Praktikum 5 - Explain der EclipseLink SQL Queries

Datenbanken 2 - Sommersemester 2019
Gruppe: Fr2y-6
750907 | Selim Sinan
752940 | Ruben van Laack

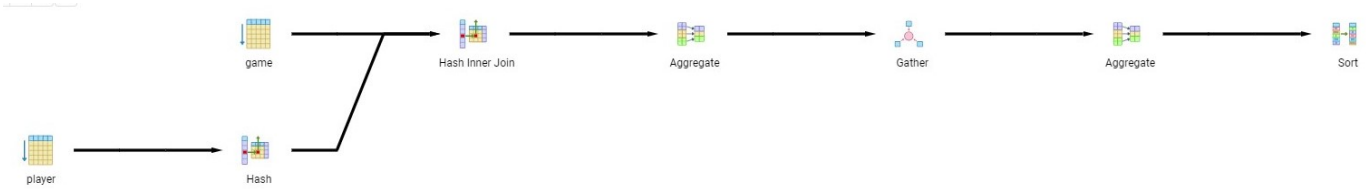
Query 1

EclipseLink SQL Query

```
SELECT t0.ID, t0.NAME
FROM master_data_knowledge_test.player t0, master_data_knowledge_test.game t1
WHERE (((t1.STARTDATETIME >= '2019-06-16 16:55:03.0') AND (t1.ENDDATETIME >= '2019-06-16 16:55:23.0'))
AND (t1.PLAYER_ID = t0.ID))
GROUP BY t0.ID, t0.NAME
ORDER BY MAX(t1.STARTDATETIME) DESC, MAX(t1.ENDDATETIME) DESC;
```

Start- und Enddatum eingesetzt

Explain Analyse



Erklärung

Postgres durchläuft zunächst die gesamte "Player" Tabelle um einen Hash-Table der Player-IDs zu erstellen.
Danach wird die "Game" Tabelle durchlaufen, dabei nach dem angegebenen Start- und Enddatum gefiltert und dann, mit einem Hash über den Player_ID-Fremdschlüssel, mit der Player Tabelle zusammengesetzt (join).
Beim Hash-Join werden also jedem Spieler aus der "Player" Tabelle die Spiele aus der nach Datum gefilterten "Game" Tabelle zugeordnet.
Nur die Spieler welchen ein Spiel zugeordnet wird, werden aufgelistet (inner join).
In dem "Aggregate" Schritt werden die Spieler mit Spielen nach dem vorher erzeugten Hash der Player-ID gruppiert.
Der nachfolgende "Gather" Schritt trennt die einzelnen Gruppen, um nachfolgende Schritte parallel für jede Gruppe in einem eigenen "Worker" durchzuführen.
In dem "Gather" Schritt werden daher parallel für jede Gruppe die Maxima der Start- und Enddaten gebildet.
Danach werden die einzelnen Gruppen, mit je einem Spieler und dessen aggregierten Spieldaten, über "Aggrigate" wieder anhand des Hash der Spieler-ID zusammengesetzt.
Schlussendlich wird die Ergebnisstabelle noch mit einem Quicksort nach Datum absteigend sortiert.

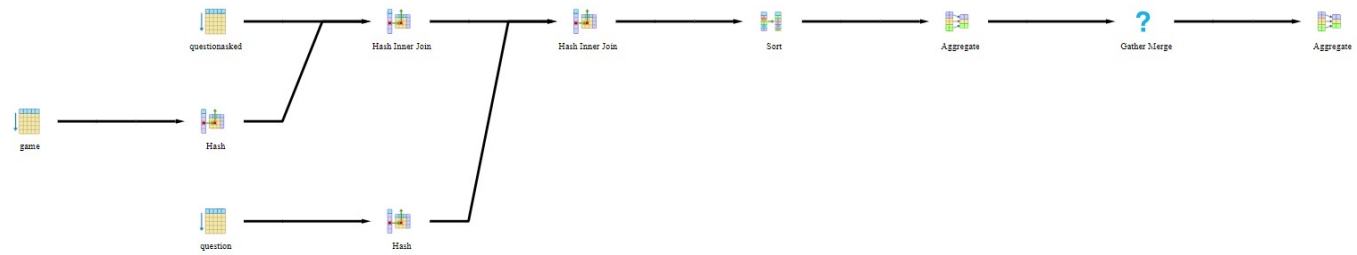
Query 2

EclipseLink SQL Query

```
SELECT t0.ID, t0.ENDDATETIME, t0.MAXQUESTIONS, t0.STARTDATETIME, t0.PLAYER_ID, COUNT(t1.ID), SUM(CASE
WHEN (t2.SELECTEDANSWER = t1.CORRECTANSWER) THEN 1 ELSE 0 END)
FROM master_data_knowledge_test.game t0, master_data_knowledge_test.questionasked t2,
master_data_knowledge_test.question t1
WHERE ((t0.PLAYER_ID = 1) AND ((t2.GAME_ID = t0.ID) AND (t2.QUESTION_ID = t1.ID)))
GROUP BY t0.ID, t0.ENDDATETIME, t0.MAXQUESTIONS, t0.STARTDATETIME, t0.PLAYER_ID
ORDER BY t0.ID ASC;
```

'Player_ID = 1' eingesetzt

Explain Analyse



Erklärung

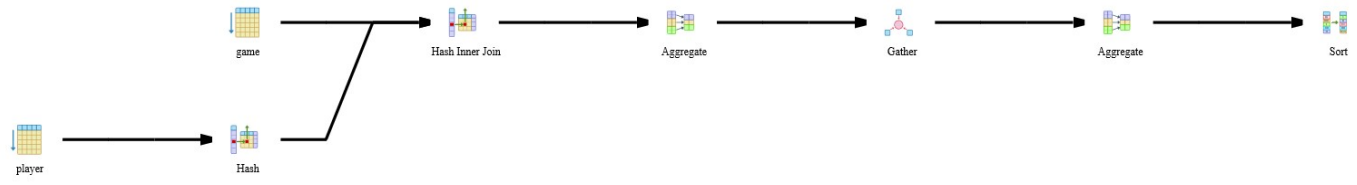
Zunächst wird die "Game" Tabelle nach dem ausgewälten Spieler gefiltert und eine Hash-Tabelle über die Spiel-ID erstellt.
Die "Questionasked" Tabelle wird nun über einen Hash für deren Spiel_ID Fremdschlüssel mit der Spiele Tabelle vereint.
Die resultierende Tabelle wird nun wieder, über Hash-Werte für den Fragen_ID Fremdschlüssel, mit der gebildeten Hash-Tabelle der Fragen vereint.
Im nächsten Schritt werden die Zeilen nach der Spiele-ID aufsteigend sortiert.
Nun wird via "Aggregate" nach Spiele-ID Gruppiert und im "Gather Merge" werden die aggregierten Werte für jede Gruppe errechnet und die Gruppen zu je einer Zeile entsprechend dem Select zusammengefasst.
Danach werden über ein weiteres "Aggregate" noch die Gruppen zu einer Tabelle zusammengebaut.

Query 3

EclipseLink SQL Query

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

Explain Analyse



Erklärung

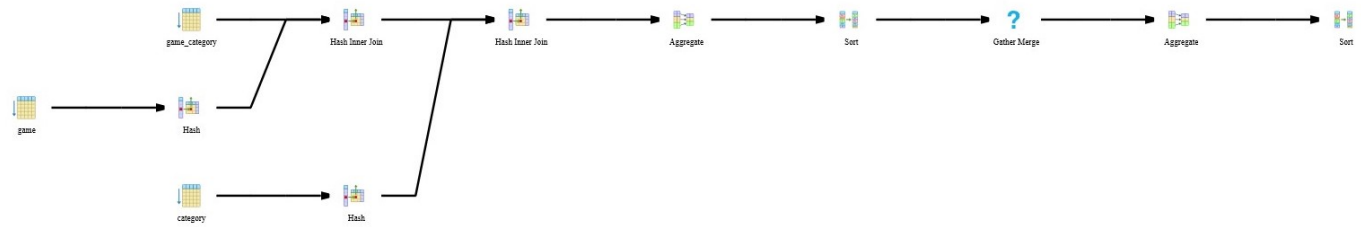
- Scan und Erstellung einer Hash Tabelle für die Spieler Tabelle.
- Zusammenführen der Spieler und Spiele Tabellen über den Hash der Spieler_ID.
- Zerteilung in Gruppen via "Aggregate" nach dem Spieler und Ermittlung der Spieleanzahl für jede Gruppe über "Gather".
- Zusammenfassung der Gruppen über "Aggregate".
- Absteigende Sortierung der Spieler nach der Anzahl an Spielen jedes Spielers in "Sort".

Query 4

EclipseLink SQL Query

```
SELECT t0.ID, t0.NAME
FROM master_data_knowledge_test.category t0, master_data_knowledge_test.game_category t2,
master_data_knowledge_test.game t1
WHERE ((t2.categoryId = t0.ID) AND (t1.ID = t2.gameId))
GROUP BY t0.ID
ORDER BY COUNT(t1.ID);
```

Explain Analyse



Erklärung

- Bildung einer Hash-Tabelle für die Spiele
- Danach wir die "Game_Category" Tabelle durchlaufen und je, via Hash des "Game_ID" Fremdschlüssels mit der Spiele-Hash-Tabelle vereinigt.
- Es wird eine Hash-Tabelle für die "Category" Tabelle erstellt und danach die bereits vereinigte Tabelle, über den Hash der "Category_ID", mit dieser Kategorie-Hash-Tabelle vereinigt.
- Die Gesamttabelle wird über "Aggregate" nach Kategorie zerteilt
- Die Gruppen werden mittels "Sort" nach nach ihrer Kategorie aufsteigend sortiert.
- Über "Gather Merge" werden die Gruppen zu je einem Tupel (einer Spalte) verbunden.
- Über "Aggregate" werden die Gruppen wieder zu einer Tabelle vereinigt und danach über "Sort" nach Anzahl der Spiele jeder Kategorie sortiert.

Praktikum 5 - Aufgabe 3

Indices

```
-- Show Indices
SELECT *
FROM pg_indexes
WHERE schemaname = 'master_data_knowledge_test';
```

Schema	Tabelle	Index	Tablespace	indexof
master_data_knowledge_test	category	category_name_key	null	CREATE UNIQUE INDEX category_name_key ON master_data_knowledge_test.category USING btree (name)
master_data_knowledge_test	category	category_pkey	null	CREATE UNIQUE INDEX category_pkey ON master_data_knowledge_test.category USING btree (id)
master_data_knowledge_test	game	game_pkey	null	CREATE UNIQUE INDEX game_pkey ON master_data_knowledge_test.game USING btree (id)
master_data_knowledge_test	player	player_name_key	null	CREATE UNIQUE INDEX player_name_key ON master_data_knowledge_test.player USING btree (name)
master_data_knowledge_test	player	player_pkey	null	CREATE UNIQUE INDEX player_pkey ON master_data_knowledge_test.player USING btree (id)
master_data_knowledge_test	question	question_pkey	null	CREATE UNIQUE INDEX question_pkey ON master_data_knowledge_test.question USING btree (id)
master_data_knowledge_test	questionasked	questionasked_pkey	null	CREATE UNIQUE INDEX questionasked_pkey ON master_data_knowledge_test.questionasked USING btree (id)
master_data_knowledge_test	game_category	unq_game_category_0	null	CREATE UNIQUE INDEX unq_game_category_0 ON master_data_knowledge_test.game_category USING btree (gameid, categoryid)
master_data_knowledge_test	game_category	game_category_pkey	null	CREATE UNIQUE INDEX game_category_pkey ON master_data_knowledge_test.game_category USING btree (categoryid, gameid)

Tabellen Statistiken

```
-- Show count statistics
SELECT relname, n_live_tup
FROM pg_stat_user_tables;
```

relname	n_live_tup
player	10000
category	51
questionasked	5127942
question	200
game_category	3501252
game	1000000

```
-- Count of Player, Games and Askedquestions
SELECT
(SELECT COUNT(player.id) FROM master_data_knowledge_test.player) AS player_count,
(SELECT COUNT(game.id) FROM master_data_knowledge_test.game) AS game_count,
(SELECT COUNT(questionasked.id) FROM master_data_knowledge_test.questionasked) AS
askedQuestions_count;
```

player_count	game_count	askedquestion_count
10000	1000000	5127942

Erstellung von Indices

Optimierungen - Neue Indices

```
-- Game
CREATE INDEX game_fk_player
    ON master_data_knowledge_test.game (player_id);
CREATE INDEX game_maxquestions
    ON master_data_knowledge_test.game (maxquestions);
CREATE INDEX game_startdatetime
    ON master_data_knowledge_test.game (startdatetime);
CREATE INDEX game_enddatetime
    ON master_data_knowledge_test.game (enddatetime);

-- QuestionAsked
CREATE INDEX questionasked_fk_game
    ON master_data_knowledge_test.questionasked (game_id);
CREATE INDEX questionasked_fk_question
    ON master_data_knowledge_test.questionasked (question_id);

-- GameCategory
CREATE INDEX game_category_fk_category
    ON master_data_knowledge_test.game_category (categoryid);
CREATE INDEX game_category_fk_game
    ON master_data_knowledge_test.game_category (gameid);
```

Query 1 - Ausführungszeiten

Vorher:

- Planning Time: 0.211 ms
- Execution Time: 1784.087 ms

Nachher:

- Planning Time: 0.467 ms
- Execution Time: 1767.881 ms

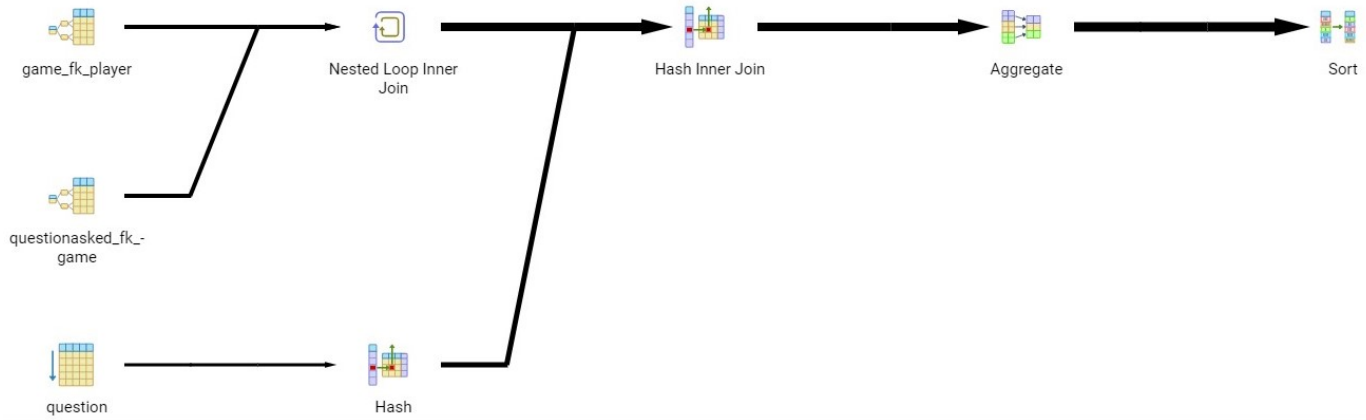
Query 2 - Ausführungszeiten

Vorher:

- Planning Time: 0.453 ms
- Execution Time: 346.246 ms

Nachher:

- Planning Time: 0.462 ms
- Execution Time: 0.059 ms



Query 3 - Ausführungszeiten

Vorher:

- Planning Time: 0.237 ms
- Execution Time: 1506.696 ms

Nachher:

- Planning Time: 0.323 ms
- Execution Time: 1461.244 ms

Query 4 - Ausführungszeiten

Vorher:

- Planning Time: 0.721 ms
- Execution Time: 9648.261 ms

Nachher:

- Planning Time: 0.678 ms
- Execution Time: 9546.708 ms