Physical Data Organization

Tom Kelliher, CS 318

Apr. 12, 2002

1 Administrivia

Announcements

Assignment

Read 11.4–5.

From Last Time

PL/pgSQL lab for some, an embarrassment for others.

Outline

1. Disk Organization.
2. Heap file organization.
3. Sorted file organization.

Coming Up

Indexed organizations.

2 Disk Organization

1. The organization of tables in memory won't affect results, but will greatly affect performance.

2. Definitions:

   (a) Storage structure: A particular organization of the rows of a table in a file.

   (b) Index: An auxiliary data structure, possible stored in another file, permitting fast access to the rows of a table.

   (c) Access Path: The particular technique used for accessing rows. The DBMS optimizer chooses the access path.

   Example: Consider a table indexed on its key. The access path for a `SELECT` on the key would utilize the index. The access path for an aggregate computation on the entire table might just access the pages of the table sequentially, eliminating the index overhead.

3. Ideally, the entire database would be in main memory. Economic realities and the ACID properties dictate otherwise.

4. Disk structure:

   (a) Platter, Read/write heads. Data stored in concentric circles as magnetic flux changes. CRC codes.

   (b) Sectors, blocks, pages.

   Contiguous storage. Fragmentation.

   (c) Cylinder/Head/Sector (CHS) addressing. Logical Block Address (LBA) addressing.

   (d) Access delay components: seek delay, rotational delay, transfer delay.

5. Assumptions:

   (a) All records of same length. Not really practical — consider a `VARCHAR` field.

(b) Table occupies a file containing F pages, R records per page.

(c) Rows addressed logically by row Id (rid): page number, slot number.

3 Heap File Organization

1. Features:

   (a) Rows stored one after another, in no particular order.

   (b) Inserts occur at the end of the file.

   (c) Delete by marking slot free.

   (d) Update in place. Common to all storage structures?

   (e) "Holes" develop with deletes — expensive compaction becomes necessary over time. Why?

2. Operation analysis:

   (a) Insert: $F + 1$ page transfers.

   F to perform duplicate checking, 1 to write new record.

   (b) Delete: $F/2 + 1$, on average if a key is involved.

   Otherwise, $2F$. Why not F as in the book?

   (c) Selects:

      i. Entire table, simple:

      ```
      SELECT *
      FROM TRANSCRIPT T;
      ```

      F transfers.

ii. Entire table, not so simple:

```
SELECT *
FROM TRANSCRIPT T
ORDER BY T.StudId;
```

$F + N$ transfers.

iii. Entire table, simple:

```
SELECT AVG(T.Grade)
FROM TRANSCRIPT T;
```

F accesses.

iv. Equality search for one record:

```
SELECT T.Grade
FROM TRANSCRIPT T
WHERE T.StudId = '123456' AND T.CrsCode = 'CS318'
AND T.Semester = 'S2002';
```

$F/2$ transfers on average.

v. Range search for several records:

```
SELECT T.Grade
FROM TRANSCRIPT T
WHERE T.StudId > '100000' AND T.StudId < '200000';
```

F transfers.

4 Sorted File Organization

1. Features:

   (a) Rows stored one after another, sorted on a key field or fields.

   (b) Inserts must maintain sorted order.

      i. Expensive — on average, F page transfers. Why F?

ii. Optimizations:

A. Fillfactor — leave some empty slots in each page.

B. Utilize a pointer in each page to a chain of overflow pages.

Disadvantage: pages of file are no longer contiguous.

(c) Delete by marking slot free.

2. Operation analysis:

(a) Insert: Assuming contiguous allocation and fillfactor, $\log n + 2$ page transfers.

Derivation?

(b) Delete: Assuming contiguous allocation *and* the file is sorted on the relevant key, $\log F + 1$.

Derivation?

If it's not sorted on the relevant key?

(c) Selects:

i. Entire table, simple:

```
SELECT *
FROM TRANSCRIPT T;
```

Transfers?

ii. Entire table, not so simple:

```
SELECT *
FROM TRANSCRIPT T
ORDER BY T.StudId;
```

Transfers?

iii. Entire table, simple:

```
SELECT AVG(T.Grade)
FROM TRANSCRIPT T;
```

Transfers?

iv. Equality seach for one record:

```
SELECT T.Grade
FROM TRANSCRIPT T
WHERE T.StudId = '123456' AND T.CrsCode = 'CS318'
AND T.Semester = 'S2002';
```

Transfers?

v. Range search for several records:

```
SELECT T.Grade
FROM TRANSCRIPT T
WHERE T.StudId > '100000' AND T.StudId < '200000';
```

Transfers?