Indexes in Real World Systems: PostgreSQL and Oracle

Werner Nutt

Introduction to Databases

Free University of Bozen-Bolzano

Example: the "Sailors and Boats" Domain

Information needs to be stored about *sailors*, *boats*, and *reservations* that sailors are making for boats.

(see Ramakrishnan/Gehrke)

There are three relations with the following schemas:

```
sailor(id: int, name: string, ranking: int, age: int)

  boat(id: int, name: string, colour: string)

reservation(sid: int, bid: int, day: date)
```

(primary keys are underlined)

Automatically Created Indexes

PostgreSQL and Oracle create a non-clustered B+-tree index for

1. the **primary key**
   - if the key consists of *more than one* attribute, the index is a *composite* or *concatenated index* *(i.e., an index for several attributes)*
   - a composite index is sorted *lexicographically* according to the order of the attributes

Example: For `PRIMARY KEY (name, age)`, we have

$$(\texttt{Kelly}, 22) < (\texttt{Kelly}, 63) < (\texttt{Smith}, 18) < (\texttt{Smith}, 36)$$

2. any attribute with a `UNIQUE` **constraint**

(*Note:* the index is needed to maintain that constraint)

Explicitly Created Indexes

Principles: An index
- has a *name*
- is created for a *sequence of attributes* over a *table* (which must already exist) *. . . exceptions come with clustered indexes in Oracle*
- can be `DROP`'ped

Examples:
- `CREATE INDEX sailor_name_idx ON sailor(name);`
- `DROP INDEX sailor_name_idx;`
- `CREATE INDEX sailor_name_and_age_idx ON sailor(name, age);`

The standard index is the B+-tree

Clustered Indexes in PostgreSQL

- A table can be clustered according to an index:

```
CLUSTER sailor_name_idx ON sailor
```

- The table is physically reordered,
  but the order is not maintained during updates
- The table needs to be reclustered periodically
  (e.g., by a cron job)
- When reclustering, the index need not be mentioned again:

```
CLUSTER sailor
```

Clustering in Oracle

1. Create a **cluster** (= shared storage for tables with common attributes)

   ```
   CREATE CLUSTER sailor_id_cluster (id int);
   ```

   A cluster is made for a *fixed number* of *typed attributes*

2. Create a **clustered index**

   ```
   CREATE INDEX sailor_id_idx ON CLUSTER sailor_id_cluster;
   ```

   There can only be *one index* for one cluster

3. Create **table** for the cluster

   ```
   CREATE TABLE sailor (id INT NOT NULL,
                        name VARCHAR(20),
                        ranking INT,
                        age INT) CLUSTER sailor_id_cluster (id) ;
   ```

Clustering in Oracle (cntd.)

Several tables can *share* one clustered index

```
CREATE TABLE boat (id INT,
                   name VARCHAR(20),
                   colour VARCHAR(10),
                   PRIMARY KEY (id));


CREATE TABLE reservation (sid INT,
                   bid INT,
                   day DATE,
                   PRIMARY KEY (sid, bid),
                   FOREIGN KEY (sid) REFERENCES sailor(id),
                   FOREIGN KEY (bid) REFERENCES boat(id))
                               CLUSTER sailor_id_cluster (sid) ;
```

Hash Indexes in PostgreSQL

- "HASH" is one of four *index types*

  ```
  CREATE INDEX sailor_id_idx ON sailor USING HASH(id) ;
  ```

  the others being "BTREE", "GIST", and "GIN"

- In release 8.2, one can specify the *fill factor* of buckets

  ```
  CREATE INDEX sailor_id_idx ON sailor USING HASH(id)
                                WITH (FILLFACTOR = 80) ;
  ```

  . . . fill factors can also be specified for other indexes and for tables

- In general, hash indexes need more space than B+-trees and their construction takes much longer than the one of of a B+-tree

  Example at FUB, table with 10 Mio rows occupying 10 GByte:
  50 sec for B+-tree, 2840 sec for hash index

Hash Clusters in Oracle

Hash Clusters realize the concept of a **hash file**

. . . and a bit more than that

```
CREATE CLUSTER sailor_id_cluster (id int)
               SIZE 2K
               HASH IS (id)
               HASHKEYS 10000;
```

This is like an ordinary cluster, but *in addition*

- `SIZE` specifies the **size of buckets**
- `HASHKEYS` gives the **number of keys**

Function Based Indexes

- An index can be based on the values of a *function* over one or more attributes

```
CREATE INDEX emp_income_idx ON emp(sal + comm);
```

- The *query optimizer* can *match* this to a condition containing the function:

```
SELECT e.ename
FROM   emp e
WHERE  e.sal + e.com BETWEEN 1000 AND 1500
```

Exist in both, PostgreSQL and Oracle

Advanced Indexing: Index Parameters

The `CREATE INDEX` statement can include various parameters, e.g.,

- space allocated for the index,
- computation of statistics for optimizer
- ...

Example in Oracle:

```
CREATE INDEX sailor_name_idx ON sailor(name)
                    STORAGE(INITIAL 40K
                            NEXT 20K)
                    COMPUTE STATISTICS;
```

... for all this you need privileges

Advanced Indexing: Index Parameters (cntd.)

In PostgreSQL 8.2, the full syntax for index creation is

CREATE [UNIQUE] INDEX [CONCURRENTLY] *name* ON *table* [USING *method*]

(*column* | (*expression*) [*opclass*] [, ...])

[WITH (*storage_parameter* = *value* [, ...])]

[TABLESPACE *tablespace*]

[WHERE *predicate*]

This is to support among other things

- user-defined indexes (*opclass*)
- partial indexes (`WHERE` *predicate*)

More Index Types in Oracle

- **Index Only tables (IOT):** entries of index are table records
- **Bitmap indexes:** for each value of a domain, there is a bitmap identifying the rid's of satisfying tuples.

  E.g., bitmaps for all tuples with `sex=male` and `sex=female`

  good for small domains tables that don't change too much
- **Bitmap join indexes:** occur in data warehousing and decision support

. . . for more info see Oracle10g Concepts [on the Web]

Queries That May Benefit From Indexes

- Point Queries
- Multipoint Queries
- Range Queries
- Prefix Match Queries
- Extremal Queries
- Ordering Queries
- Grouping Queries
- Join Queries

Exercise

We consider a library database with the following schema:

Book(<u>bookid: `int`</u>, author: `string`, title: `string`, lang: `string`, area: `string`)

Member(<u>mno: `int`</u>, name: `string`, cat: `string`, age: `int`)

Borrowed(<u>mno: `int`</u>, <u>bookid: `int`</u>, <u>day: `date`</u>).

We assume that the database management system created B-tree indexes for the primary key of each relation.

Exercise (cntd)

Suppose that queries of the following type are posed frequently

```
SELECT m.cat
FROM   Book b, Borrowed bd, Member m
WHERE  b.bookid = bd.bookid AND
       bd.mno = m.mno AND
       author = ’E.A. Poe’ AND
       age BETWEEN 20 and 22;
```

where the values of author and age are varying.

Exercise (cntd)

(i) If you could create as many indexes as you liked to speed up the execution of this query, which ones would you choose?

(ii) Explain how those indexes would support the execution of the query. Draw a relational algebra tree to represent the optimal execution plan. Annotate each node with the algorithm by which the operation of the node is executed (for example, "index lookup" or "sort merge join").

(iii) If you could only create a single index, which one would you choose to obtain the greatest speed-up? State the assumptions under which that index would be the best one and explain why that choice would be better than other possible choices.

References

The Slides are based on manuals for PostgreSQL and Oracle on the Web