the specified functionality, including search, insert (add), update () and delete. All the search results can be ranged in ascending or descending order.

2.4 Site map

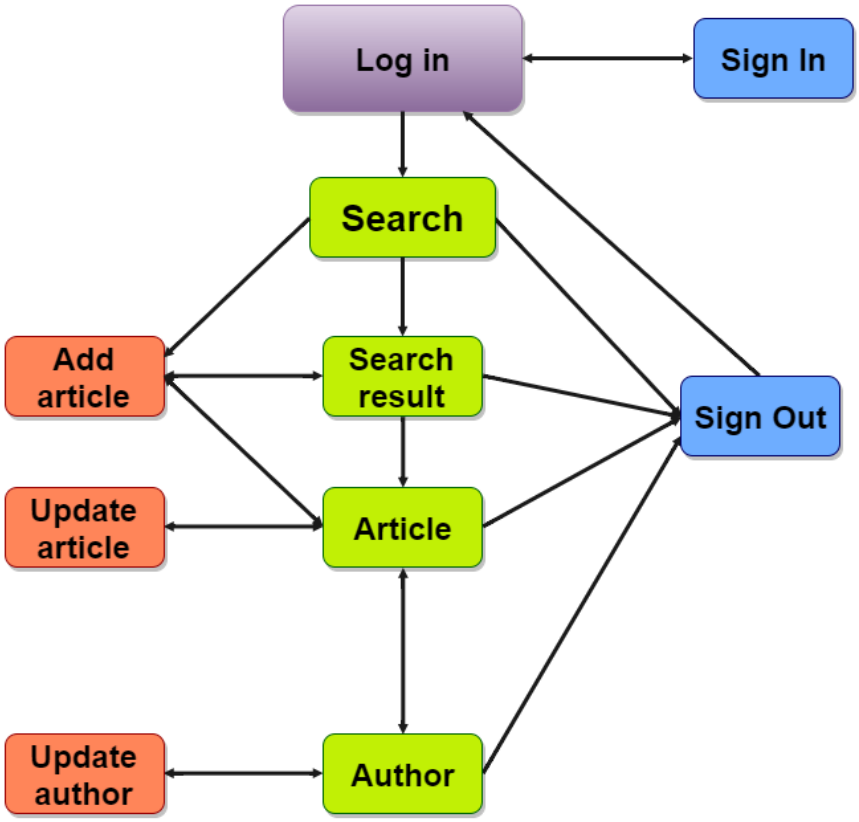

Figure 2: Site Map

2.5 Snapshots of the Interface

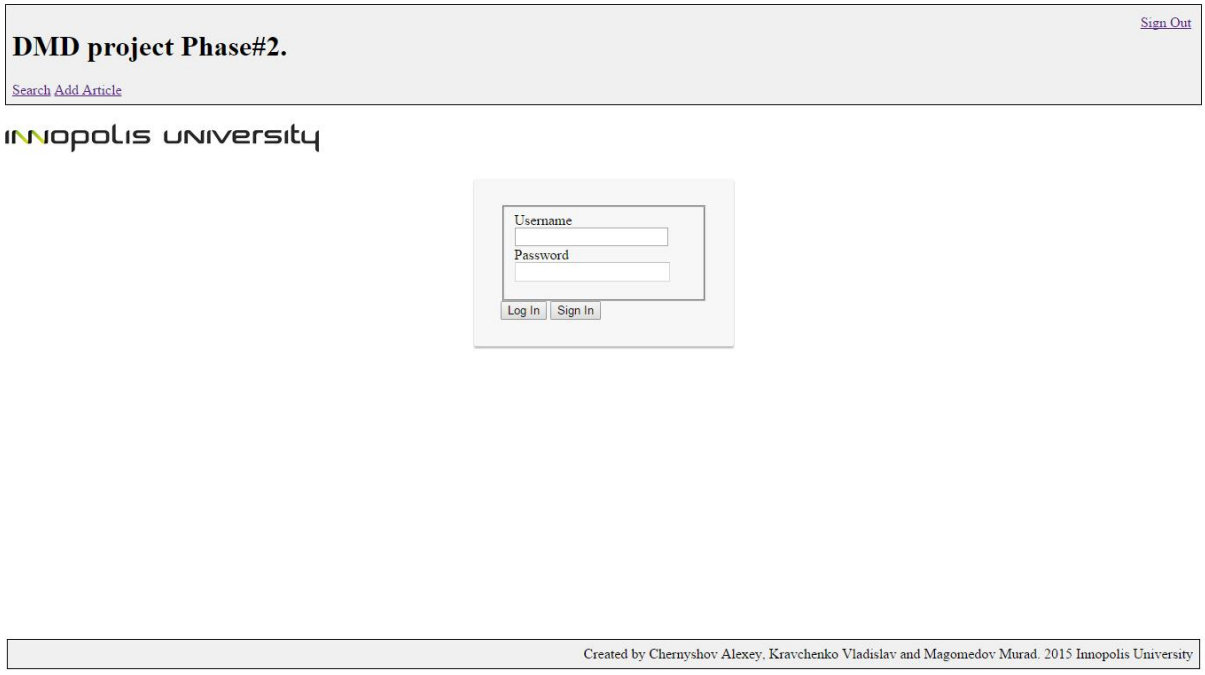

Figure 3: Log in or Sign in

Figure 4: Search

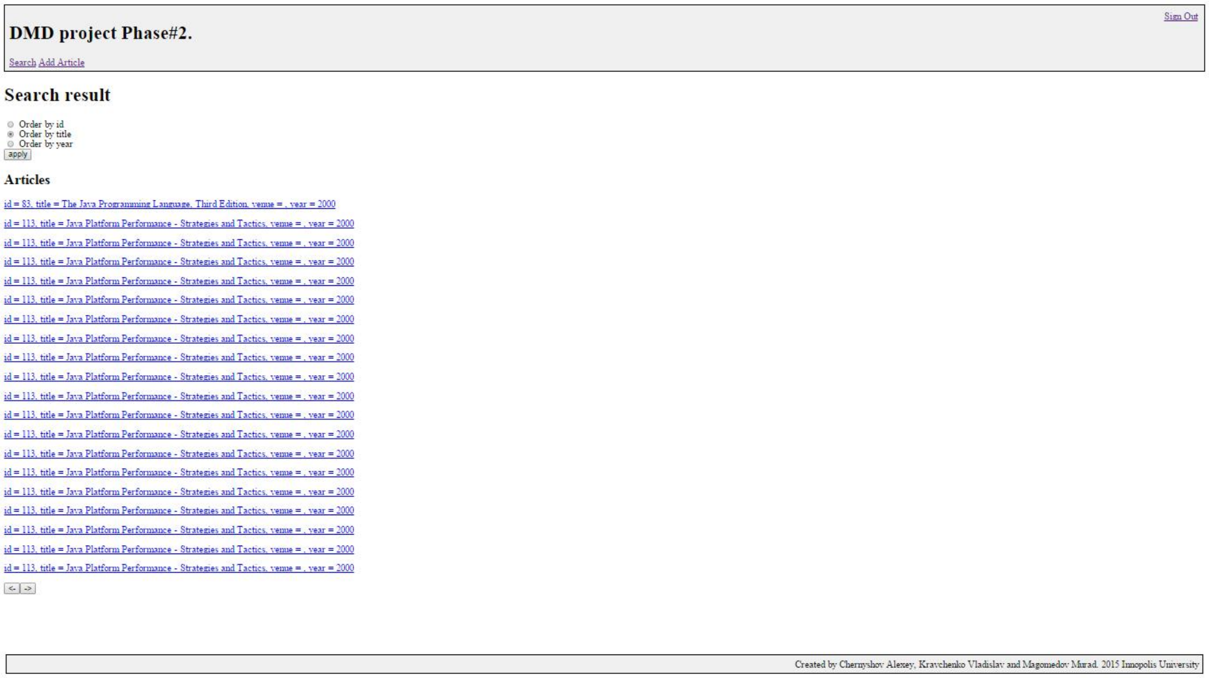

Figure 5: Search result

Figure 6: Article

3 Phase 3. DBMS

3.1 Introducing

According to the Project Phase 3 requirements we had to replace the relational database with our own implementation of a database. We have done it, including the implementation of query operators using python programming language.

3.2 Setup Instructions

1. You need to have everything installed from the Phase 2. 2. Run src/dbms/psqlloader.py to create a datafile for your dbms and to import data from Postgres. 3. Configure src/webapp/Settings.py (login, password, database file name) 4. Run web application src/webapp/webserver.py (SQL replaced application) All SQL was moved to src/webappsql. 5. Go to the http://localhost:8000

3.3 Functionality

At the Phase 3 all the SQL code was replaced with calls to the query operators. So now our database provides methods to query data, to insert new records, update existing records and delete them.

For example, SQL code of Select:

```
cur.execute("""SELECT id, name, institute FROM author WHERE id=%s;""", (id, ))
```

Replaced with:

```
qr_author = qp.getFromTable('author', )
qr_res = qp.getFromTable('author', ('id', id))
qr_res = qr_res.project('id', 'name', 'institute')
qr_res = qr_res.sort('id')
```

Delete:

```
cur.execute("""DELETE FROM article_author WHERE author_id=%s""", (id, ))
```

Replaced with:

```
qp.deleteFromTable('article_author', ('author_id', author_id))
```

Update:

```
cur.execute("""UPDATE author SET id=%s, name=%s, institute=%s
            WHERE id=%s""", (id, name, institute, id))
```

Replaced with:

```
qp.deleteFromTable('author', ('id', id))
qp.addToTable('author', ('id', id), ('name', name), ('institute', institute))
```

Insert:

```
cur.execute("""INSERT INTO reference (from_id, to_id)
            VALUES (%s, %s)""", (id, article_id))
```

Replaced with:

```
qp.addToTable('reference', ('from_id', id), ('to_id', article_id))
```

Group by:

```
qres.groupBy('id', 'name')
```

Our database also answers queries that join two tables, for example join of tables article_author and article:

Join:

```
qr_article_author = qp.getFromTable('article_author', ('author_id', author_id))
qr_article = qp.getFromTable('article')
qr_res = qr_article_author.join(qr_article, 'article_id', 'id')
qr_res = qr_res.project('article_id', 'paper_title')
qr_res = qr_res.sort('article_id')
```

One of the requirements of the Phase 3 was to use the iterator model to implement query operators, so that operators should pull tuples from underlying operators using next() calls.
In our case, class qres (query result) is iterable. It is possible to use qres.next or next(qres) to iterate through the results. In our DBMS it is possible to index on Primary keys with an option to add index on other attributes.
And according to Project limitations we are using a single file for the entire DBMS. The Database file is created using psqlloader.py that imports data from Postgres.

3.4 Disk Page Organisation

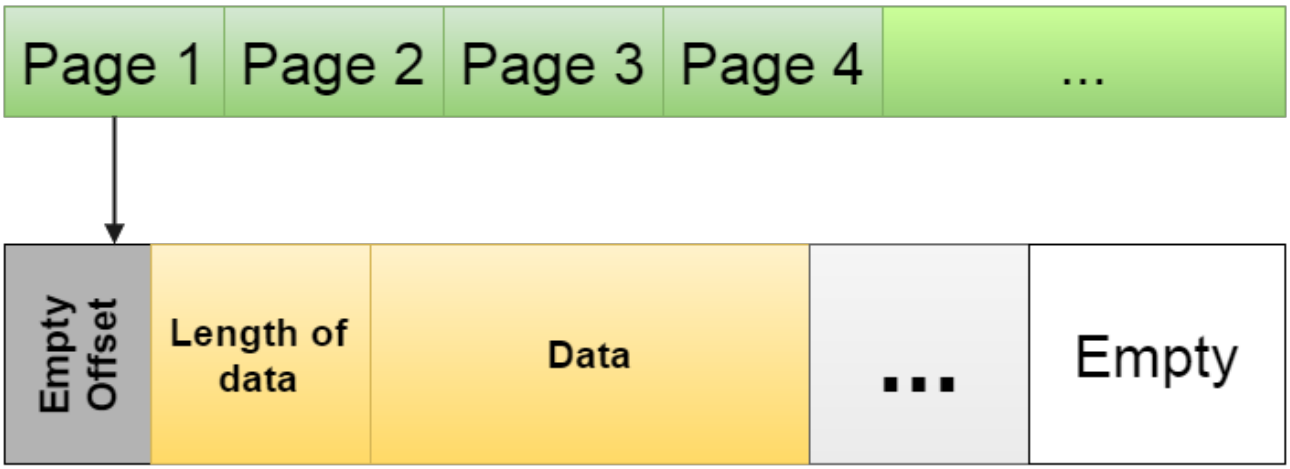

Figure 7: Disk Page Organisation

3.5 Application architecture

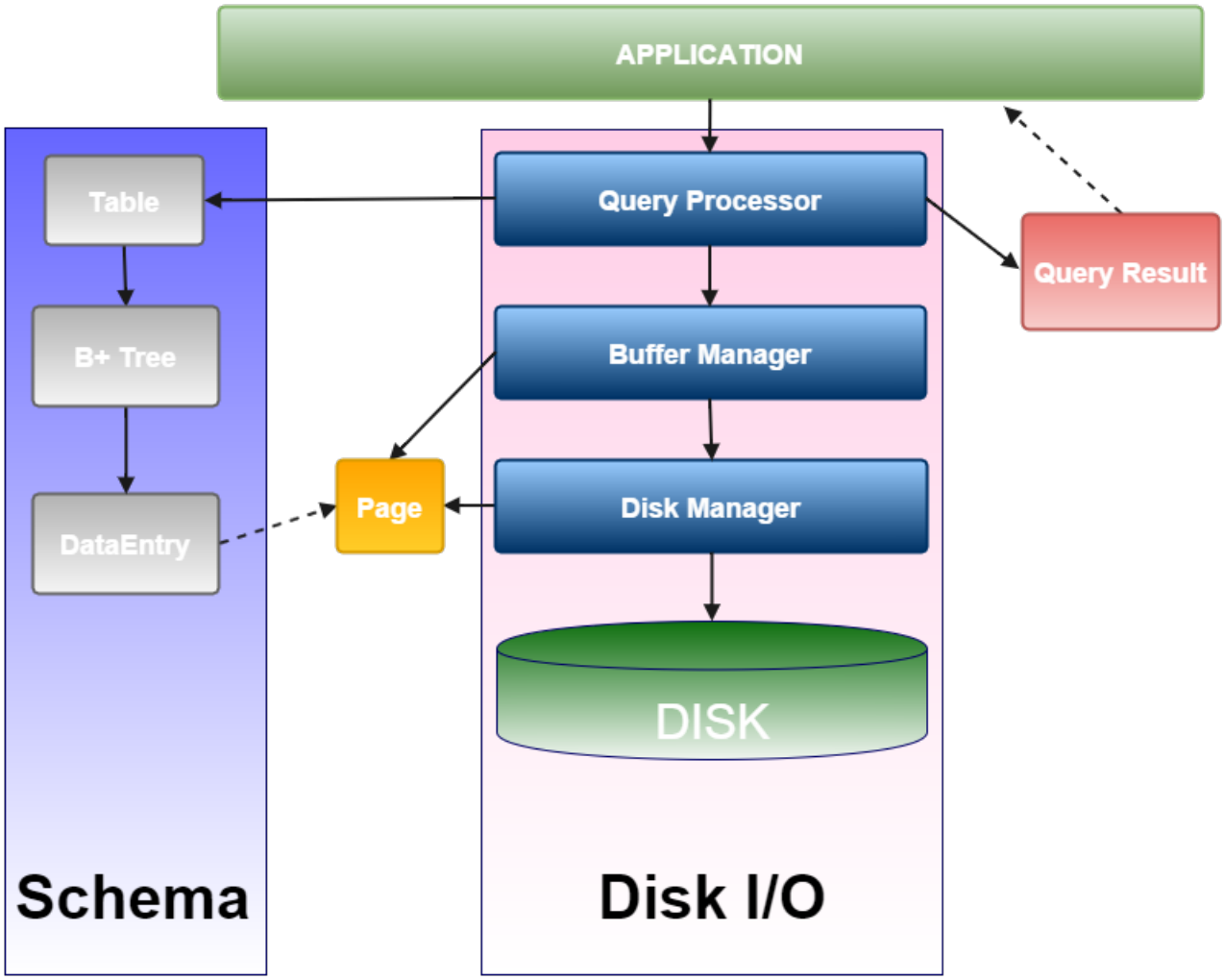

Figure 8: Application architecture

We use an unclustered file - there is no reason to arrange the data entries since it is too expensive to maintain order. To order data use B+ Tree indexes. Buffer manager is the top level of file I/O that optimises the Query processor and Disk interaction.

3.6 Insights

During the fulfilling of this Project we have implemented the knowledge acquired during the Data Modeling and Databases Course. We have touched the web development (Tornado framework), Python programming language. We organized our cooperation using the Git Hub source code management system. And we all gained some great team work experience.

References

[1] http://dblp.uni-trier.de/ DBLP

[2] https://aminer.org/billboard/citation aminer.org

[3] http://dblp.uni-trier.de/faq/dblpxml[1].pdf DBLP Some Lessons Learned by Michael Ley

[4] http://dblp.uni-trier.de/xml/docu/dblpxmlreq.pdf DBLP XML Requests

[5] https://www.python.org/ Python v.3.4

[6] http://www.tornadoweb.org Tornado v.4.2.1

[7] Introduction to Tornado. OReilly Media, Inc., 2012. ISBN: 978-1-449-30907-7

[8] http://www.postgresql.org/ Postgresql v.9.4.5

[9] https://pypi.python.org/pypi/psycopg2 Psycopg2 v.2.6.1

[10] Concepts of Database Management. Philip J. Pratt, Joseph J. Adamski. Cengage Learning, 2008. ISBN: 1-4239-0147-9

[11] Database management systems. Raghu Ramakrishnan, Johannes Gehrke. McGraw-Hill., 2003. ISBN: 0-07-246563-8-ISBN 0-07-115110-9 (ISE)

[12] A First Course in Database Systems. Jeffrey D. Ullman, Jennifer Widom. Pearson Education., 2008. ISBN: 978-0-13-502176-7