NETEZZA Question Everything™

- Getting data Distribution right
- Space Maintenance
- Performance Guidelines
- Security
- Backup & Recovery
- Summary – Next Steps

NETEZZA Question Everything™

Data Distribution

Distribution Keys

Each table in a Netezza RDBMS database has a distribution method.

Syntax

CREATE TABLE table_name (column definitions)

[DISTRIBUTE ON (column [, ...])] **/* hash algorithm */**

or **[DISTRIBUTE ON RANDOM];** **/*** round-robin **algorithm */**

The DISTRIBUTE ON clause is optional. If not specified, a hash distribution method will be chosen on the following basis:

1. **The first column in the table will be used.**

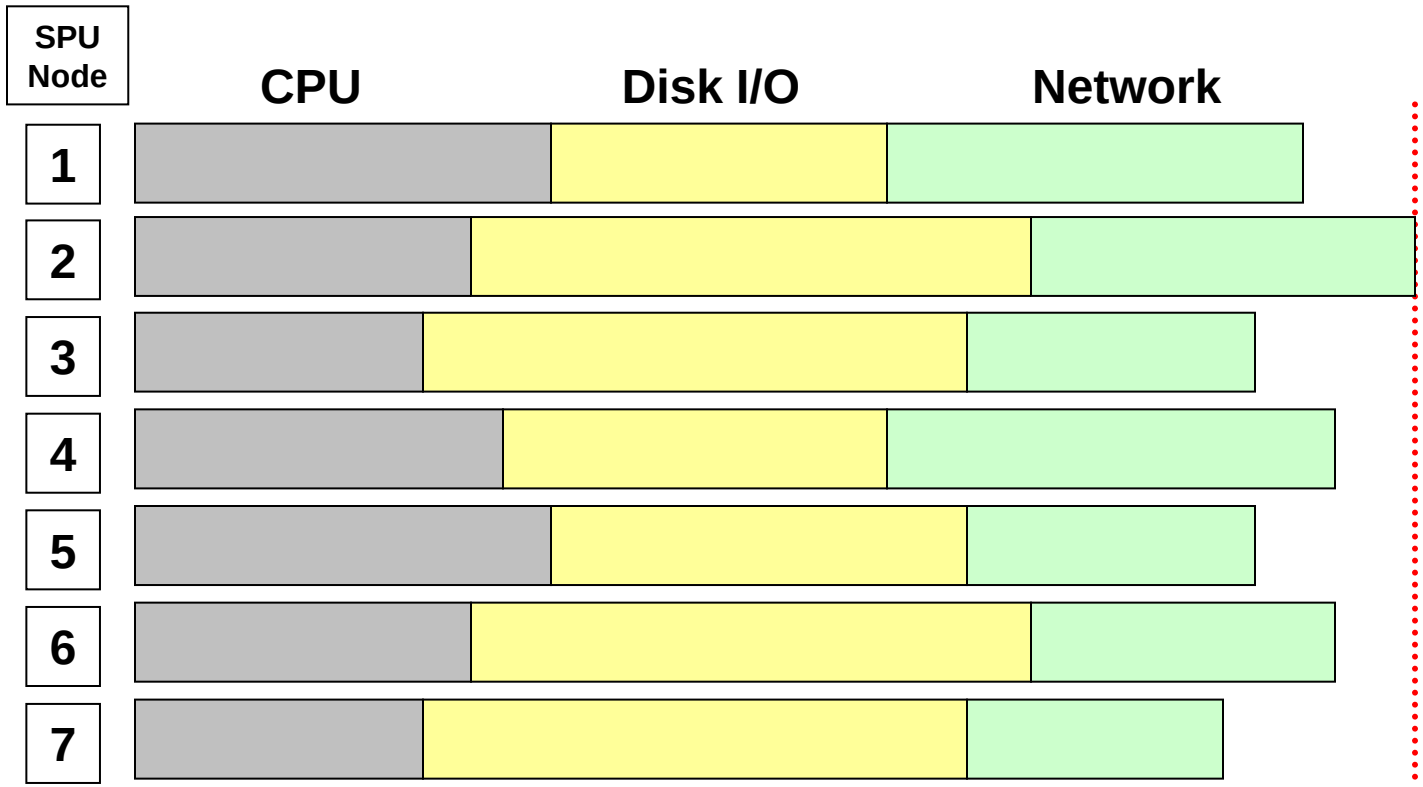
From 1 to 4 columns can be combined to make up the DISTRIBUTE ON clause.

Hashing A Distribution Key

The system distributes rows with the same hash value to the same SPU

- The distribution key is hashed to a numerical value.
- Based on the hash value, it is assigned to a specific ‘bucket’.
- Each SPU then receives a specific subset of the ‘buckets’

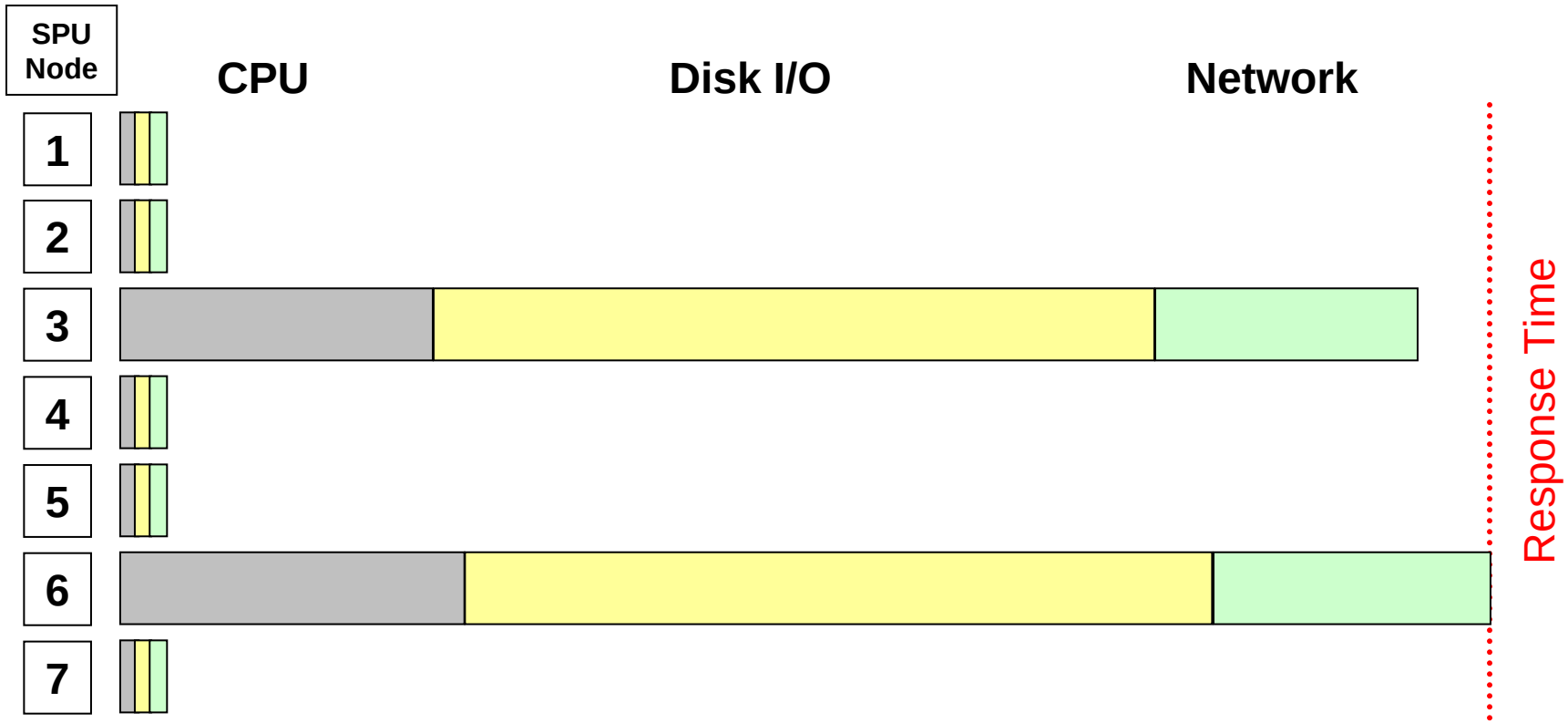
Distribution Keys



Response Time is affected by the completion time for **all** of the nodes in the MPP array.

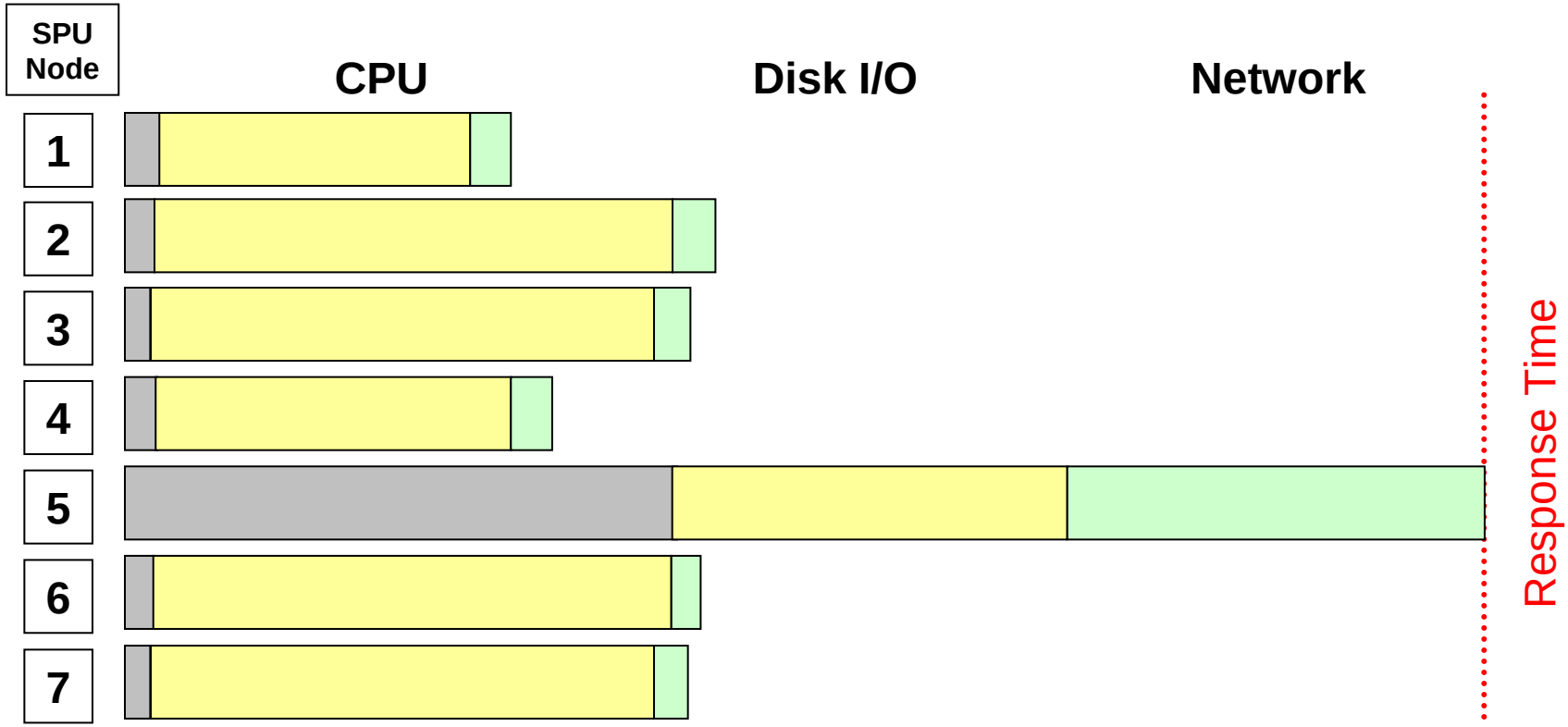
Parallel processing is more efficient when you have distributed table rows evenly across the SPUs. Each node has a comparable # of records, and performs approximately the same amount of processing.

Data Skew



The more distinct the distribution key values, the better.

Processing Skew



Don't use a DATE column as your distribution key. Though this might distribute the rows evenly across all of the SPUs, many queries are based on a date range. Parallel processing won't come into play when all of the records to be processed for a given query/date range are located on just a single SPU

Distribution Keys

- **Choose a column (or columns) with high cardinality.**
- **Tables used together should use the same column/distribution key.**

  e.g., In an order system application, choose the customer id as the distribution key for both the customer table and the orders table. This allows for co-located joins of the two tables.
- **Choose as few columns as possible for the distribution key to optimize the generality of the selection.**
- **If no good set of columns is available for the distribution key, use DISTRIBUTE ON RANDOM. Don't combine columns simply in an attempt to generate your own unique key.**
- **Columns used in the DISTRIBUTE ON clause can not be updated.**
- **For your large fact tables, do try to find a good distribution key as it will have an affect query performance. Don't choose RANDOM because it's the easy choice.**

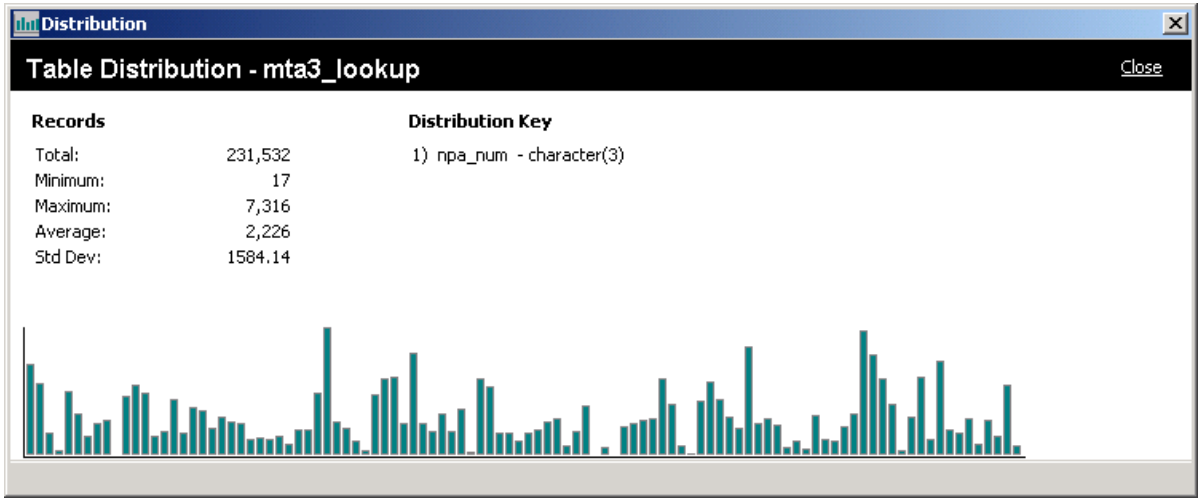
Distribution Keys

You can check the skew/distribution of an existing table with the following SQL

```
SELECT      datasliceid,
            COUNT(datasliceid) AS "Rows“
FROM tablename
            GROUP BY datasliceid
            ORDER BY "Rows";
```

```
 datasliceid |  Rows
-------------+---------
           3 | 6919541
          25 | 6919937
          20 | 6919952
          21 | 6920181
           5 | 6920302
          16 | 6920461
          24 | 6920592
           1 | 6920990
           2 | 6921629
           9 | 6921832
```

Checking Skew -- Using NzAdmin

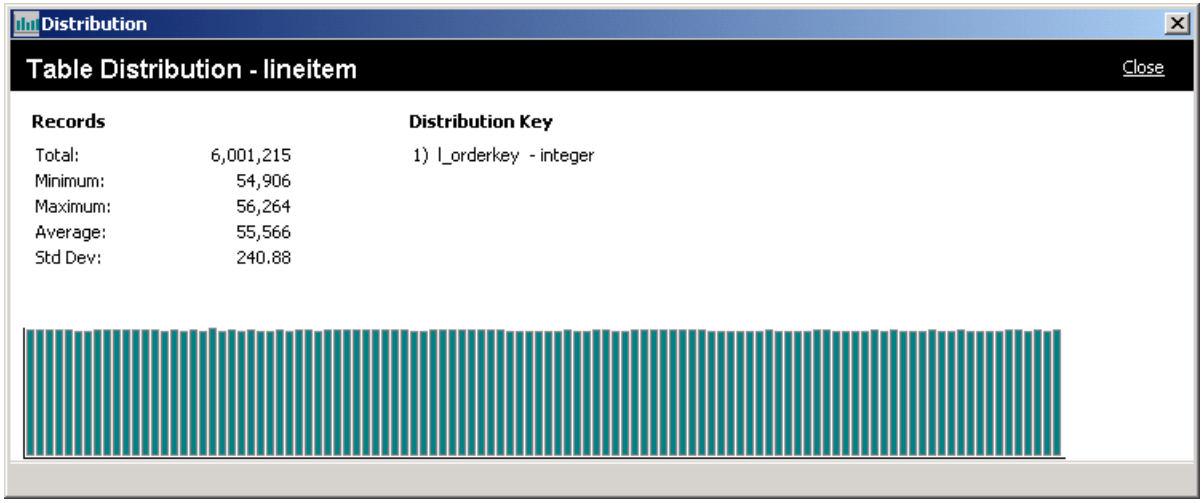


A bad distribution.

The user chose the wrong column(s) to distribute the data on.

Note: In this particular case, the user chose the first column in the table as the distribution column. Not the right decision.

Checking Skew -- Using NzAdmin



Shows a good distribution of records across all of the SPUs in the system. Thus, the workload is evenly balanced across all of the hardware resources, and will result in optimal performance.

Tuning Your Distribution Key Selection

1. **Create a table, Load the data, Check the skew**
   - + *Simple*
   - + *Allows you to also check data types & sizes for adjustment*
   - - *Requires you to reparse/reload the records to test each change*

2. **Create a table, Load the data, Check the skew**

   Create a newtable with a different distribution key

   INSERT INTO *newtable* SELECT * FROM *oldtable*;

   - + *Records have already been parsed and loaded into the oldtable*
   - + *Redistribution of the records will be fast SPU <--> SPU distribution*
   - + *Allows you to also check data types & sizes for adjustment*
   - - *Uses extra disk space because you have 2 copies of the table loaded*

Statistics

In order for the system to be able to create the best execution plan for a query, it must make some decisions based on what it knows about the database table(s) being accessed.

Without statistics the system must guess -- and it will usually guess wrong -- resulting in sub-optimal queries with longer run times.

The statistics collected includes

Per Table:	**Number of records**
Per Column:	**Minimum Value**
	Maximum Value
	# Of Distinct (non-NULL) Values
	# of NULL Values
	Reliability Indicator (ie, are they up to date?)

Statistics

Statistics are used for many purposes. For example:

The Planner

Given the number of records, the number of distinct values for a column, and assuming uniform distribution between the MIN and MAX values, the optimizer estimates the number of relevant rows and determines which is the smaller of two JOIN tables.

The Code Generator

The MIN and MAX values are used to determine the narrowest possible # of bits (32, 64, 128) to be used for arithmetic operations.

If there are any NULLs in a column, then the system must generate additional code that tests whether the current field is NULL and then handle it accordingly. Extra code = extra cpu cycles during execution time.

Statistics

Statistics are collected via the SQL statement "GENERATE STATISTICS …"

GENERATE STATISTICS;

--- All tables in a database

GENERATE [express] STATISTICS ON *tablename*;

--- A specific table (all columns)

GENERATE [express] STATISTICS ON *tablename* **(***colx***,** *coly***,** *colz***);**

--- Selected columns within a table

Generating "*Express*" statistics runs many times faster. The only difference is that it does an approximation in determining the number of distinct (non-null) values in each column. The end result is typically just as good.

Statistics

Statistics should be regenerated for a table whenever it changes by more than 5% (thru the nzload'ing of new records, or because of a bulk delete/ update of many rows)

Important for optimal handling of table joins

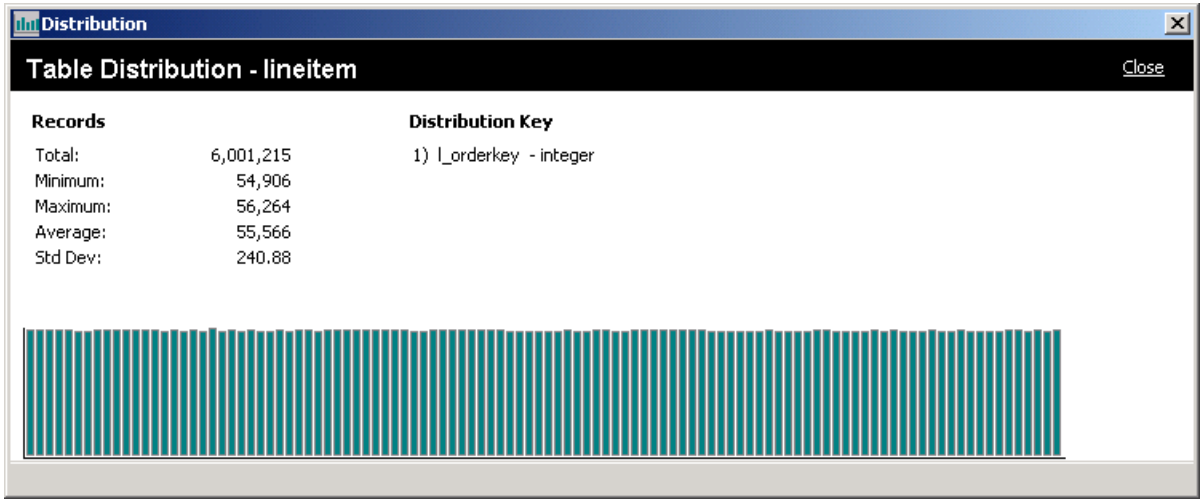
Of lesser importance when processing single table queries

You don't need to generate statistics for all columns in all tables.
You should do it for any columns that will be included in a

- **WHERE clause**
- **SORT clause**
- **GROUP BY/HAVING clause**

Don't forget temp tables -- they too will benefit from having proper statistics available for them. Note: When you do a "CREATE TABLE … AS …", the system will automatically generate statistics on the resultant table to insure that it has something available to use.

Zone Maps™



- **Rolling Data..Add a new day roll off an old day.**
  - Call records
  - Web logs
  - Financial transactions
- **1000 Days…no data skew.**
  - Why scan 999 days when the query only wants one day?

Zone Maps…Automatic Partitioning

- **Zone Maps**
  - Take advantage of inherent ordering of data
  - Within a SPU….
    - When stats are run, the zone map is created
    - For every column (Integers, timestamps, dates) in the table
      - Min and Max values per extent gathered
      - Each record is inserted into the zonemap for this table.
- **When a query runs, the SPU eliminates certain extents to scan**
- **Automatically configured**
  - During stats
  - During Loads
  - During inserts, updates, loads, and reclaims.

Maintaining Statistics Automatically

Table 10-6: Automatic Statistics

Command	Row counts	Min/Max	Null	Dispersion (estimated)	Zone maps
nzsql create table as	yes	yes	yes	yes	yes
nzsql insert	yes	yes	no	no	yes
nzsql delete	no	no	no	no	no
nzsql update	yes	yes	no	no	yes
nzreclaim	no	no	no	no	yes

Maintaining Statistics Automatically

- **The NPS system maintains certain statistics when you perform database operations.**
  - When you use the **nzsql** CREATE TABLE AS command, the system maintains the min/max, null, and estimated dispersion values automatically.
  - When you use the **nzsql** INSERT or UPDATE commands, the system maintains the min/max values for all non-character fields.
  - When you use the **nzreclaim** command to remove deleted records, the system leaves the min/max, null, and estimated dispersion values unchanged, and updates the zone map.

Testing

When working with *big data*, test your queries so you don't mistakenly bring back millions of rows

SELECT COUNT(*) …

Rather than actually returning *'n'* records, count them instead in order to get a sense as to the size of your result set.

… LIMIT *nnn*;

Limit the number of records to be returned to your application.

SELECT * FROM tablename WHERE state = 'VA' **LIMIT** 20000;

"LIMIT 0" will cause a query to be parsed but not actually run.

LAB: Skew

1. **Check the skew on your existing tables using nzsql queries**
2. **Choose one of the tables with the worst skew**
   - > Create a new table with an alternate distribution key
   - > Copy the records from the old table into the new table
   - > Compare the resulting skew
3. **For this same table**
   - > Check the INTEGER columns to see if they are appropriate
   - > Check the CHAR columns to see if they can be adjusted
   - > Check all columns to see if they contain any NULL values
4. **Based on your findings, adjust the DDL and recreate/populate the table.**

LAB: Generate Statistics

Generate statistics on your database.

Generate statistics on one table.

Generate statistics on one column.

Bonus?

What are the SQL statements one would use to generate the statistics for a given column?

LAB: Generate Statistics syntax

Command: GENERATE STATISTICS
Description: Generates statistics on a database, table, and columns within a table.
Syntax:
GENERATE STATISTICS [ON table [(column_name [, ...])]]

GENERATE EXPRESS STATISTICS ON table [(column_name [, ...])]

NETEZZA Question Everything™

Unloading External Tables

External Tables: Examples

- **create external table ‘filename‘ (or table_name) as select * from table_name;**
- **Unload data from database table to a flat file:**

  insert into ext_table_name select * from table_name;

- **Unload selected table data to the file.**
  Insert into ext_tbl select * from tbl where col_name=‘restriction’ ;

- **RemoteSource**

```
> CREATE EXTERNAL TABLE 'C://temp/foo.txt'
> USING( format 'text'
>        compress false
>        delimiter ''
>        remotesource 'odbc'
>        )
> AS
>  select …
```

External Tables: Unload

```
-- FEED SAS with data
CREATE EXTERNAL TABLE sqlresults
(
        Subacct_num   integer,
        total_seconds Int4
)
USING (DATAOBJECT ('/sas/account.unl') DELIMITER '|');


INSERT INTO sqlresults
-----
----- original SELECT statement follows
-----
SELECT
        Subacct_num,
        Int4(total_seconds)
FROM
        Bus_dt as b,
        Detail_usage_daily_load as dud
WHERE
            b.bus_dt = dud.bus_dt
        AND b.week_ending_dt =
        (       SELECT   week_ending_dt - 7
                FROM    bus_dt
                WHERE  bus_dt = date('2003-09-29')
        );
```

External Tables: Load & Unload

```
drop table foo;
drop table ext_foo;
create table foo (col1 integer, col2 integer);

insert into foo values (1, 1);
INSERT 0 1
insert into foo values (2, 2);
INSERT 0 1

create external table ext_foo SAMEAS foo
using (DATAOBJECT ('/tmp/foo.unl') DELIMITER '|' datestyle 'MDY');

insert into ext_foo select * from foo;
INSERT 0 2
insert into foo select * from ext_foo;
INSERT 0 2
insert into foo select col1+10, col2+50 from ext_foo where col1=1;
INSERT 0 1
```

```
select count(*) from foo;
 count
-------
     5
(1 row)

select * from foo;
 col1 | col2
------+------
    1 |    1
    2 |    2
    1 |    1
    2 |    2
   11 |   51
(5 rows)
```

NETEZZA Question Everything™

Recovering Space

nzreclaim

- **You can wind up with logically deleted records in a database table under the following circumstances:**
  - An aborted INSERT or nzload operation
  - An aborted UPDATE operation
  - DELETE
  - UPDATE (the original version of the tuple is marked as deleted, and a new record is added to the table)
- **These logically deleted records**
  - take extra disk space
  - take extra time for the table to be scanned
- **The records can be physically deleted by running nzreclaim**

nzreclaim -- Usage

Command line arguments include:

```
-db <db>              database name
-allDbs               reclaim entire NPS system

-t <tbl>              table name
-allTbls              reclaim all tables in specified database

-showSpuInfo          display status information on a per SPU basis
-blocks               run a block-based reclaim
-records              run a record-based reclaim (full space reclaim)

-scanOnly             run a reclaim scan to generate statistics
-scanforBlocks        report space available for block reclaim
```

nzreclaim -- Example

```
nzreclaim  -db hyperion  -allTbls  -scanOnly

                           0         25        50        75       100
 Table:       Start Time:  |-+-|-+-|-+-|-+-|-+-|-+-|-+-|-+-|-+-|-+-|
  netezza_ray     14:55:37 |*************************************|
     hamarket     14:55:38 |*************************************|
 haproductdim     14:55:39 |*************************************|
      hasales     14:55:39 |*************************************|
   hascenario     14:55:40 |*************************************|
       hatime     14:55:41 |*************************************|
Reclaim session completed successfully.


Database reclaim statistics:

ID    Name     Visible Rows Visible Size Reclaimable Rows Reclaimable Size (Est)
----- -------- ------------ ------------ ---------------- ----------------------
47764 hyperion 320          26620        100000           8400000

Table reclaim statistics:

ID    Name         Visible Rows Visible Size Reclaimable Rows Reclaimable Size (Est)
----- ------------ ------------ ------------ ---------------- ----------------------
47777 netezza_ray  0            0            0                0
47785 hamarket     20           1528         0                0
47798 haproductdim 286          24476        0                0
47811 hasales      0            0            100000           8400000
47826 hascenario   2            88           0                0
47835 hatime       12           528          0                0
```

Nzsql- Finding Rows Marked for Deletion

- **Set the deleted records variable:**
  - nzDB(admin)=> set show_deleted_records=1;
- Select the # rows marked for deletion and compare with total # rows.
- **nzDB(admin)=> select count(*) from table_name where deletexid <> 0;**
  count

  779165797
- Total # of rows in this table are:
  nzDB(admin)=> select count(*) from table_name;
  count

  2726880219
- **Deleted rows is about 28% of total # rows**
  - This table should be reclaimed

LAB: Reclaim Space

1. Pick the ‘lineitem’ table and delete half of the rows
2. Run an nzreclaim report on the table
3. Delete all the rows in lineitem
4. Reclaim all the space
5. Re-load the table using an External Table

How often to run nzreclaim?

Depends on:

- frequency of updates and deletes
- size of updates and deletes
- available space on system < 85% >

NETEZZA Question Everything™

PERFORMANCE TUNING

Guidelines

- **Pick the right data type and size**
- **For columns that you will be joining against, pick the same data type and size**
- **Specify “NOT NULL” for columns, whenever possible**
- **Choose a good distribution key**
- **Periodically (re)generate statistics for the tables**
  - Full statistics /Express Statistics
- **Watch out for intermediate data skew and take advantage of extent elimination through sampling**
  - Enable JIT statistics
- **Watch out for re-distribution or broadcast of larger table in joins**
  - Enable factrel planner if
- **nzreclaim tables that are often updated or deleted**

Guidelines cont.

- **Periodically re-order your tables to keep zonemaps current**
  - Especially true for very large transaction tables that go out of their natural date order due to deletes and updates

Pick The Right Data Type And Size

Every vote counts (except in Florida elections)

Every byte counts

- **Saving 4 bytes for a table with one record isn't worth your time and effort**
- **Saving 4 bytes for a table with 5B records is!**

If all you need is a DATE (4 bytes), use it -- rather than a TIMESTAMP (8 bytes)

If you need both a DATE (4 bytes) and TIME (8 bytes), consider using a combined TIMESTAMP (8 bytes) field instead

Use a smaller INTEGER size whenever possible

Use a smaller NUMERIC size whenever possible

NUMERICs are generally better than DOUBLE PRECISION/REAL numbers

DDL Analysis -- Integers

To determine the minimal size INTEGER that could be used

SELECT MIN (*colx*), MAX (*colx*) FROM *tablename*;

byteint (1 byte)	**-128**	**127**
smallint (2 bytes)	**-32,768**	**32,767**
integer (4 bytes)	**-2,147,483,648**	**2,147,483,647**
bigint (8 bytes)	**-9,223,372,036,854,775,808**	**9,223,372,036,854,775,807**

DDL Analysis -- NUMERICs

To determine the minimal size NUMERIC that could be used

SELECT MIN (*colx*), MAX (*colx*) FROM *tablename*;

precision <= 9	**4 bytes**
precision <= 18	**8 bytes**
precision <= 38	**16 bytes**

DDL Analysis -- CHARs

CHAR fields, with a length > 16, are represented on disk as VARCHARs

CHAR fields, with a length <= 16, take up a fixed number of bytes -- regardless of how many/how few characters are stored in each field.

Using a VARCHAR might save some space.
Note: VARCHARs add a 2 byte overhead to each field

```
SELECT      MAX ( LENGTH ( TRIM ( colx ) ) ),
            AVG ( LENGTH ( TRIM ( colx ) ) )
    FROM tablename;
```

If the MAX length is less than the CHAR size, consider adjusting the CHAR size down. If the AVG length + 2 is less than the CHAR size, consider using a VARCHAR instead.

Specify NOT NULL

Specify NOT NULL for columns whenever possible

Makes for simpler SQL if you don't have to worry about NULLs in your data

```
CASE WHEN colx IS NULL
        THEN . . .
        ELSE . . .
END
```

Requires fewer instructions for the system when processing each field

Requires less storage space on disk when you don't have to allow for NULLs

To see if a given column currently contains any NULL values

```
SELECT COUNT(*) FROM tablename WHERE colx IS NULL;
```

JIT Statistics

- **Automatically maintained by NPS if enabled**
- **JIT statistics use sampler scan functionality and leverage zone map information to collect the number of maximum extents on the SPU with the greatest skew, total number of extents, and estimated number of rows.**
- **JIT statistics do not eliminate the need to generate statistics**
  - They do not calculate dispersion values
- **To enable globally put the following lines in the /nz/data/postgresql.conf file. Then nzstop/nzstart**
  - enable_jit_stats = true
- **To enable at the session level**
  - Set enable_jit_stats = true; # note the difference in syntax

FactRel Planner

- **To enable globally put the following lines in the /nz/data/postgresql.conf file. Then nzstop/nzstart**
  - enable_jit_stats = false
  - enable_factrel_planner = true
  - factrel_size_threshold = 20000000 # any table or intermediate result set participating in the join that has 20 million rows or more is considered a fact table and will not be broadcast.
- **To enable at session level**
  - Set enable_jit_stats = false;
  - Set enable_factrel_planner = true;
  - Set factrel_size_threshold = 20000000;

NZSQA Engineering utility

- **Created by engineering to view internal workings of the system**
- **Use only under guidance of Netezza Customer Support**

Check for Processing Skew Using nzsqa

- **Processing skew may affect query performance and completion**
- **Use** nzsqa **to monitor for processing skew**
- **Run the** nzsqa responders –sys **command while the query is running**
  - There must be activity on the system else no information is returned.
- **The output wraps but the essential information to focus on is the line that contains the *Respond* keyword**
- **Immediately after the Respond keyword the *Count* number indicates how many SPUs are busy (responding)**
  - Following the *Count* is an indicator, either a 1 or 0 for each SPU
  - 1 indicates that the SPU is busy. 0 indicates that it is not active.
- **The Plan ID corresponds to plan ID for the query followed by the number of snippets in the plan and which snippet is being processed**

Data Skew during processing

- **Steps to determine if only 1 spu is providing data back for query**
- **Useful in determining intermediate data skew or a problematic spu**
- **First run**
  - nzsqa interactions –sys
  - This will help to determine if you need the second step.
  - CMD code = 206 (plan in execution), 215=rollback
  - If NumtoResp remains at “1” for many successive iterations of the command, then you should check the responder status
- **Run nzsqa responders –sys**
  - This will help to verify progress (or not) of the plan

Check for Processing Skew

Output Description:

CmdCode Command code

- 206 = Execute Plan
- 215 = Rollback

SpuId Target Data Slice Id (-1 = all)

NumToRsp Number of data slices remaining to respond

AZSCEsflLa

Flags: (A)borted (Z)ombie (S)uppress client ack (C)lient deletes (E)xpress acking (s)ent command (f)irst response received (l)ast response received from some data slice (L)ast response received from all data slices (a)bort received

```
nzsqa interactions -sys
Interaction Abort Messages: inUse = 0 available = 2048
Plan Abort Messages: inUse = 0 available = 548
Interaction Pool: inUse = 1 available = 2047

IntId IntAddr  CmdCode SpuId ClntId ClntUid TxId     PlanId    PlanAddr cliIntId NumToRsp Result   AZSCED-sflLa
----- -------- ------- ----- ------ ------- -------- --------- -------- -------- -------- -------- ------------
3181  1002a870 206     -1       525 2070    00010ca4 580 (1/1) 1640c4f0 2        82       16202fa0 000000-11000
```

Check for Processing Skew

- **Run nzsqa responders –sys**

```
IntId   Plan Id  CmdCode -      Count  1 - 8       9 - 16      17 - 24   25 - 32     33 - 40     41 - 48     49 - 56     57 - 64     65 - 72     73 - 80     81 - 88     89 - 96
    97 - 104  105 - 112
-----   --------  -------  ------      -----   --------    --------    --------    --------    --------    --------    --------    --------    --------    --------    --------
    --------  ---------
260   21 (2/2) 206     Respond 108   11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
    11111111 1111
260   21 (2/2)       XOFF    -     00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 0000
```

- **<elapsed time> nzsqa responders -sys**

```
IntId   Plan Id  CmdCode -      Count  1 - 8       9 - 16      17 - 24   25 - 32     33 - 40     41 - 48     49 - 56     57 - 64     65 - 72     73 - 80     81 - 88     89 - 96
    97 - 104  105 - 112
-----   --------  -------  ------      -----   --------    --------    --------    --------    --------    --------    --------    --------    --------    --------    --------
    --------  ---------
260   21 (2/2) 206     Respond 40    00001010 00000000 00100000 11000010 10011010 11000011 11010001 10010000 11100001 01001010 10000110 10101001
    11100110 0001
260   21 (2/2)       XOFF    -     00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    0000
```

- **<elapsed time> nzsqa responders –sys**

```
IntId   Plan Id  CmdCode -      Count  1 - 8       9 - 16      17 - 24   25 - 32     33 - 40     41 - 48     49 - 56     57 - 64     65 - 72     73 - 80     81 - 88     89 - 96
    97 - 104  105 - 112
-----   --------  -------  ------      -----   --------    --------    --------    --------    --------    --------    --------    --------    --------    --------    --------
    --------  ---------
260   21 (2/2) 206     Respond 1     00001000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    00000000 00000000 0000
260   21 (2/2)       XOFF    -     00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    0000
```

- Dataslice ID = 5

NETEZZA Question Everything™

SECURITY

Object Privileges

Permissions can be granted to

- **specific users**
- **specific groups**
- **PUBLIC (comparable to 'other/world' in Linux)**

Permissions are always additive

- **any permission you have been granted +**
- **any permission granted to any group you belong to +**
- **any permission granted to PUBLIC**

You cannot remove a privilege from a user who has been granted that privilege *as a consequence of* being a member of a group.

By definition -- the owner of each object has unrestricted access to that object.

Example: Table Permissions

grant list, select on my_dbms to the_ceo;

-- Grant the_ceo access to my_dbms

grant list on nation to the_ceo;

-- The list privilege only allows you to list (ie, see) the table name

grant list, select on region to the_ceo;

-- Basic access privileges -- allowing you to query a table

grant list,select,insert,update,delete on customer to the_ceo;

-- Grant this user additional privileges on this table

grant all on lineitem to the_ceo;

-- Grant this user all privileges on this table

Example: Table Permissions

```
\dpu a_worker
                  User object permissions for user 'a_worker'
 Database Name | Object Name | L S I U D T L A D B | D G U T V I B R C L G S H
---------------+-------------+---------------------+---------------------------
(0 rows)
```

object permissions

admin permissions

```
\dpu the_ceo
                  User object permissions for user 'the_ceo'
 Database Name | Object Name | L S I U D T L A D B | D G U T V I B R C L G S H
---------------+-------------+---------------------+---------------------------
 production    | customer    | X X X X X           |
 production    | lineitem    | X X X X X X X X X X |
 production    | nation      | X                   |
 production    | region      | X X                 |
(4 rows)

(L)ist (S)elect (I)nsert (U)pdate (D)elete (T)runcate (L)ock (A)lter (D)rop
a(B)ort

(D)atabase (G)roup (U)ser (T)able (V)iew (I)ndex (B)ackup (R)estore re(C)laim
(L)oad (G)enstats (S)ystem (H)ardware
```

Object Classes

Database

- **Table**
- **View**
- **System Table**

User

Group

Global in nature.
They are not tied to any particular database

If you grant access to a class of object, then it applies to all objects of that type in that database. If granted from within the top-level "system" database, it applies across all databases.

GRANT list, select ON table TO public;

/ A simple way to grant all user's read access to all tables */*

In Determining Access To An Object

Are you the owner of the object?

Then you have unrestricted access to the object.

Else, object permissions are considered:

Have you, or a group you are a member of (or PUBLIC) been granted any type of privileges on that object? Then that counts.

Else, class permissions within that database are considered:

Have you, or a group you are a member of (or PUBLIC) been granted any type of privileges to that object class? Then that counts.

Else, class permissions within the "system" database are considered:

Have you, or a group you are a member of (or PUBLIC) been granted any type of privileges to that object class? Then that counts.

Else -- you just weren't meant to access that object.

Admin Privileges

GRANT admin_privilege [, ...]
TO { PUBLIC | GROUP group | username } [WITH GRANT OPTION]

Admin privileges: **All Admin, Backup, Create Database, Create Group, Create Table, Create Temp Table, Create External Table, Create User, Create View, Genstats, Load, Manage Hardware, Manage System, Reclaim, Restore**

Since these are administrative privileges, they're not associated with any specific object

\h [cmd] Help on SQL syntax

To get help on a specific SQL command

nzsql

\h create user

```
Command:     CREATE USER
Description: Creates a new database user
Syntax:
CREATE USER username
    WITH PASSWORD password
     [ SYSID uid ]
     [ ROWSETLIMIT limit ]
     [ IN GROUP     groupname [, ...] ]
     [ VALID UNTIL  expiration ]
     [ SESSIONTIMEOUT limit ]
     [ QUERYTIMEOUT limit ]
```

\du \dU list users/User Groups

nzsql

\du

```
            List of Users
  username  | validuntil | rowlimit
------------+------------+----------
 admin      |            |
 amanda_lee |            |
 bill_adams |            |
 guest      |            |
```

\dU

```
List of Groups a User is a member
  username  |   groupname
------------+---------------
 amanda_lee | PUBLIC
 amanda_lee | manufacturing
 bill_adams | PUBLIC
 bill_adams | administration
 bill_adams | manufacturing
 guest      | PUBLIC
```

\dg \dG list groups/Group Users

nzsql

\dg

```
      List of Groups
   groupname    | rowlimit
----------------+----------
 administration |        0
 dba_group      |        0
 manufacturing  |        0
 mis_group      |        0
 public         |        0
 sec_group      |        0
```

\dG

```
  List of Users in a Group
   groupname    |  username
----------------+------------
 PUBLIC         | amanda_lee
 PUBLIC         | bill_adams
 PUBLIC         | guest
 administration | bill_adams
 manufacturing  | amanda_lee
 manufacturing  | bill_adams
```

NETEZZA Question Everything™

Backup & Recovery

nzbackup

nzbackup

Options:

-host <name/IP>	host name or IP address [NZ_HOST]
-v[erbose]	verbose
-db DATABASE	the name of the database to back up
-dir DIRNAME	the full path of the directory where the backup schema and data files are stored (For disk backup only)
-connector CONNNAME	the name of the connector to which to send the backup
-connectorArgs "name=value[:name=value[...]]" passthru	arguments for the connector
-differential	backup only what has changed since prior backup
-cumulative	backup only what has changed since prior full backup
-users	backup users, groups, and global permissions, rather than a database
-u USERNAME	the username for connecting to the database
-pw PASSWORD	the password for the user
-schema-only	backup only the schema without the data
-history	print a backup history report
-backupset ID	specify a backupset ID, as displayed in the backup history report

Nzbackup – example full backup

• Command to backup database “BB” in /tmp:
nzbackup -db bb -dir /tmp #all connection parameters set in env

• Result set stored in multiple locations:
Data File
/tmp/Netezza/spubox/BB/20080729171555/1/FULL/data

Schema File
/tmp/Netezza/spubox/BB/20080729171555/1/FULL/md

nzrestore

Options:

-[rR]ev or -V	print the software revision of this program
-v[erbose]	verbose
-db DATABASE	the name of the database to be restored
-dir DIRNAME	the full path of the directory where the backed up schema and data files are stored (For disk restore only)
-connector CONNNAME	the name of the connector from which to read the backup
-connectorArgs "name=value[:name=value[...]]" passthru	arguments for the connector
-backupset ID	override default backupset (see backup history report for format)
-sourcedb DATABASE	database name in backup archive
-npshost HOSTNAME	NPS hostname of source of backup archive
-tables <table list>	restore tables in space-separated table list only
-tablefile <table file>	restore tables in table file, one per line
-droptables	drop tables in table list before restoring
-suspendMViews	leave suspended materialized views affected by operation
-increment <ID>	restore up to the specified increment from the backup set
-increment NEXT	restore the next increment from the backup set
-increment REST	restore the remaining increments from the backup set
-lockdb	lock the database to prevent modifications between restore operations
-unlockdb	unlock the database
-users	restore users, groups, and global permissions, rather than a database
-u USERNAME	the username for connecting to the database
-pw PASSWORD	the password for the user
-schema-only	restore only the schema without the data
-allincs	with -schema-only, restore functions and aggregates in every increment
-contents	print a contents report of an existing backup archive (only) specify one of -db, -sourcedb, -backupset, and optionally -increment #
-history	print a restore history report
-incrementlist	print a report of the available backupsets and increments specify -connector, and optionally -npsHost and/or -sourcedb

Nzrestore - example

• Command to restore database BB to targer database BB2

nzrestore -db bb2 -sourcedb BB -dir /tmp

“Restore of increment 1 from backupset 20080729171555 to database 'bb2' committed.”

Reference: NPS sytem admin guide

NETEZZA Question Everything™

SUMMARY AND REVIEW

Summary

nzstart	**Launch the NPS system** ***(It should already/always be up)***
nzstop, nzsystem, nzadmin	**Stop NPS**
Nzsystem or nzadmin	**Pause, resume, restart NPS**
nzsql or NzAdmin	**CREATE DATABASE xxx;** **CREATE TABLE yyy (. . .)** **[DISTRIBUTE ON (zzz)]** **[DISTRIBUTE ON RANDOM];**
nzload	**Load data**
nzsql or NzAdmin	**GENERATE STATISTICS;**
nzsql / BI Tool	**Run your queries**

Helpful resource web sites

Netezza Support	**www.netezza.com**
Netezza Developer Network	**www.netezza.com/ndn**
Netezza Education	**www.netezza.com/support/education.aspx**
Netezza Community (enzee)	**www.netezza.com/community**

Next Steps

NETEZZA Question Everything™

- nz_health
- queryHistory
- Best Practices/(nz_best_practice)
- nzAdmin
- Workload Management
- Hardware Monitoring & nz Events