UV Distributed Information Management
Summer term 2021

Assignment 01 - Q&A

This document provides supplementary material regarding terms, topics, and questions that have been discussed during the lecture, and is supposed to support the students.

1 Data Management

This section covers supplementary material of part 1 on *Data Management*.

1.1 Terminology

Data integrity (Datenintegrität, in german) This term has different meanings. In the context of information security, data integrity refers to the *prevention of unauthorized modification of information*. In other words, data integrity defines the *correctness* or *validity* of the data upon modification (done by humans or machines). In our context, *consistency (or integrity) constraints* are used to describe conditions that must be satisfied for the data to be correct. The data are *consistent* (or *of integrity*) if all the constraints are satisfied.

Example: If an attribute A stores the balance of a bank account, we may want to disallow values that are smaller than EUR -5,000 (i.e., we allow a credit of at most EUR 5,000 per bank account). If a customer then tries to lend more than EUR 5,000 from our bank, this constraint is violated and the system may prevent it.

Key (Schlüssel, in german) In the relational model (record-based tables), we need a way to distinguish the rows. Let K be a subset of the attributes (columns) of a relation (table). K is called *super key* if the attributes of K suffice to uniquely identify a tuple (row) in the relation. K is a *candidate key* if K is a super key and K cannot be further reduced (i.e., no attributes can be removed) without losing the super key property. A candidate key that consists of attributes that are rarely subject to updates is typically chosen as *primary key*.

Example: The social security number, the bank account identifier, or the registration number at universities are prototypical examples of primary keys. However, also a combination of multiple attributes can serve as primary key, e.g., the combination of firstname, lastname, and birthyear.

Schema In our lecture, the schema refers to the overall design of a database, i.e., it defines the structure of the data (similar to a variable declaration in a programming language) and the relationships between the data. We distinguish between *relation* and *database schema*. The relation schema refers to the schema of a single relation (table; in the relational

model), whereas the database schema refers to the collection of all relation schemata in the database.

Caveat: The term *schema* may have a different meaning in some database systems, e.g., a schema may subsume multiple tables. Therefore, we recommend to read the manual of the database system at hand. Nonetheless, we use the term *schema* as described above.

1.2 Questions

Q1 *Regarding the ACID properties: How is the isolation guarantee implemented?*

The ACID properties are split into four parts and we will provide information about the implementation (from a logical point of view) for each single part (based on Silberschatz et al. *Database System Concepts*. 7th Edition.):

Atomicity The atomicity property guarantees that a transaction is either executed completed or not at all (*"all or nothing"*). A transaction T that completes successfully is referred to as **committed.** Contrarily, a transaction T is called **aborted** if it does not complete successfully. In the case that transaction T is aborted, the DBS must ensure that any change to the database is undone. This is commonly referred to as **rollback** of a transaction. Many systems implement this guarantee by maintaining a **log file.** The DBS then records each single modification that will be done by a transaction in this log file. A log record contains information to identify the transaction and about the data that is updated (e.g., old and new values). Only after the log record has been written to persistent storage, the DBS modifies the data in the database accordingly. The information in the log file allows the DBS to undo all the modifications of a transaction (i.e., to roll it back) in case of a hardware or software failure.

Consistency The consistency property states that the database is in a consistent state if transactions are executed in isolation. Application developers must ensure the consistency of individual transaction, for example, by defining integrity constraints.

Isolation In a DBS, many transactions may be executed concurrently. The isolation property guarantees that each transaction is unaware of other transactions that are executed concurrently (it appears to a single transaction that it is executed in isolation). For every pair of transactions T_i and T_j, it appears to T_i as if T_j either finished before or started after T_i. A simple way to implement (or rather ensure) this property is to execute transaction in a serial way, i.e., one after another. However, this would severely limit the performance (with respect to both throughput and response time) of the DBS. Therefore, so-called **concurrency control schemes** have been developed and are still subject to active database systems research. A concurrency control scheme typically allows the DBS to execute multiple transactions concurrently. Assume we have two transaction T_i and T_j, each of which consists of 8 instructions. When we execute T_i and T_j concurrently, the operating system (OS) decides which transaction is executed on the processor (CPU) and for how long (i.e., for how many cycles). For example, the OS may decide to allow T_i to execute its first 2 instructions, then T_2 is allowed to execute 4 instructions. Afterwards, T_1 is allowed to execute 4 instructions, then T_2 executes 4 instructions, and finally T_1 finishes by executing the remaining 2 instructions. This so-called **schedule** may result in a correct database state, or not. Thus, the concurrency control component of the DBS ensures that only schedules are executed that result in a consistent database state. One

possible way is to use only so-called **serializable** schedules, i.e., concurrent schedules that have the same effect as some serial schedule. There are various policies for concurrency control that can be utilized to ensure this: Lock-based protocols ensure isolation by locking data items at runtime, i.e., if one transaction accesses a data item, no other transaction is allowed to modify this data item. Optimistic validation-based protocols monitor the system such that each transaction is executed in multiple phases and consistency is validated at the end. In the case of a conflict, the corresponding transaction is restarted. Other schemes include timestamp-based protocls, the multiversion concurrency control scheme, and snapshot isolation. For interested students, we refer to Chapter 18 of Silberschatz et al. *Database System Concepts* (7th Edition).

Durability The durability property guarantees that all modifications executed by a transaction that has committed are persistent even in the case of system failure. A transaction cannot be rolled back once it has committed. In other words, after a transaction has committed, the modifications can only be undone by a subsequent transaction that reverts these modifications. The informations that is recorded in the log file (cf. Atomicity) allows the DBS to redo a modification if the system needs to restart (e.g., due to a system failure), thus ensuring durability. Only when the last information is written to the log file, the transaction completes (i.e., it transitions into the "committed" state). The recovery systems are also subject to active research and we refer interested students to Chapter 19 of Silberschatz et al. *Database System Concepts* (7th Edition) for more details.