COSC2406/2407 Database Systems

Files, Pages and Records

Xiangmin (Emily) Zhou

Onli

Thursdays

Email : xiangmin.zhou@rm

Lecture 3

References: Ramakrishnan & Gehrke Chapter 9
Garcia-Molina et al. Chapter 12
Diagrams courtesy Ramakrishnan & Gehrke

Overview

In this lect

- Dis
- Dis
- Mo

Representing Data Elements

Last week, we discussed disks, their characteristics, and how DBMS buffer managers interact with disks.

We focused on disc blocks (corresponding to pages in memory), and now we fo

1. Attr byt

2. Fields are stored together to form logical *records*
3. Records are stored in blocks (pages)
4. Blocks of records of the same type are typical form a *file* (note that a file in a DBMS is differ operating system file)

Records

Records group together related fields. Logical records might be, for example, student records or, say, order records that store logically

Each typ [illegible] dit lists the na

A record can be thought of as a unit of storage, the len [illegible] depends on the particular application.

```
CREATE TABLE character (
   name C
   clas
   leve
   gender CHAR(1),
   last_access DATE
);
```

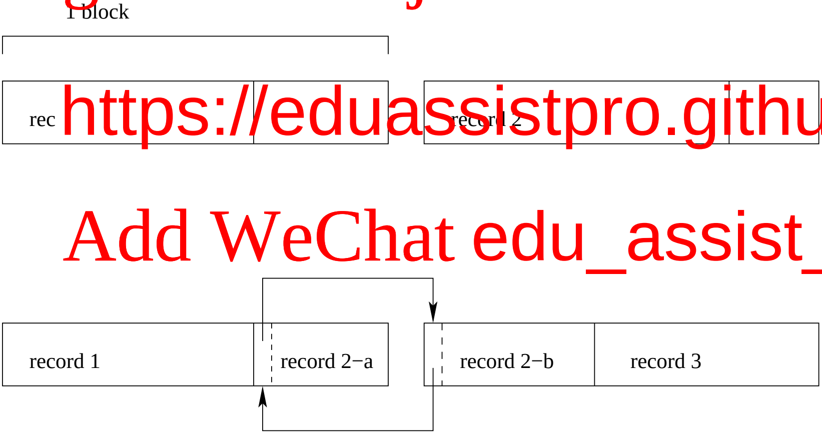
An Example Relation

				13
				01
Varra	Warrior	19	M	12/12/04

An Example Record

Spanning

A block on disk—a physical record—may contain more than one logical record. A logical record that is partly in one block, and partly in another is called a *spanning record*.



Blocking Factor

- The *blocking factor* is the block size divided by the record size.
- Suppose that a disk is formatted with a block size of 2048 bytes, and t
- Ho
spa

- How many records can we store in 10 blocks if
allowed (not taking header information int

BLOBs

- Each logical record may also take more than one physical block. For example, many modern applications store very large records, called BLOBs (Binary Large OBjects).
- Tru
  MP
- BL
  retrieval. However, it might not always be n
  BLOB in its entirety (can use a linked list of bloc
  movie)
- If it is necessary that parts of a BLOB can be retri
  third scene of a movie), an index is used.

File Organisation

File organisation deals with how the records are placed in a file.

The *file access method* is the technique used to get to records.

The facto

- spe
- stor

- file volatility

The operations that need be supported by the acc include:

- Scan the file and access all the records
- Search for records matching a value, or in a range of values
- Insert or delete a record

Record Identifiers (*rids*)

The DBMS needs a way to identify records. Each record is generally identified by a pair:

<page_id, slot_number>

Alternati
rid.

This allows for more flexible page formats, but ha
mapping table that lists the corresponding pa
each record must be maintained. Due to the overh
such a table, the first approach is more common.

Page Formats

A page consists of a collection of logical records.

How are (logical) records arranged in a page?

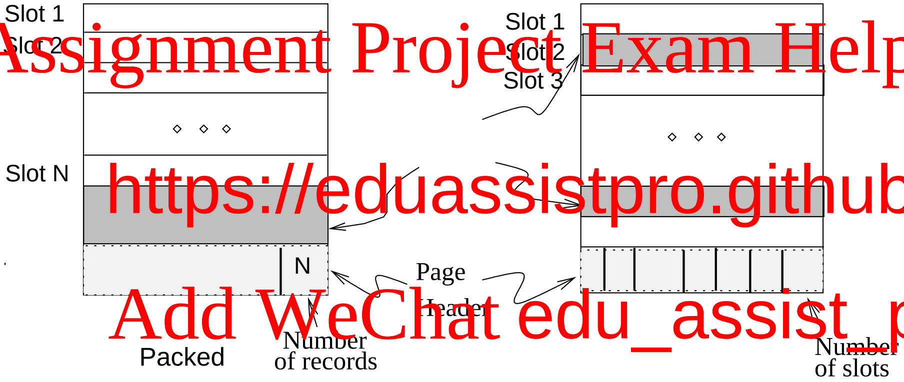
A page is th

With fixe
records.

- Packed organisation: each page has a count of n
All the free space is at one end. Record deletion wil
records.

- Unpacked: Each page has a bit map indicating which slots are full and which are empty. Deleting a record simply involves setting the corresponding bit to zero.

Packed and Unpacked Organisations



Packed Organisation

In the packed page organisation, records are stored in the first N slots (where N is the number of records on a page). When a record is deleted, the last record on the page is moved to the empty slot.

Advanta

- can cal

- all empty records are together at the end of the page

Disadvantage:

- Problems occur if there are external refere moved: its slot number will change, and the sl the *rid*

Unpacked Organisation

In the unpacked page organisation, an array of bits is used to keep track of free slot information (one bit per slot).

When a re
requires
scheme

In both organisations, a page usually stores addi
page header. This includes file-level informati
next page in the file.

Record Formats

Records may be fixed length or variable length.

If records are of a fixed-length, then compact storage is simple.

For varia
on:

- frequency of insertion
- frequency of update and deletion
- whether system is available 24 hours
- how much “scratch space” is available
- whether record order is important

All data is ultimately represented as a sequence of bytes.

Numeric
interpret
arithmet

INTEGER: typically 2 or 4 bytes
FLOAT: typically 4 or 8 bytes

Fixed-length character strings: CHAR(*n*).
These are represented using an array of n bytes. If the string is shorter than *n*, th

Exampl
CHAR(6


```
bat###
```

where # is a pad character whose 8-bit code is not
SQL string.

Variable-length character strings: VARCHAR(n).
A possible implementation is to assign $n+1$ bytes, no matter how long the string is. The extra byte is used to indicate the length of the string, and additional bytes are not used.

Exampl
VARCH


```
3bat
```

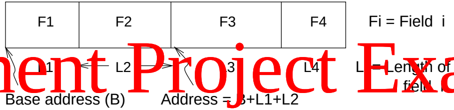
Note that 7 bytes are consumed (even though the l
actually used).

An alternative implementation would be to actu
number of bytes. This would save space, but introduces added complexity through the creation of variable-length records.

Dates: DATE
In SQL, th
like CHA

Times: TI
VARCHAR (if an upper limit on length is imposed).
Bits: BIT(n) Can store 8 in a byte.

Fixed-length Records



- Info stor wit

- All re and at the same offsets in each record
- The address of the ith field can be calcul access

- Timestamps may be stored with each that indicate last modification or read time

(The structure of fixed-length records is discussed in Section 9.7.1.)


name
class
ss
0
30
55
56
66

Variable-Length Records

The size of fields may vary, fields may repeat, fields may be optional, fields may have variable formats that are not pre-declared before use and some fields may be large in size. (More details of field formats are in Sectio

These co

If fixed- an

fixed-le

- The length of the record is stored in the header
- Pointers are stored to the beginning of each o fields (or variable-length fields are separa character, that is, a delimiter)

Variable-Length Records

Two approaches: delimiters and direct field a

Variable-length records are discussed further in Section 9.7.2.

Field Access Using Pointers

In the second approach, an array of integer offsets is stored at the start of the record.

The i^{th} i tive to the start of th

Advanta

- Direct access to any field
- Easy to deal with NULL values

We have seen that if there are variable length fields, we can use characters such as ? or $ or % as separator characters between the fields.

When the

a typical re d of just

storing v


```
NAME=Smith%ADDR=124 Ann St.%WORK=firem
```


This can be improved further by storing codes suc

“NAME”.

Page Formats and Variable-length Records

When records are of variable length, the previous approach of dividing a page into a fixed number of slots is no longer feasible.

When a ne
correct le

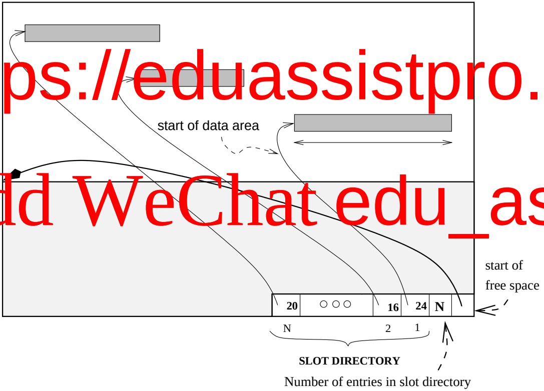
When a record is deleted, the hole that is created m
that all of the free space on the page is in one contiguo

We therefore need to be able to move records arou

Page Formats

Variable length records are handled by having a directory of slots in each page.

This organisation enables moving of records, expansion of records, and so on.



Managing Free Space

Free space can be managed by maintaining a pointer to the start of the free space region.

When a ne
the page a
deletion
contiguous order, followed by free space.

This organisation can also be used for fixed-leng
useful if the records need to be moved frequently)

IBM DB2:

- Fixed-length fields are at fixed offsets from the record's starting add
- Var fixe fixe

Oracle 8:

- Records are sequences of length and data p are treated as if they are of variable length.

Page Format in Apache Derby

- tables (and indexes) stored in Containers
- data is stored in pages in Container
- a data page contains
  -
  -
  -

    (defines identity of record on page)
  - slot offset table
  - checksum (8 bytes)

http://db.apache.org/derby/papers/pageformats.html

Apache Derby uses a variable length record, containing:

- a record header (1 byte) with bits indicating status of record, e.g.
  - whether record deleted
  -
- one
  -
  - fieldDataLength (a variable length CompressedInt)
  - fieldData

http://db.apache.org/derby/papers/pageformats.html

Slot Offset Table in Apache Derby

Slot Offset Table contains of 6 bytes (12 bytes when pagesize $\geq$ 64KiB) per record:

- 2 byt
- 2 byt
- 2 byt
  this

http://db.apache.org/derby/papers/pag

- represents JSON documents in binary format called BSON
- sup
  In-

https://www.mongodb.com/json-and-bson

BSON basic types

The following basic types are used as terminals in the rest of the grammar. Each type must be serialized in little-endian format.


```
byte 1 byte (
int32 4 byt                                              t)
int64 8 byt                                              t)
uint64 8 by
double 8 bytes (64-bit IEEE 754-2008 binary floa
decimal128 16 bytes (128-bit IEEE 754-2008 dec
```


http://bsonspec.org/spec.html

BSON grammar (incomplete extract of)

```
document ::= int32 e_list "\x00"     BSON Document:
                                     int32=total bytes in document

e_list ::= element e_list | ""
element ::= "\x01" e_name double     64-bit binary floating point
    | "\x02" e_name string           UTF-8 string
    | "\x
    | "\x
    | "\x
    | ...
e_name ::= cstring                   Key name
string ::= int32 (byte*) "\x00"      String: int32=number b
                                     (b                '\x00')
                                     (b
cstring ::= (byte*) "\x00"           0 or more modified UTF-8 chars
                                     followed by '\x00'
                                     (byte*) MUST NOT contain '\x00'
                                     hence not full UTF-8
binary ::= int32 subtype (byte*)     Binary:
                                     int32=number of bytes in (byte*)
```

The following are some example documents in JSON and their corresponding BSON representations.


```
{"hello": "world"} ->

  \x16\                                   // total do
  \x02                                    // 0x02 = type String
  hello\x00                               // field name
  \x06\x00\x00\x00world\x00               // field value
  \x00                     // 0x00 = type E00 (         t )
```


http://bsonspec.org/faq.html

BSON another example

```
{"BSON": ["awesome", 5.05, 1986]} ->
```



```
\x31\x00\x00\x00
\x04BSO
\x26\x0
\x02\x3
\x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
\x10\x32\x00\xc2\x07\x00\x00
\x00
\x00
```


http://bsonspec.org/faq.html

Database applications process logical records that must be read from secondary storage into memory.

When a logical record is required, its physical block is read into a memory buffer.

The logic

the DBM

Becaus

every request for a logical record may not require a physical block.

Similarly, a write may not necessarily cause a pag

disk. The record may be added to a buffer that is writt

complete physical record has been constructed, or when dictated by buffer management policy.

Records and Files ...

Typically the DBMS must carry out the following operations:

- Find a record, transferring the corresponding block to the memory buff
- Re
  rec

- Find the next record
- Delete a record
- Modify a record; may expand or reduce a rec

- Insert a record

1. Data should be organised such that logically associated records are stored together—this is known as *clustering*. Other types of clustering (such as by frequency of access are possible).
2. Sort
   mo

3. Try t
   indexing later).
4. Store data as densely as possible by organisi
   using compact storage to increase the likeliho
   records in a disk block.

In this lecture, we have covered:

- File
- Re
- Pa
- Apache Derby record and page formats
- MongoDB record format