Database

lecture -12

Big Data Analytics with SQL++

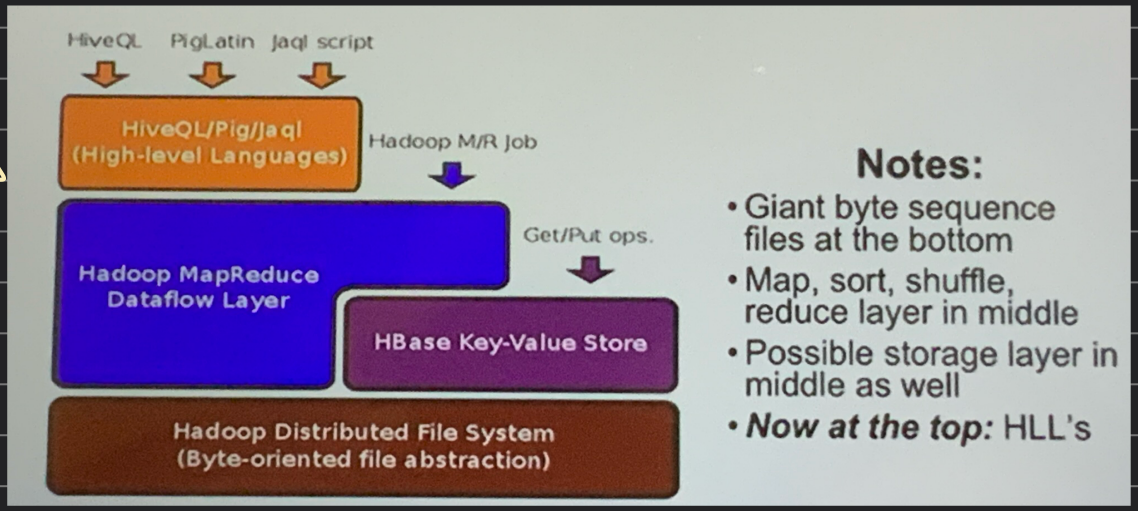
Enterpises needed to store & query historical business data (data warehouse). Late 1990's brought a need to index & query the rapidly exploding content of the web.

→ Google file system.
It Basically allows byte stream files to span hundreds / thousands machines with replication of high availability.
Came up with Map Reduce ; to perform operations parallelly consisting of 3 instruction set machine.

combine Reduce Map

→ Hadoop.
It was the open source version of what google did & has its own file system, MapReduce implementation.

Database Guys:
If you want high-level language, this is not the architecture.



→ Wishlist.

Able to manage data; not just dump it in a file system.
Have a flexible data model, not just have bytes.
Full query capability to run big query's & do declarative queries.
Data to be flowing into a system.
Use best practices for parallel runtime.
↖ hash-joins
little queries take less time to get executed.
Supports Big data datatypes.
↙ structured ↓ semi-structured ↘ un-structured

→ Big Data.

Spark wanted to keep the data stored in the memory & it can be operated faster.
You have a cluster with some number of cores, some amount of memory & attached storage devices, but the data is large enough, so it will be partly in memory & partly not.

→ Asterix Database System

There's basically a bunch of nodes that work together.

Asterix Data Model ← looks like JSON

Think of this as a DDL for describing JSON like data, but it could be other semi-structured data, there are other formats that also kind of match this data model.

Asterix database DDL alternative.

Data types doesn't have to be required. users should have the flexibility to add the data types if they want. And if declared, & someone tries to put something that's not conformat to the type declared then, they will be stopped.

It's called open typing where one can have what's declared but you can have anything else as well.

→ SQL++

The goal of SQL++ is for your SQL skills to carry over to semi-structured data.

One difference from SQL is that SQL has a schema & it could be used to look at the query for execution. SQL++ doesn't know about the schema, & as a result, it says, if you want to join 2 things the safe way of doing is to pair them; but to nest the things down a level so they won't collide.

Another difference is that, in SQL++ the results are nested, not just because these are paired but when the fields are nested the results too will be nested.

SQL++ recognizes that not everything has to be an object.

In SQL data fields maybe null but in SQL++ things can be null or they can be missing.

→ Unnesting

Using unnest keyword.

Use : (eg: o. items i)

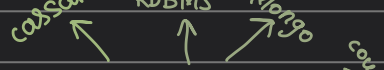
→ SQL++ Grouping & Aggregation.
It seperates aggregation from grouping & you can combine them.

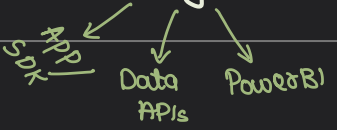
→ Analytics features.
SQL++ allows rollup, so you can create hierarchies, sort total number of stores.

→ Capella Columnar
JSON → Relational tables
→ NoSQL document collection
→ CSV & TSV files.
Think of it as a data model that encompasses relational
← special kind of JSON where everything is flat

Capella Columnar is a database service offering that deal with JSON data. It brings the knowledge from the warehousing world all the way back to gamma, to apply to JSON. Involve a mix of collections that don't change with collections that are changing.

→ Key Features
It supports SQL++, parallel query evalution.
Native columnar storage for efficient JSON analytics.
Cost based query optimization for columnar SQL++.
Business Intelligence tool integration.

→ Capella Columnar Services.


There's data coming in from different sources & is getting streamed into the columnar Service, to ask questions about the data & get answers based on the data thats currently in the operational database
On the right side you can talk to the system in various ways.


→ Enterpise IT on Capella Columnar.
You have different Sources of data, all this data is the Stuff that you'd like to exploit, analyze & understand. Data warehouse allows you to move your data either through ETL or dumping into a data lake, and the visualize the data or perform data science.

Analysts are waiting for the change / update to happen be it some hours or days.

Therefore, capella Columnar moves a bunch of those questions that they might want to ask to real time, & not have to wait.
Reducing the time to insight.

One of the things you can do with the couchbase columnar is to take your JSON data & use tools like Tableau, Power BI to analyze data.

When running a query, it uses all the compute units to process the query in parallel, projections & selections, Parallel joins, Sorting, grouped aggregation & windowing. All compute units are allocated an equal share of files to process.

→ Massively Parallel processing
The first thing that will happen is each of the compute units will carry out partial aggregation & then data will get reshuffled. Its all based on hashing & then the final answer is then aggregated there.

→ Storage / compute separation.
What people like to do today is not pay for compute they are not using. Moved from asterix DB, which was designed for a basement full of node with storage devices to a world where storage & compute are separate.

Each node operates with a partioned local storage strategy. Data is partitioned using a primary key for hashing, ensuring efficient distribution across nodes. A primary index links the PK to the record while secondary indexes map secondary keys (SK) to the PK for faster lookups. Record updates are localized to the node where the data resides, optimizing performance & reducing inter-node communication. This design balances scalability with efficient local data access.

→ Columnar storage for JSON data.
It address challenges like data bloat, heterogenity & nested structures by automatically inferring flexible schemas as the data evolves. The approach separates the schema from the data to minimize size, stores data in an efficient binary format & utilizes LSM component lifecycle for optimization. JSON collection are stored column-wise, generalizing columns to handle nesting & heterogenity while compressing & encoding them to reduce I/O & storage requirements.

→ Cost-based optimization for SQL++.
It uses sampling-based approach to estimate result cardinality & size, periodically creating small samples of each collection cardinalities. This approach targets complex predicates such as functions, correlations & helps in refining query execution plans. The optimizer then employs sample-based decision-making to determine the best query plan.

→ External Collections.

It allows seamless interaction with data in external object storage like S3, making external data appear as internal collections. They support multiple files formats such as JSON, CSV, TSV which enables flexibility. Dynamic paths & filters facilitate selecting specific folders & converting folder structures into queryable data. DML provides to read & write data including commands like copy from to load data from S3. Create collection AS for querying external data & copy to for structured external data storage.

→ Streaming data using Data change protocol (DCP)

It involves using protocols like DCP & tools like kafka for near real-time data-processing. With the DCP, data from Couchbase service is streamed, leveraging remote links to create a live shadow of source data using capella columnar. kafka, on the other hand, connects remote database like RDBMS, Mongo, acting as an enterprise-level DCP. By leveraging kafka connectors, it also produces a near real-time data shadow enabling seamless integration and real-time updates across various systems.

→ Couchbase's capella Columnar service

It provides a highly scalable platform for JSON data integration & analytics. It uses SQL++ for easy adoption & support efficient MPP-based query evaluation with native columnar storage. The service separates compute & storage for optimized cost & performance & support continuous data ingestion via DCP & kafka. Additionally, it offers BI tool integration & optimized access to external object storage. The private preview began in December 2023, & the service officially launched in september 2024.

JSON

The idea is to have data in name/value pairs & data is separated by commas

to reorganize the JSON DOCs

Pivot.

It takes the data & turns it into attribute. Pivot turns a nested collection of objects into a record where there is only one object

Unpivot.

we are doing reverse of pivot operations; here we are using attribute names as if they were data, it constructs multiple objects from one object