Actian Hybrid Data Conference
2017 London
Disclaimer

This document is for informational purposes only and is subject to change at any time without notice. The information in this document is proprietary to Actian and no part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of Actian.

This document is not intended to be binding upon Actian to any particular course of business, pricing, product strategy, and/or development. Actian assumes no responsibility for errors or omissions in this document. Actian shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. Actian does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
X100 Cookbook

Pete Lydon
Principal Consultant, Actian Services - EMEA
X100 Engine
Google - being the centre of the Universe...

- So I thought I’d ‘Google’ x100 and I came up with this
Google – trying a little harder

- So this time I thought I’d Google ‘x100 engine’ ...
So it’s a performance engine..
Like with any performance engine(s)...

- You need to look after them with regular maintenance and ensure that if you get any warnings you consult the ‘manual’

- Though as a starter for 10.. and an important ‘take away’
  - The x100 process is a ‘PER DATABASE’ engine

- X100 engines aren’t aware of one another
  - Queries for different databases running simultaneously will potentially compete for RAM.
  - They will very much more likely compete for CPU

- This means that you need to factor in (in advance ideally) the number of databases you are going to have so you can apportion memory appropriately.
Performance engine(s) cont’d

Hitting the ‘return key’ at install time sets defaults for EACH x100 database created

– Processing uses 50% of TOTAL memory
  • MAX_MEMORY_SIZE

– Buffer pool is 25% of TOTAL memory
  • BUFFERPOOL_SIZE

– To allocate different memory settings for additional databases, create a bespoke settings file for EACH Database.

– Do this before creating your database.
Vectorwise.conf

The vectorwise.conf file holds parameters such as memory, query parallelism and data block size info.

- The file exists under the default DATA location path

```bash
ingres:g1> ingprenv II_DATABASE /rdbms/group1
ingres:g1>
ingres:g1>
```

- In the vectorwise DIR

```bash
ingres:g1>
ingres:g1> cd /rdbms/group1/ingres/data/vectorwise/
ingres:g1>
```

- To override this, we need a database specific one
Vectorwise.conf (contd)

So we create our template file using the following syntax:

```
vectorwise.<database_name>.conf
```

Now when we run ‘createdb pmldb’.........
The x100 engine over allocates on how many cores you have?

If your server has 20 Cores, then the x100 engine will try to use up to 25 (it over allocates)
  • This is for concurrent processing
max_parallelism_level

So we need to plan:

- How many concurrent queries will run through our system?

- Remember X100 database engines run in isolation.

- Do I have more than one active database?
  - Typically in Vector and Vector-H we have 1

Question?

- What will happen to my performance of my queries if I run many queries at the same time?
- Will I run out of CORES?
max_parallelism_level (scenario)

Queries arrive at our x100 engine

With a 20 CORE server the x100 engine acts like the machine has 25

If we have max_parallelism set to 8 (and the query allows) we will use 8 CORES for it.

Our second query arrives;- We now have 25-8 CORES available (17), so our second query could still use 8 CORES (leaving 11)

Subsequent queries get up to 20% of the balance, which may be ONE!
Database & Table Design
Database/Table Design Considerations

- Schema Options
- Secondary Indexes
- Appropriate data types
- NULLs
- Keys
- Data Ordering and Data Load
- X100_row
Schema Options

- **Snowflake?**

- **Star?**

- **Galaxy?**
Infact....

When it comes to the Vector Engine.

We don’t care about:-
- Schema Choices/design
- Secondary Indexes
Secondary Indexes (No!!!!!!!!!!)

Traditionally used for fast retrieval on data that is not the key.

X100 data is stored in columnar blocks (on the whole)

The x100 engine does not need them, in fact creating them and keeping them will only make things slower.

Each block for each column has a min-max index included
• Maintained automatically
• Means that we already have an ‘index’ (of a fashion on everything)
Know your data!

Do not store plain character data in a NCHAR type of field
  • It uses twice the storage

’space’

Do not make everything VARCHAR(200)
  • Choose appropriate lengths for each column
  • Consider CHAR over VARCHAR for 10 bytes or less
Appropriate Data Types

If you are using a TINYINT, a CHAR(1) would be a better fit.
– It’s better for doing comparisons

– FLOAT’s are inefficient
  • They don’t compress in Vector-H
  • Would DECIMAL suit?

Do you know your data?
• If you can use an INTEGER, that would be better still
Appropriate Data Types

The right datatypes help compression.

Better compression = more data per block

More data per block = x100 engine needs to read less blocks

Reading less blocks = improves query performance

(Better compression also means a smaller database and therefore backup and recovery is quicker)
NULLs

- Null values incur more complexity in queries and updates
  - E.g. IFNULL

- They use more space
  - 50 columns in a table defined as NULL = 100 columns in that table.
    - A 300 table DB, could therefore have 1,000s of extra NULL Cols!!

- Consider using a flag
  - E.g. No of Children (a negative Number for unknown)

...  
  `Num_children TINYINT NOT NULL with DEFAULT -1`

- If we think of ‘big data’, if this data was loaded via some ETL process, we should be able to move this away from using NULLs
A concatenated key (or composite key) is typically used to combine two or more columns together to simplify SQL.

With the x100 engine it is has been seen that it is better to split these back to the component parts to improve query performance.
Keys (cont’d)

E.G – business_trans_table

<table>
<thead>
<tr>
<th>comp_country_trans_id</th>
<th>EEUK0001234567890</th>
<th>EEUK0001234567891</th>
<th>EEUK0001234567892</th>
<th>EEUFR0001234567890</th>
</tr>
</thead>
</table>

Created from

**Company** & **Country** & **Trans_Id** fields

| EE | UK | 0001234567890 |

What happens when we look for specific “FR” transactions?
Keys (solution)

business_trans_table

<table>
<thead>
<tr>
<th>company</th>
<th>country</th>
<th>trans_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE</td>
<td>UK</td>
<td>0001234567890</td>
</tr>
<tr>
<td>EE</td>
<td>UK</td>
<td>0001234567891</td>
</tr>
<tr>
<td>EE</td>
<td>UK</td>
<td>0001234567892</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>EE</td>
<td>UK</td>
<td>0001234599999</td>
</tr>
<tr>
<td>EE</td>
<td>FR</td>
<td>0001234567890</td>
</tr>
</tbody>
</table>

- Separating columns will improve retrieval speeds.

- The x100 engine when planning can easier discount blocks for each of these columns that don’t meet our criteria
Data Ordering

The order in which data is stored and thereby retrieved by our x100 engine can influence efficient data retrieval

For Example:-
• We have a customer table ordered by a create_date (the load order)
• We retrieve data mainly by “customer_name”
• Similar customer name data is spread across many/all blocks
• Better to have this tables organised on this column

• Change the ETL process (perhaps) that loads the data?
• What about the next day and the next?
  • Data is still be loaded in blocks of days
Data Ordering (cont’d)

Well the fix is to create a secondary index on the table

I know what I said earlier.........

After we create the secondary index and commit it,

After the commit we drop the index.

Internally Vector has sorted the table data

**Before**

7340032 Nov 3 08:06 actianSpml 1 x100 table name 0000000000000007d

**After**

47714304 Nov 3 09:05 _actianSpml_1_x100_table_name_0000000000000504
Small x100 data sets

There are times when we will have small Vector Tables.

- For these consider:-
  - WITH STRUCTURE = x100_row

- Stores column sets in a row on a disk block
  - Unlike the default where each column set is stored on its own disk block

- Useful when there are a large number of columns and few rows
  - Memory
  - Disk space
Small x100 data sets

Consider:

```sql
CREATE TABLE country(
    country_id INTEGER NOT NULL NOT DEFAULT,
    name CHAR(30) NOT NULL NOT DEFAULT,
    region_id INTEGER NOT NULL NOT DEFAULT,
    comment CHAR(100) NOT NULL WITH DEFAULT);
```

- The number of countries is 196 (if you include Taiwan)
- If a block is 512K bytes
- Each column will fit onto a block
- For a default x100 table country will require: 4 blocks
- For a row-based x100_row table country will require: 1 block

Is the “name” field long enough? Shouldn’t the “comment” field be VARCHAR?
Google once again.....

Was our country name big enough?
Optimization
Traditionally to update our statistics we have maintenance routines that run:
– Adhoc times
– Nightly
– Weekly
– When we remember

It’s a Utility that’s typically ‘plumbed’ in to other jobs
– Batch operations etc

‘create statistics’ can be run from within a transaction/session.
Optimization using ‘create statistics’

For x100 tables we recommend you use ‘create statistics’

You can create them when you need to
• e.g. within your batch/ETL job

CREATE STATISTICS FOR ALL TABLES
    FOR user1 TABLES

CREATE STATISTICS FOR tab1, tab2(id, name)

• This will create statistics on
  • The “tab1” table
  • The columns “id” & “name” on “tab2”
If we want to remove statistics..

As with using ‘create statistics’ there is a similar ‘drop’ command for removing them.

**DROP STATISTICS FOR** table [(column-list)] {, table [(column-list)]}

- We can list out tables and columns
- No need for –r and –a flags
  - E.g. **DROP STATISTICS FOR employee(emp_id, name)**

**DROP STATISTICS FOR ALL TABLES**

**DROP STATISTICS FOR ALL user1 TABLES**
Moving Data
Moving Data

So far...
– We’ve chosen the correct data types
– We organized our data so that it compressed well on disk

Now we want to
– Run lots of reports
– Move data around
  • Could be user data
  • Need to make a copy before making a change
Copying Data

Copydb

- Use `--with_csv`
  - It’s a lot smaller than “-c” option

- Use `INSERT INTO EXTERNAL CSV 'file.name' SELECT * FROM table`
  - Faster than standard Copy

* insert into external csv '/home/actian/pml_1_x100.csv' select * from pml_1_x100\g
* insert into external csv '/home/actian/pml_1_x100.csv' select * from pml_1_x100
Executing . . .

(5767168 rows in 6.274650 secs)

* copy pml_1_x100() into '/home/actian/pml_1_x100.binary'\g
* copy pml_1_x100() into '/home/actian/pml_1_x100.binary'
Executing . . .

(5767168 rows in 16.409628 secs)
Parallelism (Cores) – copying data

X100 engines copy data out faster
• Parallelizing the copy
• Multiple output files can be created

The engine has some smart AI in it.

Small tables will not be parallelized, as it is aware that it will take longer to set-up than just do.

```
443514880 Nov 3 13:53 pml_1_x100.csv.007
445808640 Nov 3 13:53 pml_1_x100.csv.003
444858368 Nov 3 13:53 pml_1_x100.csv.006
446496768 Nov 3 13:53 pml_1_x100.csv.005
446955520 Nov 3 13:53 pml_1_x100.csv.002
445120512 Nov 3 13:53 pml_1_x100.csv.004
446988288 Nov 3 13:53 pml_1_x100.csv.001
446103552 Nov 3 13:53 pml_1_x100.csv.000
```
Data load

So one funnel looks the wrong way up. But is it wrong?

VWLOAD when used from the command line is a single threaded operation
- Unless using cluster option (Vector-H)

When using COPY command via SQL with VWLOAD, the process will be parallelized
- As long as there are multiple input files

```sql
COPY pml_1_x100() VWLOAD FROM 'pml_1_x100.csv.000','pml_1_x100.csv.001','pml_1_x100.csv.002','pml_1_x100.csv.003','pml_1_x100.csv.004','pml_1_x100.csv.005','pml_1_x100.csv.006','pml_1_x100.csv.007' WITH WORK_DIR='/home/actian', QUOTE='''', FDELIM =',
Executing . . .

(5767168 rows in 12.980916 secs)
continue
```
Data load (cont’d)

Using this syntax we are able to parallelize the load.

- 5.76 Million rows
  - (As below) 12 Seconds
  - (Running Serially) 60 Seconds
  - (Standard ‘copy’ as CSV) 65 Seconds

```sql
COPY pml_1_x100() TO VWLOAD FROM 'pml_1_x100.csv.000',
'pml_1_x100.csv.001',
'pml_1_x100.csv.002',
'pml_1_x100.csv.003',
'pml_1_x100.csv.004',
'pml_1_x100.csv.005',
'pml_1_x100.csv.006',
'pml_1_x100.csv.007'
WITH WORK_DIR='/home/actian',
QUOTE='"', FDELIM='',
```

\p\g
Application Data Consumption
Sending data to your application

Our x100 Engine is quick

Whatever way we present our data

With x100 we could be dealing with lots of it.

Look at ‘trimming’ concatenated character columns in views.

The benefits?
– Smaller column widths
  • Smaller transmission sizes
  • Less Memory used
Compressed Data Transmission

With our latest JDBC Jar.file
– Compression is enabled by default

With applications that pull in large volumes of data, from the database.
– Up to 20x faster

IngresNET clients (IIGCC)
– Do not have compression enabled by default
Thank you!
Coffee Break - 15 minutes

#hybriddataconference