Relational DB

Oracle was the pioneer

MySQL was the open source version with same features

Primarily designed for transactions in the 1970s

They didn't have a lot of disk space back then. HDDs used to be 8 gigs.

That is why the focus on reducing redundancy & doing normalization.

In early 2000's the demands changed, disk space was pretty cheap.

speed
reliability
scalability
> low storage

Especially true for Google, Amazon, etc companies who worked on internet based application.

Google wanted to index the whole fucking internet, how the fuck would relational stuff be any useful.

Data Structures Used

→ B / B+ trees (optimize disk reads / writes)

Indexing (speed up queries)

Relational stuff was popular till early 2000s but then times changed.

cloud DB

GCP, AWS

does all the stuff

#scalable AF

we just ask for a DB and query the fuck out of it, no worries about stuff. like

→ AAIC uses Amazon RDS for their user data

vs.

self hosted DB.

my own setup.

Have to buy and maintain it.

↓

optimize

backup

→ Need to hire a full time DB admin.

→ companies do it for privacy, security, etc whatever. But BANKS use

relational database services

cloud. It's pretty secure. Even American Defense is planning to move to cloud

their own stuff

Azure & GCP have their own DB products

Relational → Transaction-Driven, Avoid duplication, Save disk space.

SQL is the querying language.
↳ Others have their own additions on top.
↳ for eg Oracle has PL-SQL → for writing scripts.

Flat files
↳ regular files.

→ logs
→ CSV / TSV / JSON → recently popular. (Javascript Object Notation)

JSON → easy to do it with Python.

JSON → text file → data stored as key-value pairs.

→ distributed file system → HDFS (Hadoop & spark)

Too big data.

↓

Distribute data across multiple boxes

→ Since these are flat files, there's no way to optimize or perform indexing

Apache Pig → can do basic preprocessing & querying.

↓

slightly different from SQL

↳ so it's easy to learn

→ no indexing, so, slow queries

It's like switching C++ to Java.

→ writing code for Hadoop & Spark is kinda hard. So Yahoo Research built Pig, a layer of abstraction.

→ Apache Hive → Hive QL
→ similar to SQL
→ has indexing, significantly faster than Pig
→ open source.
→ more widely used in big data application.

sample

```
1 DROP TABLE IF EXISTS docs;
2 CREATE TABLE docs (line STRING);
3 LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
4 CREATE TABLE word_counts AS
5 SELECT word, count(1) AS count FROM
6  (SELECT explode(split(line, '\s')) AS word FROM docs) temp
7 GROUP BY word
8 ORDER BY word;
```

spark SQL
↳ connect to any data source
↳ SQL + spark programs (DataFrames)
↳ best of all.

Quick refresher for JDBC

Java ⇌ JDBC ⇌ Data Source.

spark SQL runs on JVM (because it's in SCALA)
So it can connect to any DB that JDBC can connect to.

→ Hive is fucking fast
↳ use it with Hadoop

→ If using spark, it makes sense to use spark SQL.

They make sense when data is the range of 100's TB - PB

PostGreSQL is used by academic institutions. Why the fuck?
↳ WTF is it anyway.

knowing SQL is a necessity

→ knowing that is enough to pick up any DB

Data Scientist → → No fucks given about internal mechanics, just need to query it.

companies are looking for more students than they currently have.

Data Science, Machine Learning is in demand no doubt.

spark SQL is a SQL abstraction for spark. No need to fuck with spark code.

NoSQL DB will not replace traditional (RDB) ↳ have their strengths.

All DB types have their strengths.

→ RDB is actually the most popular on AWS.

Document Databases

↳ A form of NoSQL

JSON or XML

Each document can be considered a row.

RDB normalizes stuff to save space.

↳ when info is needed, they just use join

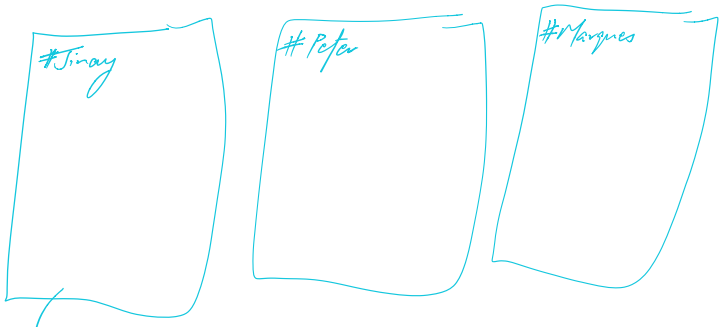
↳ This is compute expensive. even with indexing

RDB optimizes disk space

at the cost of compute.

↳ people don't give a fuck

about optimizing storage nowadays.



↳ just use a document for each user.

↳ It's easy to add more fields for a user down the line

#MongoDB → Document DB

↳ Indexing & Scalability.

↳ Optimized for CREATE
READ

UPDATE
DELETE operations.

pymongo ⇒ for python.

create a (json obj.) and db.insert(json object)

we can modify that json wherever we want.

→ Takes more space
but fewer joins

#MongoDB on cloud.

↳ AWS document DB → gives you MongoDB instance and we just connect to it using our machine. Interface is same but stuff happens on cloud.

Azure Cosmos DB → MongoDB
Cassandra
etc. NoSQL DBs
available.

MongoDB is cool!!
& flexible

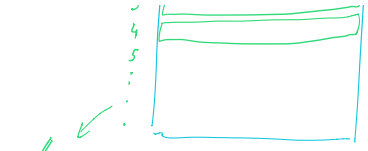
#In memory DB ⇒ Redis & Memcached

↳ In the RAM #speed AF

→ Distributed, so we have enough RAM to work with
→ key-value stores ⇒ Hash Table / Dictionary → in python.

↳ Search is O(1)
#speed.

→ Extreme speed &
low latency ML applications → to store features & weights.



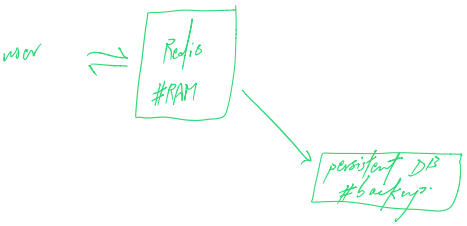
millions of rows

↳ distributed by default

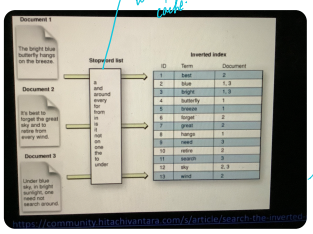
#Database Caching → for search results → cache stuff for redundant queries

AWS Elastic Cache
Azure Cache
GCP cloud memory store

⇒ All memory in RAM → not persistent
⇒ All Redis servers support changing stuff in persistent storage.



Inverted Index (for search)



words are sorted alphabetically.

search query ⟹ bright butterfly

search the inverted index for both words → not the documents themselves

1, 3 1

↳ Document 1 has both.

IRL the inverted index itself is big AF.

→ They distribute it and do some optimizations.

→ Elastic Search → most powerful inverted index

→ Apache Lucene, Solr → also fines
↳ AWS gives instances for this thing.

Timeseries DB

AWS Timestream (recent announcement)

IoT applications → query → transform → fourier transform, etc
predict, ect.

Very less info about this.

Graph DB (AWS Neptune)

Social Networks
Recommendation Systems
Knowledge Graph
Graph-based fraud detection.

Graphs have their unique ops.
So they got their special DB
eg. PageRank.

The whole point of cloud is to reduce hassle.

AWS directly sets up the server & gives the server credentials

(↳ fundamentals are important
knowing this extra stuff doesn't ensure a job offer.

Job tip ⇒ knowledge can always be acquired, mindset cannot be.

Spark is not just distributed dataframes, it is also a compute layer for distributed stuff.