CZ2007
Introduction to Databases

Querying Relational Databases using SQL

Part--5

Cong Gao

Professor
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Summary and roadmap

- Introduction to SQL
- SELECT
  FROM
  WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- Constraints: primary key
- Views
- Constraints:
  - Foreign key
  - CHECK
  - ASSERTION
  - Trigger
- Next
  - Indexes

Example

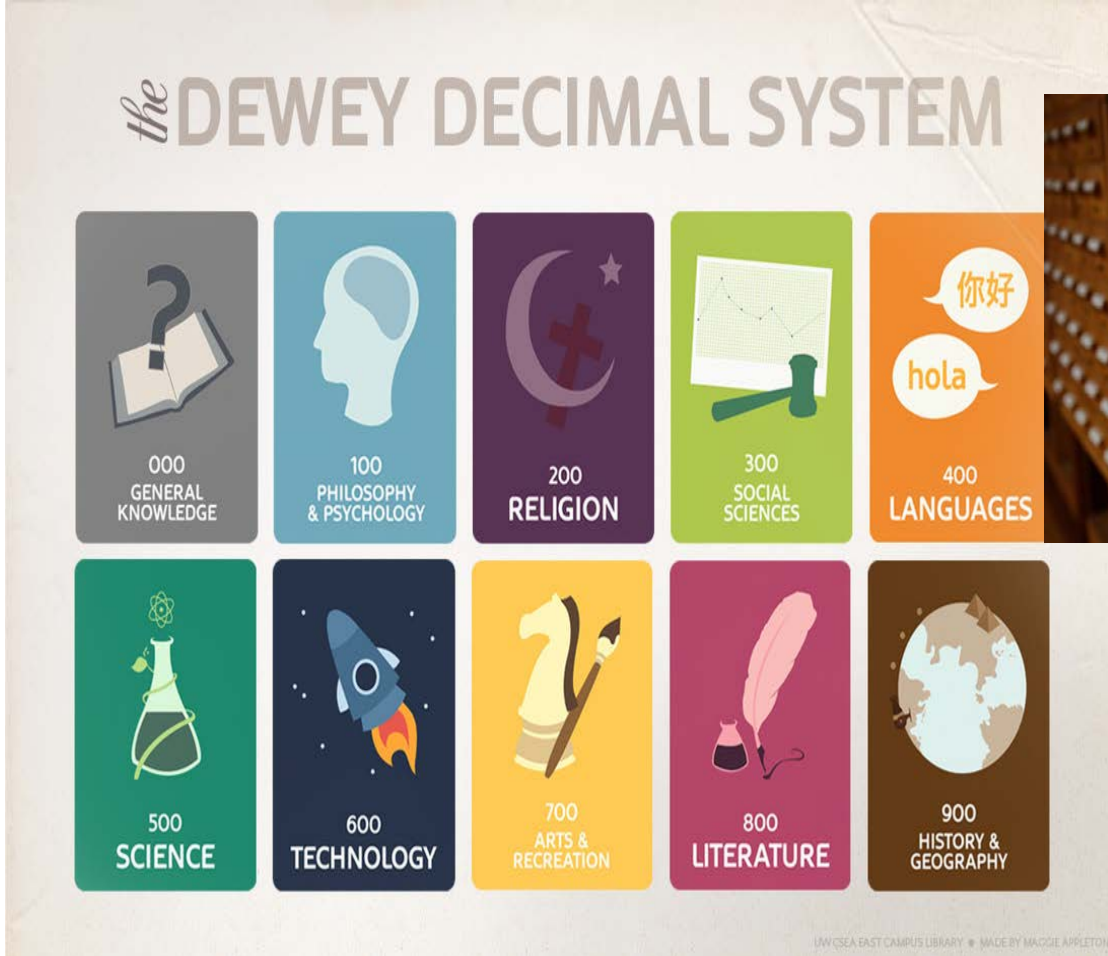
Find Book in Library

Design choices?

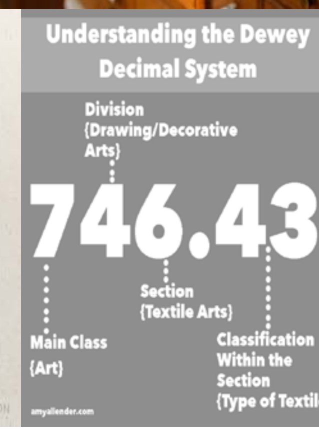
- Scan through each aisle
- Lookup pointer to book location, with librarian’s organizing scheme

Example

Find Book in Library With Index



Index Cards



Algorithm for book titles

- Find right category
- Lookup Index, find location
- Walk to aisle. Scan book titles. Faster if books are sorted

Latency numbers every engineer should know

Ballpark timings

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

(~0.25 msecs)

(~10 msecs)

(~20 msecs))

Example: Search for books

Billion_Books

BID	Title	Author	Published	Full_text
	....	....	....	
7003	Harry Potter	Rowling	1999	...
1001	War and Peace	Tolstoy	1869	...
1002	Crime and Punishment	Dostoyevsky	1866	...
1003	Anna Karenina	Tolstoy	1877	...
	....			

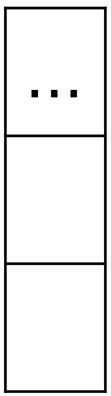
All books written by Rowling?’

```
SELECT *
FROM   Billion_Books
WHERE Author like
‘Rowling’
```

Example: Search for books

SELECT *
FROM Billion_Books
WHERE Author like 'Rowling'

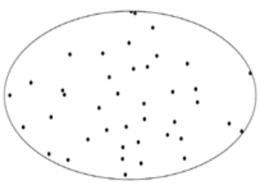
Design Choices



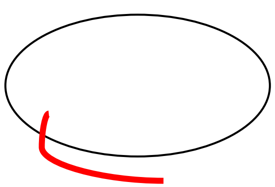
Input: Data size

1 Billion books
Each record = 1000 bytes
(i.e., 1000 GBs or 1 TB)

1. Data in RAM
    - Scan RAM sequentially & filter
        - Scan Time: 1000 GB * 0.25 msecs/1MB = 250 secs
        - Cost (@100$/16GB) ~= 6000$ of RAM



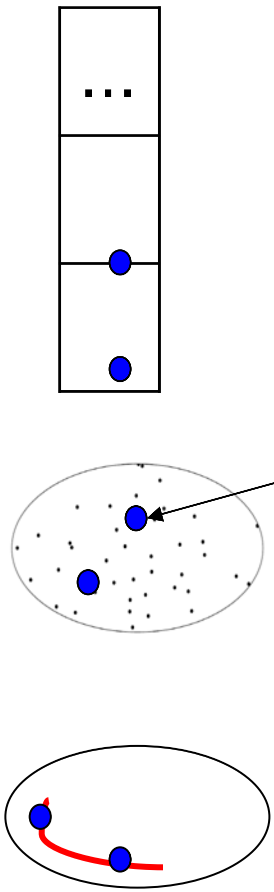
2. Data in disk (random spots)
    - Seek each record on disk & filter
        - Scan Time: (Seek) 10 msecs * 1Billion records + (Scan) 1 TB /100 MB-sec
            - = 10^7 secs (115 days) + 10^4 secs ~= 115 days
        - Cost (@100$/TB of disk) = 100$ of disk



3. Data in disk (sequentially organized)
    - Seek to table, and sequentially scan records on disk & filter
        - Scan Time: (Seek) 10 msecs + (Scan) 1 TB /100 MB-sec
            - = 10^4 secs ~= 3 hrs
        - Cost (@100$/TB of disk) = 100$ of disk

Example: Search for books

```
SELECT *
FROM   Billion_Books
WHERE Author like ‘Rowling’
```



<Disk block, position>

Index in RAM

Location	Author
	Rowling
	Tolstoy
	Rowling
	…

Index => Maintain location of record

- Memory block
- Disk block (seek positions)

Notes:

- O(n) seeks for ‘n’ results
- RAM index costs $$ but speedsup
- Or index on disk (cz4031)
- Or index on index on index….(cz4031r)

Indexes on a table

- An <u>index</u> speeds up selections on <u>search key (s)</u>
  - Any subset of fields

- Example

Books(<u>BID</u>, name, author, price, year, text)

On which attributes would you build indexes?

Example

Billion_Books

BID	Title	Author	Published	Full_text
1001	*War and Peace*	Tolstoy	1869	...
1002	*Crime and Punishment*	Dostoyevsky	1866	...
1003	*Anna Karenina*	Tolstoy	1877	...

```
SELECT *
FROM   Billion_Books
WHERE Published > 1867
```

Example

By_Yr_Index

Published	BID
1866	1002
1869	1001
1877	1003
…	

Billion_Books

BID	Title	Author	Published	Full_text
1001	*War and Peace*	Tolstoy	1869	...
1002	*Crime and Punishment*	Dostoyevsky	1866	...
1003	*Anna Karenina*	Tolstoy	1877	...
…				

Why might just keeping the table sorted by year not be good enough?

Maintain an index for this, and search over that!

Example

By_Yr_Index

Published	BID
1866	1002
1869	1001
1877	1003

Russian_Novels

BID	Title	Author	Published	Full_text
1001	*War and Peace*	Tolstoy	1869	...
1002	*Crime and Punishment*	Dostoyevsky	1866	...
1003	*Anna Karenina*	Tolstoy	1877	...

By_Author_Title_Index

Author	Title	BID
Dostoyevsky	Crime and Punishment	1002
Tolstoy	Anna Karenina	1003
Tolstoy	War and Peace	1001

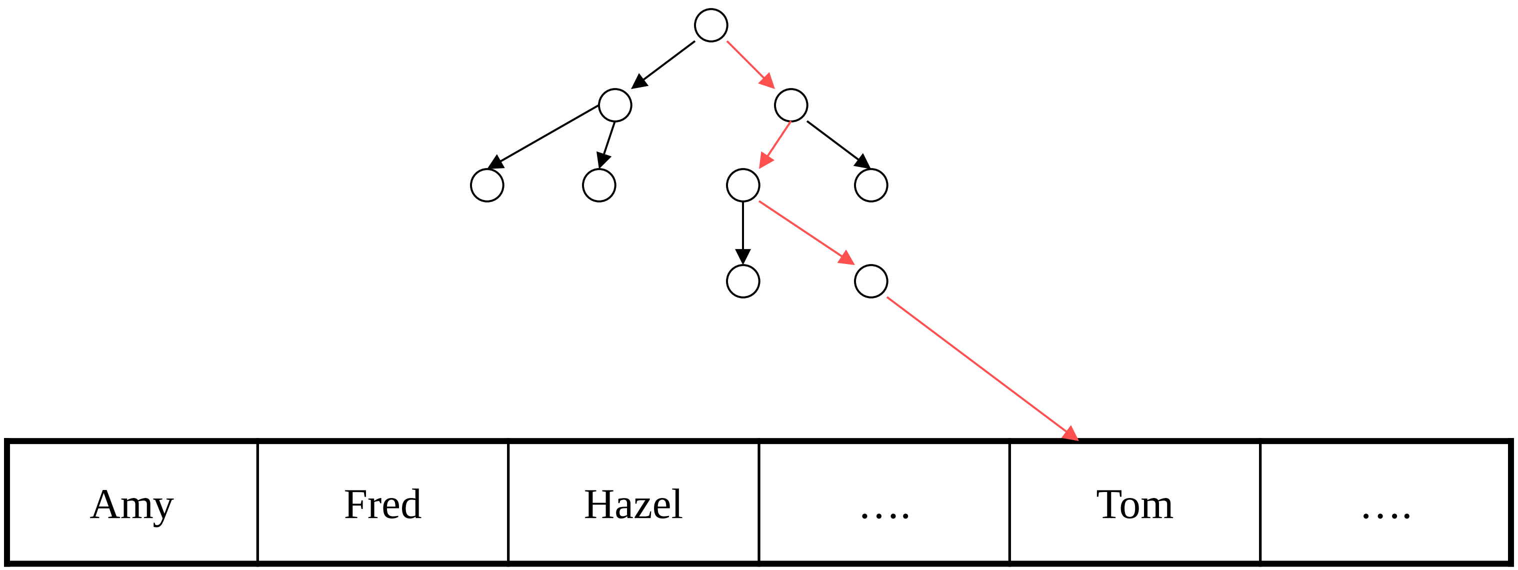
Can have multiple indexes to support multiple search keys

Indexes shown here as tables, but in reality we will use more efficient data structures…(CZ4031)

Creating Indexes in Databases

Indexes in databases

- Tree-structured (think of binary search tree)
- Hash-based



Covering Indexes

An index **covers** *for a specific query* if the index contains all the needed attributes- ***meaning the query can be answered using the index alone!***

The "needed" attributes are the union of those in the SELECT and WHERE clauses…

By_Yr_Index

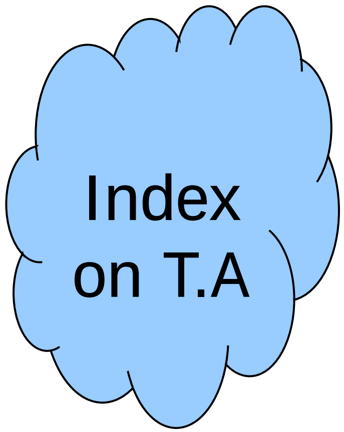
Published	BID
1866	1002
1869	1001
1877	1003

Example:

```
SELECT Published, BID
FROM   Billion_Books
WHERE Published > 1867
```

Functionality

- Used by query processor to speed up data access



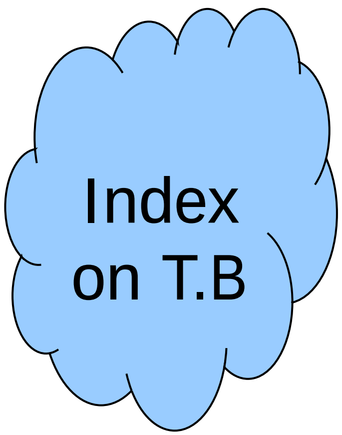
  - T.A = ‘cow’
  - T.A = ‘cat’

T

	A	B	C
1	cat	2	…
2	dog	5	…
3	cow	1	…
4	dog	9	…
5	cat	2	…
6	cat	8	…
7	cow	6	…
	…	…	…

Functionality

- Used by query processor to speed up data access



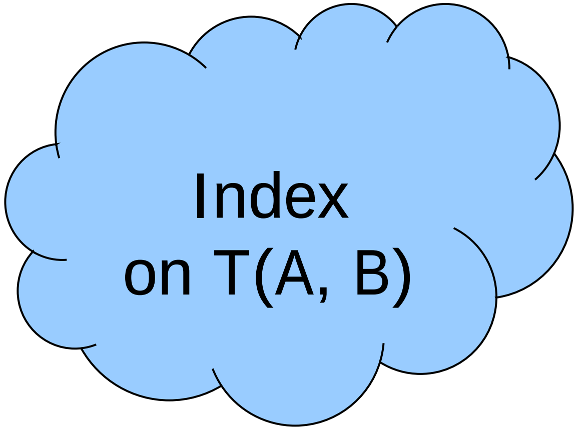
  - T.B = 2
  - T.B < 4
  - 3 <= T.B < 5

T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

Functionality

- Used by query processor to speed up data access



  – T.A = ‘cat’ and T.B = 2
  – T.A < ‘d’ and T.B < 4
  – 3 <= T.B < 5

T

	A	B	C
1	cat	2	...
2	dog	5	...
3	cow	1	...
4	dog	9	...
5	cat	2	...
6	cat	8	...
7	cow	6	...
	...	...	...

Answering Queries using Indexes

```
Select sName, cName
From Student, Apply
Where Student.sID = Apply.sID
```

- Scan Student, use an Index on Apply
- Scan Apply, use an Index on Student
- Use Indexes on both Apply and Student

Indexes (definition)

An <u>*index*</u> is a **data structure** mapping <u>search keys</u> to <u>sets of rows in table</u>

- Provides efficient lookup & retrieval by search key value (usually much faster than scanning all rows and searching)

An index can store

- full rows it points to, OR
- pointers to rows

Operations on an Index

- <u>Search</u>: Quickly find all records which meet some *condition on the search key attributes*
  - (Advanced: across rows, across tables)
- <u>Insert / Remove</u> entries
  - Bulk Load / Delete. Why?

Indexing is one of the most important features provided by a database for performance

Why Not Store Everything in Main Memory (RAM)?

- **Main memory is volatile. But** We want data to be saved.
- **Cost too much:** Main memory is much more expensive!
- **Answer is Disk**
  - Many DB related issues involve hard disk I/O!
  - Thus we will now study how a hard disk works.

Storing a Relation

Recall

- Tuples are unordered
- Focus (in SQL) is on the tuples individually

Relation table 1

E#	Salary
3	2100
1	1200
8	1900
9	1400
2	1200
4	1800
6	2300

Identical! ⟷

Relation table 2

E#	Salary
1	1200
3	2100
4	1800
2	1200
6	2300
9	1400
8	1900

$\Rightarrow$

Tuples

2	1200
4	1800
1	1200
3	2100
8	1900
9	1400
6	2300

Indexing Definition in SQL

Syntax

CREATE INDEX name ON rel (attr)

CREATE UNIQUE INDEX name ON rel (attr)
Duplicate values are not allowed

DROP INDEX name;
Note: The syntax for creating indexes varies amongst different databases.

In practice

- **PRIMARY KEY declaration:** Automatically creates a primary/clustered index
- **UNIQUE declaration:** Automatically creates a secondary/nonclustered index

Indexing Definition in SQL

- You can always specify which sets of attributes you want to build indexes
  - **Good:** Index on an attribute may speed up the execution of queries in which a value/a range of values are specified for the attribute, and may also help joins involving that attribute
  - **Bad:** it makes insertions, deletions, and updates slower

Build index on attribute list

You can build an index on multiple attributes, also called **Composite index**

- Syntax: CREATE INDEX foo ON R(A,B,C)
- Example 1:
  - CREATE INDEX PnameIndex ON FacebookUser (firstname, lastname)
- Why?

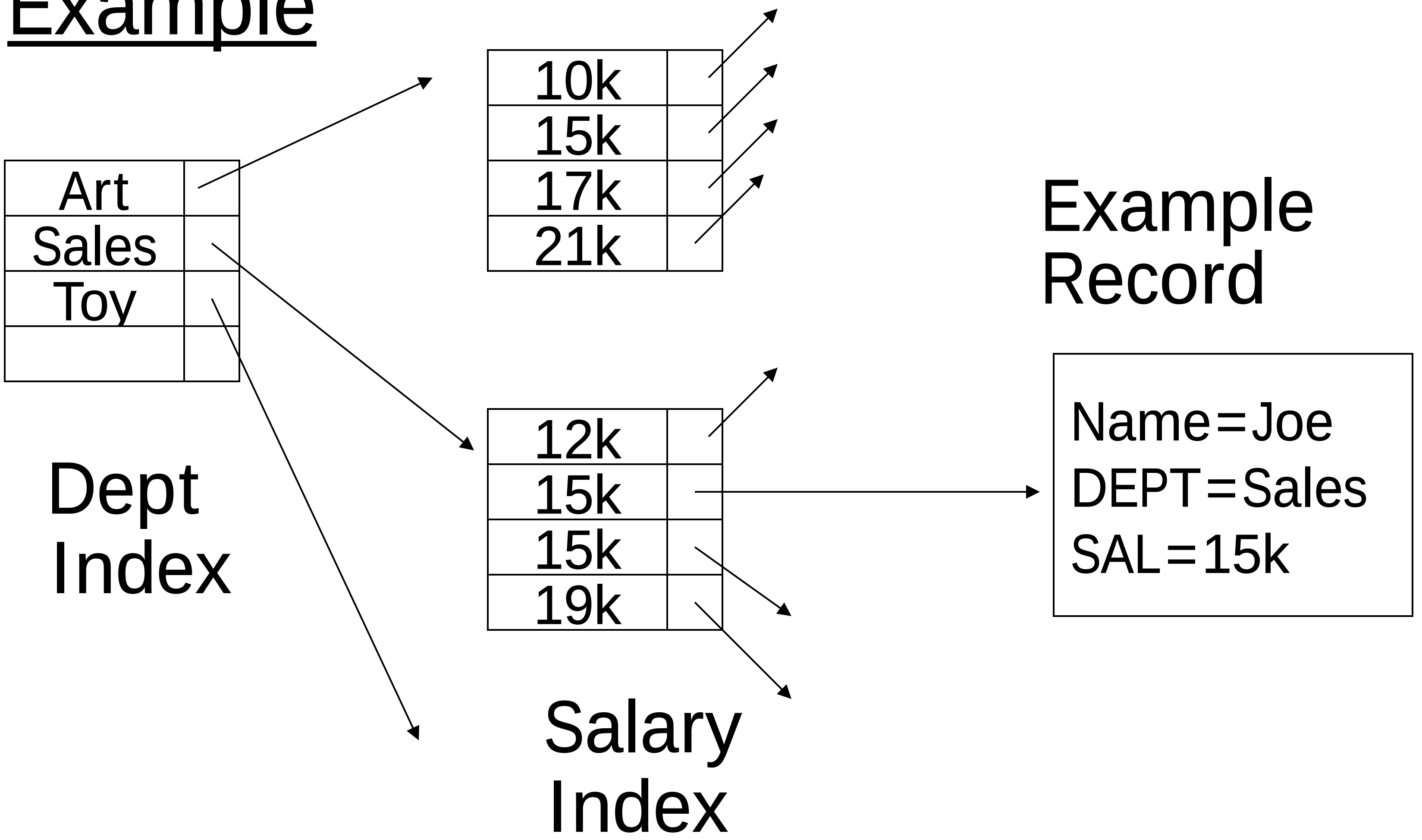
Motivation: Find records where

DEPT = “Art” AND SAL > 50k

Motivation

- Strategy I: index on single attribute
  - Use one index on Dept: Get all Dept = “Art” records and check their salary
  - Use one index on Salary: Get all Salary > 50k records and check their Dept
- Strategy 3 Composite index:
  - Create index DeptSalaryIndex on EMP (Dept, Salary)
    - See next slide
  - Create index SalaryDeptIndex on EMP (Salary, Dept)

Example



Art
Sales
Toy
Dept Index
10k
15k
17k
21k
12k
15k
15k
19k
Salary Index
Example Record
Name=Joe
DEPT=Sales
SAL=15k