

Data Fragmentation for Load and Query Performance



J. Warren Donovan

US Courts

Session: E08

Tuesday, April 28, 2009 • 10:45 – 11:45

The Power Conference for Informix Professionals

Data Fragmentation for Load and Query Performance

By J. Warren Donovan

US Courts

10+ Years Informix and DB2

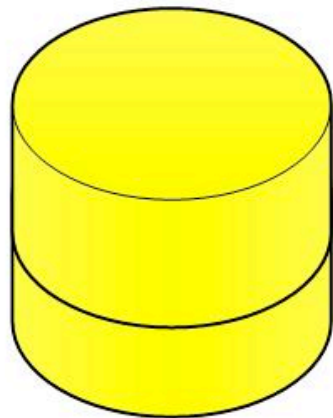
Data Warehousing Experience

Traditional Disk / Dbspace Configuration

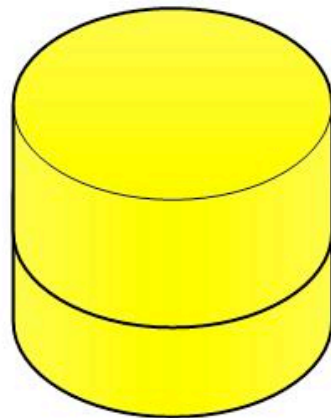
- Traditionally, one creates a table or tables in a dbspace.
- The dbspace is created from a number of chunks, and, as the database fills, you add more chunks.
- Some systems just add chunks from a pool of disks to each dbspace, some systems try to separate out the most popular tables onto different dbspaces which are on different disks.
- For DSS performance, either way leaves a lot to be desired...

Just Adding Chunks as Required

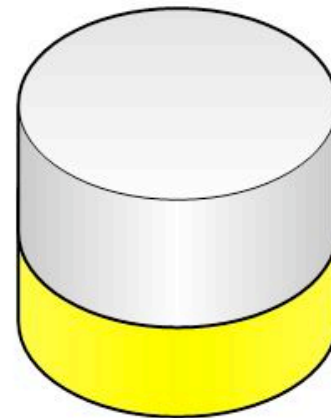
Creating a single dbspace, adding each device as a chunk...



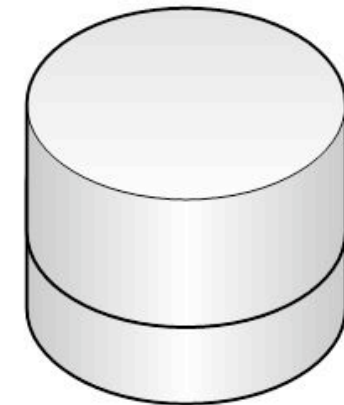
Chunk1
Device 1



Chunk 2
Device 2



Chunk 3
Device 3

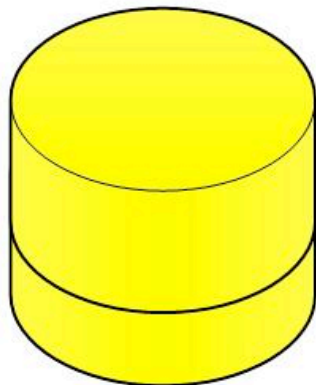


Chunk 4
Device 4

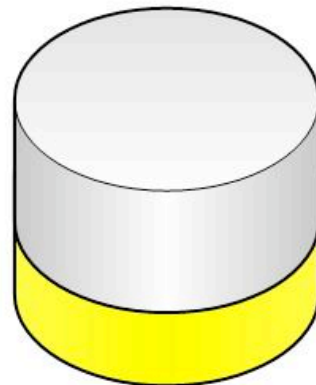
...You wind up filling the first chunk, then the second, then the third. When reading / writing you wind up with a single thread that must read through each disk.

DBSpace per Device, Spread Tables by Size / Popularity

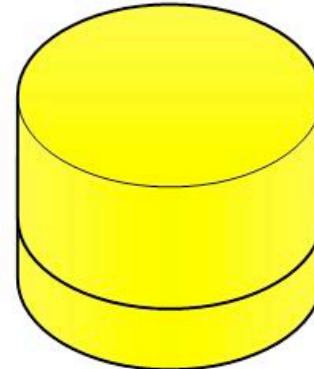
Creating a dbspace on each device and trying to spread you tables across them...



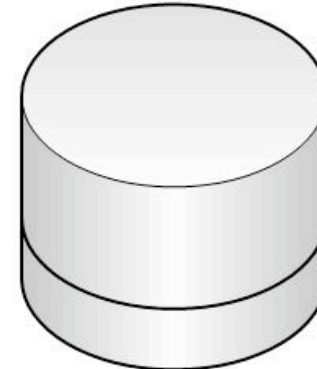
Dbspace1
Disk 1



Dbspace2
Disk2



Dbspace3
Disk3

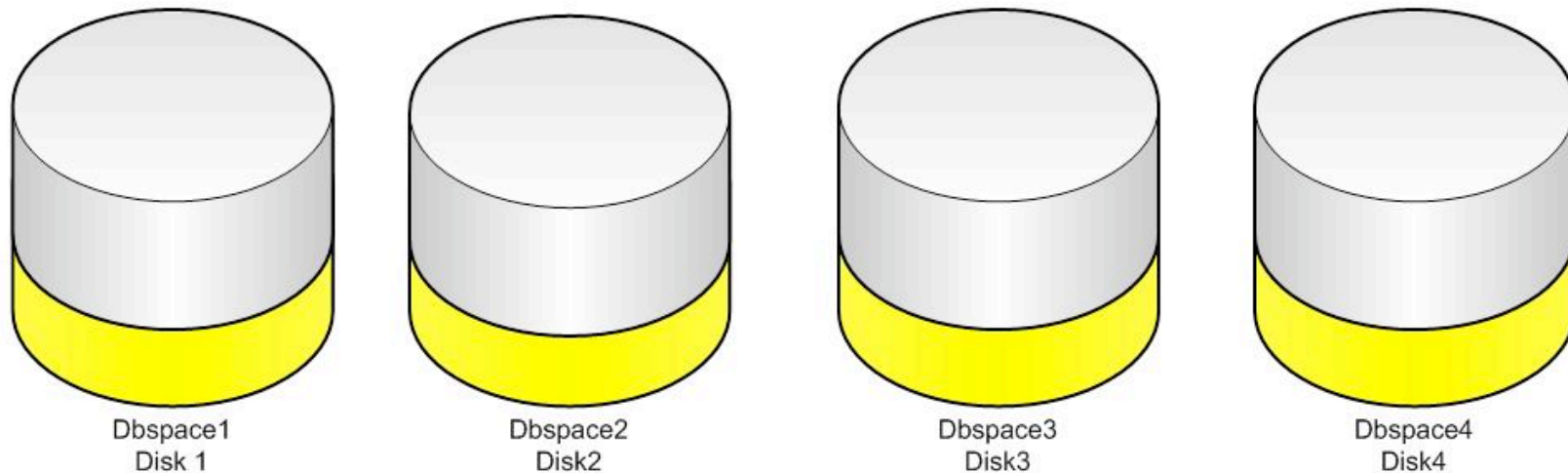


Dbspace4
Disk4

...You could wind up with your biggest table filling its dbspace, and having smaller tables wasting space on the others. Also, if your biggest tables are your most popular, you create a hotspot on that disk while the other sit idle.

Fragmentation – What you Really Want

Fragment your tables across your dbspaces that you have created on each device..



...and you can balance both your data and your I/O across your spindles.

Types of Fragmentation

ROUND ROBIN – Easy to implement, automatically spreads your data evenly across all the dbspaces you specify, balancing your disk usage and I/O. No fragment elimination.

HASH (XPS only)- Spreads data based on an algorithm Informix uses against a field you tell it to use. Use a very granular field for best results. Done right, is an easy way to balance I/O and disk and, if your granular field is often used in WHERE clauses, you could get fragment elimination.

EXPRESSION – Use your own expression: takes more time, harder to maintain, but can be better tuned to improve your users queries and your loads...and many other benefits

HYBRID – Also XPS only...anyone here use XPS? I miss it...

Purposes of Fragmentation

PERFORMANCE, obviously, but how?

- Balancing Data Across Disks
- Balancing I/O Across Disks and Parallelism
- Fragment Elimination
- Dropping / Adding partitions to improve Load Performance

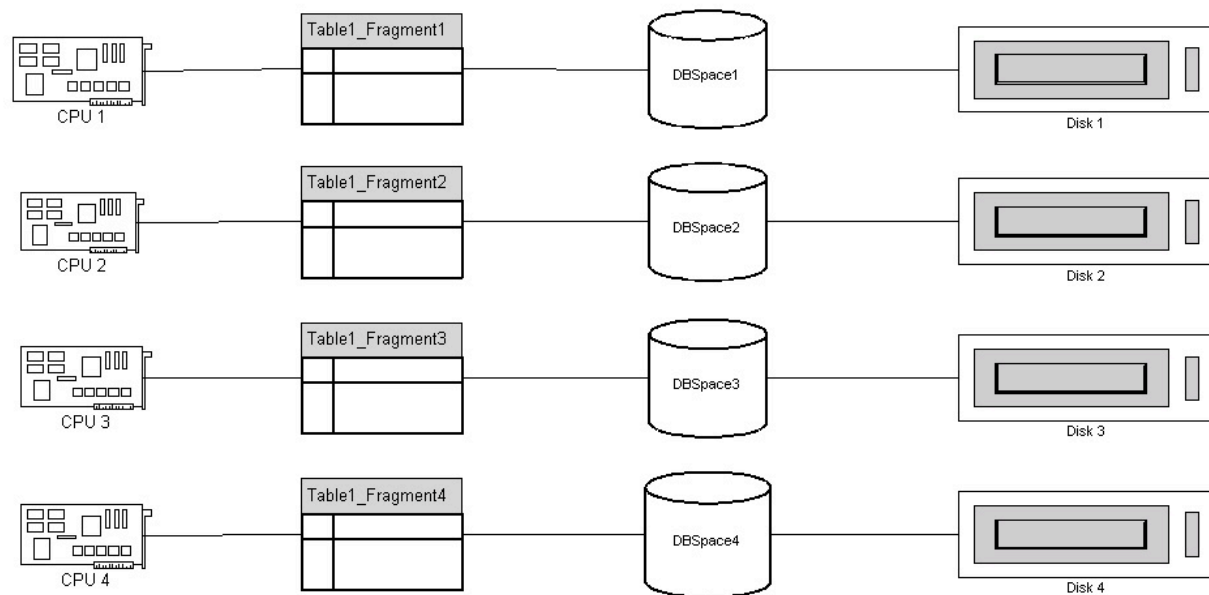
Round Robin

- Automatically balances the data in your table evenly across all the dbspaces you specify.
- Automatically balances your I/O evenly across all the dbspaces you specify.
- No Fragment Elimination (I/O reduction)
- No way I can think of to drop / add partitions to improve load performance.
- Can still combine with parallelism (PDQ) to get massive performance increases...but this is another story.

Round Robin for quick Balance, Parallelism and Performance

- It's a different story but I have this cool picture:

PDQ, Fragmentation and I/O



Fragment Table,
of Fragments =
of Active CPUs to
maximize Multiprocessing

Put each fragment in
its own DBSpace (with
only one chunk)..

...And each chunk on
its own disk...

...to maximize I/O!

Fragment BY ROUND ROBIN

- Here's the SQL to fragment a table by ROUND ROBIN.
- Simply replace your IN clause with:
- **FRAGMENT BY ROUND ROBIN IN**
<list of dbspaces delimited by comma>

```
CREATE TABLE nhl_player_stats
( player_last_name varchar(30), player_first_name varchar(30), player_number integer, nhlpa_player_id integer,
  team char(3),
  season char(8), goals integer, assists integer, plus_minus integer, pim integer )
fragment by round robin in dbspace1, dbspace2, dbspace3, dbspace4
extent size 10000 next size 1000
lock mode page;
```

Purposes of Fragmentation

PERFORMANCE:

- Balance Data Across Disks
- Balancing I/O Across Disks and Parallelism
- Fragment Elimination
- Dropping / Adding partitions to improve Load Performance

Since Round Robin cannot do the last 2, and HASH and HYBRID are XPS ONLY, I will now seek to demonstrate the steps you would take to use EXPRESSION based FRAGMENTATION to meet these goals.

Fragment by Expression Basics

- Fragmentation by Expression
 - Allows you to fragment your data across tablespaces or even in named partitions in the same tablespace by an expression you choose
 - Almost any normal evaluator is valid (=, >, <, <>)
 - Gain Fragment Elimination
 - Avoid overlap - Evaluates in the order they are put in – if you tell your table to fragment where value < 10 in tablespace1, then value > 8 in tablespace2, all the 9s will be in tablespace1.
 - Make sure your expression captures all the data. If data containing a value you do not anticipate is entered, you get an error.
 - That said, there is a remainder in <tablespace> option. Use it as a last resort since the remainder fragment must ALWAYS be read.

Balancing Data Across Disks with Fragmentation by Expression

One goal of fragmentation could be to spread your data evenly across your dbspaces and disks.

- To use your own expression takes work to set up and to maintain.
- Pick an expression that allows you group chunks of data that are relatively the same size.

Example:

```
CREATE TABLE nhl_player_stats  
( player_last_name varchar(30), player_first_name varchar(30), player_number integer,  
  nhlpa_player_id integer, team char(3), season char(8), goals integer, assists integer,  
  plus_minus integer, pim integer )
```

fragment by expression

season = '20042005' in dbspace1, season = '20052006' dbspace2,

season = '20062007' dbspace3, season = '20072008' in dbspace4

extent size 10000 next size 1000

lock mode page;

Balancing I/O Across Disks

- Perhaps an even more important goal of Table Fragmentation is to balance the I/O.
- Assuming all your disk devices are roughly the same size, if you balance the data across it, but one section of the data is accessed much more frequently than the others, you have an I/O hotpoint.
- In the previous example I used a date range for fragmentation...this might have repercussions here, since normally the newest data is the most popular!
- If everyone hits the data in one fragment, on one disk you could easily just create a hot-spot and have no performance improvement at all!

Balancing I/O Across Disks with Fragmentation by Expression

EXPRESSION – When fragmenting by expression, factor in the popularity of the data in each fragment.

For example:

- A date range is a common key to use to fragment data. Normally data in each period, say, each Fiscal Quarter, will be roughly the same. Thus the data will spread evenly across your dbspaces.
- However, often the most popular data for end-users to query is the most recent – so you could easily just create a hot-spot and have no performance improvement at all!

Fragment Elimination

- When you fragment your table based on a range of data, when a query is submitted that has that field in the where clause, only that fragment has to be read.
- For example: if I have a table fragmented by year, and a query requests the data from a specific year, or from several years, only the fragments that contain that data will have to be read from.
- Getting back to that date range, I have an I/O hotspot, but at least queries on that data don't have to read the whole table.
- However, if we go through the queries we might find another field, like team_name, which can be used to create fragments that balance both the data and the I/O evenly .

Detach / Add / Attach Partitions to improve Load Performance

- If you fragment your table by a range of values (like a date range) that corresponds to how you load your data, you can save processing time by detaching / attaching partitions of data instead expensive delete / update statements.
- Detaching / Dropping and Adding or Attaching Partitions takes seconds, versus hours or minutes to update / delete.
- You can even pre-load the new data in a table that has the same schema and Attach it, meaning your reporting table will only be unavailable to end users for seconds.

Steps to Fragment for Load Performance

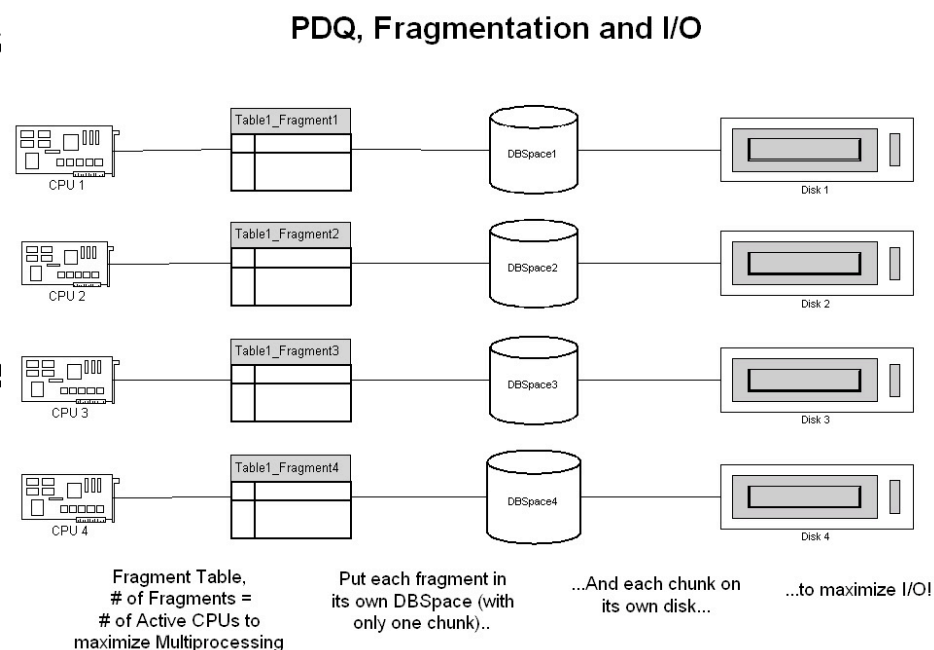
1. Have a Performance Goal and Plan in mind.
2. Know your physical dbspace – disk layout.
3. Evaluate ETL process to determine a good key to drop / add partitions.
4. Plan out the Steps of Your Load
5. Fragment your target table by expression.
6. Evaluate before / after load times.

Have a Performance Goal In Mind

- If you don't know what your goal is, how do you know you when you reach it?
- You might have a load window you need to fit into...or to reduce it and to increase data availability.
- Or, you might be more concerned with overall processing times.
- Fragmenting your data can take a lot of time and effort, especially if you have a large data warehouse...and DBAs are normally busy people. If you aren't having performance issues with your loads, maybe your time is best spent elsewhere.
- At a minimum, keep track of improvement. Time your current load process, make a change, and time the improvement. If your 2 hour load drops to 30 minutes, maybe it's worth replication on other loads. If it drops to 1 hours and 58 minutes...well...maybe it wasn't worth the effort.

Know your Physical Layout

- If you want to balance data and I/O across your disk, you need to know which disks your dbspaces are on.
- For example, it may not do you much good to fragment across a bunch of dbspaces that are actually physically located on the same disk...it could create I/O conflicts and actually hurt your performance!
- Try to avoid reading from the same disk you are trying to write to.



Evaluate your ETL Code to Determine a good Key to use

- You'll need to find a key that corresponds to how you load data.
- Typically, it will be DATE related: many systems add or update the latest week, month or quarter of data and may drop an old one.

Fragment Target Table by Expression

- Take a simple NHL_PLAYER_STATS table
- It contains regular season statistics for NHL players grouped by season.
- The latest seasons data gets updated with new data every night during the season.
- Since I know my ETL process will work on a season, I recreate the table fragmented by SEASON.
- Note that I named the partitions so that I could have multiple season on each disk and reference them by name to drop and replace them later.

Fragmented Table

```
CREATE TABLE nhl_player_stats
```

```
( player_last_name varchar(30), player_first_name varchar(30), player_number integer,  
  nhlpa_player_id integer, team char(3), season char(8), goals integer, assists integer,  
  plus_minus integer, pim integer )
```

fragment by expression (

partition 19501951s (season = '19501951') in dbspace1,

partition 19511952s (season = '19511952') in dbspace2,

...

partition 20042005s (season = '20042005') in dbspace1,

partition 20052006s (season = '20052006') in dbspace2,

Partition 20062007s (season = '20062007') in dbspace3,

Partition 20072008s (season = '20072008') in dbspace4)

extent size 10000 next size 1000

lock mode page;

- *I know the ETL process updates by season and so fragmented by that range.*
- *I named the partitions (20052006s) so that I can easily detach them later, even if I have multiple partitions in a single dbspace.*

Fragmentation to Improve Load Performance

- Just that fragmentation alone may help:
 - The ETL process can use fragment elimination for its update / inserts /deletes, so it now only has to run against a single fragment instead of against the whole table – in this case, with, say, 60 partitions, that would be 1/60th of the data it had to scan before!
- However, that still takes time and that data will be locked during that delete, update and insert...

ALTER FRAGMENT

ALTER FRAGMENT ...

INIT – To initialize a new fragmentation scheme

ADD – To add a new partition

DROP – Drop a partition...but not its data!

MODIFY – Modify a partition

ATTACH – Attach a table as a partition

DETACH – Detach a fragment to a new table.

ALTER FRAGMENT to Improve DELETE Performance.

- Say I can simplify my ETL to just DELETE then INSERT data.
- I can replace the DELETE with detaching the fragment and dropping it, then adding a new fragment to hold that data and running the insert.
- Detaching the fragment to a table, then dropping that table takes a second, a delete could take hours
- CAREFUL NOT TO JUST DROP A PARTITION – that does not drop the data in it, Informix tries to find another location for that data.

ALTER FRAGMENT to DETACH and ADD FRAGMENTS.

This statement detaches the target fragment to a table named nhl_ps_old:

```
ALTER FRAGMENT ON TABLE nhl_player_stats  
DETACH PARTITION 20072008s nhl_ps_old;
```

This statement creates a new empty fragment:

```
ALTER FRAGMENT ON TABLE nhl_player_stats  
ADD PARTITION 20072008s (season = '20072008') in  
dbspace4
```

This leaves us with an empty 20072008s partition in the nhl_player_stats and a table called nhl_ps_old with the old data (which you will want to drop at some point).

ALTER FRAGMENT to IMPROVE UPTIME

- Our delete, which used to take, say 30 minutes, now takes less than one second.
- Added bonus – when you detach a partition (and drop the resulting table) instead of running delete, you actually free up the space in the dbspace without having to run a reorg!
- However, during the INSERT, which takes an hour, that table isn't available...if only I could just attach the data...hey, you can!

Design Load Process

Goal: Maximize Table Availability

STEPS

1. Create new table with the same schema on the dbspace I want the fragment on.
2. Load data into the new table.
3. Detach the partition on the Main Table that has that data range.
4. Add the new table as a fragment to the Main Table
5. Update Statistics

ALTER FRAGMENT ATTACH

- Instead of detaching, recreating the fragment and then inserting...
- Create a table with the same schema
- Run the ETL insert into that table
- Detach the fragment as before, and attach the new table as a fragment with the same data range.

ALTER FRAGMENT ATTACH

```
CREATE TABLE nhl_ps_new (like nhl_player_stats)  
  in dbpace1;
```

Run ETL insert into nhl_ps_new...and test it.

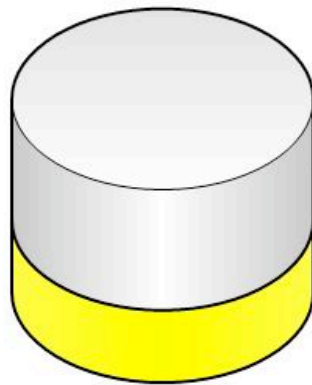
```
ALTER FRAGMENT ON TABLE nhl_player_stats  
DETACH PARTITION 20072008s nhl_ps_old;  
ALTER FRAGMENT ON TABLE nhl_player_stats  
ATTACH nhl_ps_new AS PARTITION 20072008s  
  (season = '20072008');  
DROP TABLE nhl_ps_old;
```

Steps to Fragmenting for Query Performance

1. For Query Performance – Evaluate data to determine good keys to balance data evenly across disk.
2. For Query Performance - Evaluate user SQL / Reports to determine good keys for fragment elimination.
3. For Query Performance – Evaluate system workloads to determine good keys for I/O balancing
4. Use EXPLAIN ON to confirm that fragment elimination is occurring.
5. Use before / after time to confirm individual queries performance improves.
6. Use before / after sample workloads to confirm whether overall performance improves.

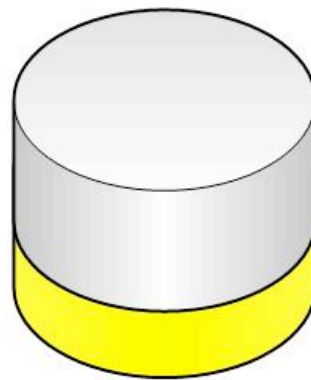
Fragment by Date Range: I/O Conflict

I balanced my data by fragmenting it by SEASON...
This was ALSO great for my load process.



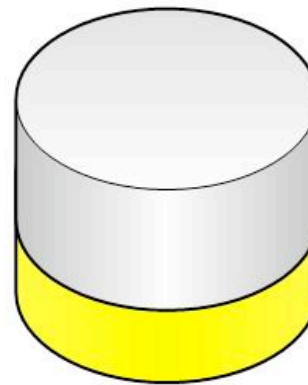
Dbpace1
Disk 1
Partitions:
20002001s
20042005s

10 Queries / Day



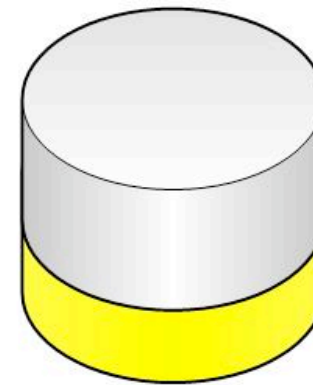
Dbpace2
Disk 2
Partitions:
20012002s
20052006s

20 Queries / Day



Dbpace3
Disk 3
Partitions:
20022003s
20072008s

90 Queries / Day



Dbpace4
Disk 4
Partitions:
20032004s
20082009s

**1000
Queries /
Day!**

BUT FROM MY iostat -D COMMANDS I SEE MOSTLY READS ON DISK4.
EVERYONE WANTS TO RUN AGAINST THE LATEST DATA!
THE DATA FOR THE 2008 – 2009 SEASON!
SO EVERY DISK EXCEPT DISK4 IS ALMOST IDLE!

Fragment for Query Performance

For query performance, ideally find a field on which the data balances well that is also used in the WHERE clause.

In this case, each team has about the same number of players, and most fans query the stats of their home team.

Some teams are more popular than others...so if you are going to group some teams on partitions that are in the same dbspace, make sure you match popular teams with unpopular teams.

Fragment for Query Performance

```
CREATE TABLE nhl_player_stats
```

```
( player_last_name varchar(30), player_first_name varchar(30), player_number integer,  
  nhlpa_player_id integer, team char(3), season char(8), goals integer, assists integer,  
  plus_minus integer, pim integer )
```

```
fragment by expression (
```

```
partition team_tor (team = 'TOR') in dbspace1, -- Toronto
```

```
partition team_fla (team = 'FLA') in dbspace1, --Florida
```

```
partition team_det (team = 'DET') in dbspace2, --Detroit
```

```
partition team_tpa ( team = 'TPA') in dbspace2, -- Tampa Bay
```

```
Partition team_nyr (team = 'NYR') in dbspace3, -- New York Rangers
```

```
Partition team_nas (team = 'NAS') in dbspace3, -- Nashville
```

```
Partition team_stl (team = 'STL') in dbspace4, -- St. Louis
```

```
Partition team_clb (team = 'CLB') in dbspace4, -- Columbus
```

```
... )
```

```
extent size 10000 next size 2000
```

```
lock mode page;
```

Fragment for Query Performance

```
CREATE TABLE nhl_player_stats  
( player_last_name varchar(30), player_first_name varchar(30), player_number integer,  
  nhlpa_player_id integer, team char(3), season char(8), goals integer, assists integer,  
  plus_minus integer, pim integer )
```

In this case, each team has about the same number of players, and most fans query the stats of their home team.

Some teams are more popular than others...so if you are going to group some teams on partitions that are in the same dbspace, make sure you match popular teams with unpopular teams.

Monitoring to see it worked

Some tools you can use to monitor to see that your fragmentation plan has worked:

`onstat -d` – see that the data is balanced

`onstat -D` – to watch reads during the day to see that I/O is balanced

`onstat -z` – to clear stats on `onstat -D`

`iostat` – watch the I/O at the disk level

`SET EXPLAIN ON AVOID EXECUTE` – to verify that individual queries are getting fragment elimination.

SET EXPLAIN ON

Use SET EXPLAIN ON AVOID_EXECUTE to make sure you are getting the fragment elimination you were aiming for without running the query.

Before your select statement, put the following statement:

```
SET EXPLAIN ON AVOID_EXECUTE;
```

This will output a sqexplain.out file, with an explain plan

In that plan, a scan statement like this shows you that it read only fragment 5 (they are numbered internally).

```
1) nhl_player_stats: SEQUENTIAL SCAN (Parallel, fragments: 5)
```

This output shows that fragmentation elimination did not help this query, it still had to access all the fragments.

```
1) nhl_player_stats: SEQUENTIAL SCAN (Parallel, fragments: ALL)
```

Fragment for Load and Query Performance

My new fragmentation scheme is great for user queries and balances my I/O – but now it doesn't help my loads!

Solution – well, it will be work, but why not MERGE the 2 schemes with COMPOUND fragmentation.

Fragmentation for Load and Query Performance Needs.

```
CREATE TABLE nhl_player_stats
( player_last_name varchar(30), player_first_name varchar(30), player_number integer, nhlpa_player_id integer,
  team char(3), season char(8), goals integer, assists integer, plus_minus integer, pim integer )
fragment by expression (
partition tor20082009 (team = 'TOR' AND season = '20082009') in dbspace1,
partition tor20072008 (team = 'TOR' AND season = '20072008') in dbspace1, ...
...
partition fla20082009 (team = 'FLA' AND season = '20082009') in dbspace1,
partition fla20072008 (team = 'FLA' AND season = '20072008') in dbspace1, ...
...
partition det20082009 (team = 'DET' AND season = '20082009') in dbspace2,
partition det20072008 (team = 'DET' AND season = '20072008') in dbspace2,
...
partition tpa20072008 (team = 'TPA' AND season = '20082009') in dbspace2,
... )
extent size 1000 next size 500 lock mode page;
```

Steps for Load with Minimal Downtime

Create a table, `nhl_new_stats`, fragmented by team in the dbpaces designated for each team.

Perform your ETL insert into there.

Detach each partition to tables named for the team and season
- `ott20082009`.

The last partition can't be detached – but it contains only data for one team...rename it – `was20082009`.

Detach and drop each old partition from the target table.

Attach each new partition as its replacement.

- It's a lot of work, but if your partition names are the same as the data range it contains, it can be scripted nicely.

References / Further Reading

The INFORMIX online references are always a wealth of useful information.

Ajay Gupte has a great whitepaper online about Flexible Fragmentation

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0610gupte/index.html>

Thanks...and Questions

Thanks so much!

Any Questions?

Hey, What about my Indexes?

- Cannot alter fragment on tables with constraints.
- PK won't work, Unique Index will work fine.
- If you have constraints, you must drop and recreate them...you probably do already for bulk loads.
- Otherwise, only 2 reasonable options:
 - 1) Drop and recreate indexes
 - 2) Use Attached Indexes.

Attached Indexes

- If you want to keep your attached indexes:
 - Detaching a data fragment detaches the corresponding index data, still takes under a second.
 - Adding a new data fragment returns instantly.
 - Attaching a table as a new fragment will take time while it rebuilds a corresponding index.

Detached Indexes

Can be used and are quite good if you fragment them by the same scheme as your table.

Can only be maintained effectively by dropping and recreating your indexes: adding a new data partition or attaching a new data partition will automatically create the new index fragment as attached, no matter how many times I tried creating an index fragment before or afterwards.

2009 IIUG Informix Conference

Session: E08

Data Fragmentation for
Load and Query Performance

J. Warren Donovan

US Courts

Warren_Donovan@ao.uscourts.gov