
Pivotal
A NEW PLATFORM FOR A NEW ERA

Physical Design Considerations in GPDB

Pivotal® Greenplum Database

Agenda

- Introduction
- Identify optimal distribution key
- Check for data skew
- Table partitioning: benefits, when to partition
- Select appropriate data types
- Define constraints
- Put this into practice in the lab

Pivotal

Key Design Considerations

The following are key design considerations to account for:

- Data distribution and distribution key selection
- Checking for data skew
- To partition or not
- Choosing appropriate data types
- Defining constraints

Pivotal

Data Distribution

In defining distribution methods, there are two options:

- Column distribution with the `DISTRIBUTED BY` clause
- Random distribution with the `DISTRIBUTED RANDOMLY` clause

```
CREATE TABLE  tablename  (
column_name1      data_type  NOT NULL,
column_name2      data_type NOT NULL,
column_name3      data_type NOT NULL DEFAULT default_value,
. . . )
    [DISTRIBUTED BY (column_name)]          hash algorithm

    [DISTRIBUTED RANDOMLY]              random algorithm
```

Note: Every table in the Greenplum database has a distribution method, whether you select it or not.

Pivotal

Distribution Key Considerations

Consider the following:

- A distribution key can be modified, even after the table has been created.
- If a table has a unique constraint, it must be declared as the distribution key.
- User defined data types and geometric data types are not eligible as distribution keys.

Note: You should choose a distribution key with high cardinality

Pivotal

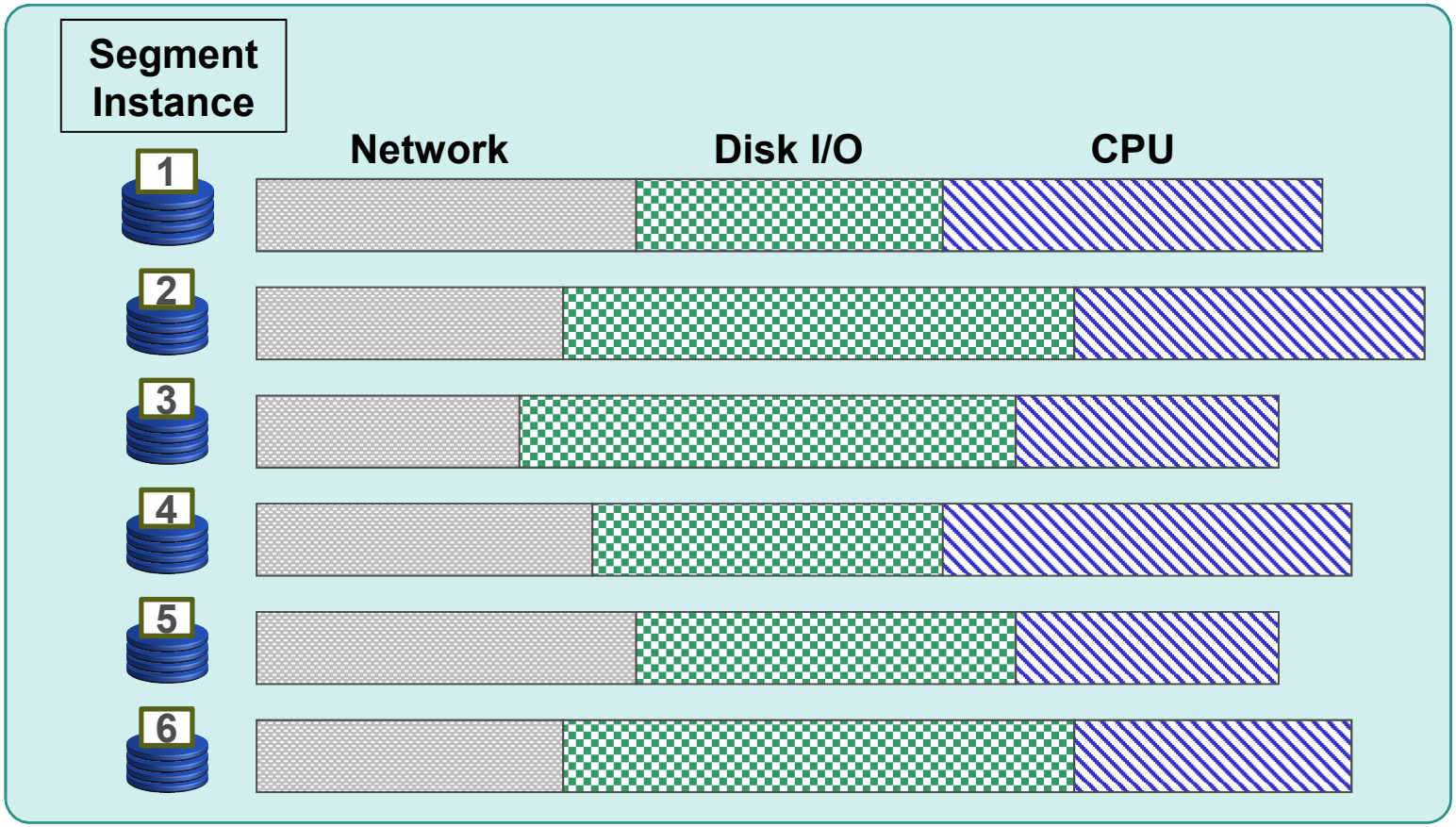
Default Distribution Use

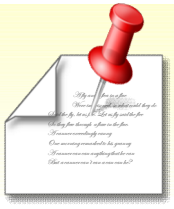
Consider the following when creating tables:

- If a table has a primary key and a distribution key is not specified, by default the primary key will be used as the distribution key.
- If the table does not have a primary key and a distribution key is not specified, then the first eligible column of the table will be used as the distribution key.
- User defined data types and geometric data types are not eligible as distribution keys.
- If a table does not have an eligible column then a random distribution is used.

Pivotal

Distributions and Performance

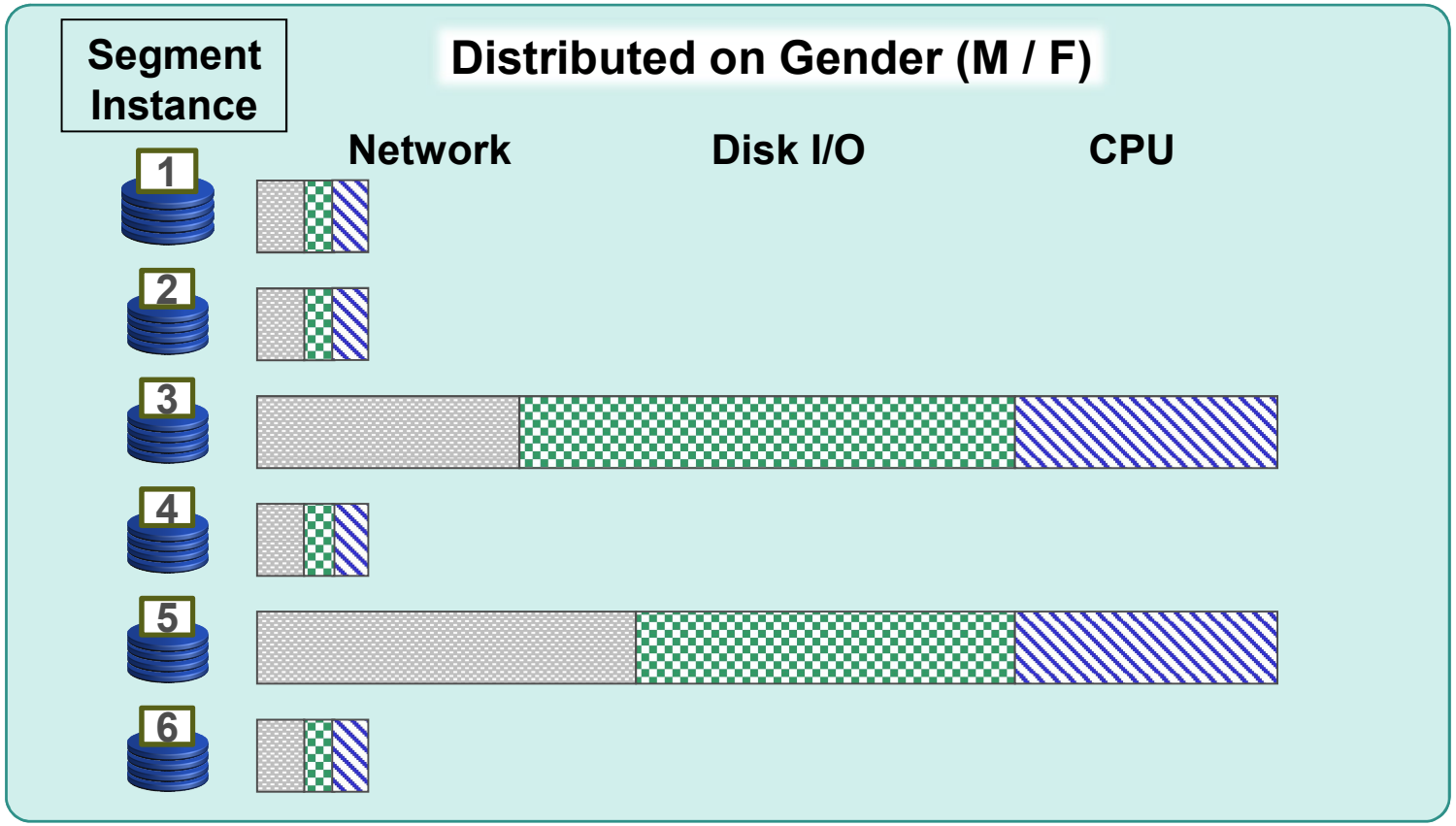


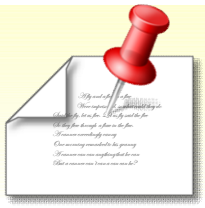


Note: To optimize performance, use a hash distribution method (`DISTRIBUTED BY`) that distributes data evenly across all segment instances.

Pivotal

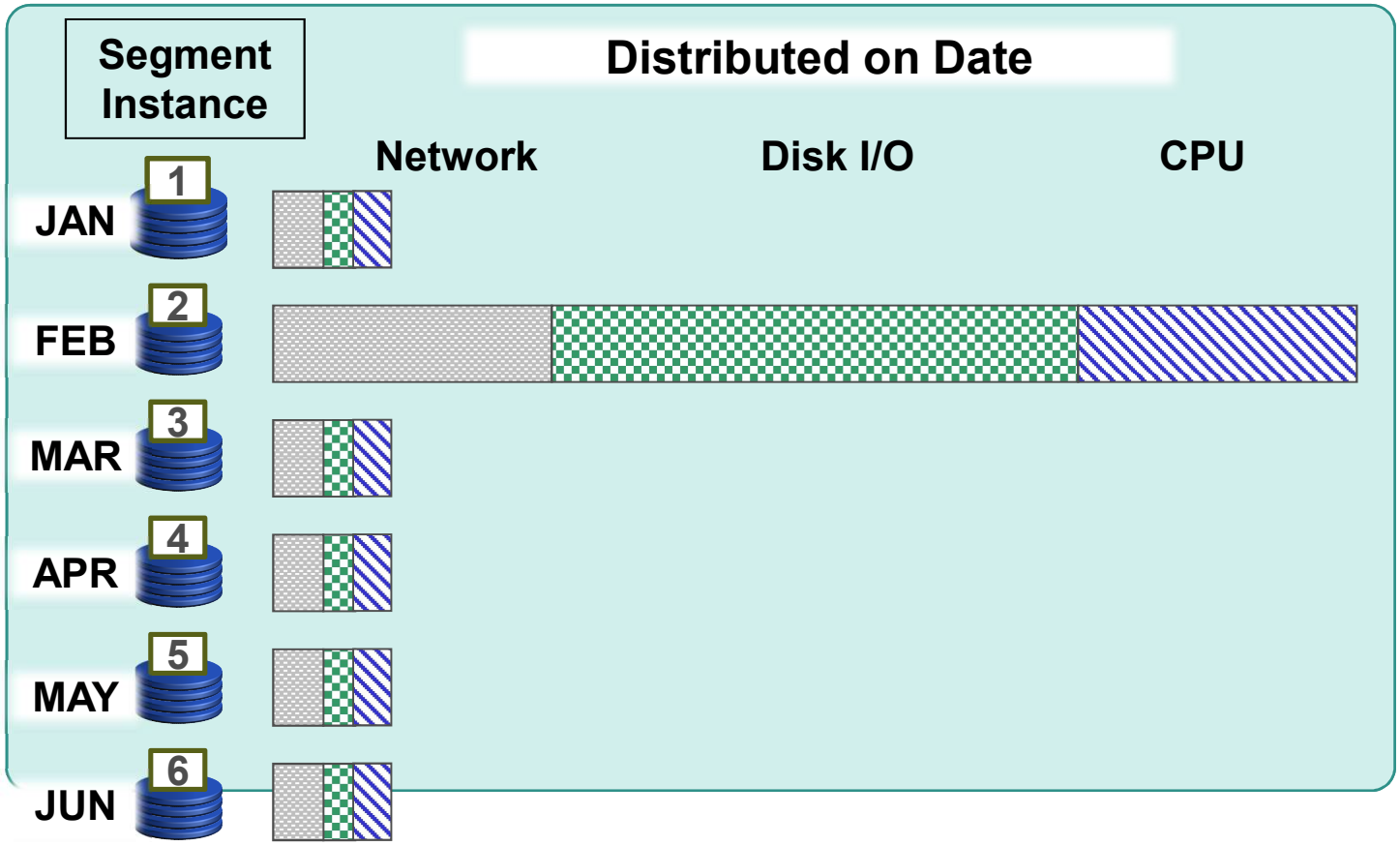
Hash Distributions and Data Skew





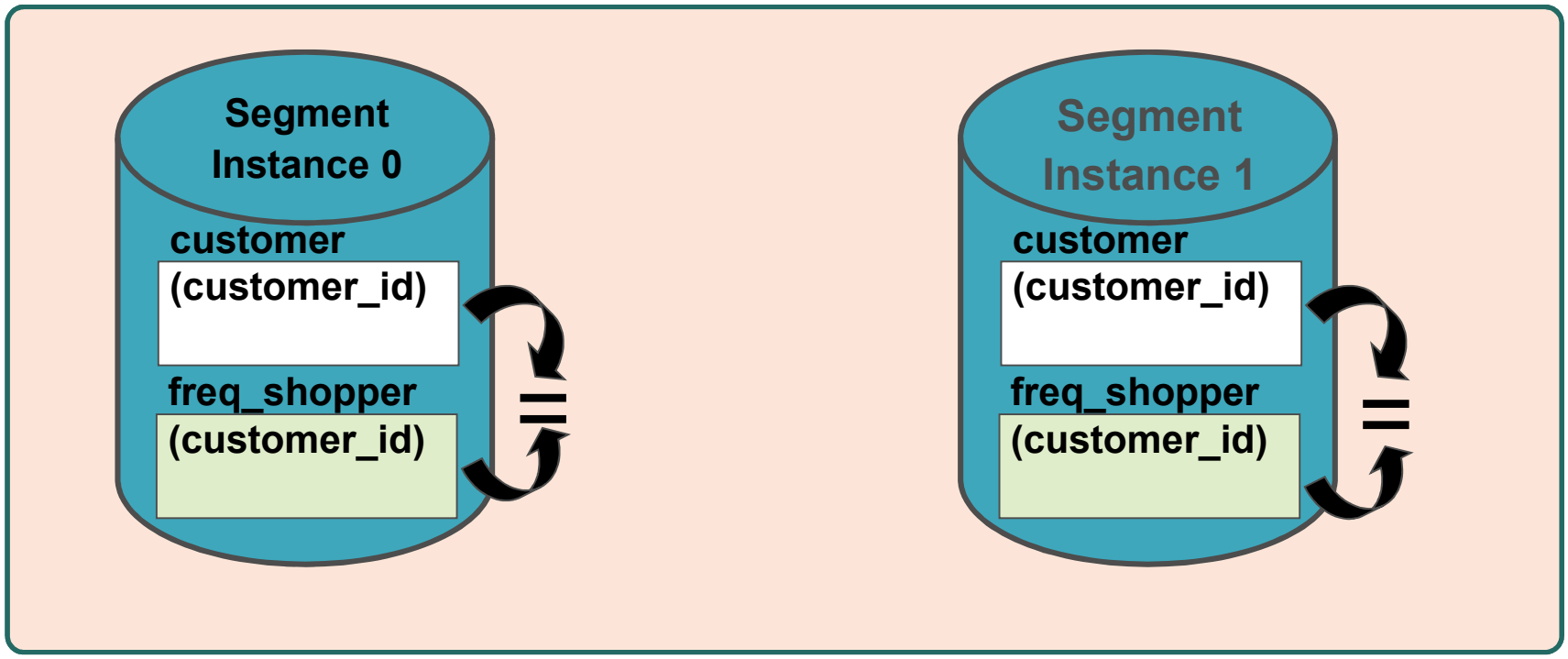
Note: Select a distribution key with unique values and high cardinality.

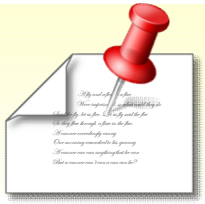
Hash Distributions and Processing Skew



Note: Select a distribution key that will not result in processing skew.

Using the Same Distribution Key for Commonly Joined Tables





Note: Optimize for local joins. Distribute on the same key used in the `JOIN`.

Pivotal

Caveat to Previous Slide: ***Use the Same Data Type*** for These Distribution Keys

If you use *different data types for identical values*:

- Different types, containing identical values, hash to different values
  - Resulting in like rows being stored on different segment instances
  - Requiring a redistribution before the tables can be joined

The following shows two distribution keys with different data types:

```
customer (customer_id)      745::int
freq_shopper (customer_id)     745::varchar(10)
```

Pivotal

Distributed With `DISTRIBUTED RANDOMLY`

The `DISTRIBUTED RANDOMLY` clause:

- Uses a random algorithm
- Requires a redistribute or broadcast motion for any query that joins to a table having this distribution policy
- Is acceptable for small tables such as dimension tables
- Is not acceptable for large tables such as fact tables

Pivotal

Distribution Best Practices

- Explicitly define a column or random distribution for all tables. Do not use the default.
- Use a single column that will distribute data across all segments evenly.
- Do not distribute on columns that will be used in the WHERE clause of a query.
- Do not distribute on dates or timestamps.
- Never distribute and partition tables on the same column.
- Achieve local joins to significantly improve performance by distributing on the same column for large tables commonly joined together.
- Validate that data is evenly distributed after the initial load and after incremental loads.

Pivotal

Check for Data Skew

Check for data skew with the following:

- `gp_toolkit` administrative schema offers two views:
  - `gp_toolkit.gp_skew_coefficients`
  - `gp_toolkit.gp_skew_idle_fractions`
- To view the number of rows on each segment, run the following query:

```
SELECT gp_segment_id, count(*)
FROM table_name GROUP BY gp_segment_id;
```

- Check for processing skew with the following query:

```
SELECT gp_segment_id, count(*) FROM table_name
WHERE column='value' GROUP BY gp_segment_id;
```

Pivotal

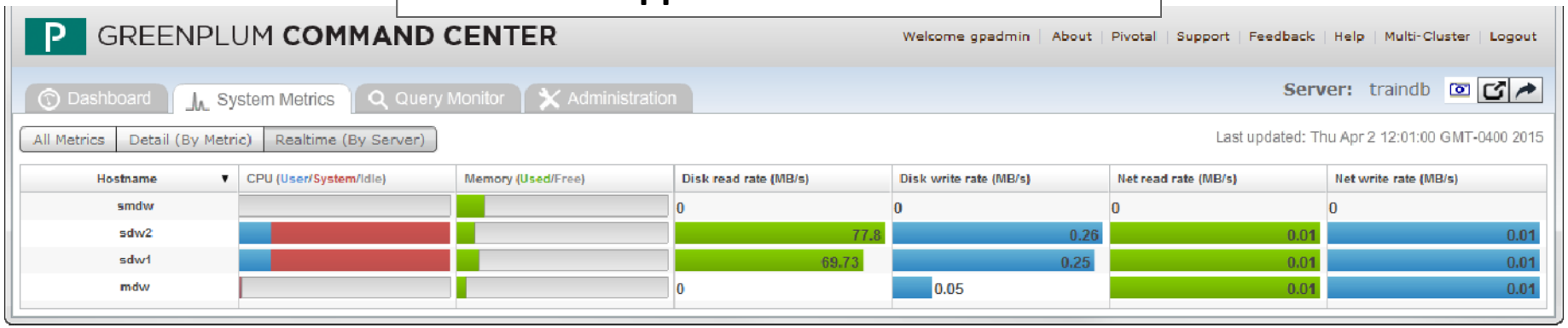
Processing and Data Skew Example

Example: Determining the processing and data skew of existing tables

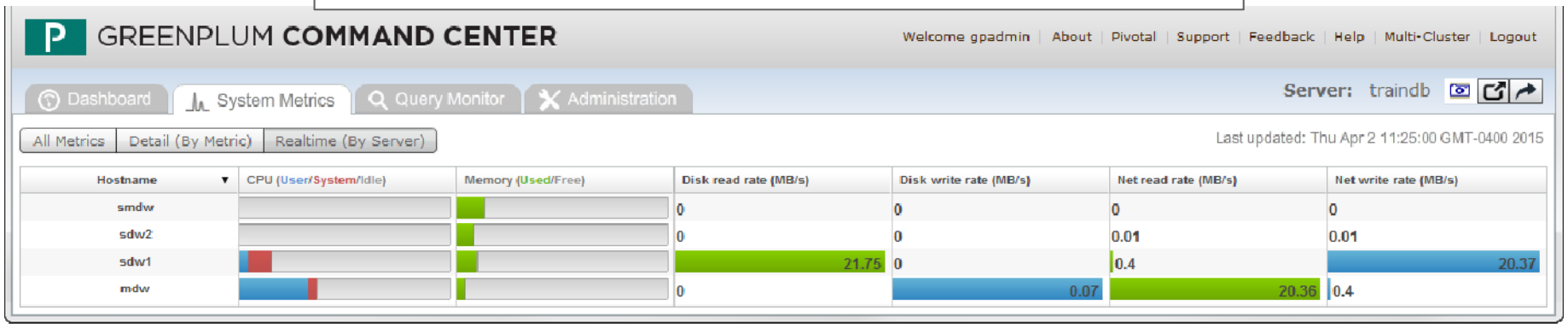
```
SELECT sifrelname, siffraction, skccoeff
from gp_skew_idle_fractions, gp_skew_coefficients
WHERE sifrelname=skcrelname AND siffraction > 0.1;
       sifrelname        |      siffraction       |          skccoeff
-------------------------+------------------------+---------------------------
 car_data                | 0.50000000000000000000 |    141.42135623730949000
 correlation             | 0.40000000000000000000 | 94.280904158206336667000
 golfevaluate            | 0.22222222222222222222 | 40.406101782088430000000
 golfnew                 | 0.22222222222222222222 | 40.406101782088430000000
 golftest                | 0.22222222222222222222 | 40.406101782088430000000
 old_regr_example        | 0.50000000000000000000 |    141.42135623730950000
 old_regr_example_test   | 0.50000000000000000000 |    141.42135623730951000
 sales_example           | 0.50000000000000000000 |    141.42135623730951000
 water_treatment_predict | 0.12037037037037037037 | 19.352396116684458526000
 winequality_white       | 0.10293040293040293040 | 16.226786893705185790000
```

Real-Time Data and Processing Skew in Command Center

Data access appears to be even distributed



Data access is skewed heavily to one segment server



Pivotal

Redistribute Using `ALTER TABLE`

Use the `ALTER TABLE` command to:

- Redistribute on the *current* policy

```
ALTER TABLE sales SET WITH (REORGANIZE=TRUE);
```

- Redistribute with a *new* distribution key

```
ALTER TABLE sales SET DISTRIBUTED BY (column);
```

Pivotal

Why Partition a Table?

The following are reasons to partition a table:

- Provide more efficiency in querying a large table, by using partition elimination to avoid full table scans
- Without the overhead and maintenance costs of an index on a date column, for example
- To handle increasing volumes of data that are not needed by the average query
- Allow instantaneous dropping of older data and simple addition of newer data
- Supports a rolling *n* period methodology for transactional data

Pivotal

Candidates for Partitioning

The following date-related tables make excellent candidates for table partitioning:

- Fact tables with dates
- Fact tables based on region (e.g. "EMEA", "APAC", ...)
- Transaction tables with dates
- Activity tables with dates
- Statement tables with numeric statement periods (YYYYMM)
- Financial summary tables with end of month dates

Pivotal

Table Partitioning Best Practices

Use table partitioning:

- On large distributed tables to improve query performance.
- If the table can be divided into rather equal parts based on a defining criteria, such as by date.
- If the defining partitioning criteria is the same access pattern used in query predicates, such as

```
WHERE txn_date = '2015-05-14'
```

- If possible, avoid using default partitions in your design (they are *always* scanned).

Pivotal

Creating Partitioned Tables

The following shows how to create the customer partition table using an interval:

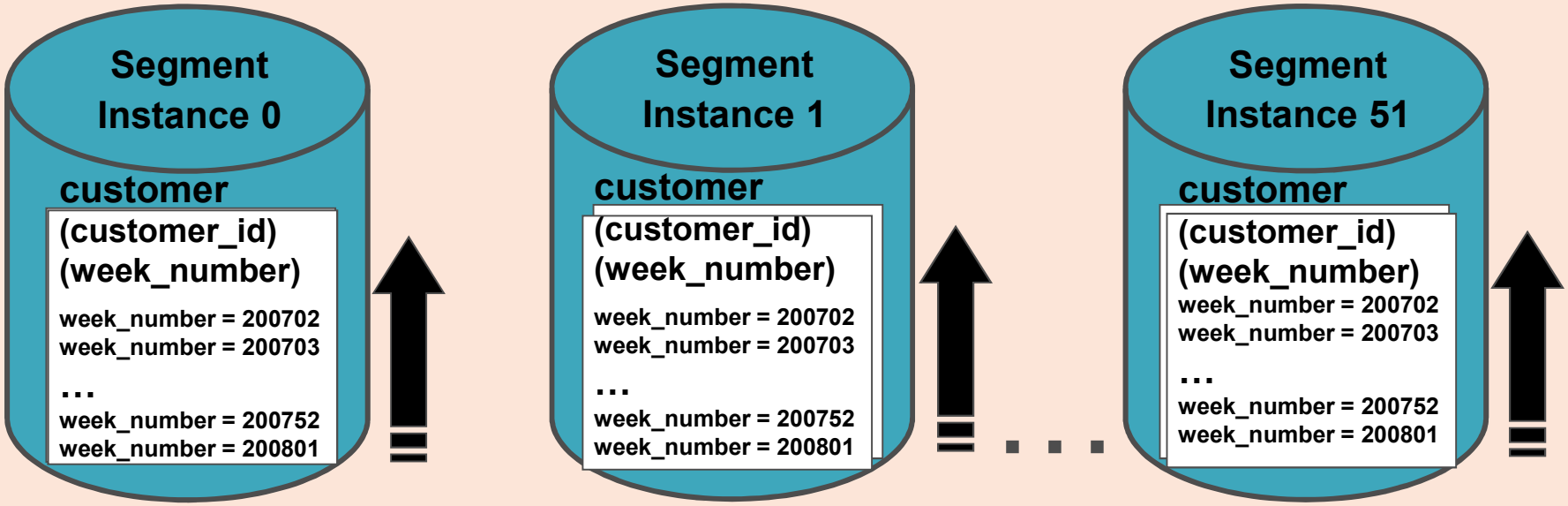
Example: Create a partition table using an interval

```
CREATE TABLE customer (
customer_id       INT,
week_number       INT,
. . .
) DISTRIBUTED BY (customer_id)
PARTITION BY RANGE ( week_number )
(START (200701) END (200752) INCLUSIVE EVERY (1));
```

Pivotal

Maintaining Rolling Windows

Use ALTER TABLE with ADD PARTITION clause to add a new child table



Use ALTER TABLE with DROP PARTITION clause to delete an old child table

Data Type Best Practices

The following should be considered when choosing data types:

- Use data types to constrain your column data:
  - Use character types to store strings
  - Use date or timestamp types to store dates
  - Use numeric types to store numbers
- Choose the type that uses the least space:
  - Do not use `BIGINT` when `INTEGER` is sufficient
  - Use identical data types for columns used in join operations
  - Converting data types prior to joining leads to unnecessary data movement

Pivotal

Table and Column Constraints

The following code creates:

- A table with a `CHECK` constraint:

```
CREATE TABLE products
( product_no integer, name text,
    price numeric CHECK (price >= 0) );
```

- A table with a `NOT NULL` constraint:

```
CREATE TABLE products
( product_no integer NOT NULL, name text NOT NULL,
    price numeric );
```

Pivotal

Table and Column Constraints (Cont)

- A table with a `UNIQUE` constraint:

```
CREATE TABLE products
( product_no integer UNIQUE, name text, price numeric)
DISTRIBUTED BY (product_no);
```

- A table with a `PRIMARY KEY` constraint

```
CREATE TABLE products
( product_no integer PRIMARY KEY, name text, price
numeric)
DISTRIBUTED BY (product_no);
```

Pivotal

Primary Key and the Distribution Key

Consider the following:

- A primary key is a logical model concept which allows each row to be uniquely identified.
- A DISTRIBUTED BY key is a physical Greenplum Database concept which determines where, within the cluster, each row is physically stored.
- The distributed by key can differ from the primary key, as long as it is a *left-subset* of the primary key.
- This applies to UNIQUE constraints as well.

Pivotal

Review

- Identify optimal distribution key
- Check for data skew
- Table partitioning: benefits, when to partition
- Select appropriate data types
- Define constraints
- Put this into practice in the lab

Pivotal


Pivotal
A NEW PLATFORM FOR A NEW ERA