Database Indexes

Quiz!

How costly is this operation (naive solution)?

course	*per*	*weekday*	*hour*	*room*
TDA356	2	VR	Monday	13:15
TDA356	2	VR	Thursday	08:00
TDA356	4	HB1	Tuesday	08:00
TDA356	4	HB1	Friday	13:15
TIN090	1	HC1	Wednesday	08:00
TIN090	1	HA3	Thursday	13:15

n

```
SELECT *
FROM   Lectures
WHERE  course = 'TDA356'
       AND period = 2;
```

Go through all *n* rows, compare with the values for course and period = 2*n* comparisons

Quiz!

Can you think of a way to make it faster?

```
SELECT *
FROM   Lectures
WHERE  course = 'TDA356'
       AND period = 2;
```

If rows were stored sorted according to the values course and period, we could get all rows with the given values faster (O(log n) for tree structure).

Storing rows sorted is expensive, but we can use an *index* that given values of these attributes points out all sought rows (an index could be a hash map, giving O(1) complexity to lookups).

Index

- When relations are large, scanning all rows to find matching tuples becomes very expensive.
- 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.
  - Example: a hash table gives amortized O(1) lookups.

Quiz!

Asymptotic complexity (O(x) notation) is misleading here. Why?

The asymptotic complexity works for data structures in main memory. But when working with stored persistent data, the running time of the data structure, once in main memory, is negligible compared to the time it takes to read data from disk. What really matters to get fast lookups in a database is to minimize the number of disk blocks accessed (could use asymptotic complexity over disk block accessing though).

Indexes help here too though. If a relation is stored over a number of disk blocks, knowing in which of these to look is helpful.

Typical costs

- Some typical costs of disk accessing for database operations on a relation stored over n blocks:
  - Query the full relation: n (disk operations)
  - Query with the help of index: k, where k is the number of blocks pointed to (1 for key).
  - Access index: 1
  - Insert new value: 2 (one read, one write)
  - Update index: 2 (one read, one write)

Example:

```
SELECT *
FROM   Lectures
WHERE  course = 'TDA356'
       AND period = 2;
```

Assume Lectures is stored in *n* disk blocks. With no index to help the lookup, we must look at all rows, which means looking in all *n* disk blocks for a total cost of *n*.

With an index, we find that there are 2 rows with the correct values for the course and period attributes. These are stored in two different blocks, so the total cost is 3 (2 blocks + reading index).

Quiz!

How costly is this operation?

```
SELECT *
FROM   Lectures, Courses
WHERE  course = code;
```

Lectures: n disk blocks

Courses: m disk blocks

No index:
Go through all *n* blocks in Lectures, compare the value for course from each row with the values for code in all rows of Courses, stored in all *m* blocks. The total cost is thus ***n * m*** accessed disk blocks.

Index on code in Courses:
Go through all *n* blocks in Lectures, compare the value for course from each row with the index. Since course is a key, each value will exist at most once, so the cost is ***2 * n + 1*** accessed disk blocks (1 for fetching the index once).

CREATE INDEX

- Most DBMS support the statement

```
CREATE INDEX index name
  ON table (attributes);
```

  – Example:

```
CREATE INDEX courseIndex
  ON Courses (code);
```

  – Statement not in the SQL standard, but most DBMS support it anyway.
  – Primary keys are given indexes implicitly (by the SQL standard).

Important properties

- Indexes are separate data stored by itself.
  - Can be created
    - ✓ on newly created relations
    - ✓ on existing relations
      - will take a long time on large relations.
  - Can be dropped without deleting any table data.
- SQL statements do not have to be changed
  – a DBMS automatically uses any indexes.

Quiz!

Why don't we have indexes on all attributes for faster lookups?

  – Indexes require disk space.
  – Modifications of tables are more expensive.
    - Need to update both table and index.
  – Not always useful
    - The table is very small.
    - We don't perform lookups over it (Note: lookups ≠ queries).
  – Using an index costs extra disk block accesses.

Rule of thumb

- Mostly queries on tables – use indexes for key attributes.
- Mostly updates – be careful with indexes!

Quiz!

Assume we have an index on Lectures for (course, period, weekday) which is the key. How costly are these queries?

Lectures: n disk blocks

```
SELECT *                    SELECT *
FROM   Lectures             FROM   Lectures
WHERE  course = 'TDA356'    WHERE  weekday = 'Monday'
       AND period = 2;             AND room = 'VR';
```

A multi-attribute index is typically organized hierarchically. First the rows are indexed according to the first attribute, then according to the second within each group, and so on.
Thus the ***left*** query costs at most ***k + 1*** where k is the number of rows matching the values. The ***right*** query can't use the index, and thus costs ***n***, where n is the size of the relation in disk blocks.

Example: Suppose that the Lectures relation is stored in 20 disk blocks, and that we typically perform three operations on this table:

- insert new lectures (Ins)
- list all lectures of a particular course (Q1)
- list all lectures in a given room (Q2)

Let's assume that in an average week there are:

- 2 lectures for each course, and
- 10 lectures in each room.

Let's also assume that

- each course has lectures stored in 2 blocks, and
- each room has lectures stored in 7 (some lectures are stored in the same block).

Example continued:

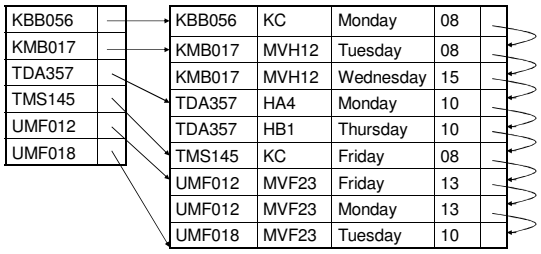
Insert new lectures (Ins)
List all lectures of a particular course (Q1)
List all lectures in a given room (Q2)

Indexes	No index	Index for (course, period, weekday)	Index for room	Both indexes
Ins	2	4	4	6
Q1	20	3	20	3
Q2	20	20	8	8
cost	$2 + 18p_1 + 18p_2$	$4 - p_1 + 16p_2$	$4 + 16p_1 + 4p_2$	$6 - 3p_1 + 2p_2$

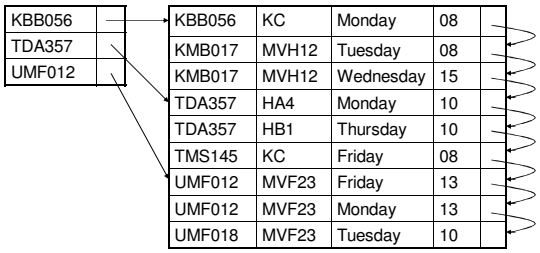
The amortized cost depends on the distribution of the operations. p_1 is proportion of operations that are Q_1 queries, p_2 similarly for Q_2, and thus the proportion of operations that are Ins modifications is $1 - p_1 - p_2$. For some different values of p_1 and p_2 we get actual costs of:

	$2 + 18p_1 + 18p_2$	$4 - p_1 + 16p_2$	$4 + 16p_1 + 4p_2$	$6 - 3p_1 + 2p_2$
$p_1 = p_2 = 0.4$	16.4	10	12	**5.6**
$p_1 = p_2 = 0.1$	5.6	**5.5**	6	5.9
$p_1 = 0.6$, $p_2 = 0.3$	18.2	8.2	14.8	**4.8**

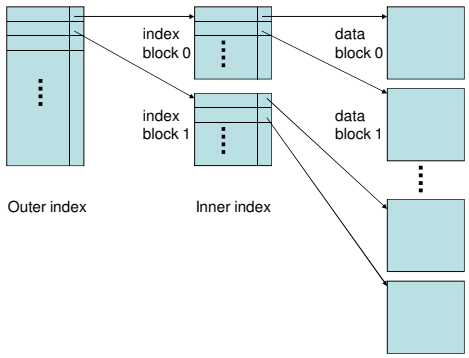
Dense index on sequential file



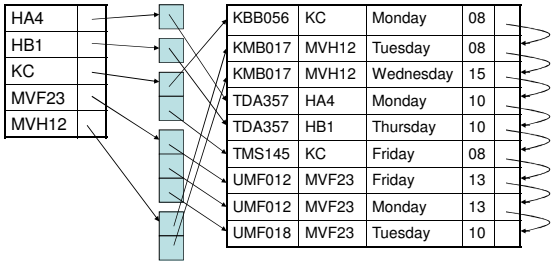
Sparse index on sequential file



Multi-level indexes



Secondary index on *room name*



Quiz!

- Indexes are incredibly useful (although they are not part of the SQL standard).
- Doing it wrong is costly.
- Requires knowledge about the internals of a DBMS.
  - How is data stored? How large is a block?
- A DBMS should be able to decide better than the user what indexes are needed, from usage analysis.

So why don't they??

Summary – indexes

- Indexes make certain lookups and joins more efficient.
  - Disk block access matters.
  - Multi-attribute indexes
- **`CREATE INDEX`**
- Dense, sparse, multi-level and secondary
- Usage analysis
  - What are the expected operations?
  - How much do they cost?

    Σ(cost of operation)x(proportion of operations of that kind)