TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe25

Alice Rey, Maximilian Reif, Tobias Goetz (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss25/impldb/

Blatt Nr. 07

Hinweise **Beachten Sie, dass dieses Blatt in KW24 und KW25 behandelt wird. Die Übungen werden zwischen 12.06. und 18.06. abgehalten. Übung 32 finden am 18.05. abweichend in 02.13.008 statt.**

Machen sie sich mit Raft vertraut: `https://thesecretlivesofdata.com/raft/`.

Hausaufgabe 1

Zeigen Sie, dass die *write-all / read-any* Methode zur Synchronisation replizierter Daten einen Spezialfall der *Quorum-Consensus*-Methode darstellt.

- Für welche Art von Workloads eignet sich dieses Verfahren besonders gut?
- Wie werden Stimmen zugeordnet um *write-all / read-any* zu simulieren?
- Wie müssen die Quoren Q_w und Q_r vergeben werden?

Dieses Verfahren fordert einen sehr großen Aufwand beim Schreiben, aber nur minimalen Aufwand beim Lesen. Daher eignet es sich besonders gut für Workloads in denen wesentlich mehr Daten gelesen als geschrieben werden. Dies entspricht z.B. analytischen Anfragen.
Siehe Übungsbuch.

Hausaufgabe 2

Um Ausfallsicherheit zu garantieren, ist ein Datenwert 'A' auf vier Rechnern verteilt. Jeder Rechner hält dabei eine vollständige Kopie von 'A'. Um Konsistenz zu garantieren, wird das Quorum-Consensus-Verfahren eingesetzt. Dabei ist jedem Rechner ein Gewicht $w_i(A)$ wie folgt zugewiesen:

Rechner	Kopie	Gewicht
R_1	A_1	3
R_2	A_2	1
R_3	A_3	2
R_4	A_4	2

Das Lesequorum ist $Q_r(A) = 4$ und das Schreibquorum is $Q_w(A) = 5$.

a) Geben Sie **alle** Lesemöglichkeiten für eine Transaktion auf dem Datum 'A' nach dem Quorum-Consensus-Protokoll an.

A_1,A_2
A_1,A_2,A_3
A_1,A_2,A_3,A_4
A_1,A_2,A_4
A_1,A_3
A_1,A_3,A_4
A_1,A_4
A_2,A_3,A_4
A_3,A_4

b) Geben Sie **alle** Schreibmöglichkeiten für eine Transaktion auf dem Datum 'A' nach dem Quorum-Consensus-Protokoll an.

A_1,A_2,A_3
A_1,A_2,A_3,A_4
A_1,A_2,A_4
A_1,A_3
A_1,A_3,A_4
A_1,A_4
A_2,A_3,A_4

c) Zeigen Sie für dieses Beispiel, dass, während eine Transaktion T_1 ein Schreibquorum auf A hält, es für andere Transaktionen T_x nicht möglich ist, ein Lesequorum für A zu bekommen.

Zum Schreiben muss die Transaktion T_1 mindestens Kopien mit einem Gesamtgewicht von 5 gesperrt haben. Insgesamt haben alle Kopien zusammen das Gewicht 8. Somit können maximal Kopien mit einem Gewicht von zusammen 3 übrig bleiben, womit das Lesequorum von 4 nicht mehr erfüllt werden kann.

Hausaufgabe (wird nicht in der Übung besprochen) Um ein Gefühl für das *Raft Consensus Protokoll* zu bekommen, führen Sie folgende Aktionen mit *RaftScope* unter `https://raft.github.io/` aus. Die Simulation kann mit einem Klick auf das Uhr Symbol gestoppt und gestartet werden. Ein Rechtsklick auf einen der Server öffnet ein Menü um Aktionen auszulösen.

Führen Sie folgende Aktion aus und beschreiben Sie in Stichpunkten was passiert.

1. Warten sie bis die erste Leader Election abgeschlossen ist.
2. Senden Sie einen Request an den Leader. (Rechtsklick auf Leader)
3. Stoppen Sie den Leader (Server ausschalten), warten Sie bis ein neuer Leader gewählt wurde, und senden Sie einen Request an den neuen Leader.
4. Wiederholen Sie den vorigen Schritt 2x bis nur noch 2 Knoten übrig sind. Wird ein neuer Leader gewählt?
5. Starten Sie (resume) wieder einen weiteren Server. Wird ein neuer Leader gewählt?

Lösung:

1. Am Ende der ersten Leader Election (z.B. von S2) sind alle Knoten blau markiert und in der zweiten Phase.

2. Der Leader hängt den Request in seinem Log an und leitet ihn an alle Follower weiter. Alle erreichbaren Follower merken sich den Request im Log vor (gestrichelte Linie) und bestätigen diesen. Wenn die Mehrheit bestätigt, updated der Leader seinen Status. In der nächsten Hearbeat-Nachricht teilt der Leader das Update seines Status allen Followern mit. Diese setzen den vorgemerkten Log auf gültig (durchgezogene Linie im Log).

3. Der gestoppte Leader sendet keine Hearbeat-Nachrichten mehr. Dadurch läuft der Timeout der Follower weiter und die ersten Follower-Knoten, bei denen der Timer abläuft werden zu Candidates. Dies löst eine neue Leader Election durch die Mehrheit aller Knoten aus. Der neu gewählte Leader verarbeitet den Request wie oben beschrieben.

4. Beim zweiten gestoppten Knoten verläuft das Stoppen des Leaders wie in der vorigen Aufgabe. Beim dritten gestoppten Knoten ist nicht mehr die Mehrheit der Knoten online und es kann kein neuer Leader gewählt werden. Die verbleibenden Knoten versuchen weiter als Candidate genug Stimmen zu erhalten. Es kann kein neuer Leader gewählt werden.

5. Der erste Candidate der einen Timeout hat, fragt alle Knoten nach einer Stimme und bekommt die nötige Mehrheit. Mit dieser Nachricht wird die Phase des neu gestarteten Knotens aktualisiert. Der Candidate wird als neuer Leader gewählt.

Hausaufgabe 3

Beantworten Sie folgende Fragen zum RAFT Protokoll. Verwenden Sie *RaftScope* unter `https://raft.github.io/` um Ihre Vermutungen zu bestätigen.

1. Wie viele Server müssen ausfallen, dass in einem Cluster mit n Servern kein neuer Leader bestimmt werden kann?
2. Wie können Sie mit `restart` und `time-out` in RaftScope einen bestimmten Knoten als neuen Leader erzwingen? Geben Sie die Schritte an.
3. Wie verläuft das Beispiel in Abbildung 1 weiter? Wie kann sicher gestellt werden, dass der neu gewählte Leader den neuesten Logeintrag hält? Beschreiben Sie in Stichpunkten.

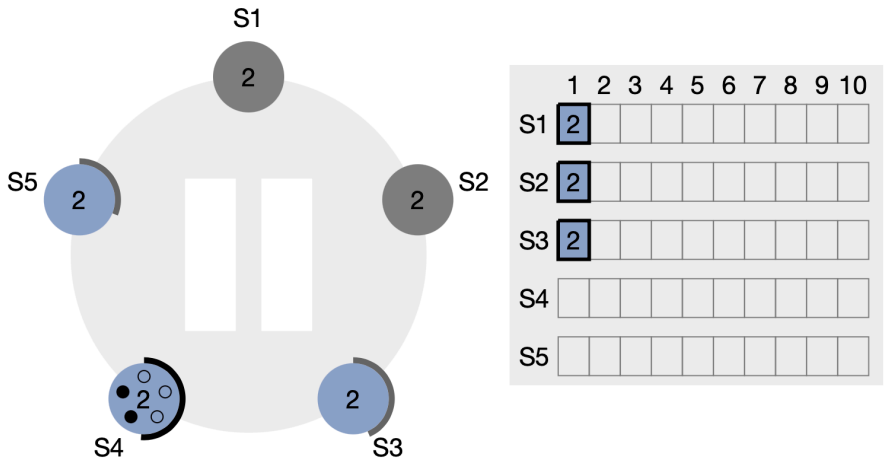

Abbildung 1: Beispiel eines Status in RaftScope

Lösung:

1. Es müssen mindestens $\lceil n/2 \rceil$ Server ausfallen.
2. Folgende Schritte sind nötig um einen Leader zu erzwingen:
   a) Starten sie den aktuellen Leader mit `restart` neu. Er started als Follower.
   b) Nutzen Sie `time-out` um beim gewünschten Knoten den Timout abzubrechen.
   c) Der Knoten wird Candidate und holt sich Stimmen ein und wird neuer Leader.

   Dies zeigt, dass der Knoten der zufällig den geringsten Time-Out hat der neue Leader wird.
3. Folgende Schritte verlaufen als nächstes in Abbildung 1.
   a) S4 ist ein Candidate, aber der Log von S4 enthält nicht den letzten comitteden Log-Eintrag.
   b) S3 stimmt nicht für S4, weil dessen eigener Log *aktueller* ist.
   c) S4 wird nicht Leader, weil die Mehrheit der Knoten einen *akutelleren* Logeintrag als S4 haben.
   d) Irgendwann hat ein anderer Knoten, zum Beispiel S3 (mit dem neusten committeden Log-Eintrag), zuerst einen Timeout haben und wird neuer Candidate.
   e) Die anderen Knoten wählen S3, weil S3 den *aktuellsten* Log-Zustand hat und somit wird S3 zum neuen Leader.
   f) S3 stellt als neuer Leader sicher, dass alle Knoten den *aktuellsten* Logeintrag haben.
   g) Dieses Szenario funktioniert immer, denn:
      i. Die Mehrheit der Server enthalten den zuletzt committeden Log-Eintrag.
      ii. Die Mehrheit der Server werden benötigt um einen neuen Leader zu wählen.

      Beide Mehrheiten des selben Clusters müssen sich zur erfolgreichen Wahl des Leaders überschneiden. Irgendwann wird ein neuer Leader mit dem neuesten committeden Log-Eintrag gewählt.

Schritte um das Beispiel aus der 3. Teilaufgabe in *RaftScope* zu simulieren:

Hausaufgabe 4

Analysieren wir die Gehälter von Professoren mittels Windowfunctions und führen Sie die Abfragen unter `hyper-db.de` aus. Dazu orientieren wir uns an der Relation *Professoren* des erweiterten Universitätsschemas:

```
with Professoren (persnr, name, rang, raum, gehalt, steuerklasse) as (
  values (2125,'Sokrates','C4',226,85000,1),
  (2126,'Russel','C4',232,80000,3),
  (2127,'Kopernikus','C3',310,65000,5),
  (2128,'Aristoteles','C4',250,85000,1),
  (2133,'Popper','C3',52,68000,1),
  (2134,'Augustinus','C3',309,55000,5),
  (2136,'Curie','C4',36,95000,3),
  (2137,'Kant','C4',7,98000,1)
)
```

1. Ermitteln Sie zu jedem Professor das Durchschnittsgehalt aller Professoren.

   ```
   SELECT *, avg(gehalt) OVER () FROM Professoren;
   ```

2. Ermitteln Sie zu jedem Professor das Durchschnittsgehalt aller Professoren partitioniert nach Rang.

   ```
   SELECT *, avg(gehalt) OVER (PARTITION BY rang) FROM Professoren;
   ```

3. Ermitteln Sie nun die wachsende Summe (das Quantil) des Gehaltes aller Professoren partitioniert nach Rang und absteigend sortiert nach ihrem Gehalt. Gleich verdienende Professoren sind im selben Quartil.

   ```
   SELECT *, sum(gehalt) OVER (PARTITION BY rang ORDER BY gehalt DESC)
   FROM Professoren;
   -- aequivalent zu
   SELECT *, sum(gehalt) OVER (PARTITION BY rang ORDER BY gehalt DESC
   RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) FROM Professoren;
   ```

4. Ermitteln Sie nun die wachsende Summe des Gehaltes aller Professoren partitioniert nach Rang und absteigend (total) sortiert nach ihrem Gehalt (reihenweise, nicht als Range-Query).

   ```
   SELECT *, sum(gehalt) OVER (PARTITION BY rang ORDER BY gehalt DESC
   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) FROM Professoren;
   ```

5. Ermitteln Sie nun das gleitende Durchschnittsgehalt aus genau zwei mehr bzw. weniger verdienenden Professoren sortiert nach Gehalt und partitioniert nach Rang.

   ```
   SELECT *, avg(gehalt) OVER (PARTITION BY rang ORDER BY gehalt DESC
   ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) FROM Professoren;
   ```

6. Ermitteln Sie nun das gleitende Durchschnittsgehalt aus den 500 Einheiten mehr bzw. weniger verdienenden Professoren sortiert nach Gehalt und partitioniert nach Rang.

   ```
   SELECT *, avg(gehalt) OVER (PARTITION BY rang ORDER BY gehalt DESC
   RANGE BETWEEN 500 PRECEDING AND 500 FOLLOWING) FROM Professoren;
   ```

7. Ergänzen Sie zu jedem Professor das Gehalt des eins besser wie eins schlechter verdienenden.

   ```
   SELECT *, lag(gehalt) OVER (ORDER BY gehalt DESC),
     lead(gehalt) OVER (ORDER BY gehalt DESC) FROM Professoren;
   ```

8. Ermitteln Sie die drei bestverdienendsten Professoren einmal mit und einmal ohne Windowfunctions.

```
SELECT * FROM (
   SELECT *, rank() OVER (ORDER BY gehalt desc) FROM Professoren
) WHERE rank < 4
```

```
SELECT * FROM professoren p_selbst WHERE 3>(SELECT count(*)
FROM professoren p_reich WHERE p_selbst.gehalt < p_reich.gehalt)
```

Hausaufgabe (wird nicht in der Übung besprochen) Lösen Sie folgende Anfrage mit SQL basierend auf dem bekannten Universitätsschema.

1. Bestimmen Sie die Durchschnittsnote für jeden Studenten.
2. Basierend auf dieser Durchschnittsnote, bestimmen Sie für alle Studenten ihren Rangplatz innerhalb ihrer Kohorte (Studenten desselben Semesters).
3. Berechnen Sie zusätzlich für jeden Studenten auch noch die **Abweichung** seiner Durchschnittsnote von der Durchschnittsnote der Kohorte (also vom Durchschnitt der Durchschnittsnote der Studenten der Kohorte) ausgegeben werden.

Lösen Sie Teilaufgaben 2 und 3 jeweils einmal mit und einmal ohne Nutzung von Windowfunktionen. Ihre Anfragen können Sie auf `hyper-db.de` testen. Nutzen Sie folgende erweiterte *pruefen* Relation:

```
with mehr_pruefen(MatrNr,VorlNr,PersNr,Note) as (
 select * from pruefen
 union
 values (29120,0,0,3.0),(29555,0,0,2.0),(29555,0,0,1.3),(29555,0,0,1.0)
)
```

Erzeugen der Hilfstabellen:

```
with mehr_pruefen(MatrNr,VorlNr,PersNr,Note) as (
  select * from pruefen
    union
  values (29120,0,0,3.0),(29555,0,0,2.0),(29555,0,0,1.3),(29555,0,0,1.0)
), noten(MatrNr,Semester,Note) as (
  select s.matrnr, semester, avg(Note)
  from studenten s, mehr_pruefen p
  where s.matrnr=p.matrnr
  group by s.matrnr,semester
)
```

Ohne Windowfunktionen:

```
select *,
(select count(*)+1 from noten x
   where x.Semester=n.Semester and x.Note<n.Note) as Rang,
(select avg(x.Note) from noten x
   where x.Semester=n.Semester) as GPA,
(select avg(x.Note) from noten x
   where x.Semester=n.Semester) - note as Abweichung
from noten n order by semester, rang
```

Mit Windowfunktionen:

```
select *,
rank() over (partition by Semester order by Note asc) as Rang,
avg(Note) over (partition by Semester) as GPA,
avg(Note) over (partition by Semester) - note as Abweichung
from noten order by semester, rang
```

Hausaufgabe 5

Betrachten wir das bekannte Uni-Schema mit den Faktentabellen `hoeren` und `pruefen`.

1. Ermitteln Sie in SQL mittels Fensterfunktionen (Windowfunctions) die Top-3 Studenten pro Vorlesung und geben Sie deren Namen aus.

   ```
   SELECT Name
   FROM(
      SELECT s.Name, rank() over (PARTITION BY VorlNr ORDER BY Note)
      FROM pruefen p, Studenten s
      WHERE s.MatrNr = p.MatrNr
   )
   WHERE rank < 4
   ```

2. Ermitteln Sie mittels SQL-92, um wieviele Notenstufen Studenten, die die Vorlesung gehört haben, in der Prüfung besser abgeschnitten haben.

```
select anwesend.Note-abwesend.Note as besser
from(
   select avg(p.Note) as Note
   from studenten s, pruefen p
   where s.MatrNr=p.MatrNr and not exists (
     select *
     from hoeren h where h.MatrNr=s.MatrNr and h.VorlNr=p.VorlNr)
  ) abwesend,
   (select avg(p2.Note) Note
   from studenten s2, pruefen p2
   where s2.MatrNr=p2.MatrNr and exists (
     select * from hoeren h2
     where h2.MatrNr=s2.MatrNr and h2.VorlNr=p2.VorlNr)
  ) anwesend
```

Hausaufgabe (wird nicht in der Übung besprochen) Wenden wir nun Fensterfunktionen an dem fiktiven TPC-H Schema an.

1. Erstellen Sie in SQL eine Abfrage nach dem jährlichen Produktionsvolumen pro Jahr und Land. Verwenden Sie diese Abfrage als Hilfstabelle (`with`-Statement) in den folgenden Abfragen, orientieren Sie sich an TPC-H Anfrage 7.

   ```
   with volume as (
     select n_name, l_year, sum(volume) as revenue
     from (
        select n_name, extract(year from l_shipdate) as l_year,
           l_extendedprice * (1 - l_discount) as volume
        from supplier,lineitem,orders,nation
        where s_suppkey = l_suppkey
           and o_orderkey = l_orderkey and s_nationkey = n_nationkey
         ) as shipping
     group by n_name, l_year
   )
   ```

2. Ranken Sie Länder anhand ihres jährlichen Produktionsvolumens, auf Platz eins ist das Land mit dem höchsten Volumen, das je in einem Jahr getätigt worden ist.

   ```
   select *, rank() over (order by revenue desc) from volume;
   ```

3. Küren Sie nun die Jahressieger. Ranken Sie dazu die Länder partitioniert nach Jahr.

   ```
   select * from (
     select *, rank() over (partition by l_year order by revenue desc)
     from volume)
   where rank=1;
   ```

4. Ermitteln Sie nun das laufende Produktionsmittel der Länge drei (Vorjahr, Nachjahr, falls erfasst).

   ```
   select *, avg(revenue) over (
     partition by n_name order by l_year
     range between 1 preceding and 1 following)
     from volume
   ```

Hausaufgabe 6

Betrachten Sie die folgende Tabelle `Waren` mit verkauften Produkten in einem Supermarkt.

Die Spalte `verkauft` besagt, wieviele Einheiten des jeweiligen Produktes verkauft worden sind.

Name	Preis	Kategorie	Verkauft
Brot	1.00	Backwaren	8128
Butter	0.80	Kühlwaren	496
Grill	60.00	Haushalt	6
Steak	8.00	Kühlwaren	28
…	…	…	…

1 Ermitteln Sie in SQL mittels Fensterfunktionen (Windowfunctions) den prozentualen Umsatzanteil jedes Produktes innerhalb seiner Kategorie.

```
select name,kategorie,
      verkauft * preis * 100.0 / sum(verkauft * preis)
      over (partition by kategorie)
from Waren;
```

2 Ermitteln Sie in SQL mittels Fensterfunktionen (Windowfunctions) für jedes Produkt das Mittel der Verkaufszahlen aus den 5 besser verkauften (höhere Verkaufszahlen) Produkten geordnet nach Verkaufszahlen.

```
select name, avg(verkauft) over (order by verkauft desc
         rows between 5 preceding and current row)
from Waren;
```

3 Ermitteln Sie in SQL mittels Fensterfunktionen (Windowfunctions) die drei Produkte mit dem meisten Umsatz pro Kategorie.

```
select *
from ( select *, rank() over (partition by kategorie
                        order by (verkauft * preis) desc)
      from waren)
where rank <= 3;
```