# Vorbereitung Praktikum 5

## Aufgabe 2: Weitere Vorbereitung zu Hause

### Query 1:

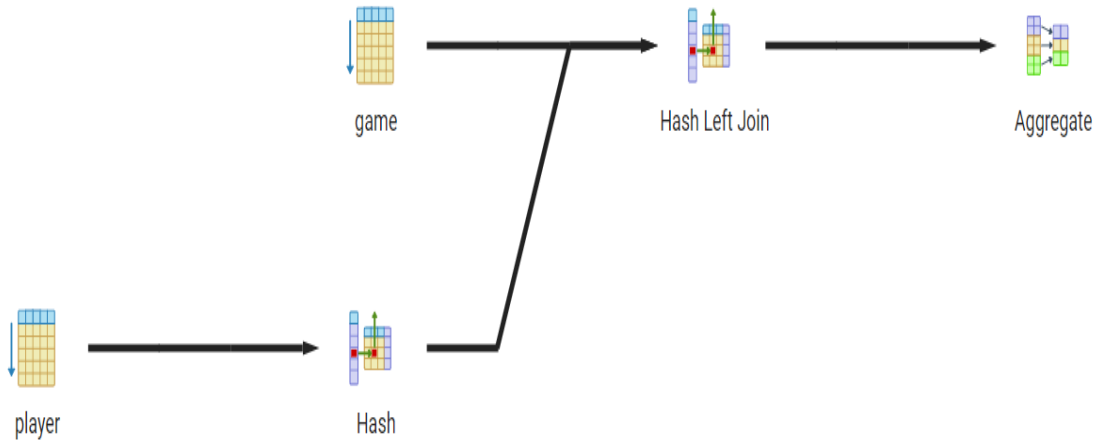
#### JPQL:

```
"select distinct g.player from Game g where g.startTime >= '2021-07-
07 11:07:55.516367+02' and g.endTime <= '2021-07-10
11:07:58.516367+02'";
```

### SQL:

SELECT DISTINCT t0.NAME

FROM public.Game t1 LEFT OUTER JOIN public.Player t0 ON (t0.NAME = t1.PLAYER_NAME)
WHERE ((t1.STARTTIME >= ?) AND (t1.ENDTIME <= ?));




Er lädt aus dem Hintergrund Speicher die komplette Tabelle Game und Player in Puffer rein.

Nachdem die Komplette Player Tabelle in Puffer rein lädt, (weil die Player Tabelle kleiner als Game ist, “Jeder Player hat 100 Games.”), dann er zeugt ein Hash für die Player Tabelle.

Dann macht full scan von Game, und holt sich denn über den Hash von dem Player (Fremdschlüssel Beziehung), den entsprechende Player_name, dann macht damit ein Hash Left Join.

Am Ende macht ein Aggregate über den Player_name.

## Query 2:

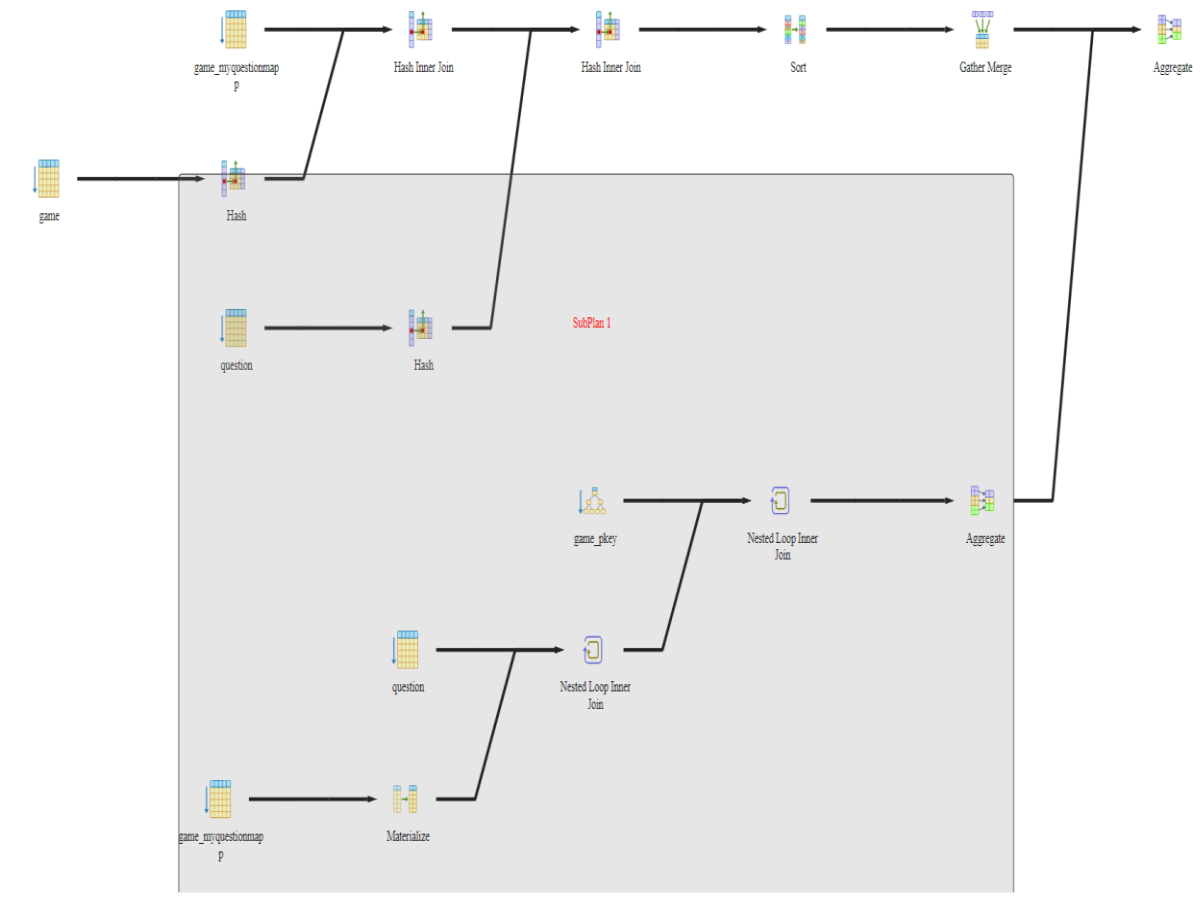
### JPQL:

```
select g.id, g.startTime, g.endTime, (select count(gp) from
g.myquestionmap gp where  KEY(gp).correctAnswerIndex=VALUE(gp)),
size(g.myquestionmap)

from Game g where g.player.name='player 1'
```

### SQL:

```
SELECT t0.ID, t0.STARTTIME, t0.ENDTIME, (SELECT COUNT(t5.antwort)
FROM Game_MYQUESTIONMAP t5, public.Question t4, public.Game t6 WHERE
(((t4.CORRECTANSWERINDEX = t5.antwort) AND (t0.ID = t6.ID)) AND
((t5.Game_ID = t6.ID) AND (t4.ID = t5.myquestionmap_KEY)))),
COUNT(t2.antwort) FROM public.Game t0, Game_MYQUESTIONMAP t2,
public.Question t1 WHERE ((t0.PLAYER_NAME = 'player 1') AND
((t2.Game_ID = t0.ID) AND (t1.ID = t2.myquestionmap_KEY))) GROUP BY
t0.ID, t0.ENDTIME, t0.STARTTIME, t0.PLAYER_NAME;
```



Er lädt aus dem Hintergrund Speicher die kompletten Tabellen Game und Game_MYQUESTIONMAP und Question in Puffer rein.

Nachdem die Komplette Game Tabelle in Puffer rein lädt, (weil die Game Tabelle kleiner als Game_MYQUESTIONMAP ist), dann er zeugt ein Hash für die Game Tabelle.

Dann macht full scan von Game_MYQUESTIONMAP, und holt sich denn über den Hash von dem Game die Fremdschlüssel Beziehung bzw. Game_MYQUESTIONMAP.Game_ID = Game.ID, den entsprechende Game_ID, dann macht damit ein Hash Inner Join.

Da wir 2 Selects in der Query haben:

Die Tabelle Question wird komplett in Puffer reingeladen, und dann ein Hash erzeugt. Dann macht full scan von Game_MYQUESTIONMAP, und holt sich denn über den Hash von der Question die Fremdschlüssel Beziehung bzw. Game_MYQUESTIONMAP. myquestionmap_KEY = Question.ID, und dann macht damit ein Hash Inner Join, und dann wird nach Game.ID sortiert.

Da die Gather Merge sortierte Daten verbraucht und diese Sortierreihenfolge bewahrt, wird before nach dem Game_id sortiert, und dann wird es mit Merge gather.

Es würde ein zweites  Game_MYQUESTIONMAP t5 gebraucht, so wird das zweite Game_MYQUESTIONMAP t5 Tabelle in Puffer rein lädt, und dann kommt ein Materialisierungsknoten was das bedeutet dass die Ausgabe dessen, was sich im Baum unter ihm befindet und hier haben wir Game_MYQUESTIONMAP t5 (das kann ein Scan oder ein vollständiger Satz von Joins oder etwas Ähnliches sein), in den Speicher ausgelagert wird, bevor der obere Knoten ausgeführt wird.

Dann wird die Question t4 in Puffer reingeladen, und dann mit join gefilter mit ((t5.antwort= t4.CORRECTANSWERINDEX) and (t4.ID = t5.myquestionmap_KEY)).

Es wird ein Index only Scan (Sie ermöglichen es, dass bestimmte Arten von Abfragen nur durch das Abrufen von Daten aus Indizes und nicht aus Tabellen erfüllt werden können.) erstellt, und dann Nested Loop ausgeführt und aggregiert und wieder abschließend nach Game.ID aggregiert

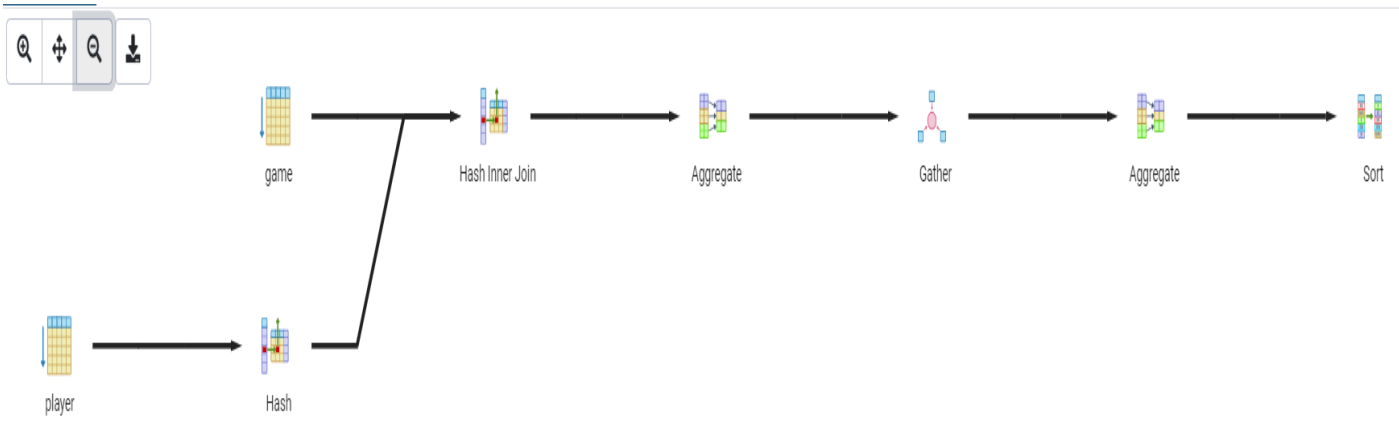
## Query 3:

JPQL:

```
select g.player.name, count(g.id) as AnzahlSpiele from Game g group
by g.player.name order by AnzahlSpiele
```

SQL:

SELECT t0.NAME, COUNT(t1.ID) FROM public.Player t0, public.Game t1 WHERE (t0.NAME = t1.PLAYER_NAME) GROUP BY t0.NAME ORDER BY COUNT(t1.ID);




Er lädt aus dem Hintergrund Speicher die komplette Tabelle Game und Player in Puffer rein.

Nachdem die Komplette Player Tabelle in Puffer rein lädt, (weil die Player Tabelle kleiner als Game ist, “Jeder Player hat 100 Games.”), dann er zeugt ein Hash für die Player Tabelle.

Dann macht full scan von Game, und holt sich denn über den Hash von dem Player (Fremdschlüssel Beziehung), den entsprechende Player_name, dann macht damit ein Hash Inner Join.

Danach macht ein Aggregate über den Player_name.

Dann wird es gather, und dann macht ein Aggregate über den Player_name, und am Ende nach COUNT(Game.ID) sortiert.

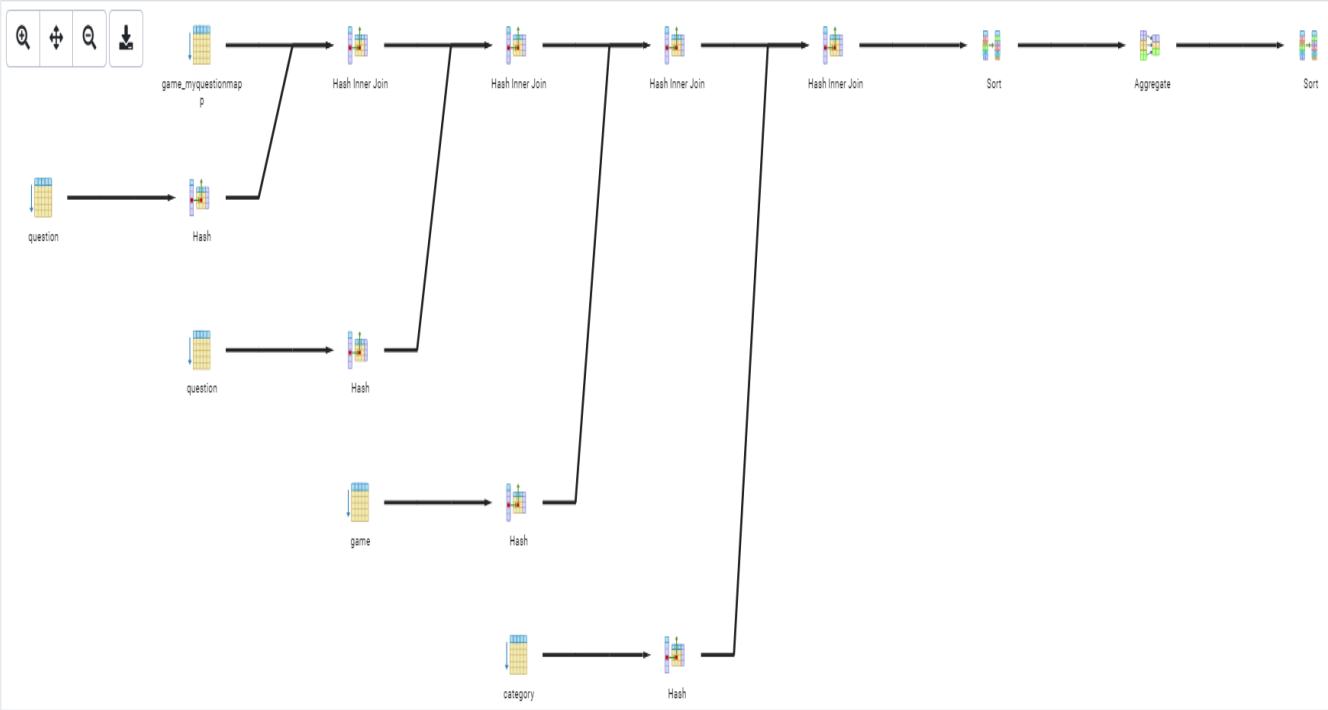
## Query 4:

JPQL:

```
select c.id, count (DISTINCT g.id) from Game g,
g.myquestionmap gm, Category c, c.myQuestions cq
    where value(cq).id=key(gm).id group by c.id order by
count (DISTINCT g.id)
```

SQL:

```
SELECT t0.ID, COUNT(DISTINCT(t1.ID)) FROM public.Question
t4, Game_MYQUESTIONMAP t3, public.Question t2, public.Game
t1, public.Category t0 WHERE ((t4.ID = t2.ID) AND
(((t3.Game_ID = t1.ID) AND (t2.ID = t3.myquestionmap_KEY))
AND (t4.Category_id = t0.ID))) GROUP BY t0.ID ORDER BY
COUNT(DISTINCT(t1.ID));
```




Er lädt aus dem Hintergrund Speicher die kompletten Tabellen Game, Question, Category und Game_Myquestionmap in Puffer rein.

Nachdem die Komplette Question Tabelle in Puffer rein lädt, (weil die Question Tabelle kleiner als Game_Myquestionmap ist), dann er zeugt ein Hash für die Question Tabelle.

Dann macht full scan von Game_Myquestionmap, und holt sich denn über den Hash von der Question (Fremdschlüssel Beziehung), den entsprechende myquestionmap_key bzw. (Question.ID = Game_Myquestionmap.myquestionmap_KEY), dann macht damit ein Hash Inner Join.

Dann wird die Game Tabelle in Puffer reingeladen, dann erzeugt der Optimizer ein Hash für die Game Tabelle, und macht ein full scan von Game_Myquestionmap, und holt sich denn über den Hash von der Game (die Fremdschlüssel Beziehung), den entsprechende game_id bzw. (Game_Myquestionmap.Game_ID =Game.ID), dann macht damit ein Hash Inner Join.

Dann wird die Category Tabelle in Puffer reingeladen, und der Optimizer erzeugt ein Hash für die Tabelle, und macht ein full scan von Question, und holt sich denn über den Hash von der Question die Fremdschlüssel Beziehung Question.Category_id = Category.ID, dann macht damit ein Hash Inner Join.

Dann wird nach dem Category.ID sortiert, und dann kommt den Aggregate über das Category.ID, und am Ende nach COUNT(DISTINCT(t1.ID) sortiert.