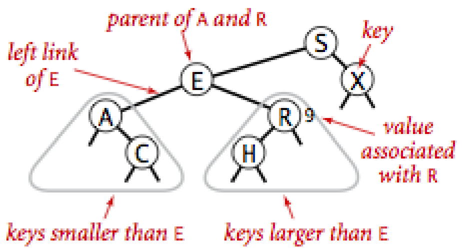
Indexes in SQL

Material covered from Chapter 8, A First Course in Database Systems by Jenifer and Garcia &

Stanford course of Database Management and Data Systems– CS145

Indexes in SQL



Anatomy of a binary search tree

An *index* on an attribute A of a relation is a data structure that makes it efficient to find those tuples that have a fixed value for attribute A.

- Data structure
- Helps in searching
- Fundamental unit of DB performance

Note:

Attributes of the index will be referred as *index key*.

Example:

Binary Search Tree (BST) – (key, value)

One of the values that attribute A may have

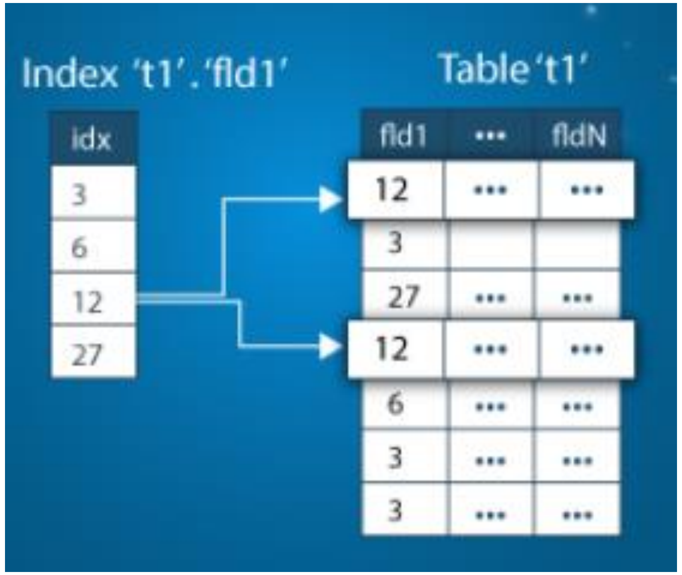
The set of locations of the tuples that have a in the component for attribute A

Definition taken from the book Fundamentals of Database System by Elmasri

Image: https://algs4.cs.princeton.edu/32bst/#:~:text=Definition.,in%20that%20node's%20right%20subtree.

Indexes in SQL

- Maps search keys to set of rows in table
- Efficient lookup and retrieval by search key
- Index can be created on any subset of attributes. Advanced: across rows, tables
- Operations: Insert, Delete, Lookup
- An index can store:
  - Full rows it points to (primary index), OR
  - Pointer to rows (secondary index)



Slide is taken from cs145 - Stanford Fall 2019 by Shiva Kumar
Image: https://www.edureka.co/blog/index-in-sql/

Why study Indexes?

Opaque to data store

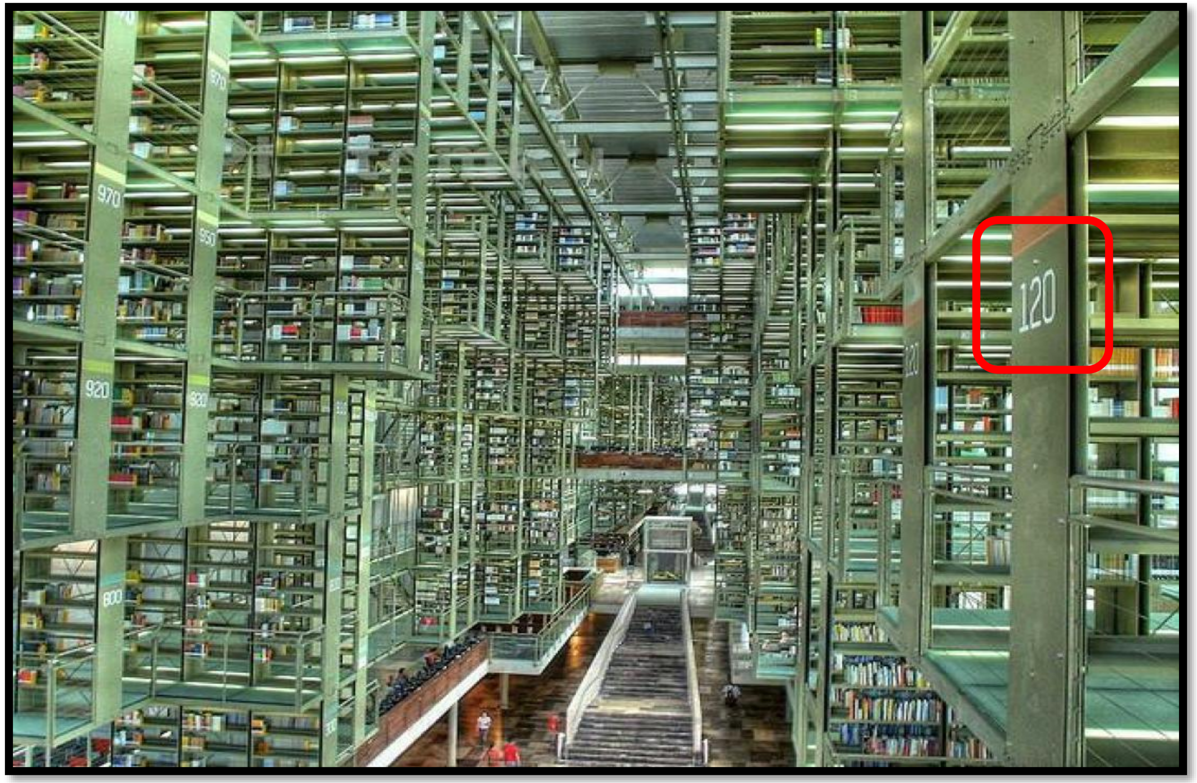
Key	Value
AAAAA	1101001111010100110101111...
AABAB	1001100001011001101011110...
DFA766	0000000000101010110101010...
FABCC4	1110110110101010100101101...

- Large relations – expensive to scan
- Core indexing ideas have become stand-alone systems
    - E.g., search in google.com
    - Data blobs* in noSQL, Key-value stores [1][4]
    - *A Binary Large Object is a collection of binary data stored as a single entity in a DBMS. They are typically images, audio and other multimedia objects, sometimes binary executable code. Database support of blobs is not universal. [2]

- **References:**
    - [1] https://www.mongodb.com/nosql-explained
    - [2] https://en.wikipedia.org/wiki/Binary_large_object
    - [3] https://www.pluralsight.com/blog/software-development/relational-non-relational-databases
    - [4] https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/data-store-overview
    - [5] https://azure.microsoft.com/en-us/services/storage/blobs/

Some text on this slide is taken from cs145 - Stanford Fall 2019 by Shiva Kumar
Image: [4]

Example – Find book in Library

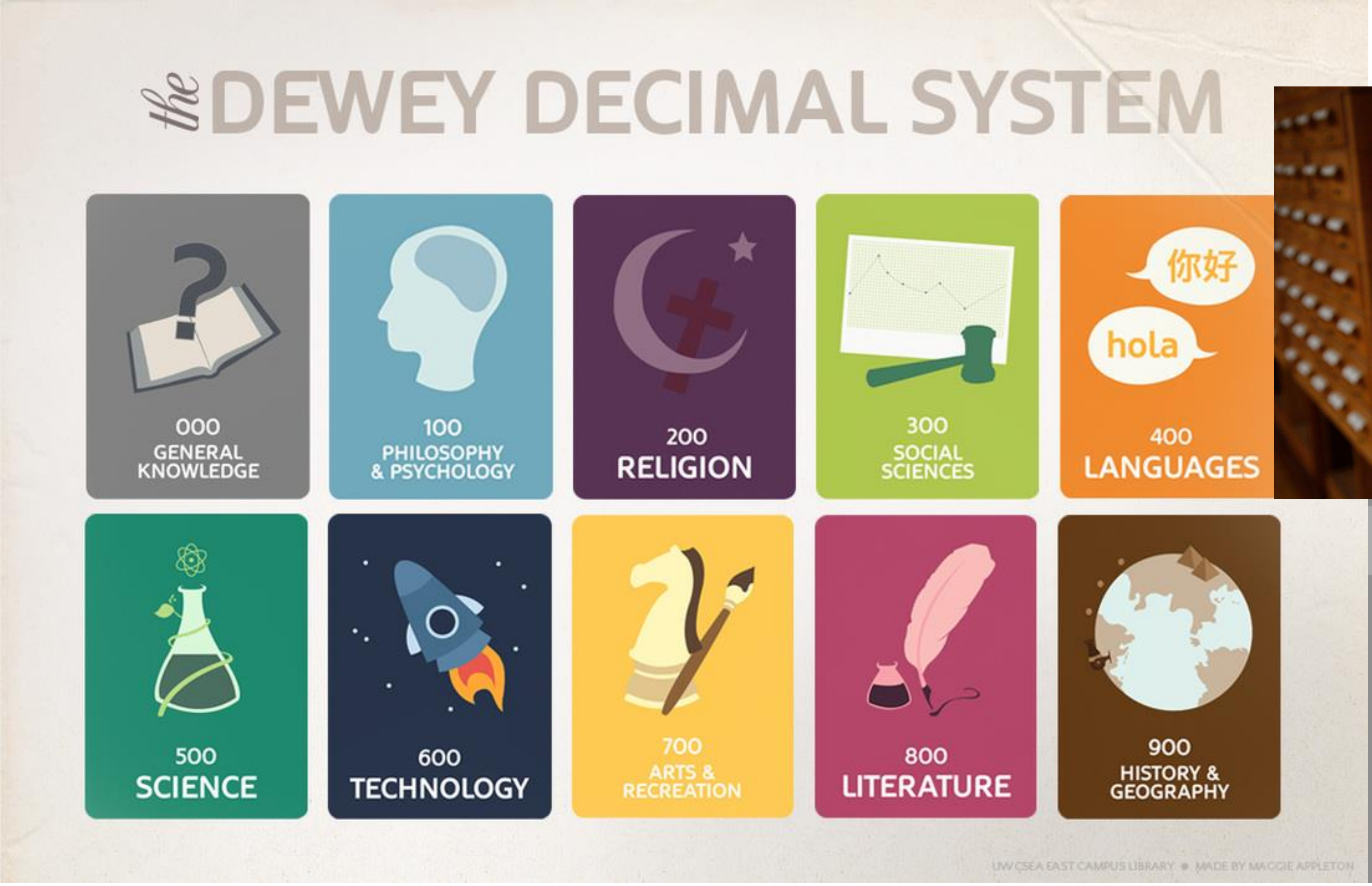


Design choices?

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

Slide is taken from cs145 - Stanford Fall 2019 by Shiva Kumar

Find book in Library with Index



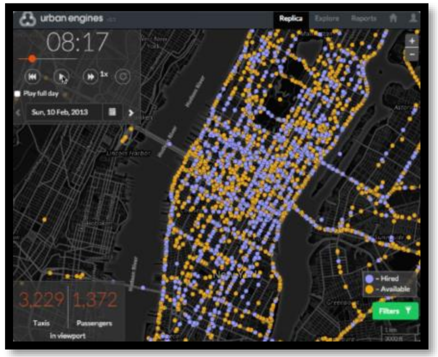
Index Cards

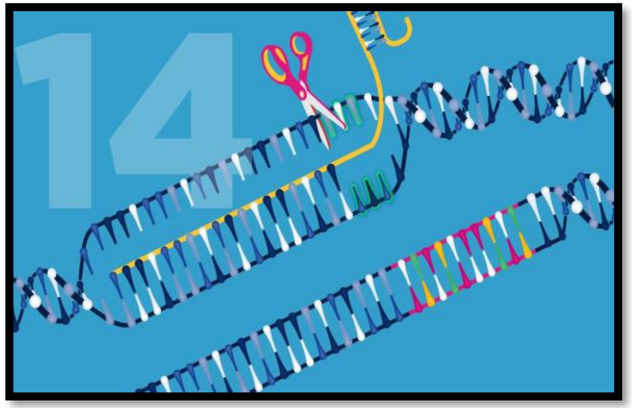
600	Technology
630	Agriculture and related technologies
636	Animal husbandry
636.7	Dogs
636.8	Cats

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

Slide is taken from cs145 - Stanford Fall 2019 by Shiva Kumar
Dewey Decimal System:
https://www.oclc.org/content/dam/oclc/dewey/resources/summaries/deweysummaries.pdf

Kinds of **Indexes** (different **data ty**pes)





Index for

- Strings, Integers
- Time series, GPS traces, Genomes, Video sequences
- Advanced: Equality vs Similarity [1], Ranges [2], Subsequences

Composites of above

- **References:**

[1] https://books.google.com.pk/books?id=-VrlBwAAQBAJ&pg=PA298&lpg=PA298&dq=equality+vs+similarity+in+dbms&source=bl&ots=ODlOVOHsqc&sig=ACfU3U0BdqwyUI9eu1pBK1Ki8v7b6rzy0Q&hl=en&sa=X&ved=2ahUKEwjOrdLbhvLpAhUHTcAKHYOUCrEQ6AEwAHoECAgQAQ#v=onepage&q=equality%20vs%20similarity%20in%20dbms&f=false
[2] https://use-the-index-luke.com/sql/where-clause/searching-for-ranges/greater-less-between-tuning-sql-access-filter-predicates

Example: Search on stocks

Company(CName, StockPrice, Date, Country)

Company			
CName	Date	Price	Country
AAPL	Oct1	101.23	USA
AAPL	Oct2	102.25	USA
AAPL	Oct3	101.6	USA
GOOG	Oct1	201.8	USA
GOOG	Oct2	201.61	USA
GOOG	Oct3	202.13	USA
Alibaba	Oct1	407.45	China
Alibaba	Oct2	400.23	China

```
SELECT *
FROM   Company
WHERE CName like 'AAPL'
```

```
SELECT CName, Date
FROM   Company
WHERE Price > 200
```

Q: On which attributes would you build indexes?

A: On as many subsets as you'd like. Look at query workloads.

Example

CName_Index

CName	Block #
AAPL	
AAPL	
AAPL	
GOOG	
GOOG	
GOOG	
Alibaba	
Alibaba	

Block #

Company			
CName	Date	Price	Country
AAPL	Oct1	101.23	USA
AAPL	Oct2	102.25	USA
AAPL	Oct3	101.6	USA
GOOG	Oct1	201.8	USA
GOOG	Oct2	201.61	USA
GOOG	Oct3	202.13	USA
Alibaba	Oct1	407.45	China
Alibaba	Oct2	400.23	China

PriceDate_Index

Date	Price	Block #
Oct1	101.23	
Oct2	102.25	
Oct3	101.6	
Oct1	201.8	
Oct2	201.61	
Oct3	202.13	
Oct1	407.45	
Oct2	400.23	

1. Index contains search key + Block #: e.g., DB block number.
    - In general, "pointer" to where the record is stored (e.g., RAM page, DB block number or even machine + DB block)
    - Index is conceptually a table. In practice, implemented very efficiently (see how soon)

2. Can have multiple indexes to support multiple search keys

Example with single relation with non-clustered index

Example:

Instructor(id, name, startdate, enddate, dept_name)

SQL:

SELECT name AS InstructorName

FROM Instructor

WHERE dept_name = ‘Computer Engineering’

Example with single relation with clustered index

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

SQL:

SELECT p.name AS ProductName

FROM PRODUCT

WHERE exp_date < '20200412'

Example with single relation with non-clustered index

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

SQL:

```
SELECT p.name AS ProductName
FROM PRODUCT
WHERE exp_date < '20200412'
```

Queries with JOINs

SQL:

```
SELECT p.name AS ProductName, c.name AS Category
FROM PRODUCT p, CATEGORY c
WHERE YEAR(exp_date) < 2020 AND c.name = 'Hardware'
```

Queries with JOINs

- Query optimizer has following possible options
  - Nested join (Just like nested for loops – n*m times tuples matching)
  - Merge join (Sorted relations – 1 to 1 mapping)
  - Hash join (1 hash table and other relation is used to probe it)

<u>SQL:</u>

```
SELECT p.name AS ProductName, c.name AS Category
FROM PRODUCT p, CATEGORY c
WHERE YEAR(exp_date) < 2020 AND c.name = ‘Hardware’
```


- **<u>References:</u>**
  - [1] https://www.itprotoday.com/sql-server/indexes-and-joins
  - [2] https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15
  - [3] http://logicalread.com/oracle-11g-hash-joins-mc02/#.XuDIXDozZPY

Example with JOINs

```
SELECT p.name AS ProductName, c.name AS Category
FROM PRODUCT p JOIN CATEGORY c
ON p.category_id = c.id
WHERE YEAR(exp_date) < 2020 AND c.name = ‘Hardware’
```

Nested Join:

- If one join input much smaller than the other
- Larger input has clustered index on the join attribute

Merge Join:

- Both relations have clustered indexes on the join attribute.

Hash Join:

- Smaller relation – Hash table
- Larger relation – probes smaller with search key.
- Faster than merge join – 1 table sorted.

- **References:**

  - [1] https://www.itprotoday.com/sql-server/indexes-and-joins
  - [2] https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15
  - [3] http://logicalread.com/oracle-11g-hash-joins-mc02/#.XuDIXDozZPY

Syntax (ADD)

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description, parent_cid)

SQL:

```
CREATE INDEX CategoryIndex ON CATEGORY(cname)
CREATE CLUSTERED INDEX ExpDateIndex ON Product (exp_date)
```

```
SELECT p.name AS ProductName, c.name AS Category
FROM PRODUCT p, CATEGORY c
WHERE YEAR(exp_date) < 2020 AND c.name = 'Hardware'
```

Syntax (DROP)

<u>Example:</u>

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

<u>SQL:</u>

CREATE INDEX ExpDateIndex ON Product (exp_date)

DROP INDEX ExpDateIndex

Index on multiple attributes/Composite Index

Example:

CLOTHES(pid, name, price, brand, npieces, comments)

SQL:

CREATE INDEX brandPriceIndex ON Product (brand, price)

Query workload:

SELECT name, price

FROM CLOTHES

WHERE brand = ‘X’ AND price < 5000

Why order of multi-attribute index matters?

- *If the key for the multiattribute index is really the concatenation of the attributes in some order, then we can even use this index to find all the tuples with a given value in the first of the attributes.*

Example:

EMPLOYEE (emp_id, fname, lname, age, gender, m_id, dpt_id)

Given:

Index on lname, fname (see order)

SQL:

Q1: Select emp_id, m_id FROM EMPLOYEE WHERE lname = 'Gregory'

Q2: Select emp_id, m_id FROM EMPLOYEE WHERE fname = 'Billi'

- **References:**

  - [1] https://dba.stackexchange.com/questions/30019/usefulness-of-a-multi-attribute-index

Covering Indexes

PriceDate_Index

Date	Price	Block #
Oct1	101.23	
Oct2	102.25	
Oct3	101.6	
Oct1	201.8	
Oct2	201.61	
Oct3	202.13	
Oct1	407.45	
Oct2	400.23	

An index covers *for a specific query* if the index contains all the needed attributes

I.e, *query can be answered using the index alone!*

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

Example:

```
SELECT Date, Price
FROM   Company
WHERE Price > 101
```

Slide is taken from Stanford Fall 19 course by Shiva Kumar

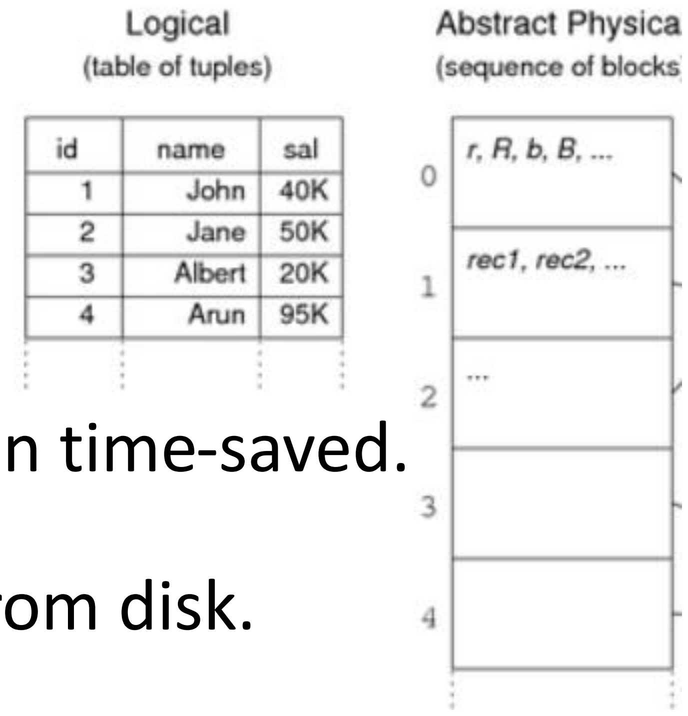
How to select Index?

- Existence of index – **Speed up queries** and maybe **joins** as well
- Every index makes insertion, deletions and updates to a relation **more complex** and **time-consuming**

Why?

Cost Model

- Tuples are distributed among many pages
- Pages are memory blocks. One page can be of several thousand bytes!
- A page can hold many tuples
- To examine 1 tuple, whole page is brough into main memory (MM)
- If the page you want is in main memory than time-saved.
- Otherwise, page will be brought into MM from disk.



Usefulness of a Key Index

- Key index – if their values are used frequently in searching (queries)
- For one value of key index, only one or no tuple will be retrieved
- Other pages for indexes will also be retrieved

Example:

STUDENT(**reg_no**, name, age, gender, bdate, matric_marks)

SQL:

```
SELECT matric_marks, name
FROM STUDENT
WHERE reg_no = '2016-CE-8'
```

Usefulness of a Key Index

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

SQL:

```
SELECT SUM(p.price)/COUNT(p.pid) AS AvgPrice, c.name AS Category
FROM PRODUCT p JOIN CATEGORY c
ON p.category_id = c.cid
GROUP BY c.category_name
```

Issue:

- Read all pages of PRODUCT and CATEGORY each!
- Too numerous to fit in MM. Several pings to disk!

Usefulness of a non-key Index*

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

SQL:

```
SELECT p.name
FROM PRODUCT p, CATEGORY c
WHERE YEAR(exp_date) < 2020 AND c.name = 'Bakery'
```

Resolution:

- Index: Product Expiry date – few pages into MM
- Index: Category Name – few pages into MM

Usefulness of a non-Key Index

It can be effective in following cases:

- If attribute is almost a key, relatively few tuples have a given value for that attribute

- If tuples are clustered (grouping the tuples) on that attribute

Usefulness of a non-Key Index

If attribute is almost a key, relatively few tuples have a given value for that attribute

Example:

STUDENT(sid, name, age, gender, gpa, degree, b_date)

Index: STUDENT(name)

SQL:

SELECT age

FROM STUDENT s

WHERE s.name = ‘Ali’

Usefulness of a non-Key Index

- If tuples are clustered (grouping the tuples) on that attribute

Example:

PRODUCT(pid, name, price, length, width, cateogory_id, m_date, exp_date)

CATEGORY(cid, name, description)

Index: PRODUCT(m_date)

SQL:

```
SELECT p.name
FROM PRODUCT
WHERE YEAR(p.m_date) = 2018
```

Find Best Index to Create

- Relation is saved on many disk pages.
- Each change in R forces us to change any index on one or more of the modified attributes of R (Still an efficiency gain)
- Reading and writing on R also requires reading and writing certain pages that hold indexes
- Modifications are twice as expensive as accessing (one disk access for read and another for write)

How to select index?

- Find cost of query or modifications to relations
- Find cost of accessing and modifying indexes

Winner:

Lower cost

Find Best Index to Create

- Assumptions on expected queries/application
- Observe history of queries
- Can use variables instead of constants
- Brute force and Greedy approach

Find Best Index to Create

StarsIn(movieTitle, movieYear, starName)

Q1:

SELECT movieTitle, movieYear

FROM StarsIn

WHERE starName = s

Q2:

SELECT starName

FROM StarsIn

WHERE movieTitle = t AND movieYear = y

I:

INSERT INTO StarsIn

VALUES(t, y, s)

Assumptions:

- **StarsIn** occupies 10 pages. Cost = 10 for accessing entire relation
- On average, a star has appeared in 3 movies and a movie has 3 stars.
- If no index on star or movie, then 10 disk accesses are required.
- One disk access will be used to read index and one for write/modifying index.
- If we have index on starName or on the combination of movieYear and movieTitle then it will take only 3 disk accesses to find the (average of) 3 tuples, for a star or a movie.
- Insertion: 1 disk access to read a page where tuple will be placed, 1 for write.

Cost Calculation

StarsIn(movieTitle, movieYear, starName)

I:

INSERT INTO StarsIn

VALUES(t, y, s)

*1 for accessing index and 3 for finding 3 pages to access three tuples

Action	No Index	Star Index	Movie Index	Both Indexes
Q1	10	4*	10	4
Q2	10	10	4	4
I	2	4	4	6
Average	?	?	?	?

Q1:

SELECT movieTitle, movieYear

FROM StarsIn

WHERE starName = s

Q2:

SELECT starName

FROM StarsIn

WHERE movieTitle = t AND movieYear = y

Average Cost Calculation

Lets make an assumption that fraction of the time we do Q1 is **p1** and for Q2 is **p2** and therefore, the fraction of time for I is **1 – p1 – p2**.

No Index:

10p1 + 10 p2 + 2(1 – p1 – p2) = **2 + 8p1 + 8p2**

Star Index:

4p1 + 10p2 + 4(1 – p1 – p2) = **4 + 6p2**

Movie Index:

10p1 + 4p2 + 4(1 – p1 – p2) = **4 + 6p1**

Both Index:

4p1 + 4p2 + 2(1 – p1 – p2) = **6 - 2p1 - 2p2**

Cost Calculation

StarsIn(movieTitle, movieYear, starName)

I:

INSERT INTO StarsIn

VALUES(t, y, s)

*1 for accessing index and 3 for finding 3 pages to access three tuples

Action	No Index	Star Index	Movie Index	Both Indexes
Q1	10	4*	10	4
Q2	10	10	4	4
I	2	4	4	6
Average	2 + 8p1 + 8p2	4 + 6p2	4 + 6p1	6 – 2p1 – 2p2

Q1:

SELECT movieTitle, movieYear

FROM StarsIn

WHERE starName = s

Q2:

SELECT starName

FROM StarsIn

WHERE movieTitle = t AND movieYear = y

Which Index to Create?

Depends on p1 and p2.

For example: if p1 = 0.1, p2 = 0.1:

(Most of the time queries are of the insertion type as we do I, 1 – 0.1 – 0.1 = 0.8 fraction of the time)

No Index:

2 + 8p1 + 8p2 = 2 + 8 (0.1) + 8 (0.1) = **3.6**

Star Index:

4 + 6p2 = 4 + 6(0.1) = **4.6**

Movie Index:

4 + 6p1 = 4 + 6(0.1) = **4.6**

Both Index:

6 - 2p1 - 2p2 = 6 – 2(0.1) – 2(0.1) = **5.6**

Which Index to Create?

Depends on p1 and p2.

For example: if p1 = 0.4, p2 = 0.4:

(Most of the time queries are of the access type as we do I, 1 – 0.4 – 0.4 = 0.2 fraction of the time)

No Index:

2 + 8p1 + 8p2 = 2 + 8 (0.4) + 8 (0.4) = **8.4**

Star Index:

4 + 6p2 = 4 + 6(0.4) = **6.4**

Movie Index:

4 + 6p1 = 4 + 6(0.4) = **6.4**

Both Index:

6 - 2p1 - 2p2 = 6 – 2(0.4) – 2(0.4) = **4.4**

Automatic Selection of Indexes

- **Database tuning:** Index selection, number of backups, amount of main memory allocation to various processes and other parameter setting
- Tools available: Tune system itself (AutoAdmin at Microsoft and SMART at IBM)
- Establish query workload/observe DBMS query logs/Application programs that use database
- The designer may be offered the opportunity to specify some constraints e.g., indexes that must, or must not, be chosen
- The tuning advisor generates a set of possible candidate indexes and evaluate each one.
- The index set resulting in the lower cost for the given workload is suggested to the designer, or it is automatically created.