

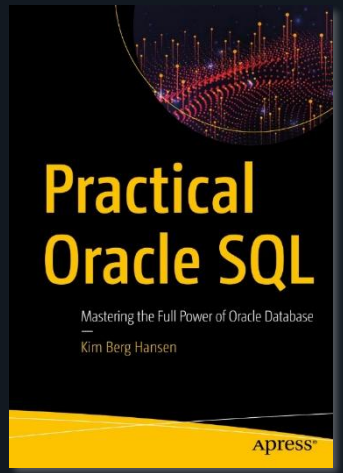
OK Oracle – Tell Me *Approximately* How Many Unique Visitors We Had Last Week

Kim Berg Hansen

Kim Berg Hansen

- SQL & PL/SQL developer since 2000
- Author of “Practical Oracle SQL”
- Cegal Danmark since 2022
- Oracle ACE Director
- Founding member of MASH Program
- Member of Symposium 42
- SQL quizmaster at devgym.oracle.com
- OCE in SQL
- Likes to cook and read sci-fi
- Member of Danish Beer Enthusiasts
- Bluesky @kibeha.dk
- Blog https://kibeha.dk
- Other links https://linktr.ee/kibeha





Oracle ACE

Tech Superstars Unite

Get worldwide recognition as an Oracle ACE



Oracle.com profile page



Exclusive content



Your own Oracle cloud account



Swag, certification exam credit & event passes



Networking events



Travel support

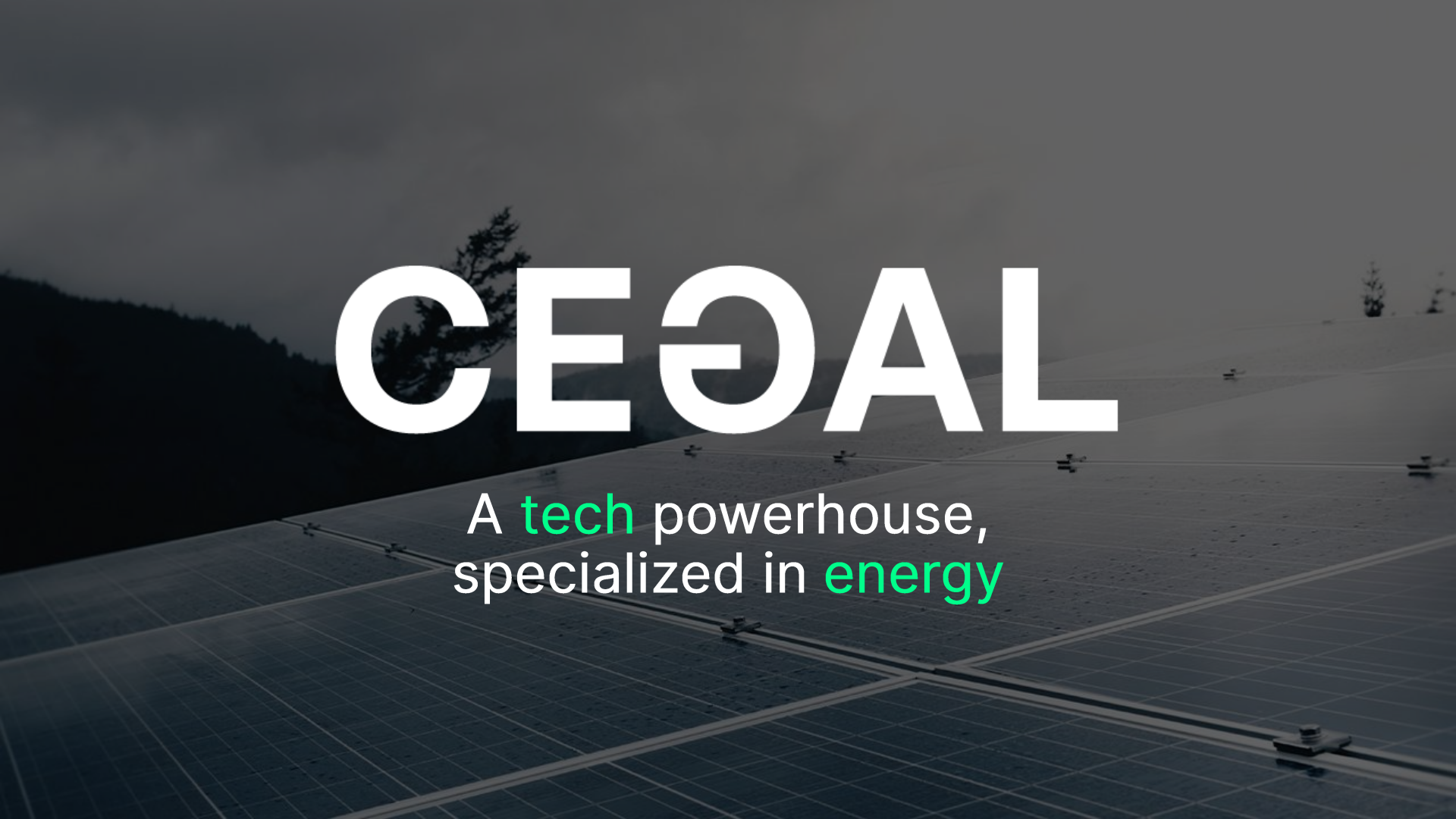
@oracleace

Linkedin.com/groups/72183

@oracleace.bsky.social

Learn more at:
ace.oracle.com



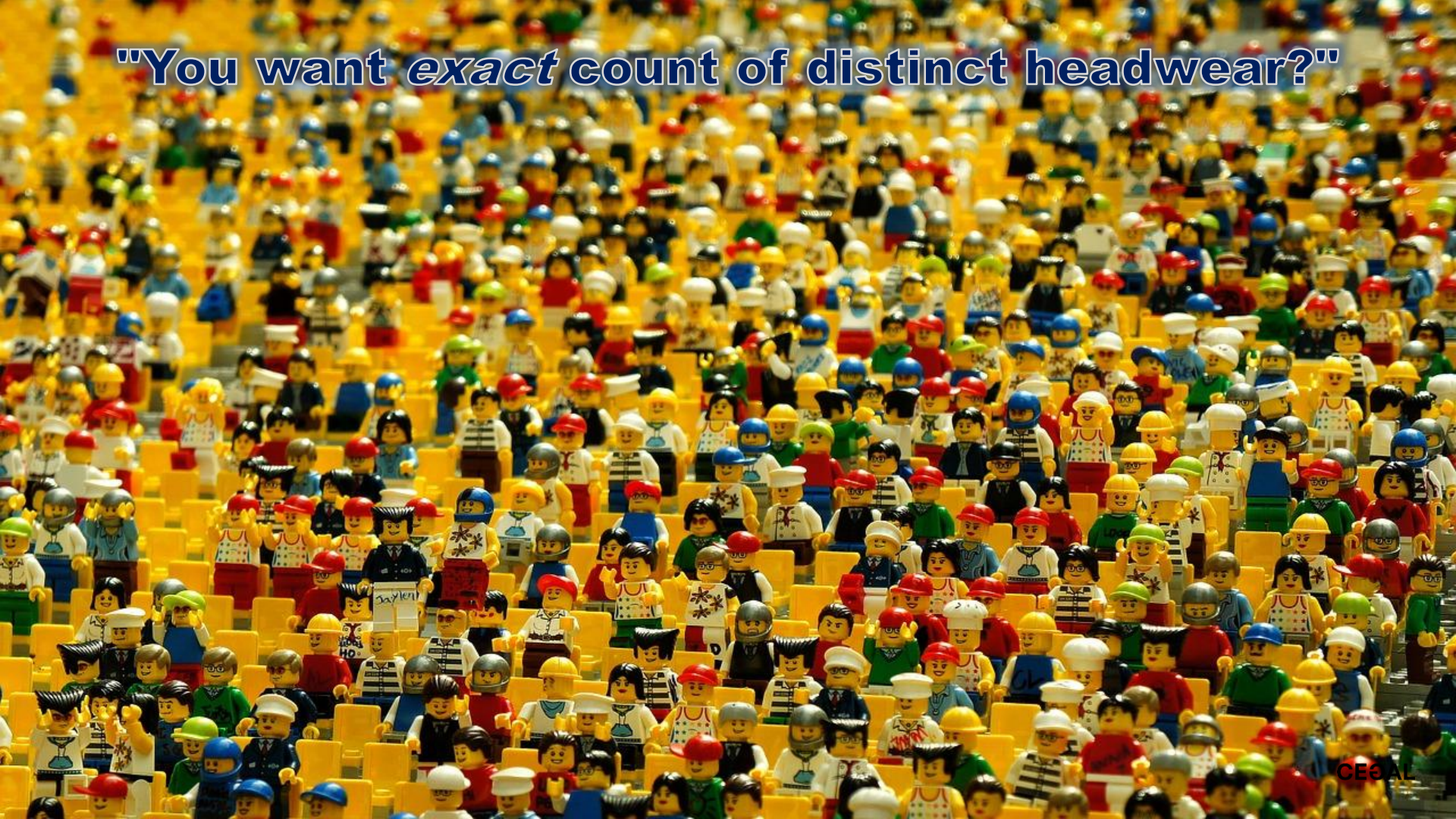

CEƎAL
A tech powerhouse,
specialized in energy

Agenda

OK Oracle – Tell Me *Approximately* How …

- Approximate query processing functions APPROX_*
- Precision and performance trade-offs
- Approximate multilevel aggregation
- Approximate materialized views and query rewrite
- Initialization parameters for executing exact queries with approximations instead
- Exactly rather than approximately

Approximate query processing functions APPROX_*

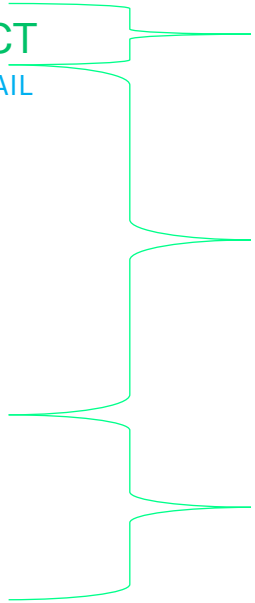

"You want *exact* count of distinct headwear?"
CE∃AL

Approximate query processing

- Alternatives for some aggregate functions
- Faster for large datasets
- Does not return exact results but approximations
- Usually within a few percent accuracy
- Scans all the data (not sampling)
- Great for exploring data quickly
  - then switch to exact when the sweet spot is found
- Uses (among others) HyperLogLog algorithm
  - Very efficient one-pass cardinality estimation algorithm

APPROX functions

- APPROX_COUNT_DISTINCT
  - APPROX_COUNT_DISTINCT_DETAIL
  - APPROX_COUNT_DISTINCT_AGG
  - TO_APPROX_COUNT_DISTINCT
- APPROX_PERCENTILE
  - APPROX_PERCENTILE_DETAIL
  - APPROX_PERCENTILE_AGG
  - TO_APPROX_PERCENTILE
- APPROX_MEDIAN
- APPROX_RANK
- APPROX_COUNT
- APPROX_SUM



12.1.0.2

12.2.0.1

18.1

CEGAL

Examples

- Examples of exact aggregates and their approximate counterparts

```
select count(distinct headgear)
  from lego_crowd


select percentile_cont(0.25)
         within group (order by score)
  from exam_results



select median(score) from exam_results
```

```
select approx_count_distinct(headgear)
  from lego_crowd


select approx_percentile(
          0.25 deterministic
       ) within group (order by score)
  from exam_results


select approx_median(
          score deterministic
       )
  from exam_results
```

APPROX_COUNT_DISTINCT

12.1.0.2

- Approximate alternative to COUNT(DISTINCT)

```
select approx_count_distinct(zip_code)
from orders
```

APPROX_PERCENTILE

12.2.0.1

- Approximate alternative to PERCENTILE_CONT / PERCENTILE_DISC
- Can show you the error rate and the confidence of the error rate

```
select approx_percentile(0.25) within group (order by score)
from exam_results


approx_percentile(0.25 deterministic) within group (order by score)

approx_percentile(0.25, 'ERROR_RATE') within group (order by score)

approx_percentile(0.25, 'CONFIDENCE') within group (order by score)
```

- DETERMINISTIC requires integer ORDER BY between -2147483648 and 2147483647
- Non-deterministic can use any numeric or datetime ORDER BY

APPROX_MEDIAN

12.2.0.1

- Approximate alternative to MEDIAN
- (Like MEDIAN it is basically a shortcut for the 50% percentile)

```
select approx_median(score)
from exam_results


approx_percentile(0.5) within group (order by score)

approx_median(score deterministic)

approx_median(score, 'ERROR_RATE')

approx_median(score, 'CONFIDENCE')
```

APPROX_RANK / COUNT / SUM

18.1

- Approximate "TOP-N" queries

```
select class
     , student
     , approx_count(*) as num_results
     , approx_rank(
         partition by class
         order by approx_count(*) desc
       ) as results_rank
from exam_results
group by class
       , student
having approx_rank(
         partition by class
         order by approx_count(*) desc
       ) <= 3
order by class, num_results desc
```

```
select class
     , student
     , approx_sum(score) as score_sum
     , approx_rank(
         partition by class
         order by approx_sum(score) desc
       ) as score_rank
from exam_results
group by class
       , student
having approx_rank(
         partition by class
         order by approx_sum(score) desc
       ) <= 3
order by class, score_sum desc
```

Precision and performance trade-offs


"Can't we call it approximately equal?"
CEGAL

COUNT(DISTINCT) vs. APPROX_COUNT_DISTINCT

- 5 tables each with 9 million rows - test all rows, test only January (= ~1/6 of rows)
- Each test run 4 times - result is average of run 2, 3 and 4

```
create table orders_o3 (
   order_id   integer      primary key
 , sales_date date         not null
 , zip_code   varchar2(10) not null
)


create table orders_o4 (
...
create table orders_o5 (
...
create table orders_o6 (
...
create table orders_o7 (
...
```

```
-- 6 months of sales dates
-- 785 distinct 3-digit zip-codes


-- 8723 distinct 4-digit zip-codes

-- 88112 distinct 5-digit zip-codes

-- 806110 distinct 6-digit zip-codes

-- 4912684 distinct 7-digit zip-codes
```

Comparisons when run in 18c

		ORDERS_03		ORDERS_04		ORDERS_05		ORDERS_06		ORDERS_07	
Aggregate	*Subset*	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***
Count(distinct)	All rows	785	**1.24**	8723	**1.34**	88112	**3.26**	806110	**7.08**	4912684	**11.03**
Approx_count_distinct	All rows	785	**0.71**	9190	**0.80**	85774	**0.81**	804032	**0.85**	4920036	**0.75**
Difference		*100%*	***57%***	*105%*	***60%***	*97%*	***25%***	*100%*	***12%***	*100%*	***7%***
Count(distinct)	January	785	**0.64**	8723	**0.66**	83527	**1.29**	527168	**2.69**	1349594	**4.53**
Approx_count_distinct	January	785	**0.51**	9190	**0.50**	81620	**0.56**	529742	**0.50**	1390180	**0.46**
Difference		*100%*	***79%***	*105%*	***76%***	*98%*	***43%***	*100%*	***19%***	*103%*	***10%***

Comparisons when run in 26ai

		ORDERS_03		ORDERS_04		ORDERS_05		ORDERS_06		ORDERS_07	
Aggregate	*Subset*	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***	*Count*	***Secs***
Count(distinct)	All rows	785	**2.08**	8723	**1.76**	88112	**1.67**	806110	**3.09**	4912684	**10.23**
Approx_count_distinct	All rows	785	**1.42**	9190	**1.20**	85774	**1.21**	804032	**1.29**	4920036	**1.39**
Difference		*100%*	***68%***	*105%*	***68%***	*97%*	***72%***	*100%*	***42%***	*100%*	***14%***
Count(distinct)	January	785	**1.06**	8723	**1.03**	83527	**1.15**	527168	**1.54**	1349594	**1.68**
Approx_count_distinct	January	785	**1.05**	9190	**1.00**	81620	**1.38**	529742	**0.99**	1390180	**1.04**
Difference		*100%*	***99%***	*105%*	***97%***	*98%*	***120%***	*100%*	***64%***	*103%*	***62%***

Takeaways

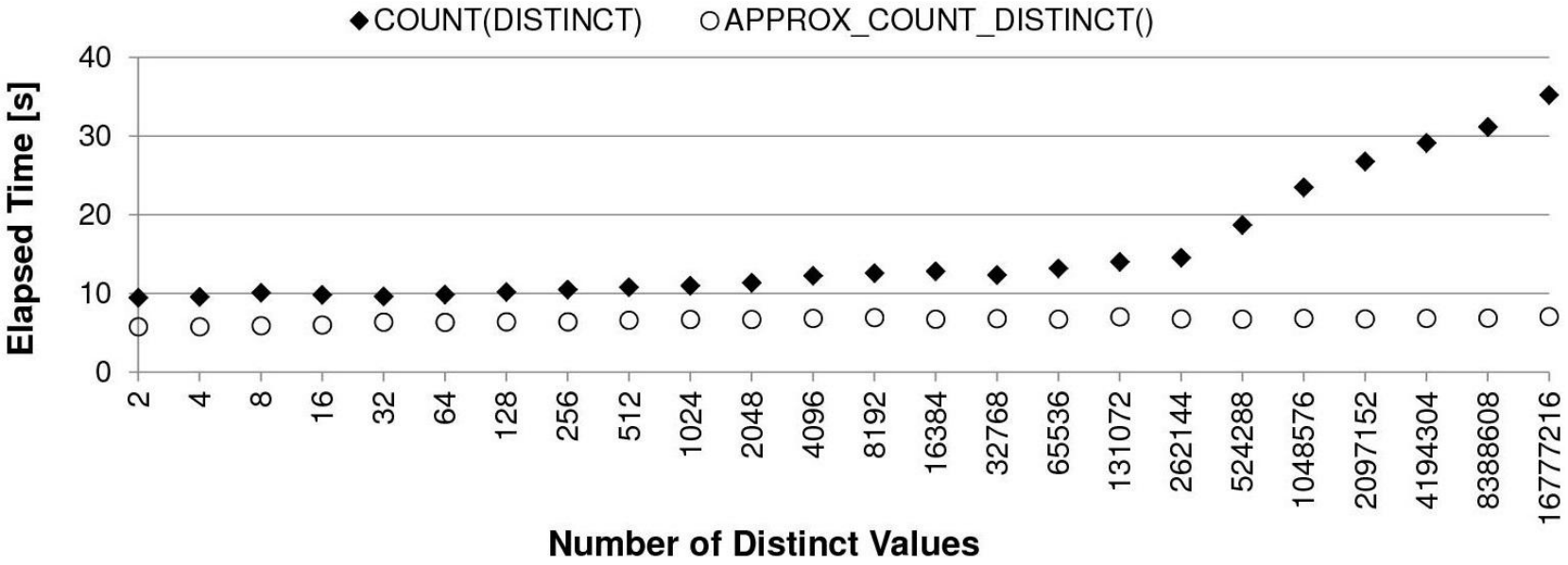
- Both methods take longer for all 9 million rows compared to counting only January
- That January isn't 6 times faster is because it's full table scanning also for January

- COUNT(DISTINCT) takes significantly longer the higher the count of distinct values
  - But in 26ai it takes a higher distinct count to make an impact versus 18c
- APPROX_COUNT_DISTINCT takes about the same time for all 5 tables

A larger test of APPROX_COUNT_DISTINCT

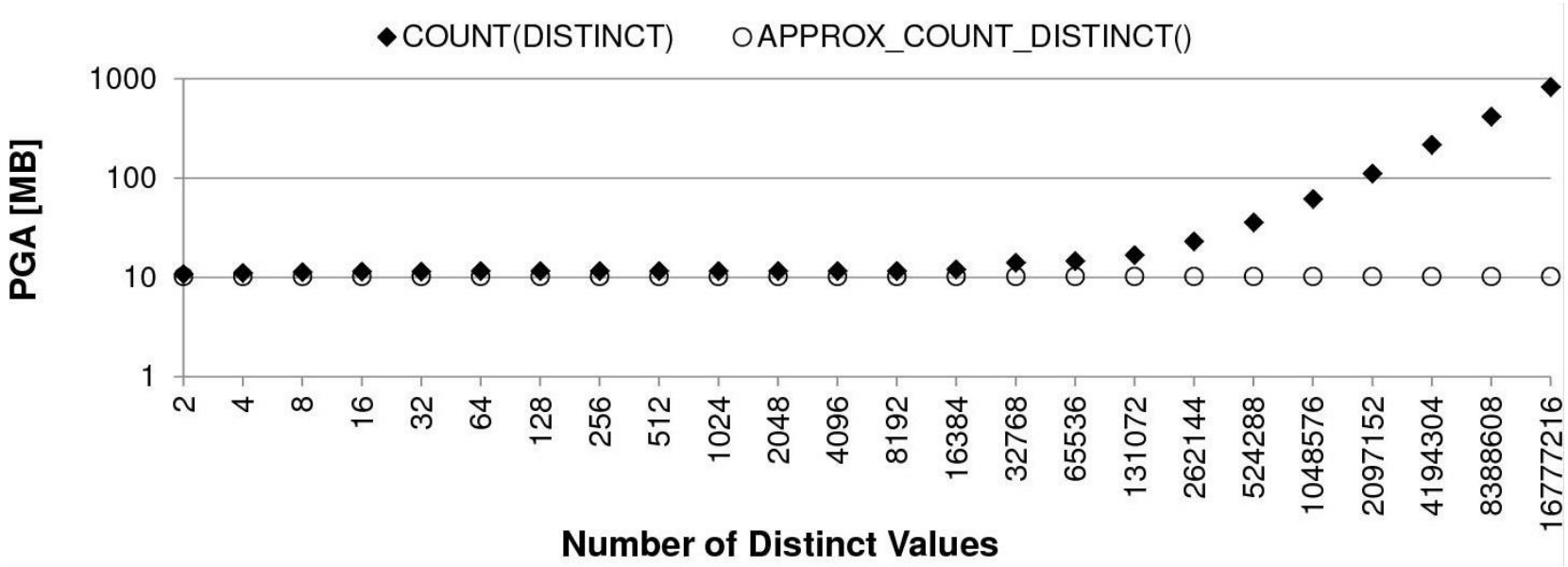
- Christian Antognini: https://antognini.ch/2014/10/the-approx_count_distinct-function-a-test-case/
- 100 million rows - 12.7 GB table
- 24 different NUMBER columns with 2^1..2^24 distinct values
- Measured elapsed time, max. PGA usage and precision

Elapsed time



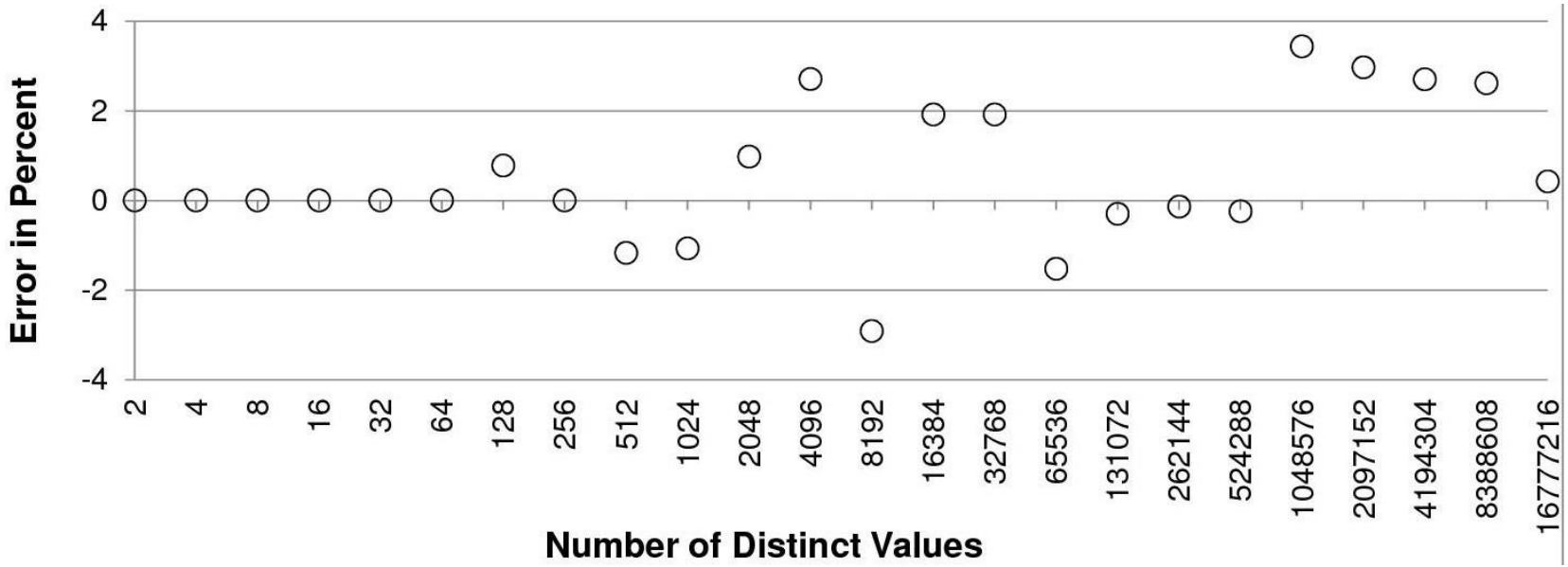
Source: https://antognini.ch/2014/10/the-approx_count_distinct-function-a-test-case/

PGA usage (logarithmic Y-axis)



Source: https://antognini.ch/2014/10/the-approx_count_distinct-function-a-test-case/

Precision



Source: https://antognini.ch/2014/10/the-approx_count_distinct-function-a-test-case/

CEƎAL

Approximate multilevel aggregation


"How did you count the straws that fast?"
CEƎAL

Aggregating distinctly?

- Aggregating a DISTINCT aggregate is not normally possible

```
create materialized view orders_day_mv
as
select sales_date
     , count(*) row_cnt
     , count(distinct zip_code)
          as day_cnt
  from orders_o7
 group by sales_date
```

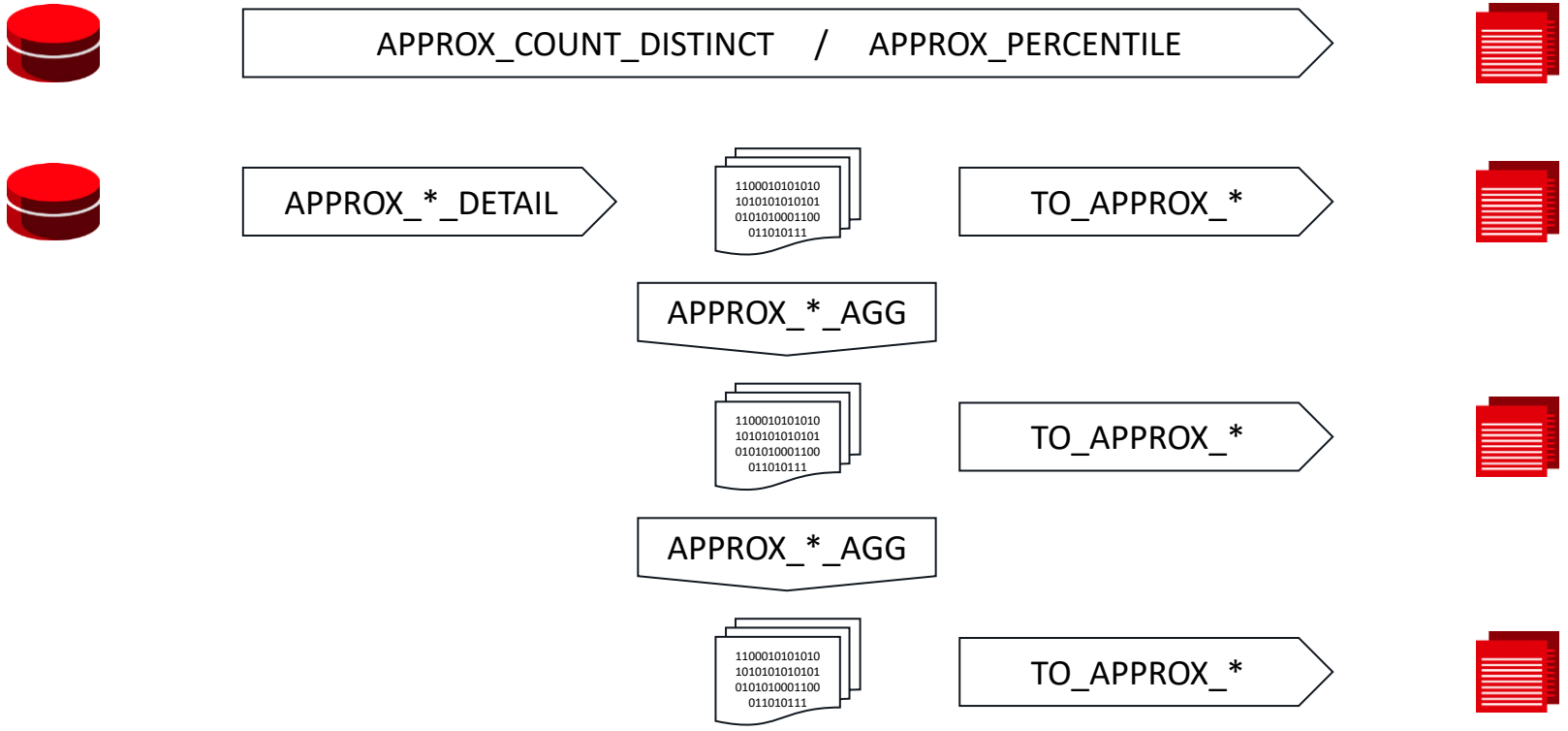
```
select sum(row_cnt)
  from orders_day_mv
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

```
select sum(distinct day_cnt)  --???
  from orders_day_mv
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```



Multilevel aggregation

12.2.0.1



CEGAL

Aggregate details

12.2.0.1

- APPROX_*_DETAIL / AGG creates a BLOB with details that can be aggregated further

```
create materialized view orders_day_mv
as
select sales_date
     , approx_count_distinct_detail(zip_code) as day_detail
  from orders_o7
 group by sales_date


create materialized view orders_mth_mv
as
select trunc(sales_date, 'MM') as sales_mth
     , approx_count_distinct_agg(day_detail) as mth_detail
  from orders_day_mv
 group by trunc(sales_date, 'MM')
```

Details to results

12.2.0.1

```
select to_approx_count_distinct(day_detail) as zip_count_per_day
  from orders_day_mv
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-31'


select to_approx_count_distinct(mth_detail) as zip_count_per_month
  from orders_mth_mv
 where sales_mth = DATE '2018-01-01'


select to_approx_count_distinct(
          approx_count_distinct_agg(day_detail)
       ) as zip_count_week_total
  from orders_day_mv
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

Use case: Stats on partitioned table

12.2.0.1

- Use incremental stats on partition level
- Set algorithm to HYPERLOGLOG
- Makes partition count distinct saved as "synopsis"
- Global partition count aggregates all synopsis
- Only stale partitions need to be regathered

```
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','INCREMENTAL','TRUE');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','INCREMENTAL_LEVEL','PARTITION');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','APPROXIMATE_NDV_ALGORITHM','HYPERLOGLOG');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','INCREMENTAL_STALENESS','USE_STALE_PERCENT');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','STALE_PERCENT','10');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','GRANULARITY','AUTO');
dbms_stats.set_table_prefs('MYSCHEMA','SALESHIST','CASCADE','TRUE');
```

Approximate materialized views and query rewrite

CEGAL


"Rewrite that query before noon, Johnson!"
Underwood
CEƎAL

Fast refresh approximate materialized views

- Materialized view logs must exist on the base tables
- In general for approximate mviews
  - FAST refresh is possible if there is no WHERE clause
- Exception to the rule
  - If the mview use APPROX_PERCENTILE_DETAIL or APPROX_PERCENTILE_AGG and the only DML on the base table are INSERTs, then FAST refresh is possible even with a WHERE clause

Materialized view for query rewrite

- Can query MV explicitly for a period

```
select to_approx_count_distinct(approx_count_distinct_agg(day_detail))
         as zip_count
  from orders_day_mv
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

- Use ENABLE QUERY REWRITE like normal MV query rewrite

```
create materialized view orders_day_mv enable query rewrite as
select sales_date
     , approx_count_distinct_detail(zip_code) as day_detail
  from orders_o7
 group by sales_date
```

Query rewrite not enabled

- Without query rewrite, APPROX_COUNT_DISTINCT uses the table
- Takes about one second on average

```
alter session set query_rewrite_enabled=false

select approx_count_distinct(zip_code) as zip_count
  from orders_o7
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

```
-------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |    16 |  9426   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE APPROX|          |     1 |    16 |            |          |
|*  2 |   TABLE ACCESS FULL  | ORDERS_O7 |   350K|  5468K|  9426   (2)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Query rewrite enabled

- With query rewrite, APPROX_COUNT_DISTINCT uses the MV
- Takes about 25 milliseconds on average (*full scan 183 rows = half year*)

```
alter session set query_rewrite_enabled=true

select approx_count_distinct(zip_code) as zip_count
  from orders_o7
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |  3134 |    29   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE APPROX        |               |     1 |  3134 |            |          |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| ORDERS_DAY_MV |     8 | 25072 |    29   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Query rewrite - approx versus normal

- Approx can do query rewrite that uses multilevel aggregation
- Normal COUNT(DISTINCT) cannot

```
create materialized view orders_day_mv
enable query rewrite as
select sales_date
     , approx_count_distinct_detail(
          zip_code
       ) as day_detail
  from orders_o7
 group by sales_date

select approx_count_distinct(zip_code)
          as zip_count
  from orders_o7
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

Rewritten

```
create materialized view orders_day_mv
enable query rewrite as
select sales_date
     , count(distinct zip_code)
          as day_count
  from orders_o7
 group by sales_date


select count(distinct zip_code)
          as zip_count
  from orders_o7
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

No rewrite

Initialization parameters for executing exact queries with approximations instead


"I'll show them FAST=TRUE!"
CEƎAL

Initialization parameters

- Three parameters can make exact code be executed with APPROX functions
  - APPROX_FOR_COUNT_DISTINCT
  - APPROX_FOR_PERCENTILE
  - APPROX_FOR_AGGREGATION

- Can be set at multiple levels
  - ALTER SYSTEM
  - ALTER SESSION
  - Statement hint: /*+ OPT_PARAM(parm, value) */

APPROX_FOR_COUNT_DISTINCT

• Value	• Exact code	• Executes as
FALSE	(Default)	(Normal execution)
TRUE	COUNT(DISTINCT)	APPROX_COUNT_DISTINCT()

APPROX_FOR_PERCENTILE

• Value	• Exact code	• Executes as
NONE	(Default)	(Normal execution)
PERCENTILE_CONT	PERCENTILE_CONT()	APPROX_PERCENTILE()
PERCENTILE_CONT DETERMINISTIC	PERCENTILE_CONT()	APPROX_PERCENTILE(DETERMINISTIC)
PERCENTILE_DISC	PERCENTILE_DISC()	APPROX_PERCENTILE()
PERCENTILE_DISC DETERMINISTIC	PERCENTILE_DISC()	APPROX_PERCENTILE(DETERMINISTIC)
ALL	(Both)	APPROX_PERCENTILE()
ALL DETERMINISTIC	(Both)	APPROX_PERCENTILE(DETERMINISTIC)

APPROX_FOR_AGGREGATION

Value

FALSE	(Default)	(Normal execution)
TRUE	Works as setting:	
	APPROX_FOR_COUNT_DISTINCT	TRUE
	APPROX_FOR_PERCENTILE	ALL

APPROX_FOR_COUNT_DISTINCT not enabled

- With parameter set to false, COUNT(DISTINCT) works as normal
- Takes about two seconds on average

```
alter session set approx_for_count_distinct=false

select count(distinct zip_code) as zip_count from orders_o7
 where sales_date >= DATE '2018-01-01' and sales_date <= DATE '2018-01-07'
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |     8 |       | 11278   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE      |           |     1 |     8 |       |            |          |
|   2 |   VIEW               | VM_NWVW_1 |   344K|  2689K|       | 11278   (2)| 00:00:01 |
|   3 |    HASH GROUP BY     |           |   344K|  5378K|  8248K| 11278   (2)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL| ORDERS_O7 |   350K|  5468K|       |  9426   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

APPROX_FOR_COUNT_DISTINCT enabled

- With parameter = true, COUNT(DISTINCT) rewrites to APPROX_COUNT_DISTINCT
- Takes about one second on average

```
alter session set approx_for_count_distinct=true

select count(distinct zip_code) as zip_count from orders_o7
 where sales_date >= DATE '2018-01-01' and sales_date <= DATE '2018-01-07'
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |     1 |    16 |  9426   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE APPROX|           |     1 |    16 |            |          |
|*  2 |   TABLE ACCESS FULL   | ORDERS_O7 |   350K|  5468K|  9426   (2)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Enabled and Query Rewrite

FAST=TRUE 😎

- COUNT(DISTINCT) => APPROX_COUNT_DISTINCT => Mat.view query
- Takes about 25 milliseconds on average

```
alter session set approx_for_count_distinct=true
alter session set query_rewrite_enabled=true

select count(distinct zip_code) as zip_count from orders_o7
 where sales_date >= DATE '2018-01-01' and sales_date <= DATE '2018-01-07'
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     1 |  3134 |    29   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE APPROX        |              |     1 |  3134 |            |          |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| ORDERS_DAY_MV |     8 | 25072 |    29   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Exactly rather than approximately

Bitmap based count distinct

19c

- New in 19c

- BITMAP_BIT_POSITION
- BITMAP_BUCKET_NUMBER
- BITMAP_CONSTRUCT_AGG
- BITMAP_OR_AGG
- BITMAP_COUNT

- Constructs bitmap BLOBs in buckets in materialized views that can give distinct count.
- (Similar to how APPROX_*_DETAIL/AGG stores aggregate approximations in mviews.)

The bitmaps

19c

- Can't be used on any datatype - input must be integers
- BITMAP_BIT_POSITION maps an input integer to a position in a bitmap, where one bitmap can have up to 32K bits
- If we need > 32K, we need multiple bitmaps (buckets)
- BITMAP_BUCKET_NUMBER maps an input integer to which bitmap(bucket) it belongs to
- BITMAP_BUCKET_NUMBER + BITMAP_BIT_POSITION together identifies unique bucket/position for that input value

Aggregating bitmaps

- BITMAP_CONSTRUCT_AGG aggregates the output of BITMAP_BIT_POSITION for each BITMAP_BUCKET_NUMBER
- Result is 4KB BLOB bitmaps to be stored in mview similar to APPROX_COUNT_DISTINCT_DETAIL
- BITMAP_OR_AGG can be used to aggregate bitmap BLOBs to a higher level similar to APPROX_COUNT_DISTINCT_AGG
- BITMAP_COUNT counts the bits of the bitmap producing usable result similar to TO_APPROX_COUNT_DISTINCT

Materialized view

- Add bucket to group by and select
- Construct bitmap from bit positions – note numeric input

```
create materialized view orders_day_bmv
enable query rewrite
as
select sales_date
     , bitmap_bucket_number(to_number(zip_code)) as bucket#
     , bitmap_construct_agg(
         bitmap_bit_position(to_number(zip_code))
       ) as day_bitmap
  from orders_o7
 group by sales_date, bucket#


create index orders_day_bmv_day_ix on orders_day_bmv (sales_date)
```

Using mview directly

- BITMAP_OR_AGG aggregates buckets across days
- BITMAP_COUNT counts 1 bits in bucket, which then can be summed

```
select sum(zip_count_pr_bucket) as zip_count
  from (
    select bucket#
         , bitmap_count(
              bitmap_or_agg(day_bitmap)
           ) as zip_count_pr_bucket
      from orders_day_bmv
    where sales_date >= DATE '2018-01-01'
      and sales_date <= DATE '2018-01-07'
    group by bucket#
)
```

Query rewrite

- Rewrites count distinct into the query from previous slide

```
alter session set query_rewrite_enabled=true

select count(distinct to_number(zip_code)) as zip_count
  from orders_o7
 where sales_date >= DATE '2018-01-01'
   and sales_date <= DATE '2018-01-07'
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                 | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                      |     1 |    13 |       |  4468   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE                          |                      |     1 |    13 |       |            |          |
|   2 |   VIEW                                   |                      |   557 |  7241 |       |  4468   (1)| 00:00:01 |
|   3 |    HASH GROUP BY                         |                      |   557 |  1091K|  7520K|  4468   (1)| 00:00:01 |
|   4 |     MAT_VIEW ACCESS BY INDEX ROWID BATCHED| ORDERS_DAY_BMV       |  3753 |  7355K|       |  3760   (1)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                    | ORDERS_DAY_BMV_DAY_IX |  3753 |       |       |    11   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------
```

BITMAP versus APPROX

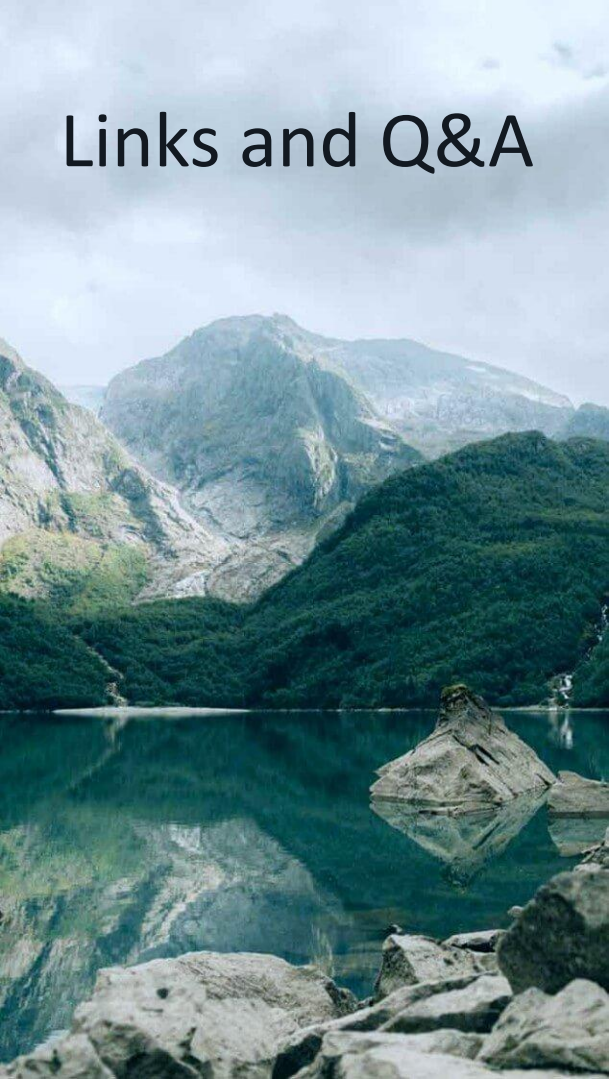
- BITMAP only count distinct, APPROX has other functions too
- BITMAP not useful adhoc but only with mview, APPROX can be helpful even adhoc
- BITMAP takes more BLOBs (ie. space) in mview to be able to be exact, APPROX saves space at cost of being approximate
- BITMAP can query rewrite directly for count distinct, APPROX needs stating explicitly that approx answers are okay
- BITMAP is exact, APPROX is approximate

Read more

Read more

- Math of HyperLogLog algorithm:
  http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf
- Tim Hall (ORACLE BASE) articles:
  https://oracle-base.com/articles/12c/approximate-query-processing-12cr2
  https://oracle-base.com/articles/12c/approx-count-distinct-12cr1
- Christian Antognini (Trivadis) test-case:
  https://antognini.ch/2014/10/the-approx_count_distinct-function-a-test-case/
- Roger MacNicol presentation: "A Faster Way to do Count(Distinct) ?":
  https://ukoug.org/resource/collection/904331A4-A794-4730-AC6B-6AB8B1DD1E36/UKOUG_Count_Distinct.pdf
- Dani Schnider blog on bitmap based count distinct:
  https://danischnider.wordpress.com/2019/04/20/bitmap-based-countdistinct-functions-in-oracle-19c/

Links and Q&A



Bluesky
@kibeha.dk

Blog
https://kibeha.dk

Other links
https://linktr.ee/kibeha

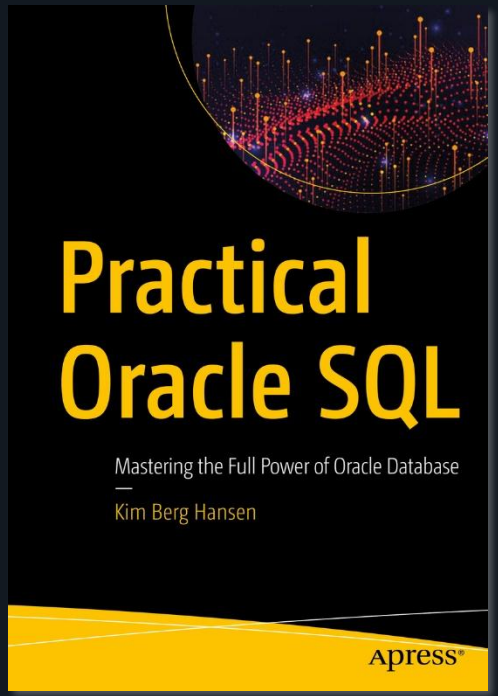
This Presentation
https://bit.ly/OKOracle

CEGAL

pixabay
FREE IMAGES

cegal.com

Author of Practical Oracle SQL



- Not a SQL-101 book
- Not a reference manual replacement
- For developers knowing basic SQL-92 syntax but wanting to advance further
- More elaborate examples relating to daily life as very simple examples are difficult to relate to work
- Useful SQL features that aren't widely used - but should be
- More background in an interview in NoCOUG Journal: http://nocoug.org/Journal/NoCOUG_Journal_202002.pdf#page=4
- The book: https://www.apress.com/gp/book/9781484256169 https://www.amazon.com/Practical-Oracle-SQL-Mastering-Database/dp/1484256166