

Databases Notes SQL Lecture 1-7

8	2024-10-25	- Data storage, I/O, buffering pools, indexes, query processing
9	2024-11-01	- Part 1: NoSQL and cloud databases.
10	2024-11-08	- Part 2: NoSQL and cloud
11	2024-11-15	- Part 1: ETL, big data, data insights
12	2024-11-22	- Part 1: ETL, big data, data insights
NA	2024-11-29	- Thanksgiving
12	2024-12-06	- Overflow - Advanced topics

---

10- 25 Lecture 8

---

**Module 2: DBMS Architecture and Implementation**

- **Bottom-up data modeling:**
  - (1) Analyze the given data sources
  - understand what the data is
  - Understand the schema, the entity sets, relationship, proteus. Does it have generalization or specialization
    - This will provide what the schema of the data
  - (2) device what the data looks like
  - (3) define a 2-D data model → analysis the data warehouse, operational databases
  - (4) Data engineering:
    - write, read pile of data
    - extract transform and load into database
- Two types of applications that interaction with and manipulate databases
  - First: operational → web application → interactive → browser based user interface, logic application, then manipulate data
    - Read, write, create, retrieve updates
  - Second: analysis data → examine data, extract information, make decisions. Data science
- The schema:
  - For operational applications the schema is not best option for analysis applications
  - Most environments have two datasub systems → operational and analysis (read, copy)
  - Event based mechanism: updates to the operational propagated to the actual updates
    - Programming track will do operational databases
- When looking at existing files, the endings tell you what they are. title.askas.tsv.gz.
  - GZ → good new zip → compressed

- TSV → tab separated values → spreadsheet

(1) [What makes something DBMS \(Database management system\)](#)

- 1. User can create database and identity their schema (SQL DDL)
- 2. Give user the ability to query, modify, update data using higher language (DML)
  - Fast, run at apple silicon, amount of disks
- 3. Support large amounts of data over long periods of time
  - credit card companies
- 4. Enable durability: Prevent error
- 5. Isolation: control access to data from many users at once, prevent unexpected interactions among users

(2) [Purpose of Database systems](#)

- In early days, database applications were build on top of file systems this resulted in
  - Data redundancy and inconsistency - data stored in multiple files lead to duplication of into in different files
  - Difficulty accessing data, data isolation, integrity problems

(3) [Disks Input/Output \(IO\)](#)

- Used to Disk is located inside our computer
  - In large databases systems the disk is not without our computer it is on some network
  - The program goes over network to get disk application

(4) [Physical storage media:](#)

- **Volatile storage:** loses content when power is switched off
  - Forgot to save file before shutting down, file was in volatile memory not written into disk
- **Non-Volatile storage:** Data persist even when power is off
  - Includes secondary and tertiary storage
  - Includes battery-backed up main memory
- Factor affecting storage media choice:
  - speed to access data, cost per unit/byte of data, reliability

(5) [Storage Hierarchy: Inside the Computer](#)

- **RAM is Called Random Access**

The term "random access" refers to its ability to directly retrieve any memory address with equal speed, unlike disks that may require sequential access (e.g., HDDs need to physically move the read/write head to the correct location).

1. **Bus (Universal Serial Bus - USB)**

- A specialized communication network within the computer.
- Transfers data, signals, and power between components of the system.

2. **Chips**

- Connected to the bus, enabling communication between hardware components.

### 3. **Network Control**

- Connects the bus to the external network.
- Facilitates communication between the internal system and external entities (e.g., internet or other networks).

### 4. **Disk Control**

- Connects to the bus and manages the disk drives.
- Handles operations like reading and writing data to storage disks.

### 5. **Semiconductor (e.g., DRAM)**

- Provides **non-volatile memory processing**.
- Often refers to memory chips like DRAM (volatile) or flash memory (non-volatile).

### 6. **CPU (Central Processing Unit)**

- The primary processor for memory and computational tasks.
- Only deals with Cash

### 7. **Cache**

- A small, high-speed memory close to the CPU.
- Stores frequently accessed data to speed up operations.
- Data flow: Memory → Disk control → Bus → Semiconductor → out → Bus → Cache → CPU → Cache
  - Data begins disk - data is retained → Disk Control - data transfer from disk into computer memory → Bus connects to → Semiconductor memory won't retain if off, temporary in RAM for fast access → Bus connect to → Cache memory won't retain if off, but faster than RAM → CPU fetches data from Cache

#### Storage Hierarchy:

- Primary storage: fastest media but volatile/not retained (cache, main memory Semiconductor)
- Secondary storage: non-volatile/retain
- Tertiary storage: lowest level, non-volatile, slow access time, for archival storage

#### (6) Hard Disk Drive:

- The disk behavior has fundamental impact on how database system manage data
- Inside vinyl records with gaps between
  - Data is stored magnetically on both side of the metal vinyl
  - It's all spinning very fast
  - Disk system that moves in and out

To find data: what disk, what side, how far is data the in, where on stripe is it
- Move head over track, wait until it spins underneath it

#### (7) Disk Configuration:

1. Seek operation: Move the disk system so its on a cylinder/Track
  - a. Trackers are broken into sectors
2. Rotation: Wait on the sector you want is under the head
3. Transfer: Head and Disk on right location, stream data from disk into computer memory

(8) Flash Storage:

- Hard disk drives and solid state drives look the same, the difference is their behavior.
- Unit of transfer is byte, large. To get single byte you need to read whole block
- On outside, the request of data comes in, you're requesting block ID
- When request for block comes in through hard driver → small processor → buffer of memory that → mechanical controller moves systems
  - You can read any block. To write you zero out then write
  - Read and write blocks are not the same.
- Computer is a processor and main memory. Disk controller and drives is often on networks

(9) Logical/Physical Block Addressing:

- Unit of transfer from a disk to the computer's memory is a block
- To get piece of data, like 1 byte, you can get 1 byte from the semiconductor the Ram
  - But to get a byte from the disk, you can't just get 1, you get unit of transfer which is block, quite large, 16 KB, 32 KB, to get single byte read the whole block, into memory, then get the byte
- When request from block comes through hard drive, there's small processor, and there's a buffer of memory, and a mechanical control that moves the system when the request comes in,
  - Block 27, code process can tell you what cylinder and sector it's on. It can move, because sectors can go bad.
- Solid states, over time block 27 is not in the same place, because you can go and read any block. But to write something, you have to zero out the block, then write the data. You can't write in place. When you read it by in block 27 but writing might be block 64

(10) I/O Architecture:

- Two types of networks that connects processors to disk
  1. Network attached storage
    - a. Disk controller and drives are on the internet, same network as the computer
    - b. The computer instead of going over the bus to the disk controller (NAS device), it a network message to get the data and bring it back
    - c. Packet based for NAS device
  2. Storage area network
    - a. Computer connect to "internet" but its separate network called SAN Switch
    - b. high performance optimized switch
    - c. Connection oriented, build synchronous connection, transfer data, then take down connection

(11) Performance Measures of Disks

- Not all access patterns to disk are the same. A disk block is 4-16 kilobytes
- First thing in storage management is deciding what size of block. There's trade off
  - How it organized the data on the disk depends on two characters:
    - Is access Sequential → but all on single cylinder, and seek once, rest is rotation and transfer
    - Is access random → all over the place, lay info out on disk
    - Mean time to failure → how long will disk work property without failing

- Two types of failure: (1) wear out over time (spinning ones) age (2) human mistake
- In general solid state drives are much better in mean time to failure and better in human mistakes
  - Semiconductor does not have a moving head unlike disk. Less to break

## (12) RAID

- From the outside looks like one disk when closed, when addressing it its one disk, one set of blocks
- When you open it the disk it's broken it smaller physical disks, there's one connection coming in, there's the circuitry
- The data is replicated across it, if one fails there's enough data on the other disks to restore it - redundant
- Many types of RAID
- Raid 0 - piece data stored once, spread across two different disks → single disk, twice fast
  - Logically faster: two seek, transfers in parallel. Rotating in parallel → twice as fast
- Raid 1 - Each piece of data on each disk, duplicated, fails still have all data, redundant
  - Half as big because data is stored twice
- Raids can be combined. → Raid 1 + 0
  - Two Raids - 4 disks - the 4 disks do Raid 1
  - Each pair does RAID 1 for reliability, then do RAID 0 across it for speed
- RAID 5 - one big disk actually 5 smaller disks
  - Cut block data into 4 pieces across 4 disks ( $\frac{1}{4}$  on each disk)
  - The 5th disk is error correction code written
  - Each disk is hot swappable, no need to take of line, plug out and plug in, will rebuild
  - Any disk, no data lost
- Parity and compression: use parity function (page 39)
  - Given arrt and parity bit you can reconstruct lost bit

## NOSQL

- The relational database things terms in table
- Not table, language for NOSQL is different
  - Intenter face one read. Face two its databases other than doc
  - Data exploited (facebook, amazon)
    - Usage pattern changed
    - NOSQL - type of wanted data was not good match for table
    - Enormous performance requirement can't do it in table/relational database because it does not scale horizontal
    - SQL - banking, data processing
    - NOSQL - different conceptual data model, meet new performance demand.
- Lots of database:
  - 1. Table - relational database SQL
  - 2.OLAP: table with different style
- NOSQL - 4 types database
  - 1. Column-family
  - 2. Graph → one fastest → social network → Neo4J
  - 3. Document → herical → MongoDB
  - 4. Key-value

## 1. Document MongoDB

- Difference between relational database and document is that document introduce the concept of attribute type that's list and an object
- Every doc has generated unique key, global unique
- Can have complex attributes
- Document is hierarchical, attributes may be multi-valued and complex
- DBS is set of tables, MongoDB its collections of documents
  - Table - set of rows
  - Collection - set of documents
- [http://localhost:8888/notebooks/Python\\_Projects%2FData%20Bases.ipynb#SQL](http://localhost:8888/notebooks/Python_Projects%2FData%20Bases.ipynb#SQL)

## 11-01 Lecture 9

### Review:

1. A table = collection of documents
2. A row = a document
3. column/attribute = a field
4. a table - files, and the rows- records in No sql
5. Each piece of data in MongoDB is stored as a **document**, and to create a document, you write it in a JSON-like format.

### (1) File Organization:

- The database is a collection of files, there's directory that has all tables, databases stored in files. Opens files, loads data. The file is a sequence of records.
- In a database the software that manages information and files is some kind of record oriented system. SQL -row
- How does it organize the record: each file has records of only one type, the default behavior is a table maps to a file.
- A disk is accessing blocks, not accessing individual bytes. To find a particular row, know the file, open the file, know where in the file the record is. Stand files do not do that, in the database it doesn't read the whole file, it knows which disk which file the record is in, and directly accesses it. Does not need to iterate
- Max block size is 64K.

### (2) Terminology

- 3 types of records:
  - **Fixed length**: in that file, every record is the same length (why can declare size, data type, db engine can compute the block size using the schema)
    - All records are not fixed because it's easier to manage records when fixed size sometimes it'll over allocate space that would not be used. Classic trade-off. Space vs time (faster,

but uses more memory | care about memory, slower)

- **Variable length:** there's a header, and length. This record is the data.
- **Variable format:** s
- Unit of transfer is a disk block, any individual disk block contains multiple records. To know the address of data know what file, what block, and how is the record organized in file → load the block then get record
  - Simplest approach **fixed-length records**
    - Insert: last row
    - Update: read read, change it
    - Deletion: delete a record, hole in file:
      - 1. Move everything up in compact, shift all the data up
      - 2. Keep empty space, when you need to do an insert look for the empty block and write the new data onto the empty slot
      - 3. Don't move them, have a free list that uses linked lines of what's allocated. In file linked list of records with data, and linked-list of empty slots.
- Keep records from two different tables same file → materialized view based on join, db engineer analyzes the queries and detects two tables are never accessed in isolation (always joined) decide to keep them in the same file
- Find columns in find fields in record. Find records in block - not all same size → the block has a header, that's the starting position, array(starting position of each record), map that to a block, get block, then know the record, then it'll get pointer to the location, then get columns uses header in record to get value
  - Block → map to physical block → read block → get header → know record number → use record header
- Blob - binary large object. Clob - character large object
  - Images, scans, pdf used to be stored in the db → clob were for doc, map the char sets. Bloc could be audio-file. There mainly because weren't URLs

### (3) Organization of Records in Files

- File → bunch of blocks → records in blocks → which block to write record → bunch or organization
- Heap: put the record anywhere with space, update set of pointers.
- Sequential: some order is imposed on the records, put a record in, find where it fits in the ordering. Iterate through the records.
  - Assign order to records in file: mostly primary key,
  - by some attribute (last name) → specify the record clustered index to determine order
- Other is B+ trees and Hashing.
- Sequential file organization: the bs engine decide to organize the record in block, checks access pattern to the file/ query pattern, if the queries iterate than itll organize by sequence
  - Ordered by search-key

- Table Partitioning: table → bunch records → decide some records to keep in one file, and some records in another file → have algorithm to determine which file its in
  - File one 1 type storage (semi or raid 5), but two files can be in different locations. If access across file is not uniform then keep them high performance disk and rest on slow disks to optimize storage cost vs performance
- Column - Oriented vs Row/Record -oriented:
  - Record oriented: the columns in a record are kept together, block sequence of records
  - Column oriented: keep columns together not records/rows. → project focused
    - If row has 20 columns , keeping columns together is less i/o operations
    - Module pandas is column oriented

#### (4) Buffer Pool Management

- Block: unit of transform from disk to the computer memory | also unit of stage
  - Every database instance has buffer | segment of memory divided into slots and blocks go into those slots
  - block size unit transfer and database buffer is slot for the block
    - This is pinned in memory by db management
- Can only access what's in main memory (semiconductor)
- Number of blocks that comprised of the aggregate DB (DB is Set of tables, each table maps to files, file set of blocks) so buffer manager applies algorithm to see what to keep in memory or cast back to disk
- To insert block but no empty slot → what block in memory to get rid of → replacement policy
- Replacement Policy: the buffer first will pin and unpin a block
  - Pin block: some query is actively using the block, will not replace, until unpinned
  - Lock block: shared, exclusive locks
- Get rid of block → replacement policy:
  - Default behavior: pick block that is least recently used → LRU
  - Clock Algorithm → maintaining LRU, keep sorted list of time when block was accessed, update each time it's touched (log N). lots pages to keep record sorted by access time is expensive, reduce the cost

#### (5) Index

- To improve performance which which index to create on table
- Index is a special table with only two columns: search-key and pointer



```

create table if not exists db_book.student
(
    ID          varchar(5) not null
        primary key,
    name        varchar(20) not null,
    dept_name   varchar(20) null,
    tot_cred    decimal(3)  null,
    constraint student_ibfk_1
        foreign key (dept_name) references db_book.department (dept_name)
        on delete set null,
    check ('tot_cred' >= 0)
);

create index dept_name
on db_book.student (dept_name);

```

- The pointer is where where in the DB Disk space is referenced, points to the block
  - Three components: disk location, block number on disk, where in block is data

### NOSQL → GraphDB

- Selectivity index

Unset

```

select *
from batting
where playerid = 'williton'
and team='BOS'
'''

```

- two indexes

```

index(playerid) --> index selectivity on average: how many rows with the same
index key value
index(teamid)

```

index finds the blocks, then look through the blocks that have 'bos' and then search for the other index. can pick one index but not both  
pick the most selective index: simple query  
'''

# The most selective index:

```

with one as (
    select playerID, count(*) as non_with_id from batting group by (playerID)
order by non_with_id desc
)

```

```

'''If used playerid fund 31 rows, then check which are for BOS (boston)
to esmabie the selve do the following'''

with one as (
    select playerID, count(*) as non_with_id from batting group by (playerID)
    order by no_With_id desc
)
select avg(no_with_id) from one; # returns 5 rows

'''on average if you used playerid index, look at 5 rows
if the index is not clustered, those 5 rows are all over the place, so that
could translate into 5 disks I/O'''
#CAN DO THE OTHER WAY

with one as (
    select TeamID, count(*) as non_with_id from batting group by (TeamID) order
    by no_With_id desc
)
select avg(no_with_id) from one; # returns 721 rows

# playerID is a more selective index

```

---

## 11-08 Lecture 10

---

### Module 2: DBMS Architecture and Implementation

- different database management system from file base data management: indexes,
  - Buffer, index existed before relational database
  - Breakthrough: declarative language such SQL, computers are inherently TM and don't understand declarative language, must translate it into procedural → query processing
- Work way up in data management
  - Storage manager = disk, lay out on disks, record format
  - Buffer manager = manages main memory, random access memory for efficiency performance
  - index/file/record manger = most impactful to improve performance, select indexes correctly = two types of indexes b+ tree, hash indexes

### Module 2: Data Storage Structure (Database System)

#### (1) [Index](#)

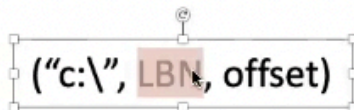
- Indexing is a mechanism to speed up finding and accessing data that a query needs.

- Core concept: an index file for every index created. Index file often is the same file that relation is in. table in file and index often in same file
  - For every index there's index file, and file that contains table: 3 indexes on table, you'll logically have 4 file, the 4th a data file
- The index file loops like a simple table. Every index is a relation with two columns. A search key for the index, and pointer pointing to the correspond record in the record file
- The pointer points to the records and the total addressable space of the storage system
- Logical the storage system are all the blocks over all the files over all the disk of the database system this is managing

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer
------------	---------

- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".



search-key	pointer
search-key	pointer
search-key	pointer

- (which disk is this record is one, which logical block number of disk, the starting position for the data record in that block )
- Two types: index ordered, each index key stored in alphabetically or hash-index
- General considerations:
- (1) what type of query does it support (what goes in the where clause)
  - One supports strict equality: where clause is like  $\rightarrow \text{uni} = \text{'dff9'} \mid \text{last\_Name} = \text{'bob'} \text{ AND first\_name} = \text{'james'}$
  - Second tupe: do you need index that can handle range  $\rightarrow \text{last\_name} \geq \text{'bob'}$ ; AND  $\text{last\_name} \leq \text{'james'}$
- Hash indexes can handle exact matches
- (2) what is access time, the performance of that index, for different queries index have different performance characteristics
  - If you didn't care about storage like cost of disk/ create index for every possible combination of queries

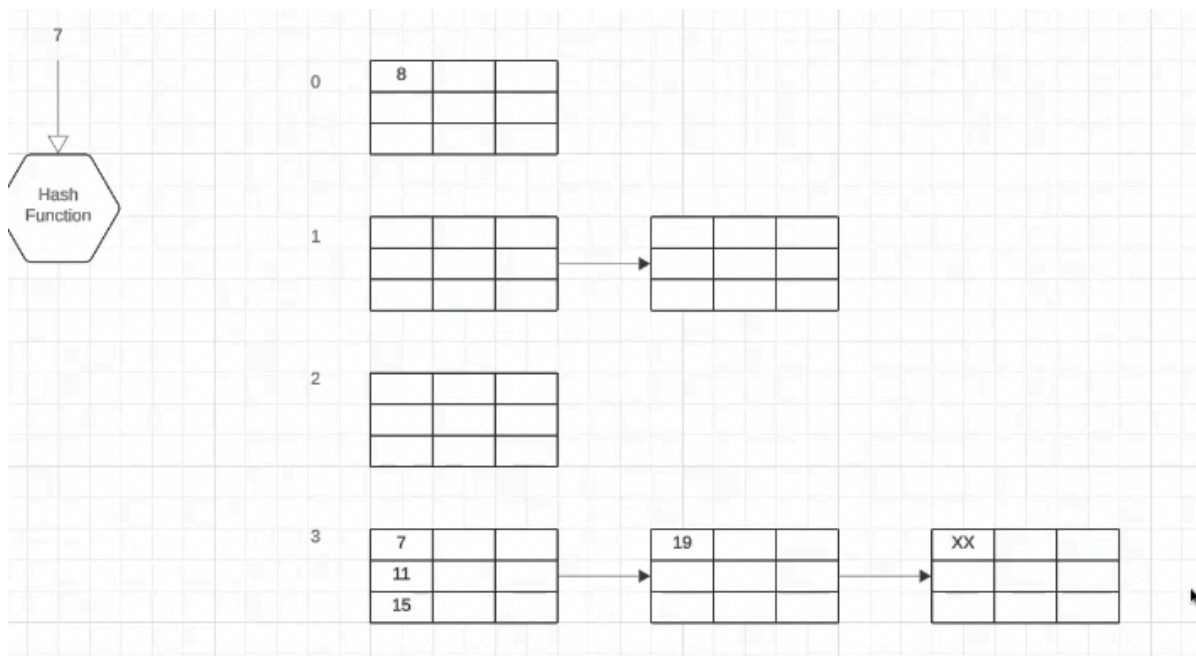
- (3) update perance - if you have unlimited storage, giving index to every query would require to update, inset, delta would require to update row and index. In general index improve reads but slow down insert, upload, delete
- Often chosen by database under the cover
  - Primary vs secondary index: only primary index can be sorted index,
  - Single level vs multilevel : index on index is multilevel
- (4) Two main types of indexes: B+ Tree and Hash

### Index: B+ Tree

- Come in with a search key, several entries, less than go right, greater than go left.
- Search tree with a degree, a spread out that more than two, the reason for that is to get the value in a node in the search tree i have to read a block → make the entire block a node → tree becomes shallower since the degree is higher, depth is less
- First characteristic: The degree is higher
- Second characteristic: when you get to leaves, the leaves are in linked list, each entry points to record
  - The leave nodes are linked and can go in order
- Third characteristic: TRee are skewed not balanced:
- Fourth Characteristic: every leaf node is same distance from the root
  - Each node not leaf or root node has same number of entries, where N is the degree, its ordered, rebalances itself

### Index: Hashing

- Set of buckets, a bucket is a disk block



- Each buckets is pre-allocated and can hold 3 records
- Get request for key to do insert, key value = 7, key gets passed to hash function
- The split hash functions is module or arithmetic

- These blocks all have numbers (0-3)
  - Look up 7 → compute module relative to # of blocks  $7 \text{ Mod } 4 = 3$  → insert 7 in block number 3
  - Look up 8 →  $8 \text{ Mod } 4 = 0$  → insert 8 in block 0
- What happens if block is filled up → insert 11,15 makes the first block of 3 full → overflow block → second bucket for 3 → 19 goes to second bucket 3 → add overflow buckets using linked list
- If these linkedlist get long, not greater at index → 4 buckets, billion records → some point if there's a lot of records → hash behaves like linear scan
  - Make it keys and pointers, pointer can point to the real disk block, the hash requires one I/O to get the right index bucket → tells which disk block to go to → don't need to store the record → save space by storing the key and pointer
- Downsides to hash index: as file grows in size, overflow chains get long, stops being effective, when that happens the database engine stops all access, adds buckets → rehash
  - Now 19 mod 5 buckets → 4 bucket, now you have to re-contribute every index key
- If there's lots of deletion → if size table expands or declines → rebuild hash index → problem 1
- Hash function in theory could skew → problem 2
  - 5 hash buckets → what if all ID ended in zero (100,110,20,1010) structure of key field would wind up going to bucket 0 → hash will skew if it's not good for the type of data → take into consideration type of data and pick good hash function
- Look up time is logarithmic, not worth to organize, disk IO is the dominant factor in performance

Query: where → uni = 'dff9'

Have N record in table

1. Without index average look at half the rows →  $N/2$  →  $O(n)$  linearly time
    - a. Million records, base 10 → 1,000,000
  2. B + tree →  $\log(N)$ , base of the log is degree (# of nodes)
    - a. Million records, base 10 →  $\log(1000000) \rightarrow 6$
    - b. Great lookup, update index search, deletes
  3. Hash:  $O(1)$ , hash find directly → 1
    - a. Downside does not cover all predicates, rebuild hash if size change
- This is the heart of index
  - Main concept: primary vs secondary | clustered vs non-clustered | dense vs sparse | ordered vs unordered | single level vs multi-level

## NOSQL

### (1) [Query Process/Compilation](#)

- Several phases:
  1. Creating the query plan
    - All languages have grammar that determines syntax accurately → parsing and build parse-tree → all languages have parsing phase → First: parse text input
    - Then: Translate the parse into relational algebra expression which is declarative
    - Second: optimize the relational algebra
    - Third: pass that to evaluation engine
    - The computer is TM and wants procedural
  2. Optimizing the query plan → convert to execute
- Query comes in → translate into relational algebra expression → translated into parse tree (nodes are operation like join or table) → Tree (pick operator)

- Select, project, join union, distinct are fixed set of operations, each one has sets of algorithms to pick
  - Select: {SA1,SA2...} what's best for that nod
- Optimal tree depends heavily on the statistics of the data, and how good the indexes are. The optimizer looks at existing data, (SQL declarative lange, when you submit query, the actual query run is not the same same you input b/c optimizer picks best)
- Tree with selection of algorithm is logical plan, what database engine executes
- Can occur parallel with I/O
- Most database have EXPLAIN function → give valid query, returns query executed by optimizer

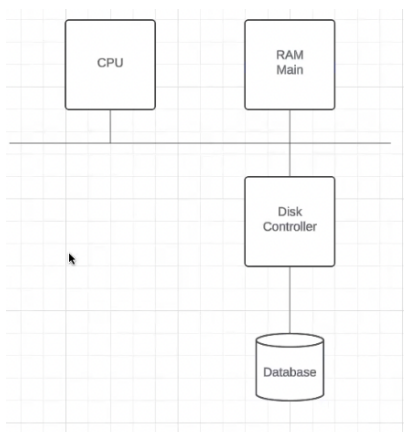
```

1 ✓ explain format=json (with q1 as (
2   select course_id, title, sec_id, semester, year
3   from
4     (select * from course natural join section) as a
5 )
6 q2 as (
7   select * from q1 natural join teaches
8 )
9 select * from
10  (select ID, Name from instructor) as b
11  join q2 using(ID))

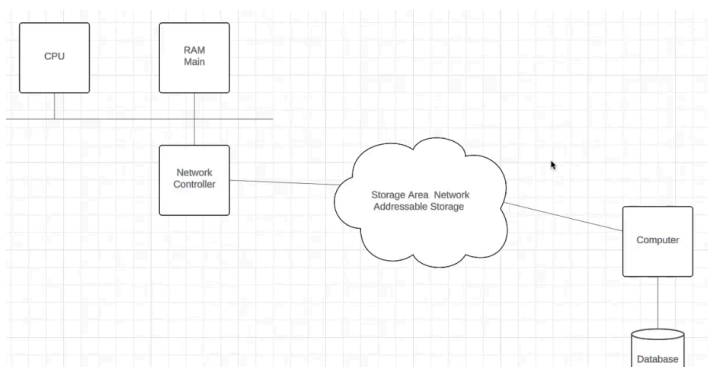
```

## (2) Query Cost

- Network communication matters because in regular storage, there's the CPU and RAM Main connected to the bus and the disk controller and then the actual disk (database) connected to the disk controller physically on the same computer



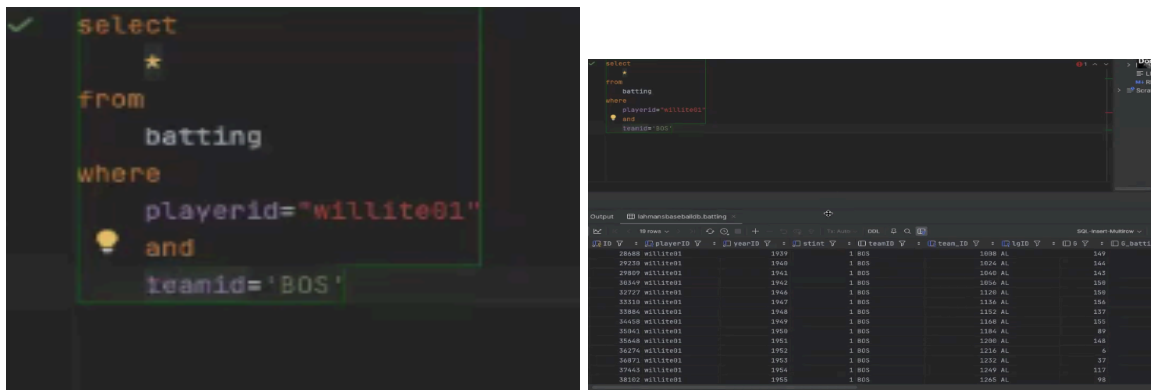
- Different kinds of storage, in very large data centers. The disks are on some kind of network, not physically attached to the processors. Two types of storage. This is common for large environments and systems.
- Somewhere in the data centers there's a bunch of disks



- Concept two: there's two dimensions of performance
  - Response time:
  - Resource consumption:

### (3) Algorithm Selection

- Read each block, then test each record. The cost is number of records / block size, this is linear with respect to number of records.
- If index is clustered, not sure where index block is,
- Predicate and what kind of indexes (often b+ tree, question is clustered or not), in generally same index type
- Next consideration:



The left screenshot shows a SQL query in a dark-themed IDE:

```
select
  *
from
  batting
where
  playerId="willite81"
  and
  teamid='BOS'
```

The right screenshot shows the query results in a table with the following columns: playerId, yearID, stint, team\_ID, and logID. The results list various players and their stints for the Boston team.

playerID	yearID	stint	team_ID	logID
28688	willite81	1937	1 BOS	1088 AL
28729	willite81	1940	1 BOS	1024 AL
29809	willite81	1943	1 BOS	1040 AL
28349	willite81	1942	1 BOS	1008 AL
32727	willite81	1944	1 BOS	1128 AL
32810	willite81	1947	1 BOS	1134 AL
32886	willite81	1948	1 BOS	1152 AL
34458	willite81	1949	1 BOS	1148 AL
28801	willite81	1950	1 BOS	1184 AL
34464	willite81	1951	1 BOS	1200 AL
34276	willite81	1952	1 BOS	1216 AL
34873	willite81	1953	1 BOS	1232 AL
37463	willite81	1954	1 BOS	1248 AL
34182	willite81	1955	1 BOS	1264 AL

- Possible there's two indexes: index on player id, and index on team id
- The selectivity of index on average how many rows are there with same index key value
- If running this query, and have the 2 index, can use either one of them. Index find the blocks, then do the predicate. Can't pick both predicates. Pick **Most selective index**

```

with one as (
  select playerID, count(I) as no_with_id from batting group by playerID order by no_with_id desc
)
select * from one;

```

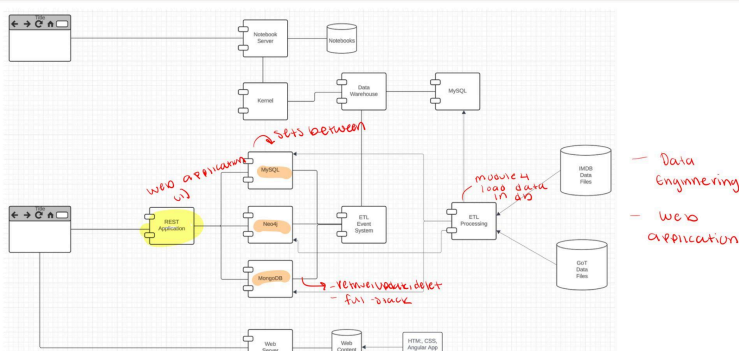
Result 3

playerID	no_with_id
cguid01	31
enderi01	29
ewsobo01	29
ohnnto01	28
aatji01	28
oyerja01	27
nsonca01	27
aineha01	27
arltst01	27
yanno01	27
roscje01	26
eattha01	26
ilheho01	26
quahch01	26

- On average if you use player id index → 5 rows → if index isn't clustered 5 rows can be all over the place → translate to maybe 5 disk → clustered then its 1 block
- Player id is more selective index id, thus pick it

### Homework 3A:

- The particular part write queries, create index, data manipulation and definition
- First thing is programming: some data engineering → bottom up data engineer → given files with data → tell you the schema for two different data base → 1 query db and operational db → load the data into working db, query process it to transfer it to those two schemas





---

## 11-15 Lecture 11

---

- How is queries implemented: update, select, insert
- When a user submits a query it comes in a query language
  - 1. gets parsed and translated, this is standard for all computer languages such as python, c, sql
    - There's a parser that parses the input text, converts it into internal data structure that the language execution involvement (compiler, interpreter) can understand and process
    - For relational database that format is relational algebra expression
      - From RA it builds a tree use the RA  $\rightarrow$  translate to SQL  $\rightarrow$  parse it  $\rightarrow$  relational table
  - 2. Optimizer: converts the relational expression to more efficient expression then convert th

## Module 2: JOIN Algorithm Selection

- Exam: Identity representative set you can understand
- The first decision in select is to check for indexes, are they helpful. Select is mainly about index
- Cost of operations without index: select will need to scan the file (every row/tuple and match the wear condition, that  $O(n)$ , project is the same  $N$  columns return) both are linearly. Union, intersection are also linear
- Expensive operation is join- multiplicative, if Table 1 has  $m$  rows, and Table 2 has  $n$  rows, the cost of it is  $n*m$ .

## Evaluation

- Two approaches
  - 1. Materialization: full execute each operation before executing operation that requires it
  - 2. Pipeline: not wait for downstream operation to complete, as soon as rows are computed will be in result and pass it to next stage in query tree
- In general pipelining is cheaper but not always possible because there's sort of this implication that (piping works fine for sorting) feeding row may not fit the operation pattern you want causes wrong ordering. Sorting is good for pipelining

## Query Optimization

- Concept of equivalent expressions: two relational algebra states are equivalent if they both produce exactly the same table
-

## 11-22 Lecture 12

## Normalization

- First design schema, is it in good form
- **Functional dependencies** are an attempt to articulate set of rules or guiding principles to decide whether data is in good form, what makes schema in a good form - normalization is the form attempt to define good schema
- 1NF - is not often expressed directly, just needs column to be atomic
- 3NF and BCNF are primary about eliminating the risk of unintentionally corrupting the data - get the schema where referential integrity constraints (fk,pk) can protect the integrity of the data
- 3NF think do you have some columns that depend on other columns always -

- `USE classicmodels; Select * from products`

- `productCode s10_1678` → if there's delimiter, dash, underscore its indicator the column/attribute is not atomic domain, the delimiter is there often to split two things up

`Select`

`substr(products.productCode, 2,2) as code_part,`

`substr(products.productScale,3) as scale_end`

`From products; # productScale is depended on productCode`

- Update anomaly : there's implied integrity the user is not enforcing. Implied constraint that's not captured, insert/delete/update produces error if rest of row is not updated manually

`USE classicmodels`

`Select * form customers;`

- All numbers are composite (country code + 6 digits) thus not atomic
- Country column → data instance not consistent such as USA, some pay state US, United States → lost ability to detect

`CREATE TABLE coustmers_fixed as select * from customers;`

`Select * from customers;`

- Modify table, add column named Country\_Code, varchar(4)
- ISO 3166 Country code download
- Import data, named table country\_infromation,

1. `Select * from country_infromation` (alpha\_2 and country\_code is standard identifier)

2. `. Select customerNumber, customerName, country, 'alpha-2'`

`From coustmers_fixed join on country_infromation on country=name;`

3. `Update coustmers_fixed`

`Set country_code =`

`(select 'alpha-2' from country_infromation where`

country\_information.name=coustmers\_fixed.country)

4. Select distinct(country) from coustmers\_fixed where country\_code is Null # issue download has usa as us, so change that to usa

5. Update coustmers\_fixed

Set country\_code='US' . country = 'United States of America' where country='USA'  
(select 'alpha-2' from country\_information where country\_information.name =  
coustmers\_fixed.  
country)

Fix norway now, has space after it

Select \*, length(country) from coustmers\_fixed where country like "%Norway%"; # some

norway len is 6,8,8

s

- No country and country\_code are co-dependent, changing any data from one column makes the other inaccurate. And country\_code is stored in another table, changing it requires change in both tables. Tables have dependencies.
- Change it, get rid of country. And country\_code becomes FK to a PK

### Transaction

- First concept in transaction is demarcation - every sql statement be executed where independent, no relationship between them
- Demarcation - abstract
- Transaction is a grouping of sql statements - sequel begin and commit rollback that brackets {} sql operations
- Durability - when the commit returns or rollback retrusn the database persisted/saved the value - the data is saved to durable storage disk
- Transition - moves data from one consistent state to another. Begin and end is consistent but in the middle not because it's changing data.
- Buffer policy force or no force: force is when a page is updated and committed, forced to disk.
- Steal or no steal: will not take pinned or dirty page for LRU
- When there's a failure such as the db system chases before it resumes transaction processing a family of recovery algorithm get executed
- ARIES Algorithm: default
- use log to get back lost data or data forced onto disk. use raid for lost disk
- Two patterns: active passive → only one zone is active at a time, the other is stand-by mode
- active/active - both are processing at the same time

### Isolation for transaction

- Easy: run one transaction at a time, but not efficient
- Instead implement on concurrency control to run transactions parallel to control the execution of the statement in called order - uses log management concurrency control
- Two types of locks: shared lock and exclusive lock
  - Read block - shared lock, as many
  - Write blok - exclusive lock, only 1
- Request lock, do operation, release lock
- Two-phase locking: request lock then release it, two phase means as long as release the lock cannot request another lock from another page. There's an acquisition phase, as the first lock released can no longer be requested.
- Strict two-phase locking - produces serical schedule, holding all locks until commits. Request commits then release all locks - achieves serializable. And can run many transaction at same time, always produce serializable result - how isolation typically occurs
- Set of control parameters on SQL for isolation. Every sql statement is transaction
- Repeatable reads:
 

```
Begin;
Select * from person;
# terminal
% ./mysql -u root -p
% use f24_examples
% update person set last_name='Smith' where uni='dffl';
# it waits, bc started transition that was serializable, read read, but have shared lock but update requires
exclusive lock cannot run until commit
Commit;
# now update happens
```
- Downside of locking is finding and handling deadlocks

---

### 12-06 Lecture 13

---

- Everything so far has been single database instance, usually one user
- Scalability is property of a system to be able to grow the amount of data it process by adding resources to a system
  - If get more request and data → should be able to add computing resources to handle larger load
- Two approaches to scalability:
  - Scale up: het bigger manger, more processes,
    - Less incremental, just getting bigger machines dispute, need to shut down, more expensive

- Price of hardware increases linearly with capacity (twice as fast twice as much)
- Scale out: when run out of capacity on a machine , get another one. In general scale out is preferred

## Big Data

- The 5 V's of big data:
  - Volume: amount of data, disk space
  - Variety: diversity of data, some can be CSV, JSON, BLOBS. Data has initialized categorized.
  - Is it a structured - well defined pattern? relational is defined with structure
  - Unstruced - bytes like videos
  - Semi-structured - structure figure out by looking but not rigid/consistent
    - MongoDB - semi-structure, not rigid
  - Velocity - rate of generation and change such as updates - credit card companies
  - Veracity - is the data accurate, consistent

## Data Engineering

- Target is lots of source of data and lots of formats into single logical place where it can be refined and analyzed
- Two terms data warehouse and data lake

## Data Analysis and Insight

- OLAP- basic core for reporting. Fundamental concept in OLAP is the fact table.
- First in excel → concept of pivot table, a way of summarizing things
- Extract, transform, load