Bulk Processing Data with SQL and PL/SQL

DOAG
15th November 2016

Martin Widlake
Database Performance, Architecture & Training
Ora600 Limited

mwidlake@ORA600.org.uk
http://mwidlake.wordpress.com/
Oh, and that twitter thing - @MDwidlake

Abstract

What is the fastest *and safest* way to process bulk data? SQL or PL/SQL? What options do you have and in which situations are they best? In this presentation I will review several ways of processing large volumes of data and what the advantages and disadvantages of them are.

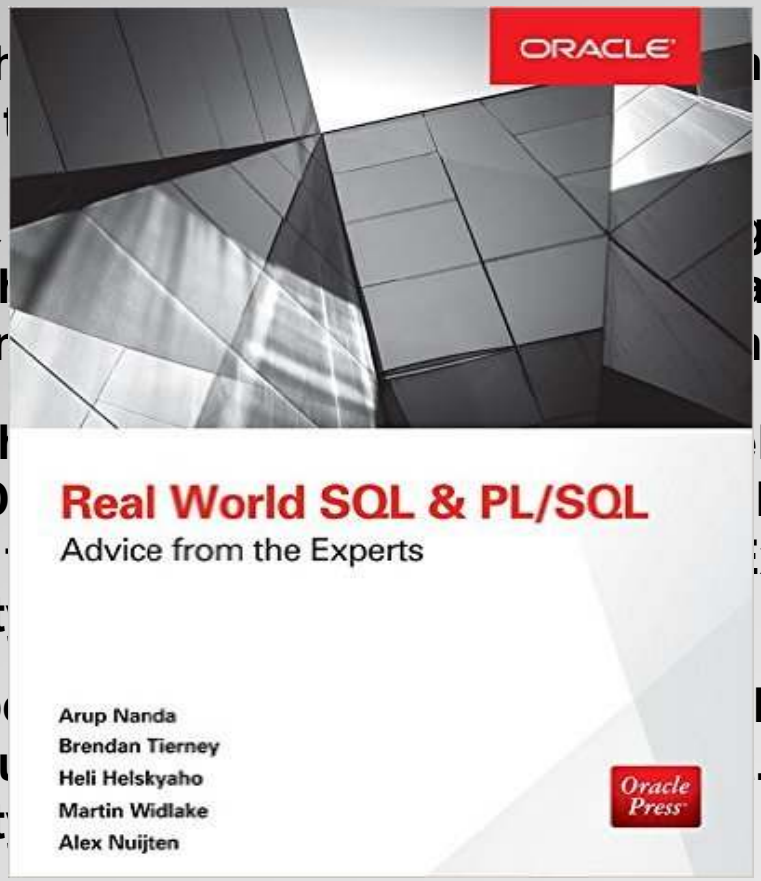
This is not a presentation about code and syntax – you will find dozens of examples on the Web about bulk PL/SQL processing and MERGE statements. It is about the concepts and the considerations. Learning syntax is easy, deciding on the best, most pragmatic way to process your data takes a bit more thought.

Suitable for beginners and intermediate.

Who am I and why am I doing this Talk?

- I've been working with[illegible]alf my life in fact. Duration is no guarant[illegible]
- I've designed, built &[illegible]g life, moving huge data volumes ("[illegible]and out of them. Size of your VLDB is r[illegible].
- Like many old oracle [illegible]ll into using PL/SQL, went to the D[illegible]h-veloper and cycled between them [illegible]xperience is no guarantee of capabilit[illegible]
- I like cats, genetics, b[illegible]hich I do a lot more of now) and User Grou[illegible]. Presenting is no guarantee of capabilit[illegible]
- I've helped write a book.

  Writing books is no guarantee of capability though.



SQL Processes Data Faster than PL/SQL

- **SQL is a set-processing language, it is designed to take arrays of data and transform them.**
- **PL/SQL cannot alter data in the database. It has to use SQL to do so.**
- **The question is more, *is it more efficient to control SQL processing of data via PL/SQL than not to*.**
- **If your processing is simple and the volumes are not massive, SQL will win over anything – (probably)**
- **However, SQL is a Single Command language. If you *cannot afford* to do the data processing in one command, you have a problem to solve.**
- **PL/SQL is the ideal language for controlling bulk data processing in the Oracle database.**

Issues with Straight SQL

- If the statement fails, all of it rolls back. All or Nothing.
- If the statement runs for too long you may get snapshot too old errors.
- You data must be static or you have to capture any data since the command started.
- True RI cannot be enforced within a single SQL statement.
- The data volume may just be too large, or at least too large to process efficiently - especially if hashing or sorting spills over to disc. Sometimes it is simply too large.

Issues with PL/SQL

- **PL/SQL is slow.** {I will show this is not really true}
- **By the very fact you are using PL/SQL you are breaking up the task into parts, so you need control.**
- **It is more complex to develop a PL/SQL solution than to use a SQL-only solution.**
- **You are writing a program, so it *needs to be tested*. Every feature you add (re-start, logging) needs to be tested.**
- **Either you may not know PL/SQL or there may be no one else who knows PL/SQL (e.g. everyone is a JAVA Head).**
  {Or you may be a very old PL/SQL programmer}

Straight SQL Insert Into Select…

```
Insert into activity2
select
  ID
, COUNTRY_ID
, R2
, NUM1 +1
, NUM2
, VC1
, VC_R1
, V_PAD
from activity
where id between 1000000 and 1999999  -- 1 million records
```

Takes 1.94 seconds
Adding indexes would make a big difference

Extra options are insert /*+ append */ and making the segment (or tablespace) nologging.

Nologging is dangerous, I generally avoid it due to the impact on backups (and standby) unless I have specific requirement.

SQL MERGE

- The SQL MERGE statement allows you to detect if a record already exists and, if it does, update it. Otherwise insert it.

```
MERGE INTO bonuses D
   USING (SELECT employee_id, salary, department_id   FROM employees
          WHERE  department_id = 80) S
   ON (D.employee_id = S.employee_id)
   WHEN MATCHED THEN UPDATE SET D.bonus = D.bonus + S.salary*.01
     DELETE WHERE (S.salary > 8000)
   WHEN NOT MATCHED THEN INSERT (D.employee_id, D.bonus)
     VALUES (S.employee_id, S.salary*.01)
     WHERE (S.salary <= 8000);
```

- Note the multiple steps in the update.
- Note the WHERE clauses on the DELETE and INSERT

Analytical Functions

- **If you do not currently use analytical functions, learn about them. You may be surprised what you can do with "just" SQL.**

Things that Slow Down SQL

- **Indexes (remove, partition, direct insert)**
- **Referential Integrity (enable novalidate)**
- **Sequences (big caches, use returning)**
- **PL/SQL Functions**
- **DML Triggers and RLS (suspend, exempt account or live with it.)**

PL/SQL Row by Row

- Row by Row = Slow by Slow.
- PL/SQL gained a reputation for being slow due to people using it to… get-a-record : modify-record : insert-record
- Really early version of PL/SQL you had to do this. But this has not been true for years (15+? 20+).
- This is not specific to PL/SQL. If you use any language to grab one row at a time and process it, that is slow.

PL/SQL Row-by-Row Slow-by-Slow

```
declare
cursor get_activity is
  select *
  from activity
  where id between 1000000 and 1999999; -- 1 million records
--
activity_rec get_activity%rowtype;
ts1          timestamp;
begin
  ts1:=systimestamp;
  dbms_output.put_Line ('started at '||ts1);
  open get_activity;
  loop
    fetch get_activity into activity_rec;
    activity_rec.num1    := activity_rec.num1+1;
--
    insert into activity2 values activity_rec;
    exit when get_activity%notfound;
  end loop;
  close get_activity;
  dbms_output.put_Line ('fin at '||systimestamp);
  commit;
end;
```

Takes 53.959 seconds

Straight SQL Insert Into Select…

```
Insert into activity2
select
  ID
, COUNTRY_ID
, R2
, NUM1 +1
, NUM2
, VC1
, VC_R1
, V_PAD
from activity
where id between 1000000 and 1999999  -- 1 million records
```

Takes 1.94 seconds

Extra options are insert /*+ append */ and making the segment (or tablespace) nologging.

Nologging is dangerous, I generally avoid it due to the impact on backups (and standby) unless I have specific requirement.

PL/SQL Batch Processing

```
declare
cursor get_activity is
  select *
  from activity
  where id between 1000000 and 1999999; -- 1 million records
type activity_array is table of get_activity%rowtype;
la_activity_rec activity_array :=activity_array();
ts1           timestamp;
begin
  ts1:=systimestamp;
  dbms_output.put_Line ('started at '||ts1);
  open get_activity;
  loop
    fetch get_activity bulk collect
    into la_activity_rec  limit 500;
    forall i in indices of la_activity_rec
    insert into activity2 values la_activity_rec(i);
    exit when get_activity%notfound;
  end loop;
  close get_activity;
  dbms_output.put_Line ('fin at '||systimestamp);
  commit;
end;
```

Takes 3.05 seconds

PL/SQL Row-by-Row Slow-by-Slow

```
declare
cursor get_activity is
  select *
  from activity
  where id between 1000000 and 1999999; -- 1 million records
--
activity_rec get_activity%rowtype;
ts1          timestamp;
begin
  ts1:=systimestamp;
  dbms_output.put_Line ('started at '||ts1);
  open get_activity;
  loop
    fetch get_activity into activity_rec;
    activity_rec.num1    := activity_rec.num1+1;
--
    insert into activity2 values activity_rec;
    exit when get_activity%notfound;
  end loop;
  close get_activity;
  dbms_output.put_Line ('fin at '||systimestamp);
  commit;
end;
```

Takes 53.959 seconds

Bulk Collect

- It is really simple to convert an existing SQL statement or a row-by-row, slow-by-slow SQL construct in PL/SQL into a BULK COLLECT version,
- Be mindful of the array size you use. Always use LIMIT and keep it to a few thousand rows.
- Setting the LIMIT too high will uses large amounts of PGA and if several sessions do this you could crash your machine.
- Most of the performance gain comes once your fetch array is a few hundred rows, about 1 or 2 blocks worth of table data.

Impact of Limit

- **Large values of LIMIT use a lot of memory and there is *no need for it.***
- **Below are the run times of my test code with different LIMIT settings.**

LIMIT	Duration (seconds)
No Bulk	53.96
5	47.40
10	27.88
50	*17.02*
100	*03.75*
250	*03.17*
500	03.05
1000	03.19
Straight SQL	01.94

Handling Exceptions

- A major issue with bulk data processing is errors or exceptions. There are three general principles:

Principle	Pros	Cons
Stop and lose everything.	Simple to Implement	Can lose a lot of work done, need to manually sort out and restart.
Stop, preserving most work and allowing restart	Allows piece-wise load, you can control the size of chunks you process, easy to continue (or pause)	Complex to implement and test. Especially test. There is a lot of testing, especially if you are going to hand this over
Record exceptions and continue	Easy to implement, allows load to continue.	Easy to ignore the level of exception and in fact to ignore the exceptions totally. {Never hand this over without a lot of explaining this point}

PL/SQL FORALL Exception Handling

```
  loop
    fetch get_activity bulk collect
    into la_activity_rec  limit 500;
--
      forall i in indices of la_activity_rec
      insert into activity2 values la_activity_rec(i);
      exit when get_activity%notfound;
  end loop;
  close get_activity;
```

PL/SQL FORALL Exception Handling

```
loop
  fetch get_activity bulk collect
  into la_activity_rec  limit 500;
  begin
    forall i in indices of la_activity_rec save exceptions
    insert into activity3
    values la_activity_rec(i);
  exception
    when others then
      if sqlcode=-24381 then
        for errno in 1..sql%bulk_exceptions.count
        loop
          dbms_output.put_line('Error in record '
                               ||sql%bulk_exceptions(errno).error_index||
                               ': Ora-'||sql%bulk_exceptions(errno).error_code);
        end loop;
      else
        raise;
      end if;
  end;
  exit when get_activity%notfound;
end loop;
close get_activity;
```

PL/SQL FORALL Exception Handling

```
started at 18-NOV-15 15.42.15.226000
Error in record 124: Ora-1438
Error in record 232: Ora-1438
finished at 18-NOV-15 15.42.40.4310 taking 00:25.2050
```

- In reality you would probably use an autonomous transaction to insert details into an error table.
- You would issue either an error to your run log, a dbms_output line or both.
- I would add a check for the number of errors and, above a certain level, abort the code *anyway*.
- Did I mention the major drawback of people ignoring the errors?

SQL "Exception Handling"

- The DBMS_ERRLOG package has been available since 10G and can be used to log errors occurring during DML and allow it to occur.
- Create an error log table based on the target table using DBMS_ERRLOG.CREATE_ERROR_LOG
- You can ignore unsupported columns (eg LOBS).
- When you run your DML, state LOG ERRORS and optionally give a REJECT LIMIT. I *strongly advise* you use a sensibly small reject limit.
- The DML will run and give *no indication* of any errors so long as they are below reject limit. So you need a control process to test for that.

SQL "Exception Handling"

```
insert into activity3 select * from activity
where id between 1000000 and 1999999;

insert into activity3 select * from activity
                              *
ERROR at line 1:
ORA-01438: value larger than specified precision allowed for this column


exec dbms_errlog.create_error_log ('ACTIVITY3','ERR$_ACTIVITY3')

insert into activity3 select * from activity
where id between 1000000 and 1999999
log errors
reject limit 1000;
999998 rows created.

select count(*) from ERR$_ACTIVITY3;

  COUNT(*)
----------
         2
```

SQL "Exception Handling"

```
ORA_ERR_NUMBER$
---------------
ORA_ERR_MESG$
--------------------------------------------------------------------------------
ORA_ERR_RO OR
---------- --
ORA_ERR_TAG$
--------------------------------------------------------------------------------
ID      COUNTRY_ID R2    NUM1  NUM2  VC1        VC_R1      V_PAD
------- ---------- ----- ----- ----- ---------- ---------- ----------
           1438
ORA-01438: value larger than specified precision allowed for this column
           I

1123123 11         8     1124  36    UYYYYYYYYY UYCDOXIKSI AAAAAAAAAA
                                                SAIXXHDVWX
           1438
ORA-01438: value larger than specified precision allowed for this column
           I

1231231 13         4     1232  44    IHHHHHHHHH BYORLCRSHM AAAAAAAAAA
                                                OTLGCMAFSA
```

Controlling The Process

- **A key factor with Bulk Processing is whether this is a single shot activity or a constantly running process.**
- **Historical data take-on is usually a single shot process. You may want to have a semi-manual process.**
- **Regular data loads need a more rigorous control and monitoring structure.**
- **Both benefit from being *instrumented*, I would go as far as saying it is essential for constantly running process.**
- **Resist "demands" to make bulk data load near-real-time. Or two way.**

PL/SQL Control Mechanism

- Writing the bulk processing code or SQL is actually the easy bit, what is more important is the control mechanism.
- You need:
  - MASTER table with a row per load type
  - RUN table with a row per run
  - ERRORS table for recording warnings and errors
  - LOG table to hold stages, record counts and timings
- How Gold Plated does this need to be? Restart stages? Allow run-time abandon?
- Do you want to be able to ramp up and down the number of records to process?

Master & Run Table

```
create table data_load_master
 (process_name         varchar2(30) not null
 ,status               varchar2(1)  not null
 ,stage                number (2)   not null
 ,process_range        number(12)   not null
 ,batch_size           number(5)    not null
 ,last_exec_timestamp timestamp(6) not null
 ,log_level            number(1)    not null
 ,abandon_fl           varchar(1)   not null
 ,last_id_num1         number
 ,window_start_id_num1      timestamp(6)   -- last run window for recovery
 ,window_end_id_num1        timestamp(6)
)
 create table data_load_run
 (process_name        varchar2(30) not null
 ,executed_timestamp timestamp(6)   not null
 ,status               varchar2(1)  not null
...
 ,start_id_num1       number
 ,end_id_num2         number
 ,records_processed   number
 ,records_skipped     number
 ,records_errored     number)
```

Instrumentation

- **Instrumentation is your code telling you what it did and how long each bit took. Without it you are blind.**
- **Instrumentation has an overhead – of *minus ten percent or so.* i.e. it long-term makes your code faster**
- **DBMS_OUTPUT is easy but a bit poor**
- **DBMS_APPLICATION_INFO is really easy and great.**
- **Logging to a table is best.**
- **If you do not instrument you bulk-processing code I will never buy you beer or whisky.**

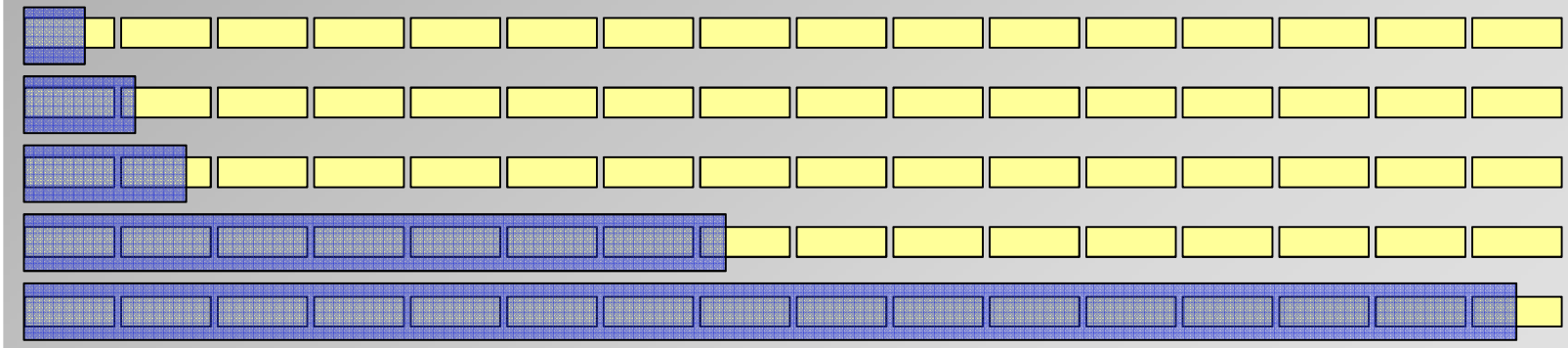
Breaking Up the Data

- If you are not processing “everything” (maybe as you do not have it, it is too large, time windows are too small) you need to be able to accurately partition up the data
- This needs to be a simple, accurate and preferably global method of sectioning the data.
- You *may* be able to get away with aiming only for final “consistency” but that ends up being a rare requirement
- An existing partitioning scheme may be a good candidate for this - but don’t be fixated on it, think of alternatives
- Ensure it scales. You might have to use a safe but limiting “guestimate window” and an accurate range within it.

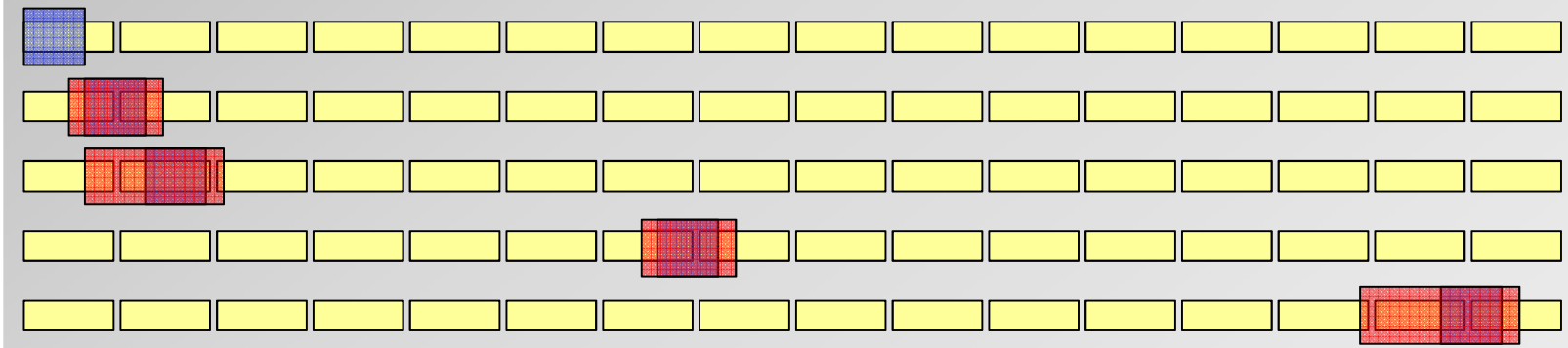
Finding your Processing Window

Select by POST date

Partition by CRE_DT



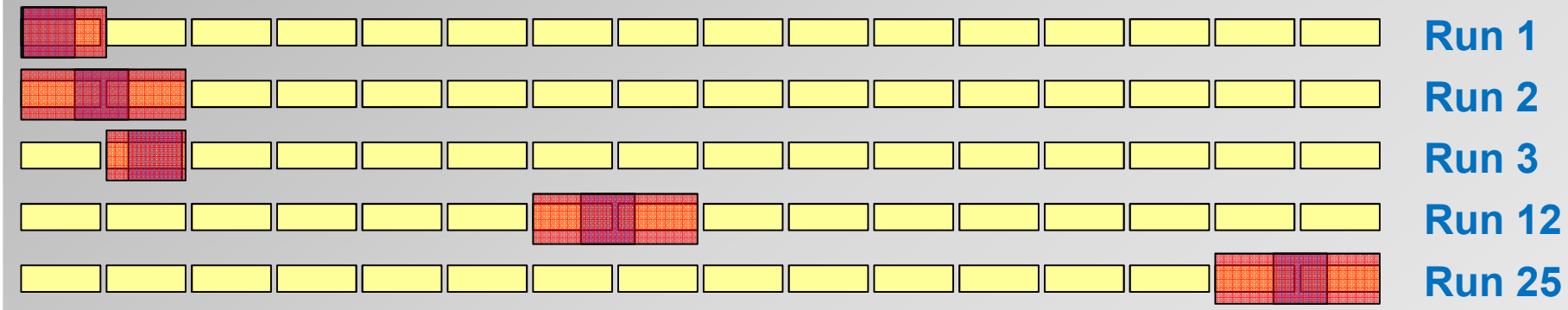
Select by POST date within 1 month of CRE_DT



Identifying Your Data Range

Correlated Partition Exclusion

Select by POST date within
Having looked up the min and max POST date for each partition



You can either calculate a safe range of values of the partitioning key for you data selection key

Or you can have a table holding e.g. the Min and Max order date for each partition defined by the ORDER ID

PL/SQL Controlling SQL

- PL/SQL and SQL are not mutually exclusive, oddly enough!
- There is NOTHING WRONG with having a PL/SQL controlling scaffold that calls straight SQL. I usually start by using bulk PL/SQL but replace with straight SQL once I am sure no surprises or left to stumble over.
- I have even written a “dirty” one-off load where I used PL/SQL to spit out a string of SQL statements into a script that I then ran – and I tracked progress in an excel spreadsheet
- I am not sure I would do it that way again though.

Example 1 – Bulk PL/SQL Processing

- **Client had a "mad" design, taking a normalised set of 6 (7…8…9…) tables from DW and merging them one massive, wide table with a pair of records per transaction – for OLTP.**
- **Initial take-on of *X*TBs of data and *then* near-real-time update.**
- **Specification lax, data quality questionable, errors and special-case handling likely. Re-start vital.**
- **This was too complex for straight SQL, the Java guys had little bulk data experience, PL/SQL was the most flexible and thus best solution.**
- **Bulk Processing of data in chunks.**

Example 1 – Bulk PL/SQL Processing

- **You can't use sequences in an ordered step so I had to have a SQL statement and thus a context switch per row:**

```
select seq_gen.nextval into lv_ttt_target(batch_loop).ID from dual;
```

- **You need a fair amount of controlling logic (the PL/SQL framework) and some control tables to keep track of progress and allow restart.**
- **Once it is up and running, living with it is easy.**
- **Performance is orders of magnitude better than row-by-row – but 1 order of magnitude slower than straight SQL.**
- **Where I could, I converted the BULK SQL step into a straight SQL INSERT or UPDATE, still controlled by the PL/SQL Framework .**

```
-- A driving cursor and control tables tracked timestamp range of data to process
 cursor get_source1 is
 select ...
 where ( driving_table.creation_ts > v_start_ts )
 and   ( driving_table.creation_ts < v_end_ts   )
    ORDER by sdtrxtra.creation_ts ) T -- ordered set of records.

-- open the cursor and start processing in batches to control memory use
open get_source1;
  <<main_loop>>
   loop
...
--Fetch data from the ordered range into an array. one step,one context switch
 fetch get_source1 bulk collect
      into lv_gst_data limit pv_batch_size;

-- For every record fetched do some checks, modify columns and copy into
-- an out array
for bl in 1..lv_gst_data.count
          loop
...
 lv_ttt_target(bl).CREATE_DATETIME   := nvl(lv_gst_data(bl).CREATE_DATETIME;
 lv_ttt_target(bl).MATCHING_KEY      := lv_gst_data(bl).MATCHING_KEY;
 lv_ttt_target(bl).ORIGINAL_TRADE_ID := lv_gst_data(bl).ORIGINAL_TRADE_ID;

-- and push the resulting array out in one step - 1 context switch
 forall insert_index in indices of lv_ttt_target save exceptions
            insert into intermediate_eric
            values lv_ttt_target(insert_index);
```

Example 2 – Bulk SQL Processing

- **Client had 4TB or so of data to process from SQL*Server to Exadata "in a day". Simple data take-on, ODI would then maintain the data set.**
- **Used Oracle Transparent Gateway to pull copies of the reference data and 6 massive tables over to Oracle – was really efficient!**
- **Data processed as a flow - rows never updated, the data was moved from table to table as INSERT-INTO-SELECT**
- **ODI code taken and modified to perform for large data sets.**
- **No dataguard so could use *nologging* and all data could be driven off creation date range (with some little tricks)**

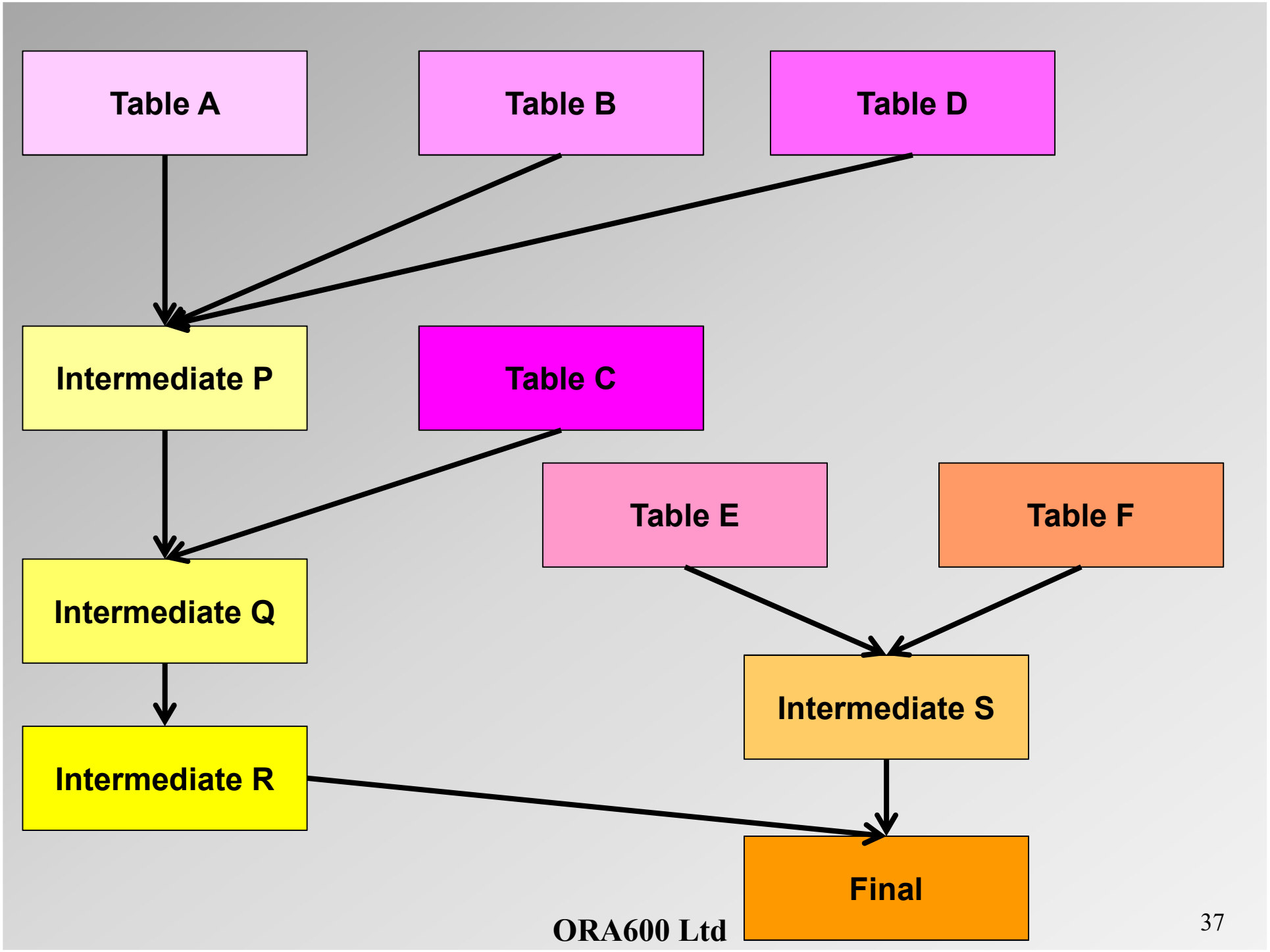


Table A
Table B
Table D
Intermediate P
Table C
Table E
Table F
Intermediate Q
Intermediate S
Intermediate R
Final

Example 2 – Bulk SQL Processing

- **Very, very fast.**

- **“Single Shot” over a weekend so no controlling framework.**
- **Spreadsheet and a processing SQL script to run the flow of SQL INSERT statements for a set range of rows.**
- **A master SQL script to call the processing script.**

- **Data processed as a *flow* - rows never updated, the data was moved from table to table as INSERT-INTO-SELECT.**
- **(would have used parallel except for a “different result error”!)**
- **Final table partitioned in line with ranges, so if a step failed you could truncate to move to a known point,**

- **Absolutely no good for continues running ☺**

Did I finish on time?
Questions?

Bulk Processing Data with SQL and PL/SQL

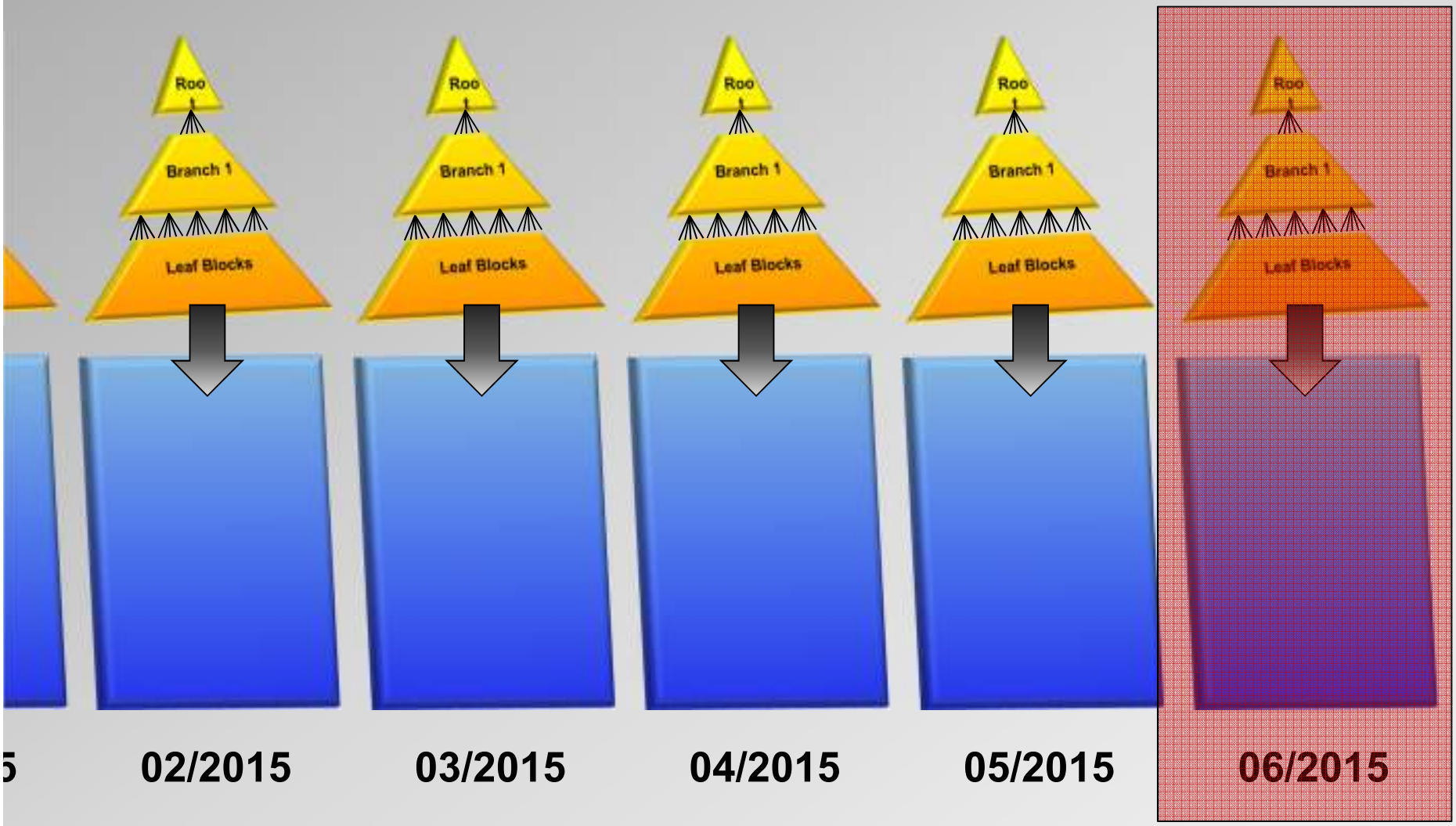
Nordic OTN Tour - Norway
12th October 2016

Martin Widlake
Database Performance, Architecture & Training
Ora600 Limited

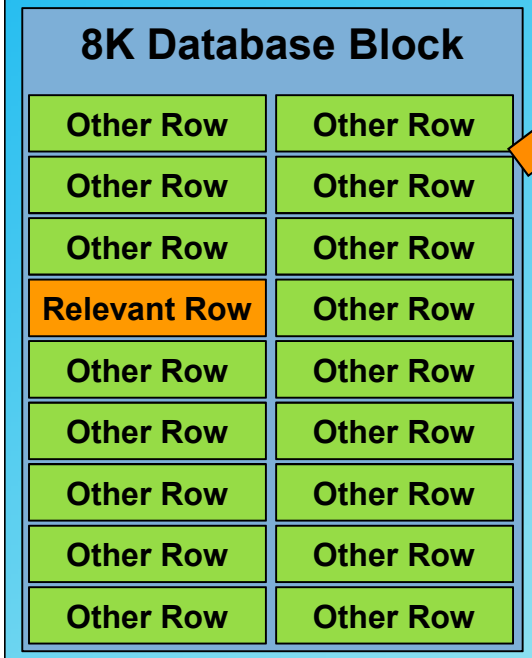
mwidlake@ORA600.org.uk
http://mwidlake.wordpress.com/
Oh, and that twitter thing - @mdwidlake

Partitioning

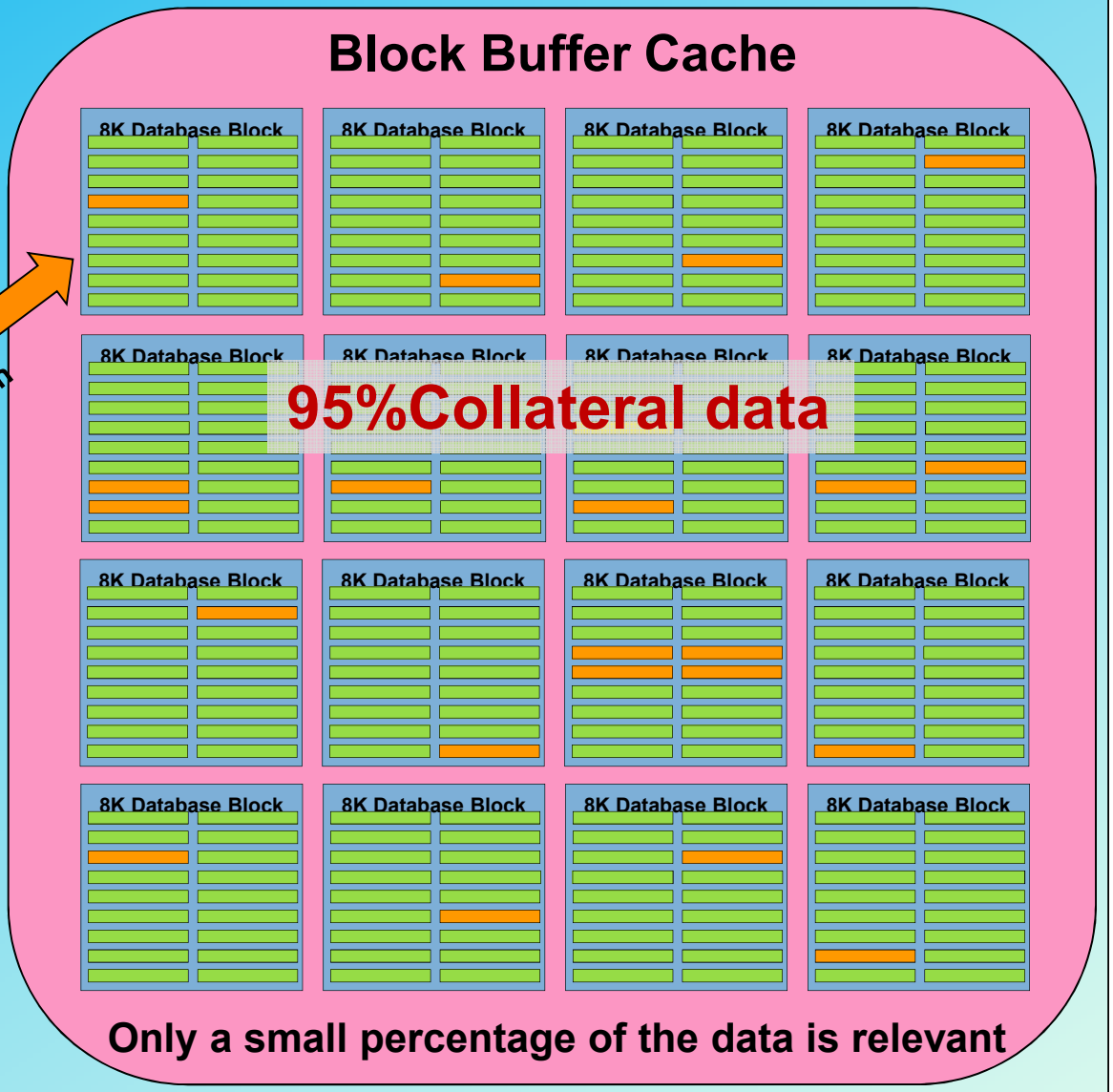
- It is all about working data set:



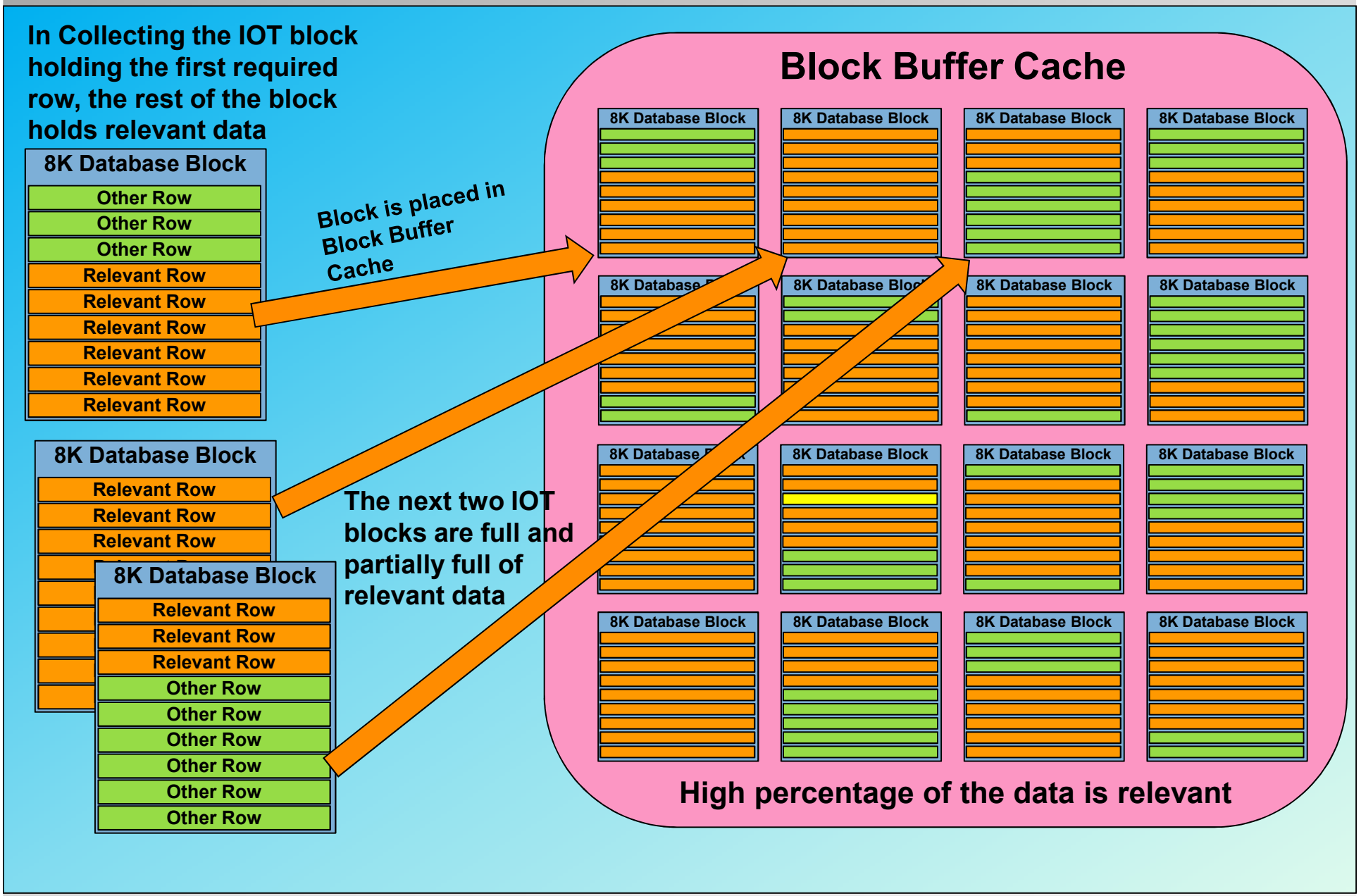
To collect a row, a whole block is fetched from disk



Block is placed in Block Buffer Cache



Often, most if not all of the other rows in the block are not relevant to the query


In Collecting the IOT block holding the first required row, the rest of the block holds relevant data
8K Database Block
Other Row
Other Row
Other Row
Relevant Row
Relevant Row
Relevant Row
Relevant Row
Relevant Row
Relevant Row
Block is placed in Block Buffer Cache
8K Database Block
Relevant Row
Relevant Row
Relevant Row
8K Database Block
Relevant Row
Relevant Row
Relevant Row
Other Row
Other Row
Other Row
Other Row
Other Row
Other Row
The next two IOT blocks are full and partially full of relevant data
Block Buffer Cache
8K Database Block
High percentage of the data is relevant

Physical Ordering of Heap

- An alternative to IOTs to gain physical clustering of data is to rebuild the HEAP via an ordered select:

```
insert into CHILD_HEAP_ORD
   (pare_id,cre_date,vc_1,date_1,num_1,num_2)
 select pare_id,cre_date,vc_1,date_1,num_1,num_2
 from CHILD_HEAP
 ORDER BY 1,2
```

- This is intrusive, it prevents proper access to the table and indexes need to be rebuilt, stats gathered...
- Is a "One Shot" physical ordering. Updates and inserts will mess it up.
- Use of partition swap can make this a usable technique.

12c CLUSTERING_BY clause

- **Oracle 12.1.0.2 introduced the new Attribute Clustering feature whereby you can partially order heap tables when you bulk direct insert into them.**

```
SQL> create table ziggy2 (id number, code number, name
varchar2(30))
clustering by linear order (code)
without materialized zonemap;
```

Then bulk insert into the table and add indexes

OR

```
alter table ziggy add clustering by linear order(code)
without materialized zonemap;
```

Then move the table and rebuild the index

Template Slide

- Point one
- Point two
- Point Three
- Point Four
- Point Five

Template Slide Animated

- **Point one**
- **Point two**
- **Point Three**
- **Point Four**
- **Point Five**

Any Time For Traditional Bulk Processing & War Stories?