NIE-PDB | **Advanced Database Systems** | 2025/26 Winter

Exam Requirements

NoSQL Introduction

- **Big Data and NoSQL** terms, **V characteristics** (volume, variety, velocity, veracity, value, validity, volatility), **current trends** and challenges (Big Data, Big Users, processing paradigms, …), principles of **relational databases** (functional dependencies, normal forms, transactions, ACID properties); **types of NoSQL systems** (key-value, wide column, document, graph, …), their data models, features and use cases; **common features** of NoSQL systems (aggregates, schemalessness, scaling, flexibility, sharding, replication, automated maintenance, eventual consistency, …)

Data Formats

- **XML**: constructs (element, attribute, text, …), content model (empty, text, elements, mixed), entities, well-formedness; document and data oriented XML
- **JSON**: constructs (object, array, value), types of values (strings, numbers, …); **BSON**: document structure (elements, type selectors, property names and values)
- **RDF**: data model (resources, referents, values), triples (subject, predicate, object), statements, blank nodes, IRI identifiers, literals (types, language tags); graph representation (vertices, edges); **N-Triples notation** (RDF file, statements, triple components, literals, IRI references); **Turtle notation** (TTL file, prefix definitions, triples, object and predicate-object lists, blank nodes, prefixed names, literals)

XML Databases

- Native XML databases vs. XML-enabled relational databases; data model (**XDM**): tree (nodes for document, elements, attributes, texts, …), document order, reverse document order, sequences, atomic values, singleton sequences
- **XPath** language: **path** expressions (relative vs. absolute, evaluation algorithm), path step (axis, node test, predicates), **axes** (forward: child, descendant, following, …; reverse: parent, ancestor, preceding, …; attribute), **node tests**, **predicates** (path conditions, position testing, …), abbreviations
- **XQuery** language: path expressions, **direct constructors** (elements, attributes, nested queries, well-formedness), **computed constructors** (dynamic names), **FLWOR** expressions (for, let, where, order by, and return clauses), typical FLWOR use cases (joining, grouping, aggregation, integration, …), **conditional** expressions (if, then, else), **switch** expressions (case, default, return), universal and existential **quantified** expressions (some, every, satisfies), **comparisons** (value, general, node; errors), atomization of values (elements, attributes)

RDF Stores

- **Linked Data**: principles (identification, standard formats, interlinking, open license), Linked Open Data Cloud
- **SPARQL**: graph pattern matching (solution sequence, solution, variable binding, compatibility of solutions), **graph patterns** (basic, group, optional, alternative, graph, minus); **prologue** declarations (BASE, PREFIX clauses), **SELECT** queries (SELECT, FROM, and WHERE clauses), query **dataset** (default graph, named graphs), **variable assignments** (BIND), **FILTER** constraints (comparisons, logical connectives, accessors, tests, …), **solution modifiers** (DISTINCT, REDUCED; aggregation: GROUP BY, HAVING; sorting: ORDER BY, LIMIT, OFFSET), **query forms** (SELECT, ASK, DESCRIBE, CONSTRUCT)

MapReduce

- **Programming models**, paradigms and languages; parallel programming models, process interaction (shared memory, message passing, implicit interaction), problem decomposition (task parallelism, data parallelism, implicit parallelism)

- **MapReduce**: programming model (data parallelism, divide-and-conquer paradigm, map and reduce functions), **cluster architecture** (master, workers, message passing, data distribution), **map and reduce functions** (input arguments, emission and reduction of intermediate key-value pairs, final output), **data flow phases** (mapping, shuffling, reducing), input parsing (input file, split, record), **execution steps** (parsing, mapping, partitioning, combining, merging, reducing), **combine function** (commutativity, associativity), additional functions (input reader, partition, compare, output writer), **implementation details** (counters, fault tolerance, stragglers, task granularity), usage patterns (aggregation, grouping, querying, sorting, …)
- **Apache Hadoop**: modules (Common, HDFS, YARN, MapReduce), related projects (Cassandra, HBase, …); **HDFS** module: data model (hierarchical namespace, directories, files, blocks, permissions), architecture (NameNode and DataNode nodes, HeartBeat messages, failures), replica placement (rack-aware strategy), FsImage (namespace, mapping of blocks, system properties) and EditLog structures, FS commands (ls, mkdir, …); **MapReduce** module: architecture (JobTracker and TaskTracker nodes), job implementation (Configuration; Mapper, Reducer, and Combiner classes; Context, write method; Writable and WritableComparable interfaces), job execution schema

NoSQL Principles

- **Scaling**: scalability definition; **vertical scaling** (scaling up/down), pros and cons (performance limits, higher costs, vendor lock-in problem, …); **horizontal scaling** (scaling out/in), pros and cons, **network fallacies** (reliability, latency, bandwidth, security, …), **cluster** architecture; design questions (scalability, availability, consistency, latency, durability, resilience)
- **Distribution** models: **sharding**: idea, motivation, objectives (balanced distribution, workload, …), strategies (mapping structures, general rules), difficulties (evaluation of requests, changing cluster structure, obsolete or incomplete knowledge, network partitioning, …); **replication**: idea, motivation, objectives, replication factor, architectures (master-slave and peer-to-peer), internal details (handling of read and write requests, consistency issues, failure recovery), replica placement strategies; mutual combinations of **sharding and replication**
- **CAP theorem**: CAP guarantees (consistency, availability, partition tolerance), CAP theorem, consequences (CA, CP and AP systems), CA spectrum, **ACID properties** (atomicity, consistency, isolation, durability), **BASE properties** (basically available, soft state, eventual consistency)
- **Consistency**: strong vs. eventual consistency; **write consistency** (write-write conflict, context, pessimistic and optimistic strategies), **read consistency** (read-write conflict, context, inconsistency window, session consistency), **read and write quora** (formulae, motivation, workload balancing)

Key/Value Stores

- Data model (key/value pairs), **key management** (real-world identifiers, automatically generated, structured keys, prefixes), basic CRUD operations, use cases, representatives, extended functionality (MapReduce, TTL, links, structured store, …)
- **Riak**: data model (buckets, objects, metadata headers); **HTTP interface**, cURL tool (options); **CRUD operations** (POST, PUT, GET, and DELETE methods, structure of URLs, data, headers), buckets operations (buckets, keys, properties); **data types** (Convergent Replicated Data Types: register, flag, counter, set, map; conflict resolution policies; usage restrictions), **Search 2.0** Yokozuna (architecture; indexation and query evaluation processes; specific extractors: text, XML, JSON; SOLR document: extracted and technical fields; indexing schema: tokens, triples; full-text index creation, association and usage; query patterns: wildcards, ranges, …); **Riak Ring** (physical vs. virtual nodes, consistent hashing, partitions, replica placement strategy, hinted handoff, handling of read and write requests)

Wide Column Stores

- Data model (column families, rows, columns), query patterns, use cases, representatives
- **Cassandra**: data model (keyspaces, tables, rows, columns), primary keys (partition key, clustering columns), column values (missing; empty; native data types, tuples, user-defined types; collections: lists, sets, maps; frozen mode), additional data (TTL, timestamp); **CQL** language: DDL statements:

CREATE KEYSPACE (replication strategies), DROP KEYSPACE, USE keyspace, **CREATE TABLE** (column definitions, usage of types, primary key), DROP TABLE, TRUNCATE TABLE; native data types (int, varint, double, boolean, text, timestamp, …); literals (atomic, collections, …); DML statements: **SELECT** statements (SELECT, FROM, WHERE, GROUP BY, ORDER BY, and LIMIT clauses; DISTINCT modifier; selectors; non/filtering queries, ALLOW FILTERING mode; filtering relations; aggregates; restrictions on sorting and aggregation), **INSERT** statements (update parameters: TTL, TIMESTAMP), **UPDATE** statements (assignments; modification of collections: additions, removals), **DELETE** statements (deletion of rows, removal of columns, removal of items from collections)

Document Stores

- Data model (documents), query patterns, use cases, representatives
- **MongoDB**: data model (databases, collections, JSON/BSON documents, field name restrictions), document identifiers (_id fields, features, ObjectId), data modeling (embedded documents, references); **insertOne** and **insertMany** operations (management of identifiers); **replaceOne** operation: upsert mode; **updateOne** and **updateMany** operations: update operators (field: $set, $rename, $inc, …; array: $push, $pop, …), upsert mode; **deleteOne** and **deleteMany** operations; **find** operation: query conditions (value equality vs. query operators), query operators (comparison: $eq, $ne, …; element: $exists; evaluation: $regex, …; logical: $and, $or, $not; array: $all, $elemMatch, …), dot notation (embedded fields, array items), querying of arrays, projection (positive, negative), projection operators (array: $slice, $elemMatch), **modifiers** (sort, skip, limit); **MapReduce** (map function, reduce function, options: query, sort, limit, out); primary and secondary **index structures** (index types: value, hashed, …; forms; properties: unique, partial, sparse, TTL)

Graph Databases

- Data model (property graphs), use cases, representatives
- **Neo4j**: data model (graph, nodes, relationships, directions, labels, types, properties), property types, structural types, composite types (lists, maps), properties (fields, atomic values, arrays); **Cypher** language: graph matching (solutions, variable bindings); query sub/clauses (read, write, general), chaining; **path patterns**, node patterns (variable, labels, properties), relationship patterns (variable, types, properties, variable length), graph patterns (uniqueness requirement); **MATCH** clause (path patterns, OPTIONAL mode), **WHERE** conditions (comparisons, IS NULL predicate, IN predicate, string matching, regular expressions, path pattern predicate, existential subquery, quantifiers, logical connectives); **RETURN** clause (DISTINCT modifier, ORDER BY, LIMIT, and SKIP subclauses), **aggregation** (grouping, aggregate functions); **WITH** clause (motivation, subclauses); query structure (chaining of clauses, query parts, restrictions), UNION operation, **map operations** (static and dynamic lookup), **list operations** (subscript, slice, list comprehension, pattern comprehension)

SQL Evaluation

- Principles: **motivation** (relational algebra operations, naive algorithms, optimization objectives, evaluation plan, database context, system memory, cost estimates); **data files** (file structure, tuple size, blocks, slots); **data file organization** (heap file, sorted file, hashed file); **index structures** (B^+ trees, non-clustered/clustered index); **access methods** (cost estimates, full scan, equality tests, range queries)
- Algorithms: **external sort** (basic approach, sort phase, merge phase, algorithms, memory usage, cost estimates, 2-pass setup); **nested loops join** (binary approach, basic, general, optimized setup, algorithms, memory usage, cost estimates, zig-zag optimization); **sort merge join** (basic approach, sort phase, join phase, algorithms, memory usage, cost estimates, duplicate handling); **hash join** (classic hash join, partition hash join, limitations, build phase, probe phase, algorithms, memory usage, cost estimates)
- Evaluation: **evaluation process** (query tree, evaluation plan, overall cost, pipelining mechanism, EXPLAIN statements, false assumptions); query optimization: **statistical optimization** (objectives, available statistics, histograms, size estimates, reduction factor); **algebraic optimization** (objectives, equivalence rules, SPJ queries); **syntactic optimization**