Xi'an Jiaotong-Liverpool University 西交利物浦大学

DTS207TC Database Development and Design

Lecture 9

Chap 13. Data Storage Structure

Di Zhang, Autumn 2025

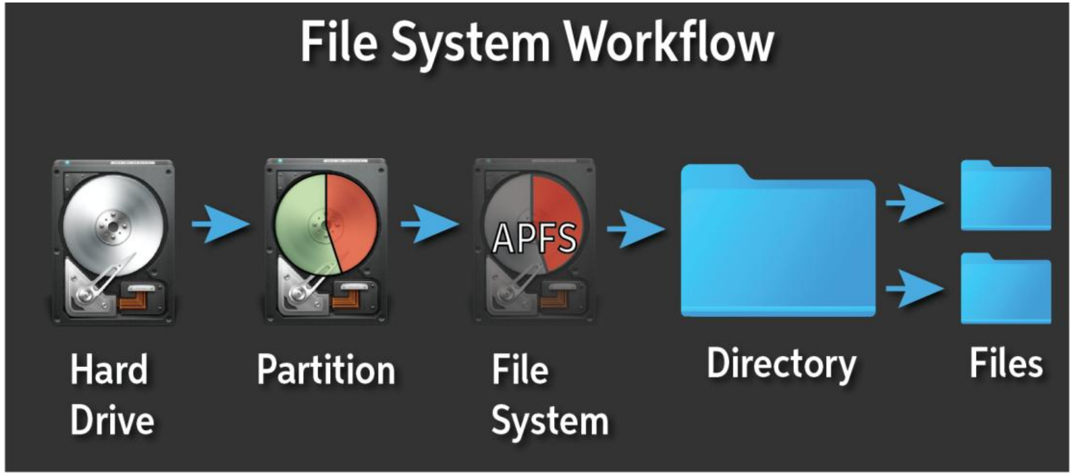
Titles with * will not be assessed

Outline

- Database vs. File system*
- File Organization
- Organization of Records
- Cache
- Column-Oriented Storage*

What is a File System?

- The Translator Between **You** and **Your Storage**
- The Core Problem:
  - A storage device (HDD, SSD) is just a "dumb" collection of fixed-size blocks.
  - How do we store, find, and manage our data (documents, photos, code) in a user-friendly way?
- The Solution: The File System
  - It is the software layer in an operating system that manages how data is stored, organized, and retrieved on a storage device.



What It Provides:

Abstraction & Structure	Core Functions
Files: Named collections of data (bytes).	Naming: Creates a human-readable name for data.
Directories/Folders: Hierarchical organization.	Metadata: Tracks info like size, creation time, permissions.
Pathnames: A logical address (/home/doc/report.txt).	Access Control: Manages who can read/write files.
	Reliability: Prevents corruption (e.g., journaling).

The Magic Trick: Mapping

- The file system's key job is to translate the logical structure (Files & Folders) into the physical reality (Blocks on a disk).

```
[User/App sees: "/Projects/data.txt"]
                 ↓
[File System maps this to:]
  - Block 1023: "Hello, "
  - Block 1045: "World!"
  - Block 2011: (Rest of file...)
```

- In a Nutshell: A File System turns a chaotic pile of raw storage blocks into an organized, usable, and reliable digital filing cabinet.

Database Systems vs. File Systems: A Layered Partnership

- File Systems: Presents a linear array of bytes within each file.
- Database System: as the "Data Expert"
  - Abstraction: Presents tables, indexes, schemas, and a high-level query language (e.g., SQL).
- Key Responsibilities:
  - Data Structures: Implements tables, B+Trees, and indexes within one or more files provided by the file system.
  - Query Processing: Translates high-level queries into low-level file operations (read, write, seek).
  - Advanced Control: Implements its own buffer pool caching and transaction logs for reliability and performance.

The Crucial Relationship

A Database System is a sophisticated user of the File System.

- The File System provides the foundation of persistent storage, while the Database System adds the logic for data organization, integrity, and access.

[Database Engine (SQL, Transactions, Query Optimizer)]
↓ (Uses)
[File System API (open, read, write, seek)]
↓ (Manages)
[Raw Disk Blocks (Hard Drive / SSD)]

How Programs Control Data Layout Within Files

- Key Distinction: Logical vs. Physical
  - Logical Layout: The order and position of bytes as seen by the application. This is what a program controls.
  - Physical Storage: The location of data on the actual disk hardware. This is managed automatically by the OS and File System.
- Core Mechanism: The File Pointer
  - Every open file has a file pointer (or file offset), which acts as a cursor within the file.
  - Read/Write operations start at the current position of this pointer and advance it automatically.
- Primary Control Method: Seeking
  - The lseek (Linux/POSIX) or SetFilePointer (Windows) system call is the primary tool.
  - It allows a program to reposition the file pointer to any byte offset within the file.

```
// 1. Open a file
int fd = open("data.bin", O_RDWR);

// 2. CONTROL: Move pointer to a specific location
(e.g., 1024 bytes from start)
lseek(fd, 1024, SEEK_SET);

// 3. Write data at that exact location
write(fd, buffer, buffer_size);

// The data "buffer" is now logically placed at
offset 1024-1024+buffer_size.
```

The Abstraction of Storage

- The Hardware Reality: Disk Sectors
- Definition: The smallest physically addressable unit on a storage device.
- Typical Size:
  - Legacy HDDs: 512 Bytes
  - Modern HDDs/SSDs: 4096 Bytes (4KB) - "Advanced Format"
- Analogy: The **individual bricks** used in construction.
- The Software Abstraction: File System Blocks
- Definition: The smallest unit of allocation managed by the file system.
- Typical Size: 4KB (most common), 8KB.
- Why? Efficiency. Managing 4KB blocks is far less overhead than managing 512B sectors.
- Analogy: Pre-assembled **wall panels** made from multiple bricks.

The Crucial Relationship: The Block-to-Sector Map

- A single File System Block spans multiple contiguous Disk Sectors, or one big sector.

```
File System View:  [      Block 0 (4KB)      ][      Block 1 (4KB)      ]
                   |                       | |                        |
Disk View:       [Sector A][Sector B]...[Sector H] [Sector I][Sector J]...
(512B Sectors)     (8 Sectors per Block)        (8 Sectors per Block)

// OR with Modern 4K Sectors
File System View:  [    Block 0 (4KB)    ][    Block 1 (4KB)    ]
                   |                   | |                    |
Disk View:        [    Sector A (4KB)    ][    Sector B (4KB)    ]
                   (1 Sector per Block)      (1 Sector per Block)
```

File Organization

- The database is stored as a collection of *files*. Each file is a sequence of *records.* A record is a sequence of fields.
- One approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  This case is easiest to implement; will consider variable length records later
- We assume that records are smaller than a disk block

.

Fixed-Length Records

- Simple approach:
  - Store record *i* starting from byte *n* ∗ (*i* – 1), where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - **move records *i* + 1, . . ., *n* to *i, . . . , n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record 3 deleted

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

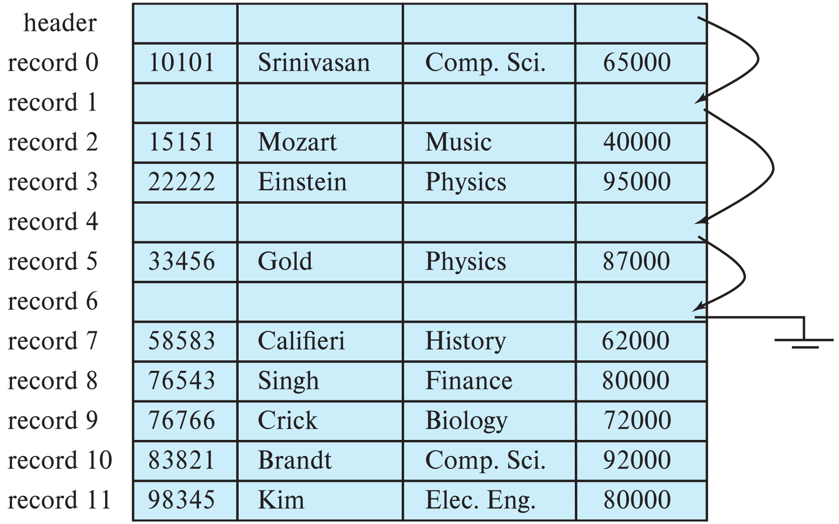
- Deletion of record *i:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1
  - **move record *n* to *i***
  - do not move records, but link all free records on a *free list*

Record 3 deleted and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fixed-Length Records

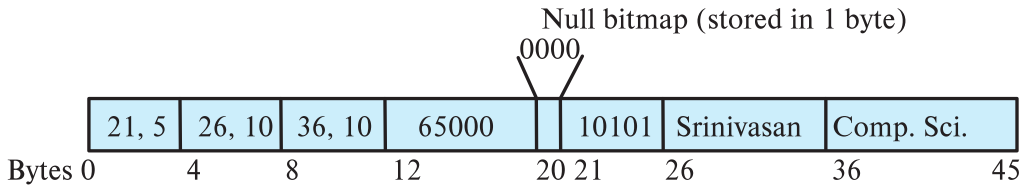
- Deletion of record *i:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i, . . . , n* – 1
  - move record *n* to *i*
  - **do not move records, but link all free records on a *free list***



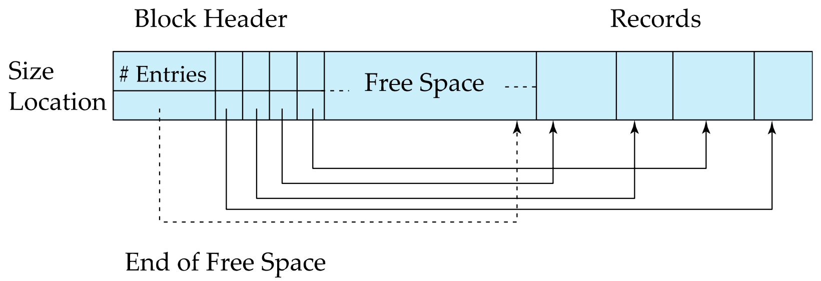
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models).
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

Storing Large Objects

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record

Heap File Organization

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map**
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

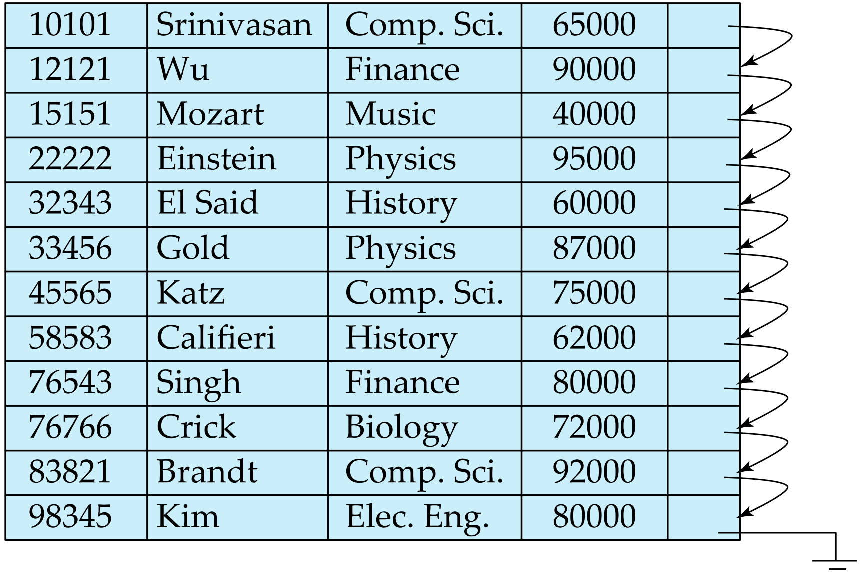
  - Can have second-level free-space map
  - In example below, each entry stores maximum from 4 entries of first-level free-space map

4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

Sequential File Organization

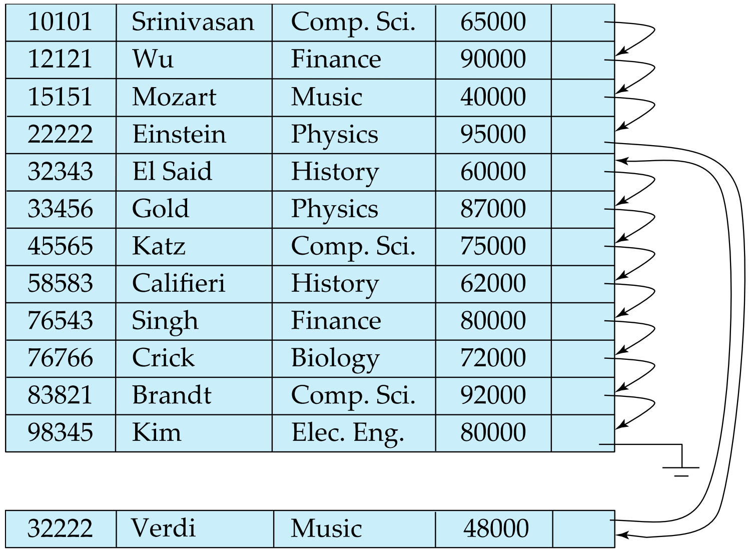
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



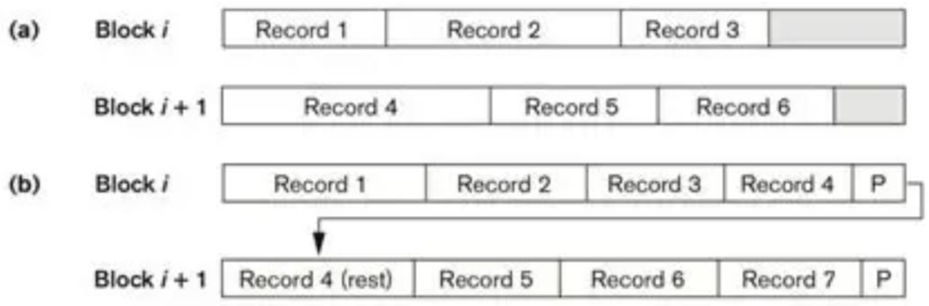
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Unspanned Organization in Databases

- In unspanned organization, a database record must fit entirely within a single database page.
- If a record is too large for the current page's free space, it is placed entirely in a new, different page.
- No record is ever split or fragmented across multiple pages.

Figure 16.6 Types of record organization. (a) Unspanned. (b) Spanned.

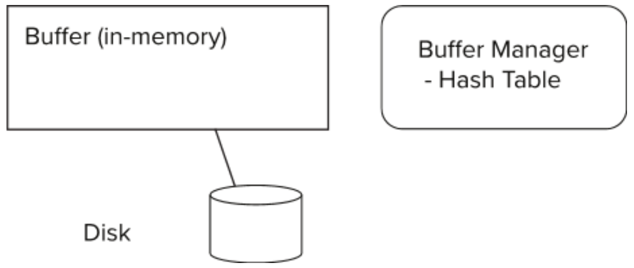


Contrast: Spanned Organization

- **Spanned Organization** allows a single record to be split across multiple pages if it doesn't fit in one.
- This is more space-efficient but adds complexity, as the DBMS must manage pointers to all the record's fragments.
- **Conclusion: Unspanned organization prioritizes read performance and simplicity at the cost of potential space inefficiency.** It is the most common default for storing regular table data.

Storage Access

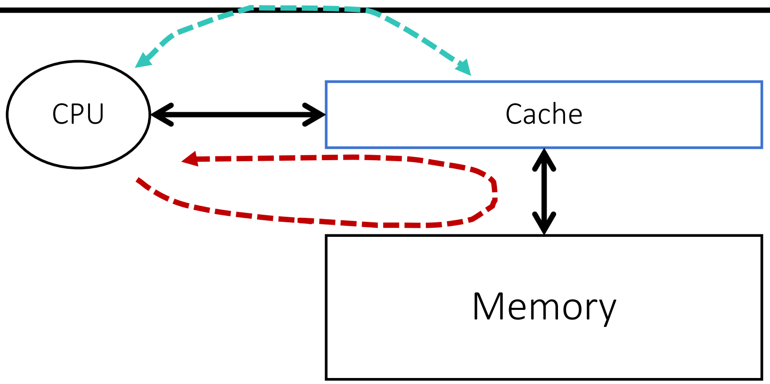
- Blocks are units of both storage allocation and data transfer.
- Database system seeks to minimize the number of block transfers between the disk and memory. We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.



Buffer Manager

- Programs call on the buffer manager when they need a block from disk.
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory
  - If the block is not in the buffer, the buffer manager
    - Allocates space in the buffer for the block
      - Replacing (throwing out) some other block, if required, to make space for the new block.
      - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
    - Reads the block from the disk to the buffer, and returns the address of the block in main memory to requester.

Cache Policy



- A **Policy** needs to be defined to keep the access to Memory in the cache as much as possible

$$T = m \times T_m + T_h$$

- m： cache miss rate
- T_m: memory access time
- T_h: cache access time

- → the lower m the better

First In First Out (FIFO)

Data Addr.	A	B	B	E	F	B	C	B	E
	A	A	A	**E**	E	E	E	E	**E**
	B	**B**	**B**	B	**F**	F	F	F	F
	C	C	C	C	C	**B**	B	**B**	B
	D	D	D	D	D	D	**C**	C	C

- Missing rate = 4/9
- Always replace the earliest entry in the cache
- This part will be Lab 1

Least recently used (LRU)

Data Addr.	A	B	B	E	F	B	C	B	E
	A(0)	A	A	A	A	A	C(7)	C	C
	B(-3)	B(2)	B(3)	B	B	B(6)	B	B(8)	B
	C(-2)	C	C	E(4)	E	E	E	E	E(9)
	D(-1)	D	D	D	F(5)	F	F	F	F

- Missing rate = 3/9
- This strategy requires keeping a "date of birth" for cache lines and tracking the "least recently used" (oldest) cache line based on it
- The difference between LRU and FIFO is that Processor's access to a record will update its "date of birth", while FIFO does not

Random Replacement Policy

- Steps:
  - If the cache is not full, save the most recently accessed data in the cache
  - If it is full, replace one of them at random.

  - Simple and stable, once used in ARM processors

Other Policies

- More policies can be found:
- https://en.wikipedia.org/wiki/Cache_replacement_policies

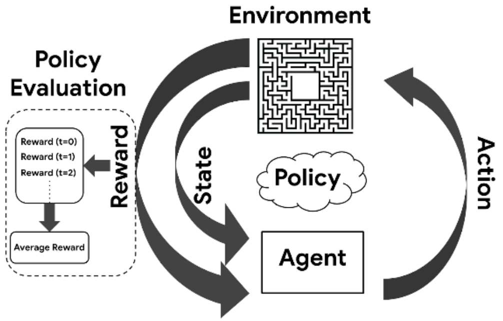
- Find or design a suitable policy for Q6 in CW2!

Further Reading*

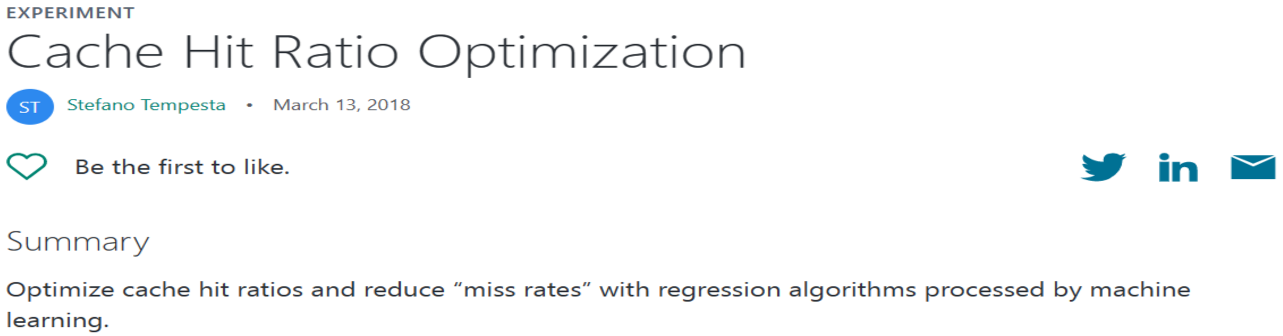
- Cache is everywhere
  - Web Content
  - Video Broadcast
  - Logistic/warehousing

- Can it be solved by ML? Yes.

Amazon Logistics



Policy in Reinforcement Learning


EXPERIMENT

Cache Hit Ratio Optimization

ST Stefano Tempesta • March 13, 2018

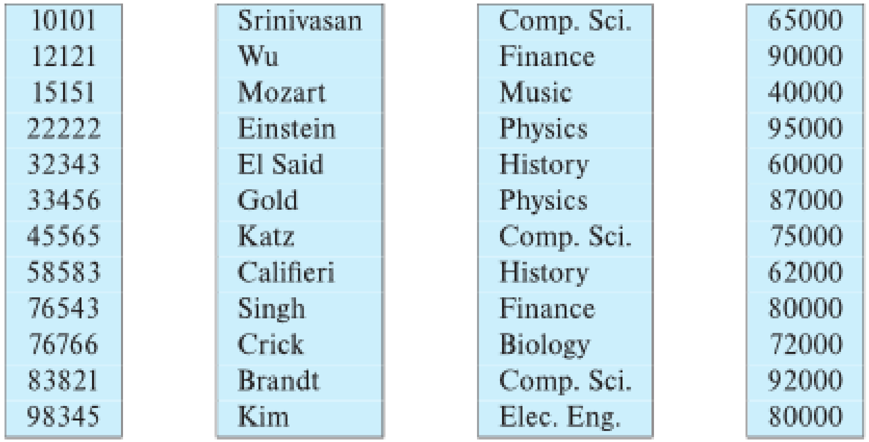
Be the first to like.

Summary

Optimize cache hit ratios and reduce "miss rates" with regression algorithms processed by machine learning.

Column-Oriented Storage

- Also known as **columnar representation**
- Store each attribute of a relation separately
- Example



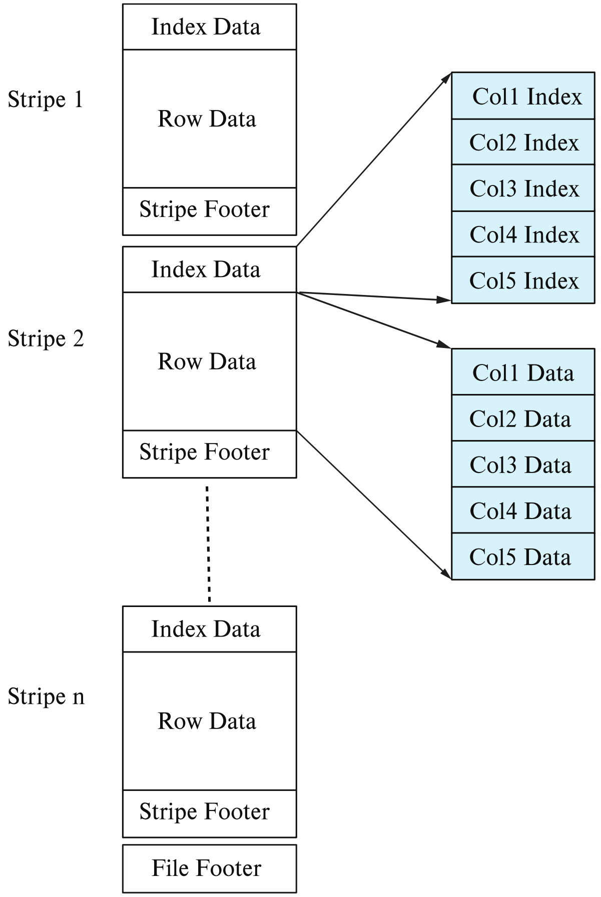
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Columnar Representation

- Benefits:
  - Reduced IO if only some attributes are accessed
  - Improved CPU cache performance
  - Improved compression
  - **Vector processing** on modern CPU architectures
- Drawbacks
  - Cost of tuple reconstruction from columnar representation
  - Cost of tuple deletion and update
  - Cost of decompression

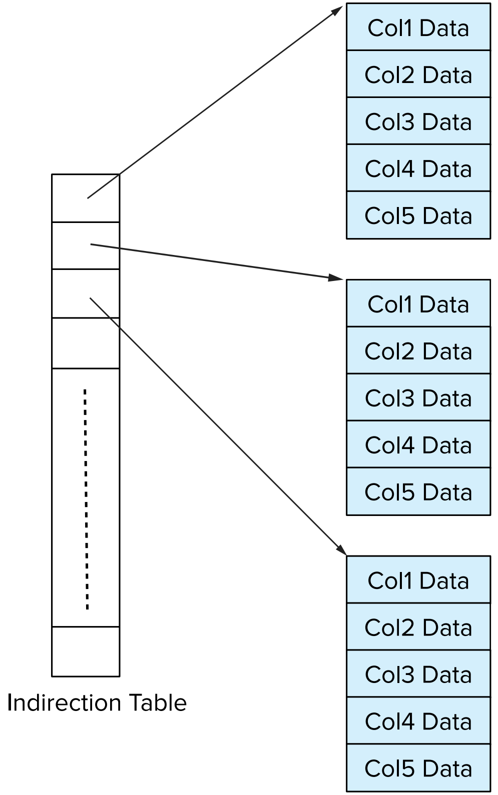
Columnar File Representation

- ORC and Parquet: file formats with columnar storage inside file
- Very popular for big-data applications
- Orc file format shown on right:



Storage Organization in Main-Memory Databases

- Can store records directly in memory without a buffer manager
- Column-oriented storage can be used in-memory for decision support applications
  - Compression reduces memory requirement



Case

APACHE
HBASE