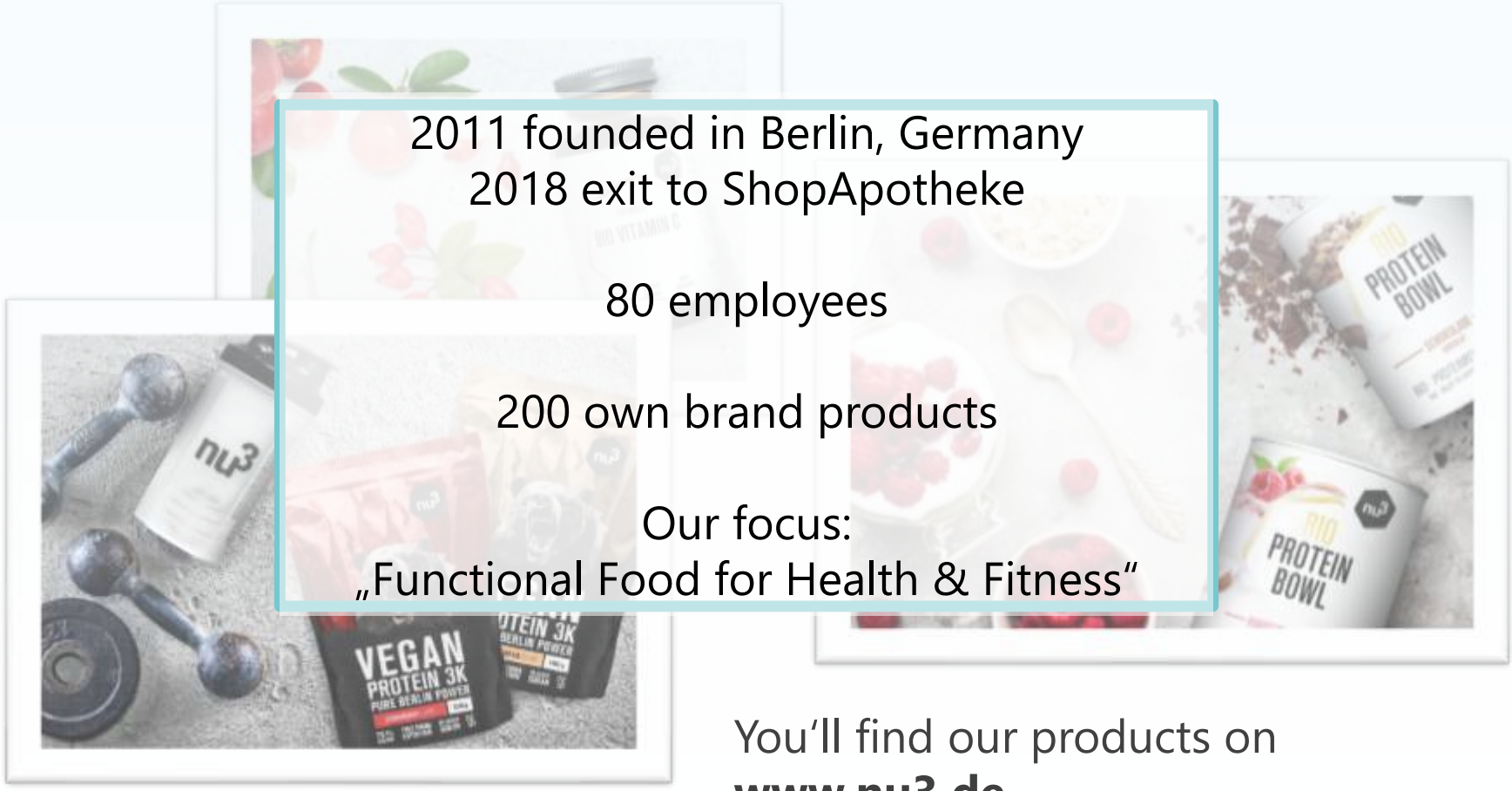
Universal Loading Pattern

nu3 | INTELLIGENT NUTRITION

nu3 GmbH

With intelligent nutrition to a healthier, fitter and happier life.



2011 founded in Berlin, Germany
2018 exit to ShopApotheke

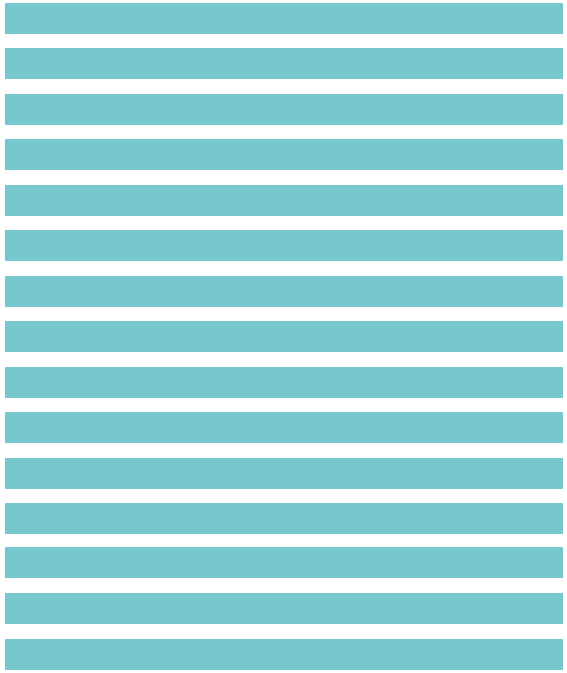
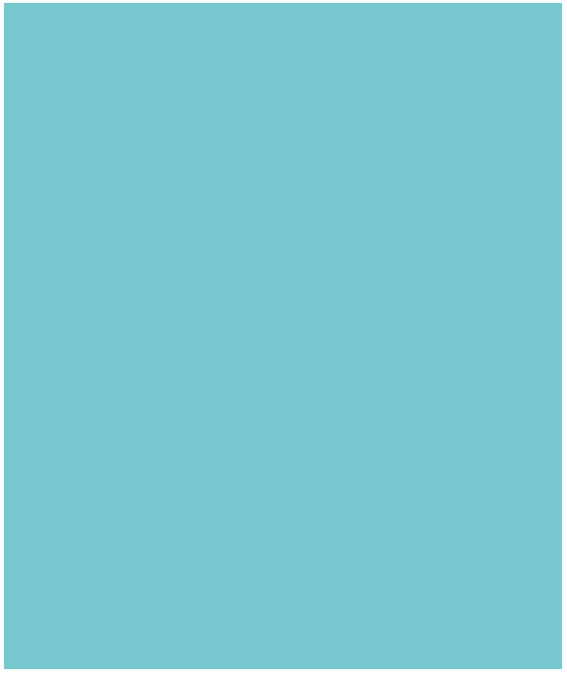
80 employees

200 own brand products

Our focus:
„Functional Food for Health & Fitness“

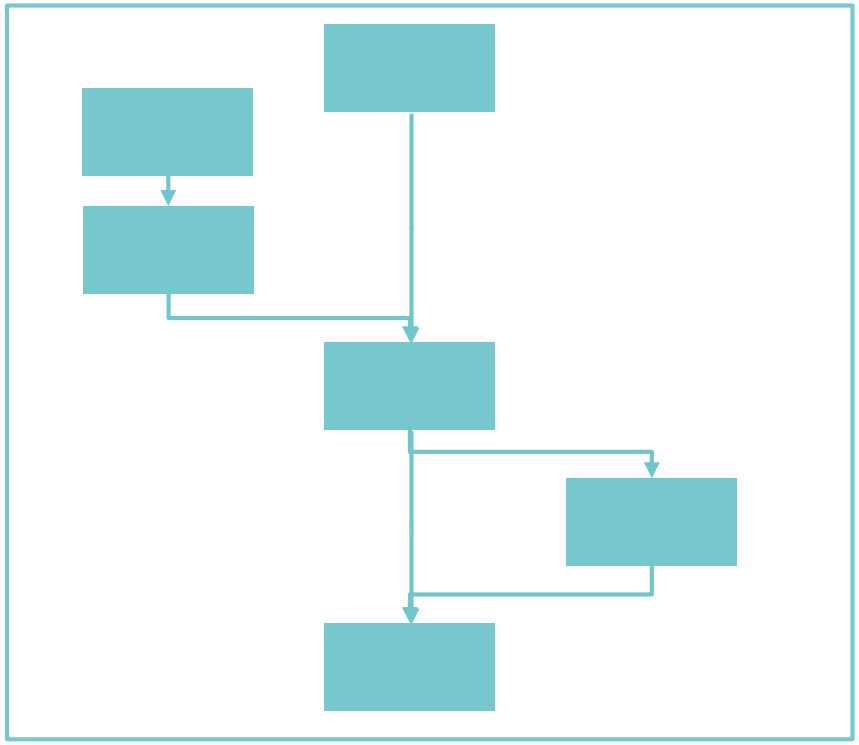
You'll find our products on **www.nu3.de**.

Data Processing Options

Row-by-Row	Batch	Set
		
Streaming – kontinuierlich, aber langsam, wenn große Datenmengen bewegt werden OLTP – Unsere Quellen	Processing by Batch. Z. B. je LoadDate Daten in Satelliten schreiben.	Kompletter Datenraum. Alle Datensätze, die bewegt werden sollen. Fasst mehrere LoadDate zusammen. Normalerweise schnellste Variante.

Programming Options

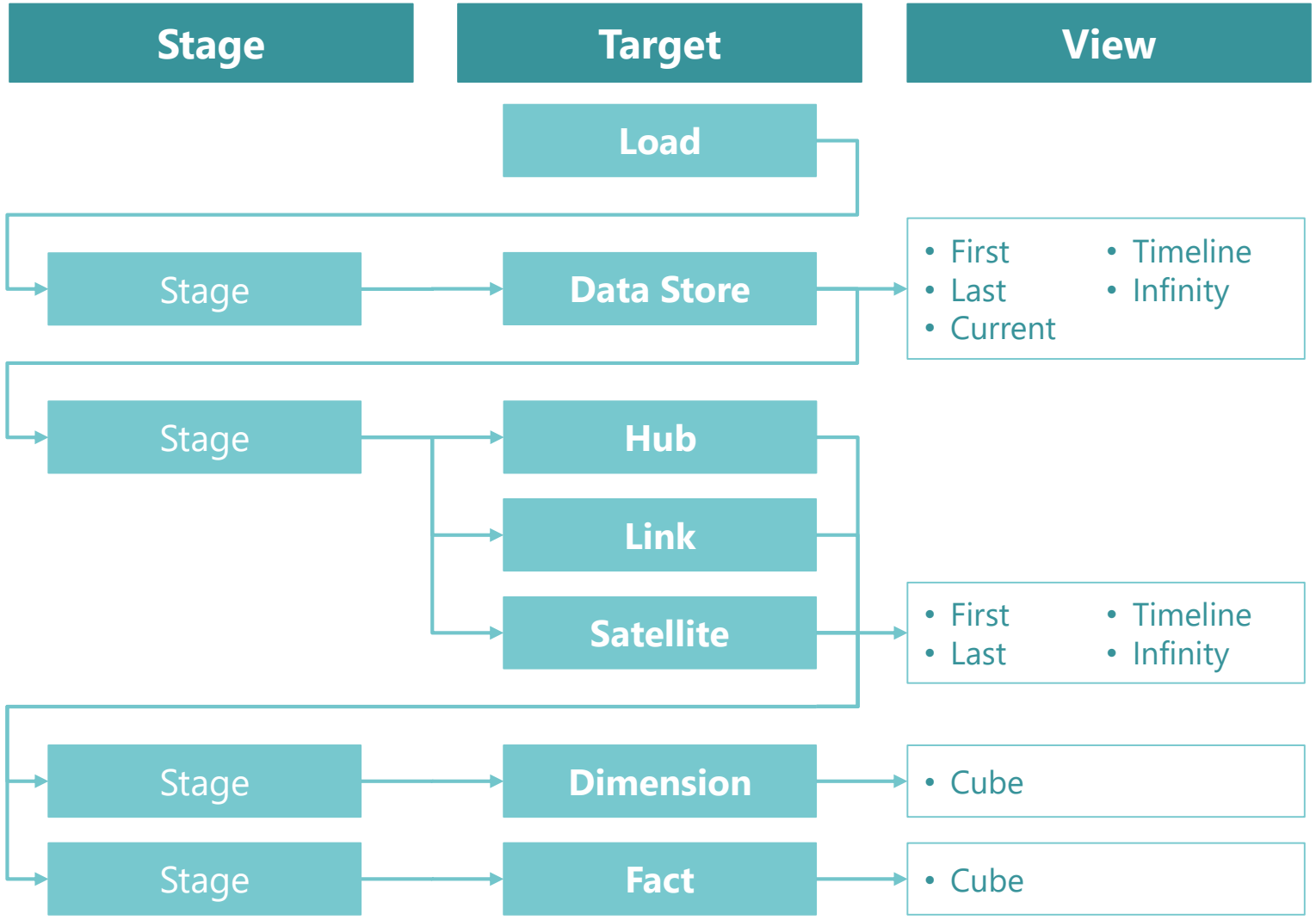
External



In-Database

```
INSERT INTO
SELECT *
FROM   (
  SELECT {calculate}
  FROM   (
    SELECT {transform}
    FROM   (
      SELECT *
      FROM   [source]
    )
  )
)
```

Beladung


Stage
Target
View
Load
Stage
Data Store
• First
• Last
• Current
• Timeline
• Infinity
Stage
Hub
Link
Satellite
• First
• Last
• Timeline
• Infinity
Stage
Dimension
• Cube
Stage
Fact
• Cube

Stage Table

Query Direkt

Extract

Load

```
SELECT
       ISNULL(NULLIF(UPPER(LTRIM(RTRIM(business_key))),''),'-1')
           AS business_key
      ,data1 AS data1
      ,data2 AS data2
      ,HASHBYTES('md5',
         CAST(
           ISNULL(NULLIF(UPPER(LTRIM(RTRIM(business_key))),'')
              ,'-1')
         AS NVARCHAR(MAX))
       ) AS hub_hash_key
      ,HASHBYTES('md5',
         ISNULL(CAST(data1 AS NVARCHAR(MAX)),'') +'||'+
         ISNULL(CAST(data2 AS NVARCHAR(MAX)),'')
         ) AS dss_change_hash
      ,dss_load_datetime AS dss_load_datetime
      ,dss_record_source AS dss_record_source
FROM   {source}
```

BK Cleaning

Hash Key Calculation

Stage Table

Hash Calculation

Extract

Hash Calculation

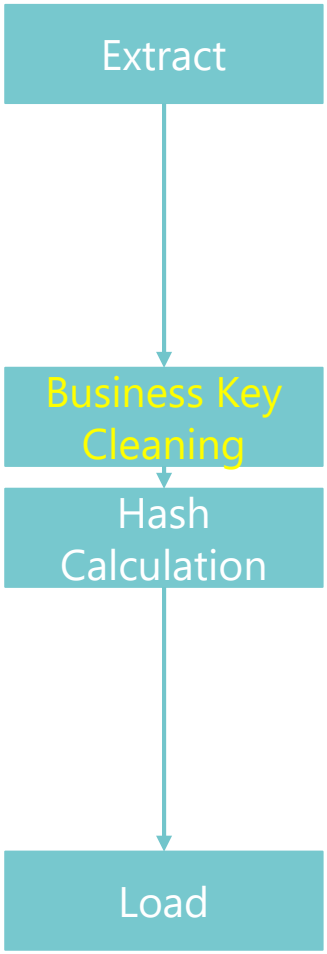
Load

```
SELECT *
      ,HASHBYTES('md5',
         CAST(business_key AS NVARCHAR(MAX))
       ) AS hub_hash_key
      ,HASHBYTES('md5',
         ISNULL(CAST(data1 AS NVARCHAR(MAX)),'') +'||'+
         ISNULL(CAST(data2 AS NVARCHAR(MAX)),'')
       ) AS dss_change_hash
FROM   (
  SELECT
         ISNULL(NULLIF(UPPER(LTRIM(RTRIM(business_key))),''),'-1')
           AS business_key
        ,data1 AS data1
        ,data2 AS data2
        ,dss_load_datetime AS dss_load_datetime
        ,dss_record_source AS dss_record_source
  FROM   {source}
) AS target
```

Hash Key Calculation

Stage Table

Hash Calculation mit Business Key Cleaning




```
SELECT *
      ,HASHBYTES('md5',
         CAST(business_key AS NVARCHAR(MAX))
       ) AS hub_hash_key
      ,HASHBYTES('md5',
         ISNULL(CAST(data1 AS NVARCHAR(MAX)),'') +'||'+
         ISNULL(CAST(data2 AS NVARCHAR(MAX)),'')
       ) AS dss_change_hash
FROM   (
  SELECT ISNULL(NULLIF(UPPER(LTRIM(RTRIM(business_key))),''),'-1')
           AS business_key
        ,…
  FROM   (
    SELECT business_key AS business_key
          ,data1 AS data1
          ,data2 AS data2
          ,dss_load_datetime AS dss_load_datetime
          ,dss_record_source AS dss_record_source
    FROM   {source}
  ) AS target
) AS target
```

BK Cleaning

„Clean" Query

Stage Table

Data Type Mapping (Edge case)

- Extract
- Data Type Mapping
- Business Key Cleaning
- Hash Calculation
- Load

Hash Example

```
DECIMAL      10.0  (03602…)
```

VS

```
INT          10    (67AE8…)
MONEY        10,00 (39A28…)
NVARCHAR     10    (67AE8…)
```

```
DATE
DATETIME
DATETIME2
DATETIMEOFFSET
TIME
```

```
NVARCHAR
DATETIME2
DATETIME
TIME
DATE
```

Data Type Mapping

```
SELECT
       ,CAST(column AS {data_type}) AS column
FROM   (
  {sub_query}
)
```

Stage Table

Delta Loading

Extract

Delta

Data Type Mapping

Business Key Cleaning

Hash Calculation

Load

View

```
SELECT ...
FROM   (
  {sub_query}
) AS target
WHERE dss_load_datetime > (
  SELECT ISNULL(MAX(dss_load_datetime),'0001-01-01')
  FROM   {target}
)
```

Stored Procedure

```
SET @dss_load_datetime = (
  SELECT ISNULL(MAX(dss_load_datetime),'0001-01-01')FROM {target})

SELECT ...
FROM   (
  {sub_query}
) AS target
WHERE dss_load_datetime > @max_dss_load_datetime
```

Stage Table

Delta Loading Extended

Extract

Delta

Data Type Mapping

Business Key Cleaning

Hash Calculation

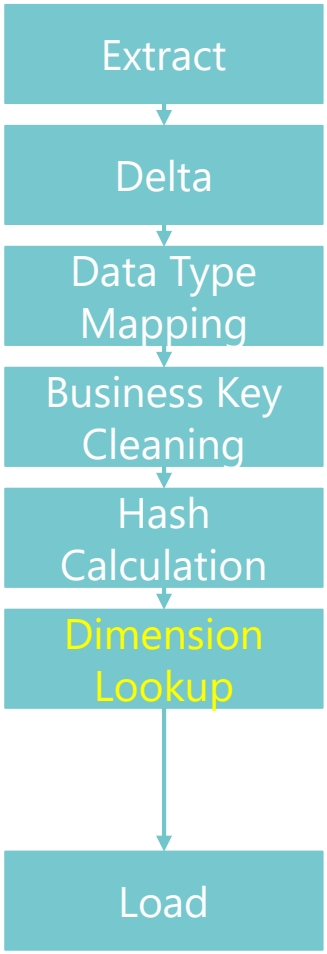
Load

Stored Procedure

```
SET @dss_load_datetime = (
  SELECT ISNULL(MAX(dss_load_datetime),'0001-01-01')
  FROM   {target}
)
SELECT ...
FROM   (
  SELECT ...
        ,(
            SELECT MAX(v)
            FROM (
              VALUES
                (sat1.dss_load_datetime)
               ,(sat2.dss_load_datetime)
               ,...
            ) AS VALUE(v)
         ) AS dss_load_datetime
  FROM   {source}
) AS target
WHERE dss_load_datetime > @max_dss_load_datetime
```

Stage Table

Dimension Lookup bei Faktentabelle




```
SELECT *
      ,ISNULL({dimension}.dim_surrogate_key,-1) AS dim_key
FROM   (
  {sub_query}
) AS target
LEFT OUTER JOIN {dimension}
ON {dimension.dim_business_key}={target.dim_business_key}
```

Stage Table

Row Condensing / Removing Duplicates (DON'T!!)

- Extract
- Delta
- Data Type Mapping
- Business Key Cleaning
- Hash Calculation
- Dimension Lookup
- **Row Condensing**
- Load

Je nach Daten, kommt man eventuell nicht umhin, Duplikate zu eliminieren. Kann man in der Extract Query von Hand erstellen, oder eine Funktion verwenden – wie hier.

```
SELECT *
      ,ROW_NUMBER() OVER (PARTITION BY {business key}
                          ORDER BY dss_load_datetime DESC,
                                   dss_sequence_no DESC) AS row_no
FROM    (
  {sub_query}
) AS target
WHERE row_no = 1
```

Stage Table

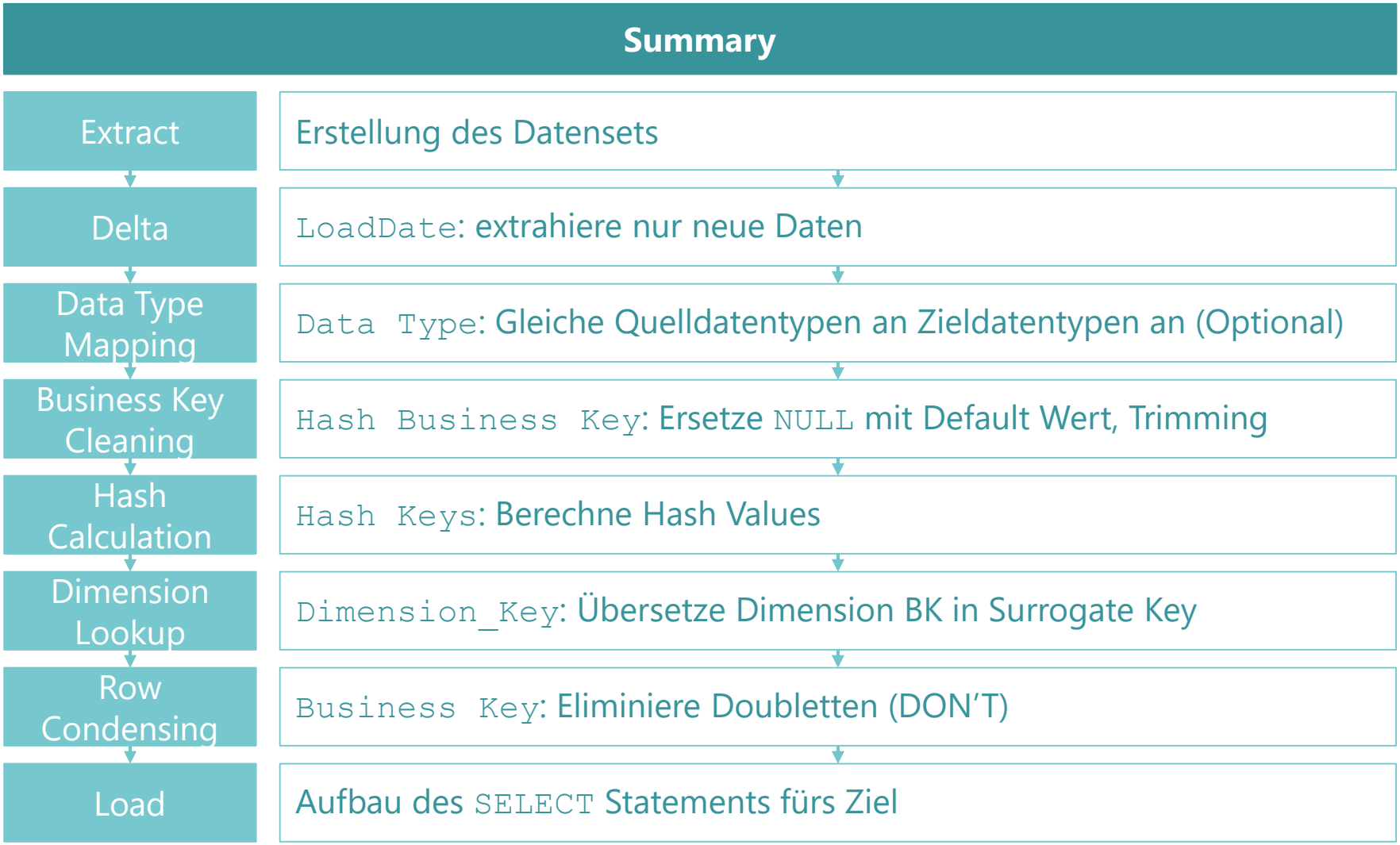
Load

- Extract
- Delta
- Data Type Mapping
- Business Key Cleaning
- Hash Calculation
- Dimension Lookup
- Row Condensing
- Load

Alle Felder und Kalkulation in der richtige Reihenfolge auflisten für VIEW oder INSERT.

```
INSERT INTO ... / CREATE VIEW ... AS
SELECT
       hub_hash_key AS hub_hash_key
      ,business_key AS business_key
      ,data1 AS data1
      ,data2 AS data2
      ,dss_load_datetime AS dss_load_datetime
      ,dss_record_source AS dss_record_source
      ,dss_change_hash AS dss_change_hash
      ,SYSDATETIME() AS dss_create_datetime
FROM   (
  {sub_query}
) AS target
```

Stage Table


Summary
Extract
Erstellung des Datensets
Delta
LoadDate: extrahiere nur neue Daten
Data Type Mapping
Data Type: Gleiche Quelldatentypen an Zieldatentypen an (Optional)
Business Key Cleaning
Hash Business Key: Ersetze NULL mit Default Wert, Trimming
Hash Calculation
Hash Keys: Berechne Hash Values
Dimension Lookup
Dimension_Key: Übersetze Dimension BK in Surrogate Key
Row Condensing
Business Key: Eliminiere Doubletten (DON'T)
Load
Aufbau des SELECT Statements fürs Ziel

Satellite

Schnellste Lademethode

Extract

Load

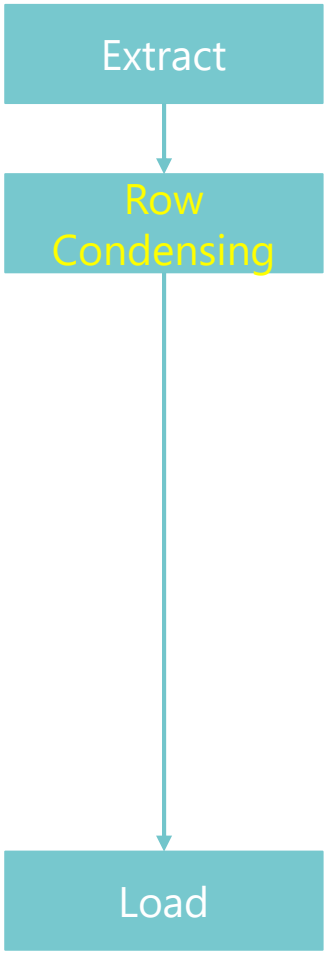
Ein direktes Beladen aus der Stage Tabelle in einen Satelliten sollte möglich sein und ohne Fehler durchlaufen.

Ein virtueller Satellite (=VIEW) wird genau so beladen. Schnellste Möglichkeit.

Satellite

Row Codensing: What & Why?

Inspired by Roelant Vos



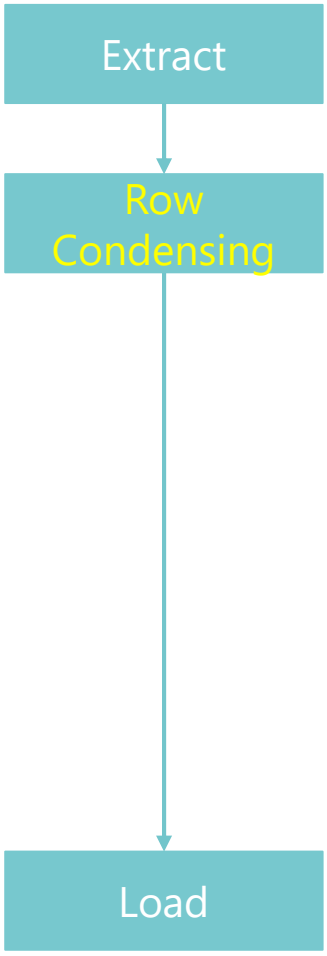
HashKey	Data	EventDate	LoadDate	ChangeHash
A…	123	15.03.2019	20.03.2019	Z…
~~A…~~	~~123~~	~~21.03.2019~~	~~22.03.2019~~	~~Z…~~
A…	234	22.03.2019	24.03.2019	X…

HashKey	Data	EventDate	LoadDate	ChangeHash
B…	123	05.03.2019	19.03.2019	Z…
B…	123	15.03.2019	20.03.2019	Z…
B…	234	10.03.2019	21.03.2019	X…
~~B…~~	~~234~~	~~10.03.2019~~	~~22.03.2019~~	~~X…~~

HashKey	Data	EventDate	LoadDate	ChangeHash
C…	123	15.03.2019	20.03.2019	Z…
~~C…~~	~~123~~	~~15.03.2019~~	~~20.03.2019~~	~~Z…~~

Satellite

Row Codensing: Verarbeitung



Level 1
- Sortiere LoadDates und erstelle RowNo

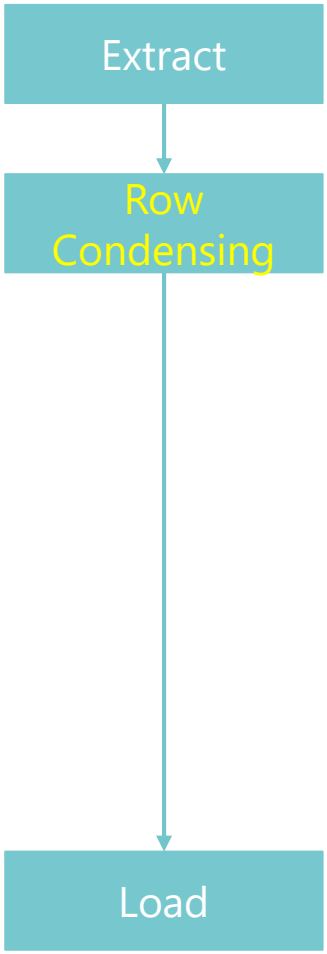
Level 2
- Pro HashKey
  - Sortiere Rows nach RowNo und berechne, ob sich ChangeHash ändert

Level 3
- Pro HashKey oder DrivingKey
  - Sortiere Rows nach EventDate und RowNo und berechne, ob sich ChangeHash ändert

Level 4
- Filtere Rows, die Änderungen enthalten

Satellite

Row Codensing: 1/4 – LoadDate sortieren



Wir pflegen 2 Timelines (Unitemporal)

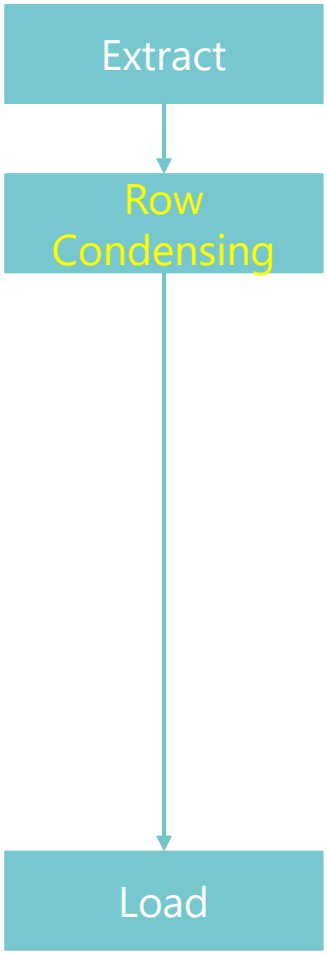
- LoadDate
- EventDate

Um Sicherzustellen, dass wir jeweils mit beiden Timelines richtig sortieren, erzeugen wir eine eindeutige Reihenfolge basierend auf dem LoadDate.

```
SELECT *
      ,ROW_NUMBER() OVER (ORDER BY dss_load_datetime)
                                                AS row_no__load_date
FROM    (
  {sub_query}
) AS target
```

Satellite

Row Codensing: 2/4 – Detect LoadDate Changes



Vergleiche den ChangeHash mit der vorhergehenden Zeile gruppiert nach dem HashKey und wenn es eine Änderung gibt, setze Wert auf 1, ansonsten 0.

```
SELECT *
      ,CASE WHEN LAG(dss_change_hash,1,CONVERT(BINARY(16),''))
                      OVER (PARTITION BY {hash_key}
                            ORDER BY row_no__load_date ASC)
                 =dss_change_hash
            THEN 0
            ELSE 1
       END AS is_load_date_change_hash_change
FROM   (
  {sub_query}
) AS target
```

Satellite

Row Codensing: 3/4 – Detect EventDate Changes

Extract

Row Condensing

Load

Satellite

```
SELECT *
      ,CASE WHEN LAG(dss_change_hash,1,CONVERT(BINARY(16),''))
                      OVER (PARTITION BY {hash_key}
                            ORDER BY dss_event_datetime ASC,
                                     row_no__load_date ASC)
                  =dss_change_hash
             THEN 0
             ELSE 1
       END AS is_event_date_change_hash_change
```

Effectivite Satellite mit Driving Key

```
SELECT *
      ,CASE WHEN LAG(dss_change_hash,1,CONVERT(BINARY(16),''))
                      OVER (PARTITION BY {driving_key}
                            ORDER BY dss_event_datetime ASC,
                                     row_no__load_date ASC)
                  =dss_change_hash
             THEN 0
             ELSE 1
       END AS is_event_date_change_hash_change
```

Satellite

Row Codensing: 4/4 – Filter Usable Rows

Extract

Row Condensing

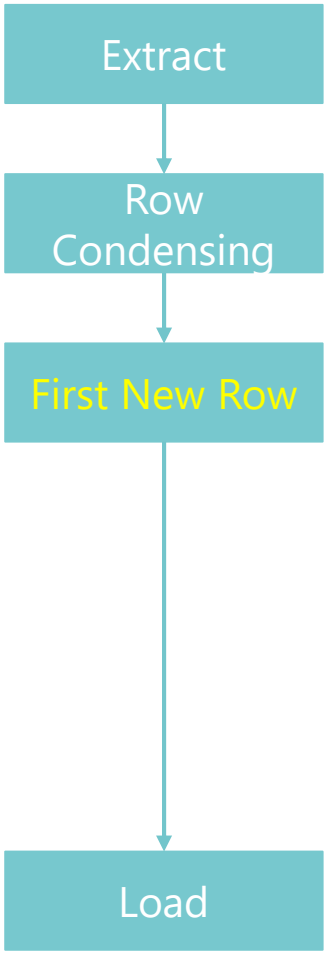
Load

Mit der Sortierung und Filterung der Daten erhalten wir jetzt ein Datenset, welches um Doubletten bereinigt ist.

```
SELECT *
FROM   (
  {sub_query}
) AS target
WHERE is_load_date_change_hash_change = 1
   OR is_event_date_change_hash_change = 1
```

Satellite

First New Row: What & Why?



Eventuell kann unser 1. neuer Datensatz eines HashKeys bereits im Satelliten vorhanden sein.
Braucht es diesen Schritt? Nein. Es macht die Satelliten nur hübsch.

Stage

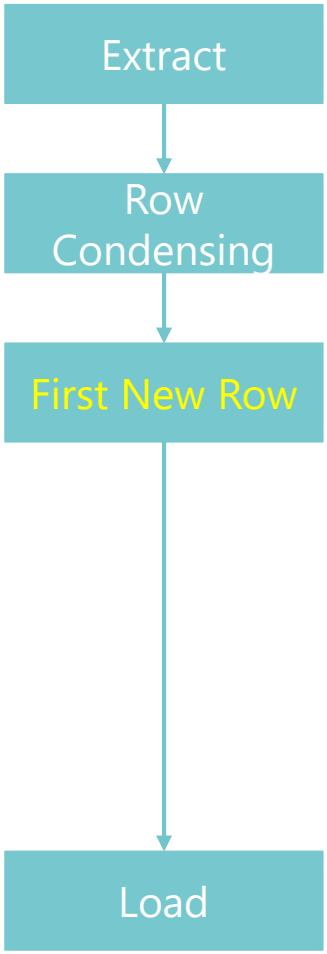
HashKey	Data	LoadDate	ChangeHash
~~A…~~	~~234~~	~~20.03.2019~~	~~X…~~
A…	345	24.03.2019	Y…

Satellite

HashKey	Data	LoadDate	ChangeHash
A…	123	10.03.2019	Z…
A…	234	15.03.2019	X…

Satellite

First New Row: Verarbeitung



Level 1

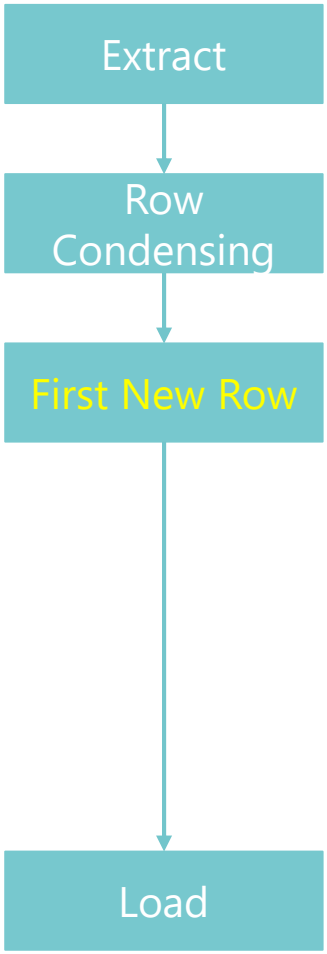
- Stage
  - Pro HashKey:
    - Sortiere nach LoadDate und erzeuge RowNo

Level 2

- `LEFT JOIN` Satellite
  - Pro HashKey:
    - Sortiere nach LoadDate und identifiziere letzter Eintrag
- WHERE
  - Wenn sich für den 1. Datensatz der ChangeHash ändert
  - Alle anderen Datensätze (2+)

Satellite

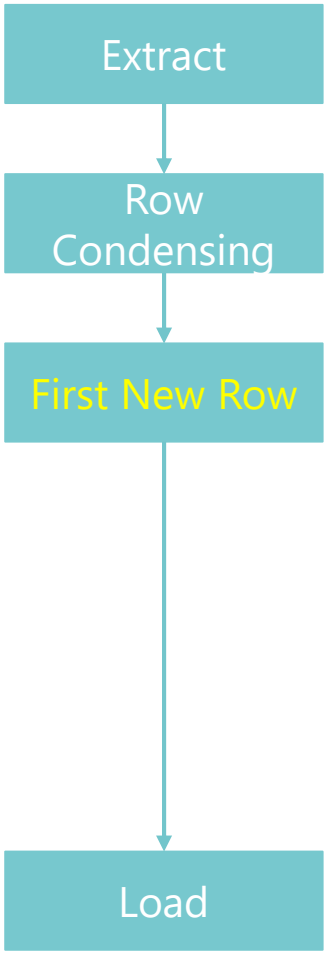
First New Row 1/2 – First Row




```
SELECT *
      ,ROW_NUMBER() OVER (PARTITION BY {hash_key}
                          ORDER BY dss_load_datetime ASC)
                                                AS first_new_row
FROM   (
  {sub_query}
) AS target
```

Satellite

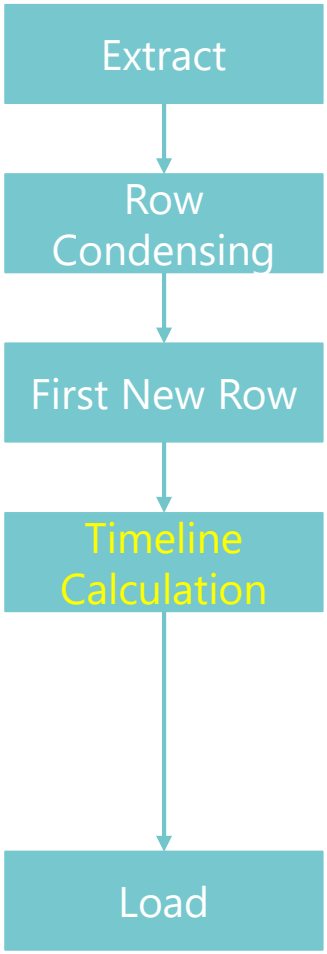
First New Row 2/2 – Last Satellite Row




```
SELECT *
FROM   (
  {sub_query}
) AS target
LEFT OUTER JOIN (
  SELECT *
  FROM   (
    SELECT *
          ,ROW_NUMBER() OVER (PARTITION BY {hub_hash_key}
                              ORDER BY dss_load_datetime DESC)
                                    AS last_existing_row
    FROM   {target}
  ) AS {target}
  WHERE last_existing_row = 1
) as current_rows
ON  current_rows.hub_hash_key = target.hub_hash_key
WHERE  (first_new_row = 1 AND change_hash <> change_hash
    OR (first_new_row > 1)
```

Satellite

Timeline: What & Why?



Alle Satelliten sind heutzutage INSERT ONLY. Schnellste Methode.

ABER, wir verschieben die Berechnung einer Timeline mit LoadEnddate oder letzter gültiger Datensatz weiter nach hinten in eine VIEW, PIT, etc.

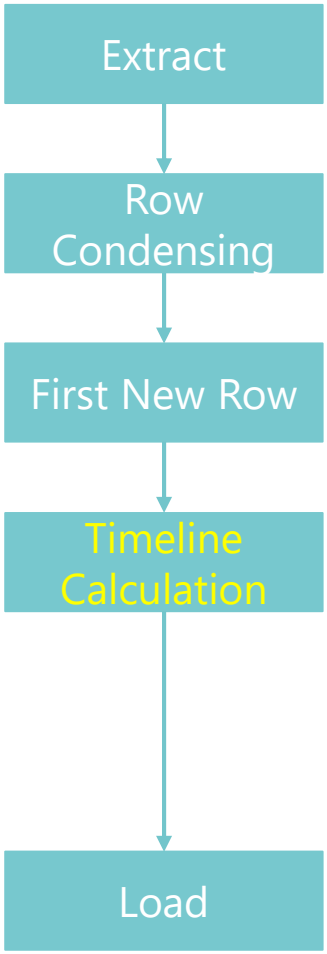
Bei großen Satelliten, ist eventuell ein persistierter Wert schneller zu erreichen, als eine nachträgliche Kalkulation.

Unsere Timelines:

- LoadDate
  - StartDate
  - EndDate
- EventDate
  - EffectivityDate
  - ExpiryDate

Satellite

Timeline: Verarbeitung



Level 1
- StartDate = LoadDate
- EndDate:
  - Pro HashKey:
    - LEAD(LoadDate,1,'9999-12-31)

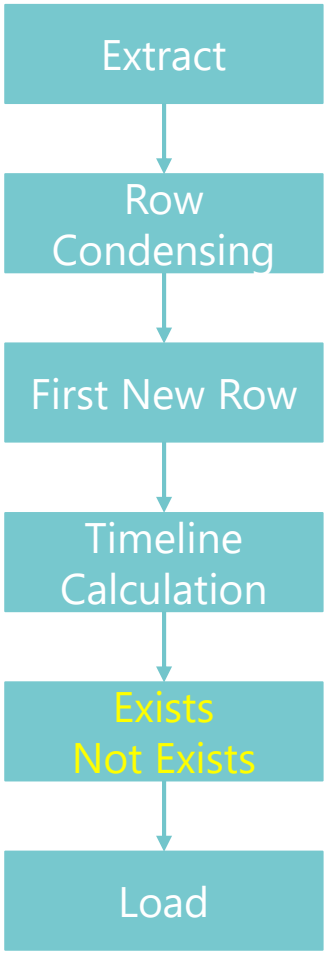
Level 2
- IsCurrent
  - CASE WHEN EndDate = 9999-12-31 THEN true ELSE false END

Level 3
- EffectivityDate = EventDate
- ExpiryDate
  - Pro HashKey oder Driving Key:
    - LEAD(EventDate) ORDER BY EventDate, LoadDate

ATTENTION: Post-Procedure
- Clean-up EndDates

Satellite

Exists/Not Exists: What & Why?



- Wir wollen keine Doubletten.
- Durch die Step-by-Step Verarbeiten könnte man den Ladeprozess in den Satelliten mehrfach anstossen.
- Primary Key Enforcement

Layer 1

- `WHERE NOT EXISTS` in Satellite

Satellite

Load

- Extract
- Row Condensing
- First New Row
- Timeline Calculation
- Exists Not Exists
- Load

Alle Felder und Kalkulation in die richtige Reihenfolge bringen für VIEW oder INSERT.

```
INSERT INTO ... / CREATE VIEW ... AS
SELECT
       hub_hash_key AS hub_hash_key
      ,data1 AS data1
      ,data2 AS data2
      ,dss_load_datetime AS dss_load_datetime
      ,dss_record_source AS dss_record_source
      ,dss_change_hash AS dss_change_hash
      ,dss_start_datetime AS dss_start_datetime
      ,SYSDATETIME() AS dss_create_datetime
FROM   (
  {sub_query}
) AS target
```

Satellite

Summary	
Extract	Laden der Daten aus der Stage Tabelle
Row Condensing	Eliminiere Doubletten gemäss ChangeHash
First New Row	Eliminiere 1. Datensatz, wenn er sich nicht ändert
Timeline Calculation	Timeline Calculation von StartDate, EndDate, EffectivityDate, ExpiryDate
Exists Not Exists	Eliminiere Datensätze, die bereits im Ziel vorhanden sind
Load	Aufbau des `SELECT` Statements fürs Ziel

Hub / Link

What & Why?

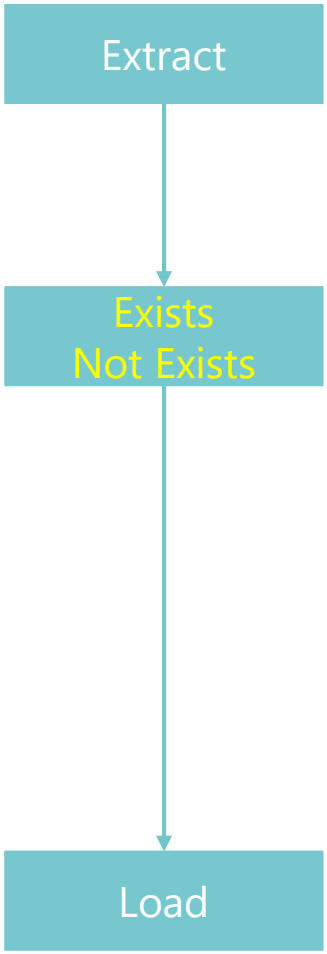
Extract

Im Hub oder Link wollen wir einen Datensatz pro HashKey speichern, bevorzugt den ersten.

Load

Hub / Link

Exists/Not Exists: Verarbeitung



SEE IN SATELLITE

Hub / Link

Exists/Not Exists: Verarbeitung

Extract

Exists
Not Exists

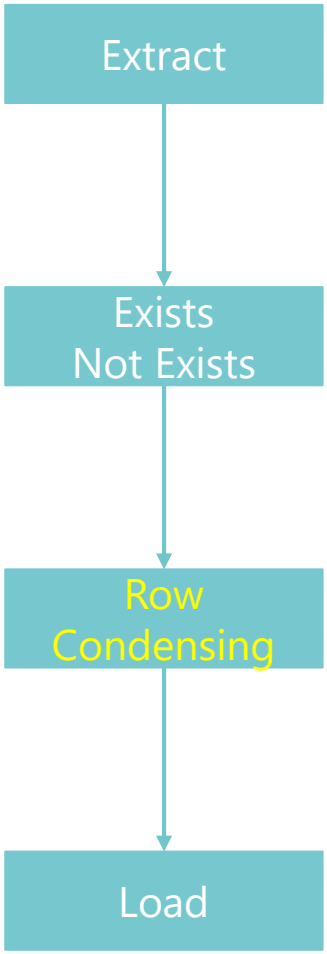
Load

Wir wollen einen Datensatz nur, wenn er noch nicht existiert.
… Aber welchen?

```
SELECT *
FROM   (
  {sub_query}
) AS {target}
WHERE NOT EXISTS (
  SELECT 1
  FROM   target AS target_not_exist
  WHERE  target.hub_hash_key = target_not_exist.hub_hash_key
     AND target.business_key = target_not_exist.business_key
)
```

Hub / Link

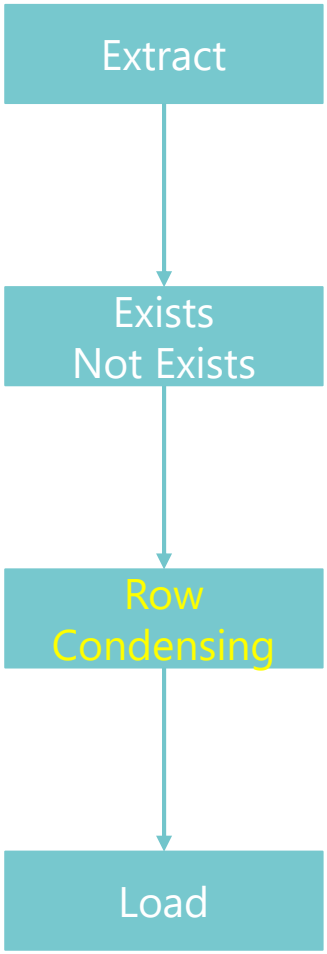
Row Codensing



SEE IN SATELLITE

Hub / Link

Row Codensing



Viele Beispiele nutzen MIN(LoadDate) als Filter. Das funktioniert, wenn im Stage Object es nur 1 Record Source gibt.

Batch-by-batch würde funktionieren. Aber gesamtes Data Set ist schneller. Daher sortieren.

Level 1

- Pro HashKey **UND** BusinessKey:
  - Sortiere nach LoadDate

Level 2

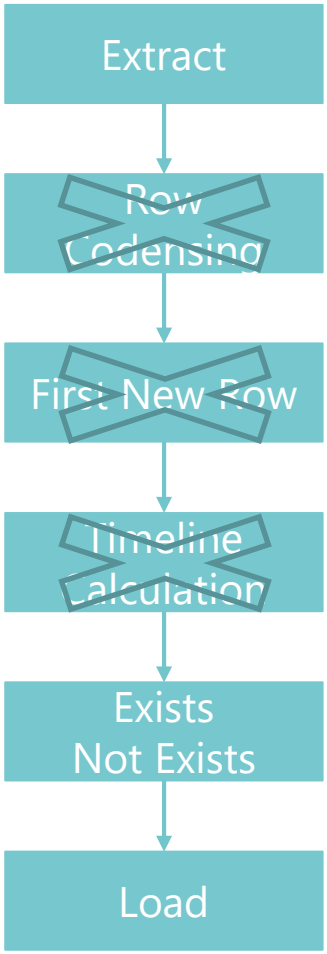
- WHERE RowNo = 1

Hub / Link

Summary	
Extract	Laden der Daten aus der Stage Tabelle
Exists Not Exists	Eliminiere Datensätze, die bereits im Ziel vorhanden sind
Row Condensing	Eliminiere Doubletten, hole ersten Datensatz
Load	Aufbau des `SELECT` Statements fürs Ziel

Dimension

Processing Options



Speichern eines Datensatzes pro Business Key. Überschreiben bei Änderung.

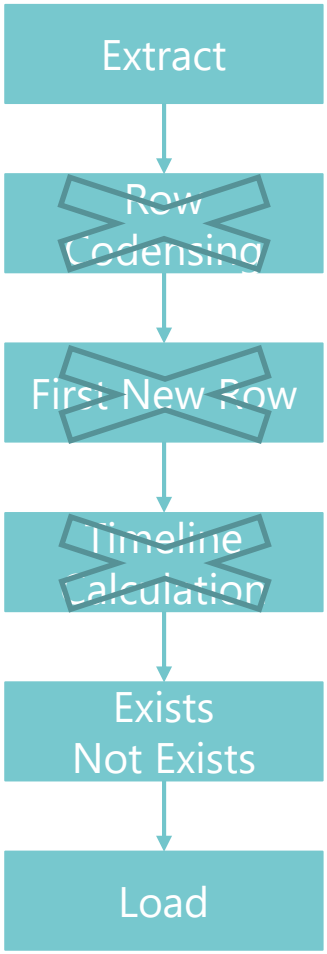
Bisher hatte ich noch keine Slowly Changing Dimensions – sorry.

```
UPDATE {dimension}
SET    ,column = stage_dimension.column
        ...
FROM   {stage_dimension}
WHERE  {stage_dimension}.business_key = {dimension}.business_key
```

```
INSERT INTO {dimension}
SELECT  *
FROM   {stage_dimension}
WHERE NOT EXISTS (
  SELECT 1
  FROM   {dimension} AS {dimension}_not_exist
  WHERE  {dimension}_not_exist.business_key = {dimension}.business_key
)
```

Facts

Processing Options




```
TRUNCATE TABLE {fact}
```

```
UPDATE {fact}
SET    ,column = stage_fact.column
        ...
FROM   {stage_fact}
WHERE  {stage_fact}.business_key = {fact}.business_key
```

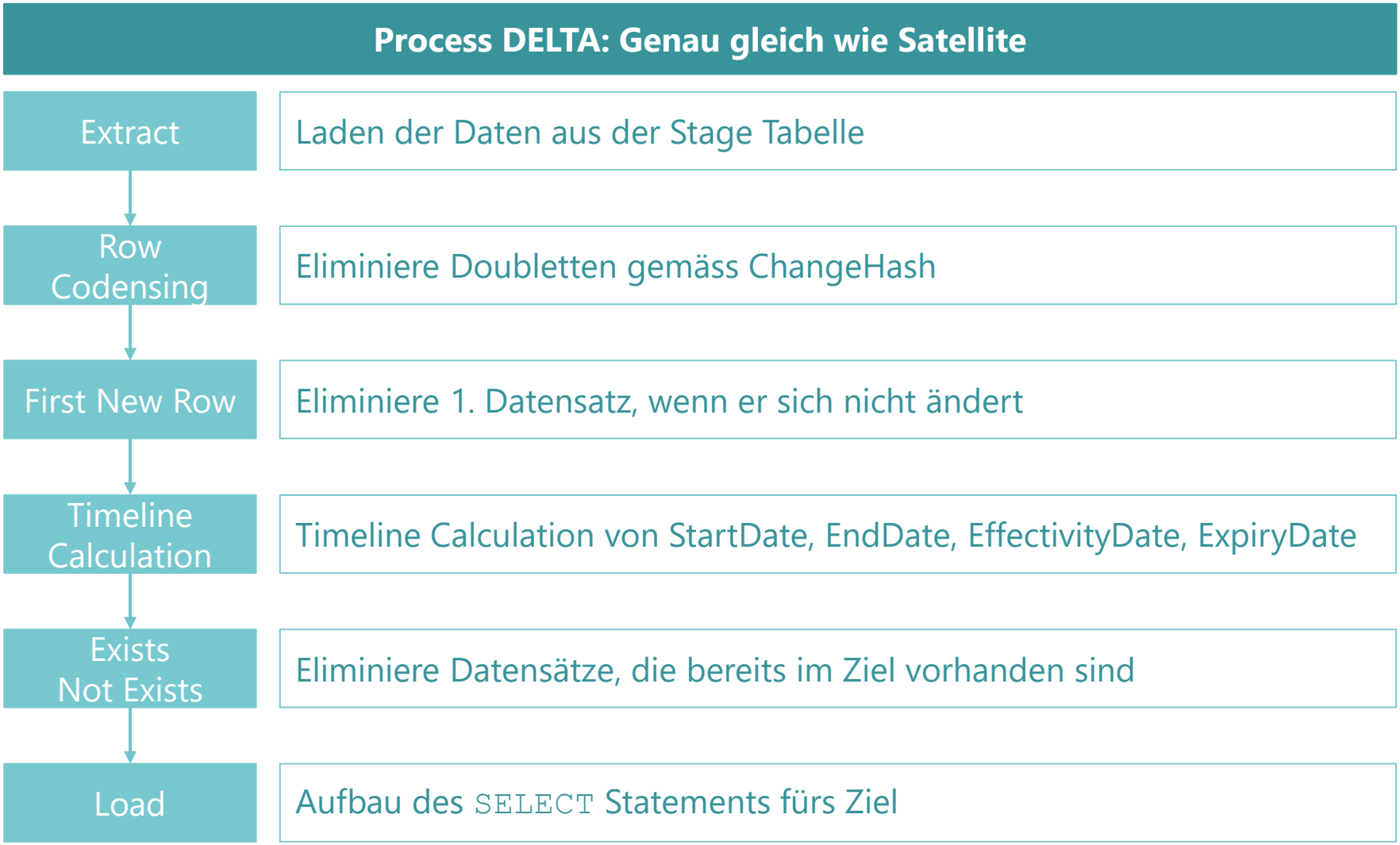
```
DELETE {fact}
FROM   {stage_fact}
      ,{fact}
WHERE  {stage_fact}.business_key = {fact}.business_key
```

```
INSERT INTO {fact}
SELECT  *
FROM   {stage_fact}
WHERE NOT EXISTS (
  SELECT 1
  FROM   {fact} AS {fact}__not_exist
  WHERE  {fact}__not_exist.business_key = {fact}.business_key
)
```

Data Store

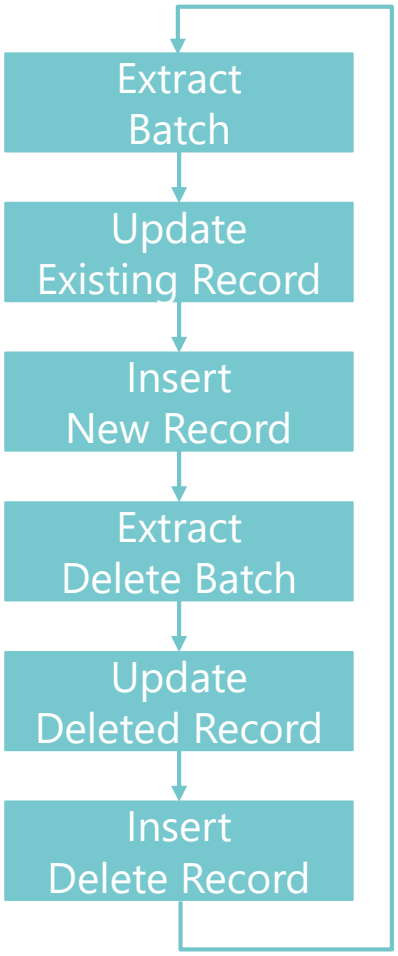
Data Set Types	
Full	Wir erhalten mit jedem Import einen kompletten Datensatz. Z. B. einen Export der Kunden einer Applikation, ein Excel, etc. Wir können erkennen, ob Datensätze gelöscht wurden
Full Business Key	Wir erhalten eine komplette Liste an validen Business Keys. Ladezyklen sind bedeutend kürzer als mit FULL. Wir können erkennen, ob Datensätze gelöscht wurden
Delta	Wir erhalten immer nur neue Daten oder Änderungen an bestehenden Daten. Z. B. Logfiles, Transactions, etc. Keine Erkennung von gelöschten Datensätzen möglich.
CDC	Enthält jede Datenbankänderung INSERT, UPDATE, DELETE Gelöschte Datensätze sind eine eigene Transaktion.

Data Store


Process DELTA: Genau gleich wie Satellite
Extract
Laden der Daten aus der Stage Tabelle
Row Codensing
Eliminiere Doubletten gemäss ChangeHash
First New Row
Eliminiere 1. Datensatz, wenn er sich nicht ändert
Timeline Calculation
Timeline Calculation von StartDate, EndDate, EffectivityDate, ExpiryDate
Exists Not Exists
Eliminiere Datensätze, die bereits im Ziel vorhanden sind
Load
Aufbau des SELECT Statements fürs Ziel

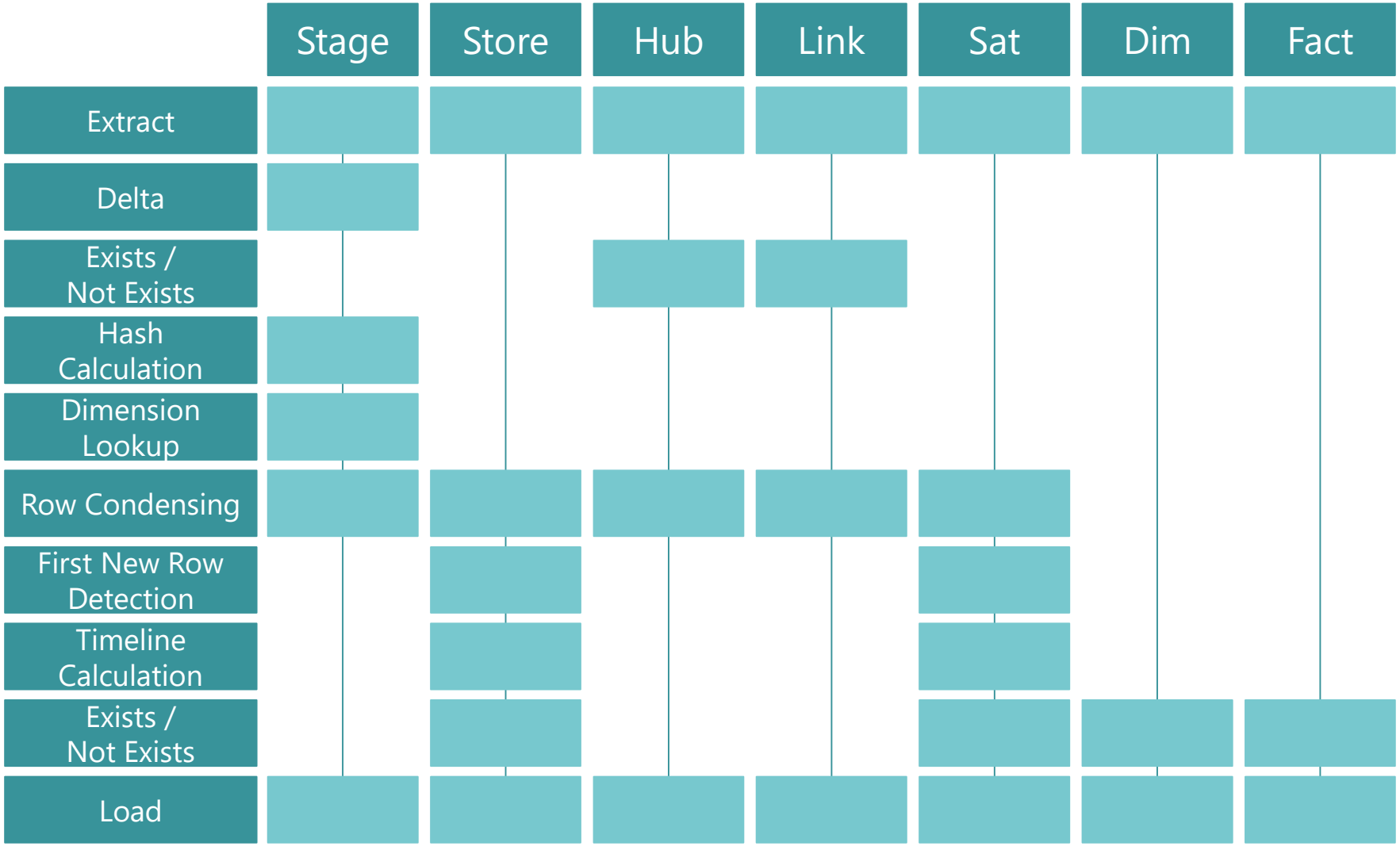
Data Store

Process FULL: Batch-By-Batch



- Extrahiere ein Batch je LoadData
- Update Timeline des zu aktualisierenden Datensatzes
- Füge neue Datensätze hinzu (genau gleich wie Delta oder Satellite)
- Füge nicht existierende BK aus dem Data Store in den Delete Batch
- Update Timeline des zu aktualisierenden Datensatzes
- Füge einen “Delete Record” mit dem letzen bekannten Zustand ein

Matrix der Muster



Eckhard Zemp


https://www.xing.com/profile/eckhard_zemp/


https://www.linkedin.com/in/eckhard-zemp/


https://twitter.com/EckisWelt


https://www.zemp.ch

THANK YOU

For your attention!