Database Systems

Course Project Instruction

Ma Dongzhe / 马冬哲
152 1062 0224
mdzfirst@gmail.com

Policy

- 2-3 persons form a team.
- 60% of your final score.
- In December, 3-4 outstanding teams will be invited to make a presentation.

The Task is

To Implement a DBMS Prototype.

What We Care

- Correctness
- Response Time
  - Storage
  - Access Method
  - Caching Strategy
  - Optimizing
  - Query Processing

What We Don’t Care

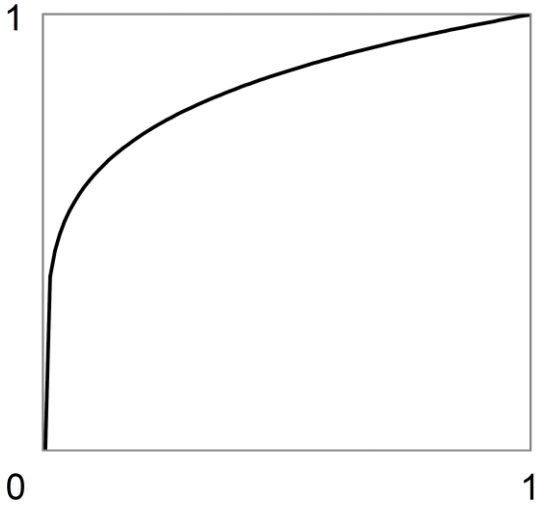
- Transaction Processing
- Concurrency Control
- Crash Recovery

Grading Criteria

Accomplishment	At least one correct run	10
Overall Evaluation	Correctness & Design & Code Quality & Contrib.	10
Performance	$S_j = \text{sum}((T_{i,best} / T_{i,j})^{0.2})$ Full * (S_j / S_{best})	30
Documentation	Content & Feature	10
Presentation	For some teams only	≤ 5

Example

	Workload 0	Workload 1
Team 0	5	100
Team 1	10	1000
Team 2	1	Fail



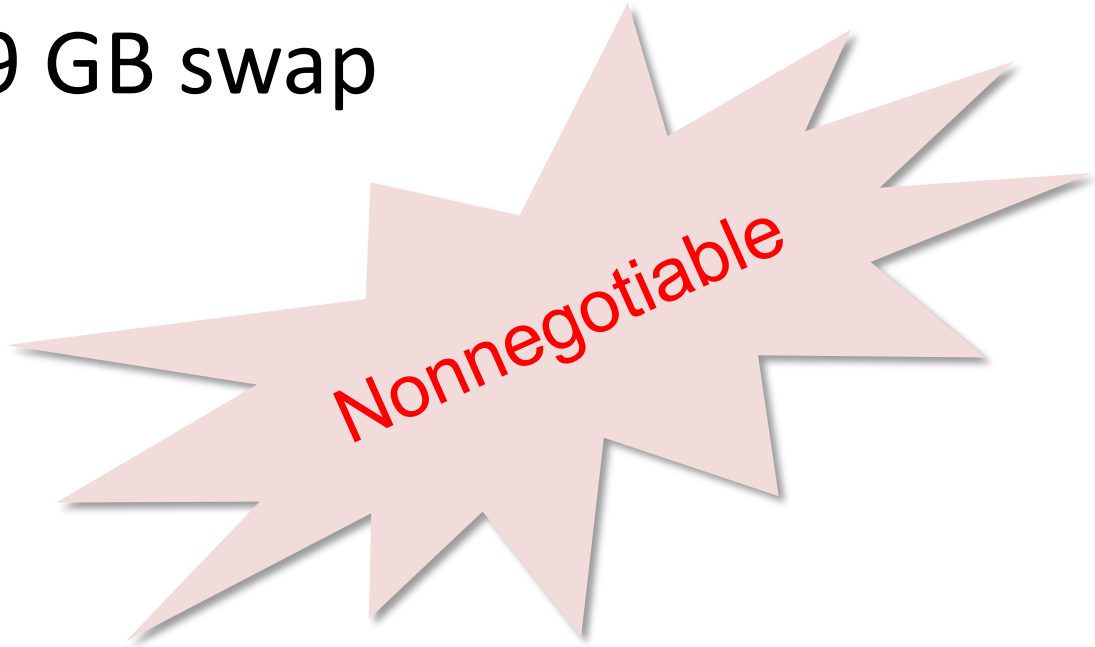
$S_0 = (1 / 5)^{0.2} + (100 / 100)^{0.2} = 1.725$
$S_1 = (1 / 10)^{0.2} + (100 / 1000)^{0.2} = 1.262$
$S_2 = (1 / 1)^{0.2} + (100 / INF)^{0.2} = 1$

$Score_0 = 30 * 1.725 / 1.725 = 30$
$Score_1 = 30 * 1.262 / 1.725 = 22$
$Score_2 = 30 * 1 / 1.725 = 17$

$(1 / 5)^{0.2} = 0.725$
$(1 / 10)^{0.2} = 0.631$
$(1 / 50)^{0.2} = 0.457$
$(1 / 100)^{0.2} = 0.398$
$(1 / 500)^{0.2} = 0.289$
$(1 / 1000)^{0.2} = 0.251$
$(1 / 5000)^{0.2} = 0.182$

The Environment is

- Ubuntu 10.04 LTS, 32-bit
- g++ 4.4.3
- Intel(R) Xeon(R) 5130 @ 2.00GHz x2
- 3.0 GB RAM, 1.9 GB swap



You Have to Implement

- create()
  Create a new table.
- train()
  Given some query information, train your system and choose the storage and access methods.
- load()
  Load initial data in csv format. The initial data set might be too large to keep in the main memory entirely.
- preprocess()
  Build the indexes and do other preprocessing.

See course/include/client.h for more details.

You Have to Implement

- execute()
  Execute a query or insert statement.
- next()
  Get the next row from the result set of the last query.
- close()
  Close the sockets and kill other threads.

See course/include/client.h for more details.

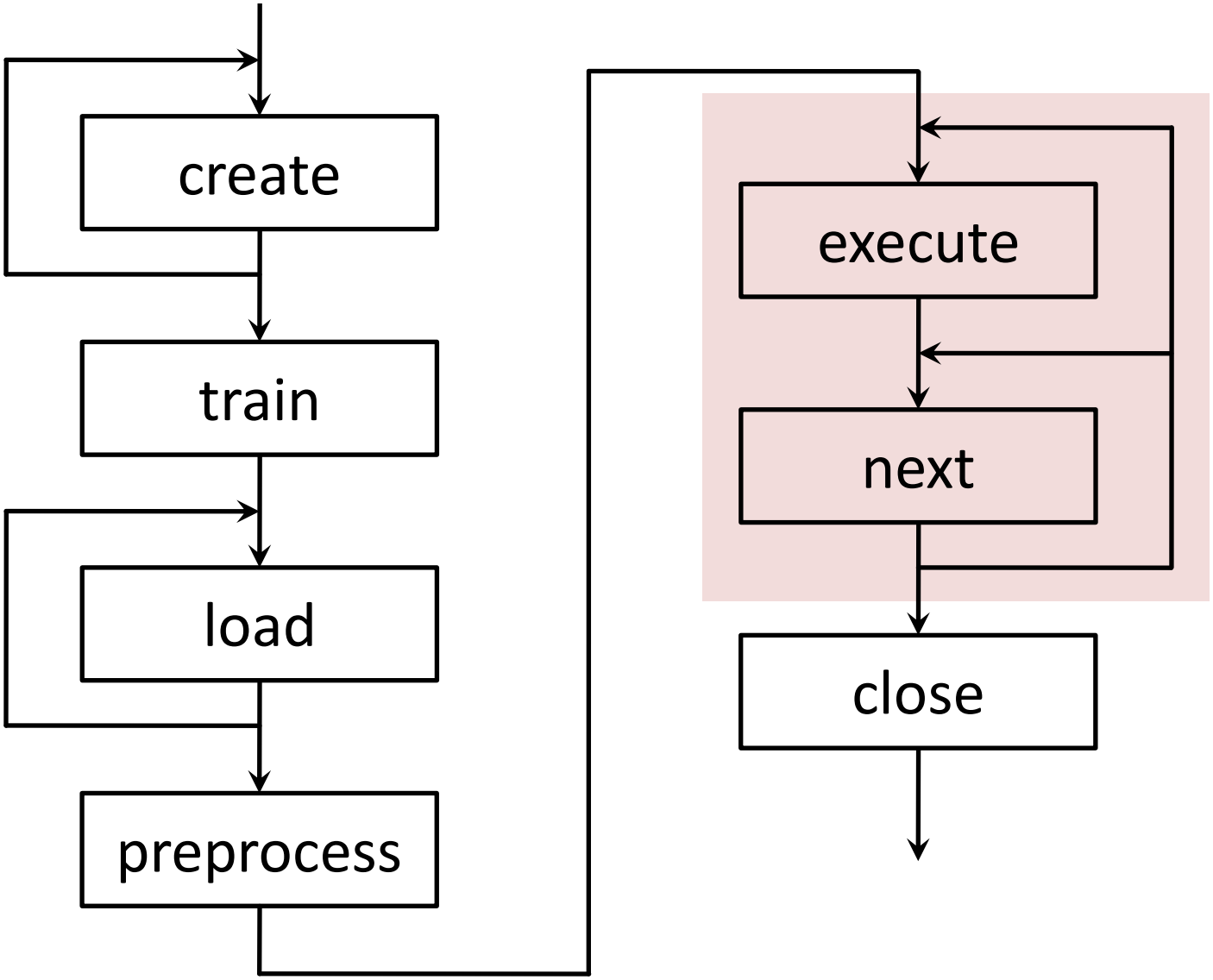
You Have to Implement

- execute()
  Execute a query or insert statement.
- next()
  Get the next row from the result set of the last query.
- close()
  Close the sockets and kill other threads.

WARNING: Run time of execute() and next() will be measured.

See course/include/client.h for more details.

Test Procedure



Query Statement

```
SELECT column0, column1, …
FROM table0, table1, …
WHERE condition0 AND … AND conditionN;
```

A condition could be

```
column = constant
column < constant
column > constant
column0 = column1
```

(For integers only)

(For integers only)

(Join condition)

Query Statement

```
SELECT column0, column1, …
FROM table0, table1, …
WHERE condition0 AND … AND conditionN;
```

No prefix

A condition could be

```
column = constant
column < constant
column > constant
column0 = column1
```

(For integers only)

(For integers only)

(Join condition)

Query Statement

```
SELECT column0, column1, …
FROM table0, table1, …
WHERE condition0 AND … AND conditionN;
```

A condition could be

```
column = constant
column < constant   (For integers only)
column > constant   (For integers only)
column0 = column1   (Join condition)
```

Same type

Query Statement

```
SELECT column0, column1, …
FROM table0, table1, …
WHERE condition0 AND … AND conditionN;
```

The only operator

A condition could be

```
column = constant
column < constant   (For integers only)
column > constant   (For integers only)
column0 = column1   (Join condition)
```

Query Statement

```
SELECT column0, column1, …
FROM table0, table1, …
WHERE condition0 AND … AND conditionN;
```

A condition could be

If the FROM-clause contains only one table, there might be no WHERE-clause.

```
column = constant
column < constant   (For integers only)
column > constant   (For integers only)
column0 = column1   (Join condition)
```

Insert Statement

```
INSERT INTO table
VALUES (value_list0), …, (value_listN);
```

All value lists are in csv format.

```
constant0,constant1,…,constantN
```

Insert Statement

```
INSERT INTO table
VALUES (value_list0), …, (value_listN);
```

No column list

All value lists are in csv format.

```
constant0,constant1,…,constantN
```

Insert Statement

```
INSERT INTO table
VALUES (value list0), …, (value listN);
```

All value lists are in csv format.

Number of rows is important for the train() routine.

```
constant0,constant1,…,constantN
```

Format

```
SELECT_a_,_b_
FROM_A_,_B_
WHERE_a_=_5_AND_b_<_10_;

INSERT_INTO_A_VALUES_
(_0,'Stalin',1879_)_,_
(_1,'Roosevelt',1882_)_;
```

Data Types

- INTEGER
  32-bit unsigned integer, ‘int’ is OK.
- VARCHAR(d)
  Consist of _, a-z, A-Z, or 0-9. Enclosed by single quotes. At most d characters (excluding the quotes).

NOTE:

All identifiers (table names and column names) are string constants not starting with 0-9.

Columns in different tables have distinct names.

String constants don’t contain space, quote, or comma.

Primary Keys

- Primary keys will be assigned to all relations.
- The primary keys will be unique. There is no need to check this constraint.
- The primary keys will be given in ascending order.
- You can just ignore them.

Join Operations

Let nodes represent tables and edges represent join conditions, then each query can be transformed into a graph. This graph should be a tree which

- is connected;
- contains no self-cycles;
- contains no duplicate edges;
- contains no cycles (at least 3 nodes).

Workloads

- Projection
- Selection
- Join

- TPC-C
- TPC-H

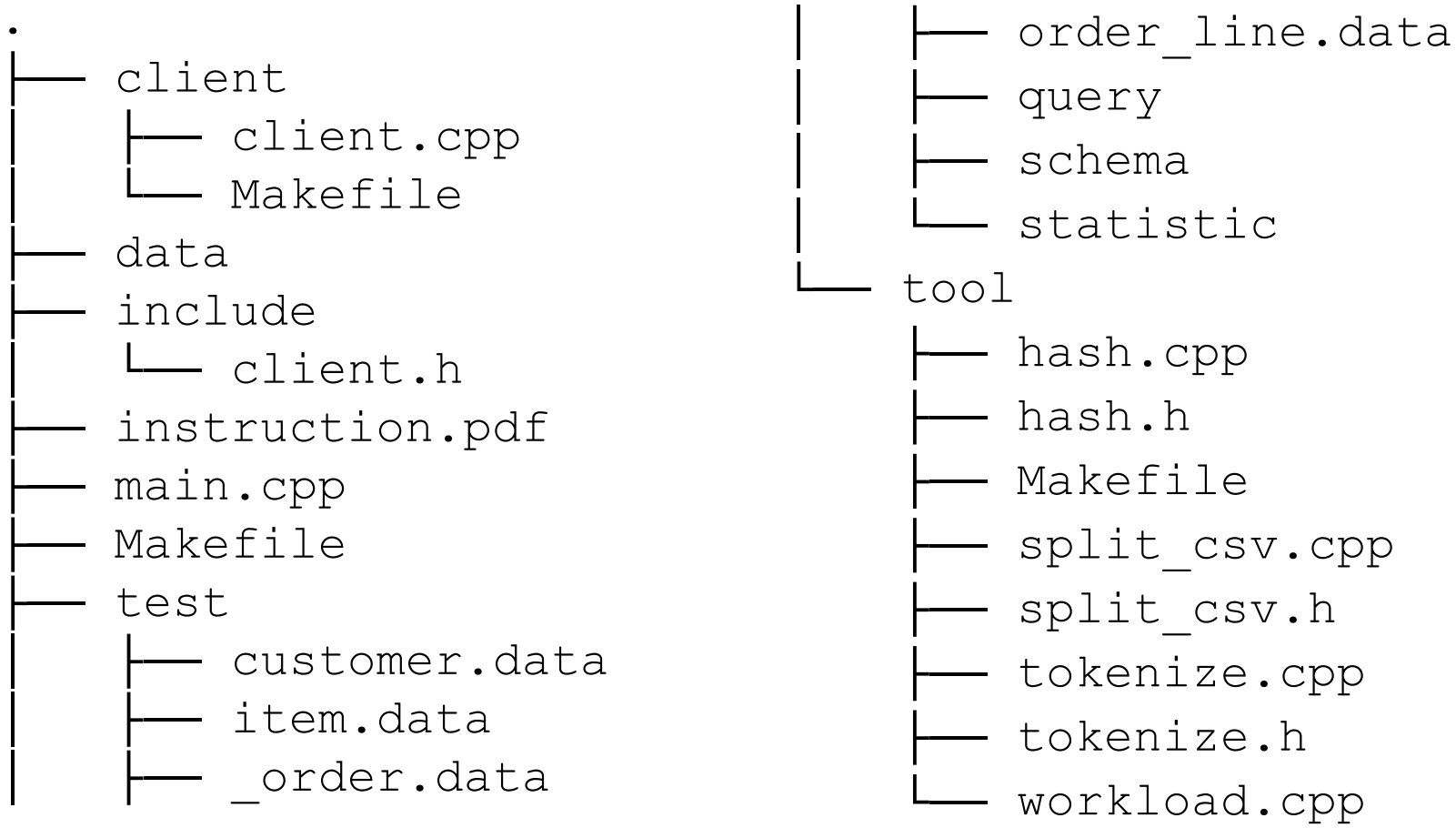
Workloads

- Projection
- Selection
- Join

```
string workload();
```

- TPC-C
- TPC-H

Directory Structure

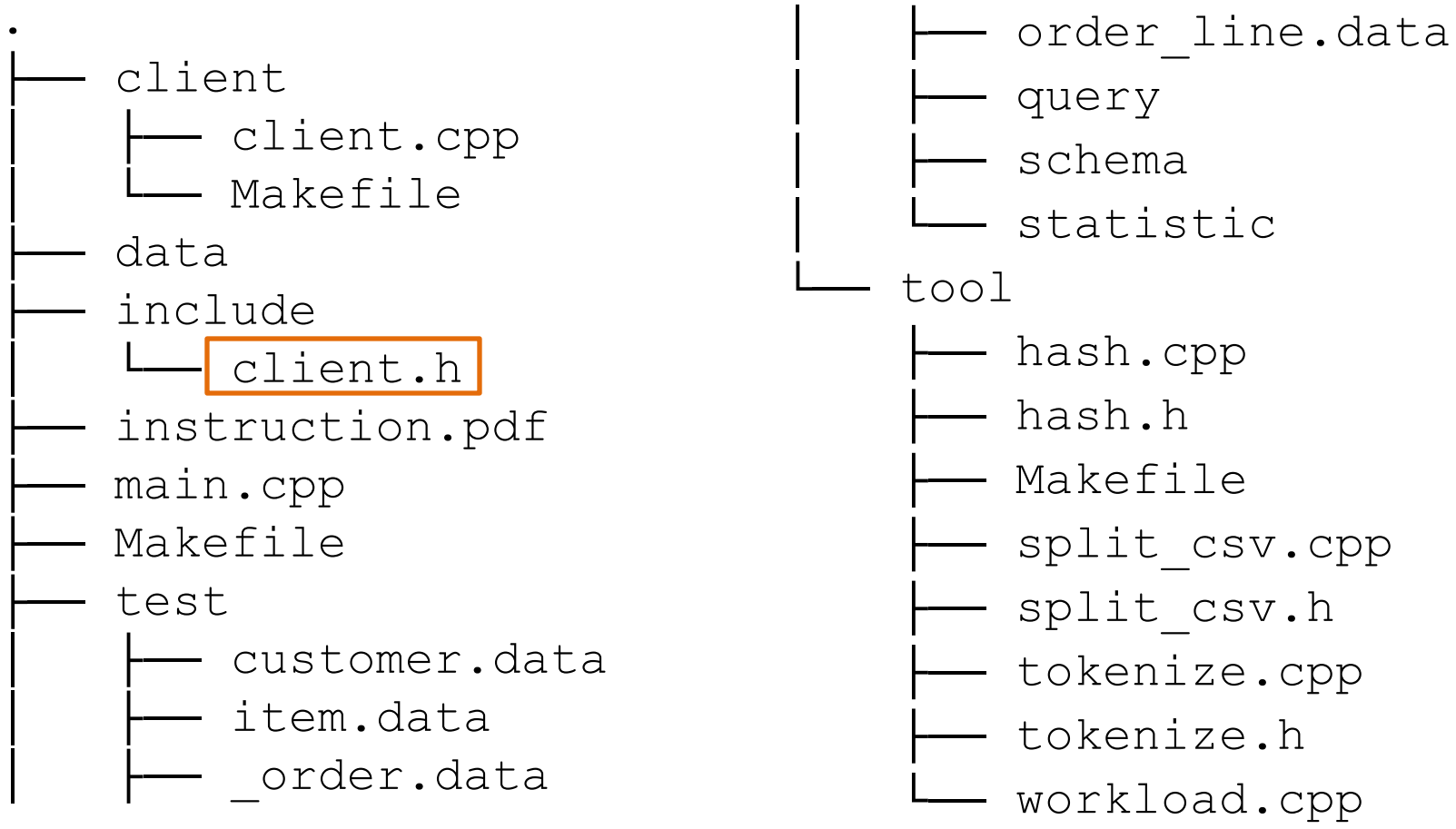


Directory Structure

```
.
├── client
│   ├── client.cpp
│   └── Makefile
├── data
├── include
│   └── client.h
├── instruction.pdf
├── main.cpp
├── Makefile
├── test
│   ├── customer.data
│   ├── item.data
│   ├── _order.data
│   ├── order_line.data
│   ├── query
│   ├── schema
│   └── statistic
└── tool
    ├── hash.cpp
    ├── hash.h
    ├── Makefile
    ├── split_csv.cpp
    ├── split_csv.h
    ├── tokenize.cpp
    ├── tokenize.h
    └── workload.cpp
```

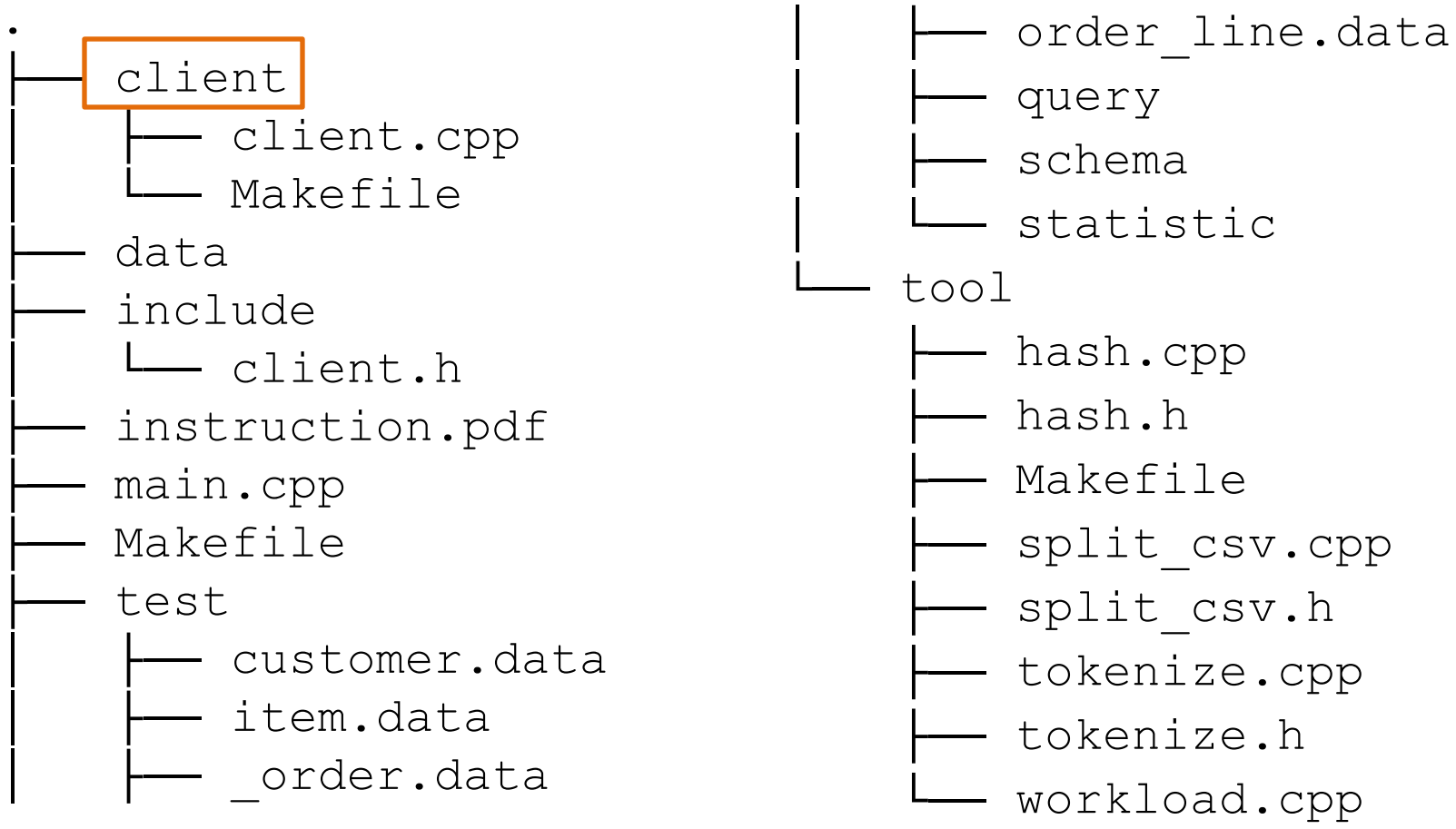
Test procedure. This file will be modified.

Directory Structure



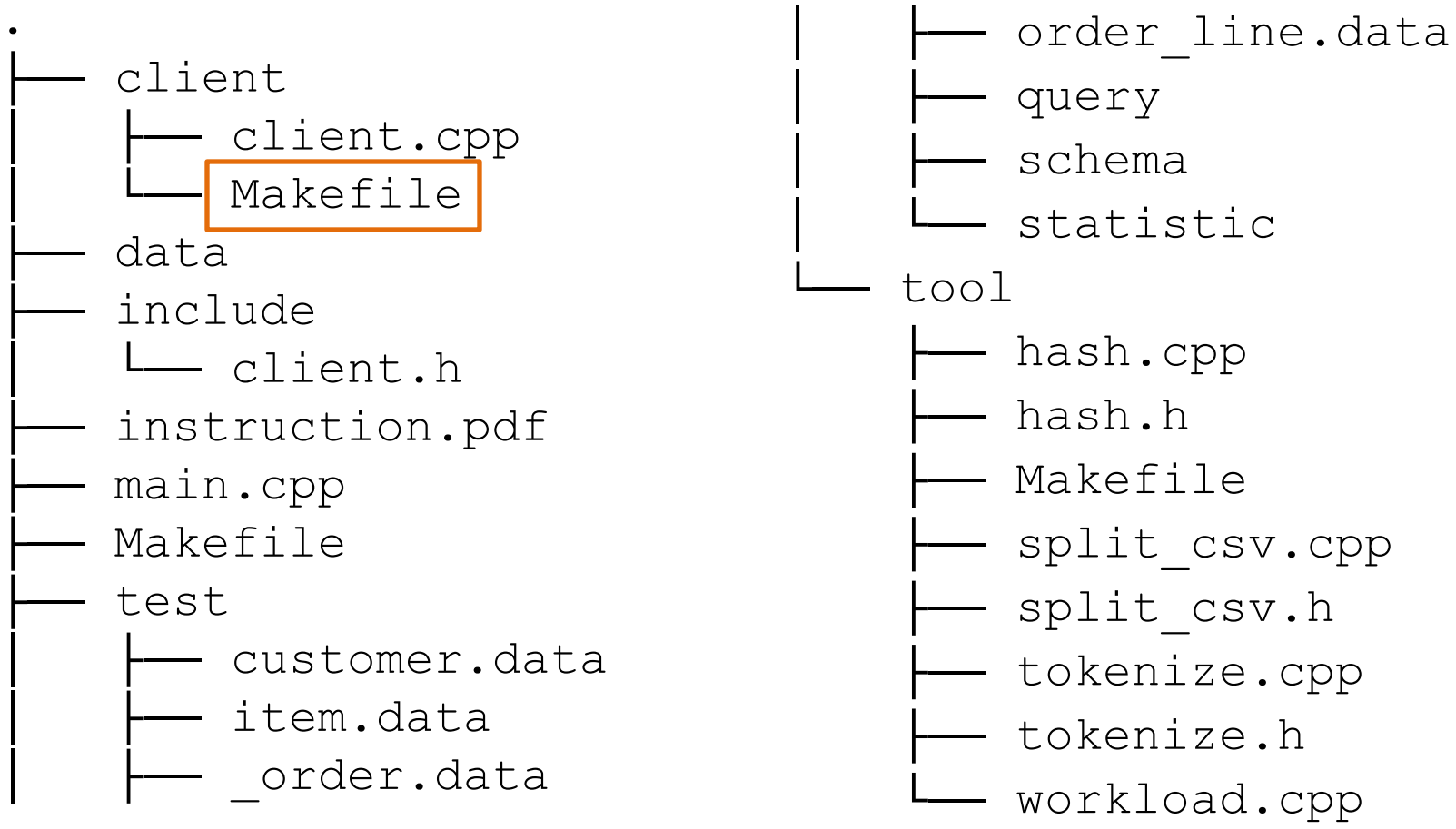
APIs to implement.

Directory Structure



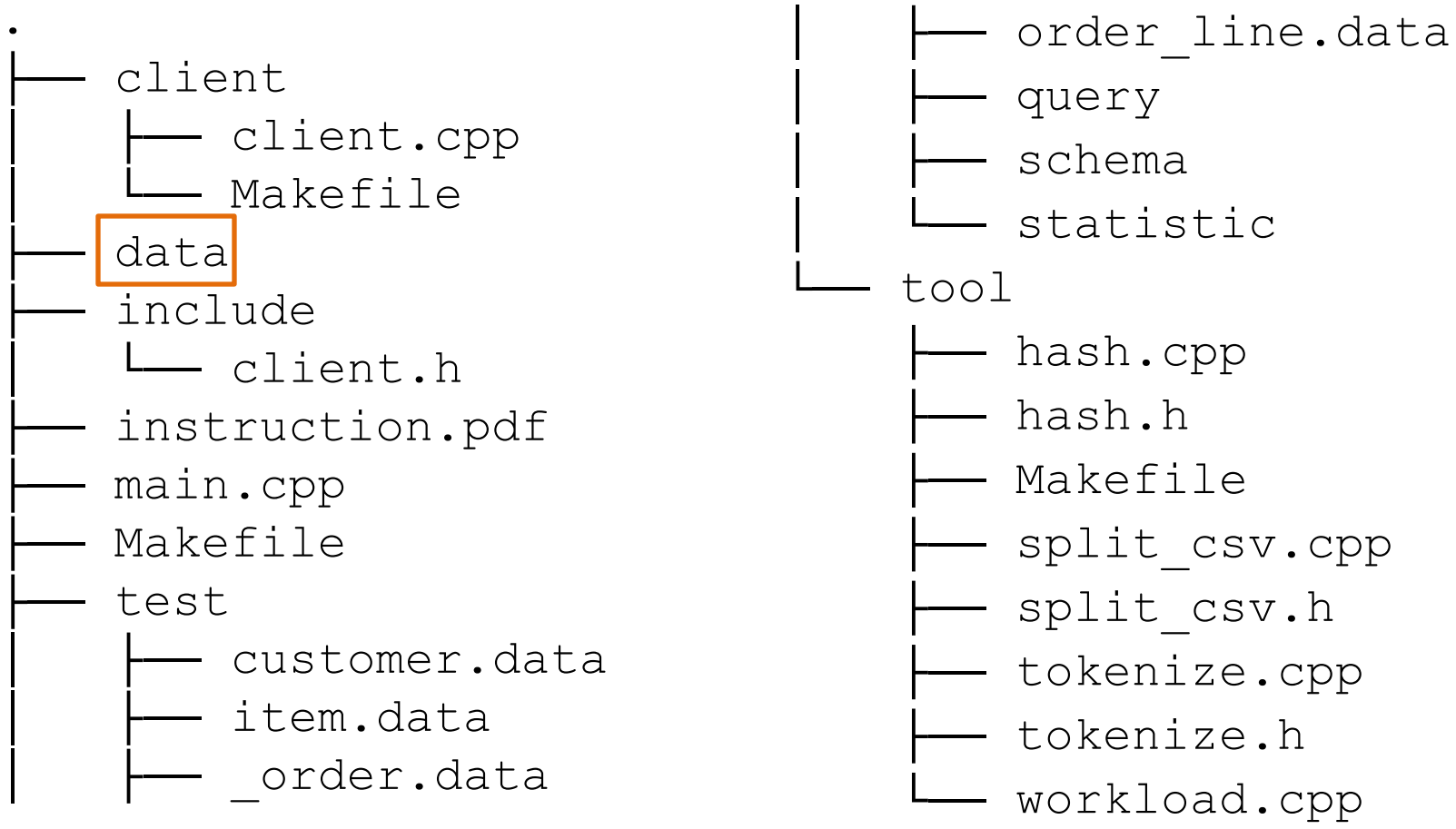
Keep all your source codes in this directory.

Directory Structure



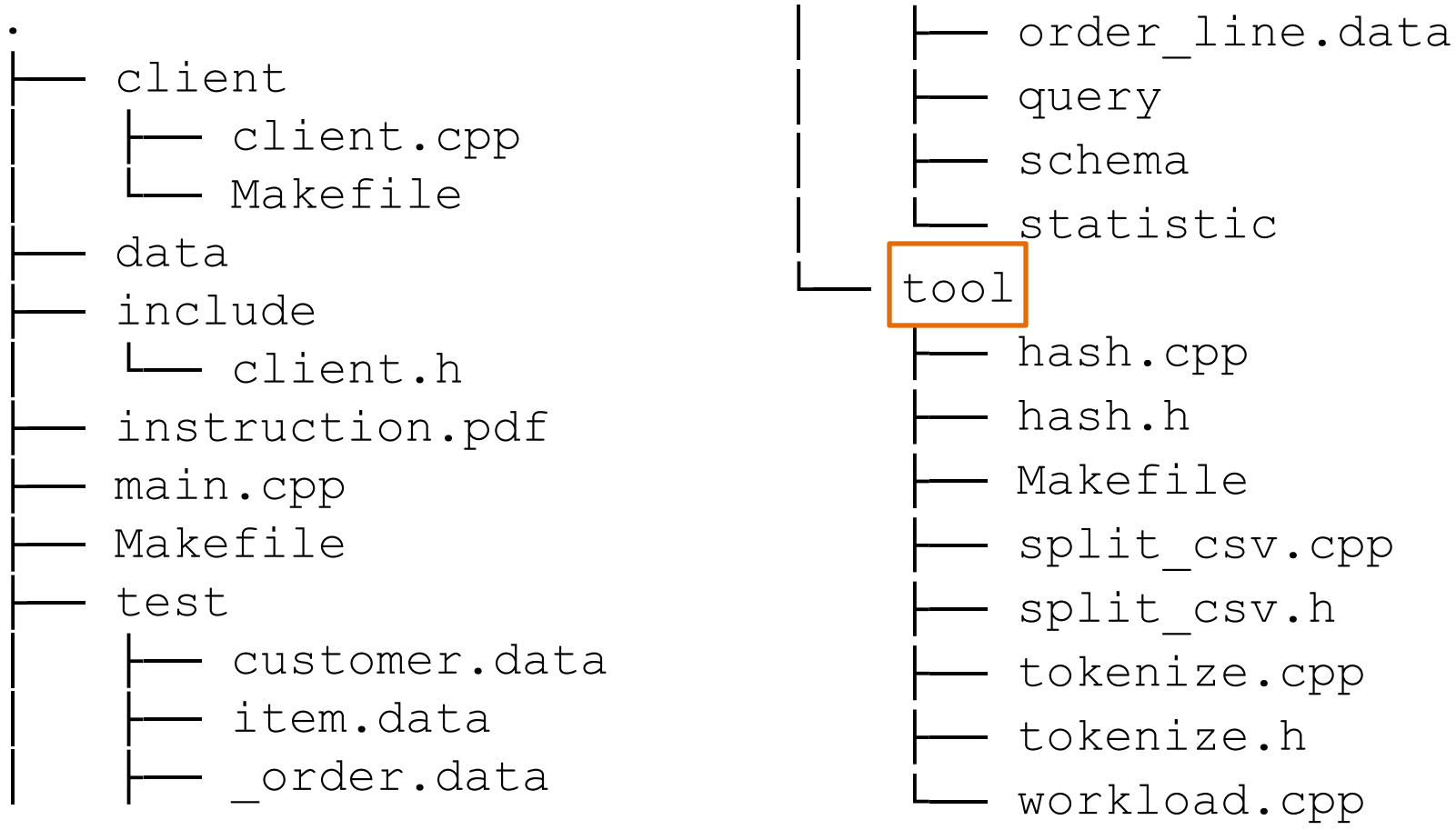
Make sure your Makefile is correct.

Directory Structure



Keep all your data in this directory.

Directory Structure



Do NOT use any routines in other folders.

About Third-party Library

- You are free to use any third-party library or code about storage, index, multi-thread, network, etc.
  e.g. Boost, Berkeley DB, open-source disk-based B-tree / hash table implementation, etc
- You are forbidden to use any system that is capable to process a SQL query.
  e.g. MySQL, PostgreSQL, etc
- Ask for confirmation if you are not sure.

Document / Presentation

- System Architecture
- Storage Model and the Selection Strategy
- Index Structure and the Selection Strategy
- Caching Strategy
- Query Processing Strategy
  - Heuristic Rules
  - Cost Model
- Other features of your system
- References
- Personal Contribution Rate (For document)

Submission

- Prepare your submission with *make tar*.
- Submit the **.tar.gz* to …
- One submission every 2 hours.
- Only the last submission counts.

Notice

- Some tests may depend on the result of another one.
- Time limitation applies to the whole program, although only execute() and next() affect your final scores.
- Not all the queries are provided in train(), although only those included are measured.

Warnings

- Never do irrelevant operations
- Never replicate other team’s work

Hints

- Read some books and research papers
- Discuss with others
- Start ASAP

create()

Keep the schema safe.

train()

- Find affinitive tables.
- Find affinitive attributes.
- Choose access methods.
- Read-intensive or update-intensive?

load()

Keep the data safe.

preprocess()

- Make some useful statistics.
- Build some indexes.
- Start some threads.

Statistics

- For uniform distribution
  - Size(R), Cnt(R), Card(A), Min(A), Max(A)
  - SF(A = value) = 1 / Card(A)
  - SF(A > value) = (Max(A) - value) / Range(A)
  - SF(A < value) = (value – Min(A)) / Range(A)
  - SF(A0 ∧ A1) = SF(A0) * SF(A1)
- For skewed distribution (e.g., Zipf)
  - Histogram

execute() and next()

- Do all the jobs in execute().
- Do all the jobs in next().
- Do all the jobs in independent threads.

Query Processing

- Google ‘query processing’
- Search on ACM Digital Library (*dl.acm.org*)

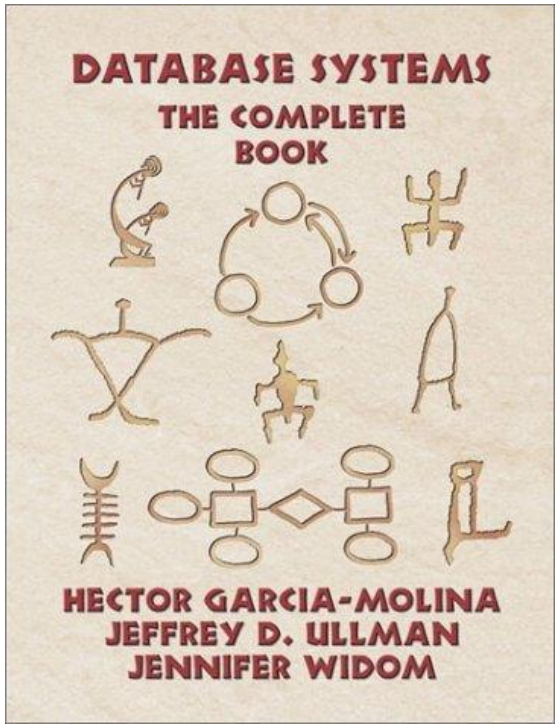
- Cost model vs. Rules

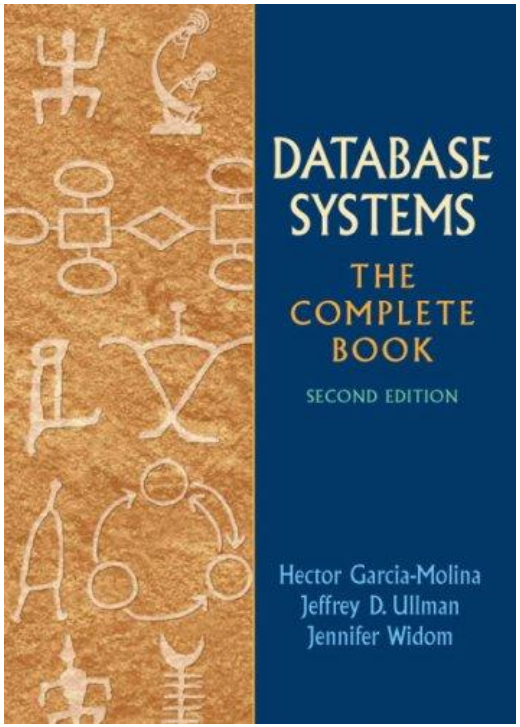
Join Operation

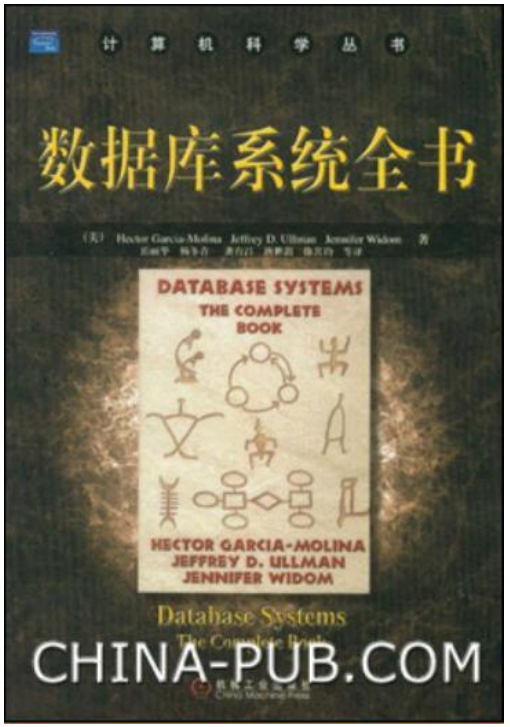
- Nested Loop Join
- Index-based Nested Loop Join
- Sort-Merge Join
- Hash Join (Pruning)

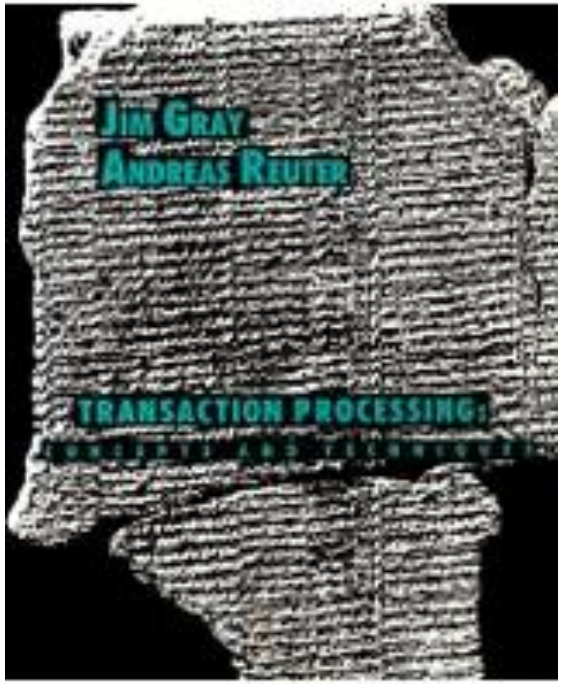
close()

Close your program safely.


DATABASE SYSTEMS
THE COMPLETE BOOK
HECTOR GARCIA-MOLINA
JEFFREY D. ULLMAN
JENNIFER WIDOM

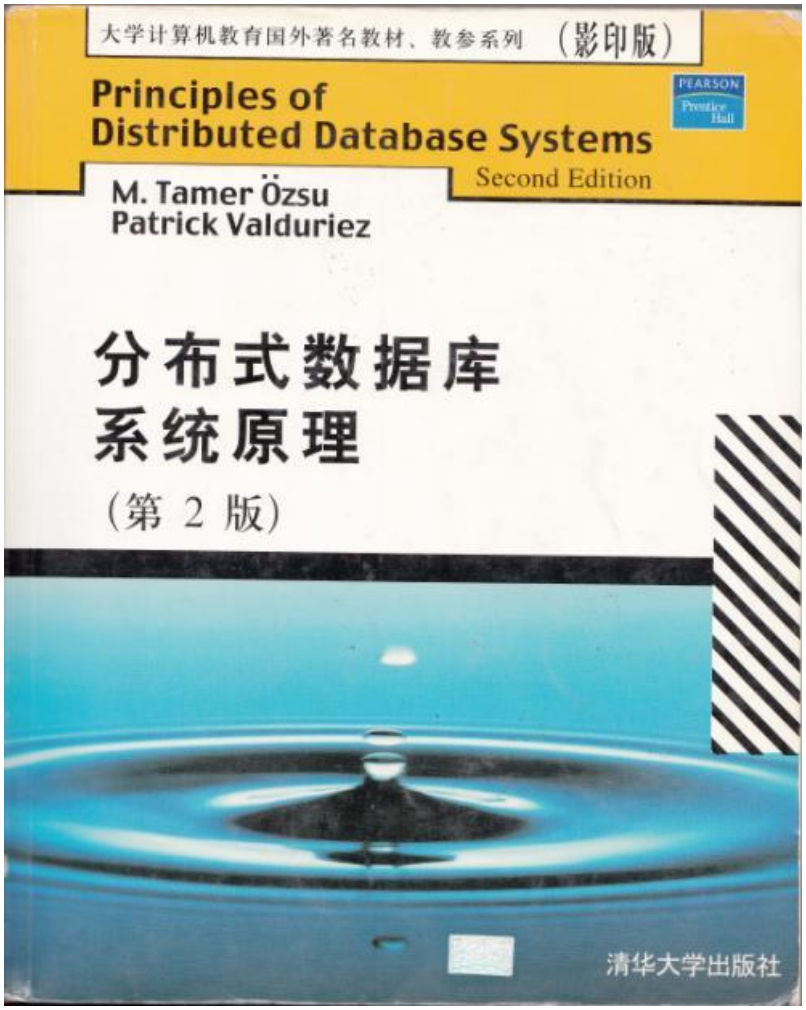

DATABASE SYSTEMS
THE COMPLETE BOOK
SECOND EDITION
Hector Garcia-Molina
Jeffrey D. Ullman
Jennifer Widom


计 算 机 科 学 丛 书
数据库系统全书
DATABASE SYSTEMS
THE COMPLETE BOOK
HECTOR GARCIA-MOLINA
JEFFREY D. ULLMAN
JENNIFER WIDOM
Database Systems
CHINA-PUB.COM


JIM GRAY
ANDREAS REUTER
TRANSACTION PROCESSING:


Transaction Processing
Concepts and Techniques
事务处理
概念与技术
(英文版)
人民邮电出版社


计 算 机 科 学 丛 书
事务处理
概念与技术
JIM GRAY
ANDREAS REUTER
TRANSACTION PROCESSING:
Transaction Processing
Concepts and Techniques


大学计算机教育国外著名教材、教参系列 (影印版)
Principles of
Distributed Database Systems
Second Edition
M. Tamer Özsu
Patrick Valduriez
分布式数据库
系统原理
(第 2 版)
清华大学出版社

Good luck and have fun!