Datascience

05 - Databases and SQL

Dávid Visontai
ELTE, Physics of Complex Systems Department
2022.03.16.

Database server main operations

- Organized data storage in non-volatile memory
  - Data are indexed
- Execute queries
  - Optimize memory, IO and CPU during query execution
- Data modification
- Transaction processing
  - Long-running, concurrent execution of atomic operations
- Maintain data consistency
- Durable storage of data
  - Must survive partial system crash

The Types of Modern Databases

- Relational or non-relational

the popularity of non-relational databases is on the rise!

The choice depends on:

- Type of the data
- Structure of the data
- Data model
- Data store
- Use-case of your data (what are typical queries?)

RDBMS

Relational databases emerged in the 70's to store data

- Originally for business applications

It is a collection of **tables** with **schema** that represents the fixed attributes and data types

Using **Structured Query Language** (SQL) statements RDBMS provide functionality for

- Reading
- Creating
- Updating
- Deleting data

ACID

- All RDBMS are **ACID-compliant**: **Atomicity, Consistency, Isolation,** and **Durability.**

- Data modification consisting of multiple steps
- Many transactions can be done concurrently

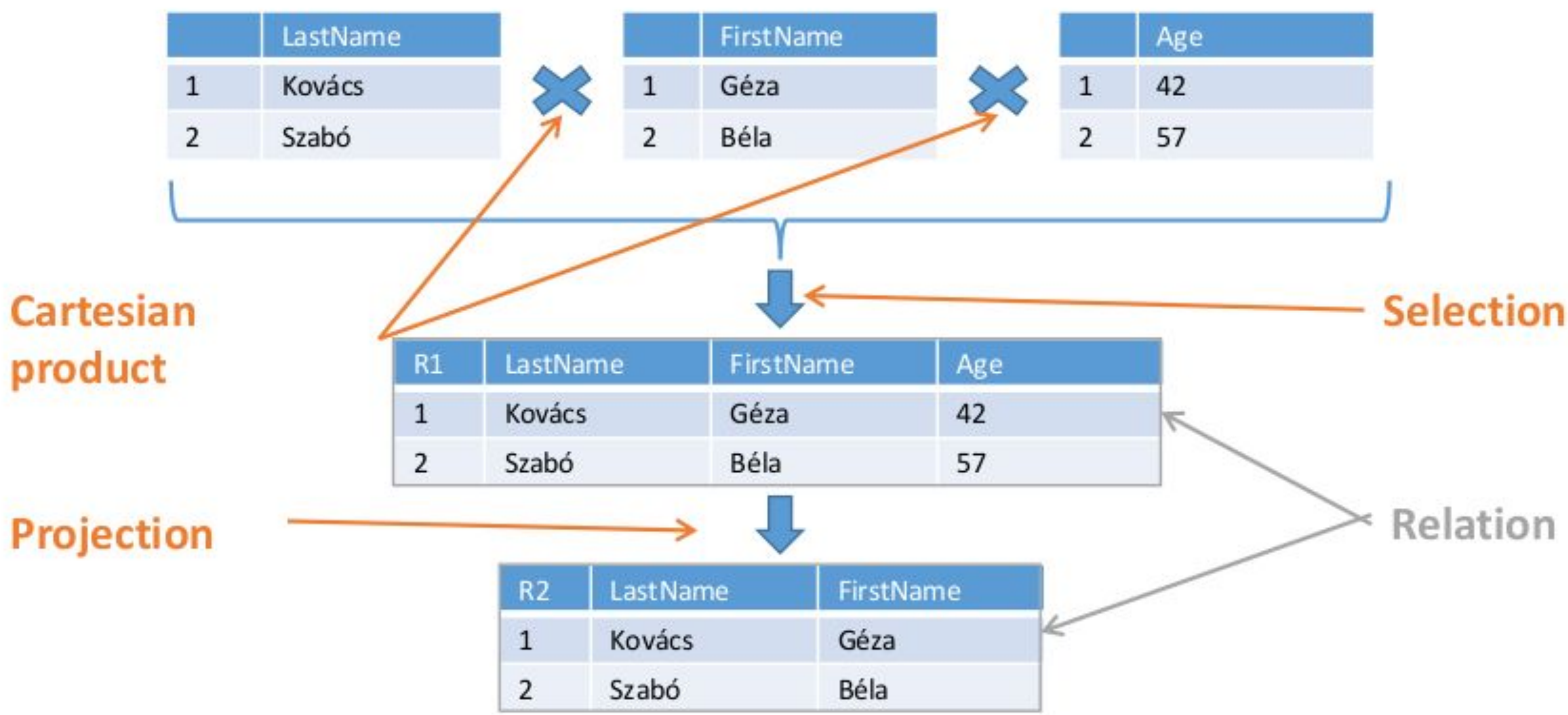
- **Atomicity**
  Transaction either finishes completely or nothing happens at all
- **Consistency**
  Transactions can take the database from consistent state to another consistent state only
- **Isolation**
  Concurrently running transactions might interfere with each other but only up to a certain, user-defined limit
- **Durability**
  Once a transaction is reported to be complete, it cannot be rolled back

Relational Data Model

- Set
  - One column of a table
  - Tuple of columns
- Operations
  - Cartesian product
  - Selection
  - Projection
  - Union, difference
- Schema
  - Constraints on which Cartesian products of the sets in what order can be taken
- Relation
  - Subset of a Cartesian product

Relational algebra

Tables with schema



Relational algebra

Product of tables

ID	LastName	FirstName	Age
1	Kovács	Géza	42
2	Szabó	Béla	57

ID	Author	Title
1	1	Könyv 1
2	1	Könyv 2
3	2	Könyv 3

LastName	FirstName	Title
Kovács	Géza	Könyv 1
Kovács	Géza	Könyv 2
Szabó	Béla	Könyv 3

Logical data storage in SQL servers

- Tables are collections of data rows
- Table defines row format
- Data types
  - Fixed size: int, bigint, real, float, char(20), binary(250) etc.
  - Variable size: varchar(50), varbinary(250) etc.
  - BLOB (binary large object) text, ntext, varbinary(max), image etc. 2 GB maximum

Tables

- Table is the fundamental data storage entity of a database
  - Predefined set of columns
  - Any number of rows

Authors			
ID	LastName	FirstName	Age
1	Kovács	Géza	42
2	Szabó	Béla	57

- Column
  - Name
  - Data type: number, text, etc.
  - Variable/fix length for text
  - Constrains: unique, interval, auto increment etc.

- Row
  - Any number of rows in each table
  - Same data structure for each row

Constraints

- Primary Key
  - Unique identifier within a single table
  - Single column or combination of columns
- Foreign Key
  - A value in a column point to an ID in another table
- A single column or combination of columns
- Must be unique in the entire table
- Well-defined ordering
  - Multi-column: lexicographical ordering
  - Specify order direction for each column independently
  - Any two keys should be comparable: < or >
- Simplest key: incrementally generated integer ID

Authors			
ID	LastName	FirstName	Age
1	Kovács	Géza	42
2	Szabó	Béla	57

Books		
ID	Author	Title
1	1	Könyv 1
2	1	Könyv 2
3	2	Könyv 3

Primary key, foreign key

Emberek	
ID	int
Vezetéknév	nchar(50)
Keresztnév	nchar(50)

Könyvek	
ID	int
SzerzőID	int
Cím	nchar(250)

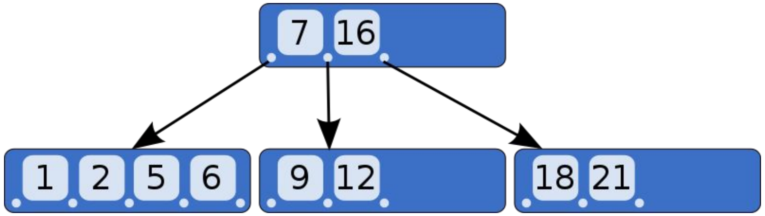
Indices

Indices allow logarithmic-time access to data

- A table can have only one clustered index
  - Determines the storage order for the entire table
  - Search and sort only by the primary key

B-tree and B+-tree

- Data structure to store ordered data
- Nodes: d data rows, d + 1 pointers
- Pointers point to additional rows



- When nodes are full, we split them into two
- Tree traversal using recursive algorithms
- Finding an item by key: o(logd)
- Tree can be scanned according to the ordering defined by the key (or in reverse order)
- Database servers: B+-tree
  a. only store keys in intermediate nodes
  b. Leaf nodes store actual data

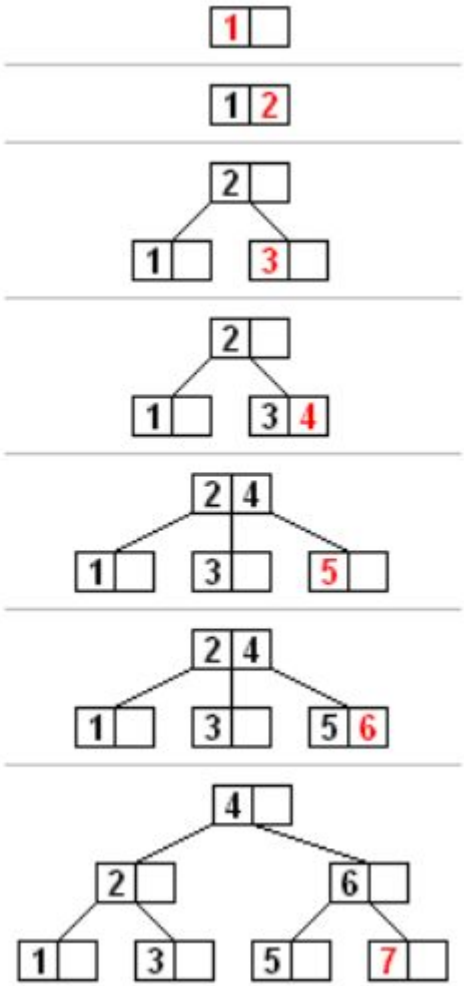
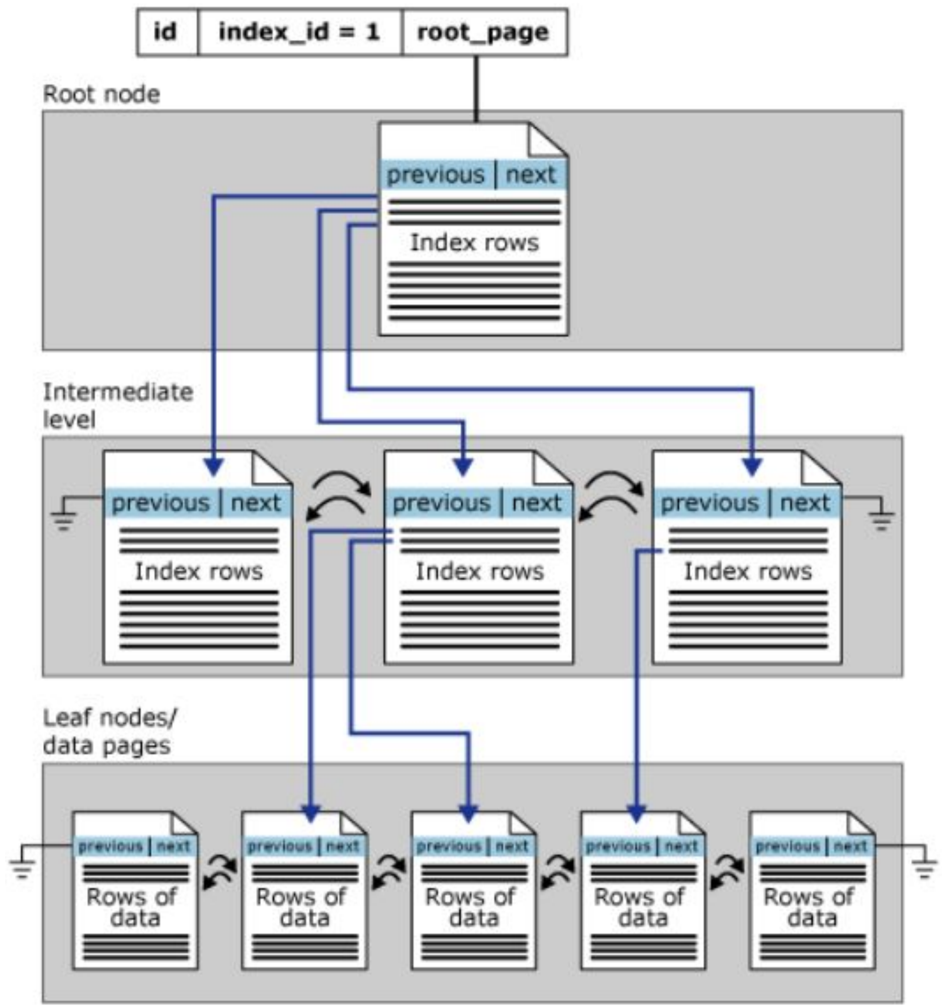


Table with clustered index

- Table stored as a B+-tree
- Tree node = data page
- Two page types
  - Index page
  - Leaf node page storing rows
- Pointers to prev/next page
  - Support sequential scan



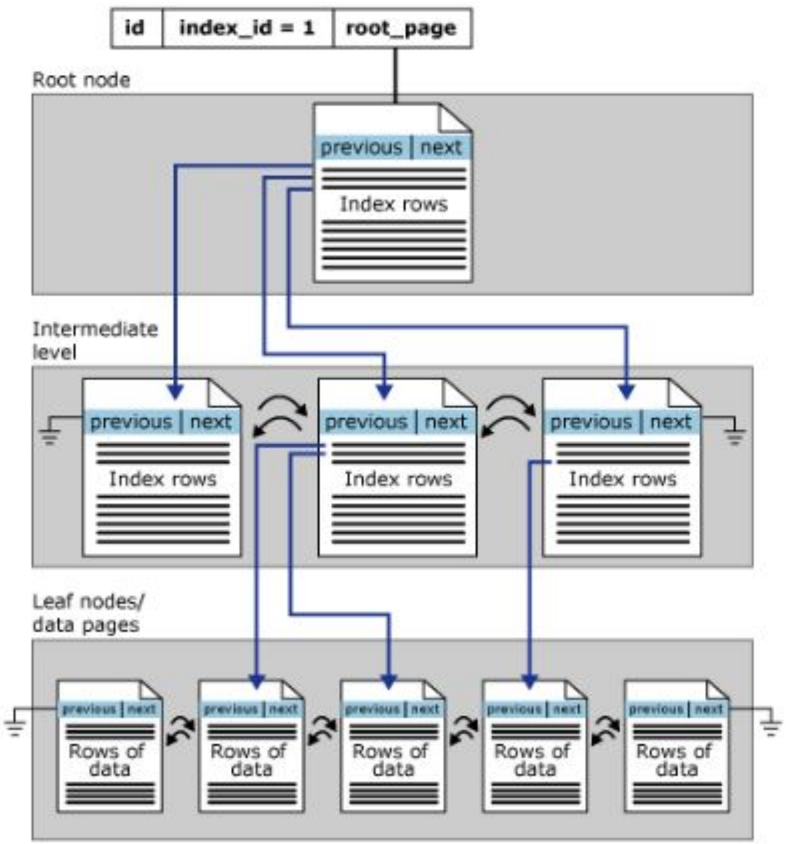
Advantage of clustered index

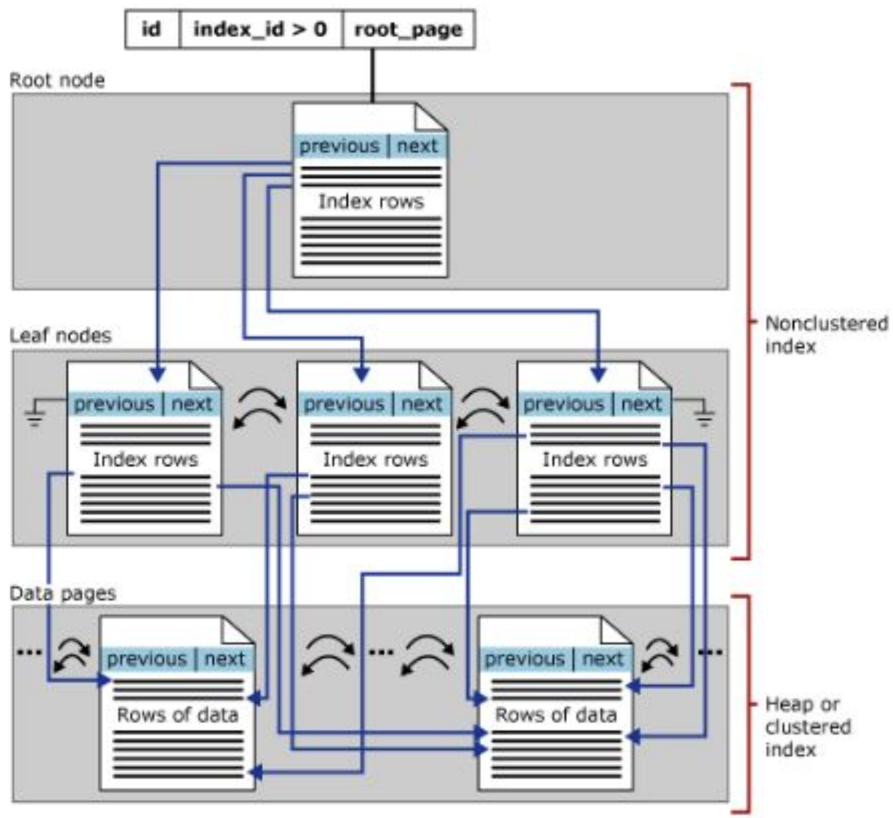
- Find row by key in n*log(n) time
  - SELECT * FROM t WHERE ID = 12
- Scan ID range very quickly
  - SELECT * FROM t WHERE ID BETWEEN 12 AND 36
- Read table in the order of the key (or in reverse)
  - Doesn’t require re-sorting!
  - SELECT * FROM t ORDER BY ID

Non-clustered indices

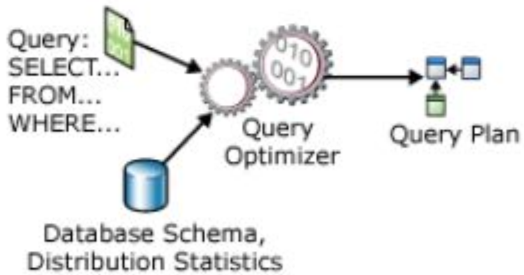
- What if we want to search by other columns?
- Additional data structure next to the table
- Built on one or more columns with given ordering
- Contains only indexed columns
- Pointers to data pages of the clustered index
- Can be unique
- Can implicitly contain non-indexed columns

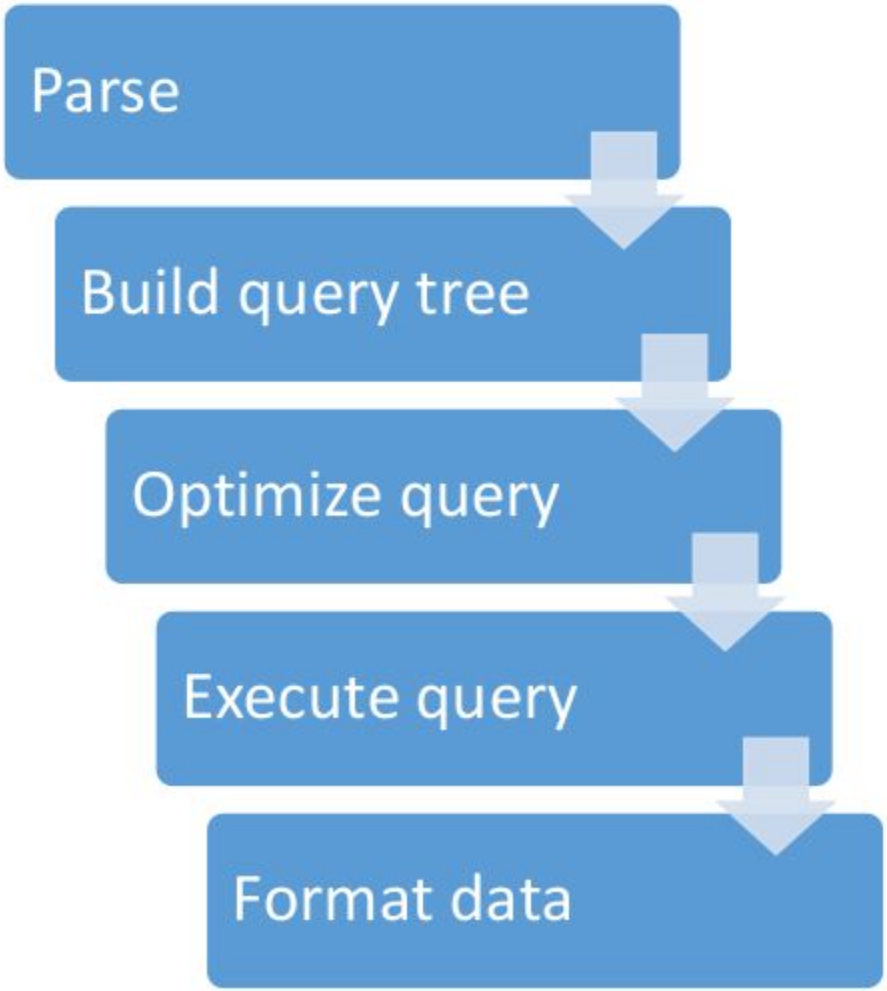
Clustered vs. Non clustered indices





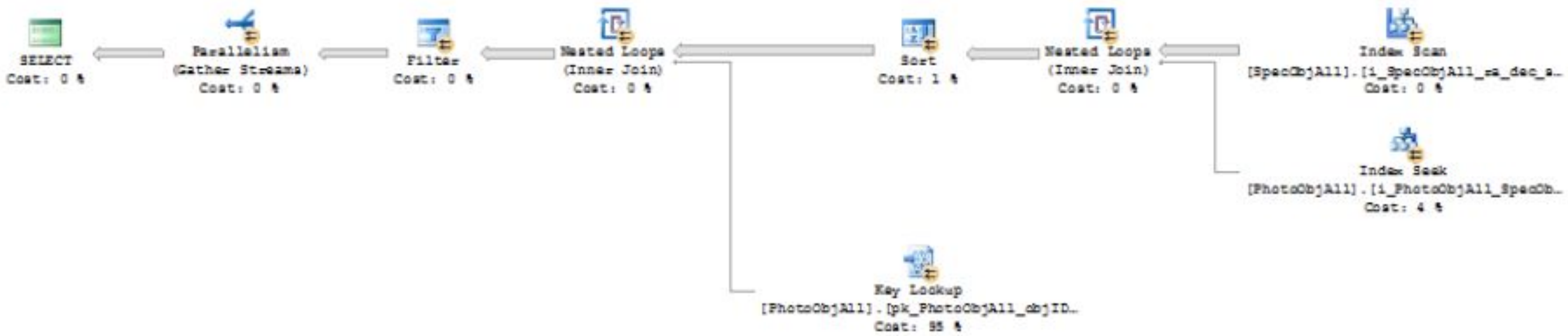
Steps of SQL query processing





Query plan example

```
SELECT s.SpecObjID, s.z
FROM   PhotoObjAll p
INNER JOIN SpecObjAll s
      ON p.SpecObjID = s.SpecObjID
WHERE p.dered_g < 17
```



- Operators describe how SQL Server executes a query
- The query optimizer uses operators to build a query plan

Operators:

- **Logical**:
  describe the relational algebraic operation used to process a statement. In other words, logical operators describe conceptually what operation needs to be performed.
- **Physical**:
  implement the operation described by logical operators. Each physical operator is an object or routine that performs an operation. For example, some physical operators access columns or rows from a table, index or view. Other physical operators perform other operations such as calculations, aggregations, data integrity checks or joins. Physical operators have costs associated with them.
  The physical operators initialize, collect data, and close

Physical operators: Table scan

- Table without any index
- Can be read only sequentially
- Have to read entire table
- No well-defined ordering
- Typical queries
  - SELECT * FROM t
  - SELECT * FROM t WHERE a = 2

Physical operators: Sort

- Table without ordering
- Output of another operator with wrong ordering
- Typical query
  - Table is not indexed by a

    > SELECT * FROM t ORDER BY a

- Have to sort rows
  - Quick sort with storage on disk
  - Algorithm optimized for large tables, still slow
- Sort happens in tempdb
  - Can be put on fast storage (SSD, NVME)

Physical operators: (Clustered) index scan

- Given a table t with an index on column c
- Column c contains unique values
- Scan range of key intervals

  > SELECT c FROM t WHERE v BETWEEN 5 AND 10

- Order by index keys
  - Index defines an ordering
  - Different ordering always requires a sort

  > SELECT c FROM t ORDER BY c

- Clustered index: contains all columns
- Non clustered index: only key and included columns
  - Querying other columns requires a bookmark lookup in clustered index
  - Results in random reads, slow

Physical operators: (Clustered) index seek

- Given a table t with index on column c
- Column c contains unique values
- Typical query

> SELECT c FROM t WHERE c = 12

- Finding a single row by ID
    - Table scan:o(N)
    - Index seek: o(log(N))
- If column c is not unique: index scan instead of seek
- Index seek mostly used for
    - Filtering by key
    - Joins on foreign keys (see later)

Physical operators: Bookmark (key) lookup

- Given a table t with non-clustered index on column c
- Query column d which is not part of the index

    > SELECT d FROM t WHERE c BETWEEN 2 AND 10

- Rows found quickly using index on c
    - Index contains only pointers to the rows
    - Column d needs to be read from clustered index
    - Fetch row by primary key or bookmark (heap tables)

- Expensive operation, scans result in random IO

Logical join operations

- Relational algebra: a join is simply a subset of the Cartesian product of two tables

CROSS JOIN: entire Cartesian product

SELECT *
FROM t1 CROSS JOIN t2

INNER JOIN: only rows satisfying join condition

SELECT *
FROM t1 INNER JOIN t2 ON f(t1.c1, t2.c2)

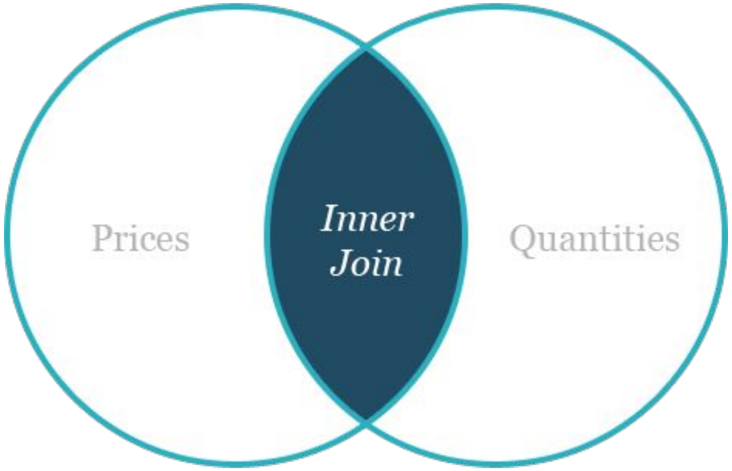


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices INNER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR INNER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66

Logical join operations

LEFT(RIGHT) OUTER JOIN

- Take all rows from left (right) table
- regardless if join condition is not satisfied
- Fill non-matches with NULL values

SELECT *
FROM t1 LEFT OUTER JOIN t2 ON f(t1.c1, t2.c2)

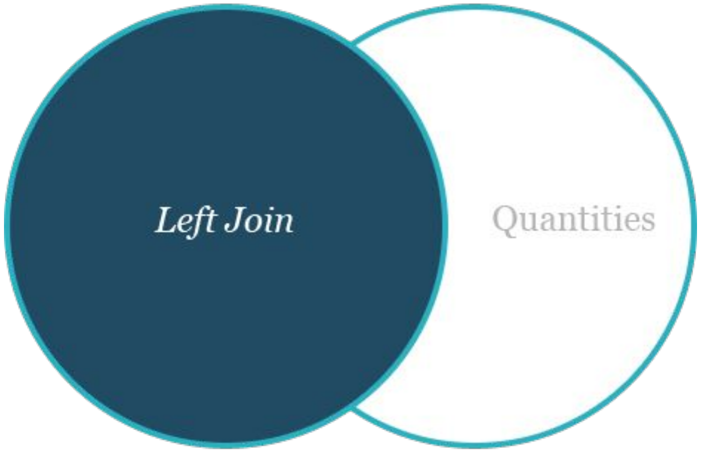


TABLE 1: PRICES		TABLE 2: QUANTITIES	
PRODUCT	PRICE	PRODUCT	QUANTITY
Potatoes	$3	Potatoes	45
Avocados	$4	Avocados	63
Kiwis	$2	Kiwis	19
Onions	$1	Onions	20
Melons	$5	Melons	66
Oranges	$5	Broccoli	27
Tomatoes	$6	Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices LEFT OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR LEFT OUTER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66
Oranges	$5	NULL
Tomatoes	$6	NULL

Additional logical join types

SEMI JOIN

- Only check if row exists in the other table

> SELECT * FROM t1 WHERE t1.c1 IN (SELECT t2.c2 FROM t2)

ANTI JOIN

- Returns rows that are not in another table

```
> SELECT * FROM t1
      WHERE t1.c1 NOT IN (SELECT t2.c2 FROM t2)
> SELECT t1.*
      FROM t1 LEFT OUTER JOIN t2 ON f(t1.c1, t2.c2)
      WHERE t2.c2 IS NULL
```

Additional join types

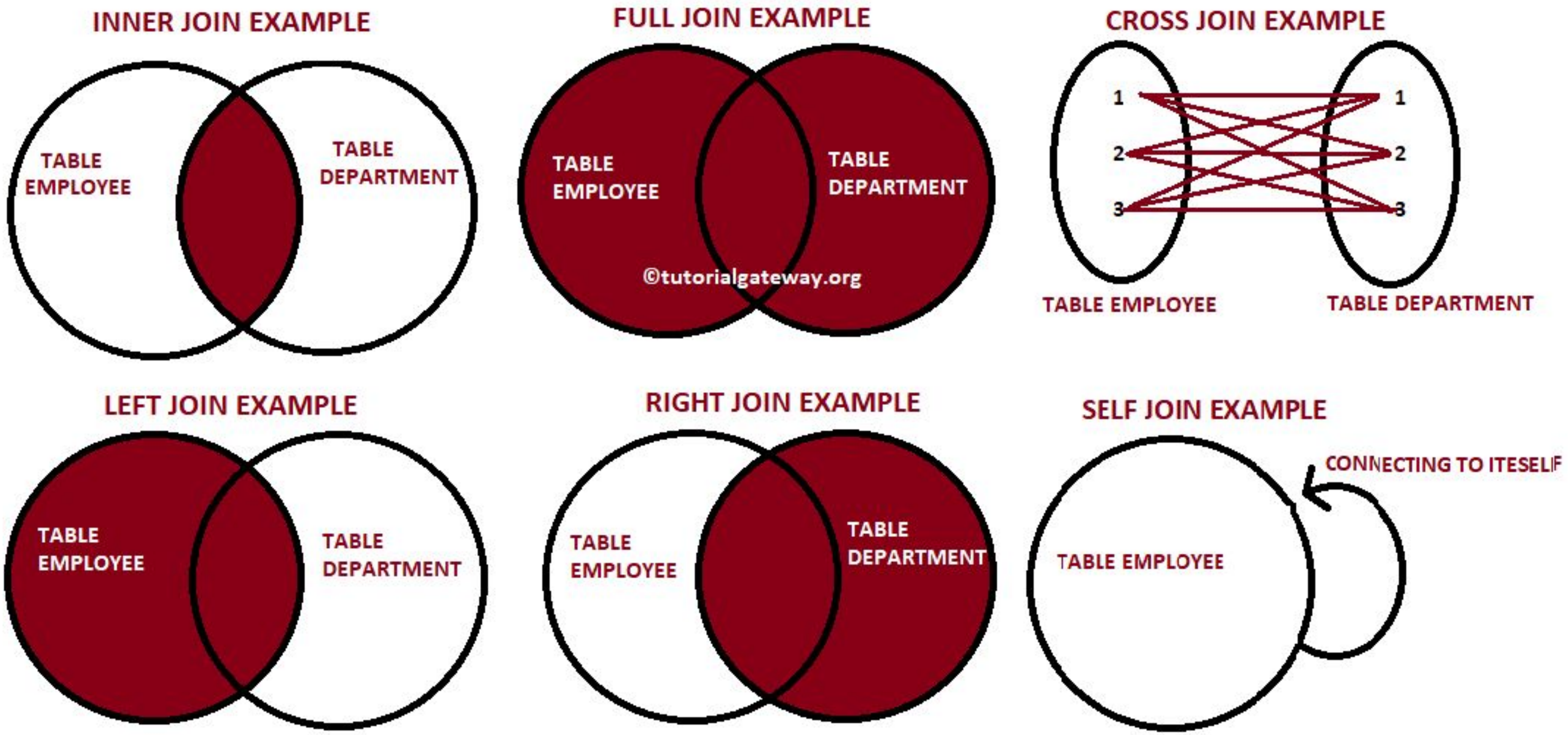
Range join

- If query restricts key not to a unique value but to a range
- Filter by interval

```
> SELECT *
    FROM t1 INNER JOIN t2
    t1.ID BETWEEN t2.start AND t2.end
```

- Will be important for hierarchical spatial indices

Join operations



Aggregates

- COUNT(*), MIN, MAX, AVG, SUM, STDEV
- GROUP BY, HAVING
- User-defined aggregates
- Have to implement three functions
  - Accumulate: process next row
  - Merge: merge results from two threads
  - Terminate: calculate final result
- Aggregates must be computable with this simple model

An example: averaging

```
void Accumulate(avgstate a, double v)
{
	a.count ++;
	a.v += v;
}

void Merge(avgstate a, avgstate b)
{
	a.count += b.count;
	a.v += b.v;
}

double Terminate(avgstate a)
{
	return a.v / a.count;
}
```

Group by operations

- Hash match
  - If the table is small enough
  - Hash table stores status of each aggregate group
- Stream aggregate
  - Read table in the order of GROUP BY clause
  - Requires an appropriate index
  - Rows can be aggregated sequentially
  - No need to keep more than one aggregate group in memory

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Query optimization

The query optimization problem

- SQL language is declarative
  - We specify what we want in the results
  - Not how we want the results to be computed
- Information available to the server
  - The SQL query itself (WHERE, JOIN, GROUP BY, ORDER BY)
  - Database schema: tables, columns, keys, storage model
  - Non-clustered and clustered indices
  - Index statistics (histogram of key distribution)
  - Available memory, number of CPUs
- Server implements numerous phyisical operators
- SQL query can be processed many ways
  - Server enumerates possible query plans solving the same problem
  - Many query plans yield the same results
  - Tries to find the one with the shortest execution time

Manual query optimization

- Query optimization is a large area of research

- Server can only use information available to it
  - (Usually) doesn't build new indices automatically
  - Can sort a resultset, if necessary

- How can we help the server?
  - Try to collect all possible queries
  - Design indices to support possible queries
  - Define physical operators explicitly by query hints

Index selection

- Primary selection aspects
  - **Index contains all required columns** to answer the query
  - Included columns can help a lot here
  - **Index** must be **ordered** according to the query
  - Pick the smallest index, if there are multiple choices

- Determine I/O requirements
  - Required columns are listed in the SQL query
  - Need to estimate number of rows from index statistics
  - Sequential scan is cheap, random seek is expensive

Transaction

Transaction interference

- How many concurrent transactions can interfere with each other
- What state of the database is visible to a transaction
- Can changes made by a concurrent transaction be seen?
- What type of changes affect concurrent transactions?

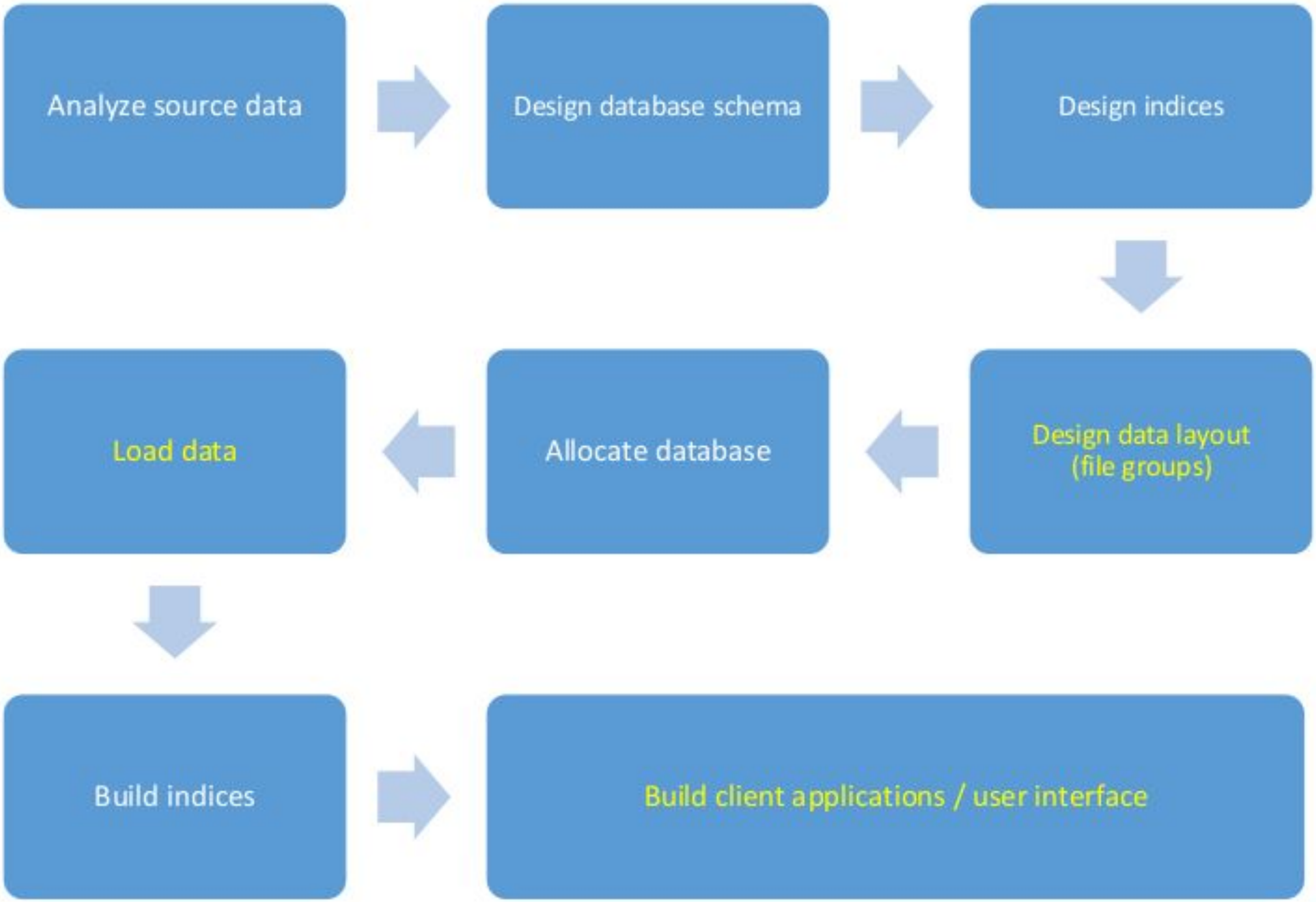
Transaction

Isolating transactions

- Goal: run many transactions in parallel but keep data integrity

- **Locking and Versioning**

Typical loading pipeline steps



Metadata

- Additional data belonging to tables and columns
- What is the data about?
  - Have to know exactly what all those numbers and strings mean
  - e.g. wavelength in vacuum or air?
  - Automatic conversion between unit systems?
  - Derived quantities? e.g. energy/wavelength/velocity
  - Measurement instructions: exact description of measuring process and calculations
  - Data quality information: Measurement errors, covariances
  - Provenance information: need to characterize veracity, reliability of data

- Need semantic information
  - Additional data about the data schema
  - Connect data model with reality

- Meta data need to be standardized for each field of science
  - Not only data format
  - Data models + meta data + ontology
  - Fundamental for easy sharing of complex data

- Display on web page of database

Metadata in databases

- Column name is not enough
- SQL Server
  - Supports extended properties (EP)
  - Every object of the schema can be tagged with EPs
  - EPs can be queries with standard SQL

- Include meta data in create table scripts
  - e.g. XML comments
  - Simple parser to process scripts and generate meta data

- Meta data on the user interface
  - Web site can automatically generate documentation from meta data

Data provenance

- Provenance: originally the history of an object, piece of art etc.

- Knowing the source of data is fundamental
    - How much data can be trusted
    - What if we just downloaded from the Internet
    - How measurement were made
    - How derived quantities were calculated

Data warehouse registry

- A data warehouse can contain hundreds of databases
- Need a thorough description of everything
  - Hardware: what machines where
  - Software configuration of machines
  - Databases: where what data in what format
- Where to direct a computation given its data and processing requirements
  - Minimize data movement
- Services available

Fundamental problems of databases in general

- Data is much bigger than the main memory

- Non-volatile storage is always much slower than the main memory and the CPU

- Sequential disk access is much faster than random
  - Store data in order, indexed

- Data processing algorithms
  - Optimize for sequential data access
  - Optimize memory for size, access pattern and NUMA
  - Fast or no transformation between in-memory and disk data formats

Goals with distributed databases

- Split large database among many servers: sharding
- Replicate data instead of back ups
  - Many options to achieve redundancy
- Mirror databases
  - Parallelize queries for higher IO throughput
  - Higher availability
  - Load balancing
- Regular data loading tasks
  - Dedicated load servers
  - Bulk insert, data validation, sorting
  - Faster to write storage

Data loading

Jim Gray’s laws for data warehouse design

- Lots of data -> one server won’t be enough
  - Scale out to multiple machines instead of scale up
  - Many machines instead of big iron
  - Easy management and expansion are very important
- Take computation to the data, not the data to the CPU
  - Moving data around takes much more time than processing
  - Try to solve problems within the database server, using SQL
  - Build servers with large storage and big CPUs
- What are the 20 most important queries you want to answer?
  - In a generic data warehouse, queries are not known beforehand
  - Users (scientists) come up with their own ideas
  - Optimize databases for the 20 queries
- Develop system gradually, from working version to working
  - Scientists want a working system as soon as possible
  - Don’t rush forward and add unstable features

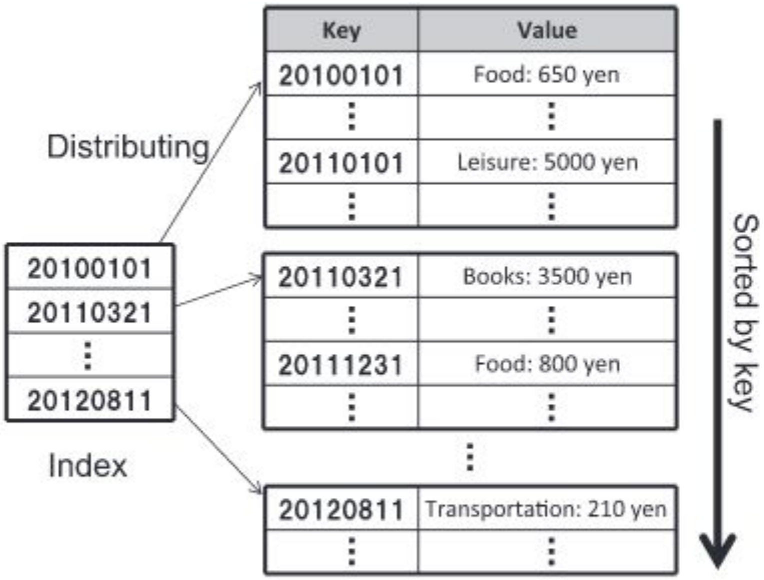
NoSQL / Non-relational databases

Alternative to RDBMS as web applications became increasingly complex

NoSQL databases can be schema agnostic, allowing unstructured and semi-structured data to be stored and manipulated.

Key-Value Stores, (Redis, Amazon DynamoDB)

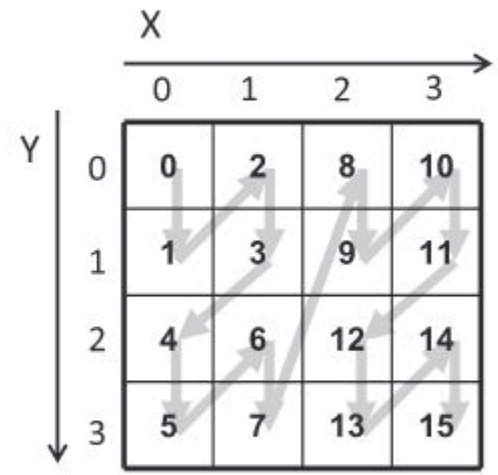
- extremely simple
- store only key-value pairs
- provide basic functionality for retrieving the value associated with a known key.
- stored data is **not particularly complex**
- **speed** is of paramount importance.



https://www.nec.com/en/global/techrep/journal/g12/n02/pdf/120216.pdf

NoSQL / Non-relational databases

Wide Column Stores (Cassandra, Scylla, HBase)



- schema-agnostic
- data in column families or table
  a multi-dimensional key-value store:
  e.g. latitude and longitude
- **Z-ordering**
- Space partitioning using the **Kd-tree method**
- scale well enough to manage
  petabytes of data within a distributed system.

https://www.nec.com/en/global/techrep/journal/g12/n02/pdf/120216.pdf

NoSQL / Non-relational databases

Document Stores (MongoDB, Couchbase)

- schema-free
- store data in JSON documents
- Key = document name
- Value = document
- Manage semi-structured data

A sample json:

```
{
"name": "notebook",
"qty": 50,
"rating": [ { "score": 8 }, { "score": 9 } ],
"size": { "height": 11, "width": 8.5, "unit": "in" },
"status": "A",
"tags": [ "college-ruled", "perforated"]
}
```

NoSQL / Non-relational databases

Graph Databases (Neo4J, Datastax Enterprise Graph)

- **data as a network** of related nodes/objects
  -> facilitates data visualizations and graph analytics
- node/object contains free-form data
  connected by relationships, grouped according to labels
- Analysis of the **relationships between heterogeneous data** points
- Fraud prevention, advanced enterprise operations, Facebook’s friends graph.

NoSQL / Non-relational databases

Search Engines (Elasticsearch, Splunk, Solr)

- schema-free JSON documents -> similar to document stores
- unstructured or semi-structured data
- easily accessible via text-based searches with strings of varying complexity..

NoSQL / Non-relational databases

Advantages

- Since there are so many types and varied applications of NoSQL databases, it’s hard to nail these down, but generally:
-
    - Schema-free data models are more flexible and easier to administer.
    - NoSQL databases are generally more horizontally scalable and fault-tolerant.
    - Data can easily be distributed across different nodes. To improve availability and/or partition tolerance, you can choose that data on some nodes be "eventually consistent".
-

Multidimensional data: few fundamental problems

- Cluster finding in point distributions
- Trajectories in space and time
  - Lagrangian mechanics
- Outlier detection
  - Find points with unusual properties
  - Fast generation of histograms
- • For analysis and visualization
- • Volume rendering methods
- • Correlation functions (pair and higher order)
- • Interactive visualization of large point clouds

Multidimensional data: few fundamental problems

- Find points in a given region of space
  - Region is given with analytic description
  - Equation of sphere, rectangle, polihedron etc.

- Find nearest neighbor of a query point
  - Find k nearest neighbors
  - Also for cluster finding
  - Point classification (machine learning)
  - Non-parametric regression, non-linear regression