Database Technologies Project – Phase 1

Overview

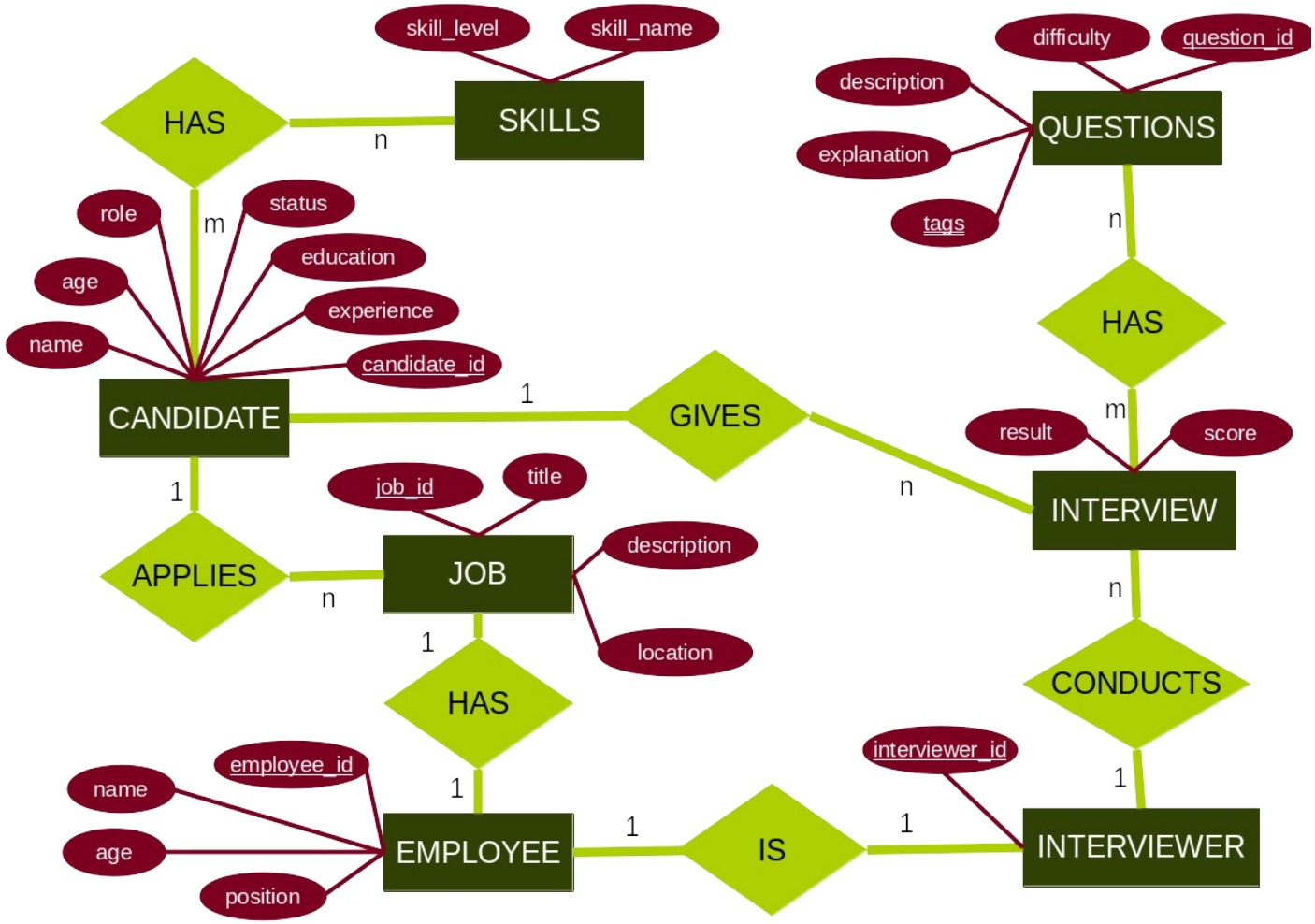
The project models a company interview database, storing data about candidates, employees, job offerings, interviews and other related information. For the submission last semester, the schema has been developed, normalized and a CLI tool was written to perform DML and DQL operations on the database. The objectives of this phase are -

1. To simulate creation of database on multiple disks, splitting the data and log files across multiple folders
2. To create file groups and partitions to control where the data is being stored
3. To analyse the performance improvement afforded by creating indices on specific attributes

The entire project can be found at https://github.com/anihm136/DBMS_Project. Relevant SQL files are uploaded in the project submission folder as well. SQL Server 2019 on an Ubuntu 18.04 Docker container has been used for the project, mssql-cli for executing the queries and Microsoft Azure Data Studio for visualizing the execution plans

ER Diagram

The ER diagram of the model is as follows -



A. Creation of Database

```
CREATE DATABASE DBMS
CONTAINMENT=NONE
ON PRIMARY (NAME=N'dbms_main', FILENAME=N'/var/opt/mssql/data/dbms_primary/dbms_primary.mdf', SIZE=8192KB, FILEGROWTH=65535KB)
LOG ON (NAME=N'dbms_log', FILENAME=N'/var/opt/mssql/data/dbms_log/dbms_log.ldf', SIZE=8192KB, FILEGROWTH=65535KB)
GO

ALTER DATABASE DBMS ADD FILEGROUP FG1
ALTER DATABASE DBMS ADD FILE (NAME=N'FG1_F1', FILENAME=N'/var/opt/mssql/data/dbms_data/disk1/fg1_f1.ndf', SIZE=8192KB , FILEGROWTH=65536KB) TO FILEGROUP FG1
ALTER DATABASE DBMS ADD FILE (NAME=N'FG1_F2', FILENAME=N'/var/opt/mssql/data/dbms_data/disk2/fg1_f2.ndf', SIZE=8192KB , FILEGROWTH=65536KB) TO FILEGROUP FG1
ALTER DATABASE DBMS ADD FILEGROUP FG2
ALTER DATABASE DBMS ADD FILE (NAME=N'FG2_F1', FILENAME=N'/var/opt/mssql/data/dbms_data/disk1/fg2_f1.ndf', SIZE=8192KB , FILEGROWTH=65536KB) TO FILEGROUP FG2
ALTER DATABASE DBMS ADD FILE (NAME=N'FG2_F2', FILENAME=N'/var/opt/mssql/data/dbms_data/disk2/fg2_f2.ndf', SIZE=8192KB , FILEGROWTH=65536KB) TO FILEGROUP FG2
GO
```

The database is created with a primary filegroup at `/var/opt/mssql/data/dbms_primary`., with a single file `dbms_primary.mdf` and a log file at `/var/opt/mssql/data/dbms_log/dbms_log.ldf`. The primary filegroup is used only to store the system database files, as all the user files will be stored on different user file groups which will be created next

B. Creation of filegroups and partitioning

Two partitions FG1 and FG2 are created, each having two data files. The data files for each file group are split across two folders to simulate a data protection scheme with data files on different disks. This can also be achieved with a single data file on a RAID array.

```
CREATE PARTITION FUNCTION PF_SMALL (INT)
AS RANGE LEFT FOR VALUES (500)
CREATE PARTITION FUNCTION PF_MED (INT)
AS RANGE LEFT FOR VALUES (10000)
CREATE PARTITION FUNCTION PF_LARGE (INT)
AS RANGE LEFT FOR VALUES (50000)

CREATE PARTITION SCHEME PS_SMALL
AS PARTITION PF_MED
TO (FG1, FG2)
CREATE PARTITION SCHEME PS_MED
AS PARTITION PF_MED
TO (FG1, FG2)
CREATE PARTITION SCHEME PS_LARGE
AS PARTITION PF_LARGE
TO (FG1, FG2)
GO
```

Three partition functions (and corresponding partition schemes) are created to partition small, medium and large relations respectively into the two user file groups. Relations which fall in between these definitions are stored on a single file group, alternating between the two to ensure even data distribution. Some sample relation sizes for the three categories are -

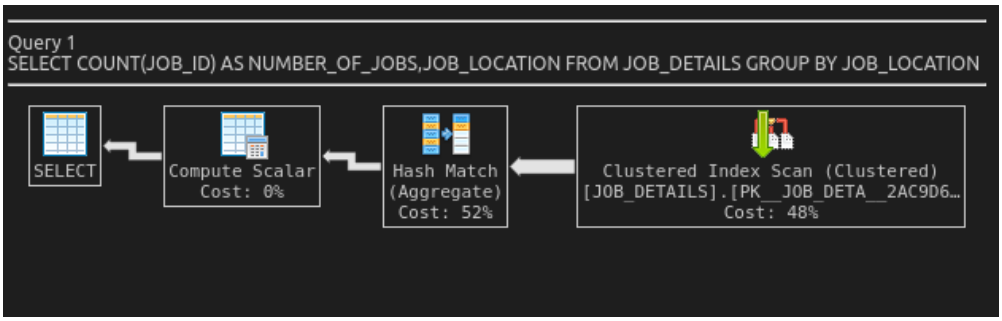
- Small: 1474 records (QUESTION)
- Medium: 7999 records (EMPLOYEE)
- Large: 1000000 records (CANDIDATE)

C. Queries and indexing optimization

Three main queries are analysed for performance, each with differing levels of complexity. In the third case, Azure Data Studio makes a suggestion for creating an index on CANDIDATE_ID in the MAP table, which is followed. The rest of the indexes are selected based on analysis of performance

1. `SELECT COUNT(JOB_ID) AS NUMBER_OF_JOBS,JOB_LOCATION FROM JOB_DETAILS GROUP BY JOB_LOCATION;` (selects the number of jobs in each location). Index is created on

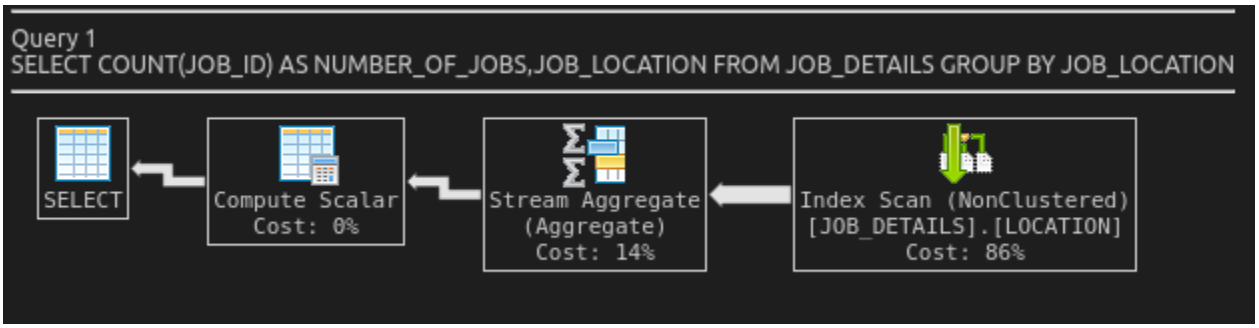
Before index



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Hash Match(Aggregate)		52	0.289947	284	284	1	0.149774	0	false		0		0	false
Compute Scalar		0	0.289947	284	284	1	0	0	false		0		0	false
Clustered Index Scan	[D…	48	0.140173	17759	17759	1	0.0198489	0.120324	false		0		0	true

After index

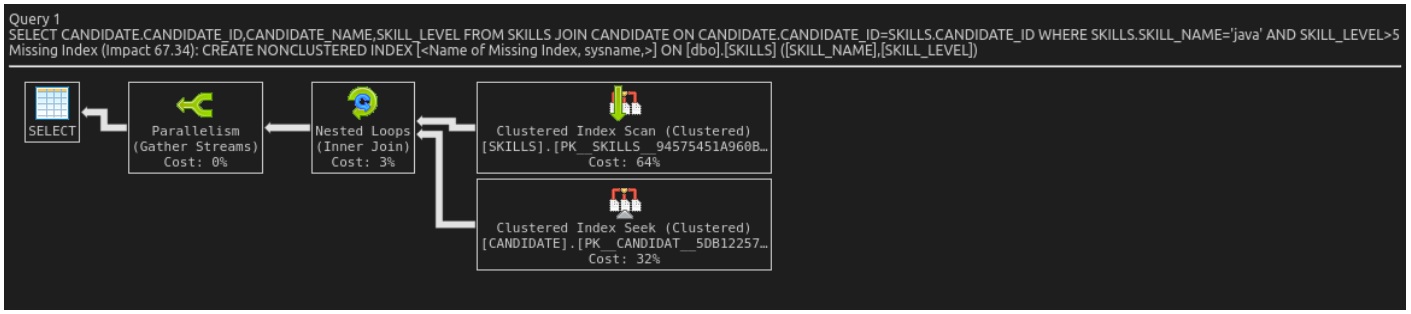
An index is created on the JOB_LOCATION attribute, allowing an index scan on the location attribute directly and changing the hash based aggregate to a stream aggregate. We can see that this makes a noticeable difference in total IO cost, dropping from 0.12 to 0.04



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Index Scan	[D…	86	0.0672613	17759	17759	1	0.0196919	0.0475694	false		0		0	false
Stream Aggregate(Ag…		14	0.0780587	284	284	1	0.0107974	0	false		0		0	false
Compute Scalar		0	0.0780587	284	284	1	0	0	false		0		0	false

2. `SELECT CANDIDATE.CANDIDATE_ID,CANDIDATE_NAME,SKILL_LEVEL FROM SKILLS JOIN CANDIDATE ON CANDIDATE.CANDIDATE_ID=SKILLS.CANDIDATE_ID WHERE SKILLS.SKILL_NAME='java' AND SKILL_LEVEL>5;` (selects the candidates with the skill 'java' and an aptitude of over 5)

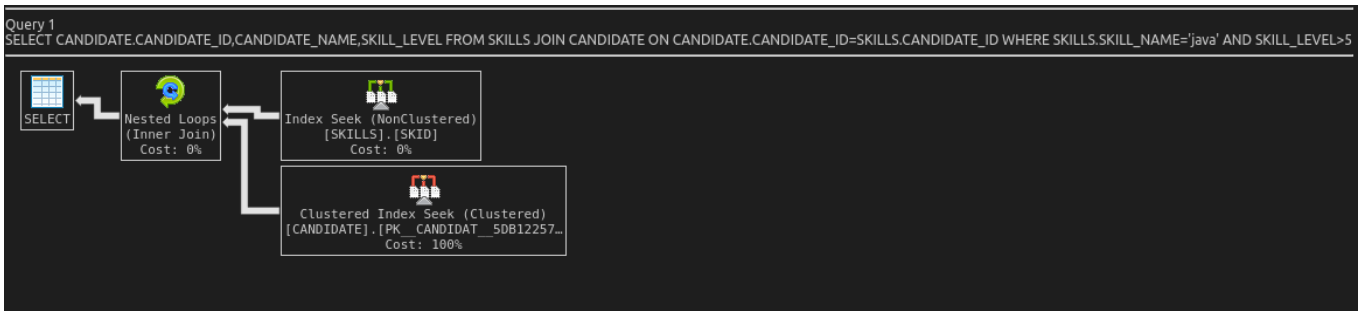
Before index



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Clustered Index Scan	[D…	64	12.7685	1012	1995.82	8	0.825195	11.9433	true		0		0	true
Clustered Index Seek	[D…	32	6.4643	1012	1	1012	0.0001581	0.003125	true		1994.82		0	true
Nested Loops(Inner Jo…		3	19.8949	1012	1995.43	8	0.00208563	0	true		0		0	false
Parallelism(Gather Str…		0	19.9268	1012	1995.43	1	0.0319129	0	true		0		0	false

After index

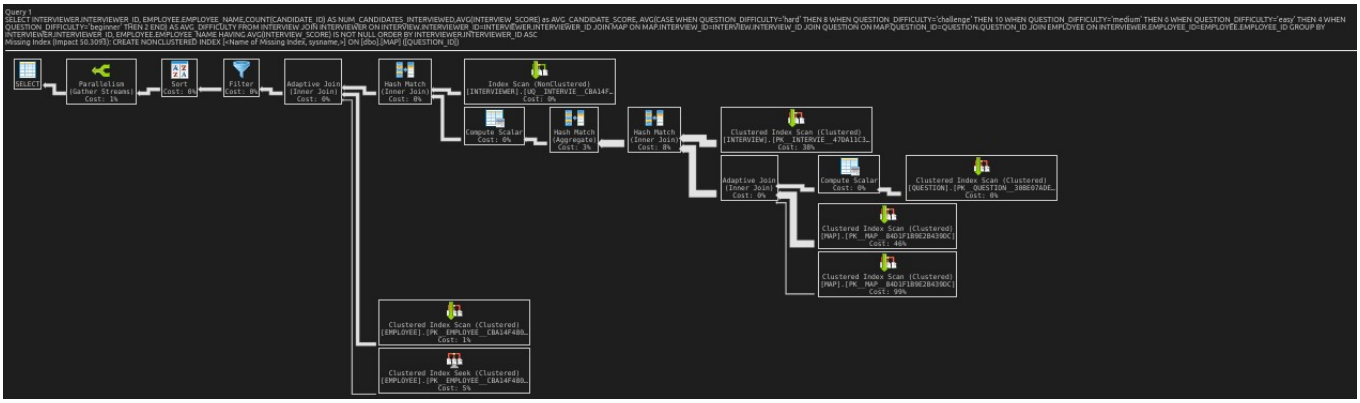
An composite index is created on the SKILL_NAME and SKILL_LEVEL attributes of the SKILLS table. Since the skill name is usually referred before searching for the skill level, placing it first in the composite index makes a difference in the performance of the index. The conversion from an index scan to an index seek brings a large reduction to IO cost



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Clustered Index Seek	[D…	100	4.92415	1012	1	1012	0.0001581	0.003125	false		1505.99		0	true
Nested Loops(Inner Jo…		0	4.9391	1012	1506.7	1	0.00629924	0	false		0		0	false
Index Seek	[D…	0	0.0086434	1012	1506.99	1	0.00181469	0.0068287	false		0		0	false

3. `SELECT INTERVIEWER.INTERVIEWER_ID,EMPLOYEE.EMPLOYEE_NAME,COUNT(CANDIDATE_ID) AS NUM_CANDIDATES_INTERVIEWED,AVG(INTERVIEW_SCORE) as AVG_CANDIDATE_SCORE, AVG(CASE WHEN QUESTION_DIFFICULTY='hard' THEN 8 WHEN QUESTION_DIFFICULTY='challenge' THEN 10 WHEN QUESTION_DIFFICULTY='medium' THEN 6 WHEN QUESTION_DIFFICULTY='easy' THEN 4 WHEN QUESTION_DIFFICULTY='beginner' THEN 2 END) AS AVG_DIFFICULTY FROM INTERVIEW JOIN INTERVIEWER ON INTERVIEW.INTERVIEWER_ID=INTERVIEWER.INTERVIEWER_ID JOIN MAP ON MAP.INTERVIEW_ID=INTERVIEW.INTERVIEW_ID JOIN QUESTION ON MAP.QUESTION_ID=QUESTION.QUESTION_ID JOIN EMPLOYEE ON INTERVIEWER.EMPLOYEE_ID=EMPLOYEE.EMPLOYEE_ID GROUP BY INTERVIEWER.INTERVIEWER_ID, EMPLOYEE.EMPLOYEE_NAME HAVING AVG(INTERVIEW_SCORE) IS NOT NULL ORDER BY INTERVIEWER.INTERVIEWER_ID ASC;` (gives a full summary of interviewers)

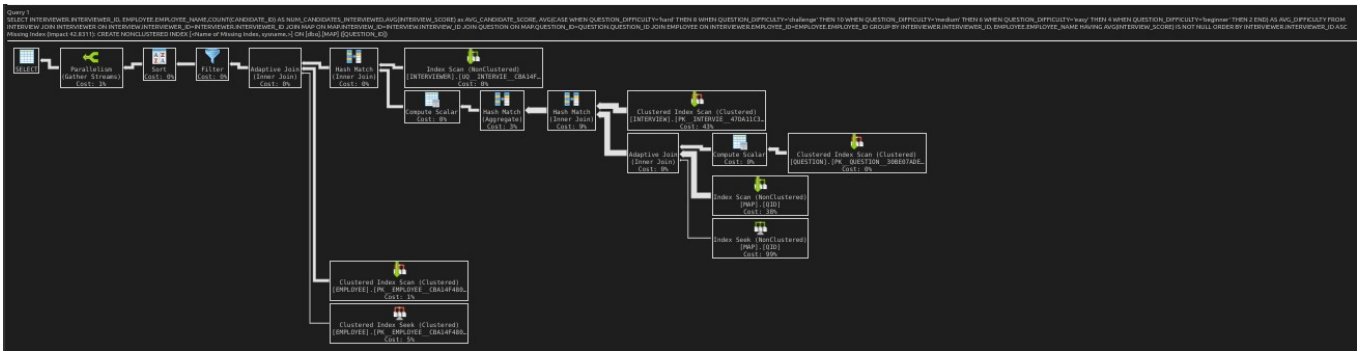
Before index



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Clustered Index Scan	[D…	99	6.5362	0	948.595	0	0.846646	1.46238	true		1473		0	true
Clustered Index Scan	[D…	46	3.04207	1398229	1.39823…	8	0.384709	2.65736	true		0		0	true
Clustered Index Scan	[D…	38	2.47598	700000	700000	8	0.192696	2.28329	true		0		0	true
Hash Match(Inner Join)		8	6.309	1398229	1.39823…	8	0.505227	0	true		0		0	false
Clustered Index Seek	[D…	5	0.311225	0	1	0	0.0001581	0.003125	true		999		3.33067e-15	false
Hash Match(Aggregate)		3	6.4851	1000	1000	8	0.176094	0	true		0		0	false
Parallelism(Gather Str…		1	6.57127	1000	1000	1	0.0350724	0	true		0		0	false
Clustered Index Scan	[D…	1	0.0409175	1000	799.9	8	0.00223698	0.0386806	true		0		0	false
Sort		0	6.5362	1000	1000	8	0.000413677	0.000938438	true		0		0	false
Filter		0	6.53484	1000	1000	8	1.2e-05	0	true		0		0	false
Hash Match(Inner Join)		0	6.49154	1000	1000	8	0.00225672	0	true		0		0	false
Index Scan	[D…	0	0.00417999	1000	1000	8	0.00031425	0.00386574	true		0		0	false

After index

Here, as suggested by the tool, an index is created on the QUESTION_ID attribute of the MAP table, which converts a group of clustered index scans to an index seek, drastically improving the corresponding IO costs



Operation	O…	Est Cost	Est Subtree Cost	Actual Rows	Est Rows	Actual Executions	Est CPU Cost	Est IO Cost	Parallel	Actual Rebinds	Est Rebinds	Actual Rewinds	Est Rewinds	*Partitioned*
Index Seek	[D…	99	5.67662	0	948.595	0	0.00120045	0.00386574	true		1473		0	false
Clustered Index Scan	[D…	43	2.47598	700000	700000	8	0.192696	2.28329	true		0		0	true
Index Scan	[D…	38	2.18249	1398229	1.39823…	8	0.384552	1.79794	true		0		0	false
Hash Match(Inner Join)		9	5.44942	1398229	1.39823…	8	0.505227	0	true		0		0	false
Clustered Index Seek	[D…	5	0.311225	0	1	0	0.0001581	0.003125	true		999		3.33067e-15	false
Hash Match(Aggregate)		3	5.62552	1000	1000	8	0.176094	0	true		0		0	false
Parallelism(Gather Str…		1	5.71169	1000	1000	1	0.0350724	0	true		0		0	false
Clustered Index Scan	[D…	1	0.0409175	1000	799.9	8	0.00223698	0.0386806	true		0		0	false
Sort		0	5.67662	1000	1000	8	0.000413677	0.000938438	true		0		0	false
Filter		0	5.67527	1000	1000	8	1.2e-05	0	true		0		0	false
Hash Match(Inner Join)		0	5.63196	1000	1000	8	0.00225672	0	true		0		0	false
Index Scan	[D…	0	0.00417999	1000	1000	8	0.00031425	0.00386574	true		0		0	false