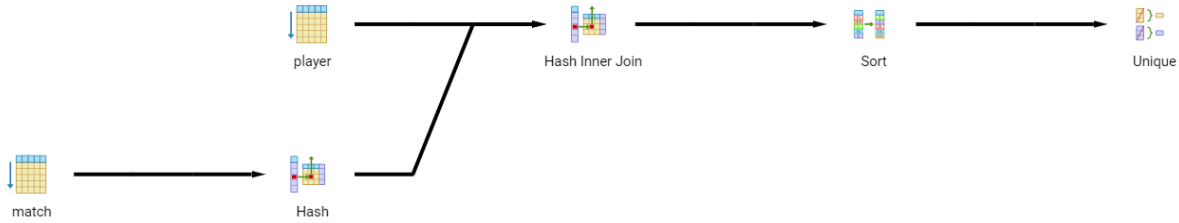
Praktikum 5 - Query Explains (vor Optimierung)

Query 1: playersPlaysInCertainTime

```
SELECT distinct p.name

   FROM myschema.Player p, myschema.Match m
   where p.id = m.playinggamer_id
            and m.matchstart between '2019-06-01' and '2019-06-20'
```



Explain – Erklärung:

1. Baut eine Hash-Tabelle* für Match table
2. Nimmt jeden Datensatz von player und vergleicht sie mit der Hash-Tabelle
3. Durch inner join, bleiben alle Datensätze, die die Bedingungen der Tabelle Player und Match erfüllen, die restlichen Datensätze werden gelöscht
4. Sortierung absteigend nach name
5. Mit ‚distinct' werden doppelte Datensätze werden entfernt

*Erklärung letzte Seite

Query 2: dataOfPlayer

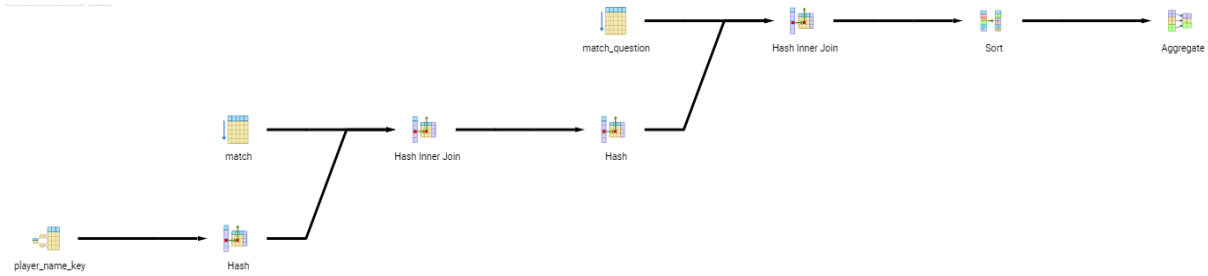
```
„SELECT m.id, m.matchStart, m.matchEnd, \n"
        + "m.numberOfCorrectAnswers, size(m.selectedQuestion) as questions \n"
        + "from Player p, Match m \n"
        + "where p.name = '" + player.getName() + "'\n"
        + "and p.id = m.playingGamer.id\n"
        + "group by m.id"
```

Query Editor | Query History

```
1  select m.id, m.matchstart, m.matchend, count(*) as questions, m.numberofcorrectanswers
2  from myschema.Player p, myschema.Match m
3  inner join myschema.match_question m_q on m_q.match_id = m.id
4  where p.name = 'Beti'
5  and p.id = m.playinggamer_id
6  group by m.id
```

Data Output | Explain | Messages | Notifications

	id integer	matchstart date	matchend date	questions bigint	numberofcorrectanswers integer
1	1	2019-06-16	2019-06-16	5	2
2	2	2019-06-16	2019-06-16	2	1



Explain – Erklärung:

1. Der Datensatz mit dem primarykey Beti wird ausgewählt
2. Baut eine Hash-Tabelle für Player table mit dem primarykey Beti
3. Nimmt jeden Datensatz von match und vergleicht sie mit der Hash-Tabelle
4. Durch inner join, bleiben alle Datensätze, die die Bedingungen der Tabelle Player und Match erfüllen, die restlichen Datensätze werden gelöscht
5. Baut eine Hash-Tabelle für die erzeugte Tabelle
6. Nimmt jeden Datensatz von match_question und vergleicht sie mit der Hash-Tabelle
7. Erfüllen manche Datensätze nicht die Bedingungen beider Tabellen, so werden diese Datensätze gelöscht
8. Sortierung absteigend nach match.id
9. Die Datensätze für jedes Match werden aggregiert(zu Gruppen zusammengefasst – für count())

Query 3: allPlayersWithGameNumbers

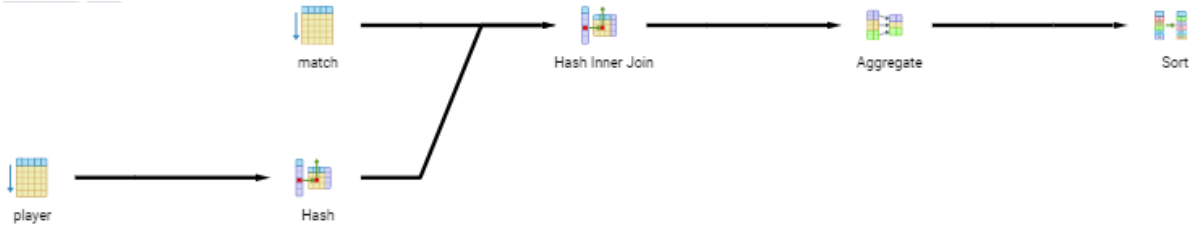
```
select p.name, count(*) as matchnumber

from myschema.Player p
inner join myschema.Match m on p.id = m.playingGamer_id
group by p.name
order by count(*) DESC
```

Query Editor | Query History

```
1  select p.name, count(*) as matchnumber
2  from myschema.Player p
3  inner join myschema.Match m on p.id = m.playingGamer_id
4  group by p.name
5  order by count(*) DESC
```

Data Output | Explain | Messages | Notifications

	name character varying (255)	matchnumber bigint
1	Beti	7
2	Fredde	1



Explain – Erklärung:

1. Baut eine Hash-Tabelle für Player table
2. Nimmt jeden Datensatz von Match und vergleicht sie mit der Hash-Tabelle
3. Durch inner join, bleiben alle Datensätze, die die Bedingungen der Tabelle Player und Match erfüllen, die restlichen Datensätze werden gelöscht
4. Die Datensätze für jeden player werden aggregiert(zu Gruppen zusammengefasst – für count())
5. Sortierung absteigend nach count-Wert

Query 4: categoryPopularity

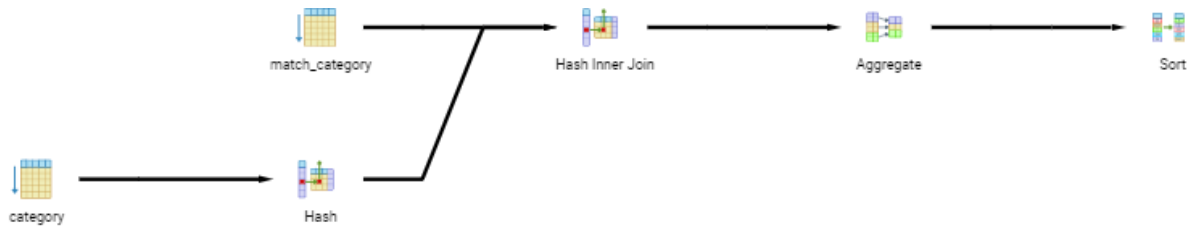
```
SELECT c.name, count(m_c.category_id) as callNumber

FROM myschema.category c
inner join myschema.match_category m_c on c.id = m_c.category_id
GROUP BY c.name
order by count(m_c.category_id) DESC
```

Query Editor | Query History

```
1  SELECT c.name, count(m_c.category_id) as callNumber
2  FROM myschema.category c
3  inner join myschema.match_category m_c on c.id = m_c.category_id
4  GROUP BY c.name
5  order by count(m_c.category_id) DESC
```

Data Output | Explain | Messages | Notifications

	name character varying (255)	callnumber bigint
1	TV	4
2	Tiere	3
3	Klassik	2
4	Rock	2
5	Physik	1
6	Liebe & Sex	1
7	Pop	1
8	Schlager	1
9	Reisen	1



Explain – Erklärung:

1. Baut eine Hash-Tabelle für Player category
2. Nimmt jeden Datensatz von Match_category und vergleicht sie mit der Hash-Tabelle
3. Durch inner join, bleiben alle Datensätze, die die Bedingungen der Tabelle Match_category und Category erfüllen, die restlichen Datensätze werden gelöscht
4. Die Datensätze für jeden category werden aggregiert(zu Gruppen zusammengefasst – für count()).
5. Sortierung absteigend nach count-Wert

Hashtabelle:

- Verwendet das mathematische Hashverfahren
- Ist eine Algorithmus zum Suchen von Datenobjekten in großen Datenmengen

- Hashfunktion berechnet die Position von einem Objekt in einer Tabelle
    - -> Zum berechnen des Hashwertes wird ein Schlüssel verwendet
    - -> Hashwert wird als Index in der Hashtabelle benutzt
    - -> Dadurch fällt das Durchsuchen vieler Datenobjekte weg
    - -> Objekt was man sucht ist schneller zu finden

- Hashwert legt fest, an welcher Stelle das Datenobjekt in der Tabelle gespeichert wird
    - -> Die Stelle wo das Datenobjekt liegt wird auch Bucket genannt
    - -> Im Idealfall bekommt jedes Objekt einen eigenen Bucket

Zugriff?

- Bei der Suche wird ein Hashwert aus dem Suchschlüssel gebildet
- Aus dem Hashwert wird dann bestimmt um welchen Bucket es sich handelt

Optimierung durch Datenbank Index

- Eine Datenbankstruktur mit deren Hilfe die Abfrageoptimierung gesteigert werden kann
- Mit einer Indextabelle werden die Daten sortiert auf dem Datenträger abgelegt

- Index selbst stellt einen Zeiger dar
    - -> Zeigt auf einen weiteren Index
    - -> oder zeigt auf einen Datensatz
    - => Dadurch erfolgt Trennung von Daten- und Index-Strukturen

Ohne Indizes?

- Ohne Indizes auf einer Tabelle müsste die Datenbank alle Datensätze sequenziell suchen
    - -> Beansprucht viel Zeit

Wann Indizes?

- Für Datenbanken die große Datenmengen speichern und sehr häufig abgefragt werden