Vertica Technical Overview for Oracle users

Northern California Oracle Users Group August 2014, San Ramon, CA
David Abercrombie

Tapjoy® | REWARDING MOBILE

Vertica’s amazing features

- Fast joins and aggregates of huge tables
- Fast data load
- Highly compressed data
    - Demoralization is almost free

- Advanced analytics
    - HyperLogLog cardinality estimate synopsis

- Business intelligence
    - “Self-service”
    - Snowflake schema

- Operational Datastore
    - Ad hoc analysis
    - Customized applications

Vertica analytics

- Windowing functions (of course)
- Time series
  - Interpolation and gap filling
  - Event based windows
  - Sessionization
- Approximate count distinct
  - HyperLogLog synopsis
- R and Distributed R
- Sentiment via “Pulse”
- Geospatial via “Place”
  - 2-D Open Geospatial Consortium (OGC) standards

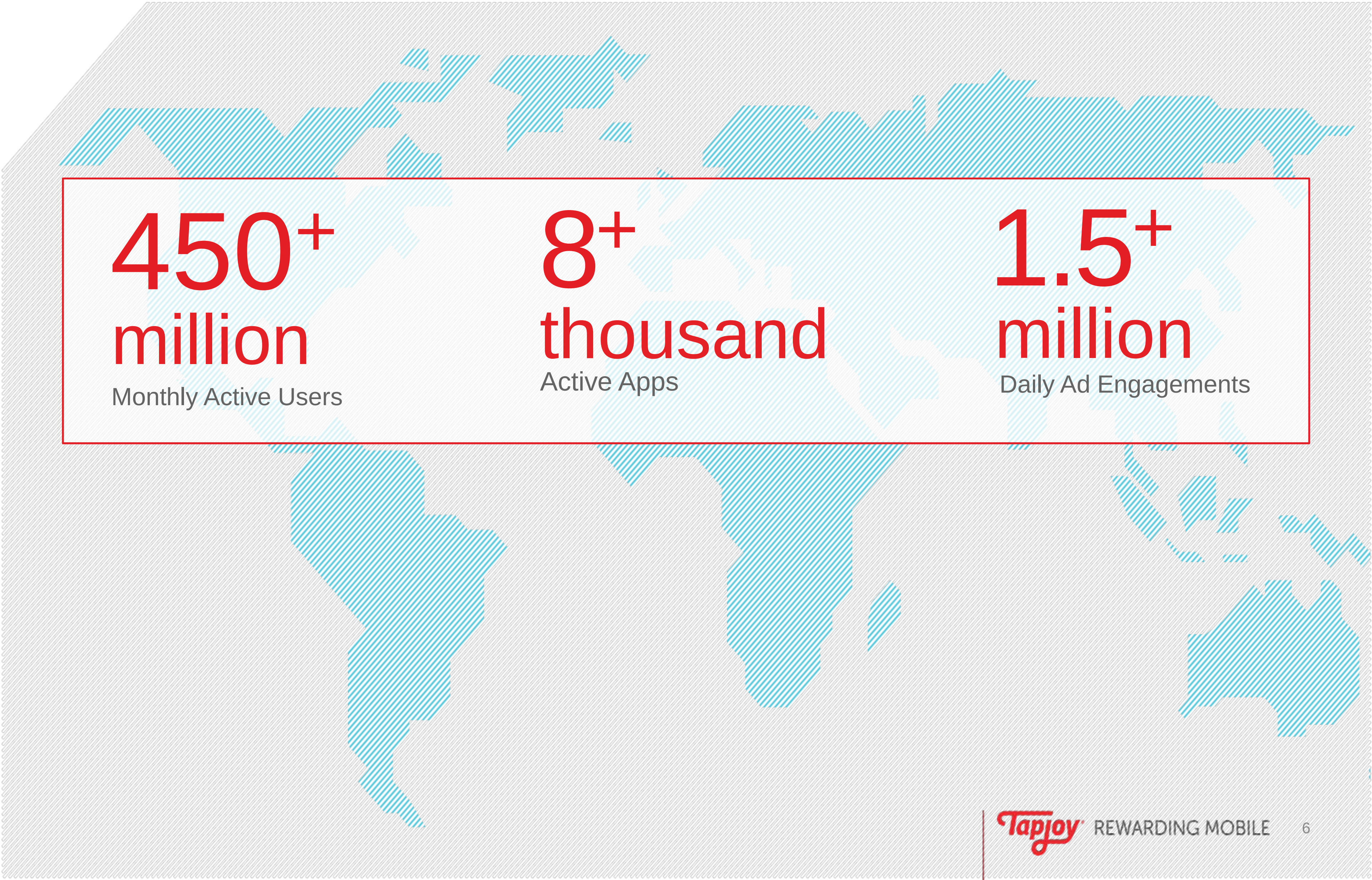
Agenda

- Tapjoy
- Architecture
- Data storage
- Tapjoy example
- Approximate count distinct
  - HyperLogLog synopsis

Tapjoy

How It Works

Massive Scale, Global Reach, Thousands of Apps



450^{+} million

Monthly Active Users

8^{+} thousand

Active Apps

1.5^{+} million

Daily Ad Engagements

My Vertica & MicroStrategy project

- Built MicroStrategy and Vertica DW together
- Indispensible: 200+ users

- 100 terabytes before compression
  - 15 terabytes per day loaded
  - 75 terabytes operational data store
  - 25 terabytes snowflake for MicroStrategy
  - 500 metrics, 40 attributes, 15 dimensions in MicroStrategy

- Less than three full-time equivalents (FTE)
  - Vertica DBA labor is minimal
  - MicroStrategy Cloud, so no administration

Vertica Architcture

Vertica architecture

- Column store
  - Run length encoding
- Shared nothing parallel
  - Massive scalability
- Full featured SQL
  - Analytic extensions
- High availability
  - No master node
- Commodity hardware
- Easy maintenance
- Requires data structure engineering

Vertica does not have these

- No indexes
- No heap storage
- No block buffer cache
- No PL/SQL
- No alert log
- No redo log
- No wait interface
- No replication

- Referential integrity works strangely

Vertica inefficiencies

- DELETE is inefficient
  - Delete vector
- UPDATE is inefficient
  - Deletes old, inserts new
- Single row operations are inefficient
  - INSERT … VALUES is inefficient
  - No B-tree index for one-row lookup
  - No nested-loop query operator

Data transform in Vertica

- Extract, Load, Transform “ELT”

- Partitions
  - Dynamically defined
  - Drop and swap
  - But limited to about 1000

- Staging tables
  - TRUNCATE
  - INSERT … SELECT … JOIN

Vertica conveniences

- Great support and engineers
- Feels like PostgreSQL
  - vsql, users & schemas, search_path, …
- Great EXPLAIN and PROFILE
- Good cluster management
- Good resource manager
- Great docs
- Hadoop integration (stage data in native format)
- JSON via “Flex”
- Data Collector tables
  - Like Automatic Workload Repository (AWR)
- Management console

Vertica Storage

Vertica Data Storage

- Column store
  - Projections
  - Sorted

- Run length encoding
  - Query execution example

- Write-once read-only
  - Deletes and defragmentation

- Parallel
  - Segmentation
  - Local joins

Disk data structure = “projection”

- Projection: subset of columns
  - Super-projection has all columns (must be one)
- Sorted and compressed
  - Almost like Oracle IOTs
  - No heap storage
- Each disk file has data from only once column
  - Columns glued together at run time
- May be parallel or replicated
- Projection design key to performance
- Can have more than one per table

Run Length Encoding (RLE)

- Data must be sorted
- Good for low-cardinality data

- Example: sending a fax
  - Do not describe each pixel, one a time
  - Instead, use “run lengths”
  - 50 white, 4 black, 60 white, 5 black, …

- Very efficient, minimal CPU

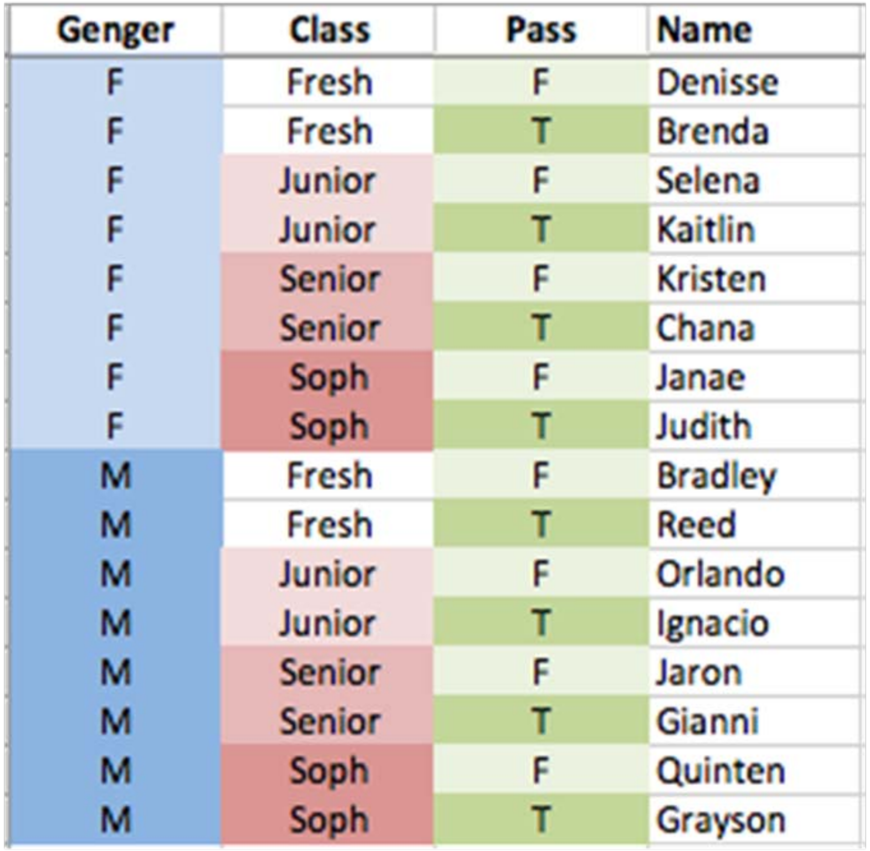
Run Length Encoding (RLE)

- Minimizes IO and disk storage

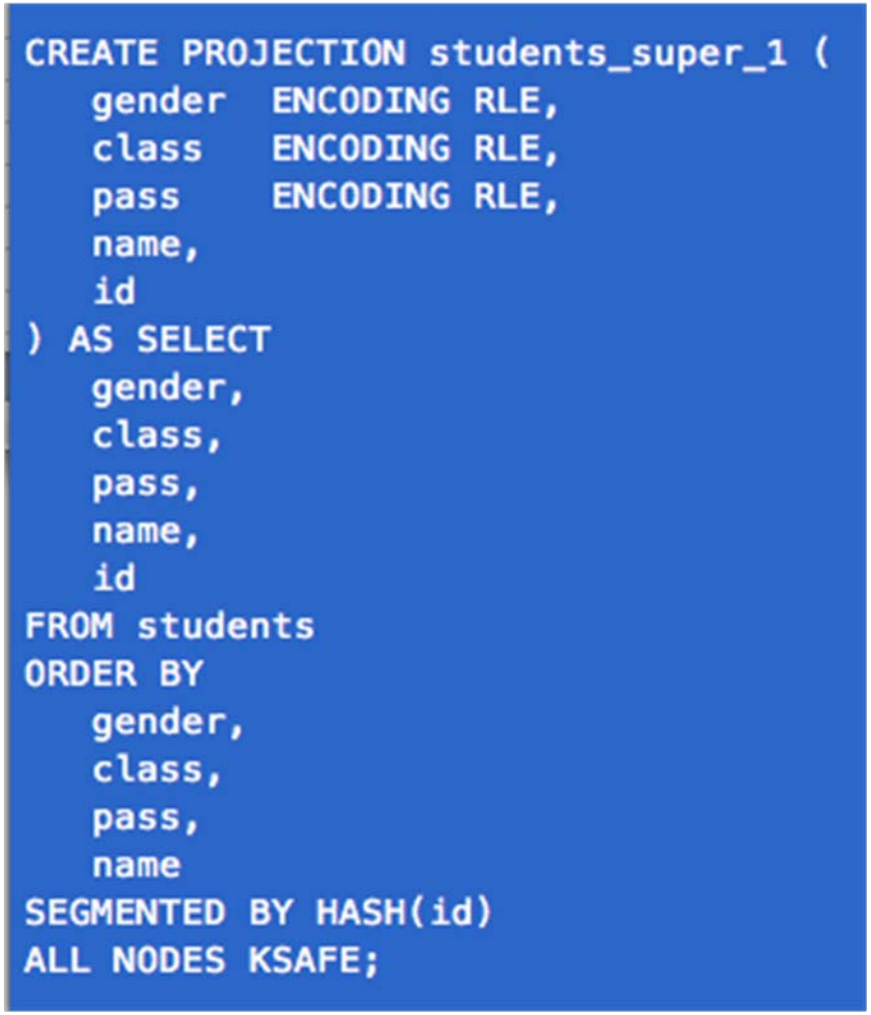
- But deletes are expensive
  - Data not actually deleted
  - Additional file listing deleted rows
  - Additional processing overhead
  - Background cleanup process

- Updates even more expensive
  - Insert and a delete

Example data and projection

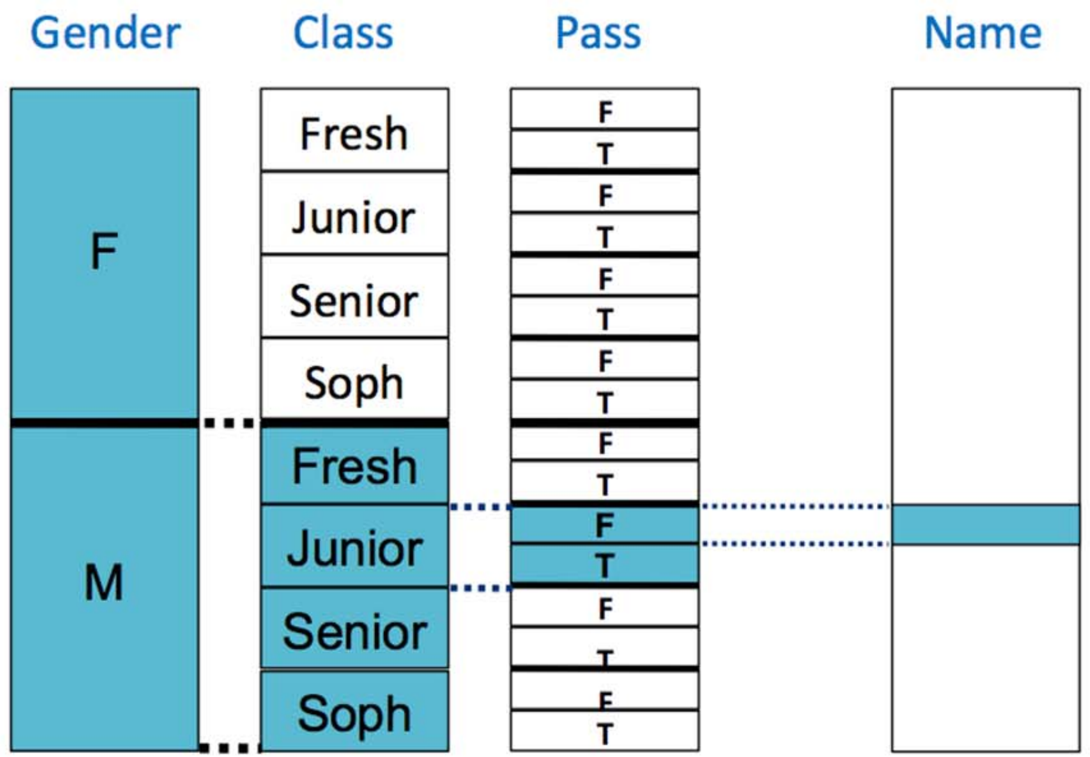


Genger	Class	Pass	Name
F	Fresh	F	Denisse
F	Fresh	T	Brenda
F	Junior	F	Selena
F	Junior	T	Kaitlin
F	Senior	F	Kristen
F	Senior	T	Chana
F	Soph	F	Janae
F	Soph	T	Judith
M	Fresh	F	Bradley
M	Fresh	T	Reed
M	Junior	F	Orlando
M	Junior	T	Ignacio
M	Senior	F	Jaron
M	Senior	T	Gianni
M	Soph	F	Quinten
M	Soph	T	Grayson



```
CREATE PROJECTION students_super_1 (
   gender  ENCODING RLE,
   class   ENCODING RLE,
   pass    ENCODING RLE,
   name,
   id
) AS SELECT
   gender,
   class,
   pass,
   name,
   id
FROM students
ORDER BY
   gender,
   class,
   pass,
   name
SEGMENTED BY HASH(id)
ALL NODES KSAFE;
```

Run Length Encoding, Column Store



```
SELECT Name
FROM Students
WHERE Class='Junior'
   AND Gender='M'
   AND Pass='F'
```

What about DML?

- Rows loaded into memory: WOS
  - Sorted columns written to disk: ROS
  - Files written once, then read-only

- DELETE operator writes a list of deleted rows
  - Called “delete vectors”
- SELECT operator reads delete vectors
  - Ignores deleted rows that it had scanned

- UPDATE is DELETE with INSERT

- Background defragmentation
  - Tuple mover mergeout

Clustered Database

- *“Move the computation to the data, never move the data to the computation”*
  - Dr. Michael Stonebraker

- *“Moving Computation is Cheaper than Moving Data”*
  - Hadoop documentation

Parallel, “share nothing”

- Data distributed: “segmentation”
  - Like automatic “sharding”
- Many segmentation choices
- Multiple physical storage designs for single table

- Can replicate instead (dimensions)

- Local joins
  - Segment on subset of join key
  - Identically Segmented Projections (ISP)

Local Joins – two node example

Customer ID	Name
1	Sam
3	Joe
5	Mary

Customer ID	Name
2	Cathy
4	Bill
6	Jane

Order ID	Customer ID	Date
1002	3	6/1/13
1005	1	7/1/13
1006	3	7/7/13
1008	5	8/1/13

Order ID	Customer ID	Date
1003	4	5/6/13
1004	6	7/2/13
1007	2	7/8/13
1009	4	8/2/13

Projection design key to performance

- Almost like designing Oracle Index Organized Table

- Compression guidelines:
  - Low cardinality columns early in sort order
  - Query predicates early in sort order
  - Pipelined GROUP BY and MERGE join

- Segmentation guidelines
  - Local joins: identically segmented projections
  - Replicate dimension lookup tables

- Need data distribution and query SQL
  - Use “Database Designer” if you have both, or
  - More common: think hard if missing data or SQL

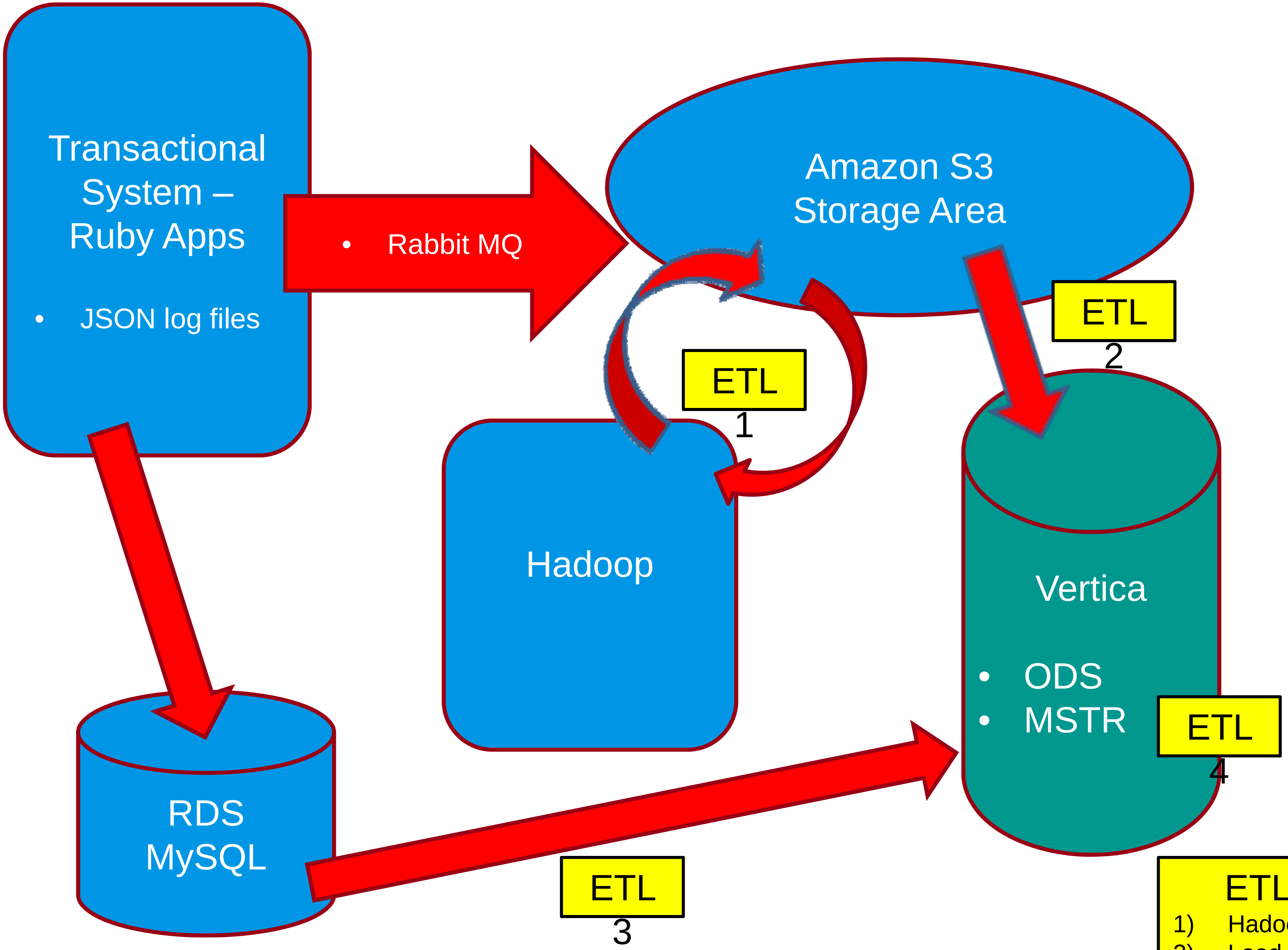
Referential Integrity (RI)

- Can create primary and foreign keys

- But can insert bad data without error
- Error message at query runtime!

- We do not use Vertica referential integrity
  - Check during ETL process with SQL
  - Many important constraints cannot be handled by RI
    - Example: non-overlapping Type-2 SCD rows

Tapjoy Example



Transactional System – Ruby Apps

- JSON log files

- Rabbit MQ

Amazon S3 Storage Area

Hadoop

Vertica

- ODS
- MSTR

RDS MySQL

ETL 1

ETL 2

ETL 3

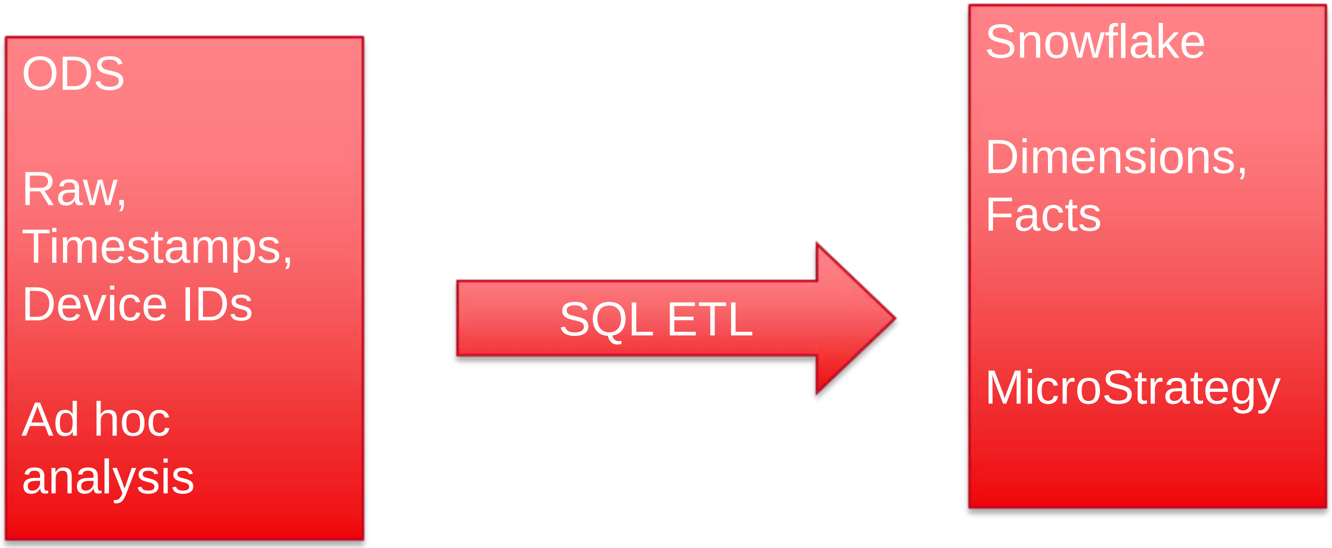
ETL 4

ETLs

1) Hadoop
2) Load Vertica
3) Load Vertica
4) Aggregation

SQL based ETL within single Vertica DB

- Operational Data Store (ODS)
  - Raw transaction-level data
  - 15 minute incremental loads
- MicroStrategy snowflake schema
  - One-hour granularity and load



SQL ETL

- Two projections on ODS source tables
  - One for ad hoc analysis
  - Other for MSTR ETL extraction
  - MSTR ETL order of magnitude faster
  - No significant increase in ODS load times

- Type 2 Slowly changing dimensions
  - We use UPDATE here only

- Vertica’s rich, full-featured SQL
  - Can deal with complexity

- Homegrown scheduling framework

Snowflake schema

- Moderately denormalized
  - Child has keys for parent (FK) and all tables above
  - DESC columns are NOT denormalized
  - Facts have many keys

- Denormalization is very cheap in Vertica
  - Run length encoding
  - Low cardinality

- Example denormalized fact table:
  - **1700 gigabytes total** disk storage
  - **37 billion rows**
  - Hourly granularity
  - Only **14 kilobytes** for denormalized Month column

Date early in fact sort order

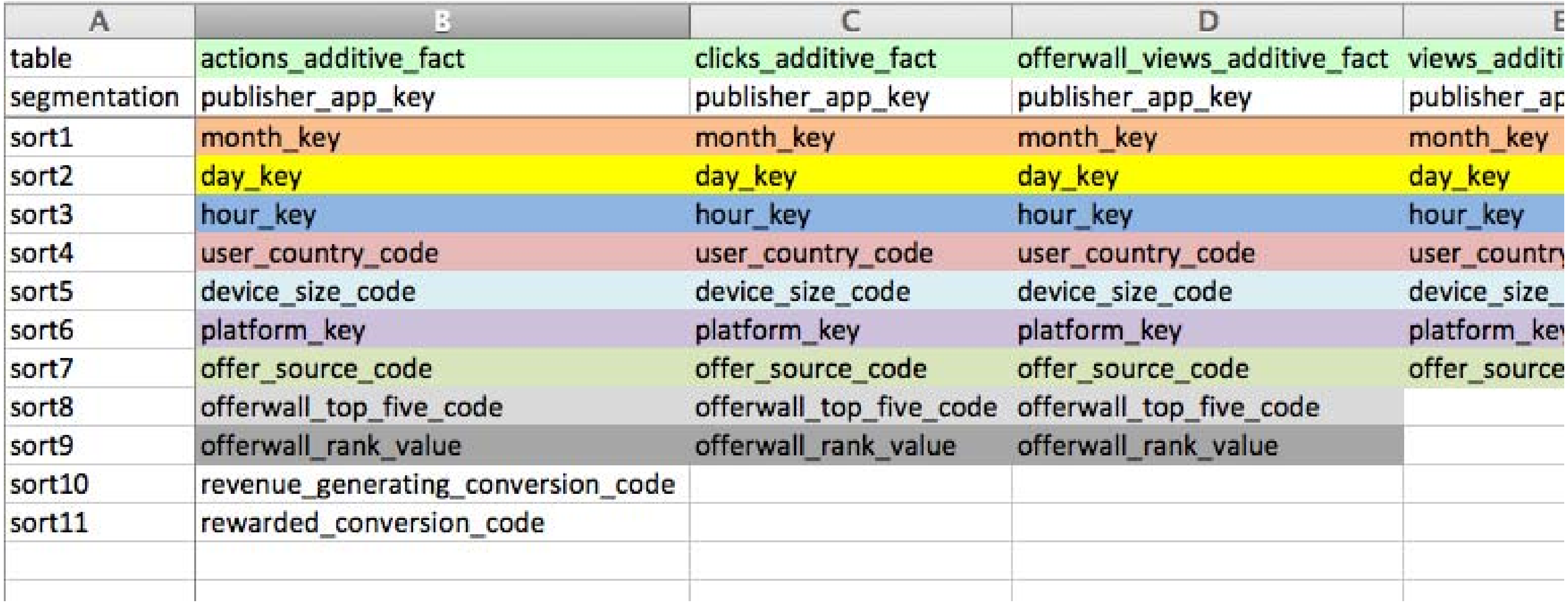
- Almost all MicroStrategy queries use date filters
- Dates and times are low cardinality

- An obvious choice for start of sort order

- Report execution time scales with date range

Consistent fact projection design

- Facilitates joining facts via MERGE (details later)
- ETL staging tables also



A	B	C	D	E
table	actions_additive_fact	clicks_additive_fact	offerwall_views_additive_fact	views_additi
segmentation	publisher_app_key	publisher_app_key	publisher_app_key	publisher_ap
sort1	month_key	month_key	month_key	month_key
sort2	day_key	day_key	day_key	day_key
sort3	hour_key	hour_key	hour_key	hour_key
sort4	user_country_code	user_country_code	user_country_code	user_countr
sort5	device_size_code	device_size_code	device_size_code	device_size_
sort6	platform_key	platform_key	platform_key	platform_ke
sort7	offer_source_code	offer_source_code	offer_source_code	offer_source
sort8	offerwall_top_five_code	offerwall_top_five_code	offerwall_top_five_code	
sort9	offerwall_rank_value	offerwall_rank_value	offerwall_rank_value	
sort10	revenue_generating_conversion_code			
sort11	rewarded_conversion_code			

Data Volumes

- Full database
  - 104 terabytes uncompressed (est.)
  - 27 terabytes disk used (volumes are 80% full)
  - Compression ratio: 3.8

- MicroStrategy schema, including ETL staging
  - 21.4 terabytes uncompressed (est.)
  - 2.7 terabytes disk used (10% of total)
  - Compression ratio: 7.9

5 node cluster

- 24 cores per node
- 128 gigabytes RAM per node (5.3 per core)
- 7 terabyte disk per node (RAID 10)
- 1 GB network
- K-Safe = 1

- In the recommended ballpark (Vertica docs PDF)

- No significant complaints about report speed!

Unique metrics

and

Count Distinct

Count Distinct Challenges

- Expensive and memory intensive

- Use pre-computed counts?
  - Complicates ETL
  - Limits analysis to predetermined dimensions
  - MicroStrategy likes to sum

- Vertica solutions:
  - Approximate Count Distinct
  - Approximate Count Distinct Synopsis
  - Approximate Count Distinct of Synopsis

Documentation and Resources

- Vertica blog
  - Avoiding the OLAP Cliff for Count Distinct Queries in Vertica

- Vertica documentation
  - APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS

- Algorithm
  - HyperLogLog: the analysis of a near-optimal
  - cardinality estimation algorithm by P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier

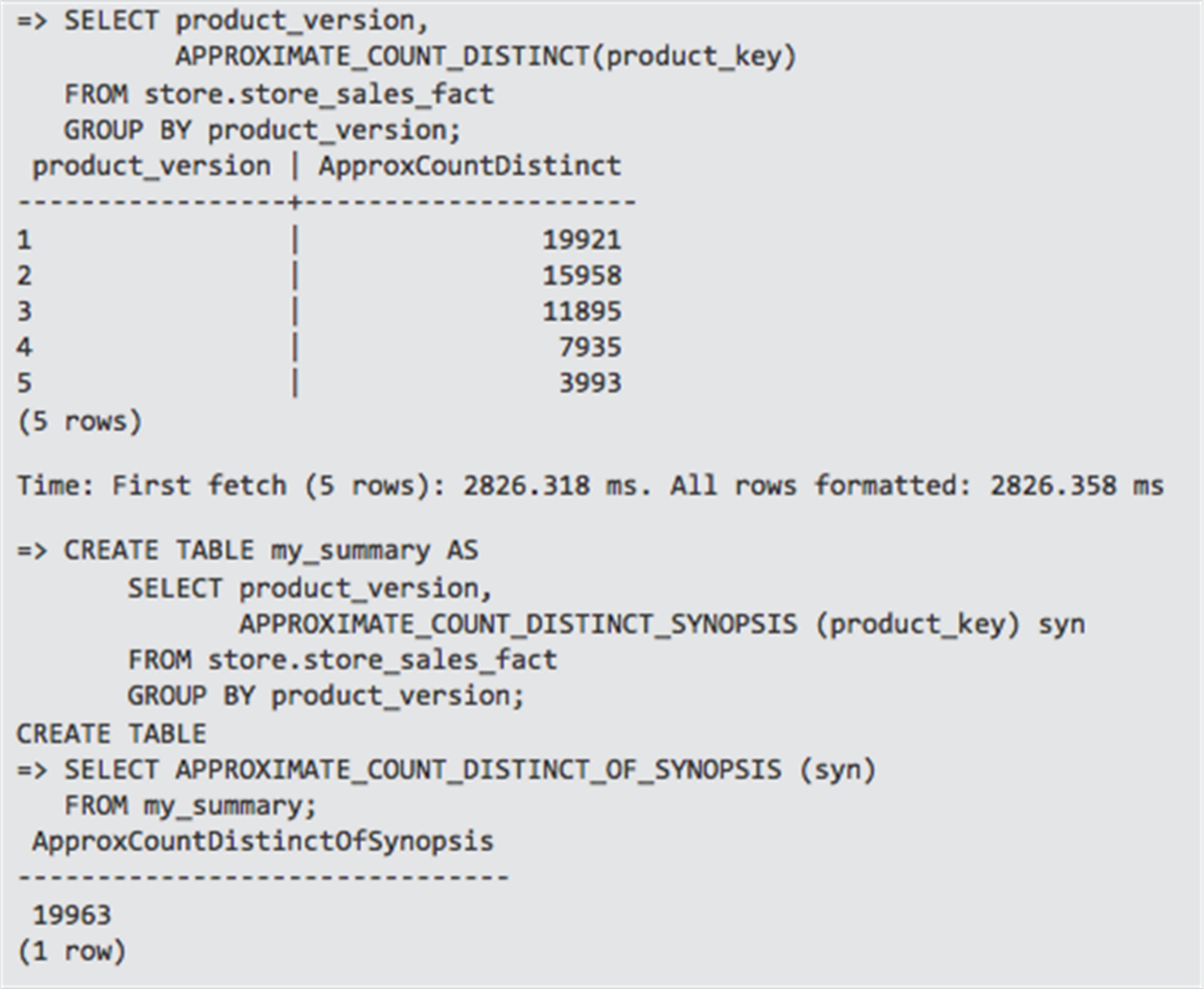
Two approximate methods

- Create and use stored “synopsis,” or
- Direct, without “synopsis”

- What is a synopsis?
  - Can be aggregated at a new level, avoids double counting
  - VARBINARY(49154), mumurhash()

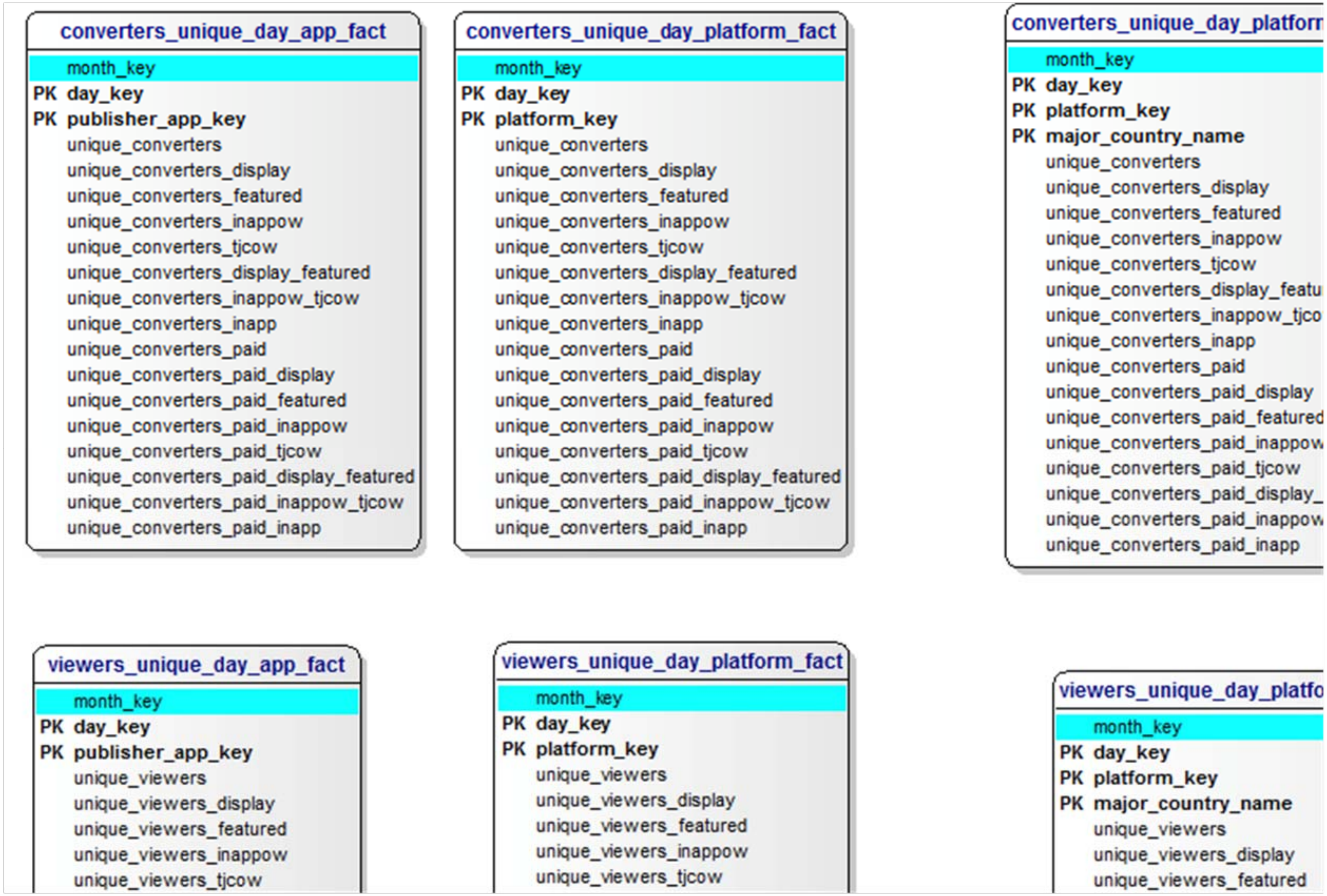
- Synopsis workflow
  - Create with approximate_count_distinct_synopsis()
  - Query with approximate_count_distinct_of_synopsis()

Example from Documentation



```
=> SELECT product_version,
          APPROXIMATE_COUNT_DISTINCT(product_key)
   FROM store.store_sales_fact
   GROUP BY product_version;
 product_version | ApproxCountDistinct
-----------------+--------------------
1                |               19921
2                |               15958
3                |               11895
4                |                7935
5                |                3993
(5 rows)

Time: First fetch (5 rows): 2826.318 ms. All rows formatted: 2826.358 ms

=> CREATE TABLE my_summary AS
       SELECT product_version,
              APPROXIMATE_COUNT_DISTINCT_SYNOPSIS (product_key) syn
       FROM store.store_sales_fact
       GROUP BY product_version;
CREATE TABLE
=> SELECT APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS (syn)
   FROM my_summary;
 ApproxCountDistinctOfSynopsis
-------------------------------
 19963
(1 row)
```

Pre-computed counts are ugly



	converters_unique_day_app_fact
	month_key
PK	day_key
PK	publisher_app_key
	unique_converters
	unique_converters_display
	unique_converters_featured
	unique_converters_inappow
	unique_converters_tjcow
	unique_converters_display_featured
	unique_converters_inappow_tjcow
	unique_converters_inapp
	unique_converters_paid
	unique_converters_paid_display
	unique_converters_paid_featured
	unique_converters_paid_inappow
	unique_converters_paid_tjcow
	unique_converters_paid_display_featured
	unique_converters_paid_inappow_tjcow
	unique_converters_paid_inapp

	converters_unique_day_platform_fact
	month_key
PK	day_key
PK	platform_key
	unique_converters
	unique_converters_display
	unique_converters_featured
	unique_converters_inappow
	unique_converters_tjcow
	unique_converters_display_featured
	unique_converters_inappow_tjcow
	unique_converters_inapp
	unique_converters_paid
	unique_converters_paid_display
	unique_converters_paid_featured
	unique_converters_paid_inappow
	unique_converters_paid_tjcow
	unique_converters_paid_display_featured
	unique_converters_paid_inappow_tjcow
	unique_converters_paid_inapp

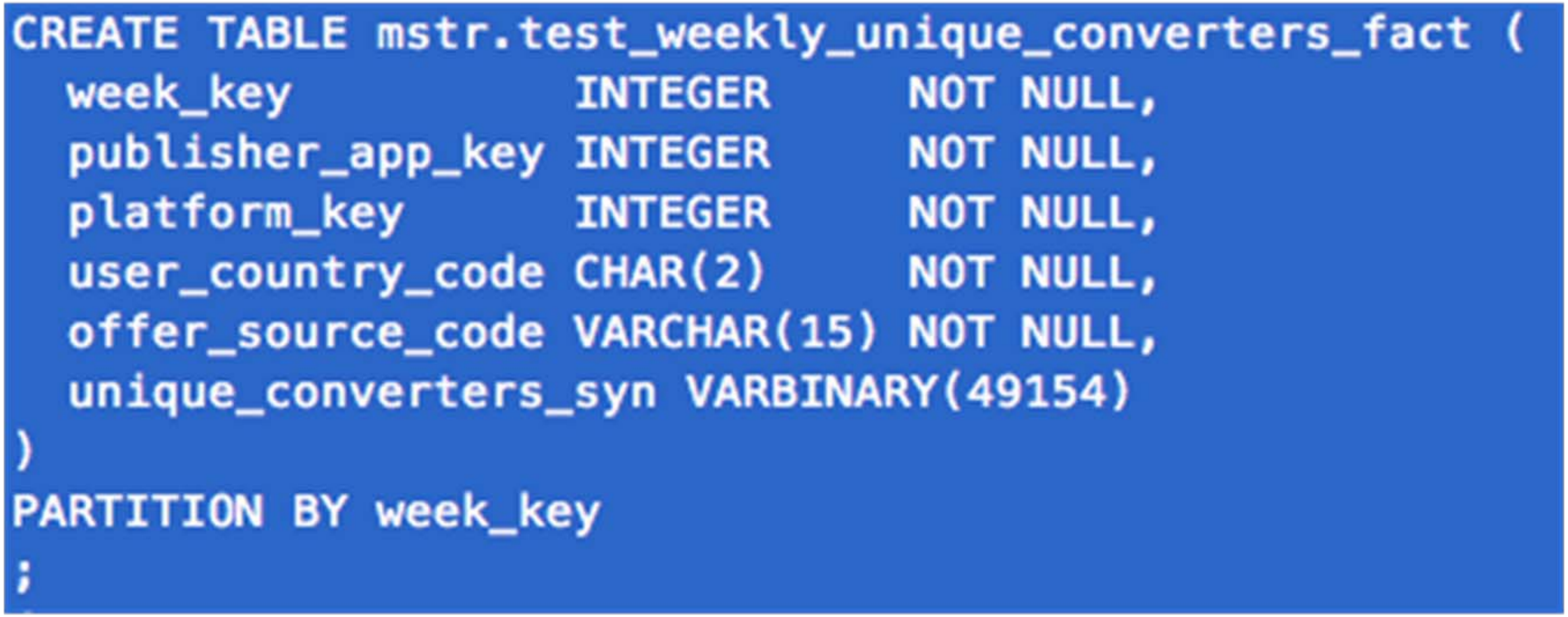
	converters_unique_day_platforn
	month_key
PK	day_key
PK	platform_key
PK	major_country_name
	unique_converters
	unique_converters_display
	unique_converters_featured
	unique_converters_inappow
	unique_converters_tjcow
	unique_converters_display_featu
	unique_converters_inappow_tjco
	unique_converters_inapp
	unique_converters_paid
	unique_converters_paid_display
	unique_converters_paid_featured
	unique_converters_paid_inappow
	unique_converters_paid_tjcow
	unique_converters_paid_display_
	unique_converters_paid_inappow
	unique_converters_paid_inapp

	viewers_unique_day_app_fact
	month_key
PK	day_key
PK	publisher_app_key
	unique_viewers
	unique_viewers_display
	unique_viewers_featured
	unique_viewers_inappow
	unique_viewers_tjcow

	viewers_unique_day_platform_fact
	month_key
PK	day_key
PK	platform_key
	unique_viewers
	unique_viewers_display
	unique_viewers_featured
	unique_viewers_inappow
	unique_viewers_tjcow

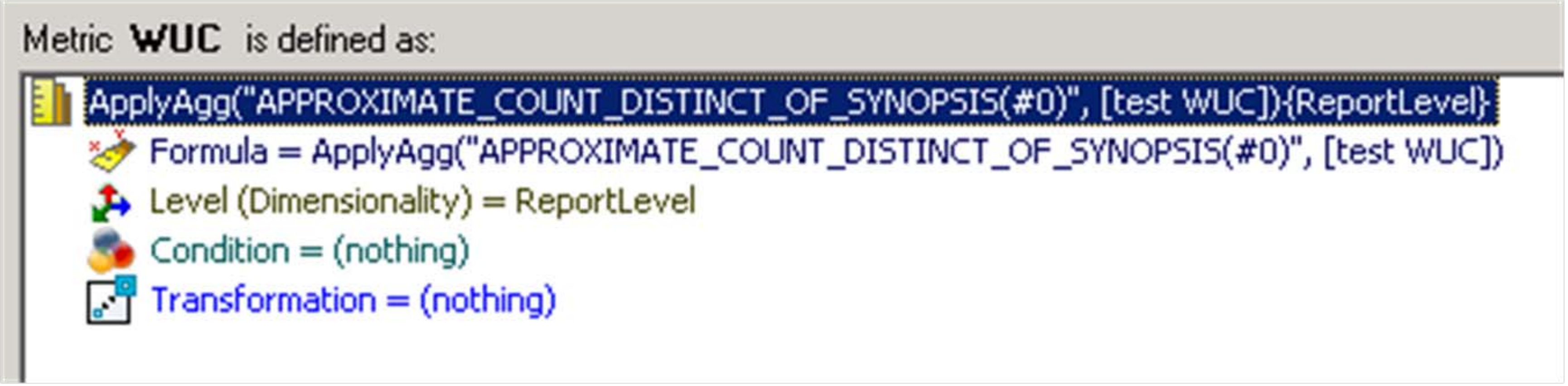
	viewers_unique_day_platfo
	month_key
PK	day_key
PK	platform_key
PK	major_country_name
	unique_viewers
	unique_viewers_display
	unique_viewers_featured

A much simpler, mode flexible design



```
CREATE TABLE mstr.test_weekly_unique_converters_fact (
  week_key          INTEGER     NOT NULL,
  publisher_app_key INTEGER     NOT NULL,
  platform_key      INTEGER     NOT NULL,
  user_country_code CHAR(2)     NOT NULL,
  offer_source_code VARCHAR(15) NOT NULL,
  unique_converters_syn VARBINARY(49154)
)
PARTITION BY week_key
;
```

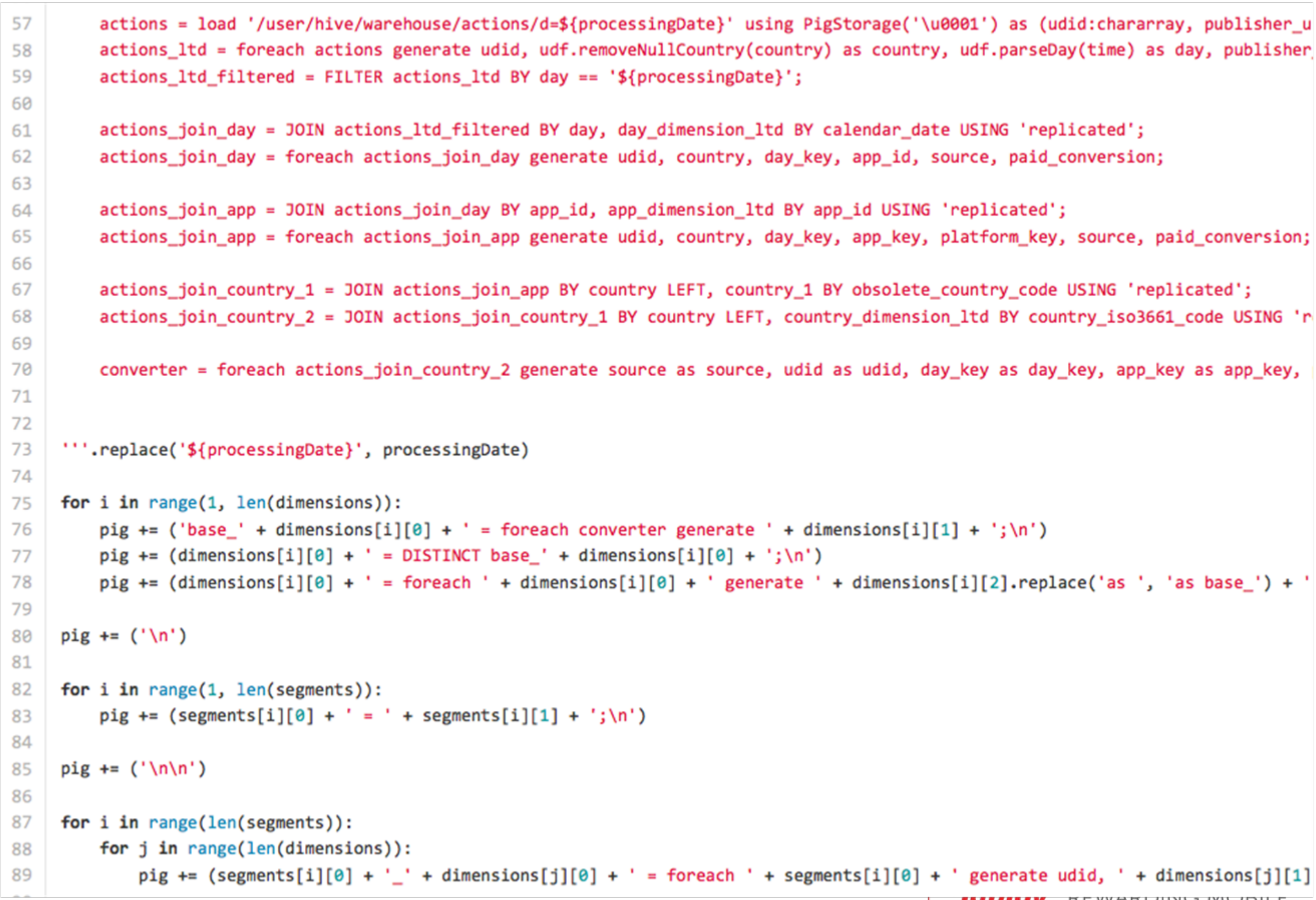
The easy part, adding to MicroStrategy ApplyAgg() pass-through function



Metric **WUC** is defined as:

```
ApplyAgg("APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS(#0)", [test WUC]){ReportLevel}
    Formula = ApplyAgg("APPROXIMATE_COUNT_DISTINCT_OF_SYNOPSIS(#0)", [test WUC])
    Level (Dimensionality) = ReportLevel
    Condition = (nothing)
    Transformation = (nothing)
```

Replaced ugly ETL with 4 insert/select

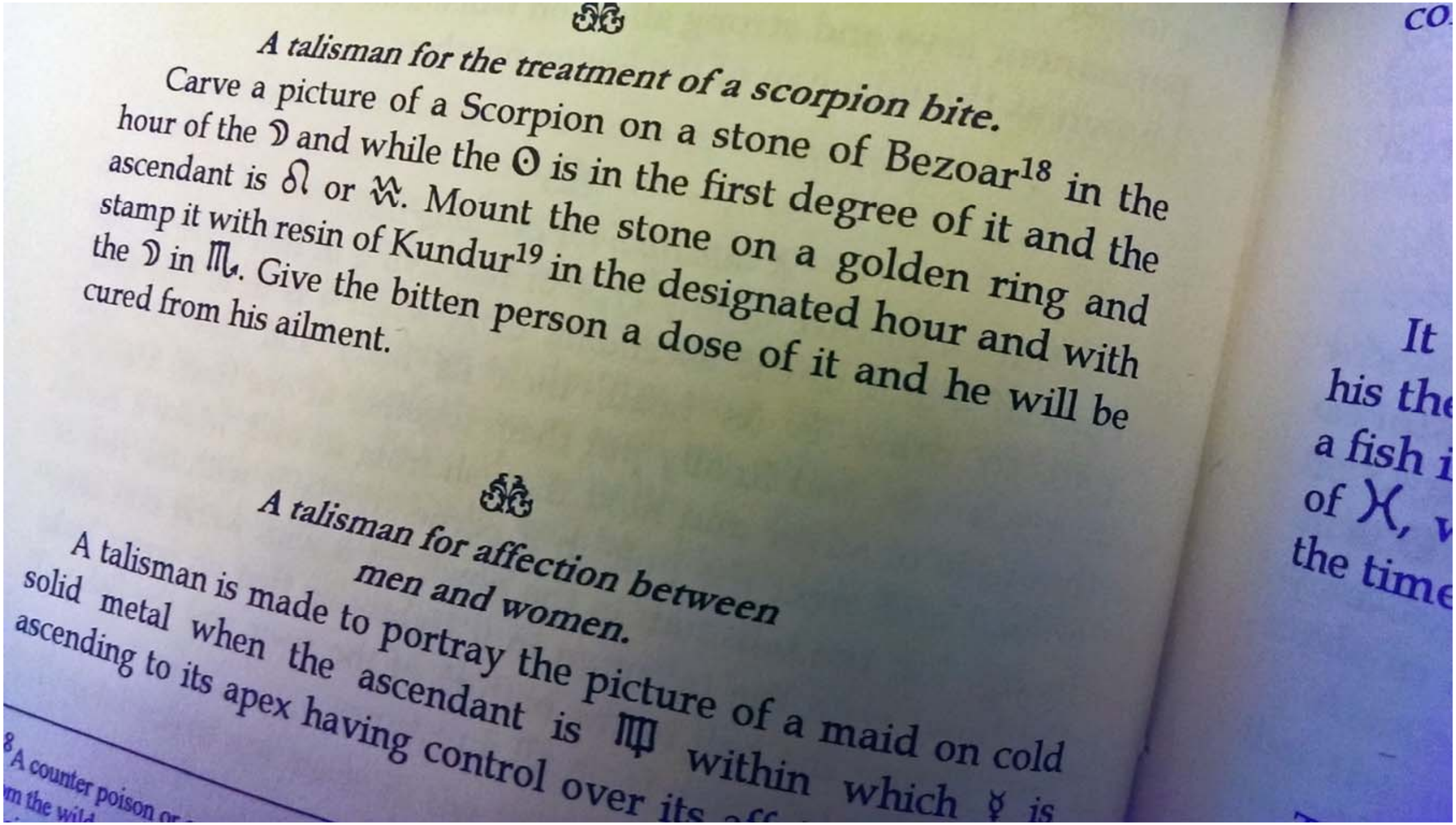


```
    actions = load '/user/hive/warehouse/actions/d=${processingDate}' using PigStorage('\u0001') as (udid:chararray, publisher_u
    actions_ltd = foreach actions generate udid, udf.removeNullCountry(country) as country, udf.parseDay(time) as day, publisher
    actions_ltd_filtered = FILTER actions_ltd BY day == '${processingDate}';

    actions_join_day = JOIN actions_ltd_filtered BY day, day_dimension_ltd BY calendar_date USING 'replicated';
    actions_join_day = foreach actions_join_day generate udid, country, day_key, app_id, source, paid_conversion;

    actions_join_app = JOIN actions_join_day BY app_id, app_dimension_ltd BY app_id USING 'replicated';
    actions_join_app = foreach actions_join_app generate udid, country, day_key, app_key, platform_key, source, paid_conversion;

    actions_join_country_1 = JOIN actions_join_app BY country LEFT, country_1 BY obsolete_country_code USING 'replicated';
    actions_join_country_2 = JOIN actions_join_country_1 BY country LEFT, country_dimension_ltd BY country_iso3661_code USING 'r

    converter = foreach actions_join_country_2 generate source as source, udid as udid, day_key as day_key, app_key as app_key,


'''.replace('${processingDate}', processingDate)

for i in range(1, len(dimensions)):
    pig += ('base_' + dimensions[i][0] + ' = foreach converter generate ' + dimensions[i][1] + ';\n')
    pig += (dimensions[i][0] + ' = DISTINCT base_' + dimensions[i][0] + ';\n')
    pig += (dimensions[i][0] + ' = foreach ' + dimensions[i][0] + ' generate ' + dimensions[i][2].replace('as ', 'as base_') + '

pig += ('\n')

for i in range(1, len(segments)):
    pig += (segments[i][0] + ' = ' + segments[i][1] + ';\n')

pig += ('\n\n')

for i in range(len(segments)):
    for j in range(len(dimensions)):
        pig += (segments[i][0] + '_' + dimensions[j][0] + ' = foreach ' + segments[i][0] + ' generate udid, ' + dimensions[j][1]
```

Magic?

- Unlimited query flexibility
- Uncanny accuracy
- Simpler table structure
- Simpler ETL
  - No need to pre-compute outside
  - Easier to maintain code
- Faster ETL

Yes, magic!



A talisman for the treatment of a scorpion bite.

Carve a picture of a Scorpion on a stone of Bezoar[18] in the hour of the ☽ and while the ☉ is in the first degree of it and the ascendant is ♌ or ♒. Mount the stone on a golden ring and stamp it with resin of Kundur[19] in the designated hour and with the ☽ in ♏. Give the bitten person a dose of it and he will be cured from his ailment.

A talisman for affection between men and women.

A talisman is made to portray the picture of a maid on cold solid metal when the ascendant is ♍ within which ☿ is ascending to its apex having control over its aff

[18] A counter poison or

m the wild

co

It

his the

a fish i

of ♓, v

the time

Vertica Community Edition

- https://my.vertica.com/community/
  - Free
  - Up to 1TB
  - No time limit.

- Documentation
  - http://my.vertica.com/docs/7.0.x/HTML/index.htm

David.Abercrombie@tapjoy.com