SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade Approved
by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER APPLICATIONS

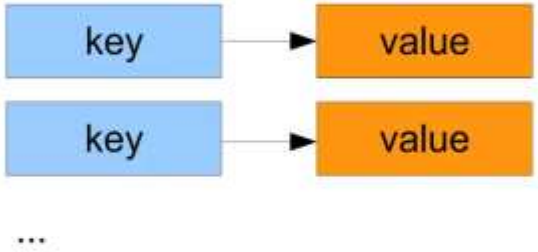
19CAE730 – Fundamentals of NOSQL database System
II YEAR III SEM

UNIT II - **COLUMN ORIENTED DATABASE**

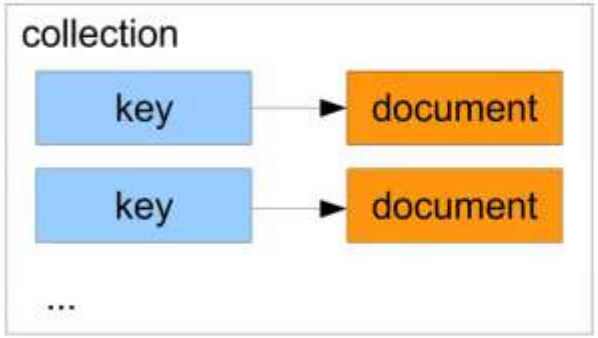
TOPIC 1 & 2 - Comparison of relational databases to new NoSQL stores,

Types of data stores

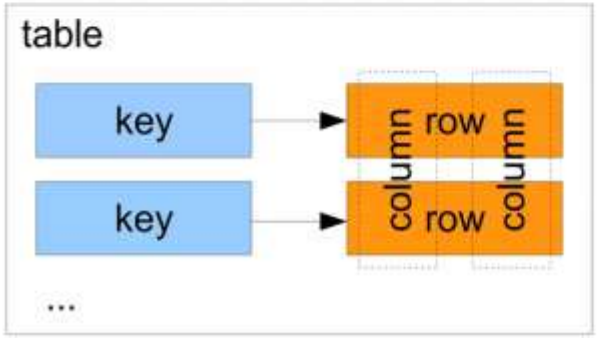
Key / value stores (opaque / typed)



Document stores (non-shaped / shaped)

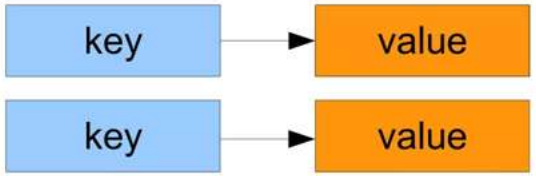


Relational databases



Key / value stores (opaque)

- Keys are mapped to values
- Values are treated as BLOBs (opaque data)
- No type information is stored
- Values can be heterogenous

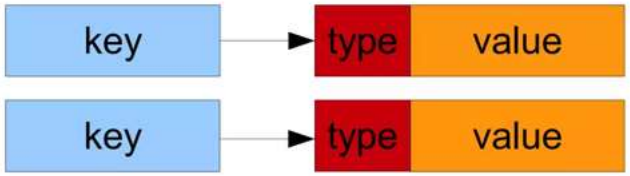


Example values:

- { name: „foo“, age: 25, city: „bar“ } => JSON, but store will not care about it
- \xde\xad\xb0\x0b => binary, but store will not care about it

Key / value stores (typed)

- Keys are mapped to values
- Values have simple type information attached
- Type information is stored per value
- Values can still be heterogenous

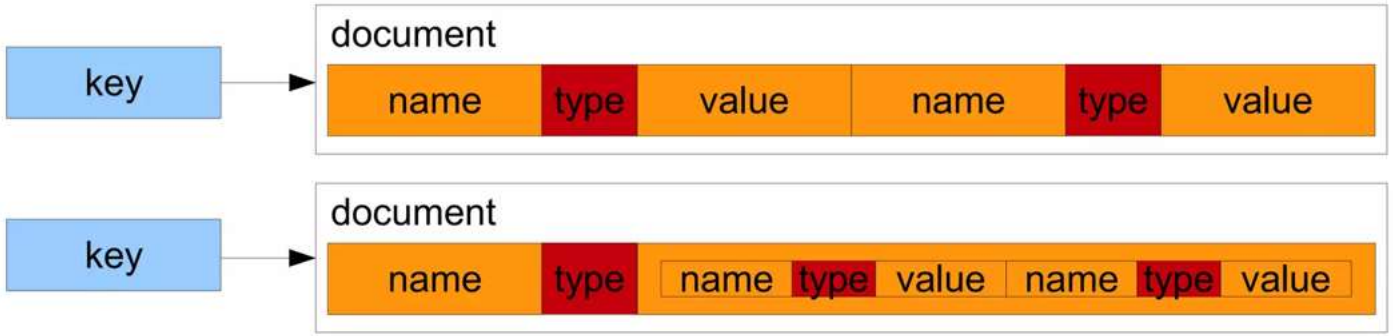


Example values:

- number: 25 => numeric, store can do something with it
- list: [1, 2, 3] => list, store can do something with it

Document stores (non-shaped)

- Keys are mapped to documents
- Documents consist of attributes
- Attributes are name/typed value pairs, which may be nested
- Type information is stored per attribute
- Documents can be heterogenous
- Documents may be organised in collections or databases

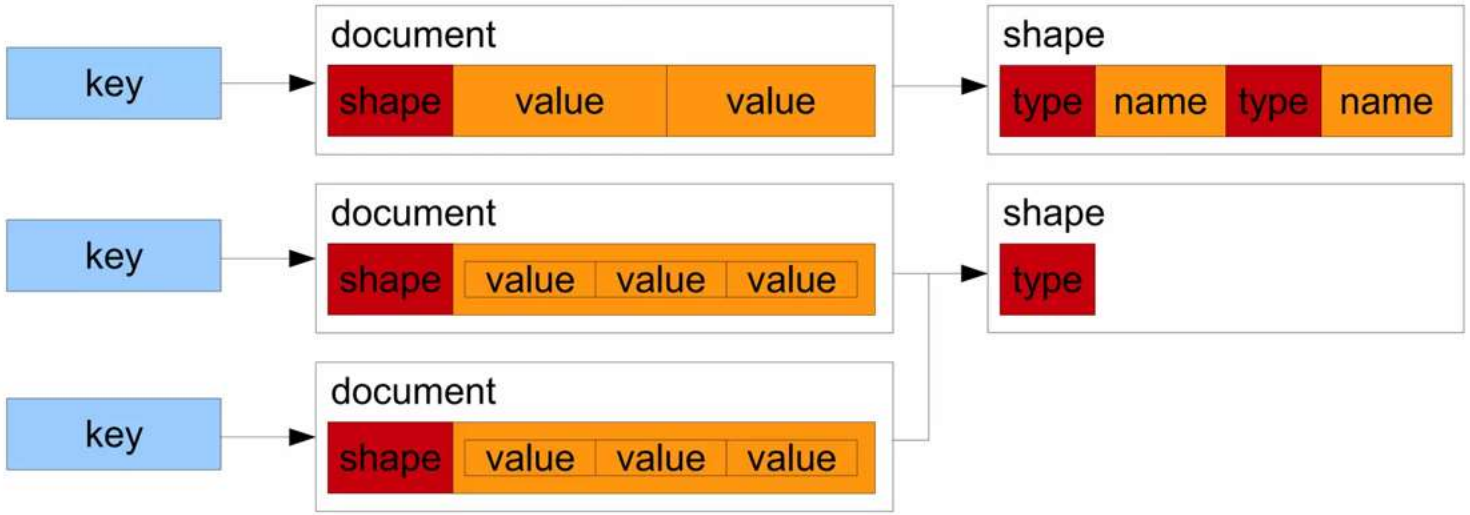


Example documents:

- { name: „foo“, age: 25, city: „bar“ } => attributes and sub-attributes
- { name: { first: „foo“, last: „bar“ }, age: 25 } are typed and can be indexed

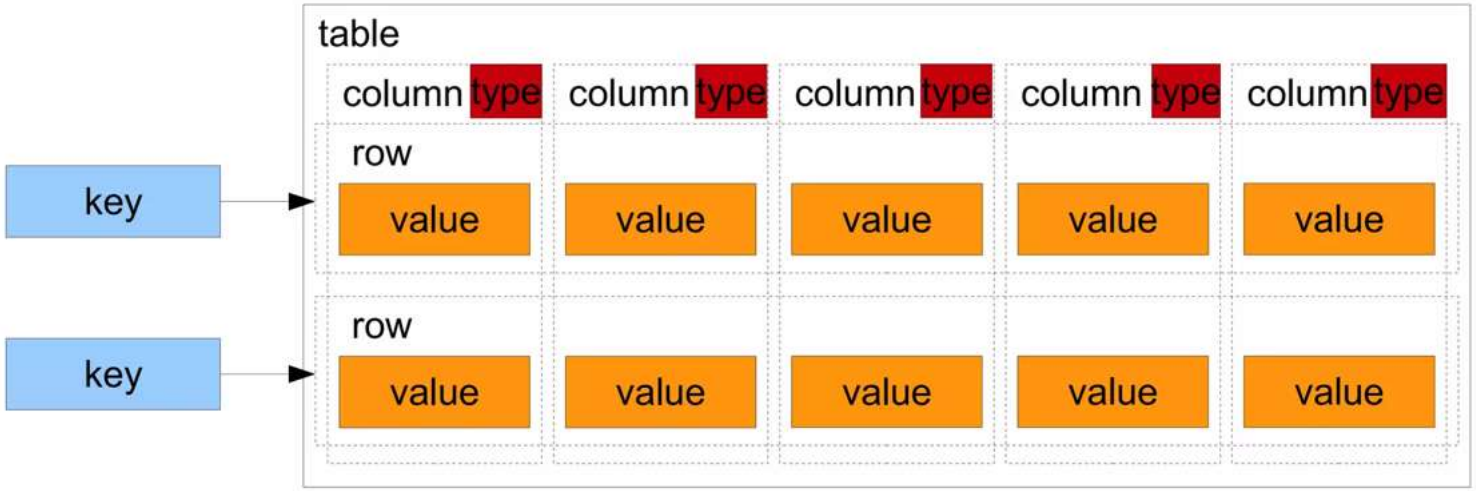
Document stores (shaped)

- Same as document stores, but...
- ...document type information is stored in shapes
- ...documents with similar structure (attribute names and types) point to the same shape



Relational databases

- Tables (relations) consist of rows and columns
- Columns have a type. Type information is stored once per column
- A rows contains just values for a record (no type information)
- All rows in a table have the same columns and are homogenous



Example rows:

- „foo“, „bar“, 25, 35.63
- „bar“, „baz“, 42, -673.342

Data stores, summary

- Documents in **document stores** can be heterogenous:
  Different documents can have different attributes and types
  Type information is stored per document or group of documents
- Values in **key / value stores** can be heterogenous:
  Different values can have different types
  Type information is either not stored at all or stored per value
- Rows in **relational databases** are homogenous:
  Different rows have the same columns and types
  Type information is stored once per column

- Which data store type is ideal for the job depends on the types of queries that must be run on the data!

Query types: OLTP

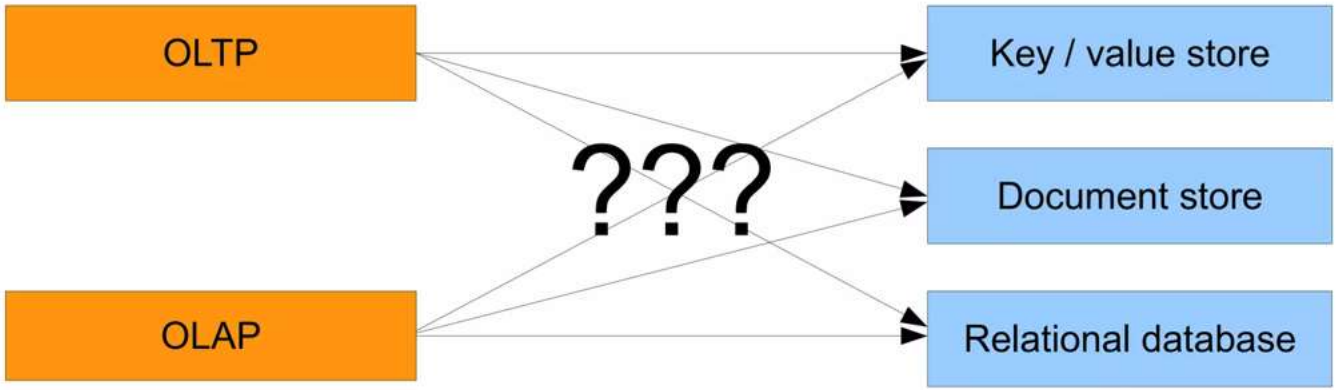
- „transactional“ processing
- retrieve or modify individual records (mostly few records)
- use indexes to quickly find relevant records
- queries often triggered by end user actions and should complete instantly
- ACID properties may be important
- mixed read/write workload
- working set should fit in RAM

Query types: OLAP

- analytical processing / reporting
- derive new information from existing data (aggregates, transformations, calculations)
- queries often run on many records or complete data set
- data set may exceed size of RAM easily
- mainly read or even read-only workload
- ACID properties often not important, data can often be regenerated
- queries often run interactively
- common: not known in advance which aspects are interesting so pre-indexing „relevant“ columns is difficult

OLTP vs. OLAP

- properties of OLTP and OLAP queries are very different
- most data stores are specialised only for one class of queries
- where to run which type of queries?



OLAP in relational databases

- Data in relational databases is fully typed
- Nested values can be simulated using auxilliary tables, joins etc.
- Most relational databases offer SQL which allows complex analysis queries
- Type overhead in relational databases is very low, allowing fast processing of data
- Overhead for (often unneeded) ACID properties may produce a signifcant performance penalty

Row vs. columnar relational databases

- All relational databases deal with tables, rows, and columns
- But there are sub-types:
  - row-oriented: they are internally organised around the handling of rows
  - columnar / column-oriented: these mainly work with columns
- Both types usually offer SQL interfaces and produce tables (with rows and columns) as their result sets
- Both types can generally solve the same queries
- Both types have specific use cases that they're good for (and use cases that they're not good for)

Row-oriented storage

- In row-oriented databases, row value data is usually stored contiguously:

row0 header	column0 value	column1 value	column2 value	column3 value
row1 header	column0 value	column1 value	column2 value	column3 value
row2 header	column0 value	column1 value	column2 value	column3 value

(the row headers contain record lengths, NULL bits etc.)

Row-oriented storage

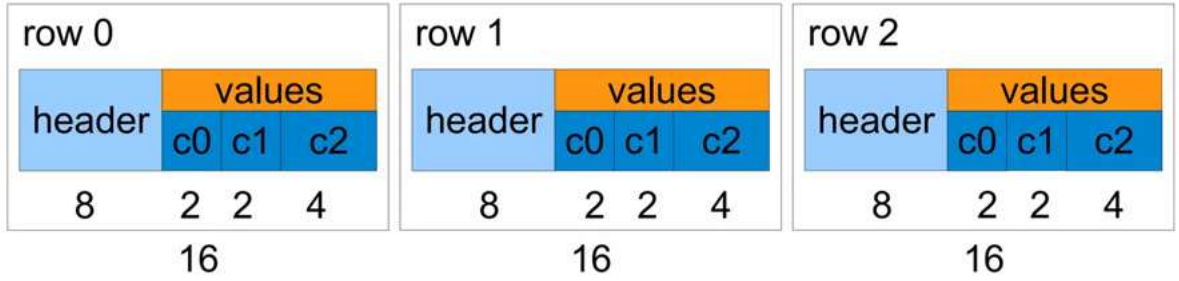
- When looking at a table's datafile, it could look like this:

- Actual row values are stored at specific offsets of the values struct:

- Offsets depend on column types, e.g. 4 for int32, 8 for int64 etc.

Row-oriented storage

- To read a specific row, we need to determine its position first
- This is easy if rows have a fixed length:
  - position = row number * row length + header length
  - header length = 8
  - row length = header length (8) + value length (8) = 16
  - value length = c0 length (2) + c1 length (2) + c2 length (4) = 8



Column-oriented storage

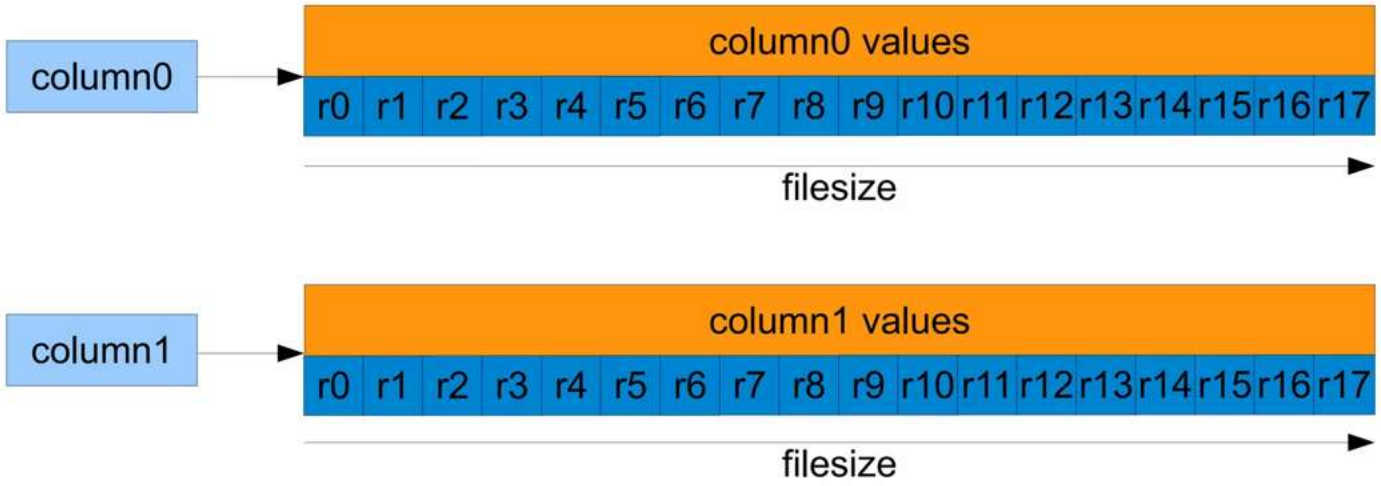
- Column-oriented databases primarily work on columns
- All columns are treated individually
- Values of a single column are stored contiguously
- This allows array-processing the values of a column
- Rows may be constructed from column values later if required
- This means column stores can still produce row output (tables)
- Values from multiple columns need to be retrieved and assembled for that, making implementation of bit more complex
- Query processors in columnar databases work on columns, too

Column-oriented storage

- Column stores can greatly improve the performance of queries that only touch a small amount of columns
- This is because they will only access these columns' particular data
- Simple math: table t has a total of 10 GB data, with
  - column a: 4 GB
  - column b: 2 GB
  - column c: 3 GB
  - column d: 1 GB
- If a query only uses column d, at most 1 GB of data will be processed by a column store
- In a row store, the full 10 GB will be processed

Column-oriented storage

- Column stores store data in column-specific files
- Simplest case: one datafile per column
- Row values for each column are stored contiguously



Difference between Relational database and NoSQL :

Relational Database	NoSQL
It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
It gives only read scalability.	It gives both read and write scalability.
It manages structured data.	It manages all type of data.
Data arrives from one or few locations.	Data arrives from many locations.
It supports complex transactions.	It supports simple transactions.
It has single point of failure.	No single point of failure.
It handles data in less volume.	It handles data in high volume.
Transactions written in one location.	Transactions written in many locations.
support ACID properties compliance	doesn't support ACID properties
Its difficult to make changes in database once it is defined	Enables easy and frequent changes to database
schema is mandatory to store the data	schema design is not required
Deployed in vertical fashion.	Deployed in Horizontal fashion.


Thank
you