Overview

- Database & Database Management System
- Relational Database
- Simple SQL Queries
- Database normalization
- RDBMS for an Inverted Text Index
Database System Today

- Tremendously huge data processing
- Horizontal Scalability
- Concurrency Model
What are DB & DBMS than?

- A database (DB) is a collection of data describing the activities of 1 or more related organization, eg. University database:
  - Entities: students, faculty, courses, classrooms
  - Relationship between entities:
    - Students' enrollment in courses
    - Faculty teaching courses
    - The use of rooms for courses
- A Database Management System (DBMS) is a software designed to assist in maintaining & utilizing large collection of data eg.:
  - Part of software industry: Oracle, Microsoft, Sybase
  - Open source:
    - Relational: MySQL, PostgreSQL, SQLite
    - Text search: APACHE Lucene (SOLR, HADOOP), Ferret, ....
Storing Data: File System vs DBMS

- Data can be stored in RAM
  - That is what most programming language offers
  - RAM is fast, random access but volatile

- File System offered by every OS:
  - Stores data in files with diverse formats in disk
    - Implication ⇒ program using these files depend on the knowledge about that format
  - Allows data manipulation (open, read, write, etc.)
  - Allows protection to be set on a file

- Drawbacks:
  - No standards of format
  - Data duplication & dependence
  - No provision for concurrency & security
Quizzes

• Quiz 1:
  – You & your colleague are editing the same file.
  – You both save it at the same time
  – Whose changes survive?

• Quiz 2:
  – You & your colleagues login in the LMU portal.
  – Both of you are editing your addresses.
  – You both click the send button at the same time
  – Whose changes survive?
Storing Data: File System vs DBMS

• Database Management system:
  – Simple, efficient, ad hoc queries
  – Concurrency controls
  – Recovery, Benefits of good data modelling
  – Stores information in disks
  – This has implication for database design:
    ● READ : transfer data from disk to main memory (RAM)
    ● WRITE : transfer data from RAM to disk
  – In relational DBMS:
    ● Information is stored as *tuples* or *records* in *relations* or *tables*.
    ● Making use of relational Algebra
Relational Database

- Relational Database Management System (RDBMS) consists of:
  - A set of tables
  - A schema

- A schema:
  - is a description of data in terms of data model
  - Defines tables and their attributes (field or column)

- The central data description construct is a relation:
  - Can be thought as records
  - eg. information on student is stored in a relation with the following schema:

  \[
  \text{Student}(\text{sid}: \text{string}, \text{sname}: \text{string}, \text{login}: \text{string}, \text{gpa}: \text{numeric})
  \]
Relational Database

- Tables $\equiv$ relation:
  - is a subset of the Cartesian product of the domains of the column data type.
  - Stores information about an entity or theme
  - Consist of columns (fields) and rows (records).
  - Rows $\equiv$ tuple, describing information about a single item, eg. A specific student
  - columns $\equiv$ attributes, describing a single characteristic (attributes) of its item, eg. Its ID number, GPA, etc
  - Every row is unique & identified by a key

- Entity is
  - an object in the real world that is distinguishable from other objects. eg. Students, lecturers, courses, rooms.
  - Described using a set of attributes whose domain values must be identified.

  - The attribute 'name of Student' $\Rightarrow$ 20-character strings
Creating Relational Database

- How to create relational database?
  - Need RDBMS (MySQL, Oracle, etc)
  - Just take MySQL as an open source RDBMS
    - With user Interface
      - eg. phpMyAdmin → providing graphical user interface
      - Free to use any scripts or programming languages
    - Using SQL commands in terminal
    - Using SQL integrated in your code
Creating Relational Database

• How to create relational database in GUI?
  – Step 1: install XAMPP (just an example) a cross-platform Apache HTTP Server, MySQL Server & interpreters for script
  – Step 2: start your XAMPP first:
    /xampp_or_lampp_path start
    eg. /opt/lampp/lampp start
  – Open your browser, and type:
    localhost/phpmyadmin
RDBMS Example

- Database Server: MySQL 5.5.27
- Web Server: Apache through XAMPP Package
RDBMS Example

- Creating table, defining attributes & their domains

Table name: _Add 1 column(s) Go_

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Length/Values</th>
<th>Default</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Index</th>
<th>A.I</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INT</td>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table comments: _Storage Engine: InnoDB_

PARTITION definition: _Collation:_
### RDBMS Example

- Creating table, defining attributes & their domains

---

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
<th>Action</th>
<th>Action</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sid</td>
<td>varchar(6)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change</td>
<td>Drop</td>
<td>Browse distinct values</td>
</tr>
<tr>
<td>2</td>
<td>SName</td>
<td>varchar(35)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change</td>
<td>Drop</td>
<td>Browse distinct values</td>
</tr>
<tr>
<td>3</td>
<td>Login</td>
<td>varchar(25)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
<td>Change</td>
<td>Drop</td>
<td>Browse distinct values</td>
</tr>
<tr>
<td>4</td>
<td>GPA</td>
<td>float(2.1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change</td>
<td>Drop</td>
<td>Browse distinct values</td>
</tr>
</tbody>
</table>

---

**Check All / Uncheck All With selected:**
- Browse
- Change
- Drop
- Primary
- Unique
- Index
- More
**RDBMS Example**

- Each relation is defined to be a set of unique tuples of rows

<table>
<thead>
<tr>
<th>Sid</th>
<th>SName</th>
<th>Login</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL0001</td>
<td>David</td>
<td>david@cis</td>
<td>1.3</td>
</tr>
<tr>
<td>CL0002</td>
<td>Wenpeng</td>
<td>hansying@cis</td>
<td>1.5</td>
</tr>
<tr>
<td>CL0003</td>
<td>Yadoll</td>
<td>yalah@cs</td>
<td>1.7</td>
</tr>
<tr>
<td>CL0004</td>
<td>Bastian</td>
<td>basti@cis</td>
<td>1.3</td>
</tr>
<tr>
<td>CL0005</td>
<td>Dewika</td>
<td>krisna@cl</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Key Constraints

- Key constraint is a statement that a certain minimal subset of the relation is a unique identifier for a tuple.

- Two Types of keys:
  - Primary key:
    - Sid is a primary key for student,
    - Cid is a primary key for Course
  - Foreign key

- Primary key:
  - a unique identifier for a tuple (row)
    - Sid is a primary key for student,
    - Cid is a primary key for Course
  - Primary key fields are indexed
Key Constraints

- Foreign key:
  - A kind of a logical pointer
  - A key to refer to relation with other tables & should match the primary key of the referenced relation
  - Foreign key fields are also often indexed if they are important for retrieval.

courses(Cid, Cname, Instructor, Semester )
Student(Sid, Sname, login, GPA)

How do you express which students take which course?
Key Constraints

- Need a new table:
  - enrolled(Cid, grade, Sid)
  - Sid/Cid in enrolled are foreign keys refering to Sid in Student table & Cid in Courses.

<table>
<thead>
<tr>
<th>Cname</th>
<th>Cid</th>
<th>Enrolled</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning</td>
<td>CL104</td>
<td>CL104 0 CL0002</td>
<td>CL0001 David</td>
</tr>
<tr>
<td>Information Retrieval</td>
<td>CL214</td>
<td>CL104 0 CL0004</td>
<td>CL0002 Wenpeng</td>
</tr>
<tr>
<td>Information Extraction</td>
<td>CL223</td>
<td>CL223 0 CL0001</td>
<td>CL0003 Yadoll</td>
</tr>
<tr>
<td>Statistics</td>
<td>CL114</td>
<td>CL114 0 CL0005</td>
<td>CL0004 Bastian</td>
</tr>
<tr>
<td>Syntax</td>
<td>CL313</td>
<td>CL313 0 CL0003</td>
<td>CL0005 Dewika</td>
</tr>
</tbody>
</table>
Relations

- One to one:
  - Each primary key relates only one record in related table

- One to many:
  - The primary key relates to one or many records in related table

- Many to Many:
  - The primary key relates to many records in related table, and a record in related table can relate to many primary keys on another table
Storing Relationships using Keys

- Modeling data is one thing, storing it in a database is another one.

- In relational database, the 'rules' are:
  - If the relationship to be stored is 1:N, place the attribute identified as the primary key from the one table as a foreign key in another table.
  - If the relationship to be stored is M:N, a new table structure must be created to hold the association. This 'bridge' table will have as foreign key attributes, the primary key of each table that is part of relationship.

- The key for the 'bridge' table then becomes either:
  - The combination of all the foreign keys OR
  - A new attribute will be added as a surrogate key
Storing Relationships using Keys
Indexes in MySQL

• A database index is
  – a data structure that improves the speed of operations in a table
  – Unseen table created by DB engine that keeps indexed fields and its pointers to each record into the actual table.

• Indexes in MySQL:
  – Primary key
  – Unique indexes:
    • All values in the indexed column must be distinct though it's unnecessarily indexed as a primary key
  – Index:
    • Refers to a non-unique index, used for speeding the retrieval
Indexes in MySQL

• Indexes in MySQL:
  – Fulltext:
    • An index created for full text searches
    • Supporting storage engines: InnoDB & MyISAM
    • Data type: CHAR, VARCHAR, TEXT
  – Spatial Index:
    • for spatial data types
    • Uses R-tree indexes

• Example of index usage:
  – „Find all students with GPA < 1.7“
    • May need to scan the entire table
    • Index consists of a set of entries pointing to locations of each search key
Data Type in MySQL

- **String:**
  - Char, varchar, text, (tiny, medium, long)
  - Binary, varbinary
  - Blob (tiny, medium, long), enum, set

- **Date & time**

- **Numeric**
  - Int (tiny, small, medium, big)
  - Decimal, float, double, real
  - BIT, boolean, serial

- **Spatial:**
  - Geometry, point, linestring, polygon, etc
Structured Query Language (SQL):
- Is a standard language used to communicate with a relational database.
- Is used in conjunction with procedural or object-oriented languages/scripts such as Java, Perl, Ruby, Python, etc.

Sql basic conventions:
- Each statement begins with a command, eg. CREATE, SELECT
- Each statement ends with delimiter usually a semicolon (;)
- Statements are written in a free-form style, eg. SELECT...FROM... WHERE...
- SQL statement is not case-sensitive, except inside string constant, eg SELECT...FROM... WHERE SName = 'Yadolf'
Simple SQL Queries

- The basic form of SQL Queries is:
  ```sql
  SELECT select-list (column_name)
  FROM from-list (table_name)
  WHERE condition
  ```
- Selecting all students with GPA above 1.7
  ```sql
  SELECT Sid, Sname FROM student WHERE GPA <= 1.7
  ```
- Selecting all information from a table
  ```sql
  SELECT * FROM enrolled
  ```
- Selecting course name with pattern matching
  ```sql
  SELECT Cname FROM Courses WHERE Cname LIKE 'Machine %'
  ```
Simple SQL Queries

- **INSERT:**
  
  ```sql
  INSERT INTO `Students` VALUES (CL0001, David, david@cis, 1, 3)
  
  INSERT INTO `Students` VALUES (sid, sname, login, gpa)
  ```

- **ALTER:**
  
  ```sql
  ALTER TABLE `Students` ADD `Intakeyear`
  
  ALTER TABLE `Lecturer` ADD INDEX(`courses`)  
  ```

- **Using logical connectives:**
  
  - AND, OR, NOT may be used to construct a condition
    
    ```sql
    SELECT `cname` FROM `courses` WHERE semester = 'summer' AND ctype = 'seminar'
    ```

- **Joining Tables:**
  
  ```sql
  SELECT `Sname` FROM `Students`, `Courses` WHERE Students.sid = Courses.sid
  ```
Simple SQL Queries

• Creating Table:

```
CREATE TABLE `Students` (
  `Sid` varchar(6) NOT NULL,
  `SName` varchar(35) NOT NULL,
  `Login` varchar(25) NOT NULL,
  `GPA` float(2,1) NOT NULL,
  PRIMARY KEY (`Sid`)
) ENGINE=InnoDB CHARSET= Latin1;
```
Creating Database Through Terminal

- Open your terminal console
- Go to the path where you save your MySQL
- If you install XAMPP:
  - You need to start XAMPP as a SU/root
  - To get the action commands (in Linux), type:
    /opt/lampp/lampp
  - Start only MySQL Server, type:
    /opt/lampp/lampp startmysql
  - To stop MySQL, type:
    /opt/lampp/lampp stopmysql
  - To start XAMPP (Apache, MySQL & others), type:
    /opt/lampp/lampp start
Creating Database Through Terminal

• If you install XAMPP:
  – go to the path where mysql is saved, in Linux it is usually saved in bin, so type:
    /opt/lampp/bin/mysql -u username -p password
  – If you are already in mysql path:
    • To see the databases. Type:
      SHOW DATABASES;
    • To create a database, use SQL command:
      CREATE DATABASE database_name;
    • Creating database does not select it for use, so type:
      USE database_name;
    • To delete database:
      DROP DATABASE database_name;
    • Use SQL commands to create tables, do table operation, etc.
Creating Database Through Terminal

```sql
mysql> show databases;
+--------------------------+
| Database                 |
+--------------------------+
| information_schema       |
| IR14                    |
| cdcol                   |
| classification          |
| mysql                   |
| performance_schema      |
| phpmyadmin              |
| test                    |
+--------------------------+
8 rows in set (0.00 sec)

mysql> create database information_retrieval;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------------+
| Database                 |
+--------------------------+
| information_schema       |
| IR14                    |
| cdcol                   |
| classification          |
| mysql                   |
| performance_schema      |
| phpmyadmin              |
| test                    |
| information_retrieval    |
+--------------------------+
9 rows in set (0.00 sec)
```
Database Normalization

- **Normalization:**
  - is the process of evaluating & correcting the structures of the tables in a database
  - The goal:
    - to minimize or remove data redundancy
    - To optimize the data structure
    - Accomplished by thoroughly investigating the various data type and their relationships with one another.

- **Data redundancy:**
  - The repeat of key fields usages in other tables
Database Normalization

- Functional dependencies:
  - Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
  - are akin to a generalization of the notion of a key
  - Let $R$ be a relation and
    
    $\alpha \subseteq R$ and $\beta \subseteq R$

    The functional dependency:
    
    $\alpha \rightarrow \beta$

    holds on $R$ and only if for any tuples $t_1$ & $t_2$ that agree on the attributes $\alpha$, they also agree on the attributes $\beta$.
  - That is, $t_1[\alpha] = t_2[\alpha] \rightarrow t_1[\beta] = t_2[\beta]$
Database Normalization

- Functional dependencies

Example: consider student(Sid, Sname, DeptId) instance of student.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>DeptId</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL12001</td>
<td>JOHN</td>
<td>13</td>
</tr>
<tr>
<td>CL13050</td>
<td>WENPENG</td>
<td>13</td>
</tr>
<tr>
<td>DE10003</td>
<td>ALDI</td>
<td>15</td>
</tr>
<tr>
<td>PS11123</td>
<td>ILJA</td>
<td>11</td>
</tr>
<tr>
<td>IT09256</td>
<td>LISANDRO</td>
<td>09</td>
</tr>
<tr>
<td>CL13075</td>
<td>MATTHEW</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Is this true?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid → Sname</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sid → DeptId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sname → DeptId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sname → Sid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeptId → Sname</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeptId → Sid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Database Normalization

- Functional dependencies

Example: consider student(Sid, Sname, DeptId) instance of student.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>DeptId</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL12001</td>
<td>JOHN</td>
<td>13</td>
</tr>
<tr>
<td>CL13050</td>
<td>WENPENG</td>
<td>13</td>
</tr>
<tr>
<td>DE10003</td>
<td>ALDI</td>
<td>15</td>
</tr>
<tr>
<td>PS11123</td>
<td>ILJA</td>
<td>11</td>
</tr>
<tr>
<td>IT09256</td>
<td>LISANDRO</td>
<td>09</td>
</tr>
<tr>
<td>CL13075</td>
<td>MATTHEW</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Is this true?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid → Sname</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Sid → DeptId</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Sname → DeptId</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Sname → Sid</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>DeptId → Sname</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>DeptId → Sid</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Database Normalization

• examine the following poor database design:

![Database Design Example]

• Problems:
  – No need to repeatedly store the class time & Professor ID
  – Which one is the key?
Database Normalization

• First Normal Form (1NF):
  – A row of data cannot contain a repeating group of data.
  – Each row of data must have a unique identifier, i.e primary key

• This can be done by
  – Eliminating the repeated groups of data through creating separate tables of related data
  – Identify each set of related data with a primary key
  – All attributes are single valued (1 data type) & non-repeating

• Student information:
  - Sid  Sname  Major  Minor  IntakeYear

• Course information
  - Cid  Cname  Lid  Time  Room

• Lecturer Information
  - Lid  Lname  Ltitle
Database Normalization

- Second Normal form (2NF):
  - A table should meet 1NF
  - There must not be any partial dependency of any column on primary key (Records should not depend on anything other than a table's primary key)

- Recall our poor database design:
  Sid → Cname or Cname → time?
Database Normalization

• Second Normal Form (2NF) solution:
  - **Create** separate tables for sets of values that apply to multiple records
  - **Relates** the tables with a **foreign key**
  - **Remove** subsets of data that apply to multiple rows of a table and **place** them in separate tables enrolled

<table>
<thead>
<tr>
<th>Sid</th>
<th>Cid</th>
<th>grade (?)</th>
</tr>
</thead>
</table>

- What do we do with the attribute time, room, & Lid?
Database Normalization

- Third Normal Form (3NF):
  - Eliminate all attributes (columns) that do not directly dependent upon the primary key
  - Each non-primary key attribute must be dependent only on primary key (no transitive dependency)

- Example:

  Student:
  
  $\begin{align*}
  Sid & \quad Sname & \quad Major & \quad Minor & \quad IntakeYear \\
  \end{align*}$

  - Which attribute is not directly dependent on $Sid$?

  Student:
  
  $\begin{align*}
  Sid & \quad Sname & \quad Major & \quad Minor \\
  \end{align*}$
Database Normalization

- Old design

- New design

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
<td>Cid</td>
</tr>
<tr>
<td>Sname</td>
<td>Cname</td>
</tr>
<tr>
<td>Major</td>
<td>Lid</td>
</tr>
<tr>
<td>Minor</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Rid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sid</td>
</tr>
<tr>
<td>Cid</td>
</tr>
<tr>
<td>Grade</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lid</td>
</tr>
<tr>
<td>Lname</td>
</tr>
<tr>
<td>Ltitle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rid</td>
</tr>
<tr>
<td>Rname</td>
</tr>
<tr>
<td>BuidingId</td>
</tr>
</tbody>
</table>
Database Normalization

- Storing the relation among tables in database
Database Normalization

• Exercise:
  – Which normal form does this table violate?
  – And how do you normalize it?

<table>
<thead>
<tr>
<th>Person</th>
<th>Title</th>
<th>Author</th>
<th>Pages</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yakup</td>
<td>Database Management System</td>
<td>Ramakhrisan, Raghu</td>
<td>903</td>
<td>2010</td>
</tr>
<tr>
<td>Wenpeng</td>
<td>Beyond Human-Computer Interaction</td>
<td>Preece, Jennifer</td>
<td>889</td>
<td>2009</td>
</tr>
<tr>
<td>Amy</td>
<td>Support Your Local Wizard</td>
<td>Duane, Diane</td>
<td>473</td>
<td>1990</td>
</tr>
<tr>
<td>Dwika</td>
<td>The Hobbit</td>
<td>Tolkien, JRR</td>
<td>389</td>
<td>1995</td>
</tr>
<tr>
<td>Yadoll</td>
<td>Beyond Human-Computer Interaction</td>
<td>Preece, Jennifer</td>
<td>889</td>
<td>2009</td>
</tr>
<tr>
<td>Irina</td>
<td>Support Your Local Wizard</td>
<td>Duane, Diane</td>
<td>473</td>
<td>1990</td>
</tr>
</tbody>
</table>
RDBMS for Inverted Text Index
RDBMS & Full Text Searching

- Applying RDBMS for full text searching
  - What is the goal?
    - Creating an Inverted index consisting of:
      - Dictionary &
      - Posting list
  - What will be the entities?
    - Document
    - Term
  - How to start?
    - You need a specific algorithm, take for examples:
      - BSBI
      - SPIMI
    - What kind of information do you want to save in posting list?
      - Term – DocId only?
      - Term – DocId, TF, DF?
Database Design for BSBI

- A review on Blocked Sort-Based Indexing Algorithm

```plaintext
BSBIndexConstruction()
1   n ← 0
2    while (all documents have not been processed)
3       do n ← n + 1
4       block ← ParseNextBlock()
5       BSBI-Invert(block)
6       WriteBlockToDisk(block, f_n)
7       MergeBlocks(f_1, ..., f_n; f_merged)
```
Database Design for BSBI

• 2 core tables:
  – Document table
  – Term tables

• How do their schemas look like?
  – Doc ( did CHAR(5),
      dname CHAR(6),
      dcontent TEXT,
      PRIMARY KEY (did), UNIQUE (dname) )
  – Doc ( did INT(INC),
      dname CHAR(6),
      dcontent BLOB,
      PRIMARY KEY (did), UNIQUE (dname) )
  – What are the advantages of the first schemas compared to the second or vice versa?
Database Design for BSBI

• How do their schemas look like?
  – Term ( tid INT(INC),
    term CHAR(25),
    PRIMARY KEY (tid),
    UNIQUE (term) )

• The number of tables for posting list?
  – N-block tables + 1 merged posting table OR
  – 1 posting list table?
Database Design for BSBI

**Block 1**

<table>
<thead>
<tr>
<th>tid</th>
<th>did</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d2</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>d1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>d3</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>d4</td>
<td>150</td>
</tr>
</tbody>
</table>

**Block 2**

<table>
<thead>
<tr>
<th>tid</th>
<th>did</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d3</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>d4</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>d1</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>d2</td>
<td>82</td>
</tr>
</tbody>
</table>

**MergedPosting**

<table>
<thead>
<tr>
<th>tid</th>
<th>did</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d2</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>d3</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>d1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>d4</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>d3</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>d2</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>d4</td>
<td>150</td>
</tr>
<tr>
<td>5</td>
<td>d2</td>
<td>82</td>
</tr>
</tbody>
</table>
Database Design for BSBI

- The former table merging is right algorithmically, but it is a bad design in relational database. Why?
- There are several strategies for improving the design for the benefit of searching process.
- This strategy depends on the application you are developing
- Some strategies are:
  - Combining the use of file system & RDBMS for storing your data:
    - Block tables → file system
    - Merged posting list → RDBMS
  - Applying the relation & normalization concepts for merged posting list table
The schema for posting list may look like as follows:

- Posting( tid INT(), did CHAR(5), tf INT(5),
  INDEX (tid, did)
  FOREIGN KEY (tid, did) REFERENCES (Term, Doc) )

- Posting( tid INT(), did STRING/TEXT(),
  tf STRING/TEXT(), INDEX (tid, did)
  FOREIGN KEY (tid, did) REFERENCES (Term, Doc) )

- Posting( tid INT(), did SET(),
  tf SET(), INDEX (tid, did)
  FOREIGN KEY (tid, did) REFERENCES (Term, Doc) )
Database Design for SPIMI

- SPIMI differs from BSBI in:
  - The processing of dictionary → using Term instead of TermID-Term pair.
  - Memory allocation for posting list of a term.
  - Adding a posting directly to a posting list.

- These differences affect little to database design.

- The former database design can be applied both to BSBI & SPIMI with one difference:
  - Term ( term CHAR(25), PRIMARY KEY (term) )
  - If you have only one field/column in a table, is it worth to save your data in a RDBMS?
Exercise

• Suppose you have 3 tables in your database, the dictionary (term), document (doc), and the posting list tables.

• Suppose you will compute the weight of each term using tf-idf weighting.

• How do you design your table schema for term_weight table? How do you state its relation to other tables in your database?
References
