Module 11

Schema Design and Nested Data Structures

In this module we will:

- **Compare Google BigQuery vs Traditional Relational Data Architecture**
- Normalization vs Denormalization: Performance Tradeoffs
- Working with Nested Data, Arrays, and Structs in Google BigQuery

Google Cloud

Let's Re-Examine our IRS Schema as an Architect

Form 990 (2016)

Part IX Statement of Functional Expenses

Section 501(c)(3) and 501(c)(4) organizations must complete all columns. Al

Check if Schedule O contains a response or note to any lin

Do not include amounts reported on lines 6b, 7b, 8b, 9b, and 10b of Part VIII.		(A) Total expenses
1	Grants and other assistance to domestic organizations and domestic governments. See Part IV, line 21	
2	Grants and other assistance to domestic individuals. See Part IV, line 22	
3	Grants and other assistance to foreign organizations, foreign governments, and foreign individuals. See Part IV, lines 15 and 16	
4	Benefits paid to or for members	
5	Compensation of current officers, directors, trustees, and key employees	
6	Compensation not included above, to disqualified persons (as defined under section 4958(f)(1)) and persons described in section 4958(c)(3)(B)	
7	Other salaries and wages	
8	Pension plan accruals and contributions (include section 401(k) and 403(b) employer contributions)	
9	Other employee benefits	
10	Payroll taxes	
11	Fees for services (non-employees):	
a	Management	
b	Legal	
c	Accounting	
d	Lobbying	
e	Professional fundraising services. See Part IV, line 17	
f	Investment management fees	
g	Other. (If line 11g amount exceeds 10% of line 25, column (A) amount, list line 11g expenses on Schedule O.)	
12	Advertising and promotion	
13	Office expenses	
14	Information technology	
15	Royalties	
16	Occupancy	
17	Travel	
18	Payments of travel or entertainment expenses	

Each of these data fields needs to be stored in a structured way

Google Cloud

Option 1: Add each Expense field as a New Column

Table Details: irs_990_2015

Schema | Details | Preview

yeebenef	payrolltx	feesforsrvcmgmt	legalfees	accntingfees	feesforsrvclobby	profndraising	feesforsrvcinvstmgmt	feesforsrvcothr	advrtpromo	officexpns	infotech	royaltsexpns	occupancy	travel
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
816623	847695	0	0	28654	0	0	0	27770	0	155715	43796	0	1156758	0
524396	539651	127071	34165	44264	0	0	0	0	567392	732920	875416	0	887599	33446
177305	209707	0	120	22000	0	0	0	11551	0	165306	11391	0	231092	0
1289799	543608	7415	14888	33514	0	0	0	0	1273856	2101383	217216	0	604569	40173
512540	170264	0	24000	64500	0	0	0	96660	344208	2128823	0	0	540746	0
217097	115324	0	0	0	0	0	0	0	0	0	0	0	0	0

Option 1: Add each Expense field as a New Column

Table Details: irs_990_2015

Schema | Details | Preview

yeebenef	payrolltx	feesforsrvcmgmt	legalfees	accntingfees	feesforsrvclobby	profndraising	feesforsrvcinvstmgmt	feesforsrvcothr	advrtpromo	officexpns	infotech	royaltsexpns	occupancy	travel
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
816623	847695	0	0	28654	0	0	0	27770	0	155715	43796	0	1156758	0
524396	539651	127071	34165	44264	0	0	0	0	567392	732920	875416	0	887599	33446
177305	209707	0	120	22000	0	0	0	11551	0	165306	11391	0	231092	0
1289799	543608	7415	14888	33514	0	0	0	0	1273856	2101383	217216	0	604569	40173
512540	170264	0	24000	64500	0	0	0	96660	344208	2128823	0	0	540746	0
217097	115324	0	0	0	0	0	0	0	0	0	0	0	0	0

… results in a really WIDE table that is **not scalable**...

Google Cloud

Option 2: Break Out Expenses into another Lookup Table

Organization Details

Company ID	Company Name
161218560	NY Association Inc.

Historical Transactions

Company ID	Expense Code	Amount
161218560	1	$10,000

Code Lookup Tables

Expense Code	Expense Type
1	Lobbying
2	Legal
3	Insurance

Option 2: Break Out Expenses into another Lookup Table

Organization Details

Company ID	Company Name
161218560	NY Association Inc.

Historical Transactions

Company ID	Expense Code	Amount
161218560	1	$10,000

Code Lookup Tables

Expense Code	Expense Type
1	Lobbying
2	Legal
3	Insurance

… this breaking apart process is called **Normalization** ...

Module 11

Schema Design and Nested Data Structures

In this module we will:

- Compare Google BigQuery vs Traditional Relational Data Architecture
- **Normalization vs Denormalization: Performance Tradeoffs**
- Working with Nested Data, Arrays, and Structs in Google BigQuery

Google Cloud

Normalization Benefit: Scalable Individual Tables

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...

Historical Transactions

Company ID	Expense Code	Amount
161218560	1	$10,000
...	...	...

Code Lookup Tables

Expense Code	Expense Type
1	Lobbying
2	Legal
3	Insurance
...	...

... schema changes no longer needed as data grows ...

Google Cloud

Normalization Drawback: JOINs are now a Necessity

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...

Historical Transactions

Company ID	Expense Code	Amount
161218560	1	$10,000
...	...	...

Code Lookup Tables

Expense Code	Expense Type
1	Lobbying
2	Legal
3	Insurance
...	...

```
SELECT Company Name, Amount, Expense Type
```

NY Association Inc.	$10,000	Lobbying

Did we go too far? **Denormalization** Improves Performance

Organization Details

Company ID	**Company Name**
161218560	NY Association Inc.
...	...

Historical Transactions

Company ID	**Expense Code**	**Amount**
161218560	Lobbying	$10,000
...	...	...

Code Lookup Tables

Expense Code	**Expense Type**
1	Lobbying
2	Legal
3	Insurance
...	...

```
SELECT Company Name, Amount, Expense Type
```

NY Association Inc.	$10,000	Lobbying

Relational Databases at Scale?

How do traditional relational databases handle record growth at scale?

Traditionally, Very Large Tables are **Hard to Scan and Compute**

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

```
SELECT Company Name ORDER BY Company Name
```

Traditional: Pre-Sorted **Indexes** Introduced to Help Common Queries

Organization Details

Company ID	Company Name
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

Index

Company Name	Ranked Order
ACME Inc.	1
...	...
...	
NY Association Inc.	**900,000**
...	

Indexes do not exist in BigQuery because data is stored and handled in a fundamentally different way as you will see next...

```
SELECT Company Name ORDER BY Company Name
```


Google Cloud

BigQuery Architecture Introduces Three Key Innovations

1. **Column-Based** Data Storage
2. **Break Apart Tables** into Pieces
3. Store **Nested Fields** within a Table

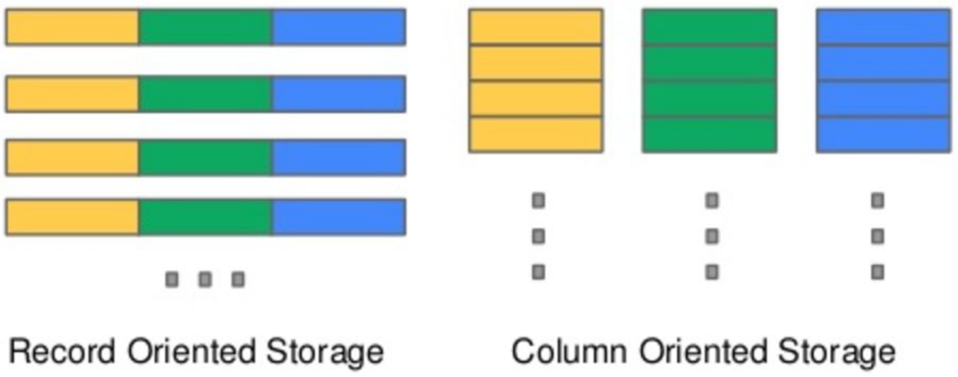
Google Cloud

BigQuery Architecture Introduces Three Key Innovations

1. **Column-Based** Data Storage
2. **Break Apart Tables** into Pieces
3. Store **Nested Fields** within a Table

Google Cloud

BigQuery **Column-Oriented Storage is Built for Speed**



- Storing related values (faster to loop through at execution time)
- Columns can be **individually** compressed
- Access values from a few columns without reading every one

BigQuery Architecture Introduces Three Key Innovations

1. **Column-Based** Data Storage
2. **Break Apart Tables** into Pieces
3. Store **Nested Fields** within a Table

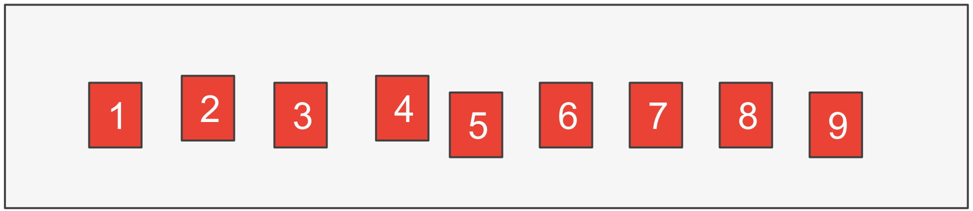
Google Cloud

BigQuery Automatically **Breaks Apart Data into Smaller Shards**

Organization Details

Company ID	**Company Name**
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

Google File System



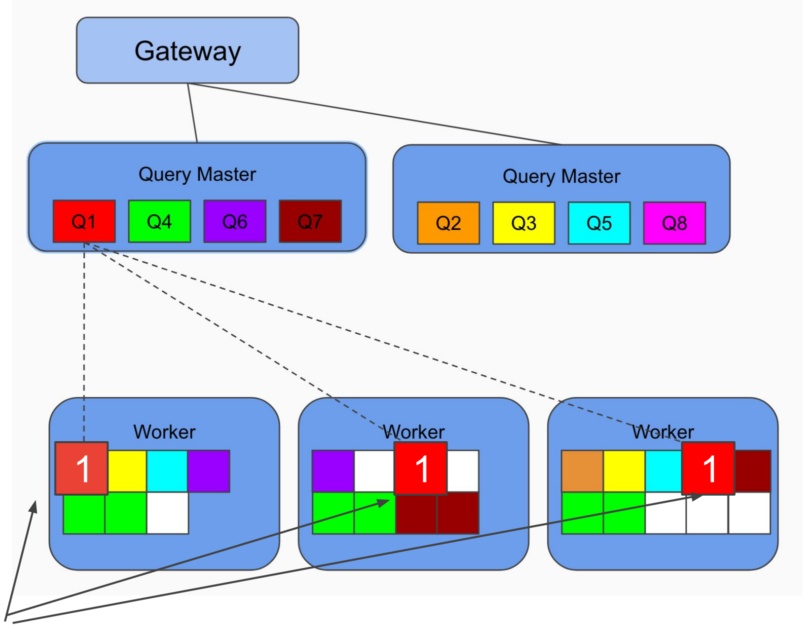
Google Cloud

BigQuery Automatically **Pieces it All Back Together for Queries**

Organization Details

Company ID	**Company Name**
161218560	NY Association Inc.
...	...
...	...
...	...
10 Billion Row Table	

```
SELECT Company Name ORDER BY Company Name
```



Shards of data are read and Processed in Parallel

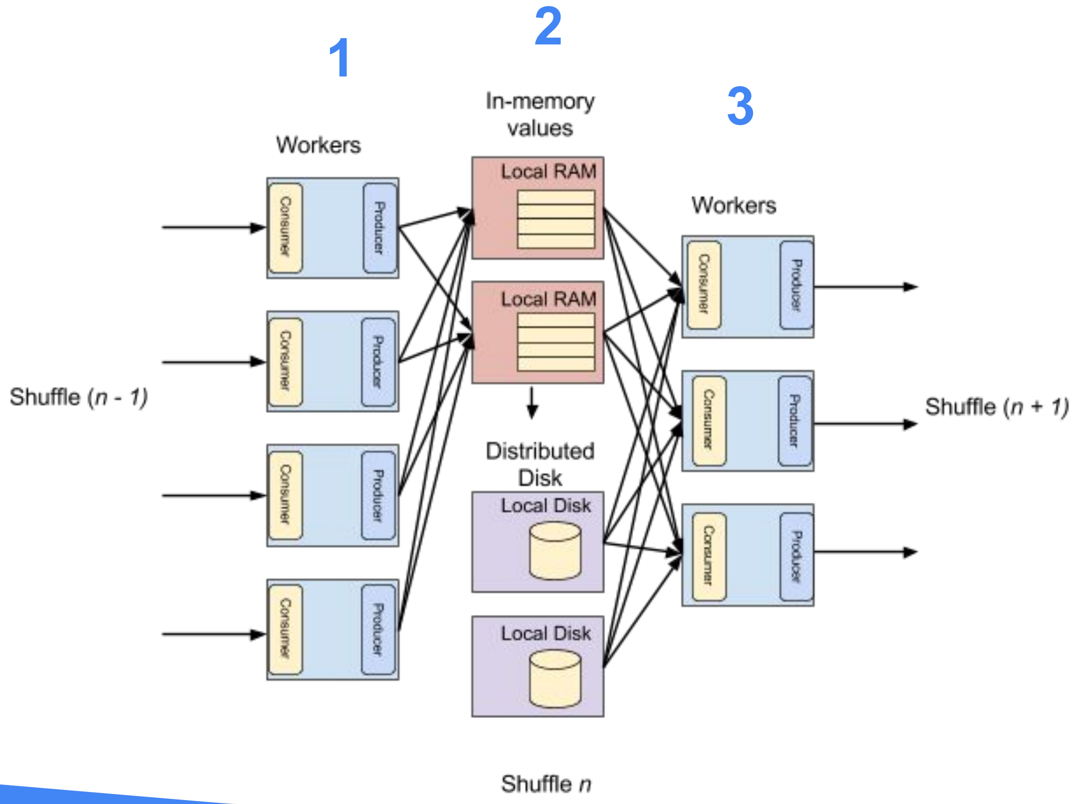
Google Cloud

BigQuery Automatically **Balances and Scales Workers**

- Up to 2,000 workers to process concurrent queries (on-demand tier)
- “Fairness model” for allocation

Google Cloud

BigQuery Workers **Communicate by Shuffling Data In-Memory**



1. Workers Consume data values and perform operations in parallel

2. Workers Produce output to the In-Memory Shuffle Service

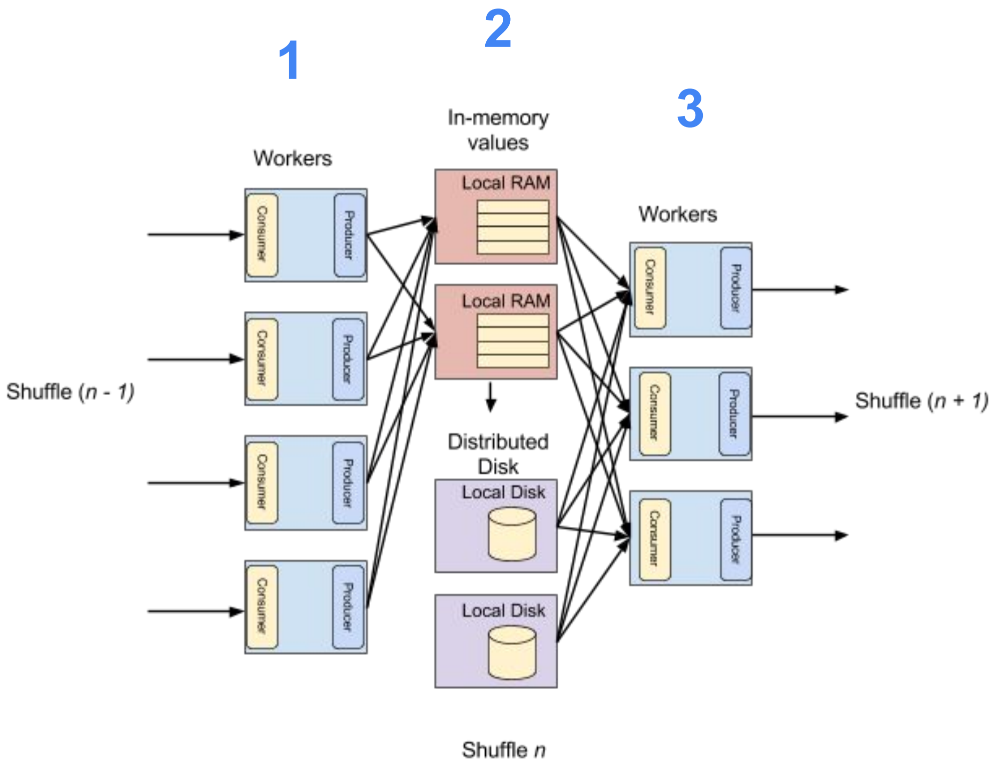
3. Workers Consume New Data and continue processing

Workers (one or more slots) scale to meet the demand of the processing task.

Read More

Google Cloud

BigQuery **Shuffling Enables Massive Scale**



- Shuffle allows BigQuery to **process massively parallel petabyte-scale data** jobs
- Everything after Query Execution is **Automatically Scaled and Managed**
- All Queries Large and Small Use Shuffle

Google Cloud

BigQuery Architecture Introduces Three Key Innovations

1. **Column-Based** Data Storage
2. **Break Apart Tables** into Pieces
3. Store **Nested Fields** within a Table

Google Cloud

Module 11

Schema Design and Nested Data Structures

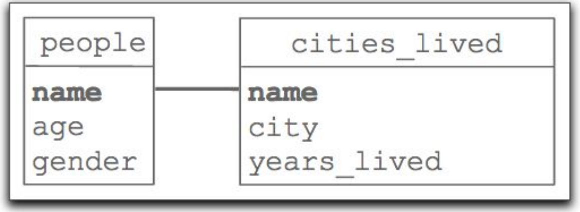
In this module we will:

- Compare Google BigQuery vs Traditional Relational Data Architecture
- Normalization vs Denormalization: Performance Tradeoffs
- **Working with Nested Data, Arrays, and Structs in Google BigQuery**

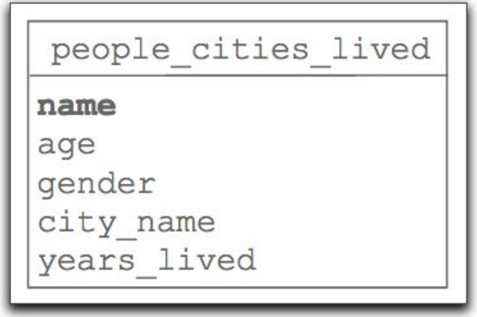
Google Cloud

BigQuery Architecture Introduces Repeated Fields

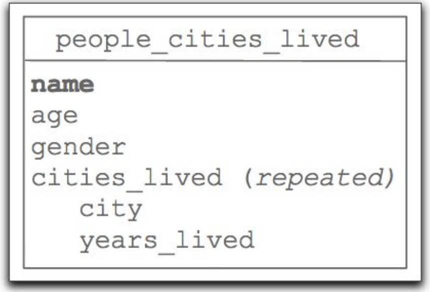
Normalized



Denormalized



Repeated



Less Performant → **High Performing**

Google Cloud

The Traditional Relational Model **Requires Expensive Joins**

Organization Details

Company ID	**Company Name**
161218560	NY Association Inc.
...	...

Historical Transactions

Company ID	**Expense Code**	**Amount**
161218560	1	$10,000
...	...	...

Code Lookup Tables

Expense Code	**Expense Type**
1	Lobbying
2	Legal
3	Insurance
...	...

BigQuery Can Use **Nested Schemas** For Highly Scalable Queries

Organization Details with Nested Historical Transactions

NESTED

Company ID	Company Name	Transactions.Amount	Code.Expense
161218560	NY Association Inc.	$10.000	Lobbying
		$5,000	Legal
		$1,000	Insurance
123435560	ACME Co.	$7,000	Travel

Google Cloud

Nested Schemas Bring **Performance Benefits**

Organization Details with Nested Historical Transactions

Company ID	Company Name	Transactions.Amount	Code.Expense
161218560	NY Association Inc.	$10.000	Lobbying
		$5,000	Legal
		$1,000	Insurance
123435560	ACME Co.	$7,000	Travel

- Avoid costly joins
- No performance punishment for `SELECT(DISTINCT Company ID)`

Google Cloud

Working with Repeated Fields

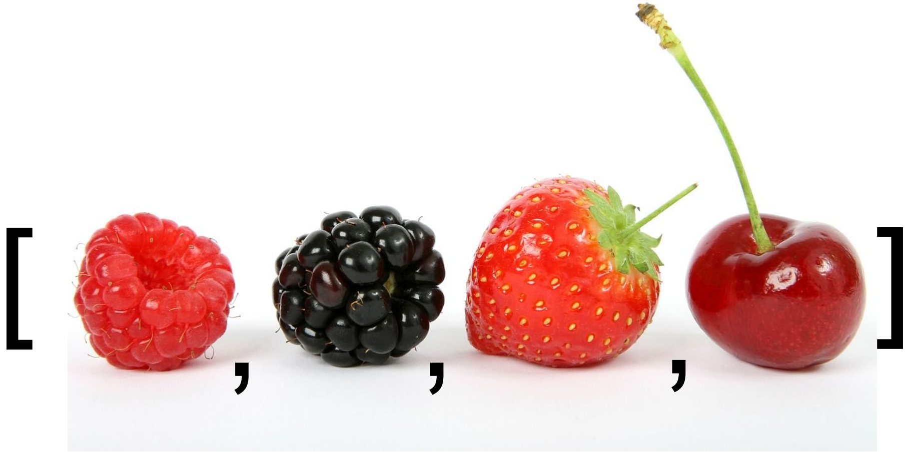
1. Introducing **Arrays and Structs**
2. **Flattening Arrays:** Legacy vs Standard
3. **Practicing SQL** with Repeated Fields

Working with Repeated Fields

1. Introducing **Arrays and Structs**
2. **Flattening Arrays:** Legacy vs Standard
3. **Practicing SQL** with Repeated Fields

Arrays are Supported Natively in BigQuery

Arrays are **ordered lists** of zero or more data values that must have the **same data type**



Google Cloud

Working with SQL Arrays

Create an array with brackets [ ]

```
SELECT
['raspberry', 'blackberry', 'strawberry', 'cherry']
AS fruit_array
```

BigQuery flattened output:

Row	fruit_array
1	raspberry
	blackberry
	strawberry
	cherry

Reminder: Use #standardSQL

Working with SQL Arrays

```
WITH fruits AS (
SELECT ['raspberry', 'blackberry', 'strawberry', 'cherry']
AS fruit_array
)

SELECT ARRAY_LENGTH(fruit_array) AS array_size
FROM fruits;
```

Row	array_size
1	4

Count the elements in an array
with ARRAY_LENGTH

BigQuery Implicitly Flattens Arrays

```
SELECT
  ['apple', 'pear', 'plum'] AS item,
  'Jacob' AS customer
```

Array = ['apple', 'pear', 'plum']

Flattened Array =
apple
pear
plum

BigQuery output:

- item → Flattened array
- Customer → Normal field

Row	item	customer
1	apple	Jacob
	pear	
	plum	

Explicitly Flatten Arrays with UNNEST()

```
SELECT
items,
customer_name
FROM
UNNEST(['apple', 'pear', 'peach']) AS items
CROSS JOIN
(SELECT 'Jacob' AS customer_name)
```

Associate all items in our array with the Customer

Flatten using a CROSS JOIN

BigQuery UNNESTED output:

Row	items	customer_name
1	apple	Jacob
2	pear	Jacob
3	peach	Jacob

UNNEST = A query that flattens an array and returns a row for each element in the array.

Google Cloud

Aggregate into an Array with ARRAY_AGG

```
WITH fruits AS
  (SELECT "apple" AS fruit
   UNION ALL
   SELECT "pear" AS fruit
   UNION ALL
   SELECT "banana" AS fruit)
```

Row	fruit
1	apple
2	pear
3	banana

← Subquery to create a table of fruits for us to aggregate later into an array

```
SELECT ARRAY_AGG(fruit) AS
fruit_basket
FROM fruits;
```

Row	fruit_basket
1	apple
	pear
	banana

Use ARRAY_AGG to aggregate values into an array

← These results are the same as saying: ["apple","pear","banana"]

Google Cloud

Sort Array Output with ORDER BY

```
WITH fruits AS
  (SELECT "apple" AS fruit
   UNION ALL
   SELECT "pear" AS fruit
   UNION ALL
   SELECT "banana" AS fruit)

SELECT ARRAY_AGG(fruit ORDER BY fruit)
AS fruit_basket
FROM fruits;
```

Row	fruit_basket
1	apple
	banana
	pear

← Notice how banana is now second

Filter Arrays using WHERE IN

```
WITH groceries AS
  (SELECT ['apple', 'pear', 'banana'] AS list
   UNION ALL
   SELECT ['carrot', 'apple'] AS list
   UNION ALL
   SELECT ['water', 'wine'] AS list)
```

Row	items
1	apple
	pear
	banana
2	carrot
	apple
3	water
	wine

← Start with a three arrays of shopping lists

```
SELECT
   ARRAY(
         SELECT items FROM UNNEST(list) AS items
         WHERE 'apple' IN UNNEST(list)
        ) AS contains_apple
FROM groceries;
```

Row	contains_apple
1	apple
	pear
	banana
2	carrot
	apple
3	

Use WHERE IN to filter an array. Note the empty third array returned back because 'apple' is not present in the original list

STRUCTs are Flexible Containers

STRUCT are a container of ordered fields each with a type (required) and field name (optional).

You can store multiple data types in a STRUCT (even Arrays!)

Google Cloud

STRUCTs are Flexible Containers

```
#standardSQL
SELECT
STRUCT(35 AS age, 'Jacob' AS name)
```

Store age as an integer
Store name as a string

wait, what's wrong with the below result?

Row	f0_.age	f0_.name
1	35	Jacob

STRUCTs are Flexible Containers

```
#standardSQL
SELECT
STRUCT(35 AS age, 'Jacob' AS name) AS customers
```

Name the overall STRUCT container

Row	customers.age	customers.name
1	35	Jacob

One STRUCT can have many values. Looks and behaves similar to a table!

Google Cloud

STRUCTs Can Even Contain ARRAY Values

```
#standardSQL
SELECT
STRUCT(35 AS age, 'Jacob' AS name,['apple', 'pear', 'peach'] AS
items) AS customers
```

STRUCTS can contain
Arrays as values

Row	customers.age	customers.name	customers.items
1	35	Jacob	apple
			pear
			peach

ARRAYS can Contain STRUCTs as Values

```
#standardSQL
SELECT
[
STRUCT(35 AS age, 'Jacob' AS name, ['apple', 'pear', 'peach'] AS items),
STRUCT(33 AS age, 'Miranda' AS name, ['water', 'pineapple', 'ice cream'] AS items)
] AS customers
```

ARRAYS can Contain STRUCTS as values

Row	customers.age	customers.name	customers.items
1	35	Jacob	apple
			pear
			peach
	33	Miranda	water
			pineapple
			ice cream

Google Cloud

Filter for Customers who Bought Ice Cream

```
#standardSQL
WITH orders AS (
SELECT
[
STRUCT(35 AS age, 'Jacob' AS name, ['apple', 'pear', 'peach'] AS items),
STRUCT(33 AS age, 'Miranda' AS name, ['water', 'pineapple', 'ice cream'] AS items)
] AS customers
)
```

CROSS JOIN and UNNEST Flattens arrays so we can access elements

```
SELECT
  customers
FROM orders AS o
CROSS JOIN UNNEST(o.customers) AS customers
WHERE 'ice cream' IN UNNEST(customers.items)
```

← Filter on items Array with UNNEST and using IN

Row	customers.age	customers.name	customers.items
1	33	Miranda	water
			pineapple
			ice cream

Google Cloud

Nested (Repeated) Records are **Arrays of Structs**

- Nested records in BigQuery are Arrays of Structs.
- Instead of Joining with a sql_on: expression, **the join relationship is built into the table**.
- UNNESTing a ARRAY of STRUCTs is similar to joining a table.

Google Cloud

Working with Repeated Fields

1. Introducing **Arrays and Structs**
2. **Flattening Arrays:** Legacy vs Standard
3. **Practicing SQL** with Repeated Fields

Google Cloud

Legacy vs Standard SQL Repeated Record Differences

Legacy SQL Syntax

- Flattening happens explicitly with FLATTEN

Functions:

- WITHIN RECORD
- NEST

Standard SQL Syntax

- **Flattening happens implicitly** or explicitly with CROSS JOIN + UNNEST

Functions:

- ARRAY_LENGTH
- ARRAY_AGG

More Details

Working with Repeated Fields

1. Introducing **Arrays and Structs**
2. **Flattening Arrays:** Legacy vs Standard
3. **Practicing SQL** with Repeated Fields

Google Cloud

ARRAY/STRUCT example

```
# Top two Hacker News articles by day
WITH TitlesAndScores AS (
  SELECT
    ARRAY_AGG(STRUCT(title, score)) AS titles,
    EXTRACT(DATE FROM time_ts) AS date
  FROM `bigquery-public-data.hacker_news.stories`
  WHERE score IS NOT NULL AND title IS NOT NULL
  GROUP BY date)

SELECT date,
  ARRAY(SELECT AS STRUCT title, score
        FROM UNNEST(titles) ORDER BY score DESC
        LIMIT 2)
  AS top_articles
FROM TitlesAndScores;
```

WITH Clause:
- Make an array of (title, score) objects
- Extract the date from the timestamp
- Group by the date (which gives us the array contents)

ARRAY(SELECT AS STRUCT:
- Unnest the array from the WITH clause
- order it and take the top 2
- Create a new array of (title, score) objects

outer query:
- Project date from WITH clause
- Project Array

Google Cloud

ARRAY/STRUCT example result

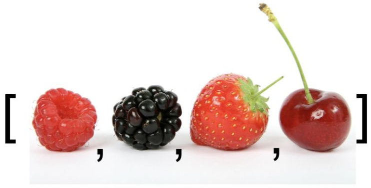
Row	date	top_articles.title	top_articles.score
1	2010-08-23	Why GNU grep is Fast	512
		Readme Driven Development	244
2	2010-04-26	Police raid Gizmodo editor's house	257
		Not even in South Park?	257
3	2009-09-15	Learning Advanced JavaScript	257
		Sub-pixel re-workings of YouTube and BBC favicons	154

Google Cloud

Summary: BigQuery architecture is designed for petabyte-scale querying performance

Tables are broken into pieces, called shards, to allow for scalability

BigQuery uses compressed column-based storage for fast retrieval



Structs and arrays are data type containers that are foundational to repeated fields

Row	date	top_articles.title
1	2010-08-23	Why GNU grep is Fast
		Readme Driven Development
2	2010-04-26	Police raid Gizmodo editor's house
		Not even in South Park?
3	2009-09-15	Learning Advanced JavaScript
		Sub-pixel re-workings of YouTube and BBC favicons

Tables with repeated fields are conceptually like pre-joined tables

Google Cloud

Lab 10

Querying Nested and Repeated Data

Google Cloud

Querying Nested and Repeated Data

In this lab, you will practice querying Nested and Repeated Fields using array manipulation and structs.

Results | Explanation | Job Information

Row	ein	name	expense_struct.type	expense_struct.amount	revenue_struct.type	revenue_struct.amount
1	510203813	IF	Lobbying	0	Contributions	110796
			Legal	0	Programs	0
			Insurance	250	Fundraising	0
			Travel	0		
			Ads Promotion	180		
			Office	9147		
2	364236669	ARF	Lobbying	0	Contributions	151818
			Legal	0	Programs	0
			Insurance	0	Fundraising	817
			Travel	0		
			Ads Promotion	5859		
			Office	16497		

Google Cloud