Problem 1

- OS process per worker
  Pros:
   - OS does scheduling
   - Private address space
   - OS protects workers from each other (isolation)
  Cons:
   - "Heavy weight"
   - Extra work to synchronize workers
   - OS differences
- OS thread per worker
  Pros:
   - OS does scheduling
   - Shared address space and other resources between threads
   - More light weight than processes
  Cons:
   - Shared memory gives less isolation (less protection agains other threads)
- Light weight thread (application thread) per worker
  Pros:
   - Efficient
   - Shared address space, shared resources
  Cons:
   - Complex implementation
   - No OS support

Problem 2

(Admission)
Parsing (incl. lexing/scanning)
Resolving
- Look up names (columns, tables, views, functions, etc.)
- Privilege checks (grants)

(Rewrite) (can be included in optimization)
- Rule based (heuristic rewrite)
- Canonical form

Optimization
- Cost based optimization
- Deciding join order
- Deciding access methods

(Compiling) (if using compiled code)
Execution
- Typically Volcano iterator model (FetchNext)
- May also execute compiled code

Problem 3

SARG = Search ARGument. A predicate that can be used as a search argument to index lookups, i.e., the expression can be used to look up values in an index. After pushing predicates as far down as possible, the sargable predicates will be positioned directly above table scans, and table scan + sargable predicate can then be transformed into an index lookup.

Problem 4

An interesting order is an order that can be useful during execution or to deliver the result in the specified order. If an interesting order comes naturally from an access method or other internal operator, this can be exploited to avoid explicit sorting, which would be a blocking operation. It may save execution time, and the chance increases that the execution can be pipelined. Typical interesting order: 1) sorting on columns in GROUP BY can be usd to handle groups one by one witout interleaving, 2) sorting on join columns can be used to pick merge join, 3) sorting on columns in ORDER BY can be used to return the result of the query in order.

Problem 5

Leis et al. (2015) argues, and shows with experiments, that cardinality is more important than a good cost model. A good cost model is not unimportant, but experiments show that good cardinality estimates mean more. Cardinality estimates are important when choosing join order. Inaccurate estimates leads to larger and larger inaccuracies the more complex the query gets. A good cost model cannot compensate for the starting estimate being wrong, so therefore it is more important that the starting point (cost estimates) are good.

Problem 6

Accordingt to Leis et al (2017): There i a tendency to under estimate cardinality. Sampling finds a more exact estimate, and this is then usually higher. If you sample in depth (height?) first, and build some 3-way, 4-way etc. estimates, while other 2-way estimates still haven't been sampled, the sampling based estimates will tend to be larger than the non-sampled estimates, and the optimizer will prefer the non-sampled estimates even if they are less accurate. This is called fleeing from knowledge.

Problem 7

The background of LSM-trees is to reduce read performance in order to increase write performance. Still, LSM-trees are still a tree structure that can do efficient read operations, but not as efficient as, e.g., a B-tree. But writing is more optimized since it uses sequential writing instead of random access.

Problem 8

Interleaving is to save rows from different tables together based on the foreign key. E.g., all the rows for photos belonging to a person may be stored directly after the row for this person. By using interleaving, Spanner stores data together that frequently will be joined together. Spanner may then do local joins on each node and by reading disk blocks sequentially.

Problem 9
----

a)

There may be multiple correct answers here. The way the student argues for their answer is important.

Availability is important. Writing (voting) and reading (countingq votes) happens in distinct phases. If we store each vote as a row (instead of updating a counter for each contestant), we may store votes completely independent of each other, and we may then choose lazy update anywhere. Lazy means that we don't have to synchronize during voting, so there is low overhead for each operation. Update anywhere means that we can write on all nodes, so we can spread the writes on many nodes an avoid performance bottlenecks. The count itself happens after all votes have been cast, so there will be no read-write conflicts.

b)

The answers may differ due to the solution picked in a).

We've prioritized availability (A). We have designed the system in such a way that it doesn't have to offer a consistent image (C) during voting. A consistent image is only necessary after all votes have been cast when the result is being counted, but at that point there is no other load (no write load) on the system.

Problem 10

There are many points. Not all must be present.

Horizontal scalability
 - Scalability may often be higher than in RDBMSs
High availability
 - Tend to choose AP over CP
 - The advantage over RDBMSs is that the system is available more
Weak consistency
 - Con: Can't give same consistency guarantees as RDBMS
Horizontal sharding
Limited cross-shard functionality
 - More responsibility on the application than wiht RDBMs, e.g., to do joins
Simple data model
 - RDBMSs have an advantage by supporting representation of complex data
Schemaless
 - Faster development in schemaless model
 - Schemas in RDBMS prevent inconsistencies in the data model
Data model close to programming language
 - An advantage if using the right language, feels natural to the developer
 - Can be hard if using multiple languages
 - RDBMSs use SQL regardless of programming language
Denormalized data model
 - Leads to duplication of data, RDBMSs try to avoid this
Less advanced query language
 - RDBMSs can do more advanced queries with SQL
 - Easier for the DBMS to implement a simpler language
 - Responsibility for complex queries pushed over on app developers
   compared to RDBMSs
No transactions
 - BASE, not ACID
 - ACID in RDBMSs gives app developers stable and simple boundaries to
   relate to
 - BASE is easier to implement in the DBMS than ACID

Problem 11

Duplication is that one data element, e.g., a polygon, is stored in multiple locations in an index. This leads to the index growing in size, and when updating the index has to be updated in multiple locations. If duplication is possible, lookups aldo have to check all places where a duplicate entry can be stored. Duplicates have to be filtered out before the result is returned.

Problem 12

Space driven: The splitting is based on the span of the coordinate system the geometric objects exist inside. Boundaries for coordinates are predefined, and the index relates to these when splitting.

Data driven: The splitting is based on the actual data that is stored and adapts to the part of the coordinate space that is actually in use.