Next-Generation Parallel Query

Robert Haas & Rafia Sabih

Overview

- v10 Improvements
- TPC-H Results
- TPC-H Analysis
- Thoughts for the Future

© 2017 EDB All rights reserved.

Parallel Table Scans

- Parallel Seq Scan
  - fully supported from v9.6
- Index Scan
  - supported for btree indexes from v10
- Index-Only Scan
  - supported for btree indexes from v10
- Bitmap Heap Scan
  - supported for all index types from v10, but the underlying index scan(s) are not parallel

© 2017 EDB All rights reserved.

ENTERPRISEDB

Parallel Joins

- In 9.6, we can do nested loops and hash joins in parallel plans.
- In 10, we can also do merge joins.
- However, all joins are still parallel-oblivious.
  - Proposed patch: parallel shared hash.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Miscellaneous Improvements

- Gather Merge merges sorted streams of tuples from all participants into a single sorted result. (This combines nicely with Parallel btree Index Scan and Parallel Merge Join.)
- Uncorrelated subplans can now be executed in workers (but each worker repeats the work, just like each worker repeats the inner side of the join).
- Access to parallel query from PLs has been improved.
- Query text is now passed to workers.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

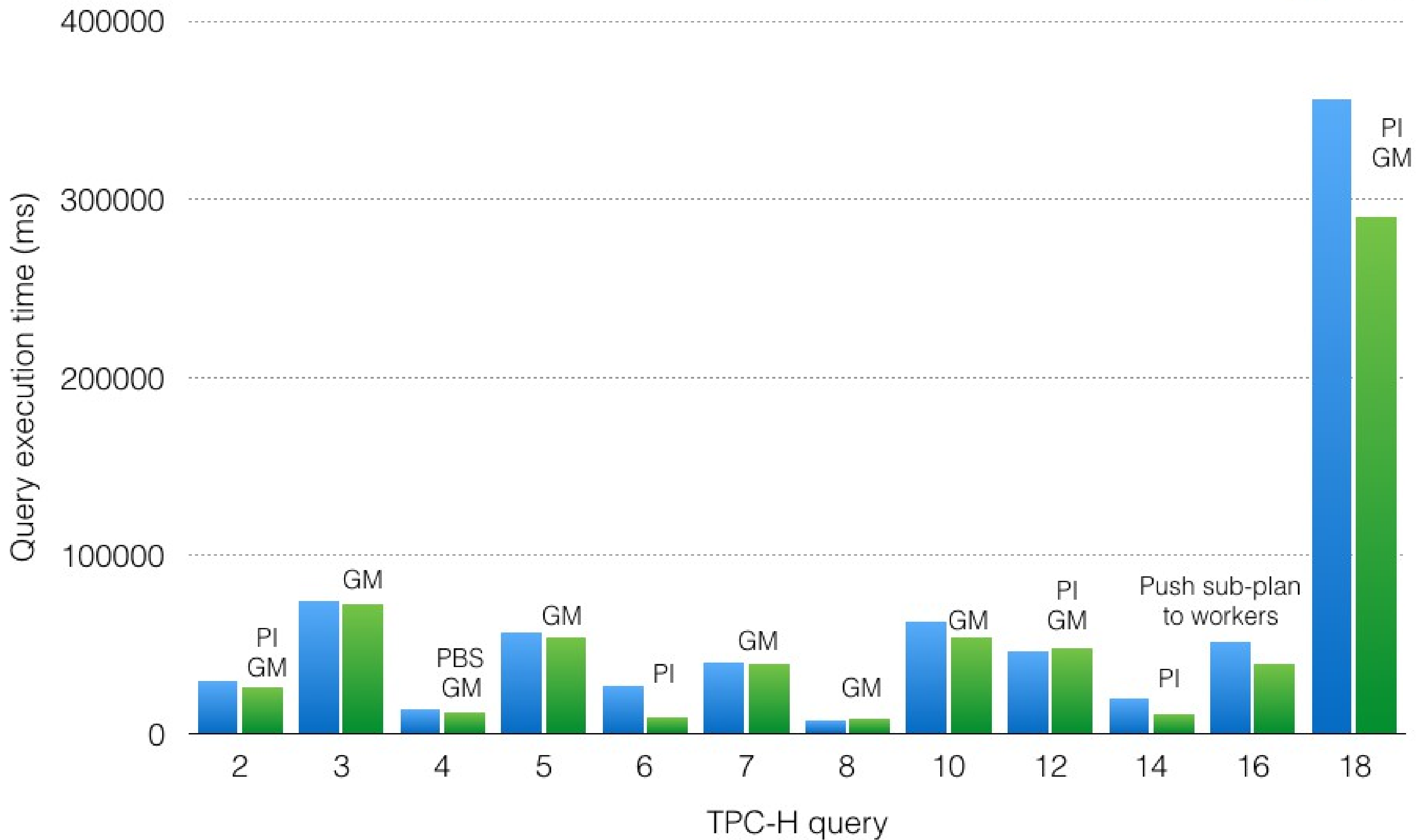
Performance Evaluation on TPC-H

- Experimental Setup
  - RAM = 512 GB
  - Number of Cores = 32
- Server Settings
  - work_mem = 64 MB @ scale 20, 1 GB @ scale 300
  - shared_buffers = 8GB
  - effective_cache_size = 10GB
  - random_page_cost = seq_page_cost = 0.1
  - max_parallel_workers_per_gather = 4
- Database Setup
  - PG 10 = Mar 9 snapshot + some bug fixes
  - Additional indexes on l_shipmode, l_shipdate, o_orderdate, o_comment

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

TPC-H Results – Scale Factor 20



© 2017 EDB All rights reserved.

EDB
ENTERPRISEDB

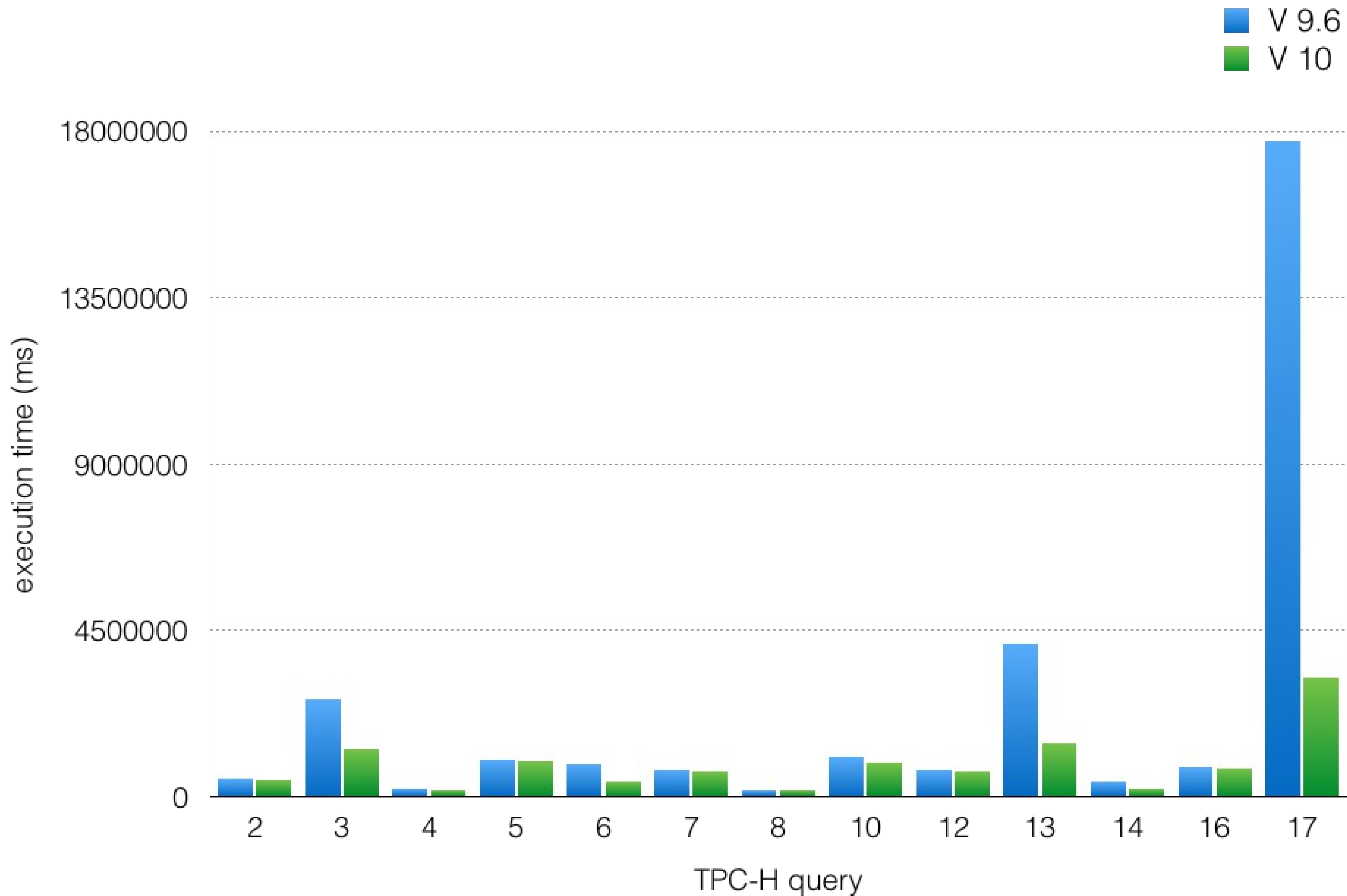
TPC-H Results – Scale Factor 20

	v9.6	v10	% Speedup	Used
Q2	29636.046	25559.894	13.75%	GM, PI
Q3	74394.648	72530.694	2.5%	GM
Q4	13706.858	12207.417	10.93%	GM, PBHS
Q5	56856.605	53708.296	5.53%	GM
Q6	27151.837	8839.169	67.44%	PI
Q7	40237.334	39356.078	2.19%	GM
Q8	7746.69	7953.079	-2.66%	GM
Q10	62751.493	54007.126	13.93%	GM, PBHS
Q12	46601.751	47829.493	-2.63%	GM, PI
Q14	20025.412	11229.026	43.92%	PI
Q16	51340.742	39242.498	23.56%	Subplan
Q18	355953.923	290311.054	18.44%	GM, PI

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

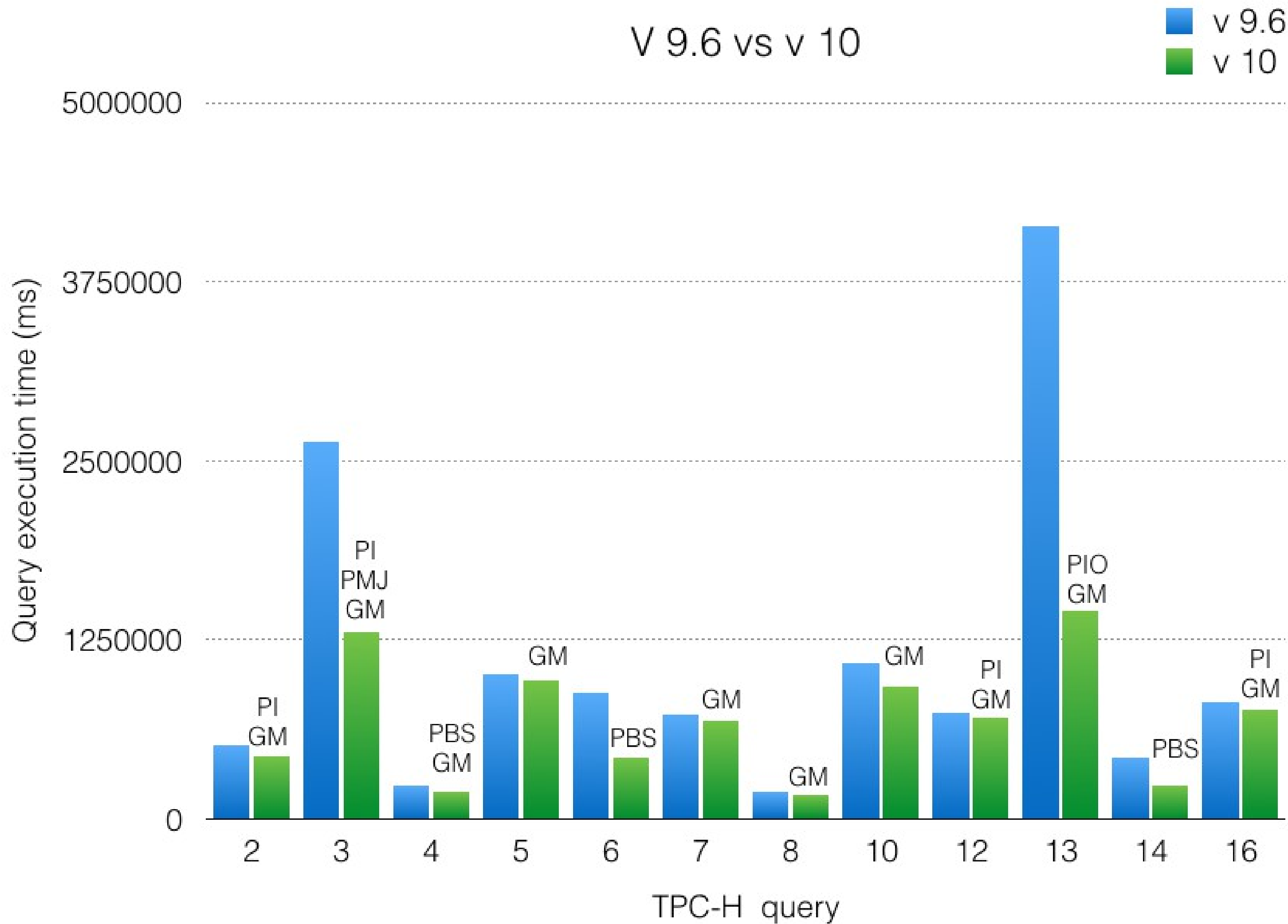
TPC-H Results – Scale Factor 300



ENTERPRISEDB

© 2017 EDB All rights reserved.

TPC-H Results – Scale Factor 300



© 2017 EDB All rights reserved.

ENTERPRISEDB

TPC-H Results – Analysis (1/4)

- Parallel Index Scan was a big win!
  - Q6 is basically nothing but an index scan, so it gets the most speedup.
  - Q14 @ SF 20 has an index scan on the inner side of a hash join; the outer side gets no faster, but the index scan does.
  - Q2 only manages to use parallel query for a small part of the query, but that part got faster.
- Parallel Index-Only Scan was only chosen once (Q13 @ scale 300), but was a win (parallelism chosen instead of not).

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

TPC-H Results – Analysis (2/4)

- Parallel Bitmap Heap Scan wins at scale factor 300.
  - At scale factor 20, it's only chosen once (Q4), but was a win when it was chosen.
  - At scale factor 300, it's chosen three times (Q4, Q6, Q14) and the queries get faster in every case.
  - Q6 and Q14 speed up massively, because they previously had no viable parallel plan.
  - On other tests, Parallel Bitmap Heap Scan sometimes regressed because the planner thinks the extent to which the bitmap lossifies doesn't matter. The planner is wrong! Fixed here by choosing the “right” value for work_mem.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

TPC-H Results – Analysis (3/4)

- Gather Merge got used a lot, but didn't always help.
  - **Good**: The plan contains an expensive sort, and Gather Merge lets us avoid it.
    - Example: in Q17 @ scale 300, we Gather Merge 300 million rows and then perform Finalize GroupAggregate.
  - **Harmless**: The plan contains a sort that was already cheap. This is fairly common.
    - Example: In Q8 @ scale 300, we Gather Merge 6 rows.
  - **Bad**: Some workers take a long time to produce a tuple, forcing other workers to wait.
    - Example: In Q12 @ scale 300, nothing gets faster despite Parallel Index Scan + Gather Merge; pipeline stalls likely to blame.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

TPC-H Results – Analysis (4/4)

- Parallel Merge Join is useful at the higher scale factor.
  - In Q3 @ scale 300, the availability of Parallel Merge Join allows a substantial chunk of the plan to be parallelized where that otherwise wouldn't have been practical.
  - But could win in many more cases with Parallel-Aware Merge Join.
- Relaxing subplan restrictions is also useful.
  - Q16 @ scale 20 was using parallelism before, but now it can be used by a much larger chunk of the plan, hence the speedup.
  - This area could benefit from a lot more work.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Amdahl's Law

$$S(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

© 2017 EDB All rights reserved.

ENTERPRISEDB

Amdahl's Law – Parallel Aggregate

```
Hash Join
-> Finalize Aggregate
   -> Gather
     -> Partial Aggregate
         -> Partial Plan
 -> Hash
   -> Non-Partial Plan
```

- If Finalize Aggregate itself is the slow part, then this plan isn't helping us much.
- Also, we've terminated parallelism, so the surrounding hash join can't be done in parallel.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Better Parallel Aggregate?

```
Gather
-> Hash Join
  -> Parallel Aggregate
    -> Partial Plan
  -> Parallel Shared Hash
    -> Partial Plan
```

- For a HashAggregate, it's fairly easy to see how to implement this: use a shared hash table.
  - Could contend if # of groups is small.
  - Serialize/deserialize might be expensive.
- For a GroupAggregate, it's not so obvious how to make this work.
  - Merging sorted streams will likely contend badly.
  - Also, Merge → Parallel Index Scan is a bad idea....

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Amdahl's Law – Parallel Join

- Current parallel joins work by having each worker join some of the rows on the outer side of the join to all of the rows on the inner side of the join.
- This means that parallel speedup is possible only on one side of the join; the work on the other side has to be redone by each worker (and may even be slower).
- Parameterized nested loops are an exception.
- The problem gets worse as the join nest gets deeper.
- Goal: Make joins parallel-aware so that each worker needs to only scan part of *each* input.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Amdahl's Law – Parallel-Aware Join

- Parallel-aware hash join (patch exists)

```
Gather
-> Parallel Hash Join
   -> Partial Plan
   -> Parallel Shared Hash
      -> Partial Plan
```

- Parallel-aware merge join?

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Better Parallel Merge Join - Ideas

- Parallel-Aware:
  - Partition-wise Join + Parallel Append
  - Break into N sub-merge-joins using hashing
- Still Parallel-Oblivious, But Maybe Faster:
  - Parallel Materialize
  - Skip Scans

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Amdahl's Law – More Applications

- **More Work on InitPlan/SubPlan Restrictions**. Parameter references make tables parallel-restricted. Uncorrelated subplans are re-executed.
- **Parallel Bitmap Index Scan**. Parallelizing the heap scan is good, but sometimes the index scan is quite expensive.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Planner Problems

- Need to improve costing for Parallel Bitmap Heap Scan (ignores lossification).
- Need to investigate cost of Bitmap Heap Scan vs. Bitmap Index Scan.
- Need to improve costing for HashAggregate (sometimes picked even when Gather Merge + GroupAggregate executes faster).
- Need to improve selectivity estimation (some estimates are very, very bad).
- Need to improve algorithm for selecting the # of workers.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Awful Estimate

```
-> Merge Join  (cost=54848846.88..62064940.91 rows=92 width=16)
(actual time=2241882.471..3255130.278 rows=1696742 loops=1)
    Merge Cond: ((lineitem.l_partkey = partsupp.ps_partkey) AND
(lineitem.l_suppkey = partsupp.ps_suppkey))
    Join Filter: ((partsupp.ps_availqty)::numeric > ((0.5 *
sum(lineitem.l_quantity))))
      Rows Removed by Join Filter: 3771
```

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Conclusions

- Parallel query in PostgreSQL 10 is substantially improved from PostgreSQL 9.6.
- But there's a lot more to do.
- Stay tuned.
- Please help.

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB

Thank You

Output: Thank You
Gather
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Index Scan on Common_phrases
    Index Cond: (value = 'Thank You')
      Filter: Language = 'English'

© 2017 EDB All rights reserved.

EDB ENTERPRISEDB