TDT4225 – Exercise 1 (theory)

> 1. **SSD**: How does the Flash Translation Level (FTL) work in SSDs?

The FTL makes sure all blocks in an SSD are uniformly worn out, by implementing wear leveling. Wear leveling are distributing wwrites over the whole disk, to avoid repeated erase-write cycles on the same blocks.

> 2. **SSD**: Why are sequential writes important for performance on SSDs?

Random writes in SSDs can cause internal fragmentation on SSDs, and reduce performance. With random writes, more complex and amount of garbage collection is necessary. After around 15 minutes of writes, the write speeds drops to 25MB/s if random writes are used and 250MB/s if sequential writes are used.

> 3. **RocksDB**: Describe the layout of MemTable and SSTable of RocksDB.

The format of the **SSTable** in a RocksDB is block based. All blocks stores sorted key/value pairs, along with indexes and metadata. The data blocks are the first elements in the SSTable. Afte the data blocks, there are meta blocks. These blocks contains for example a Bloom filter, statistics, compression and other metadata. Next are the metaindex-block, which holds indicies to the metablocks. Last there are a index block. This block cointains one entry for every data block.

The **memtable** is stored as a skip list. A skip list is a probabalistic version of linked lists with layers. The first layer is an ordinary linked list and the layers above are acting as a "express lane". These express lanes are also linked lists, but with fewer elements than the linked list in layer 0. For example, consider layer 0 as 1-2-3-4-5-6-7-8-9, a layer 1 could be 3-6-9. This way we can go directly to 3, check if the element we want is less than or equal to 3. If not, we can move on to 6 and do the same check. If the check is true, we can jump down to layer 0 and continue our search from, for example, element 6. In the memtable, this skip list is sorted, so that binary search is poosible. All updates to the DB is inserted into the memtable and once the memtable has reached it's maximum capacity, RocksDB creates a new memtable in which subsequent updates are inserted. At this point, the old memtable is being frozen and made immutable. Later, all records are flushed to disk into a SSTable.

> 4. **RocksDB**: What happens during compaction in RocksDB?

Compaction in RocksDB is done in a few different styles; Leveled compaction, Universal compaction and FIFO compaction.

Leveled compaction is, as the name suggest, a form of leveling. The most recent data lays in level 0 and the oldest in level n. The idea of leveled compaction is that SSTables from level L_i are merged into SSTables in level L_{i+1}. Here, only SSTables with overlapping key ranges are merged.

Universal compaction works in the same way as a **leveled** compaction, with som adjustments. Instead of merging SSTables with overlapping key ranges, **universal** compaction merges SSTables with overlapping time ranges. The input to the compaction is two SSTables with overlapping time ranges, and the output is a sorted run containing the earliest entry in the first SSTable and ranging to the latest entry in the last SSTable.

FIFO compaction is the easiest of the three compaction styles. It works kind of like a cache. All files are stored at L_0 and after some time or when the size limit is reached, the oldest data is deleted.

> 5. **LSM**-trees vs B+-trees. Give some reasons for why LSM-trees are regarded as more efficient than B+-trees for large volumes of inserts.

?

> 6. Regarding fault tolerance, give a description of what hardware, software and human errors may be?

Hardware	Software	Human
Hard disks crash	Corrupt drivers	Wrongly designed api
Faulty RAM	Functionality errors - Functionality has errors or works in an unexpected way	Wrongly connected devices
Power outage	Syntactic errors	Wrongly configured devices
Overheating	Bad error handling	Giving devices shock

> 7. Compare SQL and the document model. Give advantages and disadvantages of each approach.

The document model are fairly lightweight and easy to implement, but has bad support for joins and many-to-many and many-to-one relationships. This means that you have to have your logic in the application code instead of the database. SQL is a declratative language, making it easier than an imperative language (such as the document model) to explicitly tell what you want to, for example, retrieve:

Imperative query

```
const getSharks = () => {
    return animals.filter((animal) => animal.family === 'Sharks');
};
```

vs declarative SQL: `SELECT * FROM animals WHERE family = 'Sharks';`.

> 8. When should you use a graph model instead of a document model? Explain why.

You should use a graph model instead of a document model when anything is potentially connected to everything. This is because the document model have bad support for many-to-many relationships, and you'll have to implement join logic in your application code, which can be cumbersome.

> 9. **Column compression**: You have the following values for a column:
> 43 43 43 87 87 63 63 32 33 33 33 33 89 89 89 33
> a. Create a bitmap for the values.

> b. Create a runlength encoding for the values

?

> 10. We have different binary formats / techniques for sending data across the network:
>
> - MessagePack
> - Apache Thrift
> - Protocol Buffers
> - Avro
>
> In case we need to do schema evolution, e.g., we add a new attribute to a Person structure: Labour union, which is a String. How is this supported by the different systems? How is forward and backward compatibility supported?

?