CS222/CS122C: Principles of Data Management

UCI, CS, Fall 2019
Notes #03

Row/Column Stores, Heap Files, Buffer Manager, Catalogs

Instructor: Chen Li

Column Stores vs Row Stores

- Row stores: store data row by row
  - Traditional DBs, e.g., MySQL
  - Example: select * from sales WHERE sid = 9414;
- Column stores: store data column by column
  - Benefits: easy compression, good for queries accessing few columns
  - Example: select AVG(price) from sales;
  - Examples: MonetDB, Vertica
- Nowadays many commercial DBs support both, e.g., SQL Server

OLTP vs OLAP queries

Row store

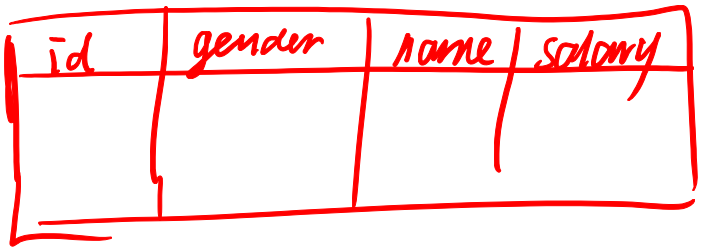
- OLTP: Online Transaction Processing
  - Many concurrent requests by many users
  - Tend to be simple
  - High requirements on latency (in ms) and throughput
  - Indexes are very critical
  - Row stores are often good

OLTP vs OLAP queries

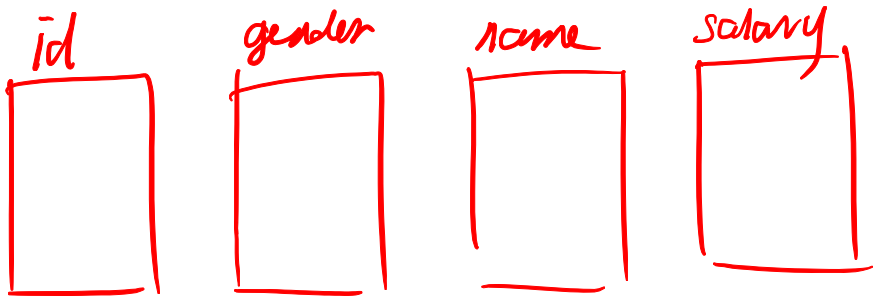
Column store

- OLAP: Online Analytical Processing
  - Need to access large amounts of data
  - Often aggregation operations (e.g., AVG, SUM, MIN) for reporting purposes
  - Few, complicated queries. Take long time to run
  - Indexes can be less helpful. Often needs scan
  - Columns are often helpful

Table "Employee"



① column store



select AVG(salary) from Employee.

(OLAP application)

② row store.

select * from employee where id=18

(OLTP application)

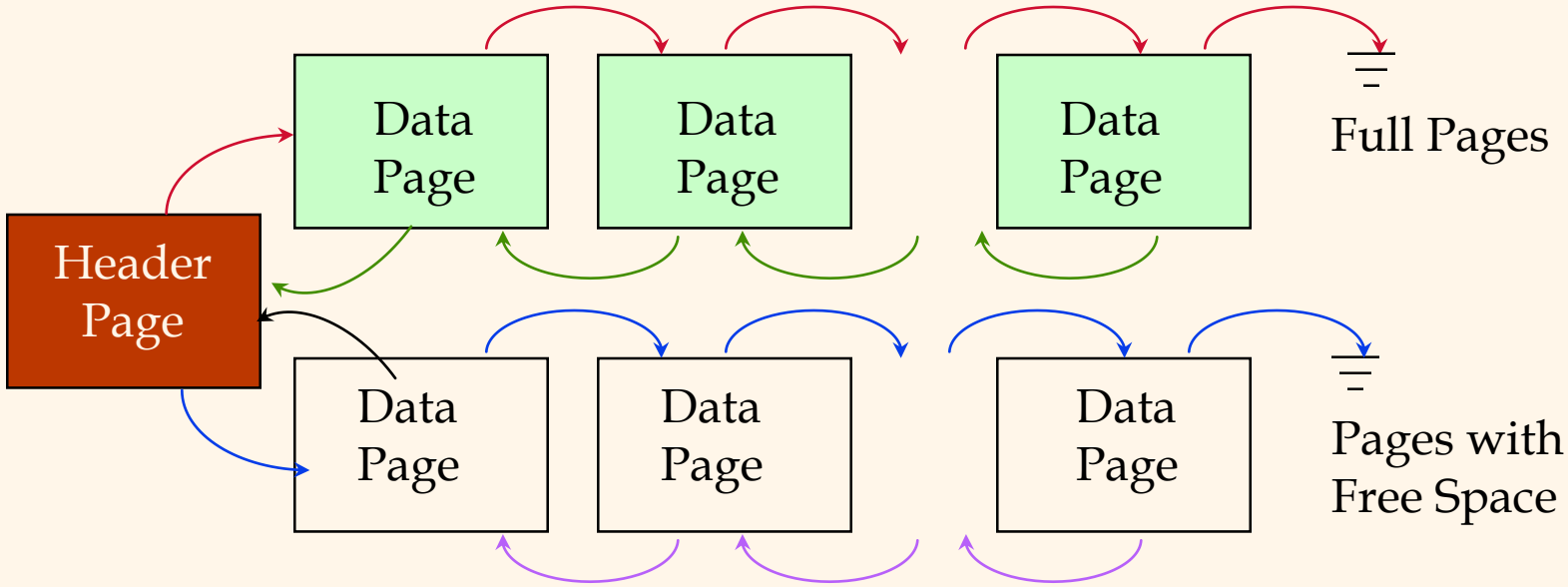
Files of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and thus want *files of records*.
- FILE: A collection of pages, each containing a collection of records. Must support:
  - Insert (append)/delete/modify record
  - Read a particular record (specified using *record id*)
  - Scan all records (possibly with some conditions on the records to be retrieved)

Unordered ("Heap") Files

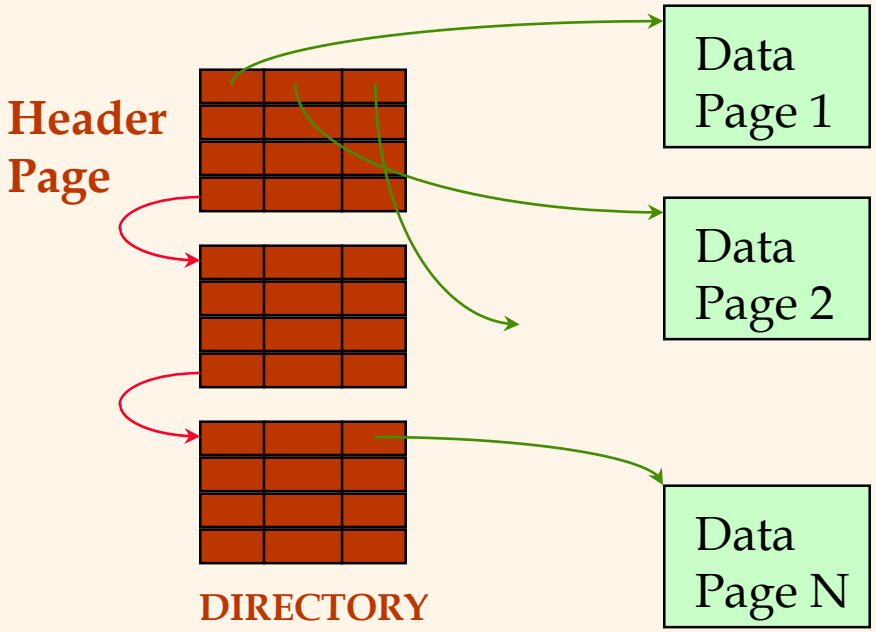
- Simplest file structure that contains records in no particular (logical) order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* within and across pages
  - keep track of the *records* on a page
  - keep track of *fields* within records
- There are many alternatives for each.

Heap File Implemented as a List



- The header page id and Heap file name must be stored someplace. (Project 1 note: The OS filesystem can help…! ☺)
- Each page contains two extra "pointers" in this case.
- Refinement: Use several lists for different degrees of free space (to mention just one of many possibilities).

Heap File Using a Page Directory



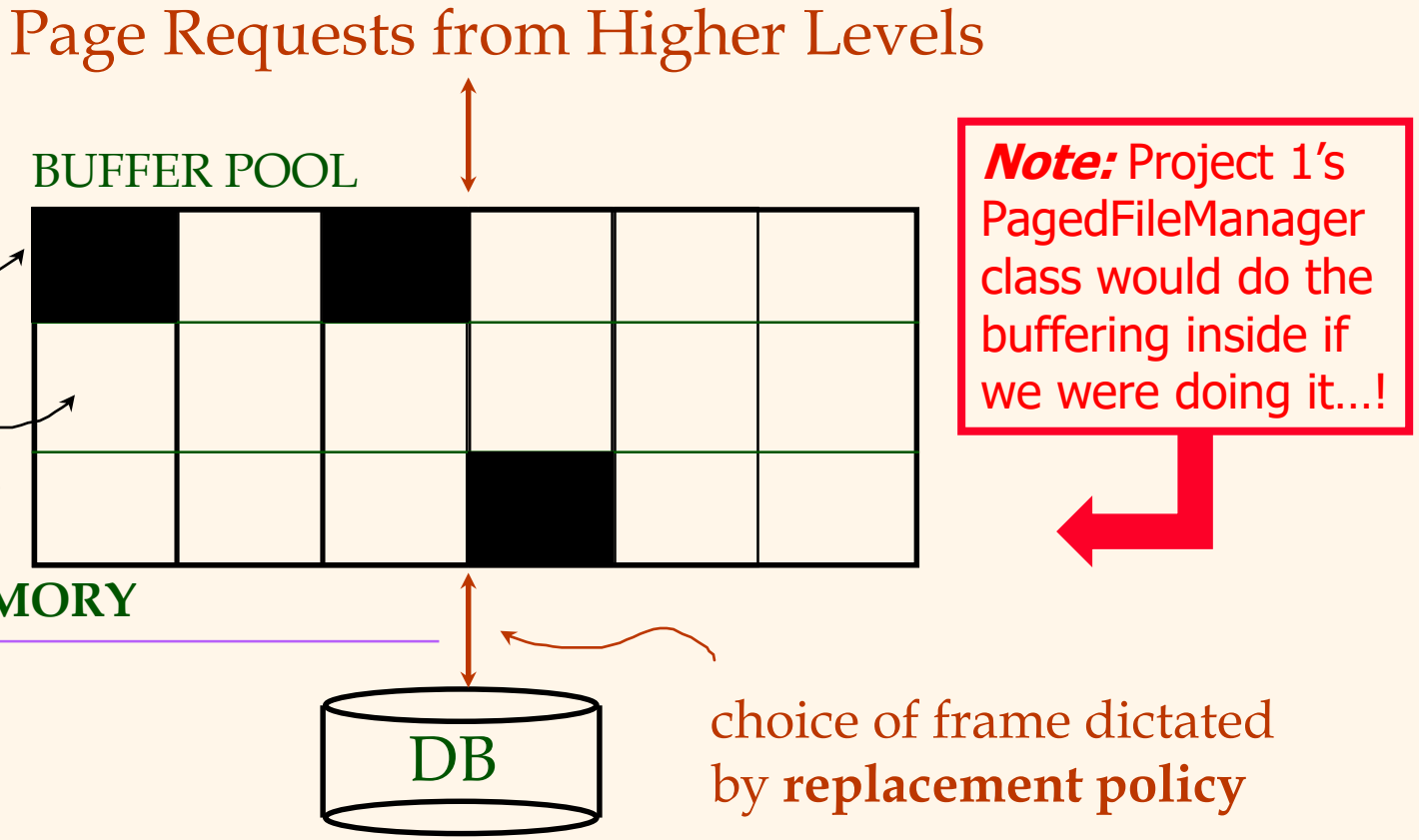
- Page entries can include the number of free bytes on each page
- Directory is a collection of pages; linked list just one possible implementation. (*Note*: Can also do *extents*!)

Next topic: Buffer Management

dirty bit: whether the page is changed

PIN/UNPIN: counter which store the number of the processes currently using the file

shared-ptr



Note: Project 1's PagedFileManager class would do the buffering inside if we were doing it...!

- *Data must be in RAM for DBMS to operate on it!*
- *Table of <frame#, pageid> pairs is maintained.*

When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for *replacement*
  - If that frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address

add counter

discress counter

Unpin

A counter stores # of processes using this file

* *If requests can be predicted (e.g., sequential scans) pages can be prefetched several pages at a time!*

More on Buffer Management

- Requestor of page must unpin it, and indicate whether page has been modified, when done:
  - *dirty* bit used for the latter purpose
- Page in pool may be requested many times
  - a *pin count* is used, and a page is a candidate for replacement iff *pin count* = 0.
- CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol; more in CS 223.)

Buffer Replacement Policy

- Frame is chosen for replacement using a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU, etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- *Sequential flooding*: Nasty situation caused by LRU + (repeated) sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

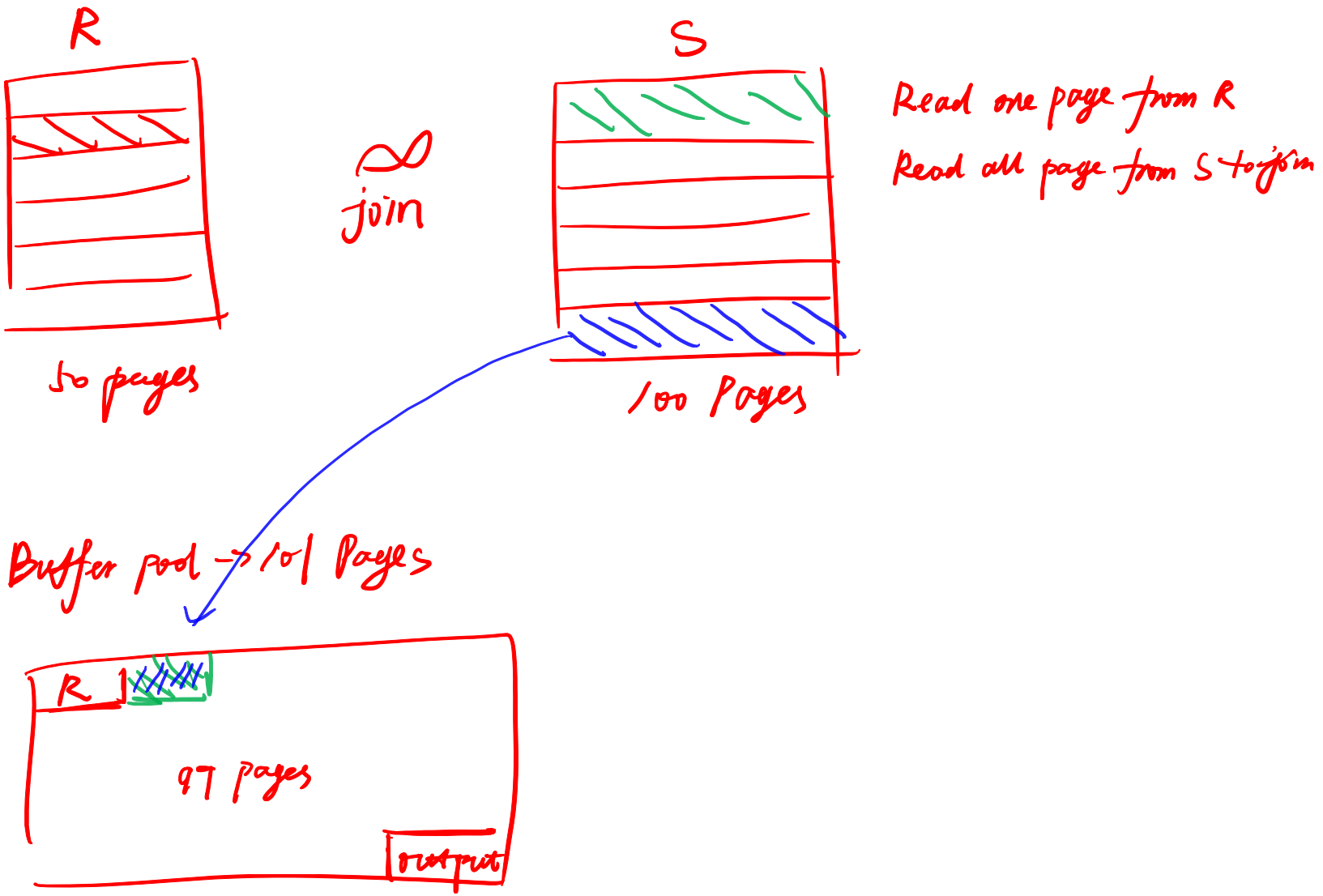
Buffer management: DBMS vs. OS File System

OS does disk space & buffer management – so why not let the OS manage these tasks…?

- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery), and
  - adjust replacement policy, and prefetch pages based on access patterns in typical DB operations.
- DBMS buffer manager can do a better job using knowledge about disk behaviors

Example

- Consider a join of R and S using nested loop
- R has 50 pages, and S has 100 pages
- The buffer pool has 101 pages:
  - 1 used for reading an R page,
  - 1 used for the output
  - 99 used for reading S
- What's the behavior of the LRU policy?
- What's the behavior of the MRU policy?


R
S
join
Read one page from R
Read all page from S to join
50 pages
100 Pages
Buffer pool -> 101 Pages
R
97 pages
output

Topic: System Catalogs

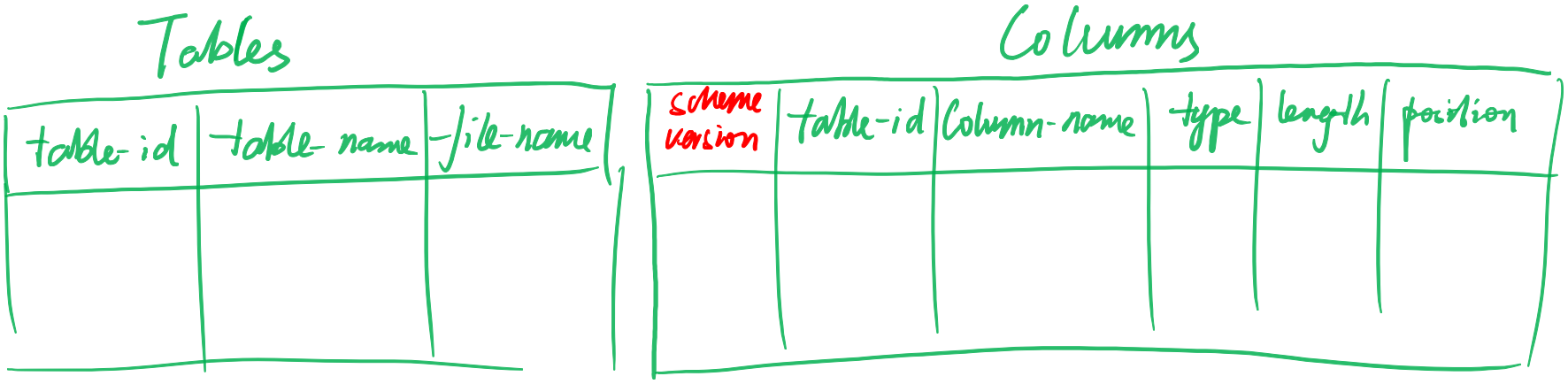
- For each relation:
  - name, file name, file structure (e.g., Heap)
  - name, type, and length (if fixed) for each attribute
  - index name, target, and kind for each index
  - also integrity constraints, defaults, nullability, etc.
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each view:
  - view name and definition (including query)
- Plus statistics, authorization, buffer pool size, etc.

* *Catalogs themselves stored as record-based files too*!

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attr_Cat	string	1
rel_name	Attr_Cat	string	2
type	Attr_Cat	string	3
position	Attr_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3
…	…	…	…

Catalog: Emps (id INT, gender CHAR, name VARCHAR, sal FLOAT)

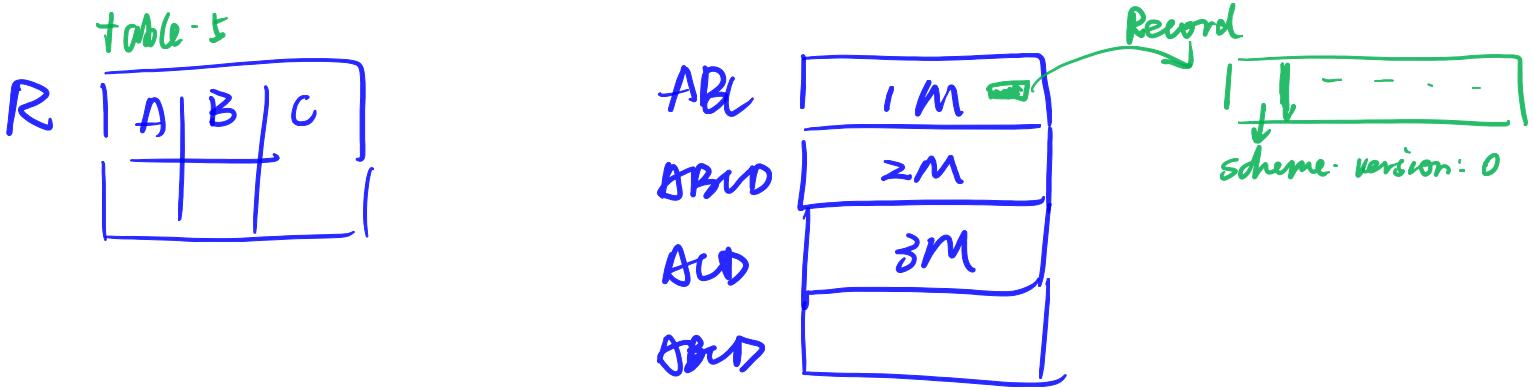


"Table" and "Columns" are fixed tables for DBMS.

IMPORTANT:
"Tables" and "Columns" are also tables that need to store in the Tables and Columns.

Scheme versioning:

R(A, B, C) -> insert 1M records -> add attribute D -> insert 2M records -> Drop Attribute B -> Add 3M records -> Add attribute B(This B is different from the previous B, design a mechanism to differentiate previous B, like position)



Columns

scheme version	table id	name	length	position
0	5			
0	5			
0	5			
1	5			
1	5			
1	5			
1	5			