NANYANG TECHNOLOGICAL UNIVERSITY

CZ2007
Introduction to Databases

Querying Relational Databases using SQL
Part-7

Arijit Khan

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

arijit.khan@ntu.edu.sg

Schedule after Recess Week

SQL

8 Lectures

- Week 8 (Oct 07-Oct 11)
- Week 9 (Oct 14-Oct 18)
- Week 10 (Oct 21-Oct 25)
- Week 11 (Oct 28-Nov 01)

Semi-Structured Data, Quiz-2

2 Lectures

- Week 12 (Nov 02-Nov 08)
- Quiz during Tutorial session
- Quiz syllabus: everything on SQL (Week 8, 9, 10 11)

Summary

- Week 13 (Nov 11-Nov 15)

Recap: Roadmap (SQL)

- Introduction to SQL
- Querying single relation

✓ Lecture-1

- Ordering Tuples
- Multi-relation queries
- Subqueries

✓ **Lecture-2**

- Set operations
- Bag semantics
- Join expressions
- Aggregation

✓ Lectures-3 & 4

Recap: Roadmap (SQL)

- Groupings
- Creation of tables
- Database modifications
- Constraints
- Views

✓

Lecture-5 & 6

Today's lecture: Chapter 13 of the Book "Database Systems: The Complete Book; Hector Garcia-Molina Jeffrey D. Ullman, Jennifer Widom

- Triggers ✓
- Indexes

Lecture-7 & 8

That would be all about Quiz-2!!

Questions?

- **Indexes**

How to Process Queries Faster?

```
SELECT      price
FROM        Sells
WHERE       supplName = ‘Apple’ AND prodName = ‘iPhone’;
```

Fact

When the relation is large, it is expensive to scan all tuples to find a few relevant ones

How Relations are Stored?

Pages

- Data files are decomposed into *pages*
- These are fixed size pieces of contiguous information in the file
- A page is the unit of exchange between disk and main memory (typical page size is 4096 bytes)

Blocks

- Disk are divided into page size *blocks* of storage
- Disk consists of a sequence of blocks

Unit of Access

Physical unit of access is always a block even if only a single bit is affected

How Relations are Stored?

Recall

- Tuples are unordered
- Focus (in relational algebra and 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

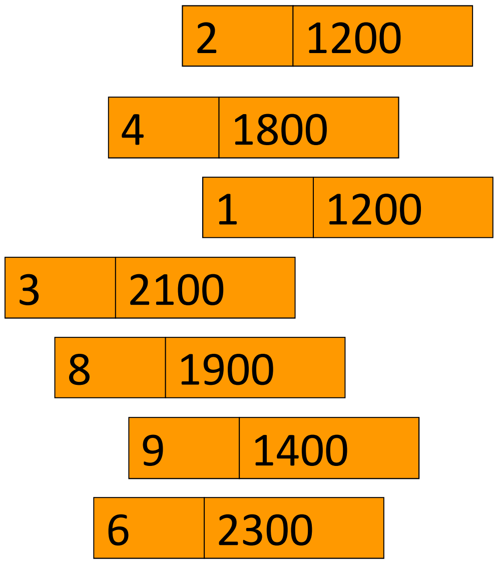
Identical! ⟷

Relation table 2

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

⇒

Tuples



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

Key Cost for Query Processing

Assumptions

- Reading a block costs one time unit
- Writing a block costs two time units *(retrieve and write back)*
- Processing in RAM is free

Fact

IO cost is important in database operations

Approach

Key to efficiency is to organize files of data records on disk so that IO costs can be minimized

Indexes

What it is?

A data structure

Input

A property of records (value of one or more fields)

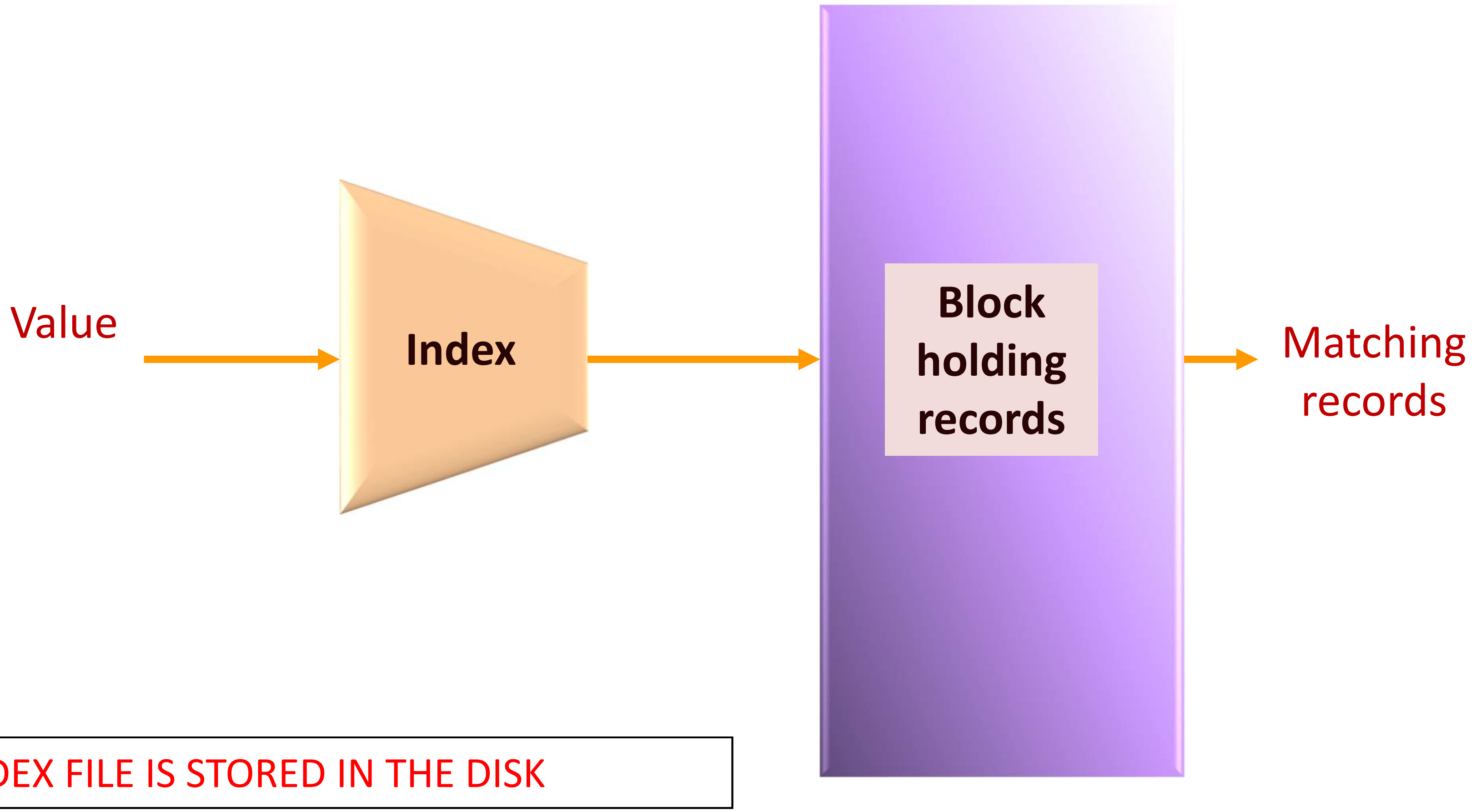
Output

Finds the records with that property "quickly"

What it does?

Index let us find records without having to look at more than a small fraction of all possible records

Indexes



Value

Index

Block holding records

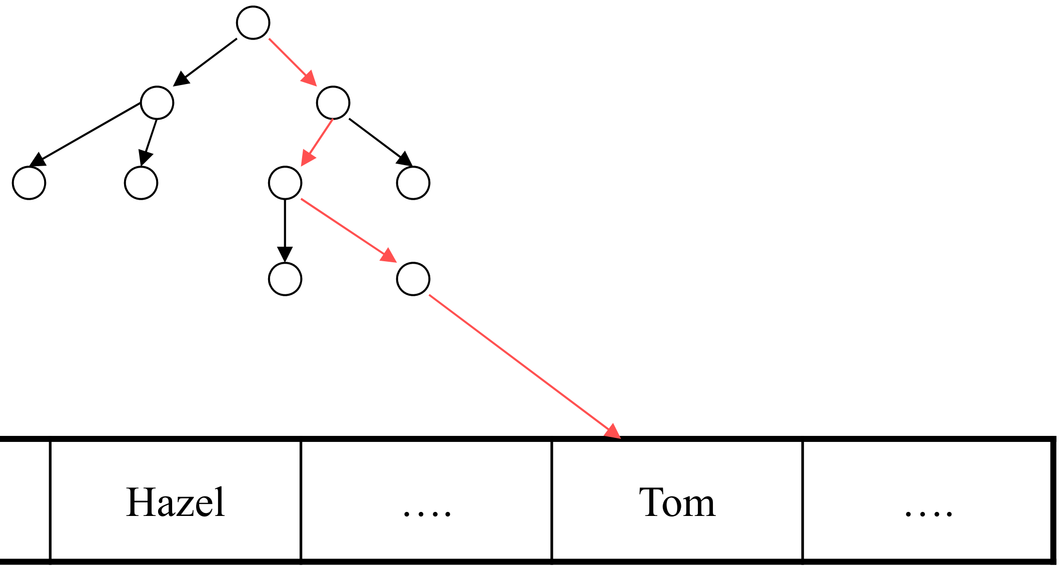
Matching records

AN INDEX FILE IS STORED IN THE DISK

Creating Indexes in Databases

Indexes in databases

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



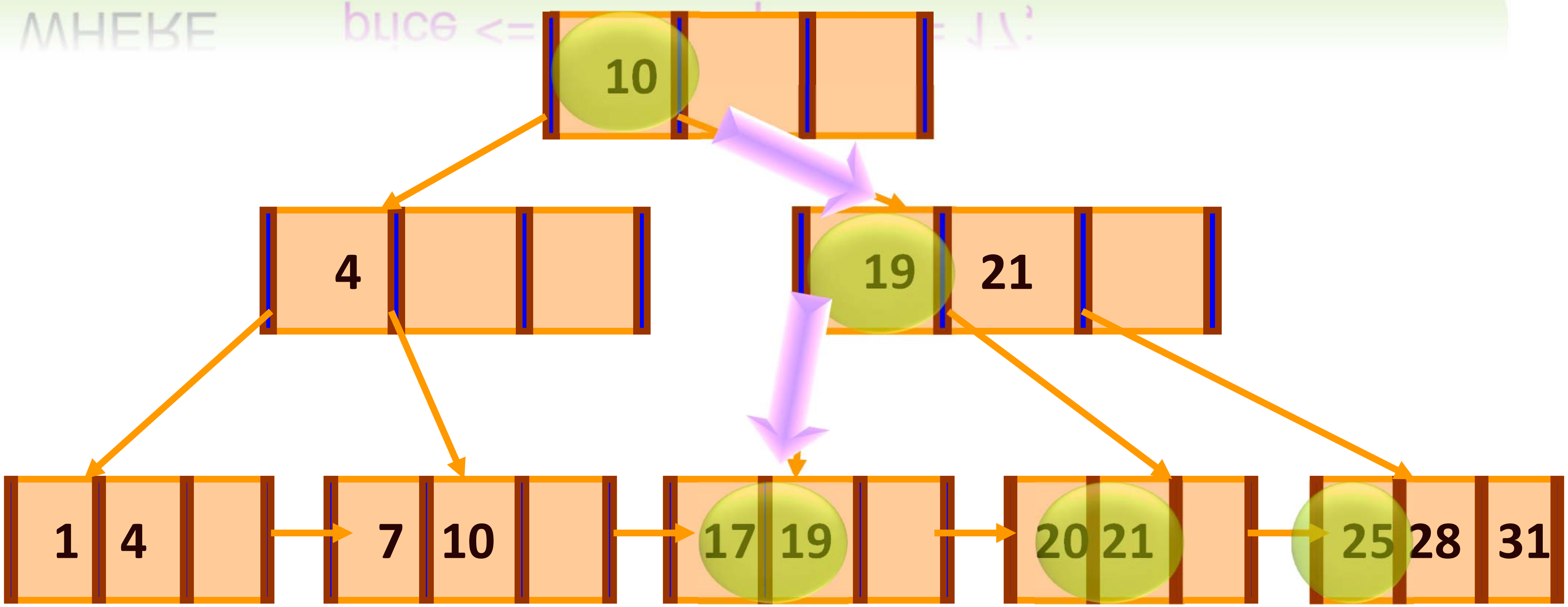
Amy	Fred	Hazel	….	Tom	….

```
CREATE INDEX  nameIndex ON Customer(name)
```

Useful for Range Queries

```
CREATE INDEX priceIndex ON  Sells (price)
```

```
SELECT      *
FROM        Sells
WHERE       price <= 26 AND price >= 17;
```



Useful for Join Queries

```
SELECT  prodName
FROM    Preferences AS P, FrequentCust AS F
WHERE supplName=`Apple’ AND F.custName = P.custName;
```

```
CREATE INDEX supplIndex ON  FrequentCust (supplName)
```

```
CREATE INDEX nameIndex ON  Preferences (custName)
```

Preferences

custName	prodName
Melissa	iPhone
Sean	iPad
………	…………
Sally	iPhone

FrequentCust

custName	supplName
Sally	Xiaomi
Sally	Apple
…….	……………
Melissa	Apple

Multi-Attribute Indexes

On multiple attributes

- Indexes can be created on more than one attribute
- Ordering matters!

Example:

```
CREATE INDEX doubleindex ON
                        Customer (age, city)
```

Helps in:

```
SELECT *
FROM   Customer
WHERE age = 55 AND city = “Singapore”
```

and even in:

```
SELECT *
FROM   Customer
WHERE age = 55
```

But not in:

```
SELECT *
FROM   Customer
WHERE city = “Singapore”
```

Pros and Cons

Pros

- Existence of an index on an attribute may greatly speed up queries in which a value, or a range of values, is specified on that attribute
- May speed up joins involving that attribute

Cons

- Makes insertion, deletions, and updates on a relation more complex and time consuming

Questions ??

Thank You !