Redshift is SUPER

A More Flexible Cloud Data Warehouse Design

SPECIAL THANKS TO ALL OUR AWESOME CAMP SPONSORS!

TruStage™

HEARTLAND BUSINESS SYSTEMS

Intellibus

aws

Unspecified SOFTWARE CO

Architect.io

ionic

CAMUNDA

alexa

Google Cloud

CORE BTS

GrapeCity®

Join Us Next Year!

WI 24

THAT CONFERENCE

JULY 29TH - AUG. 1ST

My Story

Nick Heidke, Principal Data Architect at SimpleRose

Live in Appleton, WI

Bachelor's in Comp Sci from University of Wisconsin-Eau Claire 2009

Alright, so, catchy title Nick, lots of buzzwords, well done, but what do I actually need to know so this makes any sense?

Glad you asked. Couple of key points:

- The keyword in cloud data warehouse is warehouse meaning we're working with OLAP instead of OLTP

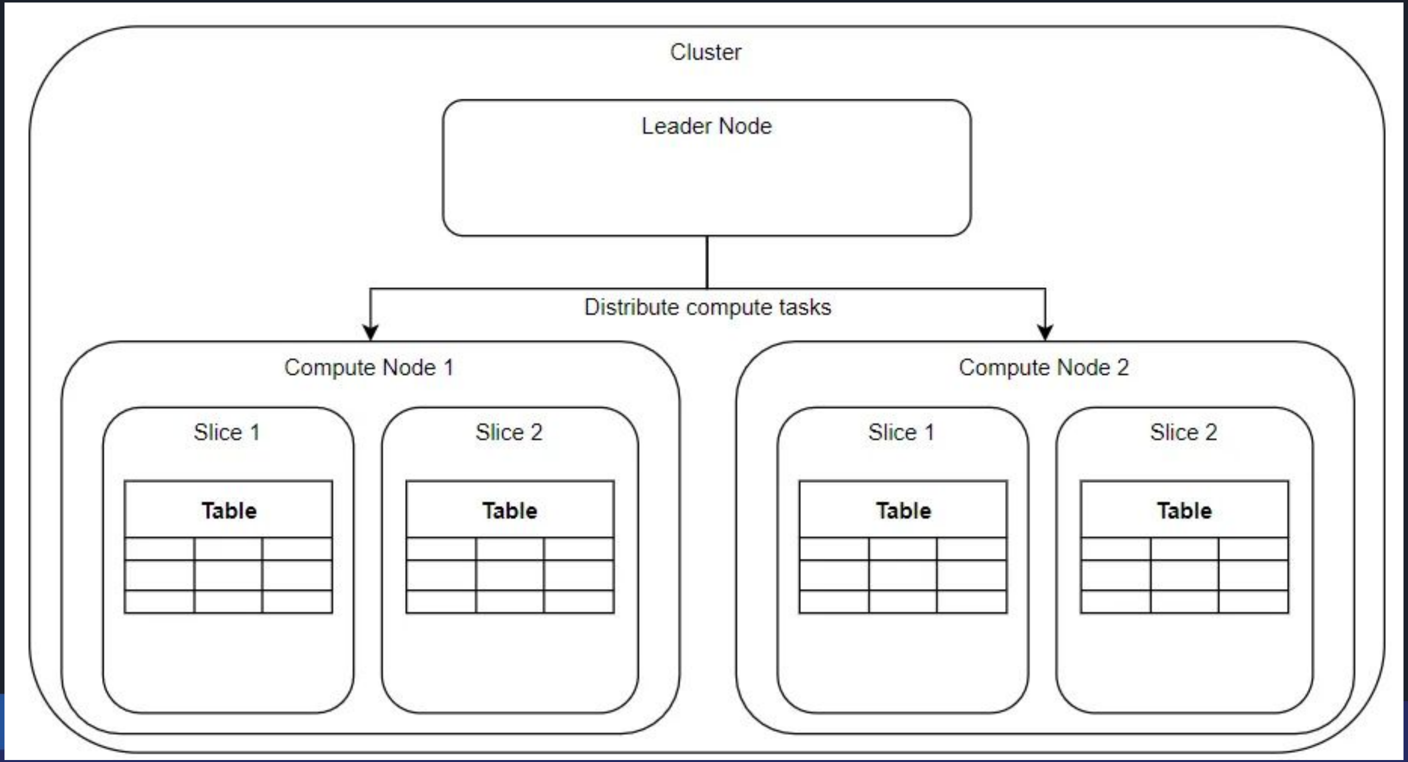
- Our specific technology, and several like it, is also capable of Massive Parallel Processing (MPP)

- While MPP databases are incredibly fast due to the amount of compute power available to them, our design must be cognizant of how the data is laid out

Key Features of MPP

- Parallel Processing - Data and queries are divided across nodes, each independently performing a chunk of the work
- Massive scale - On the order of petabytes
- Shared nothing architecture
- Horizontal instead of vertical scaling
- Distribution and sort keys - no traditional indexes

MPP Architecture



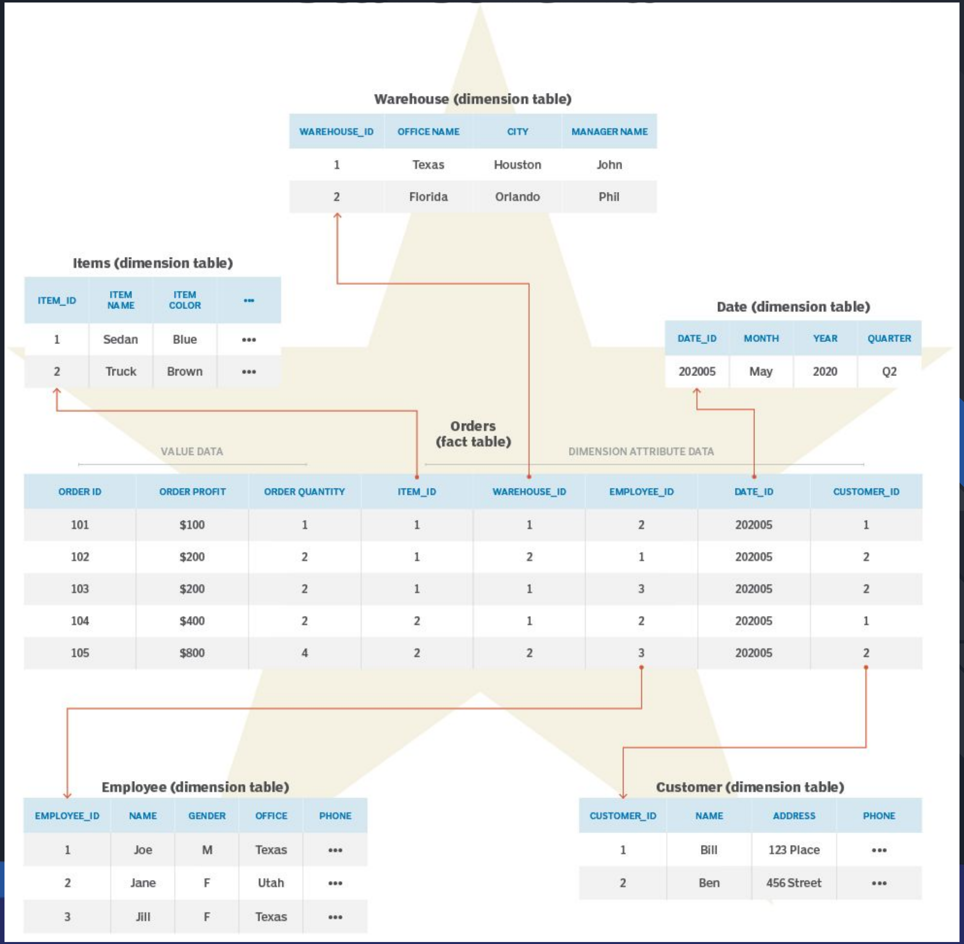
https://towardsdatascience.com/amazon-redshift-architecture-b674513eb996

Schema Design with MPP In Mind

- Typically, a single table's data is distributed across all of the nodes
- Data on each node is sorted by one key (though it can include several columns)
- If required data is not on a given node, it must be copied from another
- Where does that lead us?

To the stars...

Specifically, the star schema



Warehouse (dimension table)

WAREHOUSE_ID	OFFICE NAME	CITY	MANAGER NAME
1	Texas	Houston	John
2	Florida	Orlando	Phil

Items (dimension table)

ITEM_ID	ITEM NAME	ITEM COLOR	•••
1	Sedan	Blue	•••
2	Truck	Brown	•••

Date (dimension table)

DATE_ID	MONTH	YEAR	QUARTER
202005	May	2020	Q2

Orders (fact table)

VALUE DATA — DIMENSION ATTRIBUTE DATA

ORDER ID	ORDER PROFIT	ORDER QUANTITY	ITEM_ID	WAREHOUSE_ID	EMPLOYEE_ID	DATE_ID	CUSTOMER_ID
101	$100	1	1	1	2	202005	1
102	$200	2	1	2	1	202005	2
103	$200	2	1	1	3	202005	2
104	$400	2	2	1	2	202005	1
105	$800	4	2	2	3	202005	2

Employee (dimension table)

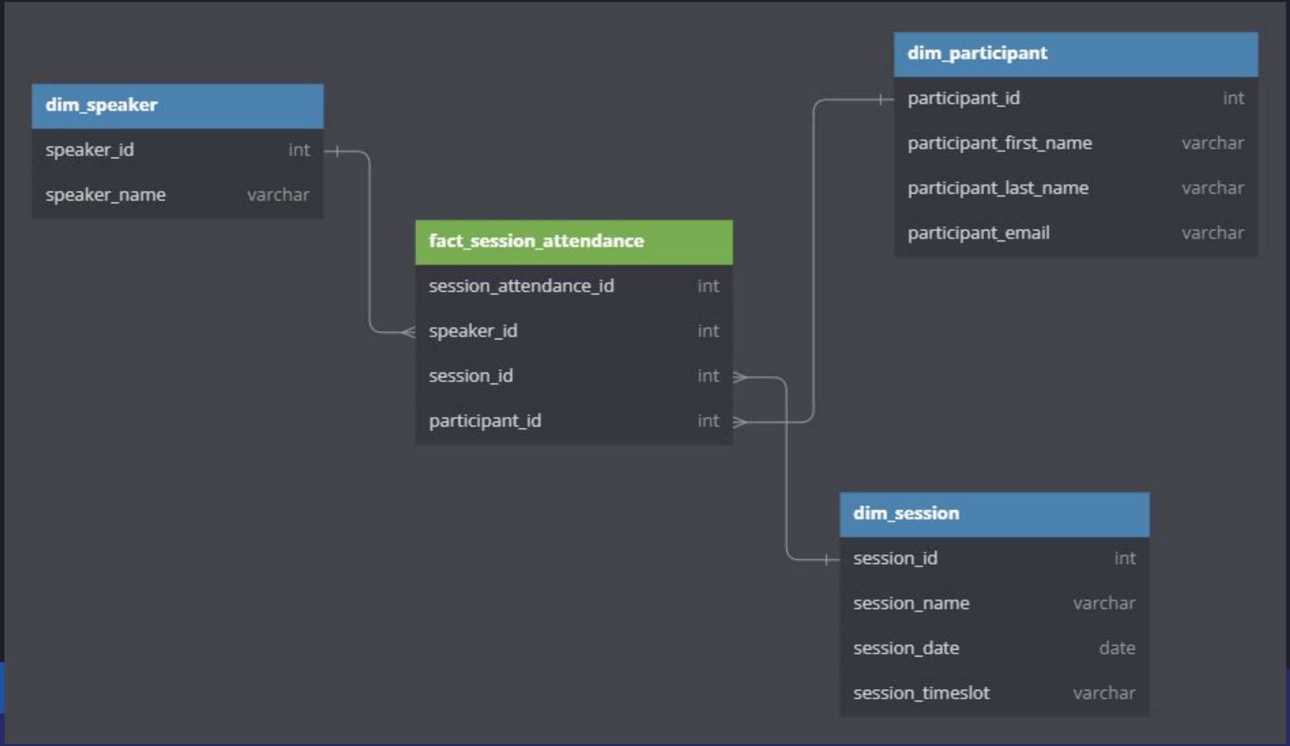
EMPLOYEE_ID	NAME	GENDER	OFFICE	PHONE
1	Joe	M	Texas	•••
2	Jane	F	Utah	•••
3	Jill	F	Texas	•••

Customer (dimension table)

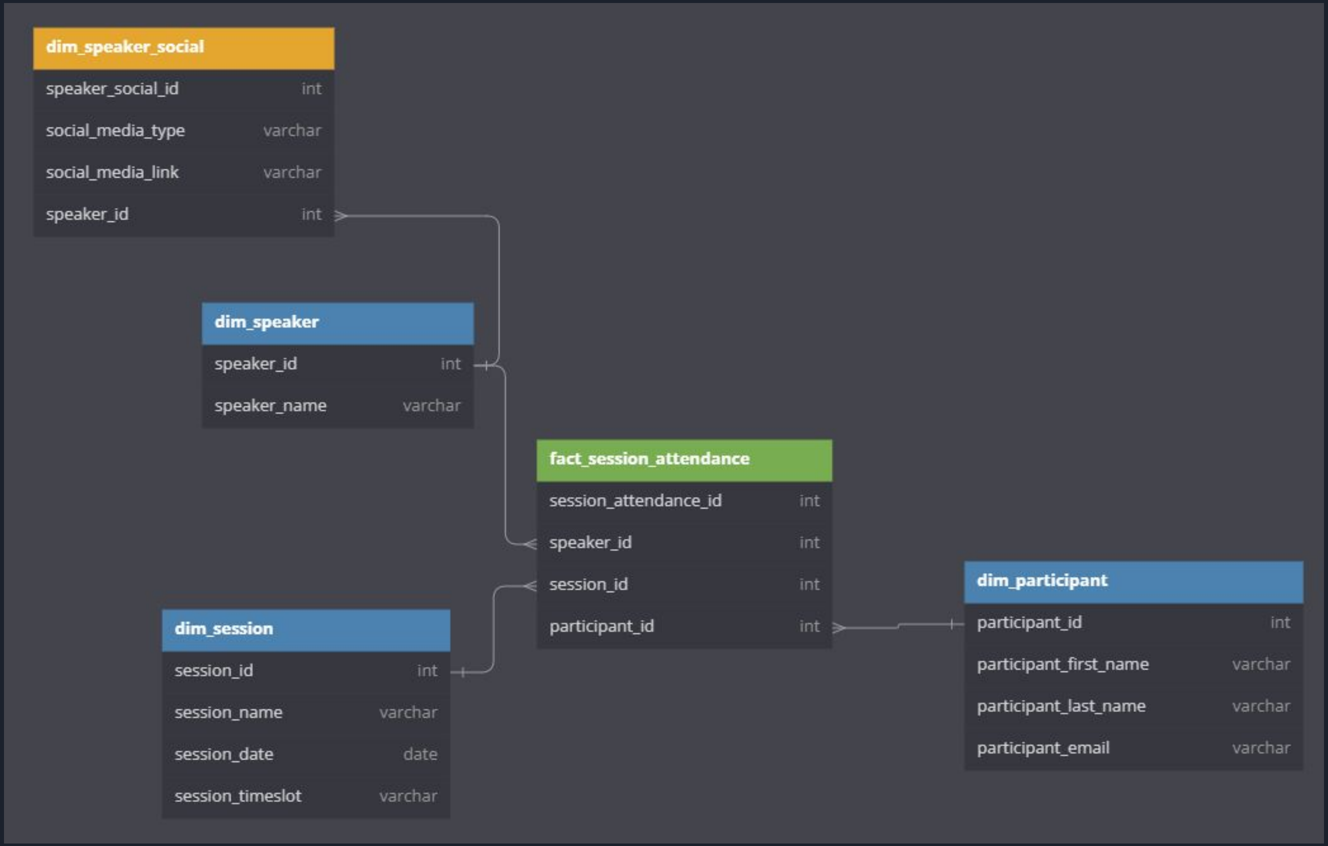
CUSTOMER_ID	NAME	ADDRESS	PHONE
1	Bill	123 Place	•••
2	Ben	456 Street	•••

Source: https://www.techtarget.com/searchdatamanagement/definition/star-schema

Our domain problem today



What happens when we need to add social media contacts for our speakers?



We might start out by adding an additional table linked to our existing speaker dimension

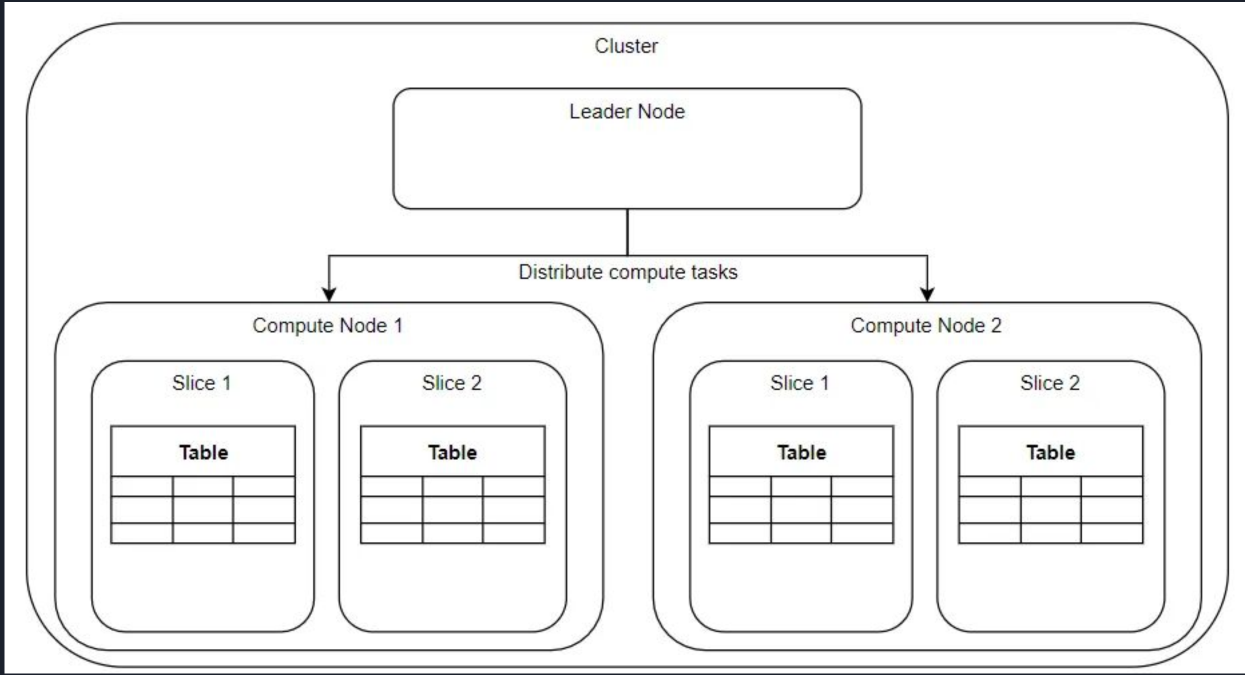
Our first query requirement comes in

```
SELECT ds.speaker_name, dss.social_media_link, dss.social_media_type, d.session_name
FROM conf_star_schema_v1.fact_session_attendance fsa
INNER JOIN conf_star_schema_v1.dim_speaker ds
ON fsa.speaker_id = ds.speaker_id
INNER JOIN conf_star_schema_v1.dim_speaker_social dss
ON dss.speaker_id = ds.speaker_id
INNER JOIN conf_star_schema_v1.dim_sessiond
ON fsa.session_id = d.session_id
WHERE speaker_name = 'John Smith';
```

What’s the potential issue?

Slices of data are each on independent nodes

Joining across the nodes is expensive



Query Cost - Network Ops

Network operation	Distribution Behavior	Relative Cost
DS_DIST_NONE DS_DIST_ALL_NONE	No redistribution required	Low
DS_DIST_INNER DS_DIST_OUTER DS_DIST_BOTH	One or more of the tables is redistributed	Medium/High
DS_BCAST_INNER	A copy of the entire inner table is broadcast to all the compute nodes	High

How does our query plan look currently?

```
QUERY PLAN
XN Hash Join DS_BCAST_INNER  (cost=640000.60..1240000.87 rows=2 width=89)
  Hash Cond: ("outer".speaker_id = "inner".speaker_id)
  ->  XN Seq Scan on dim_speaker_social dss  (cost=0.00..0.11 rows=11 width=51)
  ->  XN Hash  (cost=640000.60..640000.60 rows=1 width=50)
        ->  XN Hash Join DS_BCAST_INNER  (cost=280000.36..640000.60 rows=1 width=50)
              Hash Cond: ("outer".session_id = "inner".session_id)
              ->  XN Seq Scan on dim_session d  (cost=0.00..0.10 rows=10 width=31)
              ->  XN Hash  (cost=280000.36..280000.36 rows=1 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=0.13..280000.36 rows=1 width=27)
                          Hash Cond: ("outer".speaker_id = "inner".speaker_id)
                          ->  XN Seq Scan on fact_session_attendance fsa  (cost=0.00..0.10 rows=10 width=8)
                          ->  XN Hash  (cost=0.12..0.12 rows=1 width=19)
                                ->  XN Seq Scan on dim_speaker ds  (cost=0.00..0.12 rows=1 width=19)
                                      Filter: ((speaker_name)::text = 'John Smith'::text)
```

Option #1 - Distribution Styles

Distribution Style	Description
EVEN	Data is distributed across all slices evenly
KEY	Specified key is used to distribute data across slices; rows with the same distribution key are co-located within the same slice
ALL	Entire table is replicated to every slice in the cluster
AUTO	Allows Redshift to automatically determine optimal distribution based on table size and heuristics

https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_dist_sort.html

What distribution style is best for our case?

Table Design - Version 2

```
create table dim_speaker
(
    speaker_id    integer distkey,
    speaker_name  varchar(256)
) DISTSTYLE KEY;
```

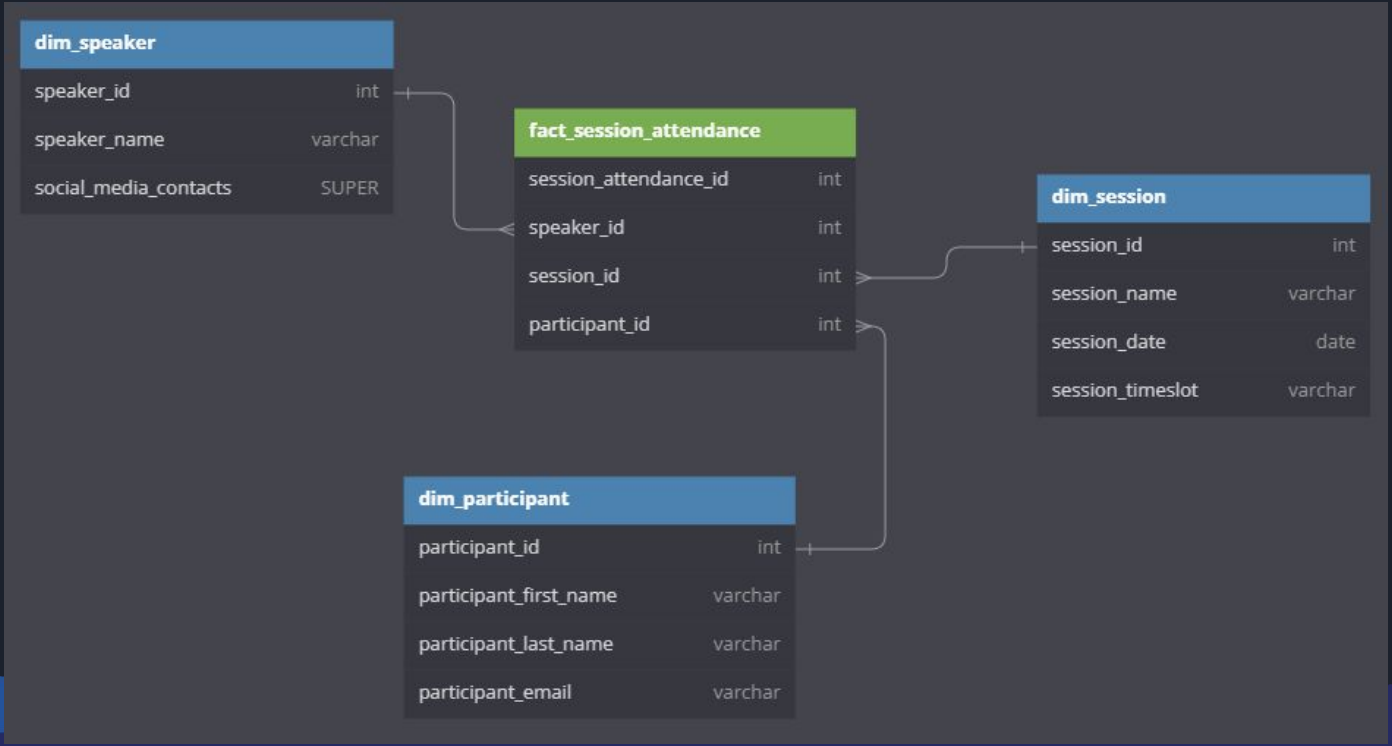
```
create table dim_speaker_social
(
    speaker_social_id integer,
    social_media_type varchar(256),
    social_media_link varchar(256),
    speaker_id        integer distkey
) DISTSTYLE KEY;
```

Query Plan - Version 2

```
QUERY PLAN
XN Hash Join DS_DIST_ALL_NONE  (cost=0.38..0.74 rows=2 width=89)
  Hash Cond: ("outer".session_id = "inner".session_id)
  ->  XN Hash Join DS_DIST_NONE  (cost=0.25..0.57 rows=2 width=66)
        Hash Cond: ("outer".speaker_id = "inner".speaker_id)
        ->  XN Hash Join DS_DIST_NONE  (cost=0.13..0.40 rows=2 width=70)
              Hash Cond: ("outer".speaker_id = "inner".speaker_id)
              ->  XN Seq Scan on dim_speaker_social dss  (cost=0.00..0.11 rows=11 width=51)
              ->  XN Hash  (cost=0.12..0.12 rows=1 width=19)
                    ->  XN Seq Scan on dim_speaker ds  (cost=0.00..0.12 rows=1 width=19)
                          Filter: ((speaker_name)::text = 'John Smith'::text)
        ->  XN Hash  (cost=0.10..0.10 rows=10 width=8)
              ->  XN Seq Scan on fact_session_attendance fsa  (cost=0.00..0.10 rows=10 width=8)
  ->  XN Hash  (cost=0.10..0.10 rows=10 width=31)
        ->  XN Seq Scan on dim_session d  (cost=0.00..0.10 rows=10 width=31)
```

What if we want to avoid a second table altogether?

Option #2 - Our SUPER-hero



What does this look like in practice?

speaker_id	speaker_name	social_media_contacts
1	John Smith	[{"platform":"twitter","username_url":"@johnsmith"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/johnsmith"}]
2	Jane Doe	[{"platform":"twitter","username_url":"@janedoe"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/janedoe"}]
3	Michael Johnson	[{"platform":"twitter","username_url":"@michaeljohnson"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/michaeljohnson"}]
4	Emily Brown	[{"platform":"twitter","username_url":"@emilybrown"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/emilybrown"}]
5	David Davis	[{"platform":"twitter","username_url":"@daviddavis"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/daviddavis"}]
6	Laura Miller	[{"platform":"twitter","username_url":"@lauramiller"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/lauramiller"}]
7	Andrew Wilson	[{"platform":"twitter","username_url":"@andrewwilson"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/andrewwilson"}]
8	Olivia Lee	[{"platform":"twitter","username_url":"@olivialeee"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/olivialeee"}]
9	Daniel Clark	[{"platform":"twitter","username_url":"@danielclark"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/danielclark"}]
10	Sophia Wang	[{"platform":"twitter","username_url":"@sophiawang"},{"platform":"linkedin","username_url":"https://www.linkedin.com/in/sophiawang"}]

Let’s pretty-print-ify this

```
[
    {
        "platform": "twitter",
        "username_url": "@johnsmith"
    },

    {
        "platform": "linkedin",
        "username_url": " https://www.linkedin.com/in/johnsmith "
    }
]
```

Our mission to alleviate joins is complete, but our users are going to HATE it.

Also, what about query performance?

Final Step - Materialized View

For ease of querying and consuming by other systems, a materialized view can be put over the data to simulate the data being stored in separate columns

Here’s our goal

speaker_id	speaker_name	platform	username_url
1	John Smith	twitter	@johnsmith
1	John Smith	linkedin	https://www.linkedin.com/in/johnsmith
2	Jane Doe	twitter	@janedoe
2	Jane Doe	linkedin	https://www.linkedin.com/in/janedoe
3	Michael Johnson	twitter	@michaeljohnson
3	Michael Johnson	linkedin	https://www.linkedin.com/in/michaeljohnson
4	Emily Brown	twitter	@emilybrown
4	Emily Brown	linkedin	https://www.linkedin.com/in/emilybrown
5	David Davis	twitter	@daviddavis
5	David Davis	linkedin	https://www.linkedin.com/in/daviddavis
6	Laura Miller	twitter	@lauramiller
6	Laura Miller	linkedin	https://www.linkedin.com/in/lauramiller
7	Andrew Wilson	twitter	@andrewwilson
7	Andrew Wilson	linkedin	https://www.linkedin.com/in/andrewwilson
8	Olivia Lee	twitter	@olivialeee
8	Olivia Lee	linkedin	https://www.linkedin.com/in/olivialeee
9	Daniel Clark	twitter	@danielclark
9	Daniel Clark	linkedin	https://www.linkedin.com/in/danielclark
10	Sophia Wang	twitter	@sophiawang
10	Sophia Wang	linkedin	https://www.linkedin.com/in/sophiawang

Our materialized view code

```
CREATE MATERIALIZED VIEW vw_dim_speaker_social_contacts
    DISTSTYLE KEY
    DISTKEY (speaker_id)
    AS
SELECT speaker_id,
       speaker_name,
       REPLACE(socials.platform::varchar, '"', '')       platform,
       REPLACE(socials.username_url::varchar, '"', '') username_url
FROM conf_star_schema_v3.dim_speaker ds,
     ds.social_media_contacts socials;
```

What if we want the data in columns?

speaker_id	speaker_name	twitter	linkedin	threads
1	John Smith	@johnsmith	https://www.linkedin.com/in/johnsmith	<null>
2	Jane Doe	@janedoe	https://www.linkedin.com/in/janedoe	<null>
3	Michael Johnson	@michaeljohnson	https://www.linkedin.com/in/michaeljohnson	<null>
4	Emily Brown	@emilybrown	https://www.linkedin.com/in/emilybrown	<null>
5	David Davis	@daviddavis	https://www.linkedin.com/in/daviddavis	<null>
6	Laura Miller	@lauramiller	https://www.linkedin.com/in/lauramiller	<null>
7	Andrew Wilson	@andrewwilson	https://www.linkedin.com/in/andrewwilson	<null>
8	Olivia Lee	@olivialeee	https://www.linkedin.com/in/olivialeee	<null>
9	Daniel Clark	@danielclark	https://www.linkedin.com/in/danielclark	<null>
10	Sophia Wang	@sophiawang	https://www.linkedin.com/in/sophiawang	<null>

Our final materialized view code

```
CREATE MATERIALIZED VIEW vw_dim_speaker_social_contacts_pivoted
   DISTSTYLE KEY
   DISTKEY (speaker_id)
   AS
SELECT *
FROM (SELECT speaker_id,
            speaker_name,
            REPLACE(socials.platform::varchar, '"', '')::varchar     platform,
            REPLACE(socials.username_url::varchar, '"', '')::varchar username_url
     FROM conf_star_schema_v3.dim_speaker ds,
          ds.social_media_contacts socials) PIVOT (
           max(username_url) FOR platform IN ('twitter', 'linkedin', 'threads')
);
```

Final Takeaways

For distributed, massively parallel databases, JOIN statements can be very costly, especially if data is not distributed correctly

Using the correct distribution style and key will improve performance, but imposes limitations on how we query and design the rest of our model

To minimize joining entirely we can store sub-dimension data into JSON fields instead of their own tables

Materialized views can be put over the top to retain performance and increase usability

Thank you!

@nickheidke on ?Twitter?

Presentation materials available on: github.com/nickheidke

Session Feedback

https://that.land/3NOVbYe

Resources

[Optimizing for Star Schemas on Redshift]

[Distribution and Sort Key Best Practices]

[PartiQL for Querying Semistructured Data]

[Using Materialized Views with the SUPER Datatype]