INDEXES

CS-A1153 - Databases (Summer 2020)

LUKAS AHRENBERG

Aalto University
School of Science

INDEXES

- What is an index
- Why DB indexes are needed
- What indexes to create
- U&W 8:3-8:4
- Additional Example by Kerttu Pollari-Malmi (similar to 8:4, ex 14)

THE PROBLEM

- Databases are typically large
- The data needs to be stored somewhere, *and in some order* (often 'random')
- The tuples accessed by some query could require the DB system to go over all data on disk just to find those which matches a very narrow condition

INDEXES - A SOLUTION

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

- Indexing a way to build up information of where some information is located in the DB table
- The attributes of the index is called the *index key* (these need not be the same as the keys of the relation)
- Note that the index itself needs to be stored
- And, crucially, *the index needs to be updated*
- Often implemented using B-trees

EXAMPLE

```
SELECT * FROM Movies
       WHERE studioName = 'Disney'
       AND year = 1990;
```

- No index means that all movies needs to be searched
- Index on e.g. `year` means DB can quickly narrow down search
- Indexes also products and joins where a table might need to be traversed multiple times

CREATING AN INDEX IN SQL

Creation of indexes not part of SQL standard, but commonly done as

```
CREATE INDEX <index_name> ON <table_and_attributes>;
```

For example:

```
CREATE INDEX KeyIndex ON Movies(title,year);
```

WHICH INDEXES SHOULD BE CREATED?

- How are the tuples of a relation spread over the storage medium?
  - Usually spread over several 'pages' (sequential blocks)
- What are common queries? When does it make sense to create an index?
- In practice there is also a cost associated with reading and updating the index

INDEXES ON RELATION KEYS

- Queries involving the relation key are common, and
- An index on a key yields a single page (because the key is unique)

INDEXES ON 'ALMOST' KEYS

Queries involving 'almost' keys can result in few page hits

```
SELECT * FROM Movies
    WHERE title = 'King Kong';
```

- `title` is not a key of **`Movies`**, but
  - In general there are only a handful of movies with the same name
  - So, few pages to read (compared to the total number of the DB)

INDEXES ON CLUSTERED ATTRIBUTES

If we know that the attribute is *clustered* on a few storage pages, it can also make sense to create an index.

```
SELECT * FROM Movies
       WHERE year = 1990;
```

- `year` is not a key of Movies, but
  - if the movie table should happen to be clustered on year on the storage medium it still makes sense to create an index
  - instead of searching all pages for the right year, we would look up the correct page(s) in the index

U&W 8:4.3 - EXAMPLE 14

StarsIn(movieTitle,movieYear, starName)

$Q_1(s)$: Look for movie title and year for some star, s

```
SELECT movieTitle, movieYear
       FROM StarsIn
       WHERE starName = s;
```

$Q_2(t, y)$: Look for star in some movie t, released in year y

```
SELECT starName
       FROM StarsIn
       WHERE movieTitle = t
       AND movieYear = y;
```

$I(t, y, s)$: Insert entry with title t, year y, and star s.

```
INSERT INTO StarsIn VALUES(t,y,s);
```

EXAMPLE 14 - ASSUMPTIONS

- `StarsIn` Occupies 10 pages
- On average a star has appeared in 3 movies
- On average a movie has 3 stars
- An index fits on one page
- Inserting is easy (one read and one write)
- Usage patterns, we expect
  - Fraction p_1 of queries to be type Q_1
  - Fraction p_2 of queries to be type Q_2
  - Meaning that fraction $1 - p_1 - p_2$ is of type I

EXAMPLE 14 - COSTS WITHOUT INDEX

Q_1 : 10

Need to scan the whole table

Q_2 : 10

Need to scan the whole table

I : 2

One access to read page and one to write it back modified

Average

$$10p_1 + 10p_2 + 2\left(1 - p_1 - p_2\right) = 2 + 8p_1 + 8p_2$$

EXAMPLE 14 - COSTS WITH INDEX ON `starName` ONLY

Q_1 : 4

One to read the index, and then three pages on average

Q_2 : 10

Need to scan the whole table

I : 4

Two to write new data plus two to update the index

Average

$$4p_1 + 10p_2 + 4\left(1 - p_1 - p_2\right) = 4 + 6p_2$$

EXAMPLE 14 - COSTS WITH INDEX ON `(movieTitle,movieYear)` ONLY

Q_1 : 10

Need to scan the whole table

Q_2 : 4

One to read the index, and then three pages on average

I : 4

Two to write new data plus two to update the index

Average

$$10p_1 + 4p_2 + 4\left(1 - p_1 - p_2\right) = 4 + 6p_1$$

EXAMPLE 14 - COSTS WITH BOTH INDEXES

Q_1 : 4

One to read the index, and then three pages on average

Q_2 : 4

One to read the index, and then three pages on average

I : 6

Two to write new data plus four to update the indexes

Average

$$4p_1 + 4p_2 + 6\left(1 - p_1 - p_2\right) = 6 - 2p_1 - 2p_2$$

EXAMPLE 14 - WHICH INDEX?

No index	Star index	Movie index	Both indexes
$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

Best selection - the function with minimum average cost - depends on region of (p_1, p_2).