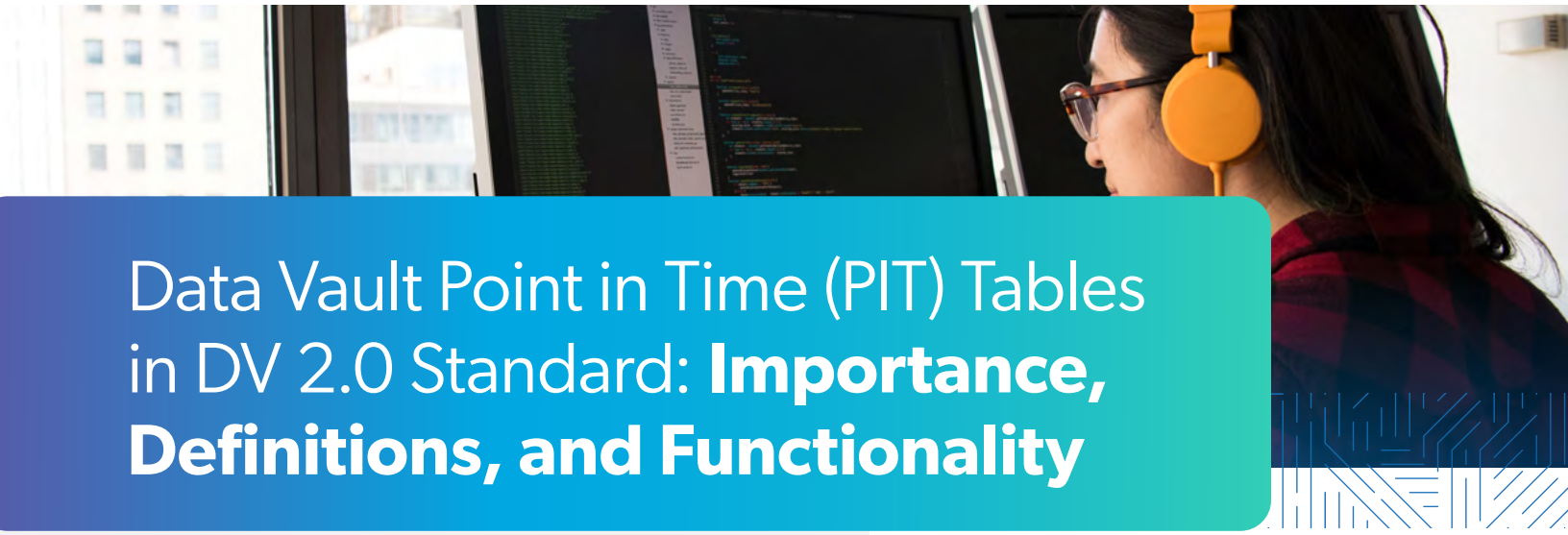
Resultant



Data Vault Point in Time (PIT) Tables in DV 2.0 Standard: **Importance, Definitions, and Functionality**

CONTENTS

INTRODUCTION

Point in Time (PIT) tables in the business vault will greatly reduce the amount of compute time needed in the raw and business data vaults when it's necessary to find the correct set of satellite records for a particular point in time. By implementing PIT tables, points in time are set in advance, eliminating the need for reporting users to find the most appropriately matching satellite record per business key per requested point in time. The PIT tables anticipate and figure out this information ahead of time, saving valuable manpower, time, and budget. In this whitepaper, you will learn about the journey of PIT tables, how to create them, and their importance as part of your data vault implementation.

BACKGROUND

Raw data vaults consist of three object types: hubs, links, and satellites.

- **HUB:** Holds business keys in your enterprise. Each different kind of business key (clients, accounts, orders, etc.) has its own hub object.
- **LINK:** Holds all the relationships between the various hubs. One link may show how clients associate with accounts, a different link may show how clients associate to orders, and so on.
- **SATELLITE:** Holds all the attributes that describe the business key in a hub, or attributes that describe the relationship in a link. They are either hub satellites or link satellites. All satellites behave the same way, but a particular satellite instance cannot associate with a hub and link (or multiples of them) at the same time. The DV 2.0 standard is that a satellite object associates with just one hub or link object.

However, each hub or link object can associate with zero to infinity different satellite objects. It's a one-to-many relationship, not a many-to-many relationship between hubs or link and satellites. Multiple satellite objects can be associated with a particular hub or link for a multitude of reasons. While this article doesn't detail all those reasons, you can visit https://danlinstedt.com/ for more information on how and why.

One example reason for a hub or link to have multiple satellites is that the attributes in the satellites may have different frequencies of changes. If there are ten attributes for a hub and seven of them change infrequently, those seven attributes may be in one satellite. You would set the other three attributes up in a different satellite because their values change frequently. Continually copying the values of the seven infrequently changing attributes every time one of the three frequently changing attributes is modified would create a tremendous amount of unnecessary data duplication in the raw data vault.

Satellites are the objects in a raw data vault where **all** changes to data are tracked over time. It is exactly these changes to a satellite's attributes that drive the need for point-in-time (PIT) objects.

Assume we have a hub called *Sample_h* and it has three satellites: *Sample_SlowChg_sh, Sample_MedChg_sh*, and *Sample_FastChg_sh*

We name the satellites starting with the base hub name, then a *rolename* the satellite is performing, followed by sh so we know it is a satellite to a hub.

Here's the data definition language (DDL) for demonstration purposes of each satellite. Other possible data information could be included in satellites depending on the kind of satellite it is. These satellite fields will suffice to demonstrate the use of PITs.

SAMPLE_SLOWCHG_SH

Hk_Sample_h	CHAR(32)	HUB's hash key this SAT is associated to
Bk_Sample_h	VARCHAR(50)	business key of the Hub Hash key above
Attrib_A	VARCHAR(10)	
Attrib_B	VARCHAR(25)	
LoadDate	DATETIME	When did the record get loaded to the DV
RecordSource	VARCHAR(1024)	Which source system provided the data
HashDiff	CHAR(32)	Has the record changed since previous

SAMPLE_MEDCHG_SH

Hk_Sample_h	CHAR(32)	HUB's hash key this SAT is associated to
Bk_Sample_h	VARCHAR(50)	business key of the Hub Hash key above
Attrib_C	VARCHAR(5)	
Attrib_D	VARCHAR(12)	
LoadDate	DATETIME	When did the record get loaded to the DV
RecordSource	VARCHAR(1024)	Which source system provided the data
HashDiff	CHAR(32)	Has the record changed since previous

SAMPLE_FASTCHG_SH

Hk_Sample_h	CHAR(32)	HUB's hash key this SAT is associated to
Bk_Sample_h	VARCHAR(50)	business key of the Hub Hash key above
Attrib_E	VARCHAR(6)	
Attrib_F	VARCHAR(8)	
LoadDate	DATETIME	When did the record get loaded to the DV
RecordSource	VARCHAR(1024)	Which source system provided the data
HashDiff	CHAR(32)	Has the record changed since previous

DATA VALUE SCENARIOS IN THE SATELLITES

SAMPLE_SLOWCHG_SH

Hk_Sample_h	Bk_Sample_h	Attrib_A	Attrib_B	LoadDate	RecordSource	HashDiff
0x00000....000	GhostValue	Ghost	Ghost	1/1/1900 00:00:00 AM	Internal	0x36234....235
0x7F587....895	Biz_Value_123	ABC	DEF	10/5/2020 2:34:56 PM	SystemA	0x12385....284
0x7F587....895	Biz_Value_123	XYZ3	ABC	3/16/2021 11:32:23 PM	SystemA	0x61092....302

SAMPLE_MEDCHG_SH

Hk_Sample_h	Bk_Sample_h	Attrib_C	Attrib_D	LoadDate	RecordSource	HashDiff
0x00000....000	GhostValue	Ghost	Ghost	1/1/1900 00:00:00 AM	Internal	0x36234....235
0x7F587....895	Biz_Value_123	CAT	BIRD	11/14/2020 9:16:45 AM	SystemB	0x82730....591
0x7F587....895	Biz_Value_123	TALL	TREE	1/30/2021 3:56:12 PM	SystemB	0x50367....622
0x7F587....895	Biz_Value_123	TALL	SHORT	2/1/2021 6:23:45 PM	SystemB	0x51128....990
0x7F587....895	Biz_Value_123	BIRD	CAT	4/27/2021 2:18:16 PM	SystemB	0x12385....284

SAMPLE_FASTCHG_SH

Hk_Sample_h	Bk_Sample_h	Attrib_C	Attrib_D	LoadDate	RecordSource	HashDiff
0x00000....000	GhostValue	Ghost	Ghost	1/1/1900 00:00:00 AM	Internal	0x36234....235
0x7F587....895	Biz_Value_123	ABC	DEF	12/26/2020 11:11:36 AM	SystemA	0x12385....284
0x7F587....895	Biz_Value_123	XYZ	ABC	1/30/2021 3:56:12 PM	SystemA	0x50367....622
0x7F587....895	Biz_Value_123	XYZ2	ABC	2/1/2021 1:23:45 PM	SystemA	0x51128....990
0x7F587....895	Biz_Value_123	XYZ3	ABC	3/15/2021 11:32:23 PM	SystemA	0x61092....302
0x7F587....895	Biz_Value_123	ABC	DEF	4/22/2021 2:18:16 PM	SystemA	0x12385....284
0x7F587....895	Biz_Value_123	CAT	DOG	4/22/2021 2:45:56 PM	SystemA	0x39294....912
0x7F587....895	Biz_Value_123	SNAKE	BIRD	4/22/2021 2:50:13 PM	SystemA	0x63853....293
0x7F587....895	Biz_Value_123	CAT	BIRD	4/23/2021 10:45:13 AM	SystemA	0x82730....591
0x7F587....895	Biz_Value_123	CAT	TREE	4/23/2021 10:50:13 AM	SystemA	0x60238....003
0x7F587....895	Biz_Value_123	ABC	DEF	4/26/2021 11:15:13 AM	SystemA	0x12385....284
0x7F587....895	Biz_Value_123	BCD	DEF	4/27/2021 9:32:13 AM	SystemA	0x63853....293
0x7F587....895	Biz_Value_123	ABC	GHJ	4/30/2021 4:45:13 PM	SystemA	0x82730....591

You can see that the attributes in the three different satellites for the Sample_h hub changed at different frequencies and different days and times.

THE PERFORMANCE REPORTING PROBLEM

Let's assume a reporting user wants to know what the Sample_h hub and all its attributes within its three satellites looked like at a particular date and time. As our example, let's select April 26, 2021 at 9am. This date and time are referred to as the reporting user's point-in-time and we are seeking to learn exactly what the data looked like in the source systems.

For hub key "Biz_Value_123" which is Hash Key 0x7F587....895 the most appropriate matching satellite record from each satellite table are these records:

SAMPLE_SLOWCHG_SH

0x7F587....895	Biz_Value_123	XYZ3	ABC	3/16/2021 11:32:23 PM	SystemA	0x61092....302

SAMPLE_MEDCHG_SH

0x7F587....895	Biz_Value_123	TALL	SHORT	2/1/2021 6:23:45 PM	SystemB	0x51128....990

SAMPLE_FASTCHG_SH

0x7F587....895	Biz_Value_123	CAT	TREE	4/23/2021 10:50:13 AM	SystemA	0x60238....003

None of the satellite record's load dates matched the requested point-in-time. In this case, we used the closest record value of each satellite table on the requested date and time, the maximum load date value that is less than or equal to the requested point-in-time date.

But finding the satellite record closest to the point-in-time requested can involve significant compute time. Further compute time is increased when adding more satellites to the hub or link, adding additional or different records or business keys to the satellite, adding more simultaneous users requesting different points-in-time at the same moment from the hub or link and satellites, or when adding additional changes over time to the satellites. These various point-in-time requests can be a significant drain on the data vault's compute resources.

In another scenario, assume that a reporting user wants to know what the Sample_h hub and its attributes in its three satellites looked like on November 10, 2020 at midnight.

For hub key "Biz_Value_123", which is Hash Key 0x7F587....895, the most appropriate matching satellite record from each satellite table are these records:

SAMPLE_SLOWCHG_SH

0x7F587....895	Biz_Value_123	ABC	DEF	10/5/2020 2:34:56 PM	SystemA	0x12385....284

SAMPLE_MEDCHG_SH

0x00000....000	GhostValue	Ghost	Ghost	1/1/1900 00:00:00 AM	Internal	0x36234....235

SAMPLE_FASTCHG_SH

0x00000....000	GhostValue	Ghost	Ghost	1/1/1900 00:00:00 AM	Internal	0x36234....235

Because attributes Attrib_C, Attrib_D, Attrib_E, Attrib_F of the Sample_MedChg_sh and Sample_FastChg_sh satellites respectively didn't exist on November 10, 2020 at midnight, we assign ghost values to those attributes. The ghost record is chosen when no appropriate satellite record exists for the point-in-time the user is requesting.

USING PIT TABLES TO SOLVE THE PERFORMANCE PROBLEM

Performance can degrade quickly depending on how much data exists in the satellites and how many concurrent reporting users are analyzing data in the data vault. How can we improve this performance?

By architecting and implementing PIT tables. Establishing these pre-set, points-in-time in advance anticipates and avoids the need for users to find each individual and the most appropriate matching satellite records per business key with queries.

WHAT DOES A PIT LOOK LIKE?

According to the Data Vault 2.0 standard, a PIT table includes these main DDL components below. This white paper covers some optimizations (though not all that exist) that can be applied to PITs while not breaking the Data Vault 2.0 standard, making the PIT storage needs more efficient. **An important goal of a PIT design is to keep the PIT record as small as feasibly possible.**

SAMPLE_P

SnapshotDate	DATETIME	Point-In-Time Date and Time
Bk_Sample_h	VARCHAR(50)	PK of the PIT – The Bus keys of HUB/LINK
Hk_Sample_h	CHAR(32)	HUB's (or LINK's) hash key for this PIT
Hk_Sample_SlowChg_sh	CHAR(32)	
LoadDate_Sample_SlowChg_sh	DATETIME	
Hk_Sample_MedChg_sh	CHAR(32)	
LoadDate_Sample_MedChg_sh	DATETIME	
Hk_Sample_FastChg_sh	CHAR(32)	
LoadDate_Sample_FastChg_sh	DATETIME	

The grain of the PIT can determine the frequency the point-in-time (SnapshotDate) date field will represent, such as once a year, once a month, once a day or once a minute, etc. The frequency can be evenly spaced or can be complex, jumping around from daily to weekly across a month span of time for one stretch, then go back to daily (that is not typical, and not recommended, but can be accomplished).

Once the frequency is determined, the PIT is filled with the necessary point-in-time/snapshot frequency dates from a beginning to ending date period and can then utilize all the separate point-in-time dates that exist inside the beginning to ending date period. By creating point-in-time datetime instances of the desired frequency, the most appropriate records from all the hub's satellites are located that best match each point-in-time date. Once the best matching satellite records are located from each satellite, the hash keys and load dates of each of the satellite's records are loaded into their respective columns in the PIT record for this point-in-time date.

Assume the frequency grain of our Sample_p PIT table is monthly. We'll start the PIT at January 1, 2021 and end it on April 1, 2021. **The PIT table would look like this:**

SAMPLE_P

Bk_Sample_h	Hk_Sample_h	SnapshotDate	HK_Sample_ SlowChg_sh	LoadDate_Sample_ SlowChg_sh
Biz_Value_123	0x7F587....895	1/1/2021 0:00	0x7F587....895	10/5/2020 2:34:56 PM
Biz_Value_123	0x7F587....895	2/1/2021 0:00	0x7F587....895	10/5/2020 2:34:56 PM
Biz_Value_123	0x7F587....895	3/1/2021 0:00	0x7F587....895	10/5/2020 2:34:56 PM
Biz_Value_123	0x7F587....895	4/1/2021 0:00	0x7F587....895	3/16/2021 11:32:23 PM

Bk_Sample_h	HK_Sample_ MedChg_sh	LoadDate_Sample_ MedChg_sh	HK_Sample_FastChg_sh	LoadDate_Sample_ FastChg_sh
Biz_Value_123	0x7F587....895	11/14/2020 9:16:45 AM	0x7F587....895	12/26/2020 11:11:36 AM
Biz_Value_123	0x7F587....895	1/30/2021 3:56:12 PM	0x7F587....895	1/30/2021 3:56:12 PM
Biz_Value_123	0x7F587....895	2/1/2021 6:23:45 PM	0x7F587....895	2/1/2021 1:23:45 PM
Biz_Value_123	0x7F587....895	2/1/2021 6:23:45 PM	0x7F587....895	3/15/2021 11:32:23 PM

Many times the load date of the satellite record that best matches the point-in-time frequency date can be quite different on a time line because there were no changes to the satellite record over an extended period of time.

It's important to note: the exact load date from the satellite record is necessary to capture in the PIT record for each matching satellite record, because load date is one of the key fields of the satellite table. When reporting users find the PIT record they want, it's then a simple INNER JOIN back to the appropriate satellite table to pull the exact record of satellite attributes as they looked on the point-in-time date with the satellite's hash key and load date.

The PIT table is an extraordinarily valuable time-saver as it pulls together all the right satellite records for a hub or link for a particular point-in-time in advance, reducing both compute time and wait time.

The example above didn't need to pull in any ghost records from the satellites because there was always a good matching record. Had there been a point-in-time date where no satellite record existed, the ghost record for that satellite would have automatically been chosen and placed in the PIT.

SAMPLE CODE TO BUILD A PIT TABLE

The following code is the general pattern for loading data into a PIT table. Alternatively, PIT views can be used for a more virtualized instantiation of a PIT table. The data isn't actually moved into the PIT table but rather the PIT just points to the correct data in the raw data vault or business data vault hub/link and satellite records. The PIT view can be dynamic or materialized; though you may not be able to determine your range of point-in-time dates in advance, you could implement the PIT as more of a dynamic virtualized View—the recommended first option according to the Data Vault 2.0 standard. However, if PIT views create performance issues, it may be necessary to materialize and not virtualize your PIT.

```
INSERT INTO
SchemaName.Sample_p                    -- The PIT table for the Sample_h HUB
(SnapshotDate,                         -- The Point-In-Time for a particular PIT rec
bk_Sample_h,
hk_Sample_h,
hk_Sample_SlowChg_sh,
LoadDate_Sample_SlowChg_sh,
hk_Sample_MedChg_sh,
LoadDate_Sample_MedChg_sh,
hk_Sample_FastChg_sh,
LoadDate_Sample_FastChg_sh)
```

SELECT

```
Snapshot_Dates.as_of_date                -- Table of Point-In-Time dates
,Sample_h.Sample_bk                      Sample_h_BizKey                    -- The HUB Business Key
,Sample_h.Sample_hk                      Sample_h_HashKey                   -- The HUB Hash Key
,COALESCE(MAX(Sample_SlowChg_sh.hk_Sample_h),
                                         CONVERT(binary(16), 0x00, 2))      hk_Sample_SlowChg_sh
,COALESCE(MAX(Sample_SlowChg_sh.LoadDate),
                                         CONVERT(datetime,  '1900-01-01 00:00:00')) LoadDate_Sample_SlowChg_sh
,COALESCE(MAX(Sample_MedChg_sh.hk_Sample_h),
                                          CONVERT(binary(16), 0x00, 2))     hk_Sample_MedChg_sh
,COALESCE(MAX(Sample_MedChg_sh.LoadDate),
                                         CONVERT(datetime,  '1900-01-01 00:00:00')) LoadDate_Sample_MedChg_sh
,COALESCE(MAX(Sample_FastChg_sh.hk_Sample_h),
                                         CONVERT(binary(16), 0x00, 2))      hk_Sample_FastChg_sh
,COALESCE(MAX(Sample_FastChg_sh.LoadDate),
                                         CONVERT(datetime,  '1900-01-01 00:00:00')) LoadDate_Sample_FastChg_sh
```

FROM

```
Sample_h                    -- The HUB (or LINK) associated to all the SATs for the HUB below
INNER JOIN Snapshot_Dates   -- This is a table or inline view of all the Point-In-Time dates
                            -- we want to create PIT records for.  What dates are to be used for Point-In-Time
                            -- dates are purely up to the business requirements
 ON (1=1)                   -- Essentially a CROSS JOIN of the
                            -- Point-In-Time Dates table to the Hub (Sample_h)
LEFT OUTER JOIN     Sample_SlowChg_sh                   Sample_SlowChg_sh
        ON          (  Sample_SlowChg_sh.hk_Sample_h  = Sample_h.Sample_hk
        AND         Sample_SlowChg_sh.LoadDate   <= Snapshot_Dates.as_of_date)
LEFT OUTER JOIN     Sample_MedChg_sh                    Sample_MedChg_sh
        ON          (  Sample_MedChg_sh.hk_Sample_h  = Sample_h.Sample_hk
        AND         Sample_MedChg_sh.LoadDate    <= Snapshot_Dates.as_of_date)
LEFT OUTER JOIN     Sample_FastChg_sh                   Sample_FastChg_sh
        ON          (  Sample_FastChg_sh.hk_Sample_h  = Sample_h.Sample_hk
        AND         Sample_FastChg_sh.LoadDate   <= Snapshot_Dates.as_of_date)
```

```
GROUP BY
Snapshot_Dates.as_of_date
,Sample_h.Sample_hk
,Sample_h.Sample_bk     -- Although it can be questioned why this is needed in the GroupBy
                        -- if the HashKey is guaranteed to be unique by Business Key
--ORDER BY 1, 2;
```

STANDARD DV 2.0 SQL TO ACCESS DATA IN THE PIT TABLE

The following code is the general pattern for selecting data from a PIT table. You can also add a WHERE clause for row filtering, a GROUP BY clause for aggregating satellite attributes and others if desired for further reporting enhancement.

Note how we are now able to use just INNER JOINs (in particular "Equi Joins") back to the satellites, increasing the efficiency of the query to select data from the PIT.

```
-- HOW TO ACCESS HUB (OR LINK) DATA AND ALL ITS ASSOCIATED SAT DATA IN THE PIT ABOVE

SELECT
Sample_p.SnapshotDate,          -- PIT Point-In-Time value
Sample_p.bk_Sample_h,           -- Hub Business key for Sample_h HUB
Sample_p.hk_Sample_h,           -- Hub Hash key for Sample_h HUB
Sample_SlowChg_sh.*,            -- SlowChg Sat flds associated with Sample_h HUB
Sample_MedChg_sh.*,             -- MedChg  Sat flds associated with Sample_h HUB
Sample_FastChg_sh.*             -- FastChg Sat flds associated with Sample_h HUB
FROM                    SchemaName.Sample_p  Sample_p                   -- The PIT table
INNER JOIN              Sample_SlowChg_sh    Sample_SlowChg_sh          -- Sat #1 for Sample_h HUB
        ON              Sample_SlowChg_sh.hk_Sample_h = Sample_p.hk_Sample_SlowChg_sh
        AND             Sample_SlowChg_sh.LoadDate    = Sample_p.LoadDate_Sample_SlowChg_sh
INNER JOIN              Sample_MedChg_sh     Sample_MedChg_sh           -- Sat #2 for Sample_h HUB
        ON              Sample_MedChg_sh.hk_Sample_h  = Sample_p.hk_Sample_MedChg_sh
        AND             Sample_MedChg_sh.LoadDate     = Sample_p.LoadDate_Sample_MedChg_sh
INNER JOIN              Sample_FastChg_sh    Sample_FastChg_sh          -- Sat #3 for Sample_h HUB
        ON              Sample_FastChg_sh.hk_Sample_h = Sample_p.hk_Sample_FastChg_sh
        AND             Sample_FastChg_sh.LoadDate    = Sample_p.LoadDate_Sample_FastChg_sh
```

OPTIMIZING THE DATA STORAGE FOOTPRINT OF THE PIT WITHOUT SACRIFICING PERFORMANCE

PIT tables can require significant amounts of storage to represent all the points-in-time requested by users for all the business keys in the hub or link. There are techniques beyond the scope of this document that can be employed against a PIT to significantly reduce the data size footprint of a PIT, but you lose some point-in-time granularity implementing them.

However, the following technique significantly reduces the data size required by a PIT by making some modifications to the PIT's DDL, slight code changes while creating the data in the PIT, and a small modification when querying the PIT.

First, recall that at the beginning of this document the DV 2.0 standard DDL of a PIT table is:

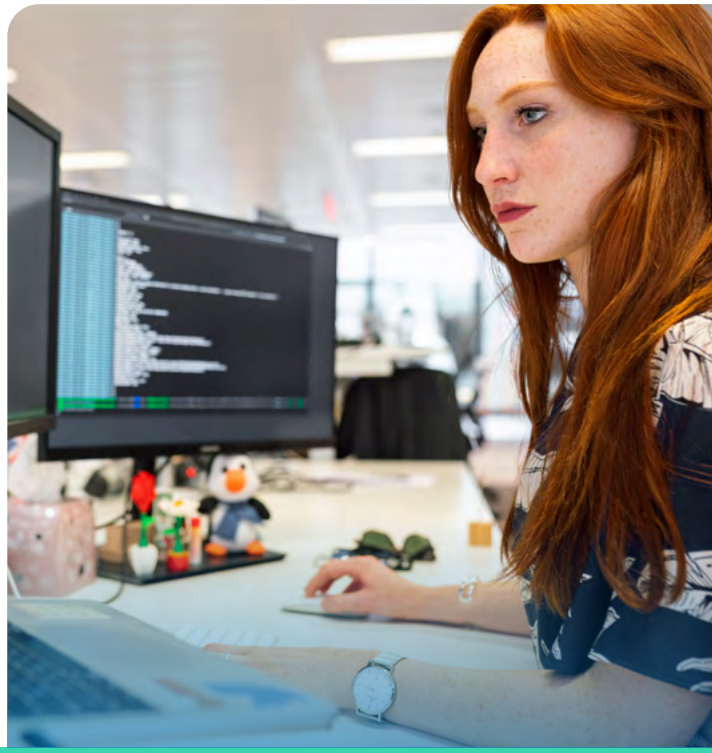
SAMPLE_P (PIT DDL ACCORDING TO DV 2.0 STDS)

SnapshotDate DATETIME	Point-In-Time Date and Time
bk_Sample_h VARCHAR(50)	PK of the PIT – The Bus key of the Hub/Link
hk_Sample_h CHAR(32)	HUB's hash key this PIT rec is for
hk_Sample_SlowChg_sh	CHAR(32)
LoadDate_Sample_SlowChg_sh	DATETIME
hk_Sample_MedChg_sh	CHAR(32)
LoadDate_Sample_MedChg_sh	DATETIME
hk_Sample_FastChg_sh	CHAR(32)
LoadDate_Sample_FastChg_sh	DATETIME

Note that all the hash key fields (starting with hk_) require a significant amount of storage size per PIT record. CHAR(32) is required to store an MD5 hash. It gets even larger for more enhanced hash value types (e.g. SHA1, SHA256, etc.).

It would be very helpful to reduce or eliminate as many of those as possible.

If you refer to the sample data earlier in this article, you'll notice that unless the satellite hash key value in the PIT was set to the ghost record value, its hash value was always equal to the hub (or link) hash key field value in the PIT. In the DDL above, this means the hk_Sample_SlowChg_sh, hk_Sample_MedChg_sh, and hk_Sample_FastChg_sh hash values are identical to the hk_Sample_h hash value for the associated hub hash key value (or link if this was a PIT for a link's satellites).



We can make use of that fact and eliminate the HashKey hk satellite fields from the PIT. Taking the hub business key out of consideration of PII size savings (because it will vary by PIT), then removing the hk SAT fields will reduce the size of the PIT by 34% to 56% or more in space savings for MD5 hashes depending on the number of satellites in the PIT, 36% to 60% or more in space savings for SHA1 hashes, and 40% to 67% or more in space savings for SHA256 hashes. Smaller PIT record sizes will likely equate to faster performance, while not losing any PIT functionality.

The new PIT structure would look like this:

SAMPLE_P (PIT DDL TWEAKED)

SnapshotDate DATETIME	Point-In-Time Date and Time
bk_Sample_h VARCHAR(50)	PK of the PIT – The Bus key of the Hub/Link
hk_Sample_h CHAR(32)	HUB's hash key this PIT rec is for
LoadDate_Sample_SlowChg_sh	DATETIME
LoadDate_Sample_MedChg_sh	DATETIME
LoadDate_Sample_FastChg_sh	DATETIME

To build the PIT for this new DDL, simply leave out the hk satellite fields when INSERTING into the PIT. Everything else is the same with the previous code. Here is that adjusted code:

INSERT INTO

```
SchemaName.Sample_p                    -- The PIT table for the Sample_h HUB
(SnapshotDate,
bk_Sample_h,
hk_Sample_h,
~~hk_Sample_SlowChg_sh,~~ --------     < ---- Remove this SAT hash key here and below
LoadDate_Sample_SlowChg_sh,
~~hk_Sample_MedChg_sh,~~ --------      < ---- Remove this SAT hash key here and below
 LoadDate_Sample_MedChg_sh,
~~hk_Sample_FastChg_sh,~~ --------     < ---- Remove this SAT hash key here and below
 LoadDate_Sample_FastChg_sh)
```

```
SELECT
Snapshot_Dates.as_of_date                    -- Table of Point-In-Time dates
,Sample_h.Sample_bk      Sample_h_BizKey     -- The HUB Business Key
,Sample_h.Sample_hk      Sample_h_HashKey    -- The HUB Hash Key
,COALESCE(MAX(Sample_SlowChg_sh.hk_Sample_h),
        CONVERT(binary(16), 0x00, 2))        hk_Sample_SlowChg_sh
,COALESCE(MAX(Sample_SlowChg_sh.LoadDate),
        CONVERT(datetime,  '1900-01-01 00:00:00'))        LoadDate_Sample_SlowChg_sh
,COALESCE(MAX(Sample_MedChg_sh.hk_Sample_h),
        CONVERT(binary(16), 0x00, 2))        hk_Sample_MedChg_sh
,COALESCE(MAX(Sample_MedChg_sh.LoadDate),
        CONVERT(datetime,  '1900-01-01 00:00:00'))        LoadDate_Sample_MedChg_sh
,COALESCE(MAX(Sample_FastChg_sh.hk_Sample_h),
        CONVERT(binary(16), 0x00, 2))        hk_Sample_FastChg_sh
,COALESCE(MAX(Sample_FastChg_sh.LoadDate),
        CONVERT(datetime,  '1900-01-01 00:00:00'))        LoadDate_Sample_FastChg_sh

FROM
Sample_h                                     -- The HUB associated to all the SATs for the HUB below
INNER JOIN          Snapshot_Dates           -- This is a table or inline view of all the Point-In-Time dates
                                             -- we want to create PIT records for.  What dates are to be used for
                                             -- Point-In-Time dates are purely up to the business requirements
ON (1=1)                                     -- Essentially a CROSS JOIN of the Point-In-Time Dates table to
                                             the Hub (Sample_h)
 LEFT OUTER JOIN    Sample_SlowChg_sh        Sample_SlowChg_sh
        ON          (  Sample_SlowChg_sh.hk_Sample_h     = Sample_h.Sample_hk
        AND         Sample_SlowChg_sh.LoadDate           <= Snapshot_Dates.as_of_date)
 LEFT OUTER JOIN    Sample_MedChg_sh         Sample_MedChg_sh
        ON          (  Sample_MedChg_sh.hk_Sample_h      = Sample_h.Sample_hk
        AND         Sample_MedChg_sh.LoadDate            <= Snapshot_Dates.as_of_date)
 LEFT OUTER JOIN    Sample_FastChg_sh        Sample_FastChg_sh
        ON          (  Sample_FastChg_sh.hk_Sample_h     = Sample_h.Sample_hk
        AND         Sample_FastChg_sh.LoadDate           <= Snapshot_Dates.as_of_date)

GROUP BY
Snapshot_Dates.as_of_date
,Sample_h.Sample_hk
,Sample_h.Sample_bk    -- Although it can be questioned why this is needed in the GroupBy if the HashKey is
                       -- guaranteed to be unique by Business Key
--ORDER BY 1, 2;
```

Remove the highlighted yellow areas, and you'll have a storage optimized PIT without loss of PIT functionality.

But to SELECT data from the storage optimized PIT, the selection SQL code modifies slightly as shown here:

```
-- How to access HUB (or LINK) and its SAT data in the PIT above
-- But this tweak to accessing the PIT data allows us to leave out the
-- HUB hash key fields for all the SAT objects in the PIT saving a significant
-- amount of storage since PITs can run very heavy on data size footprint.
-- Can do this because the HUB hash key (or LINK) field in the PIT is identical to
-- the HUB hash key for all of the SATs in the PIT.
SELECT
        Sample_p.SnapshotDate,                  -- PIT Point-In-Time value
        Sample_p.bk_Sample_h,                   -- Hub Business key for Sample_h HUB
        Sample_p.hk_Sample_h,                   -- Hub Hash key for Sample_h HUB
        Sample_SlowChg_sh.*,                    -- SlowChg Sat flds associated with Sample_h HUB
        Sample_MedChg_sh.*,                     -- MedChg  Sat flds associated with Sample_h HUB
        Sample_FastChg_sh.*                     -- FastChg Sat flds associated with Sample_h HUB
FROM
SchemaName.Sample_p  Sample_p                   -- The PIT table
INNER JOIN    Sample_SlowChg_sh   Sample_SlowChg_sh   -- Sat #1 for Sample_h HUB
        ON      Sample_SlowChg_sh.hk_Sample_h =
        CASE WHEN Sample_p.LoadDate_Sample_SlowChg_sh = '1900-01-01' THEN
        CONVERT(binary(16), 0x00,2)             -- Find GHOST record
        ELSE Sample_p.hk_Sample_h               -- Find actual SAT record
        END
AND             Sample_SlowChg_sh.LoadDate = Sample_p.LoadDate_Sample_SlowChg_sh
INNER JOIN    Sample_MedChg_sh   Sample_MedChg_sh   -- Sat #2 for Sample_h HUB
        ON      Sample_MedChg_sh.hk_Sample_h =
        CASE WHEN Sample_p.LoadDate_Sample_MedChg_sh = '1900-01-01' THEN
        CONVERT(binary(16), 0x00, 2)            -- Find GHOST record
        ELSE    Sample_p.hk_Sample_h            -- Find actual SAT record
        END
AND             Sample_MedChg_sh.LoadDate = Sample_p.LoadDate_Sample_MedChg_sh
INNER JOIN    Sample_FastChg_sh   Sample_FastChg_sh   -- Sat #3 for Sample_h HUB
        ON      Sample_FastChg_sh.hk_Sample_h =
        CASE WHEN Sample_p.LoadDate_Sample_FastChg_sh = '1900-01-01' THEN
        CONVERT(binary(16), 0x00, 2)            -- Find GHOST record
        ELSE    Sample_p.hk_Sample_h            -- Find actual SAT record
        END
AND             Sample_FastChg_sh.LoadDate = Sample_p.LoadDate_Sample_FastChg_sh
```

The data fields have been selected the exact same way as before using the DV 2.0 standard. And as highlighted in GREEN, the very important INNER JOIN (Equi Join) technique is performed on the satellites and the same satellite load date comparison is also highlighted in GREEN.

Leveraging the fact that the satellite hash key value in the PIT is always the same as the hash key field in the satellite which is the same as the hub or link hash key field, you are able to remove the PIT satellite hash keys because they are all redundant with the hub or link hash key in the same PIT record.

This example uses the satellite load date fields that still remain in the PIT and checks for the ghost value load date (January 1, 1900 in PURPLE). If that ghost date is found, return the ghost hash key (in CYAN) to look up the matching satellite record. If no ghost date is detected for the satellite in the PIT, then use the PIT's hub hash key (in YELLOW) to look up the matching SAT record. The PIT's hub hash key is identical to the SAT hash key in the PIT in the standard DV 2.0 standard.

This is a small enhancement tweak to the PIT selection SQL which should perform about as efficiently as the DV 2.0 standard code, but you cut down 34%-60% or more of the PIT storage. Less storage is used, and in many cases, this can mean better performance response time because there is less data to scan through.

CONDITIONS WHEN SAT LOAD DATE MAY NOT BE THE BEST POINT-IN-TIME FIELD CHOICE

According to the DV 2.0 standard, PIT tables are based off a hub or link hash key and a load date from the satellite that is stored in the PIT for finding the proper satellite later when accessing the PIT records.

Occasionally, there are data circumstances within the source data that make using the satellite load date a difficult way to manage point-in-time requests from the reporting users.

One such case is late-arriving data from the source. Some source systems can send data out of sequence, such as when a transaction takes longer to execute even though it started prior to another transaction which completes sooner than the original transaction. Hence, source records can come into the data vault in a different order than they actually occurred.

In this instance, the accurate history would be altered from how the data changes actually occurred in the source system and there would not be a correct load date representation in the satellite.

For these or similar cases, it would be better to use a reliable source system provided date and time to represent the load date to manage the PIT table. Using a business date and time that represents the true order of how the source data processed at the source will allow the satellites to be self-healing, allowing the late arriving data to find its rightful order in the PIT. However, changing this load date behavior modification is not allowed in the raw data vault satellites per the DV 2.0 standard, so you will likely need to switch these types of load dates (wall clock date and time vs. business date and time field) in the business data vault satellites. The DV 2.0 standard refers to this business date from the source as the "Applied Date".

Based on the knowledge of the source data, it may be necessary to make adjustments to the way PITs operate to best capture the proper data order, thus creating an accurate point-in-time experience from the PITs for the reporting user.

SUMMARY OF THE VALUE OF PITS

PIT tables help tremendously with reducing coding complexity, allowing the reporting user to find the correct satellite record for a particular point-in-time most efficiently. Most importantly, PITs will greatly reduce the amount of compute time needed to find the correct set of satellite records for a hub or link for a particular point-in-time, saving valuable resources and possibly budget.

Remember that a particular PIT table is associated with only one hub or one link object along with one to all of the hub or link's associated satellite objects. A different hub or link object would need a different PIT table. Additionally, a particular hub or link object may have more than one PIT table, because it may need to support different point-in-time/snapshot frequencies, or different subset combinations of the satellites that belong to a single hub or link.

About **Resultant**

Our team believes solutions are more valuable, transformative, and meaningful when reached together. Through solutions rooted in data analytics, technology, and digital transformation, Resultant serves as a true partner by solving problems with our clients rather than for them.

Author:

JIM MIHALICK
Sr. Principal BI Consultant

jmihalick@resultant.com