Managing Statistics of Volatile Tables in Oracle

Iordan K. Iotzov

Senior Database Administrator

News America Marketing (NewsCorp)

iiotzov@newsamerica.com

Blog: http://iiotzov.wordpress.com/

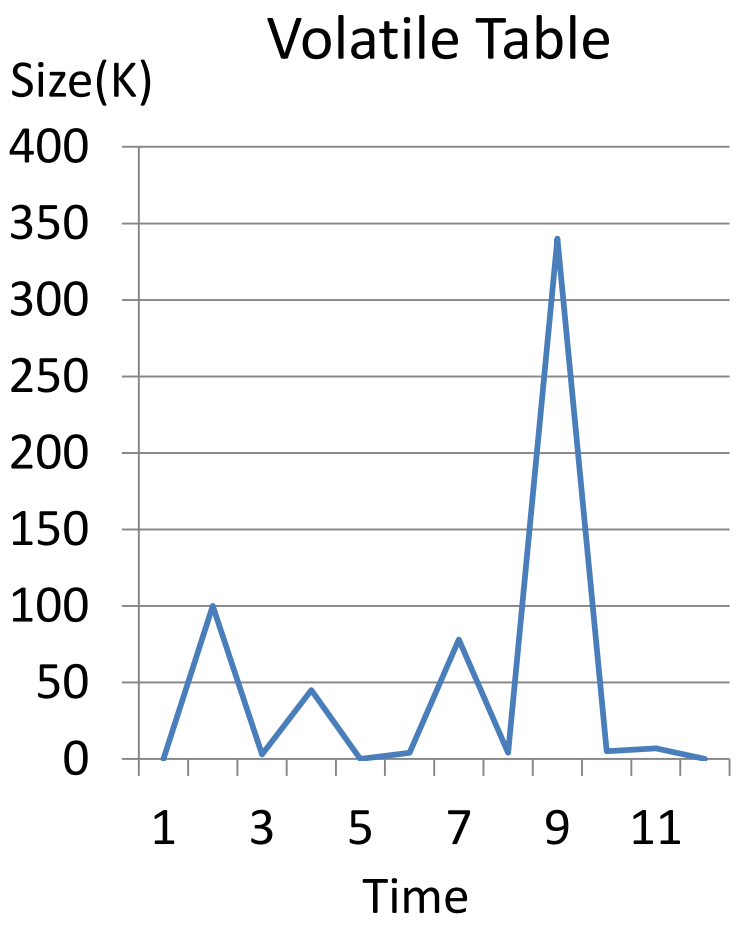
About me

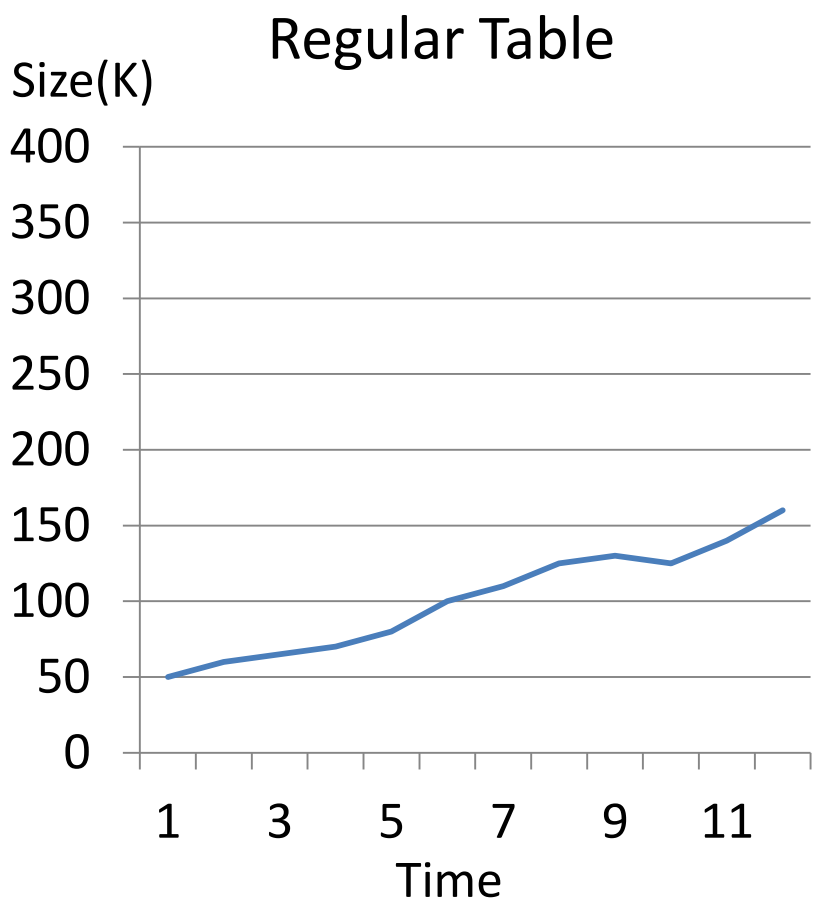
- 10+ years of database administration and development experience
- MS in Computer Science, BS in Electrical Engineering
- Presented at Hotsos, NYOUG and Virta-Thon
- Active blogger and OTN participant
- Senior DBA at News America Marketing (NewsCorp)

Agenda

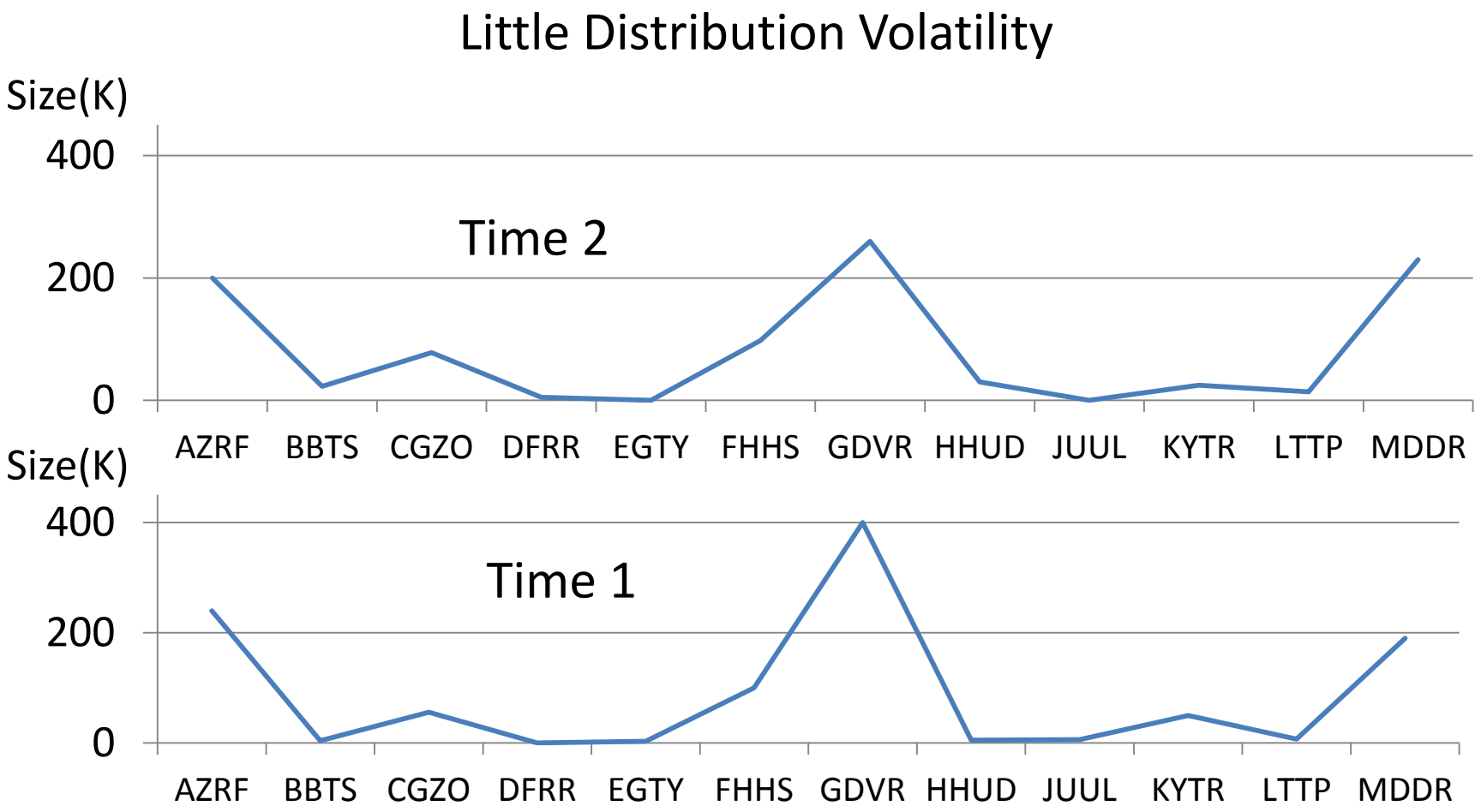
- Definition of volume and distribution volatility
- Reducing volatility
  - tradeoffs
- Dealing with volatility
  - robust execution plans
    - adaptive stats locking
  - follow the change
    - gather stats in places you never thought you could
- Conclusions

Definition of volume volatility

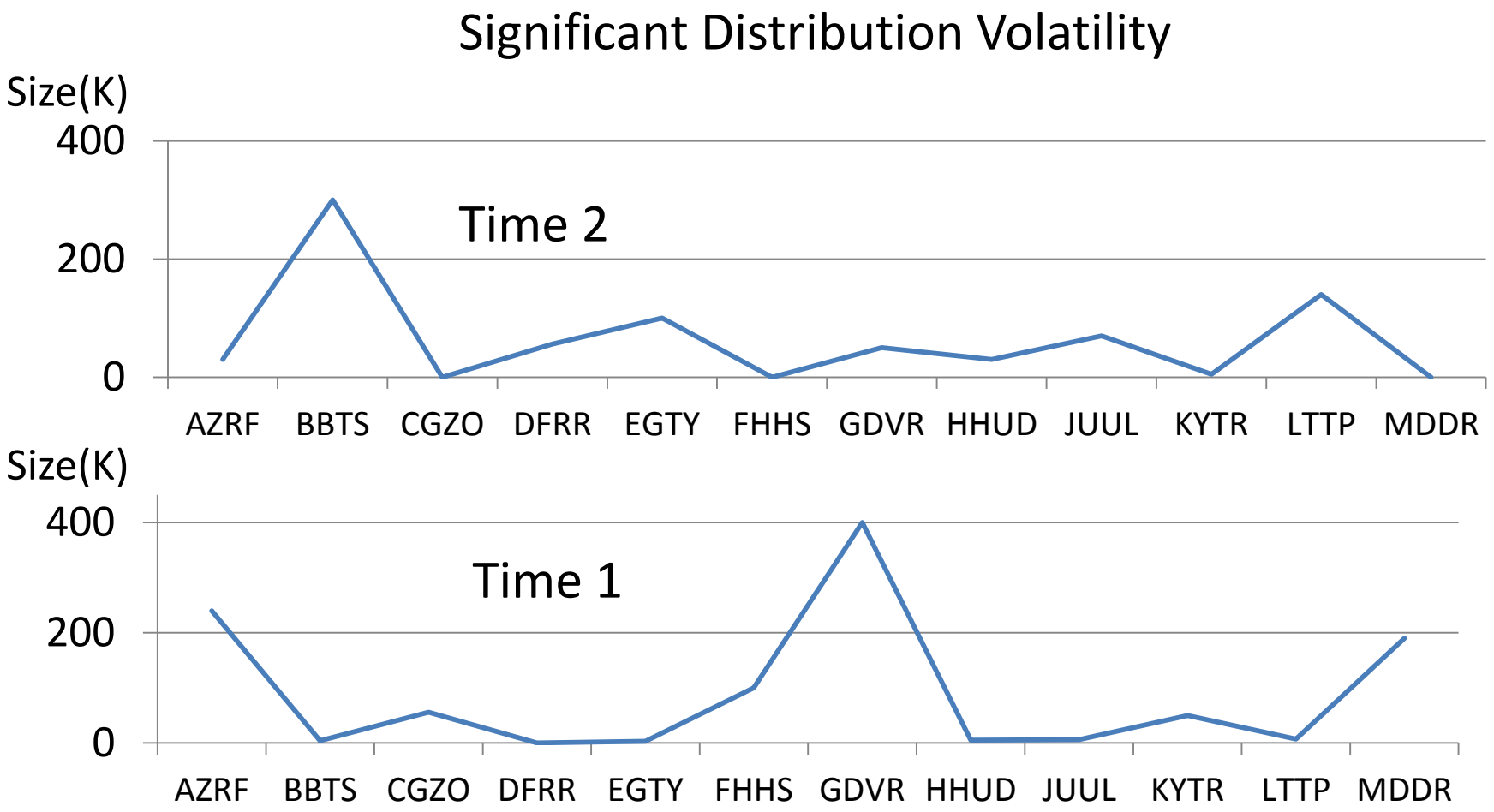




Definition of distribution volatility



Definition of distribution volatility



Reducing volatility

- Proactive
  - Rethink database design
    - does this temporary set have to be stored in the DB?
- Reactive
  - Two-phase removal of data
    - Delete => Update flag (logical removal)
    - Physical removal by a scheduled batch process
    - Addresses volume volatility only

Reducing Volatility

- Pros
  - ✓No need to change select statements
  - ✓Stable execution plans

- Cons
  - ✓Does not help with distribution volatility
  - ✓Limited options for CBO(no FTS)
  - ✓Column statistics represent average
  - ✓Larger footprint

Reducing Volatility

Original

```
table tab
( col1 NUMBER,
…
col10 VARCHAR2)
```

Logical Removal

```
table tab_internal
( col1 NUMBER,
…
col10 VARCHAR2,
deleted VARCHAR2(1)
constraint del check (deleted
in ('Y','N')))

view tab as
select col1, col2, … col10
from tab_internal
where deleted = 'N'
```

Reducing Volatility

Keeping bulk DML operations solution

- Requires changes to the application code

- Pros
  - ✓Ability to achieve high performance by utilizing bulk operations

- Cons
  - ✓Have to change the code

Reducing Volatility

Keeping bulk DML operations solution

```
insert into tab
(col1,..col10)
values
(col1,..col10)
```

```
insert into tab_internal
(col1,..col10,deleted)
values
col1,..col10,'N')
```

```
delete tab
where  col1=
..
```

```
update into tab_internal
set deleted = 'Y'
where deleted = 'N'
and col1= ..
```

Reducing Volatility

Trigger-based solution

-Does not requires changes to the application code

✓Pros

- ✓No need to change the application code

✓Cons

- ✓Some DML performance limited by row-by-row processing

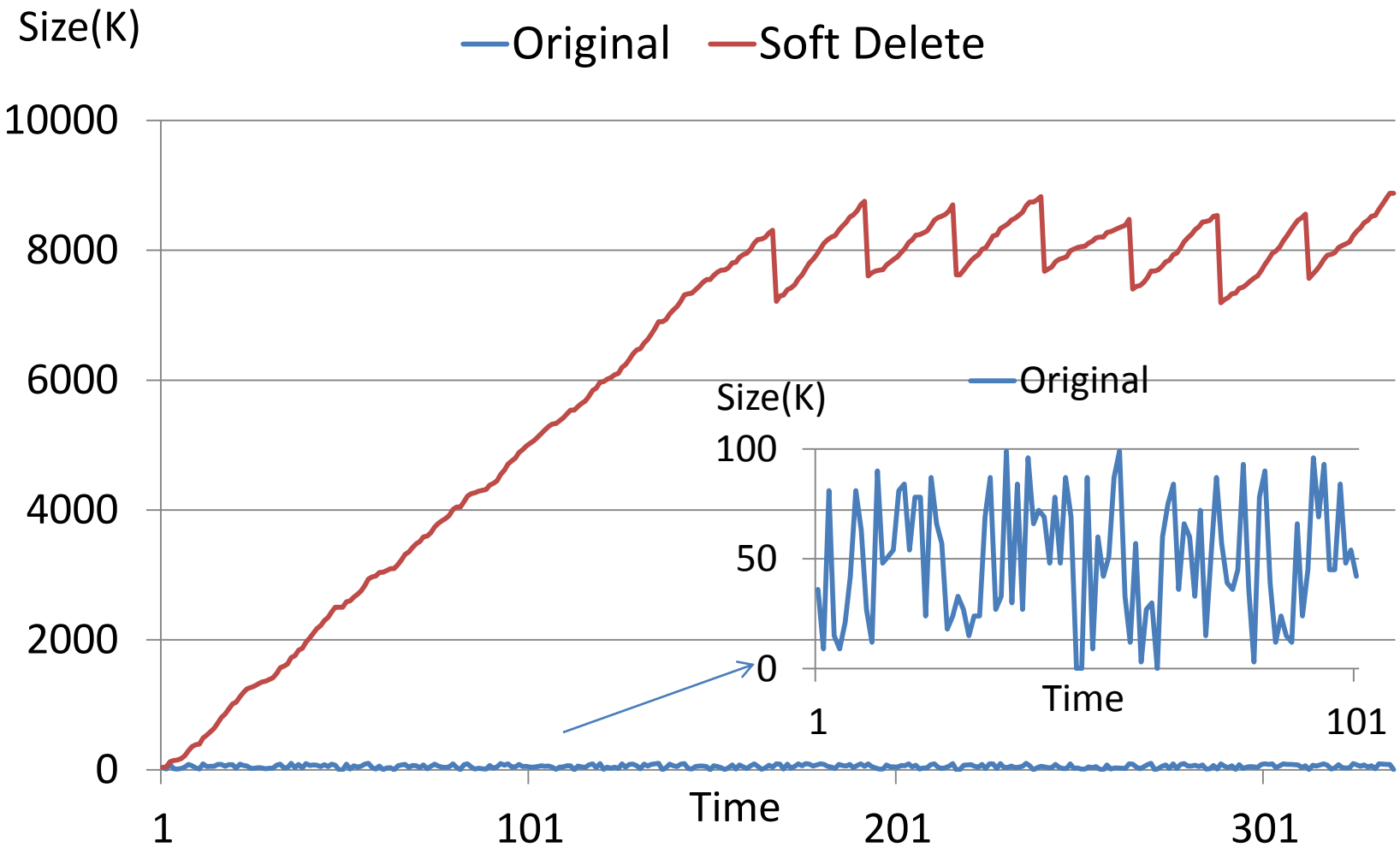
Reducing Volatility

Trigger-based solution

```
create or replace trigger v_t_tr instead
of insert on tab
 begin
   insert into tab_internal (col1,..col10,deleted)
   values (col1,..col10, 'N'  );
 end;

create or replace trigger v_t_del
 instead of delete on tab  referencing new
as new old as old
   begin
     update tab_internal       set deleted = 'Y'
    where col1  = :old.col1
    and col2 = …
end;
```

Reducing Volatility



Dealing with volatility

Robust execution plans

- What is robust?
  - capable of performing without failure under a wide range of conditions (Merriam Webster)

- Paradigm shift
  - Looking for optimal is no longer the goal
  - Searching for "reasonable" performance, execution time within certain limits

Dealing with volatility

Robust execution plans

Statistics are used for determining:

- Join Method
  - Hash Join vs Nested Loops
- Join Order
  - The sequence the tables would be joined

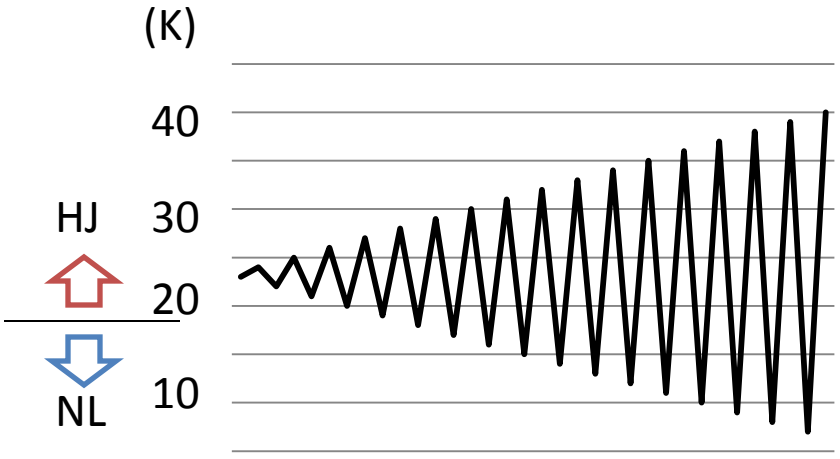
Dealing with volatility

Robust execution plans

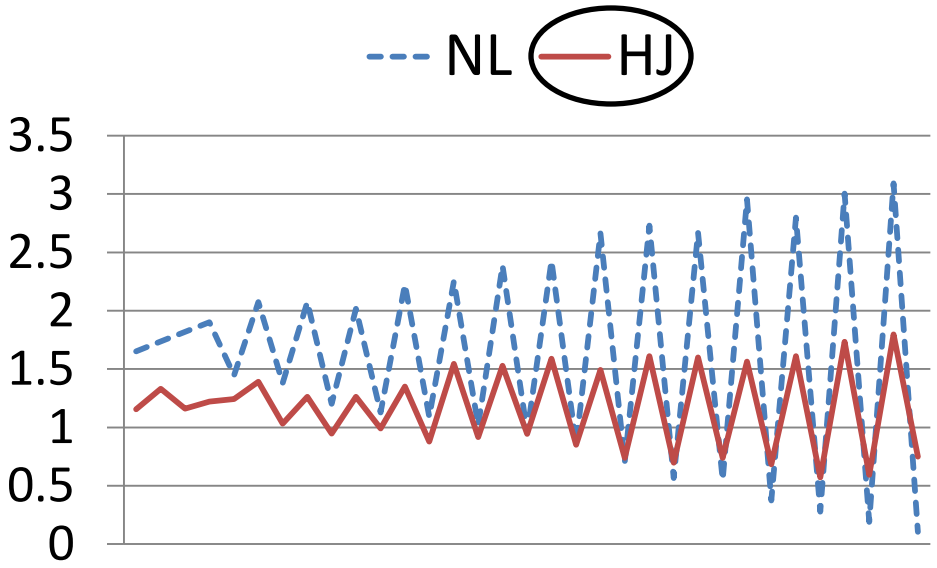
Join Method

Variance Reduction => Robust System (Taguchi)

Size of volatile table



Execution time
for join involving volatile table



Dealing with volatility

Robust execution plans

Join Method

Oracle 12c – Adaptive Execution Plans

Join Method selected at run time and based on the actual row count

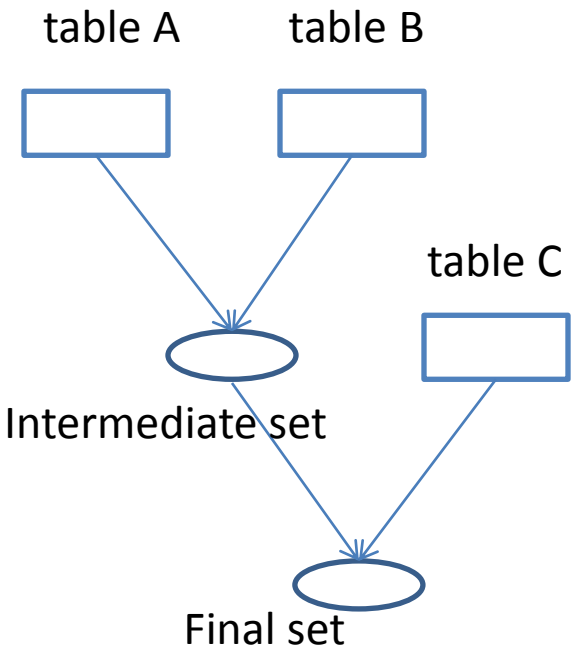
- Able to mitigate some of the problems related to cardinality miscalculations, including those caused by data volatility.

- Run-time decision NL/HJ done only with the first execution. Adjust expectations when reusing SQL.

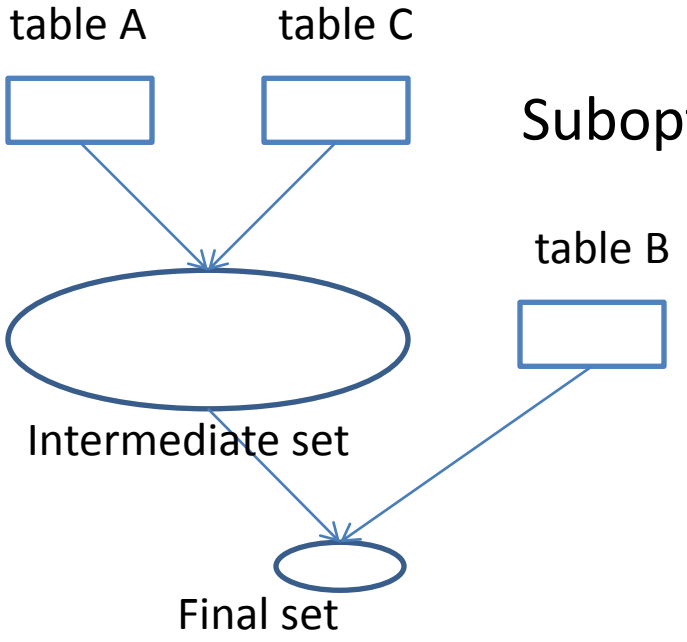
Dealing with volatility

Robust execution plans

Join Order



Good



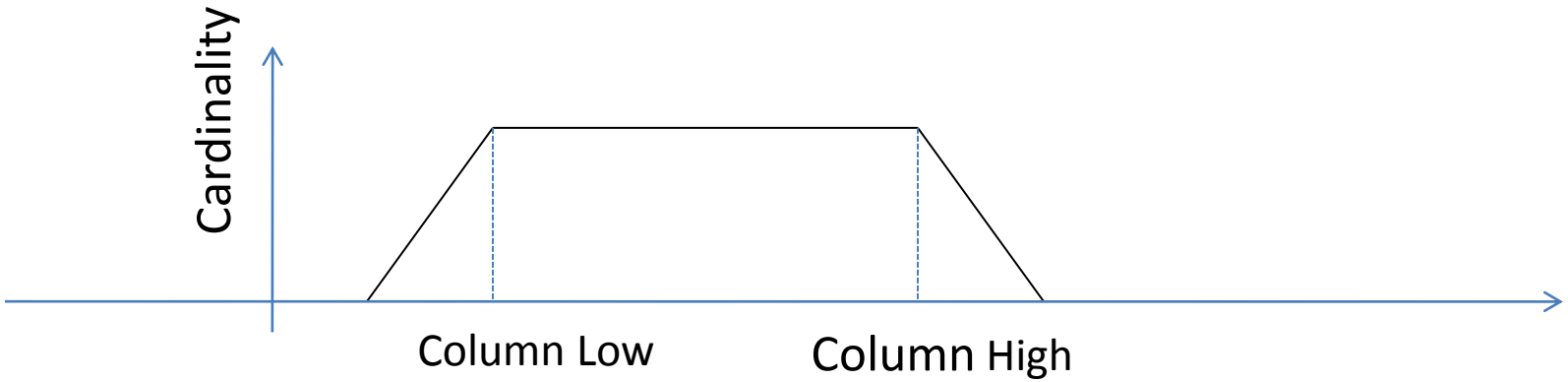
Suboptimal

Suboptimal join order frequently results in huge intermediate sets.

Dealing with volatility

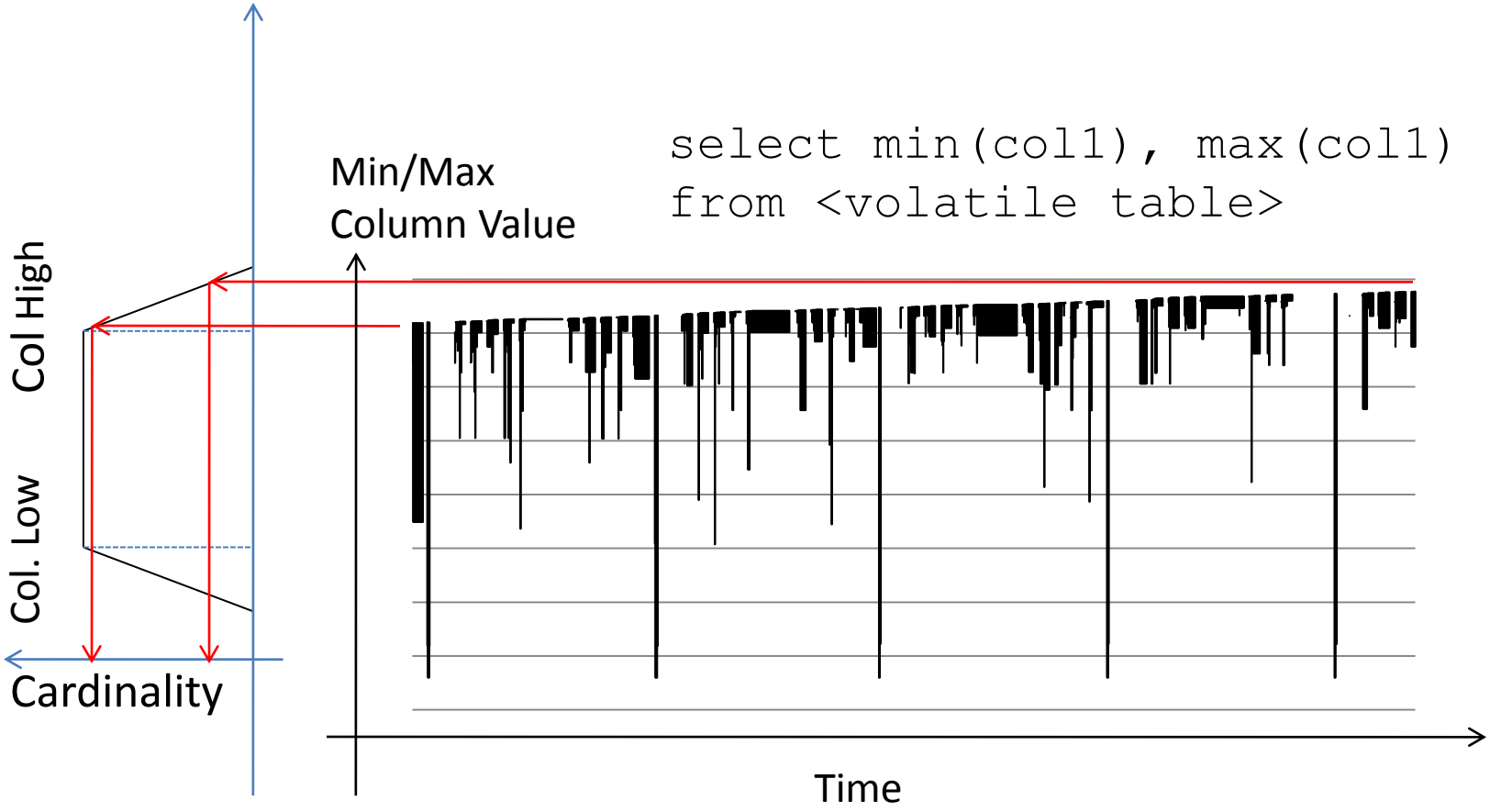
Robust execution plans

- Locking statistics
  - Best Practices for Automatic Statistics Collection [ID 377152.1]
- (Long-term) Issues with locking statistics
  - How to know the maximum size in advance?
  - Data changes…



Dealing with volatility

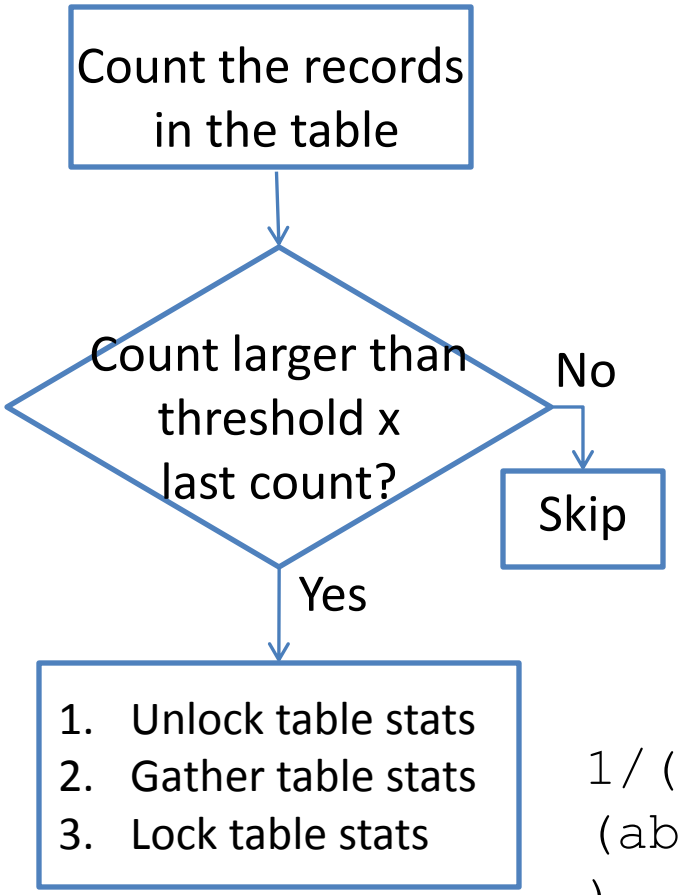
Robust execution plans

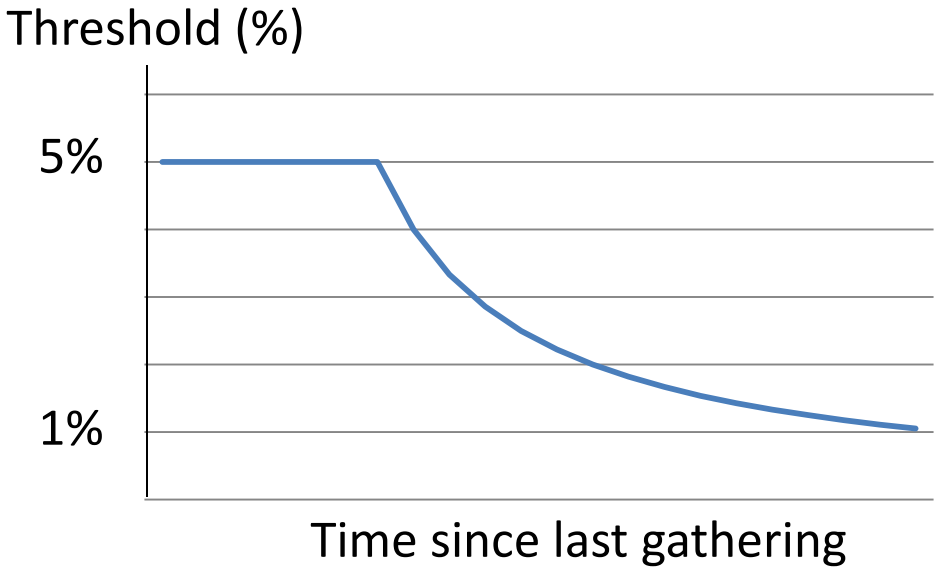


Dealing with volatility

Robust execution plans

Adaptive Stats Locking



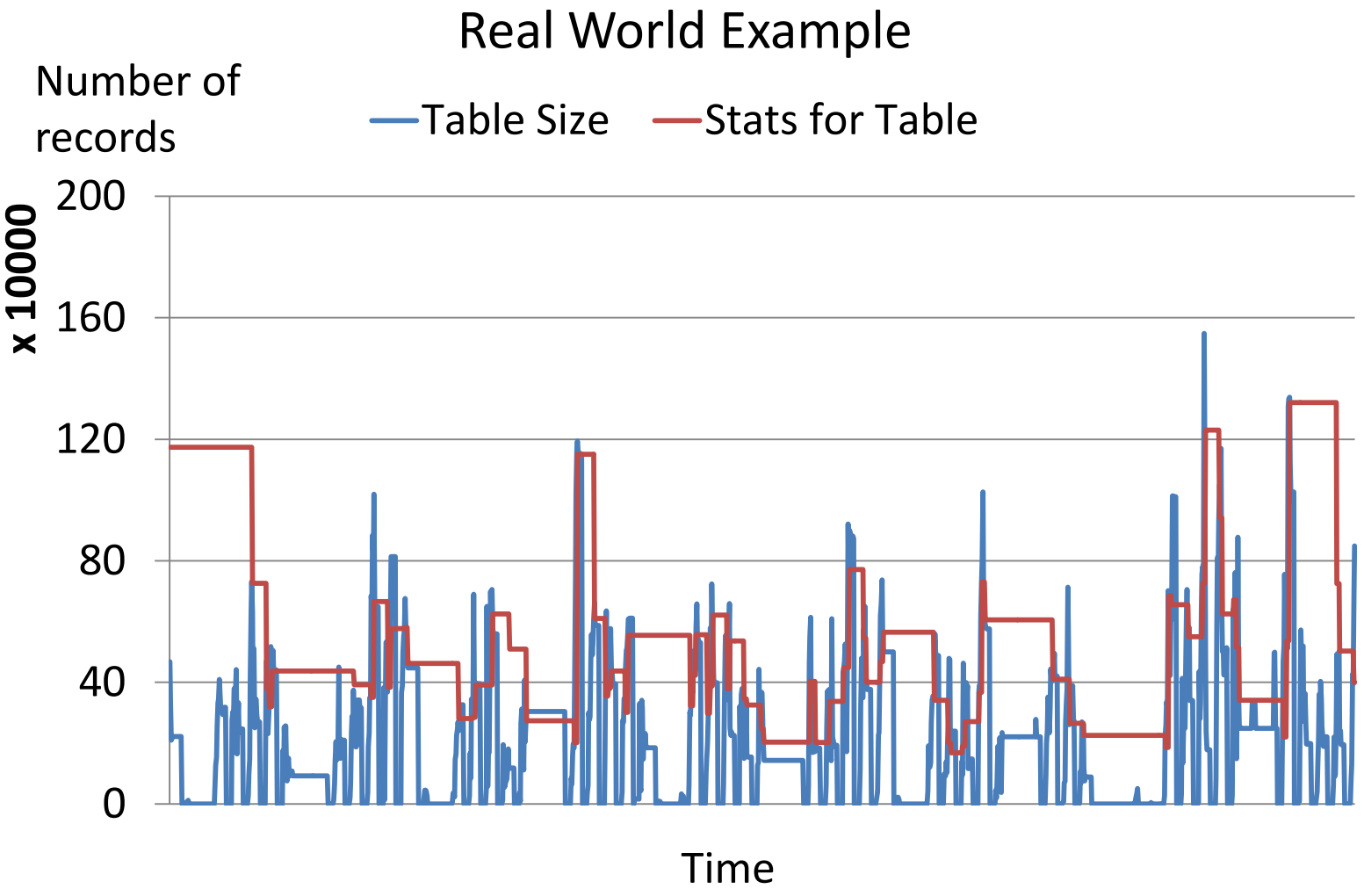


Example:

```
1/(5*
(abs(i.days_since_last_analyze)-3)
)
```

Dealing with volatility

Robust execution plans



Dealing with volatility

Robust execution plans

Implementation highlights: The table size can change significantly at any time! – Oracle 10g

Step	SQL
Backup existing statistics	`truncate table prev_stats ;` `execute DBMS_STATS.EXPORT_TABLE_STATS (<DB_USER>,<TAB>, stattab => 'prev_stats');`
Gather statistics	`exec dbms_stats.gather_table_stats(<DB_USER>,<TAB>)`
Verify that the gathered stats are what was expected?	`select num_rows from dba_tables where owner = <DB_USER> and table_name = <TAB>`
If not – restore statistics from backup	`exec DBMS_STATS.IMPORT_TABLE_STATS (<DB_USER>,<TAB>, stattab => 'prev_stats');`

Dealing with volatility

Robust execution plans

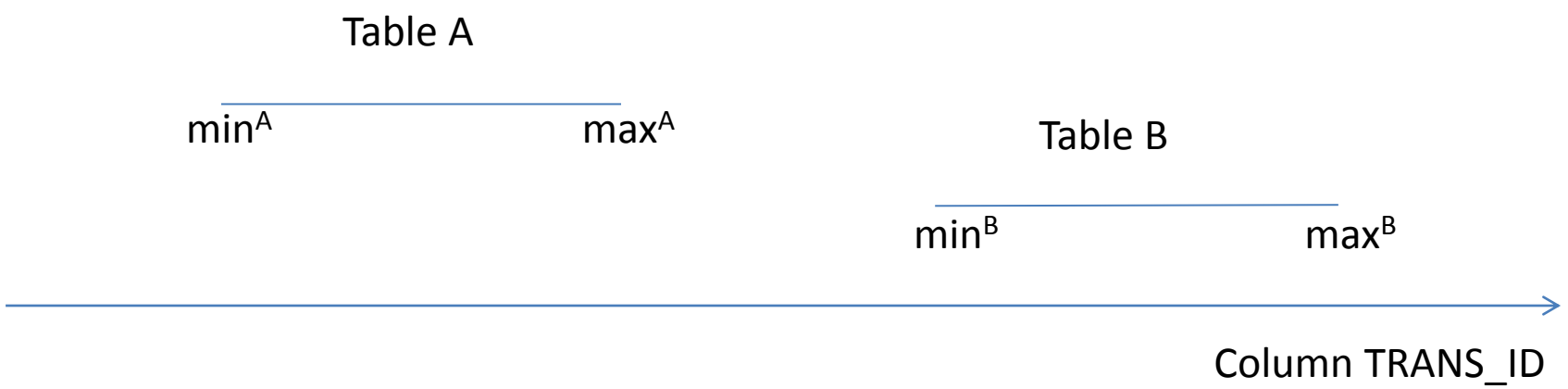
Implementation highlights: The table size can change significantly at any time! – Oracle 11g

Step	SQL
Keep new stats in pending state	`exec dbms_stats.set_table_prefs((<DB_USER>, <TAB>,'PUBLISH', 'false');`
Gather statistics	`exec dbms_stats.gather_table_stats(<DB_USER>,<TAB>)`
Verify that the gathered stats are what was expected?	`select num_rows from dba_tab_pending_stats where owner = <DB_USER> and table_name = <TAB>`
If yes – publish the statistics	`exec dbms_stats.publish_pending_stats(<DB_USER>,<TAB>);`

Dealing with volatility

Robust execution plans

Implementation highlights: Unintended side affects when manually managing statistics



Estimated cardinality of

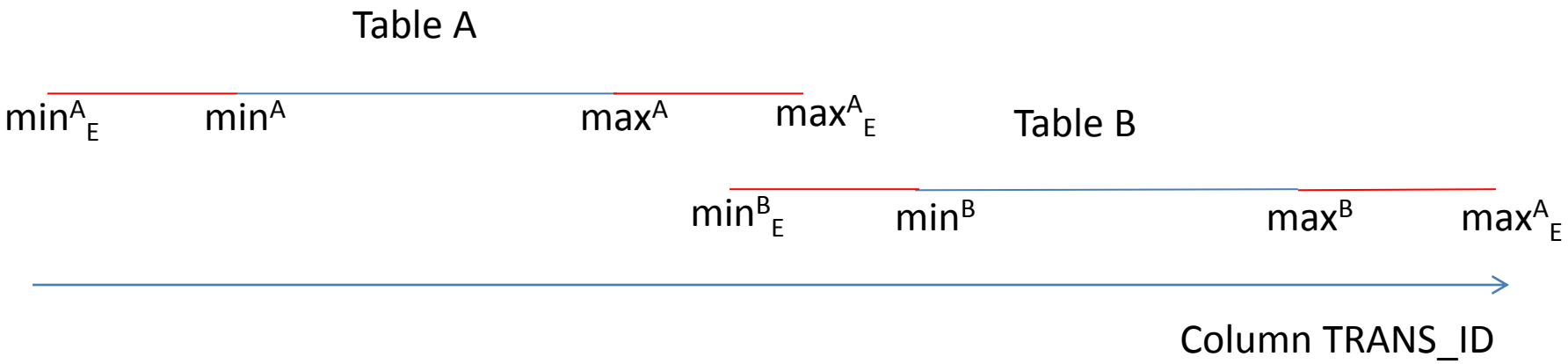
```
SELECT *
FROM A , B
WHERE A.TRANS_ID = B.TRANS_ID
```

is 1!!!

Dealing with volatility

Robust execution plans

Implementation highlights: Unintended side affects when manually managing statistics



Artificially extending min/max ranges:

- Improve join selectivity
- Deteriorate single table selectivity

Sample range extending techniques:

- Number: $min_{E =}$ min /2 ; $max_{E =}$ max * 2
- Date: $min_{E =}$ min - 365 days ; $max_{E =}$ max + 365 days

Dealing with volatility

Follow the change

- Goal:
  - ✓Statistics should precisely represent the underlying data at any point in time
- Implementation:
  - ✓Dynamic sampling
  - ✓Explicit stats gathering after significant data change

Dealing with volatility

Follow the change

Dynamic sampling

Hard-parse triggered on-the-fly statistics gathering where the resulting statistics are used for the generation of a single SQL plan only

- Pros
  - ✓Easy to set up (for basic level dynamic sampling)
  - ✓No need for functional testing
- Cons
  - ✓Needs hard parsing (manageable)
  - ✓Suboptimal in load once, select many times scenarios

Dealing with volatility

Follow the change

Explicit statistics gathering after
every significant data change (DBMS_STATS package)

- Pros
  - ✓Suitable for all scenarios
- Cons
  - ✓Needs application code changes (possibly numerous)
  - ✓Issues an implicit COMMIT

Dealing with volatility

Follow the change

Explicit statistics gathering after
every significant data change (DBMS_STATS package)

Oracle 12c introduced
Session-Private Statistics for Global Temporary Tables

- GLOBAL_TEMP_TABLE_STATS table preference allows gathering session level statistics
- Greatly improves the ability to handle volatile tables in multi-user environments

Dealing with volatility

Follow the change

Required testing after a change

Non-functional testing	Functional testing
✓Create/drop indexes	✓Change/Create application SQL
✓Change init.ora parameters (most cases)	✓Materialized views (most cases)
✓Hints	✓Custom de-normalizations/aggregations
✓Dynamic Sampling	✓COMMIT
✓JUST_STATS (no COMMIT!) ⇦	✓DBMS_STATS (implicit COMMIT)

Dealing with volatility
Follow the change

When to a COMMIT a transaction?

- … data integrity is *the driving force* behind the size of your transaction (Tom Kyte)
- transaction should be committed when it must and never before (Tom Kyte)
- ~~to perform non-functional operations such statistics gathering~~

Then why DBMS_STATS issues a COMMIT?

- To shorten the duration of DDL locks(Tom Kyte) /valid point/
- Because we should gather the statistics only after the application change has been successful and committed (MOS Analyst)

Dealing with volatility

Follow the change

JUST_STATS package functional overview

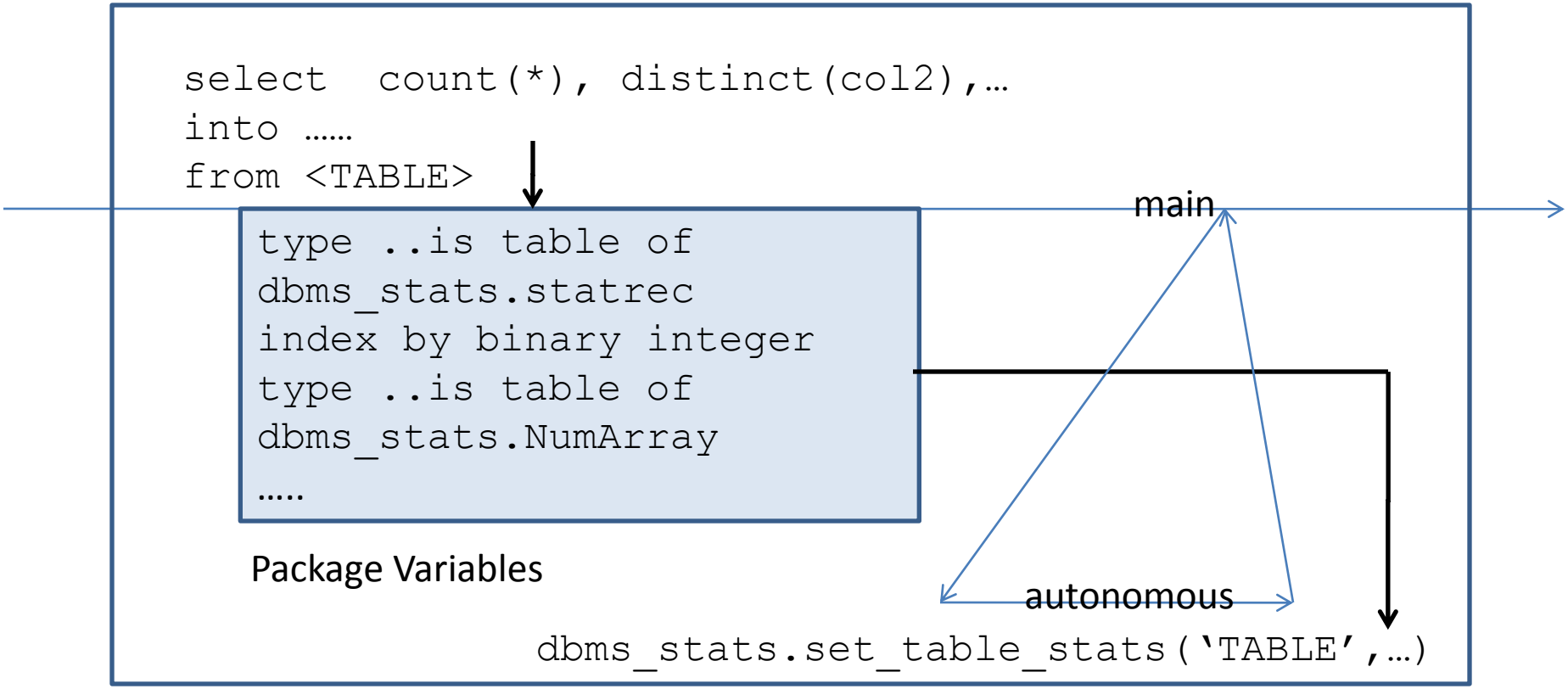
- Functionally equivalent to DBMS_STATS, but with limited features
  - ✓ GATHER_TABLE_STATS
  - ✓ GATHER_INDEX_STATS
  - ✓ Limited Histograms
  - ✓ Most data types
  - ✓ No “Auto” options

- Does not issue a COMMIT
  - ✓ Stats do not rollback after the transaction is rolled back
  - ✓ Other sessions can sees the new stats without seeing the data behind those stats

Dealing with volatility

Follow the change

Inside JUST_STATS package



Dealing with volatility

Follow the change

Uses for JUST_STATS package

- Wherever DBMS_STATS should be used, but COMMIT is not desired – batch processes
- Post statement table triggers
  – a great place to gather statistics!

Example:

```
create or replace trigger cust_stats
after insert or delete or update on <TAB>
begin
   just_stats.gather_table_stats('<USER>','<TAB>');
end;
```

Dealing with volatility

Follow the change

Oracle 12c alternative

Online Statistics Gathering for Bulk Loads

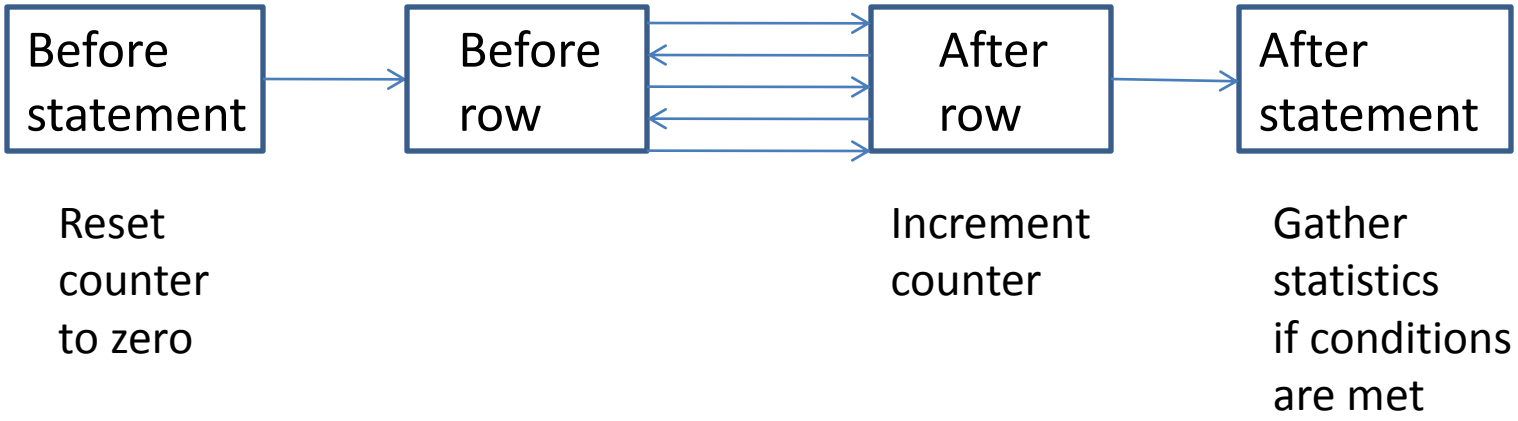
- After CTAS or direct path INSERT INTO .. SELECT
- Does not collect index statistics and histograms
- Check “Notes” column in DBA_TAB_COL_STATISTICS to confirm

Dealing with volatility

Follow the change

Customizations for stats gathering in triggers

Frequently, it is not wise to gather stats after every DML…



Dealing with volatility

Follow the change

Customizations for stats gathering in triggers

Auxiliary package

```
create package stats_aux as
   cnt number;
end stats_aux;
```

Before statement

```
create or replace
trigger stats_cnt_reset
before insert or delete
or update on <TABLE>
begin
   stats_aux.cnt:=0;
end;
```

After row

```
create or replace
trigger stats_cnt_increment
before insert or delete
or update on <TABLE>
for each row
begin
  stats_aux.cnt:=stats_aux.cnt+1;
end;
```

Dealing with volatility

Follow the change

Customizations for stats gathering in triggers

Gather statistics only after a single DML modifies at least 10% of the records

```
create or replace trigger cond_stats_gather
after insert or delete or update on <TAB>
declare
dd_cnt number;
begin
  select num_rows
  into dd_cnt
  from user_tables
  where table_name = '<TAB>';
  if stats_aux.cnt*10 > dd_cnt then
       just_stats.gather_table_stats('<USR>','<TAB>');
 end if;
end;
```

Dealing with volatility

Follow the change

More about JUST_STATS package:

Free to download and use (I wrote it!)

No support

No liability

Would like to make JUST_STATS functionality mainstream?

Support "resolution" of
Oracle bug# 12897196 !

Thank you