DBMS architecture/implementation:
Disk I/O: volatile storage loses contents when power is switched off, non-volatile storage contents persist when power is switched off.
Storage hierarchy: CPU, cache, main memory, flash memory, magnetic disk, optical disk, magnetic tapes..
Storage hierarchy: primary storage fastest but volatile, more expensive, limited (cache, main memory), secondary storage non-volatile (flash, magnetic disks), tertiary storage non-volatile, slow, unlimited (magnetic tape).
Disk configuration: read-write disk heads, cylinder, platter, tracks, sectors.
Seek: move head to cylinder/track, rotation: wait for sector to get under head, transfer: move data from disk to memory.
Flash storage: NAND flash cheaper than NOR flash. Solid state disks use block-oriented disk interfaces, but store data on multiple flash storage devices internally.
Block is the unit of transfer from disk to computer memory. Disk controller translates logical address of blocks to physical address, computes checksums to verify data read back correctly, performs remapping of bad sectors.
Disk performance measures: access time (time taken from read/write request to data transfer) consists of seek time (time taken to reposition arm over correct track) and rotational latency (time taken for sector to be accessed to appear under the head), data-transfer rate (rate at which data is retrieved from or stored to disk). Sequential access pattern: disk seek required only for first block, random access pattern: each access requires a seek, IOPS, MTTF.
Redundant array of independent disks (RAID): data storage virtualization technology combining multiple physical disk drives into a single logical unit for the purposes of data redundancy and performance improvement. 0: two physical disks make one logical fast disk, striping without mirroring or parity, 1: two physical disks make one logical reliable disk, mirroring without parity or striping, 2: bit-level striping with Hamming-code parity, 3: byte-level striping with dedicated parity, 4: block-level striping with dedicated parity, 5: block-level striping with distributed parity, one big logical disk is five smaller logical disks with improved performance through parallelism, 6: block-level striping with double distributed parity, nested 0+1: creates two stripes and mirrors them, 1+0: creates a striped set from a series of mirrored drives, JBOD N+N: just a bunch of disks.
Even-odd parity: 0 if even number of bits, 1 if odd. Block parity applies to a set of blocks bitwise. Can recompute missing blocks from remaining blocks and the parity block.

File organization:
Db is stored as a collection of files, each file is a sequence of records, a record is a sequence of fields.
Fixed-length records: store record i from byte n(i-1), where n is record size. Delete record i: move each record back one, move last record to deleted record, link all free records on free list.
Variable-length records: represented by fixed size offset/length, with actual data stored after. Slotted page header contains number of records, end of free space in block, location and size of each record. Large blob/clob types, records must be smaller than pages.
Heap: record can be placed anywhere in the file where there is space (free-space map), sequential: store records in sequential order based on search key value, multitable clustering: records of several different relations can be stored in the same file, b+-tree: ordered storage with inserts/deletes, hashing: hash function computed on search key, result specifies which block of file to place record in.
Partitioning: records in relation can be partitioned into smaller relations that are stored separately. Column-oriented: store each attribute of a relation separately.

Buffer pool management:
Buffer: portion of main memory available to store copies of disk blocks.
Buffer manager: subsystem responsible for allocating buffer space in main memory.
Pinned block: memory block that is not allowed to be written to disk. Pin before reading/writing data from block, unpin when complete.
Shared/exclusive locks on buffers prevent concurrent operations.
Buffer-replacement policies: least recently used (LRU) uses past pattern of block references as predictor of future references. Toss-immediate frees space occupied by a block as soon as the final tuple has been processed. Most recently used (MRU) pins the currently processed block, unpins after processing and it becomes the MRU block. Buffer manager can reorder writes using heuristics. Clock algorithm is a less expensive approximation of LRU, arrange frames in a logical circle and mark 1 when the block is added to the frame. Sweep hand, replace if bit is 0, set to 0 and continue if bit is 1.

Indexes:
Search key: attribute to set of attributes used to look up records in a file. Index file consists of entries of the form search key, pointer. Ordered index stores entries sorted on the search key value. Clustering/primary index has a search key that specifies the sequential order of the file. Secondary/ non-clustering index has a search-key that specifies an order different from the sequential order of the file. Dense index: index record appears for every search-key value in the file. Sparse index: contains index records for only some search-key values, use less space. Composite search key.
B+-tree: all paths from root to leaf are the same length, each node that is not a root/leaf has (n/2)-n children. Leaf node has between (n-1)/2 and n-1 values. If the root is not a leaf, it has at least 2 children. If the root is a leaf, it can have 0-(n-1) values. Node has P_1, K_1, …, K_{n-1}, P_n, K search-key values, P pointers to children. Non-leaf levels form a hierarchy of sparse indices. Tree height is no more than $\log_{n/2}(K)$.

Hashing:
Bucket is a unit of storage containing 1+ entries, obtained from its search key-value using a hash function. In a hash index, buckets store entries with pointers to records. In a hash file organization, buckets store records. Closed addressing uses overflow chaining, overflow buckets of a given bucket are chained together in a linked list.
Static hashing uses a fixed set of buckets. Can allow the number of buckets to be modified dynamically.

Query Processing:
Parser/translator verifies syntax correctness and generates a parse tree, converts to logical plan tree that defines how to execute the query. Optimizer modifies the logical plan to define an improved execution, transforms query, and determines how to choose among multiple operator implementations. Engine executes the plan, may optimize the plan using indexes.
MySQL EXPLAIN <query>.
Query optimization: amongst all equivalent evaluation plans choose the one with lowest cost. Cost measures: disk access, CPU, network communication, query response time, resource consumption. Disk cost: number of block transfers from disk: $b*t_T+S*t_S$, t_T time to transfer one block, t_S time for one seek, b block transfers, S seeks.
Algorithm selection:
File scan: A1 (linear search) scan each file block and test all records to see whether they satisfy the selection condition.
Index scan: A2 (clustering index, equality on key) retrieve a single record that satisfies the corresponding index search-key equality condition, A3 (clustering index, equality on non-key) retrieve multiple records. A4 (secondary index, equality on key/non-key) retrieve a single record if the search-key is a candidate key, else retrieve multiple records.
Comparisons: A5 (clustering index, comparison) relation is sorted A and use index to find first tuple then scan sequentially, A6 (clustering index, comparison) use index to find first index entry and then scan sequentially.
Conjunction: A7 (conjunctive selection using one index) select a combination of algorithms that results in the least cost, A8 (conjunctive selection using composite index) use appropriate composite index if available, A9 (conjunctive selection by intersection of identifiers) requires indices with record pointers and take intersection of sets of corresponding record pointer for each condition.
Disjunction: A10 (disjunctive selection by union of identifiers) applicable if all conditions have available indices and take union of all the obtained sets of record pointers, negation uses linear scan if very few records satisfy the condition.
Index creation: engine may build either a sorted or hash index. The cost of building the index and then using it in the algorithm is cheaper than running the default algorithm with an index.

JOIN:
Nested-loop join: expensive since it examines every pair of tuples in scan/probe tables, cost $n_r b_s + b_r$. Indexed nested-loop join: can replace file scans if equi/natural join and available index on inner relation join attribute, cost $b_r(t_T + t_S) + n_r C$.
Merge-join: sort both relations on join attribute, merge sorted relations, cost $b_r + b_s + (b_r/b_b) + (b_s/b_b)$. Hybrid merge-join merges sorted relation with leaf entries of B+-tree.
Hash-join: used for equi/natural joins, hash function used to partition tuples of both relations.
Duplicate elimination: implemented via hashing/sorting. Projection: perform projection on each tuple followed by duplicate elimination.
Aggregation: sorting/hashing followed by aggregate functions applied on each group. Materialization: generate results of an expression whose inputs are relations and store on disk. Pipelining: pass on tuples to parent operations as an operation is being executed.
Evaluation:
Materialized evaluation: evaluate one operation at a time, use intermediate results materialized into temporary relations to evaluate next-level operations. Cost of writing results and reading them back can be high.
Double buffering uses two output buffers for each operation, when one is full write it to disk while the other is getting filled, reducing computation time.
Pipelining: evaluate several operations simultaneously, passing the results of one operation on to the next. Much cheaper than materialization, may not always be possible. Demand driven: system repeatedly requests next tuple from top level operation, operation has to maintain state in between calls to know what to return next. Producer driven: operators produce tuples eagerly and pass them up to their parents, system schedules operations that have space in the output buffer and can process more input tuples.

Equivalent expressions:
Relational algebra expressions are equivalent if two expressions generate the same set of tuples on every legal db instance. Equivalence rule says that expressions of two forms are equivalent if the first expression form can be replaced by the second or vice versa. 16 equivalence rules. Pushing selections: performing the selection as early as possible reduces the size of the relation to be joined. Join ordering: join smaller relations first so that a smaller temporary relation is stored.
Equivalent expression enumeration: space requirements are reduced by sharing common subexpressions, not generating all expressions. Query optimizers use dynamic programming, cost-based search, heuristics to choose evaluation plans.

Normalization:
Redundancy causes several problems associated with relational schemas. Decomposition of relations can cause update/insert/delete anomalies, loss of information. Normalization theory: if a relation R is not in good form, decompose it into a set of relations such that each relation is in good form and the decomposition is lossless. Functional dependencies: relation that satisfies all real-world constraints, $t_1(a)=t_2(a) \rightarrow t_1(b)=t_2(b)$. Set of all functional dependencies logically implied by set F is the closure of F.
Boyce-Codd normal form: if A is a super key, I can set a primary key/unique constraint on A. Not always possible to achieve both BCNF and dependency preservation. If a relation is in BCNF it is in 3NF, the third condition is a minimal relaxation to ensure dependency preservation. It is always possible to obtain 3NF without sacrificing losslessness, but may have to use null values to represent some of the possible meaningful relationships among data items. May use non-normalized schema for performance.
1NF ensures each column contains atomic values, 2NF eliminates partial dependencies, 3NF eliminates transitive dependencies, 4NF deals with multivalued dependencies, 5NF addresses join dependencies.
Armstrong's axioms of reflexivity, augmentation, and transitivity are sound and complete inference rules for functional dependencies.
Wide Flat tables improve query performance by precomputing joins, aggregation, and derived columns.

REST:
Representational state transfer is a web standards based architecture that uses HTTP protocol. A server provides access to resources and the REST client accesses and modifies the resources defined by global IDs. Methods: get, post, delete, put.

Transactions:
ACID properties: atomic (transactions cannot be subdivided), consistent (transaction transforms db from one consistent state to another), isolated (transactions execute independently and db changes aren't revealed to users until completed), durable (db changes are permanent).
Transaction is a unit of program execution that accesses/updates data. Syntax: begin transaction { read(), write(), commit/rollback }.
Main issues: hardware failures/system crashes, concurrent execution of multiple transfers. States: active, partially committed, failed, aborted, committed.
Subsystems: query processor schedules/executes queries, buffer manager controls reading/writing blocks/frames to/from disk, log manager records events to disk, transaction manager coordinates query scheduling buffer read/write and logging, recovery manager processes log to ensure ACID after failures, log is a special type of block file used to optimize performance. Log contains sequence number, previous LSN, ID, type.
Write ahead logging forces every write to disk, has poor response time but provides durability. Desire to steal bufferpool frames from uncommitted transactions but no force every write.
ARIES algorithms for recovery and isolation exploiting semantics: analysis pass, redo pass, undo pass.
Availability and replication: active/passive all requests go to master during normal processing. Active/active both environments process requests.
Isolation: a transaction schedule is serializable if its outcome is equal to the outcome of its transactions executed serially without overlapping in time. Levels: serializable, repeatable reads, read committed, read uncommitted. Cursors define isolation but are client conversation state and cannot be used in REST.
Strict two-phase locking protocol: each act must obtain a shared lock on object before reading and exclusive lock on object before writing, all locks held by a transaction are released when completed. Allows only serializable schedules and simplifies transaction aborts.

Scalability/availability:
Scale-up (replace system with bigger machine) is less incremental, more disruptive and expensive, does not improve availability. Scale-out (add another system) is incremental, enables availability, does not work well for joins or referential integrity. Share disks (scale-up for disks) requires distributed locking to control access from multiple servers, share nothing (scale-out for disks) partitions data into shards based on a function and can improve availability with data replication.

Data analytics:
Processing of data to infer models for prediction. ETL: data is extracted from source formats, transformed to common schema, and loaded into the data warehouse. Machine learning techniques find patterns in data and make predictions, data mining extends ML to run on very large datasets.
Data warehouse: data is processed and organized into a single schema before being put into the warehouse, analysis is done on cleansed data in the warehouse. Data lake: raw and unstructured data goes into the lake, data is selected and organized when needed.
Online analytical processing OLAP: interactive analysis of data, allowing for summarization and viewing in different ways in an online fashion.
Cross-tabulation/pivot-table: values for each dimension attributes form headers, values in individual cells are aggregates of attributes. Data cube is a multidimensional generalization of a cross tab. Operations: pivoting, slicing, rollup, drill down.

Locks:
Exclusive: data can be read and written, Shared: data can only be read. Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on the item, no other transaction may hold any lock on the item. Deadlock: both transactions are waiting for each other to release locks, must rollback one of them and release its locks. Starvation: transaction is waiting for X-lock on item that several other transactions are granted an S-lock on and is repeatedly rolled back.
Two-phase locking protocol: growing phase (transaction may obtain and not release locks), shrinking phase (transaction may release locks and not obtain locks). Strict: transaction must hold all X-locks until commit/abort. Rigorous: transaction must hold all locks until commit/abort.

Schedule is legal if it can be generated by a set of transactions that follow the locking protocol. Protocol ensures serializability if all legal schedules under the protocol are serializable. Upgrade: S to X-lock. Downgrade: X to S-lock. Lock manager can be implemented as a separate process, transaction request locks/unlocks as messages.
Deadlock prevention: wait-die (older transaction waits for younger to release data item, younger transactions never wait but are rolled back), wound-wait (older translation forces rollback of younger transaction, younger transactions may wait for older ones) results in fewer rollbacks, timeout-based waits for lock for specified amount of time and then is rolled back. Recovery: total rollback (abort transaction and restart it), partial rollback (roll back transaction as far as necessary to release locks).

Serializability:

Conflict and view serializability. Instructions of transactions conflict if there exists some item accessed by both and at least one wrote the item. Schedules S and S' are conflict equivalent if one can be transformed into the other by a series of swaps of non-conflicting instructions. S is conflict serializable if it is conflict equivalent to a serial schedule.
Schedule is conflict serializable if the precedence graph is acyclic.
Schedule is recoverable if commit of T1 appears before commit of T2 and T2 reads a data item previously written by T1. Cascading rollback: a single transaction failure leads to a series of transaction rollbacks.
Levels of consistency: serializable (default), repeatable read (only committed records to be read), read committed (only committed records can be read), read uncommitted (uncommitted records may be read).

BASE:

Basically available (read/write operations are available as much as possible but without any consistency guarantees), soft state (have some probability of knowing the state), eventually consistent (will eventually know the db state).
CAP theorem: consistency (every read receives the most recent write or an error), availability (every request receives a non-error response without guarantee that it contains the most recent write), partition tolerance (system continues to operate despite an arbitrary number of messages being dropped by the network between nodes). Can pick 2.

Big data/data engineering:

Structured, unstructured, semi-structured.
Volume, variety, velocity, veracity, value.
MapReduce is a data flow program with relatively simple operators on the data set with each operator implemented in parallel on multiple nodes for performance. Normal file system abstraction is a stream.
Spark engines enable programmers to add new operators which convert reliable distributed datasets/frames to new RDDs/frames. Operators in a job are used to construct a directed acyclic graph. RDD abstraction is a collection of records that can be stored across multiple machines.
Pyspark dataframe is a distributed collection of rows under named columns. AWS Glue is a serverless data preparation service that makes it for data engineers to extract, clean, enrich, normalize, and load data.

Streaming data:

Refers to data that arrives in a continuous fashion, in contrast to data-at-rest.
Query: windowing (break up stream into windows and run queries on windows), continuous (queries output partial results based on stream seen so far and update continuously), algebraic operators (each operator consumes tuples from a stream and outputs tuples), pattern matching (queries specify patterns that the system detects and triggers actions), lambda (split stream into two, one goes to stream processing system and the other to a db for storage). Tumbling, hopping, sliding, session windows.
Publish-subscribe systems provide a convenient abstraction for processing streams. Apache Kafka is a popular parallel pub-sub system.

Data mining:

Classification, regression, associations, clusters, decision trees, bayes theorem, SVM, NN

NoSQL:

"Not only SQL" stores data in non-tabular format rather than relational tables. Simplicity, scalability, performance, flexibility.
Offer compromise eventually consistency in favor of availability, partition tolerance, and speed. Column-family, graph, document, key-value.
Document: handles weak entities, may have multi-valued, complex attributes.
Query: {field: "value"}
Flexible schema, JSON compatibility, horizontal scalability, fast querying due to optimized indexing.
MongoDB: create database, create collection, insert(), find(), update(), remove(), $limit, sort, merge, union, lookup, match, merge, sample.
Graph: nodes/edges have labels/properties.
Query: p1(n)-p2(e)-p3(m). N,m are nodes, e is an edge, p1,p2,p3 are predicates on labels.
Efficient storage of semi-structured data, express queries as traversals, fast for graph-related JOIN queries, ACID transactions with rollback support.
Neo4j: match, where not, return, count, order by.

MongoDB command syntax:

Show dbs
Use <db_name>
db.coll.insertOne({}), insertMany([{name: "A"}, {name: "B"}])
db.coll.find(), distinct, find({"year": {$gt: 1970}})
db.coll.aggregate([{$match: {status: "A"}},{$group: {_id: "$cust_id", total: {$sum: "$amount"}}},{$sort: {total: -1}}])
db.coll.updateOne({"_id": 1}, {$set: {"year": 2016, name: "Max"}})
db.coll.deleteOne({name: "Max"})
db.createCollection(), createIndex(), addShard()

Neo4j cypher syntax:

```
WITH 30 AS minAge
MATCH (a:Person WHERE a.name = 'Andy')-[:KNOWS]->(b:Person
WHERE b.age > minAge)
RETURN b.name (distinct, order by, limit)
CREATE (n:Label {name: $value})
CREATE (n:Label)-[r:TYPE]->(m:Label)
MATCH
  (a:Person {name: $value1}),
  (b:Person {name: $value2})
MERGE (a)-[r:LOVES]->(b)
MATCH (n:Label)-[r]->(m:Label)
WHERE r.id = 123
DELETE r
MATCH (n:Label)
WHERE n.id = 123
REMOVE n:Label
```