Platz Nummer	

Unterweisung für Studierende
COVID-19-Schutzmaßnahmen an der TU Wien

Diese Unterweisung bezieht sich auf die COVID-19 Schutzmaßnahmen, die an der TU Wien einzuhalten sind.

Inhalt der Unterweisung (siehe beiliegendes Infoblatt)

Allgemeine Vorgaben

- Die Hände sind regelmäßig und gründlich zu reinigen.
- Schutzabstand einhalten.
- Richtige Husten- und Niesetikette beachten (bitte Ellenbogenbeuge vorhalten)
- Händeschütteln und Körperkontakt vermeiden
- Es dürfen keine Gruppen innerhalb der Gebäude bzw. auf dem Gelände der TU Wien gebildet werden.
- Bei Unwohlsein oder Erkrankung besteht die Möglichkeit sich auch kurzfristig noch von der Prüfung abzumelden.
- Bei Erkrankung UNBEDINGT zu Hause bleiben.

Sicherheitsabstand einhalten

Die wichtigste Schutzmaßnahme gegen COVID-19-Infektionen ist der Sicherheitsabstand von mindestens 1 m (besser 1,5 m). Dieser muss innerhalb aller Areale der TU Wien eingehalten werden.

Schutzmasken

MNS-Masken müssen von den Studierenden bereits vor dem Zutritt und bis zum Verlassen des Gebäudes getragen werden. Das Tragen von Mund-Nasen-Schutzmasken (MNS-Masken) ist in allen allgemeinen Räumlichkeiten verpflichtend. Hier ist ein Kontakt mit anderen Personen sehr wahrscheinlich und es kann nicht sichergestellt werden, dass der notwendige Sicherheitsabstand eingehalten wird. Der Mindestabstand von 1 m (besser 1,5 m) ist trotz MNS-Maske anzustreben.

Während der Prüfung können die Masken abgenommen werden.

Präventions- und Hygienemaßnahmen

Grundsätzlich gilt: Waschen Sie Ihre Hände regelmäßig und gründlich mit Seife. Das ist für die Haut weniger belastend als Desinfektion und erhält die als Ansteckungsschutz nötige Hautbarriere.

Direkt bei den Eingängen sind Desinfektionsstellen eingerichtet, an denen man sich beim Betreten der Gebäude der TU Wien die Hände desinfizieren kann.

Alle Räumlichkeiten, in denen Prüfungen bzw. Distance-Learning-Aktivitäten stattfinden, werden täglich mehrmals gründlich gereinigt und häufige Kontaktstellen desinfiziert.

Nach jeder Prüfung werden die Tischflächen gereinigt und desinfiziert. In allen Lehrräumen werden Handdesinfektionsmittel zur Verfügung gestellt.

Allgemein: Auf die Hygiene ist unbedingt zu achten (regelmäßiges Händewaschen, keine Berührungen der Mitmenschen, Atemhygiene)!

Risikominimierung

Jede_r, der Krankheitssymptome wie Husten, Fieber, Geschmacksverlust, etc. aufweist oder befürchtet, muss jedenfalls zu Hause bleiben und sofort die telefonische Gesundheitsberatung unter 1450 kontaktieren.

Ich erkläre hiermit, dass ich über die Schutzmaßnahmen unterrichtet wurde, diese verstanden und zur Kenntnis genommen habe und verpflichte mich, diese einzuhalten.

……………………………… ………………………………………………….…

Datum Name und Unterschrift

DO NOT TOUCH BEFORE EXAM STARTS

Stand: Mai 2020

Group **A**

Please fill in your name and registration number (Matrikelnr.) **immediately**.

Exam 1, 2, and 3 ON	**SOLUTION KEY** Advanced Database Systems (**184.780**)	29 June, 2020 **GROUP A**
Matrikelnr.	Last Name	First Name

Duration: 120 minutes. Provide the solutions on the designated pages. **Good Luck!**

Question 1: (12)

For each of the statements below, decide if it is true or false and tick the corresponding circle. You get +2 credits for each correct answer, −2 credits for each wrong answer and 0 credit if you leave the answer open. Each of the parts (a) – (c) is graded separately and you always get ≥ 0 credits on each part of exam question 1, that is: 0 – 4 credits for each of the parts (a) – (c).

(a) Part 1: [4 credits]

1. Consider two relations $R(\underline{A}, B)$ and $S(\underline{C}, A)$, where $S.A$ is a foreign key from S to R, and suppose that there exists a B^*-index for each primary key. Then the implementation of the natural join $R \bowtie S$ by an index nested loops join always requires a smaller number of I/O-operations than the implementation of the same join by a hash join. true ◯ false ⊗
2. Consider two relations $R(ABC)$ und $S(ABD)$. Then the following equalities always holds: $R \bowtie (\pi_{AB}(R) \cap \pi_{AB}(S)) = \pi_{ABC}(R \bowtie S) = \pi_{ABC}(S \bowtie R)$ true ⊗ false ◯

(b) Part 2: [4 credits]

1. HDFS splits files into blocks with a recommended block size between 64 and 128 kBytes. true ◯ false ⊗
2. In MapReduce, combiners allow the user to reduce the communication cost by pushing some of the map logic into the reduce task. true ◯ false ⊗

(c) Part 3: [4 credits]

1. If a distributed database management system provides linear scale up, then doubling the number of CPUs allows one to halve the time needed for a given operation. true ◯ false ⊗
2. In a document store, the data may have an aribtrarily deeply nested structure (e.g., JSON, XML). In contrast, in key value stores, no nesting of values is allowed. true ◯ false ⊗

Question 2: (8)

The American presidential elections 2020 are coming closer and the parties are already making plans for their campaigns in the next months. Above all, the parties want to make sure of the support of their own members. To this end, they maintain a database of volunteers, members, and activities how to contact the members. The database contains the following relations:

`Volunteers(`<u>`Id`</u>`, Name, Age, Sex)` (short v)
`Members(`<u>`Id`</u>`, Name, Education, Age, State)` (short m)
`Contact(`<u>`Id`</u>`, VolId, MemId, Method, Date)` (short c).

Further, suppose that the following query has to be processed:

```
SELECT v.Id, m.Name, m.Education, c.Date
FROM   Volunteers v, Members m, Contact c
WHERE  v.Sex = 'female'
AND    m.Age > 60
AND    c.VolId = v.Id
AND    c.MemId = m.Id
```

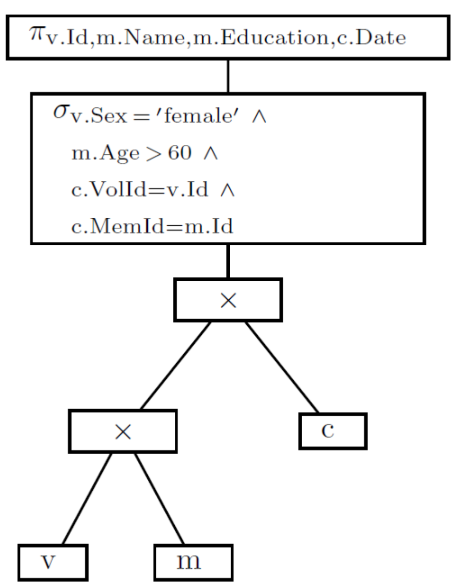
i.e., we are interested in information on female volunteers who are supposed to contact elderly party members.

a) In the first box below, draw the query tree of the canonical translation. To save space, you may use the abbreviations v, m, and c, for the relations Volunteers, Members, and Contact, respectively. You may also abbreviate any attribute name to a unique prefix (e.g., m.N for Members.Name or c.Mem for Contact.MemId).

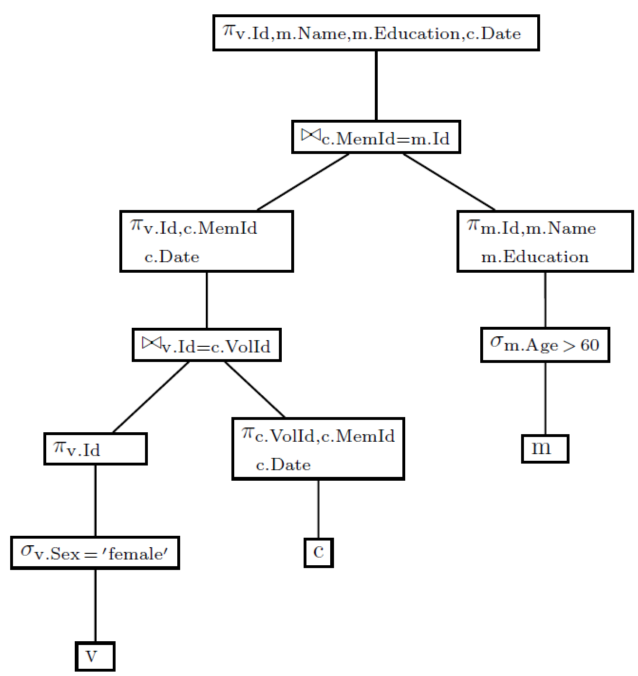
b) In the second box below, draw the query tree of the optimized Relational Algebra expression. For the optimization, apply the following rules:

- Push selections as deep in the tree as possible,
- project out attributes as soon as they are not needed anymore,
- replace cross products by joins.

(a) Canonical Translation: [3 credits]



(b) Optimized Expression: [5 credits]



Question 3: (8)

a) [4 credits]

Consider a relation $\texttt{Meta}(\underline{s},\ vid)$ with 1000 elements and a relation $\texttt{Video}(\underline{vid},\ digital)$ with 6000 elements. Attribute *vid* of `Meta` is a **foreign key** to *vid* in `Video` with a `NOT NULL` constraint. Furthermore, you know that there are 300 unique values for attribute *digital* in `Video`.

Using the assumption that occurrence of values in *digital* is uniformly distributed, compute the **cardinality** of the following relational algebra expression.

$$\texttt{Meta} \bowtie \sigma_{digital="besser"}(\texttt{Video})$$

We see that the selection $digital = "besser"$ has a selectivity of $1/300$. Because we always assume uniformity, the tuples in Meta have a $1/300$th chance of keeping their join partner (they have exactly one before the selection because of the FK constraint) . Hence, only $1000/300 = 3.3$ tuples are left in the join. Since tuples can not be fractional, we arrive at a cardinality of 3 for the expression.

b) [4 credits]

Now consider a DBMS that maintains *equi-depth* histograms to support selectivity calculation for query planning. In particular, for the column `score` of **integers** of a table `adbsexam` with 280 rows, the following 5 values divide the column values into 6 groups of equal size: $41, 52, 67, 73, 85$. The maximum value in `score` is 100. **Assume that the dividing values in the histogram are always part of the left bucket, i.e., the value 41 is in the first bucket, 52 in the second, and so on.**

Estimate the **selectivity** for the following relational algebra expression. Assume that values are evenly distributed inside the buckets.

$$\sigma_{score<52 \vee score \geq 91}(\texttt{adbsexam})$$

The disjuncts clearly capture disjoint events and we can therefore simply add their respective individual selectivities.

i) score $<$ 52 : $1/6$ + range from 42 to 51 (10 values) of the bucket that ranges from 42 to 52 (11 values) $(\frac{1}{6} \cdot r)$.

$$r = \frac{51 - 41}{52 - 41} = \frac{10}{11}$$

Thus, the selectivity of the first disjunct is $\frac{1}{6}(1 + \frac{10}{11})$.

ii) score $\geq$ 91: part of last bucket $(\frac{1}{6} \cdot r')$. The range $[91, 100]$ contains 10 values. The total bucket spans the values $[86, 100]$, i.e., 15 values. Hence, assuming uniformity

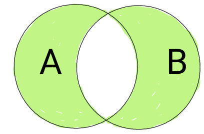
$$r' = \frac{10}{15}$$

The overall selectivity is therefore $\frac{1}{6}(1 + \frac{10}{11} + \frac{2}{3})$.

Question 4: (8)

Below we define the symmetric set difference operation.

- **Symmetric Set Difference**:
  $A \triangle B = (A \setminus B) \cup (B \setminus A)$, is the set of tuples which are in exactly one of the two sets A or B, and not in their intersection. A Venn diagram illustrating this can be seen here:



Sketch out a MapReduce algorithm for computing the symmetric set difference $A \triangle B$. In addition to this, also identify the communication cost of your algorithm, as a function of the input sizes.

Mapper

For each input t from A produce a tuple $(t, \text{'A'})$ and for each input t from B produce a tuple $(t, \text{'B'})$

Reducer

Count the occurrences of 'A', called a, and occurrences of 'B', called b, in the value list of t. In case that $a = 0$ and $b > 0$, then emit t, or if $a > 0$ and $b = 0$, then emit t. In all other cases, do nothing.

Communication cost

The mapper emits for every tuple from A and every tuple from B one output record, therefore the communication cost is $|A| + |B|$.

Question 5: (8)

You are given a database for a pharmaceutical testing facility with the following relational schema.
Note: underlines represent primary keys, *italics* represent foreign keys. We do not state the type of attributes here, just assume some reasonable defaults (string, int, float, . . .).

pharma_comp	(name, country)
drug	(name, producer: *pharma_comp.name*)
med_version	(pred: *drug.name*, succ: *drug.name*)
trial	(title, date, length)
test_group	(name, size, title: *trial.title*, date: *trial.date*)
testing	(company: *pharma_comp.name*, drug: *drug.name*, title: *trial.title*, date: *trial.date*)

Assume the dataset is loaded and the corresponding views have already been created in Spark SQL. For the dataframes, simply use the name of each relation appended with "DF" (e.g.: trialDF, pharam_compDF, etc.). You do not need to generate these dataframes, just assume they are already loaded and usable.

You are given four queries, either in Spark SQL or in the Dataframe API. Your task is to state an equivalent query of the other type, i.e. give a Spark SQL equivalent if the Dataframe API version is given, or vice-versa.

a)
```
val query1SQL = spark.sql("SELECT t.title, t.date, COUNT(tg.name)
                           FROM trial t NATURAL JOIN test_group tg
                           GROUP BY t.title, t.date")
```

```
val query1DF = trialDF.join(test_groupDF, trialDF("title") === test_groupDF("titl
e") && trialDF("date") === test_groupDF("date") ).groupBy(trialDF("title"),trialD
F("date")).agg(count(test_groupDF("name")))
```

b)
```
val query2DF = pharma_compDF.join(drugDF, pharma_compDF("name") === drugDF("producer"))
                            .groupBy(pharma_compDF("name")).agg(count(drugDF("name")))
```

```
val query2SQL = spark.sql("SELECT p.name, COUNT(m.name) FROM pharma_comp p, drug
m WHERE m.producer = p.name GROUP BY p.name")
```

c)
```
val query3SQL = spark.sql("SELECT p.name as comp, m.name as med, SUM(mv.pred) as sum
                          FROM pharma_comp p, drug m, med_version mv
                          WHERE p.name = m.producer AND m.name = mv.succ
                          GROUP BY p.name, m.name
                          HAVING sum >= 2")
```

```
val query3DF = pharma_compDF.join(drugDF, pharma_compDF("name") === drugDF("produ␣
cer")).join(med_versionDF,drugDF("name") === med_versionDF("succ")).groupBy(pharm␣
a_compDF("name"),drugDF("name")).agg(sum("pred").as("sum")).select(pharma_compDF(␣
"name").as("comp"), drugDF("name").as("med"), $"sum").filter("sum >= 2")
```

Note: You may shorten dateframe names (e.g.: "pcDF" instead of "pharma_compDF"), as long as it is unambiguous what the shortened name refers to.

d)
```
val query4DF = testingDF.groupBy(testingDF("company"),testingDF("drug")).agg(count("*"))
   .except(
            testingDF.join(test_groupDF, testingDF("title") === test_groupDF("title") &&
                       testingDF("date") === test_groupDF("date"))
                      .select(testingDF("*")).distinct()
                      .groupBy(testingDF("company"),testingDF("drug")).agg(count("*"))
           )
```

```
val query4SQL = spark.sql("SELECT t.company, t.drug, COUNT(*) as count FROM tes␣
ting t WHERE NOT EXISTS (SELECT * FROM test_group tg WHERE (tg.title,tg.date) =
(t.title,t.date) ) GROUP BY t.company, t.drug ")
```

Question 6: (8)

Suppose that a streaming provider maintains information on movies and users in a relational database with the following schema:

Users (id, name, email, balance),
Movies (id, title, genre, released), and
Watched (uid, mid, date, time),

where the underlined attributes are primary keys and relation Watched has foreign keys *uid* to the Users relation and *mid* to the Movies relation. Suppose that these tables have the following content:

Users			
id	**name**	**email**	**balance**
u1	Alice	a@gmx.at	-150.00
u2	Bob	b@gmx.at	0.00
u3	Carol	c@gmx.at	200.00

Movies			
id	**title**	**genre**	**released**
m1	title1	drama	2019
m2	title2	action	1998
m3	title3	comedy	2010

Watched			
uid	**mid**	**date**	**time**
u1	m1	2020-04-01	20:15
u1	m3	2020-04-02	21:00
u2	m1	2020-05-01	22:00
u3	m2	2020-06-01	23:00

In part (a), this relational database should be transformed into a **document store (in JSON)**. Recall that for the data design it is sometimes advantageous to apply some form of **denormalization** if you want to speed up certain queries or update operations. In your transformation of the relational database into a document store, you will be requested to apply at least one denormalization. Moreover, you will be requested to discuss the pros (in part b) and cons (in part c) of the chosen denormalization.

(a) Give a representation of the relational database as a document store (in JSON) and make sure that you apply at least one denormalization. [4 credits]

```
db.question6.users([
{ _id: "u1",
name: "Alice",
email: "a@gmx.at",
balance: -150.00 },

{ _id: "u2",
name: "Bob",
email: "b@gmx.at",
balance: 0.00 },

{ _id: "u3",
name: "Carol",
email: "c@gmx.at",
balance: 200.00 }
]);
```

```
db.question6.watched([
{ _id: "w1",
uid: "u1",
mid: "m1",
title: "title1",
genre: "drama",
released: 2019,
date: "2020-04-01",
time: "20:15" },

{ _id: "w2",
uid: "u1",
mid: "m3",
title: "title3",
genre: "comedy",
released: 2010,
date: "2020-04-02",
time: "21:00" },
```

```
% db.question6.watched continued
{ _id: "w3",
uid: "u2",
mid: "m1",
title: "title1",
genre: "drama",
released: 2019,
date: "2020-05-01",
time: "22:00" },

{ _id: "w4",
uid: "u3",
mid: "m2",
title: "title2",
genre: "action",
released: 1998,
date: "2020-06-01",
time: "23:00" }
]);
```

Remark. Many students solved this problem by realizing the watched-information as an array inside the users- or movies-collection. Strictly speaking, this is not a denormalization; it is just one possible way of realizing an m:n relationship in a document store. However, this was only mildly penalized (-1 credit); moreover, if the reasoning in parts (b) and (c) made sense, full credits were given for those parts.

(b) Present a query or update operation that should **profit from your denormalization** and **explain why** it should profit, i.e., what kind of work by the dabase engine can be avoided because of your denormalization. You may use plain text for the presentation of your query or update operation; no formal query language is required here. [2 credits]

```
Database access:  a query that asks for the most popular genre, i.e.:  movies of which genre
are watched most often?

Savings:  due to the denormalization of the movies-information into the watched-collection,
the database engine only needs to read in the watched-collection; no join with a separate
movies-collection is needed.
```

(c) Present a query or update operation that might **suffer from your denormalization** and **explain why** it might suffer, i.e., what kind of extra work by the dabase engine might be required because of your denormalization. You may use plain text for the presentation of your query or update operation; no formal query language is required here. [2 credits]

```
Database access:  an update that changes the genre of some movie

Additional cost:  due to the denormalization of the movies-information into the
watched-collection, the database engine needs to iterate through the watched-collection
and change the genre-field of all entries that refer to that movie.  Without the
denormalization, only a single entry in the movies-collection would have to be modified.
```

Question 7: (8)

(a) Evaluate the following *Cypher* query on the Database given on the last sheet of the exam:

```
match (f)<-[:likes]-(b:Band)-[:plays]->(v:Venue)-[:blocks]->(f)
return f,b,v
```

```
(5kHD,Dives,Flex), (GY!BE,RTJ,B72)
```

(b) Evaluate the following *Cypher* query on the Database given on the last sheet of the exam:

```
MATCH p = shortestPath( (x:Venue)-[*]->(y:Venue) )
WHERE x<>y AND x.cap < 300
RETURN x.name, y.name, length(p);
```

```
(Flex, Arena, 5),(Flex, Werk, 2), (Flex, B72, 4), (B72, Arena, 2), (B72, Werk, 3)
```

(c) *Assume the data model described on the last sheet of the exam.* Write a *Cypher* query that returns all bands with the number of venues that they have been blocked from. Sort the output by the number of venues that have blocked the band.

```
MATCH (b:Band)-[r:blocks]-(v:Venue) return b, count(r) as c order by c
```

(d) *Assume the data model described on the last sheet of the exam.* Write a *Cypher* query that finds all bands that have played at or been blocked from a venue that does *not* contain a `cool` attribute. Return the band name, the kind of connect (played or blocked) and the venue for each occurence. If a band has both a played and a blocked connection to a venue, return a tuple for both cases, i.e., *(band1, played, venue1)* and *(band1, blocked, venue1)*.

```
MATCH (b:Band)-[r:blocks|:plays]-(v:Venue)
WHERE v.cool is null
RETURN b.name, type(r), v.name
```

Good luck!

Overall: 60 points

Graph DB Data Model

The graph contains bands (`:Band`) and venues (`:Venue`). A venue always has an attribute `cap` that stores the capacity.

There are three types of relationships. A band `:likes` other bands. A band `:plays` at venues. A venue `:blocks` bands. A visual representation of the data model is given in Figure **??**.

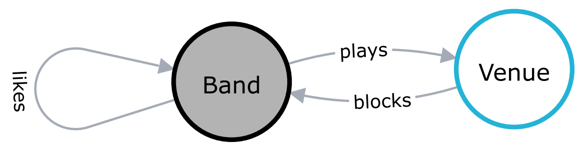


Figure 1: Data Model

Graph Database

The labels contain the ***`name`*** *attributes for* ***`Band`*** *nodes and the* ***`name`*** *and* ***`cap`*** *attributes for* ***`Vanue`*** *nodes.*

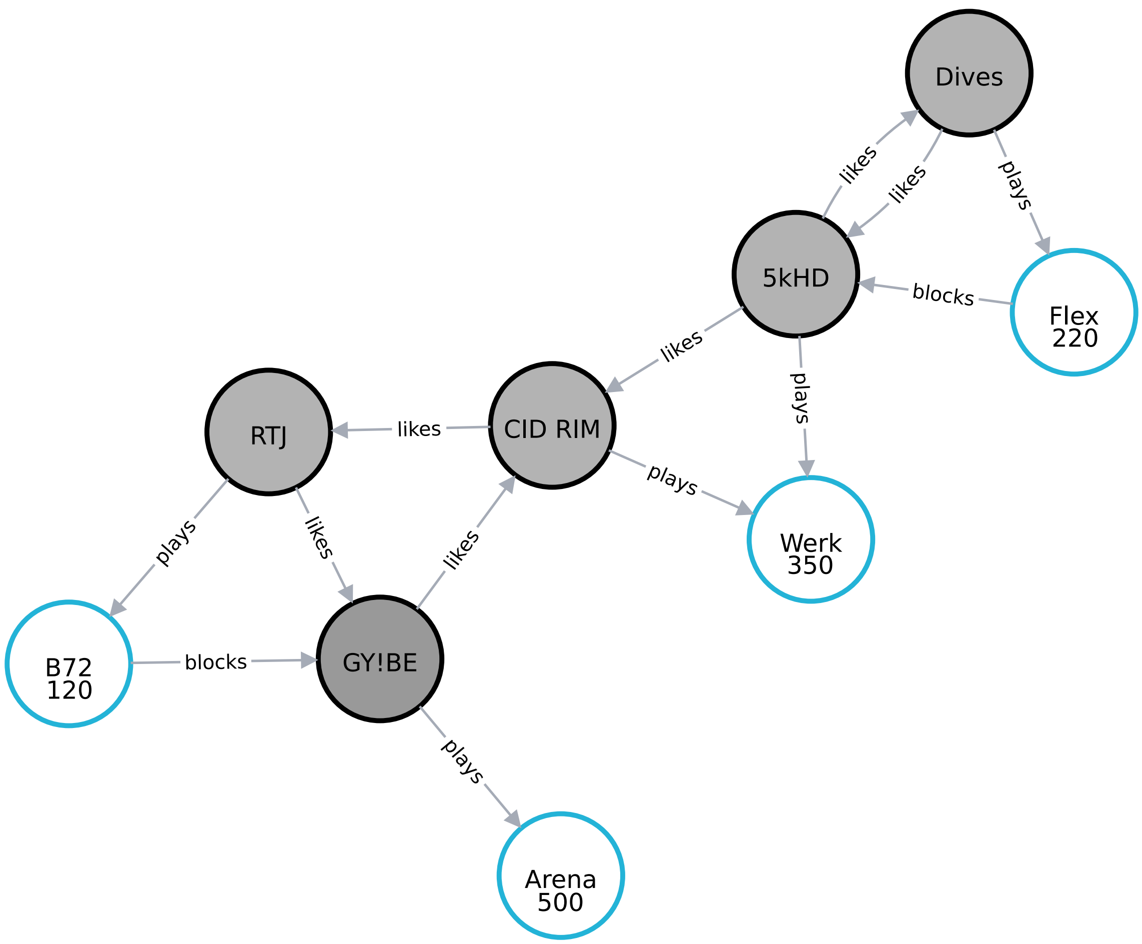


Figure 2: Database