Vorbereitung Praktikum 5

JPQL Queries

Query 1

Ausgabe aller Spieler (Spielername), die in einem bestimmten Zeitraum gespielt hatten:

```
List<String> players = em.createQuery( "SELECT p.name FROM Player p INNER JOIN
Game g "
      + "ON p.name = g.player.name "
      + "WHERE g.start >= :start "
      + "AND g.end <= :end " + "GROUP BY p.name" ).setParameter("start",
start).setParameter("end", end).getResultList();
```

Query 2

Ausgabe zu einem bestimmten Spieler: Alle Spiele (Id, Datum), sowie die Anzahl der korrekten Antworten pro Spiel mit Angabe der Gesamtzahl der Fragen pro Spiel:

```
List results = em.createQuery(
            "SELECT p.name, g.id, g.start, g.end, COUNT(KEY(q)) AS questions, "
                  + "SUM(CASE WHEN VALUE(q) = TRUE THEN 1 ELSE 0 END) AS
correctAnswers "
                  + "FROM Game g JOIN g.player p JOIN g.askedQuestion q "
                  + "WHERE p.name = :name "
                  + "GROUP BY p.name, g.id")
      .setParameter("name", name)
      .getResultList();
```

Query 3

Ausgabe aller Spieler mit Anzahl der gespielten Spiele, nach Anzahl absteigend geordnet:

```
String query3 = (" select p.name, count(g) as playedGames from Player p, Game g where
g.player = p group by p.name order by playedGames desc");
List pl = em.createQuery(query3).getResultList();
```

Query 4

Ausgabe der am meisten gefragten Kategorie:

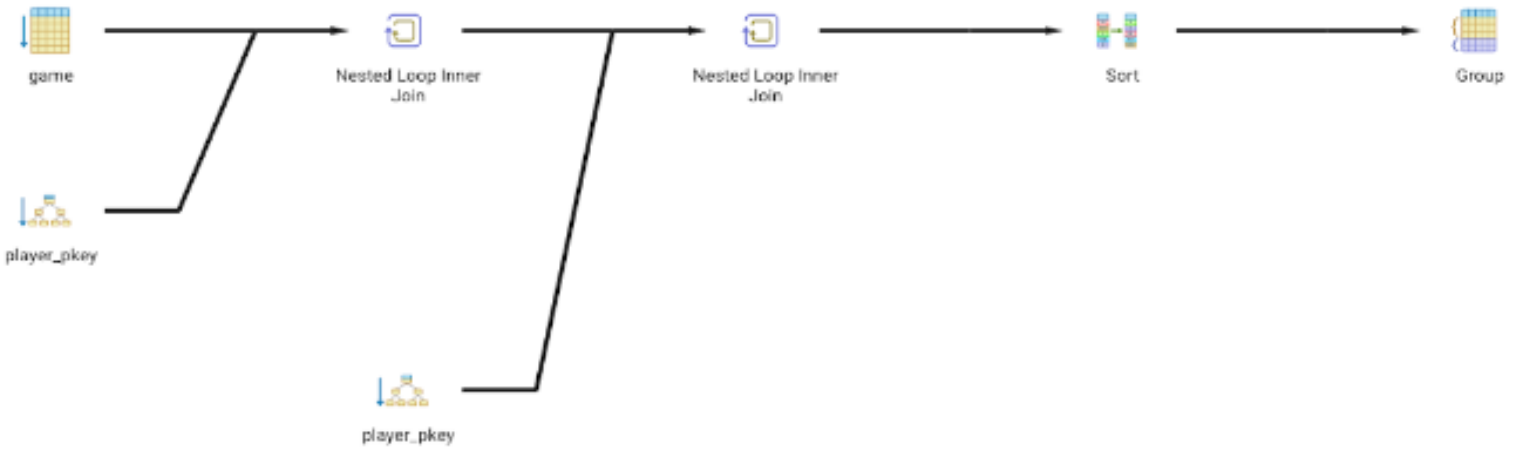
```
List results = em.createQuery(
            "SELECT c.name, COUNT(q.quest) AS times_chosen "
                  + "FROM Game g JOIN g.askedQuestion qog JOIN KEY(qog) q JOIN q.ctgry
                  c "
                  + "GROUP BY c.name "
                  + "ORDER BY times_chosen DESC")
      .getResultList();
```

SQL-Anweisungen von EclipseLink zur DB:

Query 1:

[EL Fine]: sql: Connection(2131597042)--SELECT t0.NAME FROM db2p.Player t0, db2p.Player t2, db2p.Game t1 WHERE (((t1.startTime >= ?) AND (t1.endTime <= ?)) AND ((t2.NAME = t1.PLAYER_NAME) AND (t0.NAME = t2.NAME))) GROUP BY t0.NAME
bind => [2022-07-13 13:00:00.0, 2022-07-13 14:30:00.0]

SQL Befehl in PgAdmin: SELECT t0.NAME FROM db2p.Player t0, db2p.Player t2, db2p.Game t1 WHERE (((t1.startTime >= '2022-07-13 13:00:00.0') AND (t1.endTime <= '2022-07-13 14:30:00.0')) AND ((t2.NAME = t1.PLAYER_NAME) AND (t0.NAME = t2.NAME))) GROUP BY t0.NAME

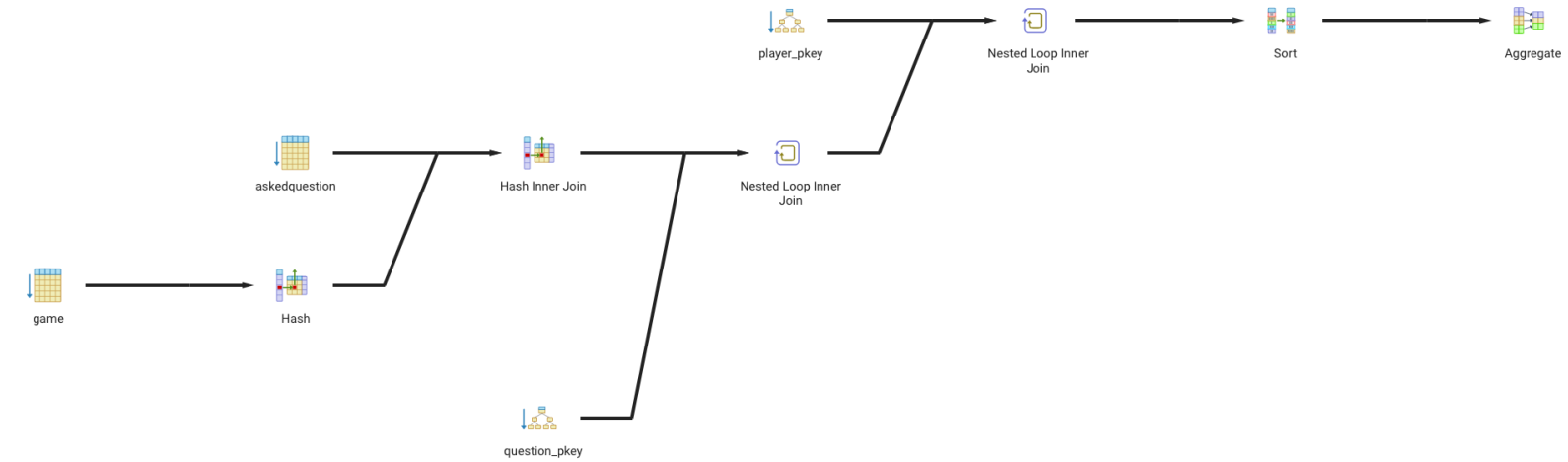


Es wird ein Full Table Scan von Game gemacht da durch alle Spiele iteriert werden muss. Anschließend wird ein Nested Loop Inner Join gemacht um für jede Zeile die entsprechenden Einträge in der Player Tabelle zu suchen.. Dies geschieht zweimal da wir ein Start- und Enddatum haben. Anschließend werden die Matches sortiert und gruppiert.

Query 2:

[EL Fine]: sql: Connection(2131597042)--SELECT t0.NAME, t1.ID, t1.startTime, t1.endTime, COUNT(t2.ID), SUM(CASE WHEN (t3.ASKEDQUESTION = ?) THEN ? ELSE ? END) FROM db2p.AskedQuestion t3, db2p.Question t2, db2p.Game t1, db2p.Player t0 WHERE ((t1.PLAYER_NAME = ?) AND ((t0.NAME = t1.PLAYER_NAME) AND ((t3.Game_ID = t1.ID) AND (t2.ID = t3.askedQuestion_KEY)))) GROUP BY t0.NAME, t1.ID bind => [true, 1, 0, Can]

SQL Befehl in PgAdmin: SELECT t0.NAME, t1.ID, t1.startTime, t1.endTime, COUNT(t2.ID), SUM(CASE WHEN (t3.ASKEDQUESTION = true) THEN 1 ELSE 0 END) FROM db2p.AskedQuestion t3, db2p.Question t2, db2p.Game t1, db2p.Player t0 WHERE ((t1.PLAYER_NAME = 'Can') AND ((t0.NAME = t1.PLAYER_NAME) AND ((t3.Game_ID = t1.ID) AND (t2.ID = t3.askedQuestion_KEY)))) GROUP BY t0.NAME, t1.ID



Da wir alle gestellten Fragen und gespielten Spiele brauchen werden auf Game und Askedquestion jeweils ein Full Table Scan gemacht. Für Game wird ein Hash Wert berechnet und anschließend in einem Hash Inner Join die Hashwerte vergliche und die entsprechenden Datensätze mit den Askedquestion Einträgen gebündelt. Anschließend wird mit der Question ID ein innerer Verbund gemacht. Danach passiert das Gleiche mit dem Player_PK. Dies wird gemacht damit nur die gespielten Fragen übrig bleiben, die vom Spieler gespielt wurden. Diese werden dann sortiert und zuletzt wird die Aggregatfunktion behandelt.

Query 3:

[EL Fine]: sql: Connection(2131597042)--SELECT t0.NAME, COUNT(t1.ID) FROM db2p.Player t0, db2p.Game t1 WHERE (t1.PLAYER_NAME = t0.NAME) GROUP BY t0.NAME ORDER BY COUNT(t1.ID) DESC

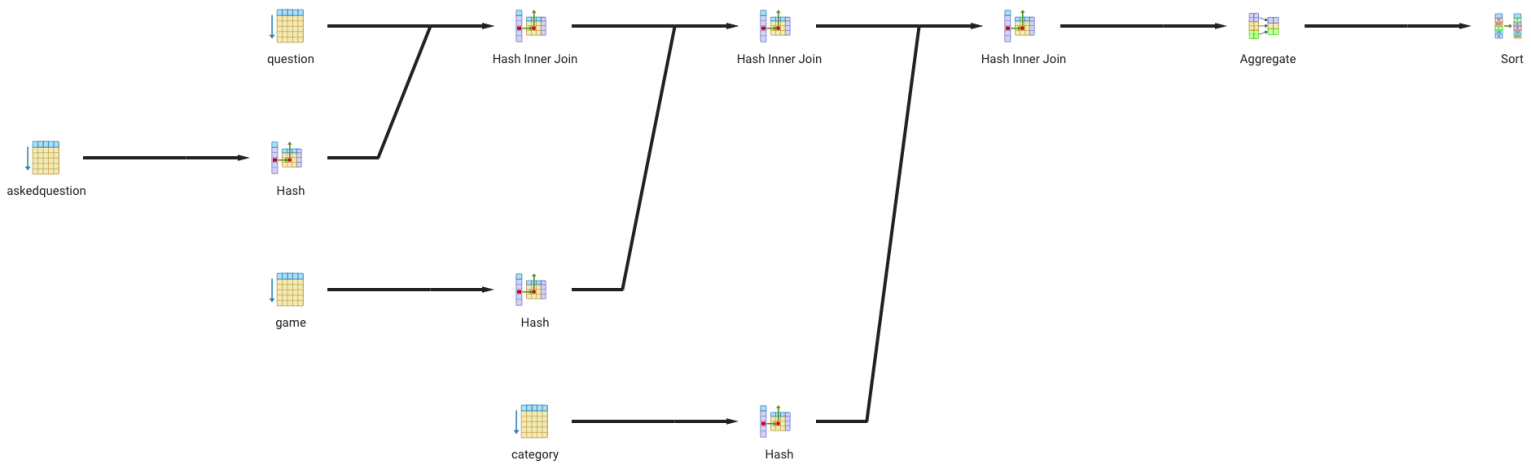
SQL Befehl in PgAdmin: SELECT t0.NAME, COUNT(t1.ID) FROM db2p.Player t0, db2p.Game t1 WHERE (t1.PLAYER_NAME = t0.NAME) GROUP BY t0.NAME ORDER BY COUNT(t1.ID) DESC

Zuerst wird die komplette Player-Tabelle gescannt, dabei wird zu jedem Index ein Hashwert berechnet. Die Game-Tabelle wird auch gescannt und dann werden mit dem Hash Inner Join die Hashwerte verglichen und die entsprechenden Datensätze verbündet. Aufgrund der Verwendung der Aggregatfunktion „Count“ wird die Ergebnistabelle entsprechend ausgewertet. Anschließend werden die Einträge sortiert.

Query 4:

[EL Fine]: sql: Connection(2131597042)--SELECT t0.NAME, COUNT(t1.QUEST) FROM db2p.Game t3, db2p.AskedQuestion t2, db2p.Question t1, db2p.Category t0 WHERE (((t2.Game_ID = t3.ID) AND (t1.ID = t2.askedQuestion_KEY)) AND (t0.CATEGORYID = t1.CTGRY_CATEGORYID)) GROUP BY t0.NAME ORDER BY COUNT(t1.QUEST) DESC

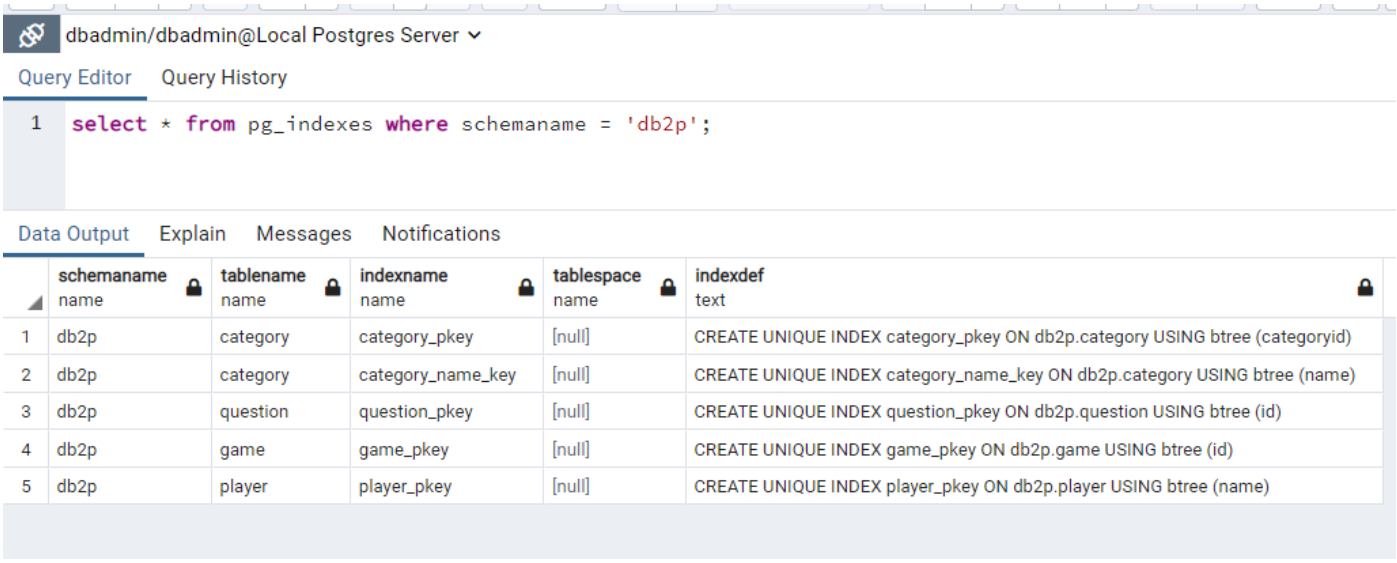
SQL Befehl in PgAdmin: SELECT t0.NAME, COUNT(t1.QUEST) FROM db2p.Game t3, db2p.AskedQuestion t2, db2p.Question t1, db2p.Category t0 WHERE (((t2.Game_ID = t3.ID) AND (t1.ID = t2.askedQuestion_KEY)) AND (t0.CATEGORYID = t1.CTGRY_CATEGORYID)) GROUP BY t0.NAME ORDER BY COUNT(t1.QUEST) DESC



Zuerst wird die komplette Askedquestion-Tabelle gescannt, dabei wird zu jedem Index ein Hashwert berechnet. Die question-Tabelle wird auch gescannt und dann werden mit dem Hash Inner Join die Hashwerte verglichen und die entsprechenden Datensätze verbündet. In dem anschließenden Hash Inner Join werden für die neu erstellte Tabelle die entsprechenden Einträge in der Game-Tabelle, nachdem für diese die Hash Werte berechnet wurden, gebündelt. Anschließend werden die Hashwerte der Category Indizes mit den Hashwerten der Ergebnistabelle verglichen und dann erfolgt ein Hash Inner join. In der SQL-Anweisung kommt wieder eine Aggregatfunktion vor, deshalb der Node Type Aggregate und anschließend das Sort.

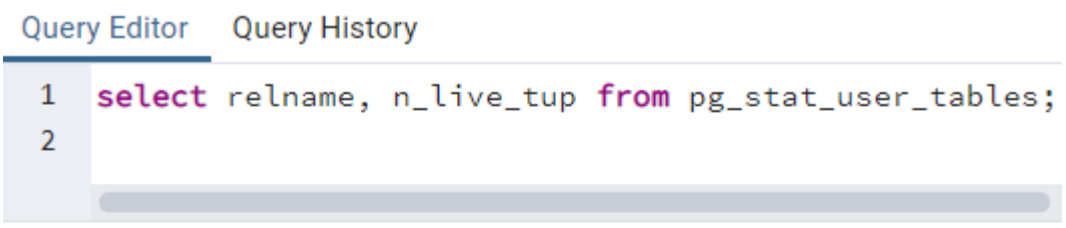
Praktikum 5 Aufgabe 3

select * from pg_indexes where schemaname = 'name ihres schemas';



	schemaname name	tablename name	indexname name	tablespace name	indexdef text
1	db2p	category	category_pkey	[null]	CREATE UNIQUE INDEX category_pkey ON db2p.category USING btree (categoryid)
2	db2p	category	category_name_key	[null]	CREATE UNIQUE INDEX category_name_key ON db2p.category USING btree (name)
3	db2p	question	question_pkey	[null]	CREATE UNIQUE INDEX question_pkey ON db2p.question USING btree (id)
4	db2p	game	game_pkey	[null]	CREATE UNIQUE INDEX game_pkey ON db2p.game USING btree (id)
5	db2p	player	player_pkey	[null]	CREATE UNIQUE INDEX player_pkey ON db2p.player USING btree (name)

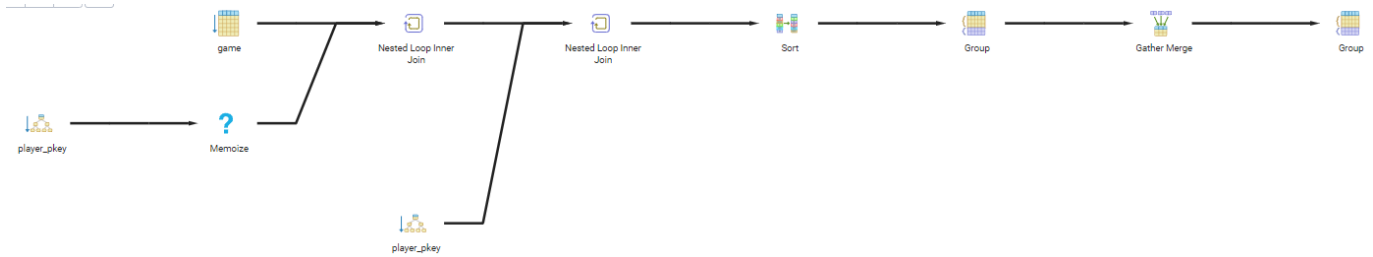
select relname, n_live_tup from pg_stat_user_tables;



	relname name	n_live_tup bigint
1	askedquestion	6
2	game	2
3	category	51
4	player	1
5	question	200
6	answer	800

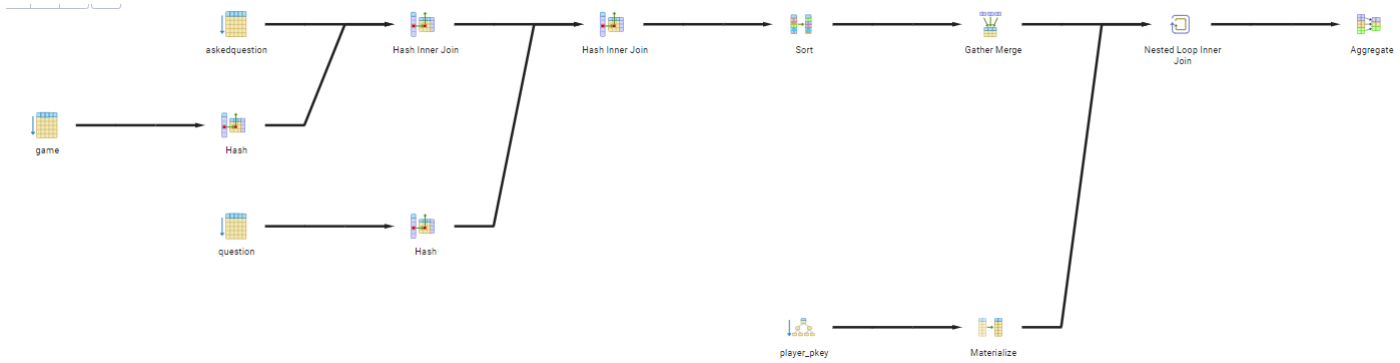
Nach Massendaten Generierung

Query 1



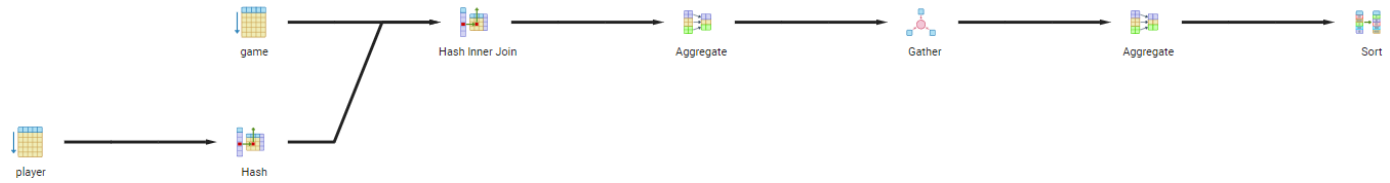
Cost davor 9.85, nach Massendaten 16200.59

Query 2



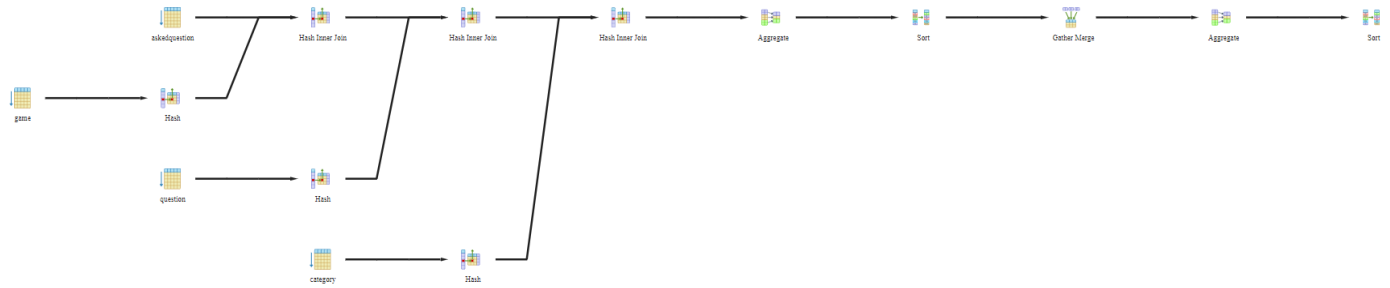
Cost davor 15.96, nach Massendaten 208378.34

Query 3



Cost 13.44, nach Massendaten 19946.95

Query 4



Cost 12.34, nach Massendaten 354156.17

Index auf Query 1

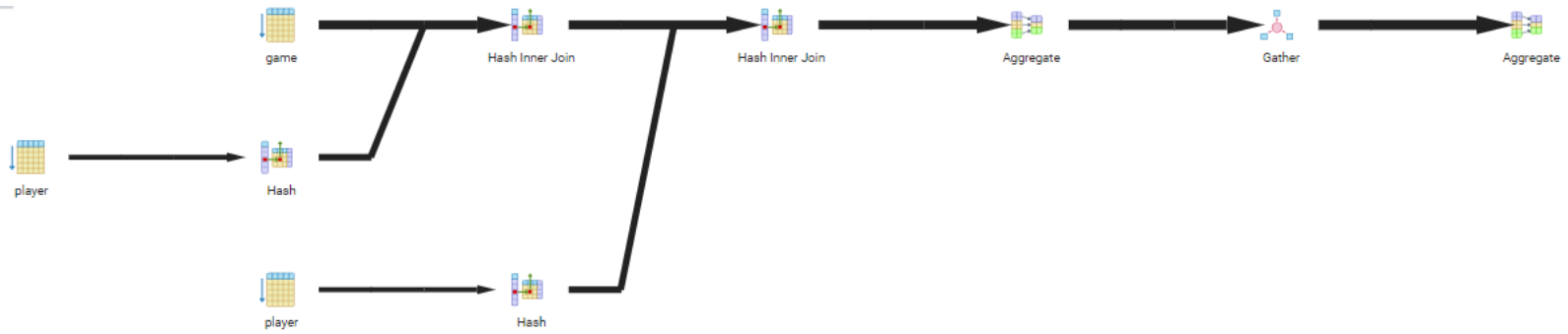
Query Editor | Query History

```
1  create index "GameStart" on db2p.game (starttime);
2  create index "GameEnd" on db2p.game (endtime);
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 1 secs 149 msec.
```



Man sieht das auf player jetzt ein Full-Table Scan durchgeführt wird. Statt Nested Loop Inner Joins haben wir Hash Inner Joins an deren Stelle. Das Sort fällt weg und stattdessen haben wir ein Aggregate, ein Gather(sammeln) und dann nochmal ein Aggregate (Spielzeiten der Anfrage auswählen).

Cost vor Index 16200.59, Cost Nach Index 7467.87.

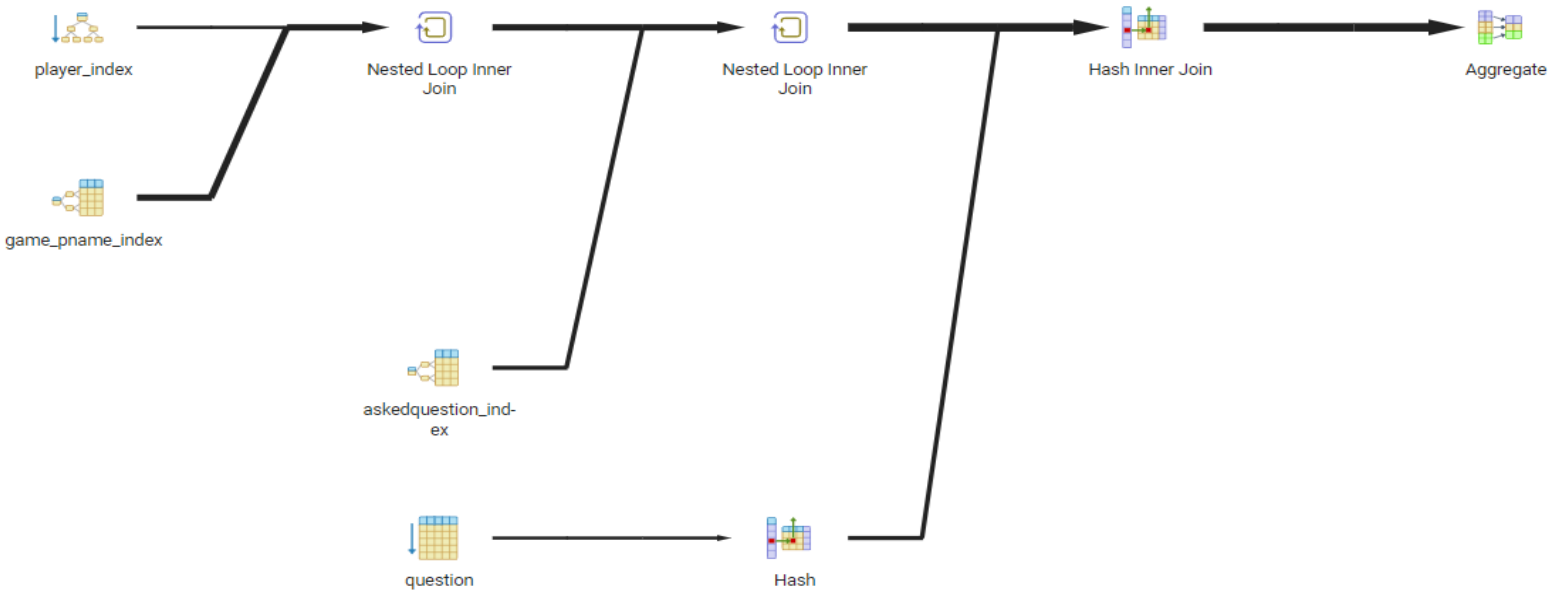
Index auf Query 2

```
1  create index "askedquestion_index" on db2p.askedquestion (game_id);
2  create index "game_index" on db2p.game (id);
3  create index "question_index" on db2p.question (id);
4
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 9 secs 23 msec.
```



Auf die Tabelle Player, Game und Askedquestion werden die erstellten Indexe verwendet. Beim Join zwischen Game und Askedquestion existiert kein Hash Inner Join mehr sondern ein Nested Loop Inner Join. Das Sort, Gather Merge und ein weiteres Nested Loop Inner Join fallen weg.

Cost vor Index 208378.34, Cost Nach Index 115292.53.

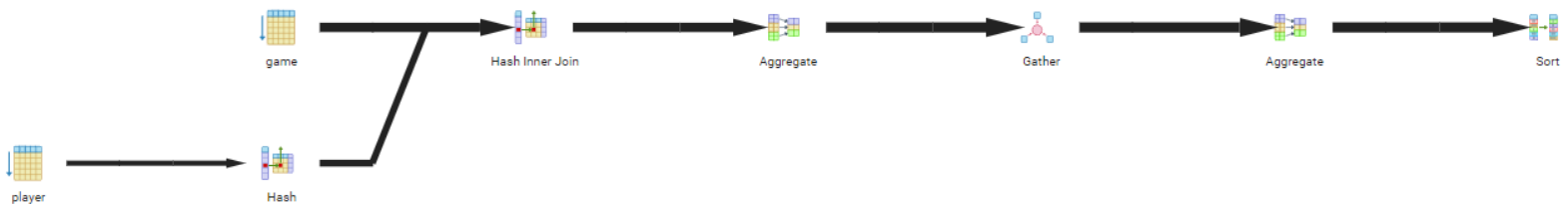
Index auf Query 3

```
1  create index "game_pname_index" on db2p.game (player_name);
2  create index "player_index" on db2p.player (name);
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 927 msec.
```

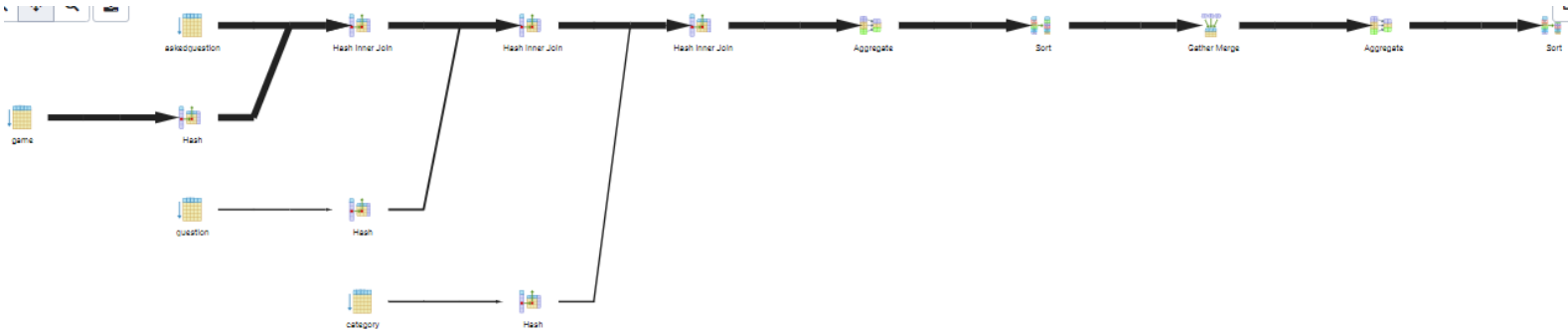


Der Index wurde nicht benutzt. Das bedeutet, dass ein Full Table Scan schneller ist als die Indexe, da in dieser Abfrage durch die kompletten Tabellen iteriert werden muss.. Das Grafische Explain ist gleich wie davor.

Cost vor Index 19946.95, Cost Nach Index 19946.95.

Index auf Query 4

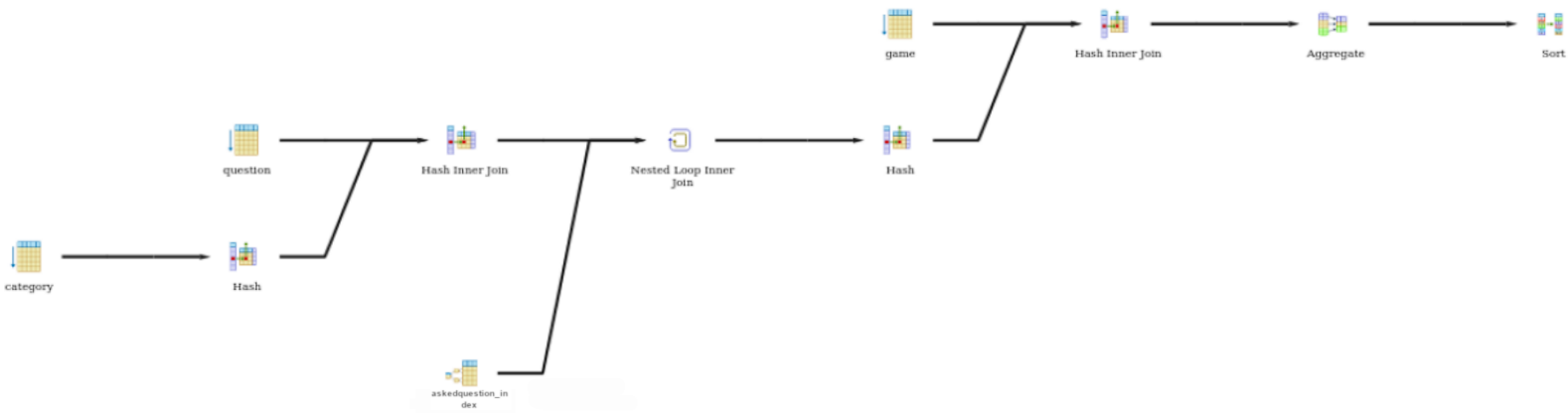
```
1  create index "category_id_index" on db2p.category (categoryid);
```



Die Indexe für Game, Askedquestion, Question wurden bereits vorher erstellt doch diese wurden aber hier nicht verwendet. Es ist schneller keine Indexe zu verwenden für diesen Query. Es ist das gleiche wie bei Query 3, es werden alle Informationen aus allen Tabellen benötigt und daher ist ein Full Table Scan notwendig und die Indexe werden nicht verwendet.

Mit dem folgendem Befehl konnten wir eine Optimierung erzielen.

SELECT t0.NAME, COUNT(t1.QUEST) FROM db2p.Game t3, db2p.AskedQuestion t2, db2p.Question t1, db2p.Category t0
WHERE (((t2.Game_ID = t3.ID)
AND (t1.ID = t2.askedQuestion_KEY))
AND (t0.CATEGORYID = t1.CTGRY_CATEGORYID))
AND t0.NAME = 'Gesellschaft'



GROUP BY t0.NAME ORDER BY COUNT(t1.QUEST) DESC

Cost vor Index 354156.17, Cost Nach Index 228854.4.