**Distributed Data Management (WiSe 2025/2026)**

**Exercise Assignment Sheet 1 - Recapitulation**

**Submitted by: Suvansh Shukla Matriculation No: 256245**

**1. SQL and Relational Algebra**

1. The following are some primitive operations of relational algebra:

* Selection
* Projection
* Cartesian product (cross product)
* Set union
* Set difference
* Rename

The following are some non-primitive operations of relational algebra:

* Natural join
* Theta-Join and Equi-Join
* Semi-Join
* Outer-Join
* Set Intersection

2. A natural join is a type of join that returns values from two tables by taking rows from both tables that have the same value for a given column. A simple example of this would be trying to find out which employees work in department 2. To do so we would perform a natural join on the employees table and the departments table and with the department\_id column being common in both tables we would perform a selection with a where clause where the department\_id would equal to 2.

3. A semi-join only returns the results or columns from one of the tables, skipping the columns of the other. The condition applied to join the tables is the same as a natural join.

A semi-join can be expressed using natural join, selection and projection in the following way:

It can also be expressed using a Cartesian product like so:

4. a)

b)

**2. Transactions**

2.1: Transactions are a set of commands that are executed on a database in an ordered or unordered manner.

2.2: Atomicity: Transactions cannot be broken down into smaller transactions and they must be processed completely.

Consistency: Data that is saved in the database should be the same throughout and should not have any differing properties.

Isolation: parallel transactions being executed on the database should not interfere with each other

Durability: changes caused by transactions should persist in the database, even if there is a system failure.

**3. Database Design – Functional Dependencies and Normal Forms**

3.1 - Functional dependencies are a type of relation where one attribute uniquely identifies another attribute uniquely.

3.2 – The **first Normal form** stipulates that only atomic values may be stored in fields or columns. To bring a table to first normal form, none of its columns should have multiple values, it should have a primary key and it should not have any duplicated rows or columns.

The **Second Normal Form** stipulates that for a table to be in 2NF it should already be in 1NF and it should have no partial dependency, i.e. all keys should depend on the whole of the primary key. To bring a table in 1NF to 2NF we generally have to separate out columns into new tables so that each of their columns only rely on new whole primary keys of their respective tables.

The **Third Normal Form** states that for a table to be in 3NF it should already be in 2NF and it should have no transitive dependencies. To move a table from 2NF to 3NF you usually need to move the transitively dependent key along with its parent into a separate table.

3.3 – Based on the given data, the following functional dependencies exist:

Id -> name

Id -> size

Id -> pos

Team -> division

From -> to

The table is already in 1NF (first normal form by default) because there are no columns with multiple values.

For the table to be in 2NF, we need to make sure that there are no partial dependencies.

So according to the functional dependencies the tables will be as follows:

T1(id, name, size) T2(team, division) T3(from, to)

For a table to be in 3NF it the table must not have any transitive dependencies. From the above three tables (T1, T2, T3) we cannot find any transitive dependencies hence the tables are now in 3NF.