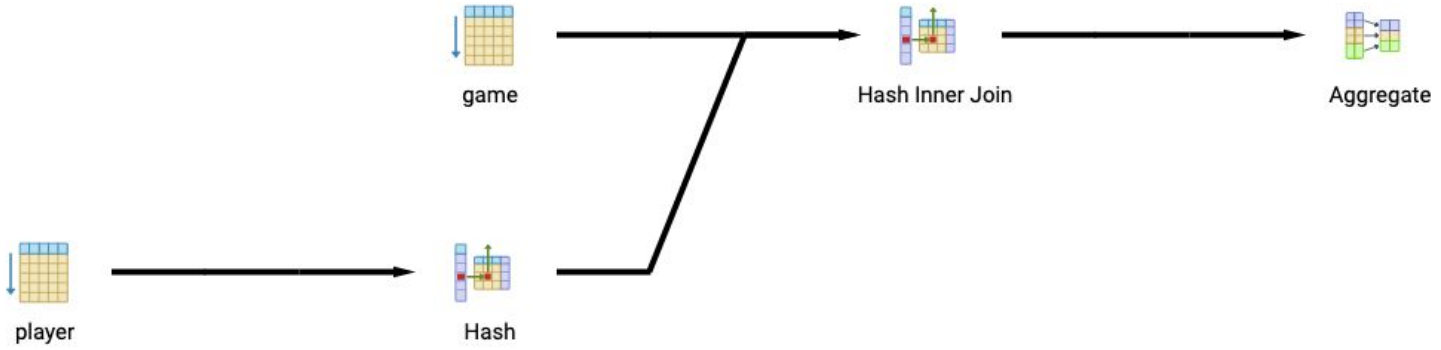
DB2 Praktikumsbericht 5

I. Ausgabe aller Spieler (Spielername), für einen bestimmten Zeitraum

- **Query:**
  Query query = em.createQuery(
  "SELECT g.player.name from Game as g where g.timeStampStart > :start and g.timeStampEnd < :end Group by g.player.name");

- **von EclipseLink erzeugte SQL- Anweisung:**
  SELECT t0.NAME FROM Player t0, Game t1 WHERE (((t1.TIMESTAMPSTART > ?) AND (t1.TIMESTAMPEND < ?)) AND (t0.ID = t1.PLAYER_ID)) GROUP BY t0.NAME
  bind => [2019-01-01 00:00:00.0, 2019-03-01 00:00:00.0]

- **im Query Tool von pg-admin eingegebene Abfrage:**
  Explain analyze SELECT t0.NAME FROM Player t0, Game t1 WHERE (((t1.TIMESTAMPSTART > '2019-01-01') AND (t1.TIMESTAMPEND < '2019-03-01')) AND (t0.ID = t1.PLAYER_ID)) GROUP BY t0.NAME;



Erläuterung zur Explain-Grafik

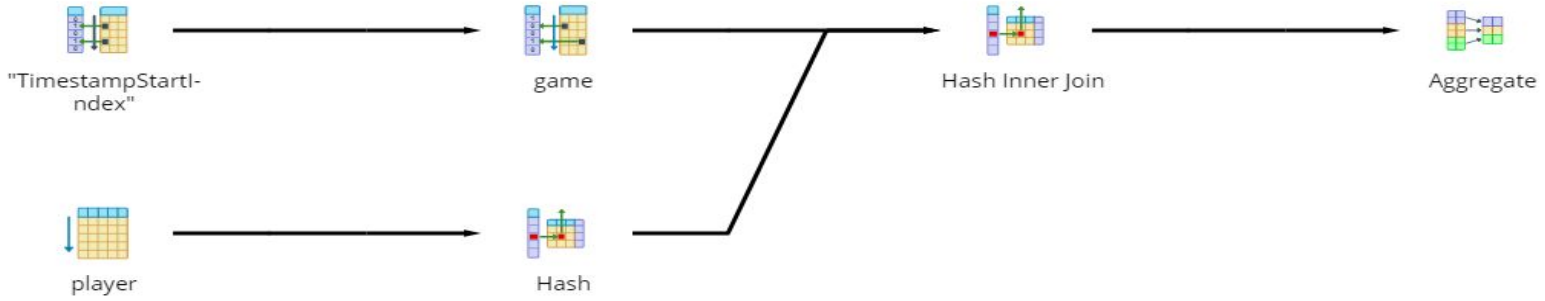
1. Für die Player-Table wird ein Hash-Index erstellt, wobei durch die Anwendung der Hash-Funktion auf die Player-ID die Position des Player-Records in der neu erstellten Hash-Tabelle bestimmt wird. Damit kann später sehr schnell nach den Spieler-Ids gesucht werden, die zu einem Spiel gehören.
2. Die Game Tabelle wird in einem sequenziellen Scan nach dem angegebenen Zeitraum gefiltert.
3. Inner join der Game Tabelle mit der Player-Hashtabelle nach dem Kriterium Player.Id=Game.player_id.
4. Es folgt eine Hash-Aggregation, um die join-Tabelle nach den Spielernamen zu gruppieren. Das bedeutet, dass player.name der Key für die Hash-Aggregation ist. Die Hash-Aggregation liefert für jeden Key, bzw. für jede Gruppe eine Zeile zurück.

Optimierung für Query 1

Maßnahme	vor Optimierung	nach Einführen von Indizes
Einführen von B-tree Index in der Game Tabelle auf die Spalte TimestampStart	Planning: 0.316 ms Execution: 109.025 ms	Planning: 1.410 ms Execution: 6.432. ms
Einführen von B-tree Index in der Game Tabelle auf die Spalte TimestampEnd	Planning: 1.410 ms Execution: 6.432. ms	Planning: 0.420 ms Execution: 5.003 ms

→ in der Explain Grafik auf der nächsten Seite ist zu erkennen, dass der Index auf TimeStamp-End nicht genutzt! Die Kostenverbesserung ist deswegen wahrscheinlich auf Messschwankungen zurückzuführen.

Explain nach Optimierung:

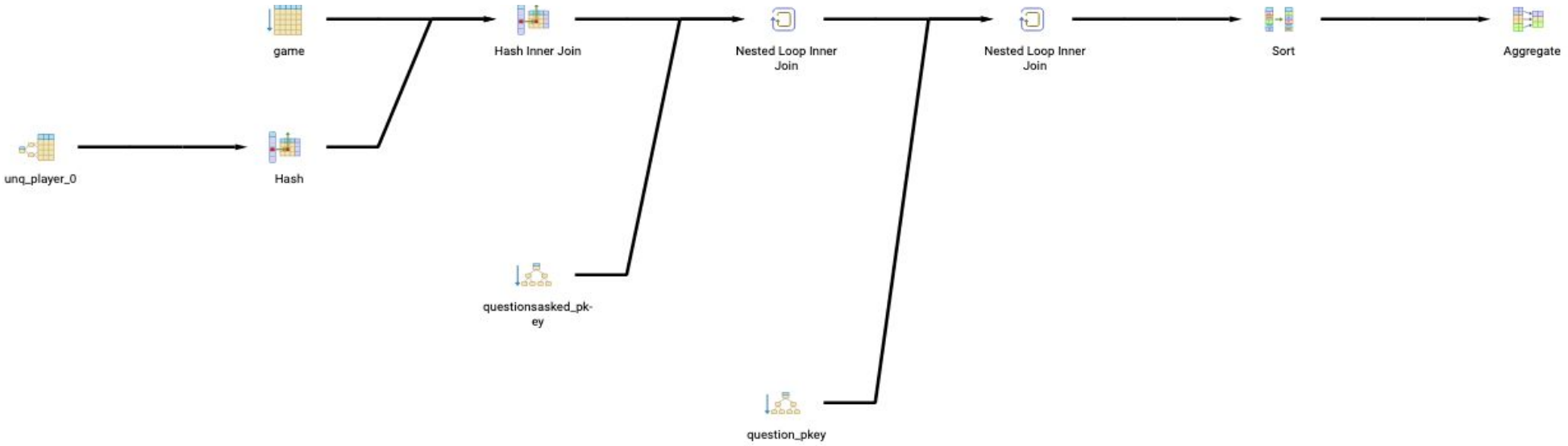


II. Ausgabe zu einem bestimmten Spieler: Alle Spiele (Id, Datum), die Anzahl der korrekten Antworten pro Spiel mit Angabe der Gesamtanzahl der Fragen pro Spiel

- **Query**
  Query query = em.createQuery(
  "SELECT g.id, g.timeStampStart, g.timeStampEnd, g.rightAnswers, count(g.questionsAsked) from Game as g where g.player.name = :name group by g.id order by g.id");

- **von EclipseLink erzeugte SQL- Anweisung:**
  SELECT t0.ID, t0.TIMESTAMPSTART, t0.TIMESTAMPEND, t0.RIGHTANSWERS, COUNT(t1.ID) FROM questionsAsked t3, Player t2, Question t1, Game t0 WHERE ((t2.NAME = ?) AND ((t2.ID = t0.PLAYER_ID) AND ((t3.Game_ID = t0.ID) AND (t1.ID = t3.questionsAsked_ID)))) GROUP BY t0.ID ORDER BY t0.ID bind => [player1]

- **im Query Tool von pg-admin eingegebene Abfrage:**
  SELECT t0.ID, t0.TIMESTAMPSTART, t0.TIMESTAMPEND, t0.RIGHTANSWERS, COUNT(t1.ID) FROM questionsAsked t3, Player t2, Question t1, Game t0 WHERE ((t2.NAME = 'player1') AND ((t2.ID = t0.PLAYER_ID) AND ((t3.Game_ID = t0.ID) AND (t1.ID = t3.questionsAsked_ID)))) GROUP BY t0.ID ORDER BY t0.ID;



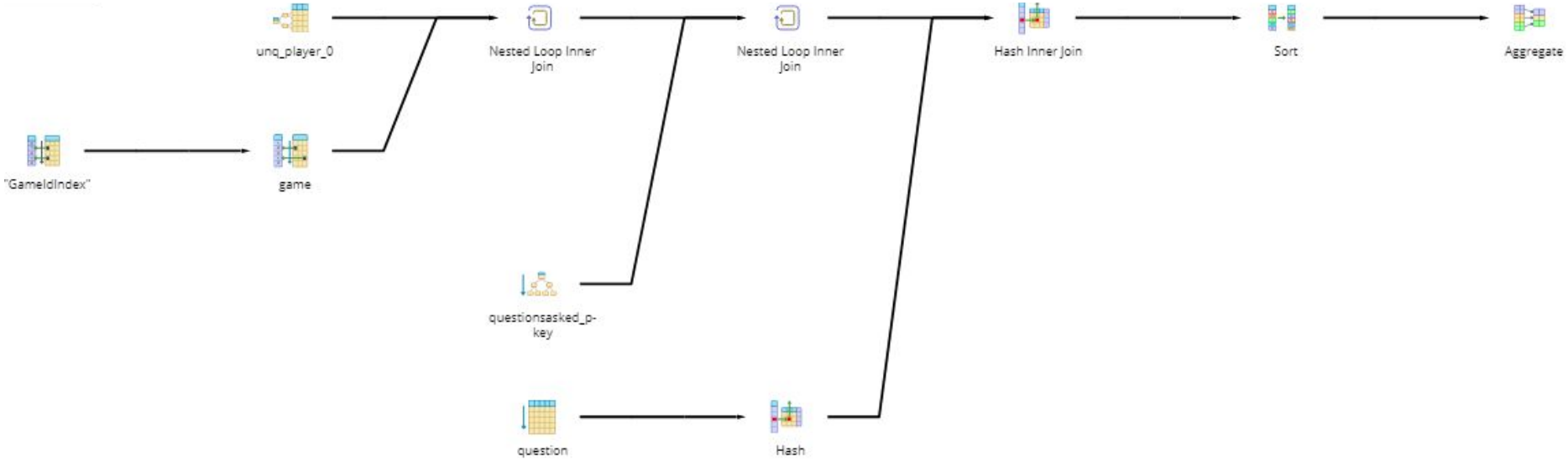
Erläuterung zur Explain-Grafik

1. Ein Index Scan wird angewandt, um die Spieler zu finden, die den angegebenen Namen haben → eine Zeile wird zurückgegeben
2. Bildung einer Hashtabelle auf das Ergebnis → Hashtable auf die ID des gefundenen Spielers
3. Hash Join zwischen der Spieler-Hashtabelle und der Game-Tabelle nach dem Kriterium Player.Id=Game.player_id. Dafür wird ein sequenzieller Scan auf die Game- Tabelle ausgeübt.
4. Index-Scan auf die Questionsasked Tabelle → es wird nach allen Records gesucht, deren Questionsasked.Game_id mit einer Game.Id in der gejointen Tabelle enthalten sind
5. Das Ergebnis des Index-Scans wird mit der Tabelle zusammengeführt.
6. Index Scan auf die Question-Tabelle → es wird nach allen Records gesucht, deren Question.id mit einer Id in der gejointen Tabelle enthalten ist
7. Das Ergebnis des Index-Scans wird mit der Tabelle zusammengeführt.
8. Es wird ein quick-sort mit Sortierkriterium game.id ausgeführt.
9. Gruppieren nach game.id und Berechnung der Gesamtanzahl der Fragen.

Optimierung für Query 2

Maßnahme	vor Optimierung	nach Einführen von Indizes
Anlegen eines Hash-Indizes auf in der Game Tabelle auf die Spalte Game.player_id	Planning: 5.188 ms Execution: 34.729 ms	Planning: 1.637 ms Execution: 3.156 ms

Explain nach Optimierung:



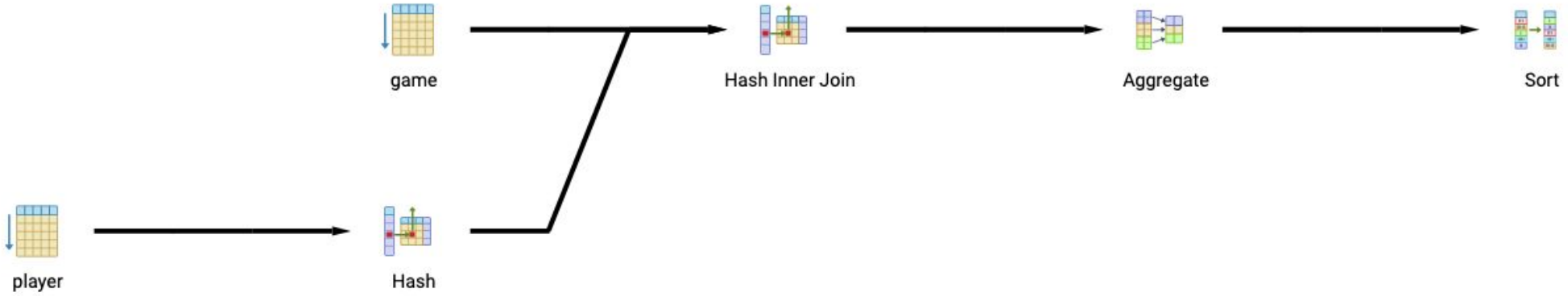
III. Ausgabe aller Spieler mit Anzahl der Spielen, nach Anzahl absteigend geordnet.

- **Query**

```
Query query = em.createQuery(
"SELECT g.player.name, count(g.id) as numberOfGames from Game as g group by g.player.name order by numberOfGames desc");
```

- **von EclipseLink erzeugte SQL- Anweisung:**

```
SELECT t0.NAME, COUNT(t1.ID) FROM Player t0, Game t1 WHERE (t0.ID = t1.PLAYER_ID) GROUP BY t0.NAME ORDER BY COUNT(t1.ID) DESC;
```



Erläuterung zur Explain-Grafik:

1. Für die Player-Table wird ein Hash-Index erstellt, wobei durch die Anwendung der Hash-Funktion auf die Player-ID die Position des Player-Records in der neu erstellten Hash-Tabelle bestimmt wird. Damit kann später sehr schnell nach den Spieler-Ids gesucht werden, die zu einem Spiel gehören.
2. Inner join der Game Tabelle mit der Player-Hashtabelle nach dem Kriterium Player.Id=Game.player_id.
3. Es folgt eine Hash-Aggregation, um die join-Tabelle nach den Spielernamen zu gruppieren und die Anzahl der Spiele zu berechnen. Player.name is der Key für die Hash-Aggregation und die Hash-Aggregation liefert für jeden Key eine Zeile zurück.
4. Es wird ein quick-sort mit der Anzahl der Spiele als Sortierkriterium ausgeführt.

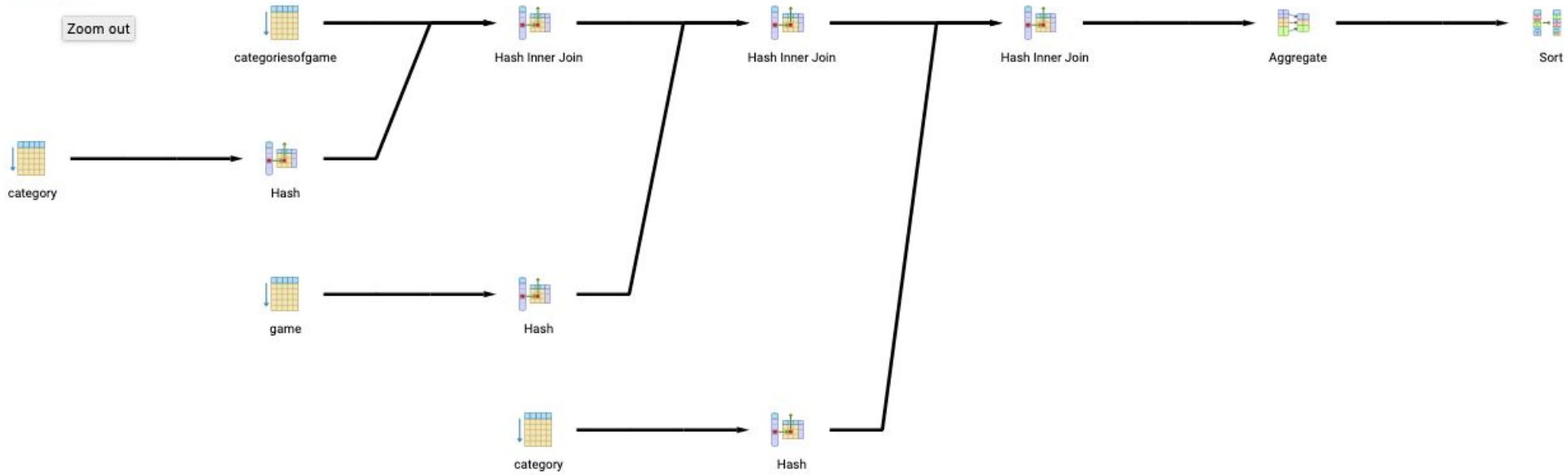
Optimierung für Query 3

- zur Optimierung wurden Hash-Indizes in der Game-Tabelle auf die Spalte Game.player_id und in der Player Tabelle auf die Spalte Player.id gelegt
- bei Ausführung der Query wurde festgestellt, dass jedoch keiner der beiden Indizes genutzt wird.

IV. Ausgabe der am meisten gefragten Kategorien

- **Query:**
  Query query = em.createQuery(
  "SELECT c.name, count(g.id) as numberOfGames from Game g, Category c where c "member of g.categoriesOfGame Group by c.name order by numberOfGames desc");

- **von EclipseLink erzeugte SQL- Anweisung:**
  Explain analyze SELECT t0.NAME, COUNT(t1.ID) FROM categoriesOfGame t3, Category t2, Game t1, Category t0 WHERE ((t0.ID = t2.ID) AND ((t3.Game_ID = t1.ID) AND (t2.ID = t3.categoriesOfGame_ID))) GROUP BY t0.NAME ORDER BY COUNT(t1.ID) DESC;



Erläuterung zur Explain-Grafik:

1. Zunächst werden Hashtabellen für die Category und Game Tabelle angelegt, um schnell nach der Category bzw. Game Id suchen zu können
2. Danach wird die categoriesOfGame Tabelle zuerst mit der Category-Hashtabelle gemäß categoriesOfGame.categoriesOfGame_id = Category.id und danach mit der Game-Hashtabelle gemäß categoriesOfGame.game_id = game. id gejoint
3. Danach wird eine weitere Hashtabelle von der Category-Tabelle erstellt und mit der aus den vorherigen Schritten erzeugten Tabelle erzeugt (notwendig??)
4. Es folgt eine Aggregatsfunktion, welche nach Category.name gruppiert und aufsummiert, in wievielen Spielen die Category gewählt wurde.
5. Zuletzt wird nach der Anzahl der Spielen absteigend sortiert.

Optimierung für Query 4

- zur Optimierung wurde ein Hash-Index in der Category Tabelle auf der Spalte id angelegt, sowie in der Game Tabelle auf die Spalte Game.id
- Gemäß unserer Recherchen würde ein Hash-Index nur im Falle eines Nested-Loops helfen, jedoch nicht bei einem Hash-Join.

Zu Beginn vorhandene Indizes:

	schemaname name	tablename name	indexname name	tabl nam	indexdef text
1	public	category	category_p...	[null]	CREATE UNIQUE INDEX category_pkey ON public.category USING btree (id)
2	public	answer	answer_pkey	[null]	CREATE UNIQUE INDEX answer_pkey ON public.answer USING btree (id)
3	public	question	question_p...	[null]	CREATE UNIQUE INDEX question_pkey ON public.question USING btree (id)
4	public	game	game_pkey	[null]	CREATE UNIQUE INDEX game_pkey ON public.game USING btree (id)
5	public	player	player_pkey	[null]	CREATE UNIQUE INDEX player_pkey ON public.player USING btree (id)
6	public	answerschosen	answersch...	[null]	CREATE UNIQUE INDEX answerschosen_pkey ON public.answerschosen USING btree (game_id, answerschosen_id)
7	public	categoriesofgame	categoriesо...	[null]	CREATE UNIQUE INDEX categoriesofgame_pkey ON public.categoriesofgame USING btree (game_id, categoriesofgame_id)
8	public	questionsasked	questionsa...	[null]	CREATE UNIQUE INDEX questionsasked_pkey ON public.questionsasked USING btree (game_id, questionsasked_id)
9	public	category	unq_catego...	[null]	CREATE UNIQUE INDEX unq_category_0 ON public.category USING btree (name)
10	public	player	unq_player_0	[null]	CREATE UNIQUE INDEX unq_player_0 ON public.player USING btree (name)