Ignore The Recipe

AGGREGATION

Part 0: A introductory rant

Part Negative-1: Pre-Rant Rant

But...

Ok. Part 0: Rant

But more than all that...

Part 1: An example

Why would one want to create a UDF aggregate?

Why do aggregates have all those rules?

```
SELECT MIN(CHARACTER_NAME) AS MIN_NAME
FROM COMICS_CHARACTER;
```

```
SELECT MIN(CHARACTER_NAME) AS MIN_NAME
FROM COMICS_CHARACTER;
```

MIN_NAME
ALIEN

```
SELECT MAX(CHARACTER_NAME) AS MAX_NAME
FROM COMICS_CHARACTER;
```

```
SELECT MAX(CHARACTER_NAME) AS MAX_NAME
FROM COMICS_CHARACTER;
```

MAX_NAME
blade

blade ... ?

Or, another example

```
SELECT MAX(OS_NAME) AS MAX_OS_NAME
FROM OPERATING_SYSTEM;
```

Or, another example

```
SELECT MAX(OS_NAME) AS MAX_OS_NAME
FROM OPERATING_SYSTEM;
```

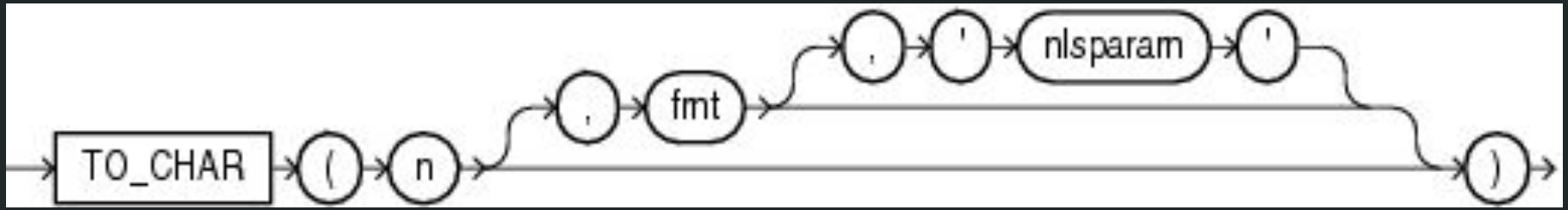
MAX_OS_NAME
macOS

macOS ... ?

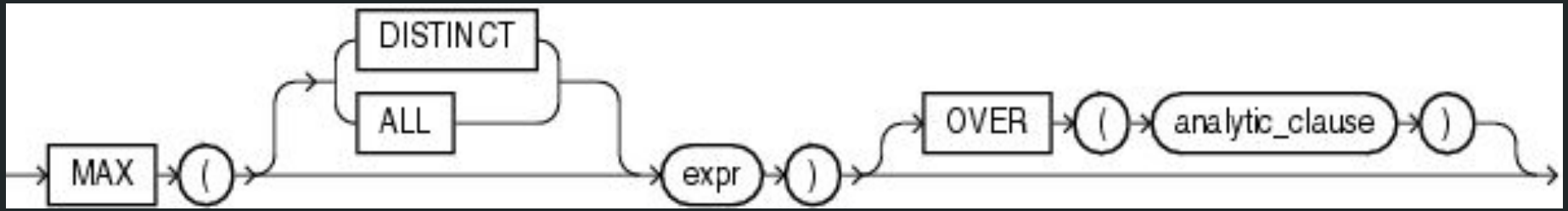
Apple… why?

There must be some way

Compare



Compare



Grrr.... fine.

```
ALTER SESSION SET NLS_COMP = 'LINGUISTIC';

ALTER SESSION SET NLS_SORT = 'BINARY_AI';
```

Now, it “works”…

```
ALTER SESSION SET NLS_COMP = 'LINGUISTIC';
ALTER SESSION SET NLS_SORT = 'BINARY_AI';

SELECT MAX(CHARACTER_NAME) AS MAX_NAME
FROM COMICS_CHARACTER;
```

```
ALTER SESSION SET NLS_COMP = 'LINGUISTIC';
ALTER SESSION SET NLS_SORT = 'BINARY_AI';

SELECT MAX(CHARACTER_NAME) AS MAX_NAME
FROM COMICS_CHARACTER;
```

MAX_NAME
??

Or, another example

```
ALTER SESSION SET NLS_COMP = 'LINGUISTIC';
ALTER SESSION SET NLS_SORT = 'BINARY_AI';

SELECT MAX(OS_NAME) AS MAX_OS_NAME
FROM OPERATING_SYSTEM;
```

MAX_OS_NAME
??

Or, another example

```
ALTER SESSION SET NLS_COMP = 'LINGUISTIC';
ALTER SESSION SET NLS_SORT = 'BINARY_AI';

SELECT MAX(OS_NAME) AS MAX_OS_NAME
FROM OPERATING_SYSTEM;
```

MAX_OS_NAME
Z/Os

But...

X-session query portability

Couldn’t it just take care of all that, internally?

And what about the NULLs? Can I override that?

But...

Did you remember to set those session variables back....?

Maybe this?

```
SELECT OS_NAME AS MAX_OS_NAME
FROM OPERATING_SYSTEM
ORDER BY NLSSORT(OS_NAME,'NLS_SORT = BINARY_AI')
OFFSET 0 ROWS FETCH FIRST 1 ROWS ONLY;
```

MAX_OS_NAME
Z/Os

Wait a second...

That’s not even aggregating!

How ‘bout this??

```
SELECT MAX(UPPER(CHARACTER_NAME)) AS MAX_NAME
FROM COMICS_CHARACTER;
```

Yeah... how about that....

This one! Booyah!

```
SELECT MAX(OS_NAME)
 KEEP (DENSE_RANK LAST ORDER BY
NLSSORT(OS_NAME,'NLS_SORT=BINARY_AI'))
FROM OPERATING_SYSTEM;
```

Booyah? Really? But..

Part 2: Aggregation mechanics

A common theme

In the docs:

<<function>> returns the <<whatever data>> where *expr* is not null

Why do they all say that?

And other questions

How costly is a rollup?
How costly are multi-attribute groupings?
Is an analytic function more costly than an aggregate?
How does windowing work?
Why are aggregate functions so strict?

An experiment to explore

Our dataset

```
SELECT COMICS_UNIVERSE,
    CHARACTER_NAME,
    CHARACTER_ALIGNMENT AS ALIGNMENT,
    CHARACTER_SKIN_COLOR AS SKIN_COLOR
FROM COMICS_CHARACTER
ORDER BY CHARACTER_NAME ASC;
```

Our dataset

COMICS_UNIVERSE	CHARACTER_NAME	ALIGNMENT	SKIN_COLOR
DARK HORSE COMICS	ALIEN	EVIL	BLACK
MARVEL COMICS	ANT-MAN	GOOD	null
DC COMICS	AQUAMAN	GOOD	null
DC COMICS	BANE	EVIL	null
MARVEL COMICS	BANSHEE	GOOD	null
DC COMICS	BATMAN	GOOD	null
DC COMICS	BEAST BOY	GOOD	GREEN
DC COMICS	BIZARRO	NEUTRAL	GRAY

ELEMENT_COUNT

```
SELECT
        ELEMENT_COUNT(CHARACTER_NAME)
        AS CHARACTER_COUNT
FROM COMICS_CHARACTER;
```

CHARACTER_COUNT
50

What really just happened?

Aggregation is just Reduction

Aggregation = Reduction

Simplest reduction steps:

- Grab every candidate element
- Apply some function that combines them in pairs

But what if

- What if there’s only one element?

Aggregation = Reduction

Simplest reduction steps:

- Grab every candidate element
- Apply some function that combines them in pairs
- **Some initialization to reduce the first element**

But what if

- What if the final result requires some calculation?
  - ***SUM*/*COUNT*/*LISTAGG***

    vs

  - ***MEDIAN*/*PERCENTILE_DISC*/*PERCENT_RANK***

Aggregation = Reduction

~~Simplest~~ Reduction steps:

- Grab every candidate element
- Apply some function that combines them in pairs
- Some initialization to reduce the first element
- **A finalizer to return the calculated reduction**

But what if

- What if the data set is big and I want to run in parallel?

Aggregation = Reduction

~~Simplest~~ Reduction steps:

- Grab every candidate element
- Apply some function that combines them in pairs
- Some initialization to reduce the first element
- A finalizer to return the calculated reduction
- **A partial-reduction merger**

Measures

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
12	345	6	7

Disclaimers!!!

- Benchmarks are relative!
- Builtins are...builtin. **
- Don’t forget parse, compilation, optimization

Remember our dataset?

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Simplest Aggregation

```
SELECT ELEMENT_COUNT(CHARACTER_NAME)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Simplest Aggregation

```
SELECT ELEMENT_COUNT(CHARACTER_NAME)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
1	50	0	1

Those docs all say … when not NULL

Those docs all say … when not NULL

```
SELECT ELEMENT_COUNT(CHARACTER_SKIN_COLOR)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Those docs all say … when not NULL

```
SELECT ELEMENT_COUNT(CHARACTER_SKIN_COLOR)
FROM COMICS_CHARACTER;
```

THE_COUNT
11

Those docs all say … when not NULL

```
SELECT ELEMENT_COUNT(CHARACTER_SKIN_COLOR)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
1	11	0	1

1-Attribute Group-By

```
SELECT COMICS_UNIVERSE,
    ELEMENT_COUNT(CHARACTER_NAME) AS COUNT_PER_UNIVERSE
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE;
```

COMICS_UNIVERSE	COUNT_PER_UNIVERSE
DARK HORSE COMICS	5
DC COMICS	22
MARVEL COMICS	23

1-Attribute Group-By

```
SELECT COMICS_UNIVERSE,
    ELEMENT_COUNT(CHARACTER_NAME) AS COUNT_PER_UNIVERSE
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
3	50	0	3

2-Attribute Group-By

```
SELECT
  COMICS_UNIVERSE,
  CHARACTER_ALIGNMENT,
  ELEMENT_COUNT(1) AS COUNT_PER_FACTION
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE, CHARACTER_ALIGNMENT;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

2-Attribute Group-By

COMICS_UNIVERSE	CHARACTER_ALIGNMENT	COUNT_PER_FACTION
DC COMICS	GOOD	12
DC COMICS	NEUTRAL	3
DC COMICS	EVIL	7
MARVEL COMICS	GOOD	15
MARVEL COMICS	NEUTRAL	2
MARVEL COMICS	EVIL	6
DARK HORSE COMICS	GOOD	1
DARK HORSE COMICS	EVIL	4

2-Attribute Group-By

```
SELECT
  COMICS_UNIVERSE,
  CHARACTER_ALIGNMENT,
  ELEMENT_COUNT(1) AS COUNT_PER_FACTION
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE, CHARACTER_ALIGNMENT;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
8	50	0	8

GROUP BY impact

2-D Aggregate ROLLUP

```
SELECT ALL COMICS_UNIVERSE,
        ELEMENT_COUNT(1) AS CHARACTER_COUNT
FROM COMICS_CHARACTER
GROUP BY ROLLUP (COMICS_UNIVERSE);
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

2-D Aggregate ROLLUP

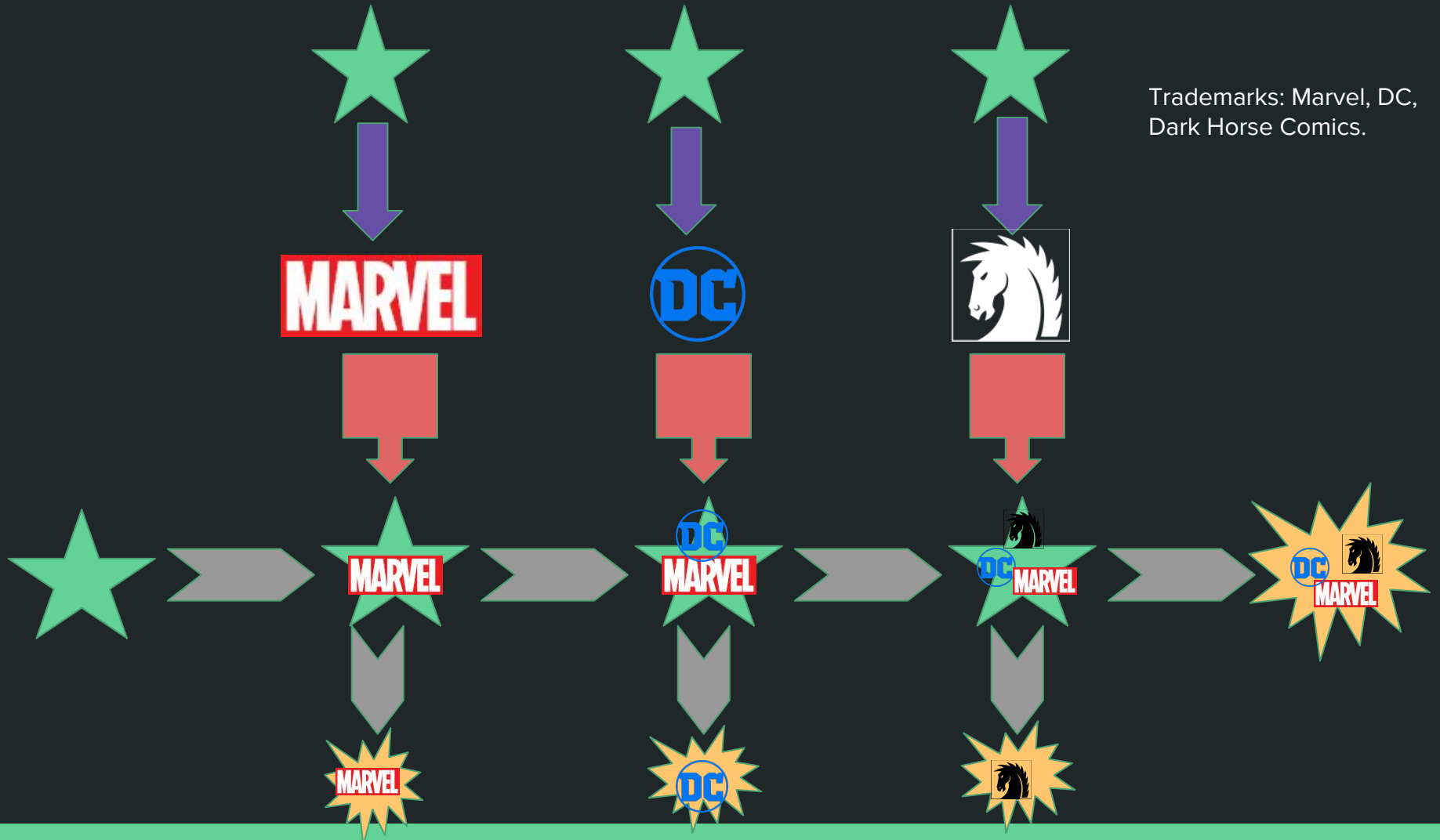
```
SELECT ALL COMICS_UNIVERSE,
        ELEMENT_COUNT(1) AS CHARACTER_COUNT
FROM COMICS_CHARACTER
GROUP BY ROLLUP (COMICS_UNIVERSE);
```

COMICS_UNIVERSE	CHARACTER_COUNT
DARK HORSE COMICS	5
DC COMICS	22
MARVEL COMICS	23
null	50

2-D Aggregate ROLLUP

```
SELECT ALL COMICS_UNIVERSE,
        ELEMENT_COUNT(1) AS CHARACTER_COUNT
FROM COMICS_CHARACTER
GROUP BY ROLLUP (COMICS_UNIVERSE);
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
4	50	3	4



Trademarks: Marvel, DC, Dark Horse Comics.

2-Attribute CUBE

```
SELECT
    COMICS_UNIVERSE,
    CHARACTER_ALIGNMENT,
    ELEMENT_COUNT(1)
FROM COMICS_CHARACTER
GROUP BY CUBE(COMICS_UNIVERSE, CHARACTER_ALIGNMENT);
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Rewind

COMICS_UNIVERSE	CHARACTER_ALIGNMENT	COUNT_PER_FACTION
DC COMICS	GOOD	12
DC COMICS	NEUTRAL	3
DC COMICS	EVIL	7
MARVEL COMICS	GOOD	15
MARVEL COMICS	NEUTRAL	2
MARVEL COMICS	EVIL	6
DARK HORSE COMICS	GOOD	1
DARK HORSE COMICS	EVIL	4

Let’s do the math

Let’s do the math

8x Universe-Alignment pairs

Plus:

3x Universe Rollups

3x Alignments Rollups

1x Set Rollup

Let’s do the math

15x Initializations ?

15x Terminations ?

Let’s do the math

How many Iterations? 50? 100?

How many Merges? 8? 11?

2-Attribute CUBE

```
SELECT
    COMICS_UNIVERSE,
    CHARACTER_ALIGNMENT,
    ELEMENT_COUNT(1)
FROM COMICS_CHARACTER
GROUP BY CUBE(COMICS_UNIVERSE, CHARACTER_ALIGNMENT);
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
15	101	10	15

Wait, **101??**

2 partial aggregates

50 rows

2 * 50 = 100

Oracle has an optimizer. It optimizes!

100 + 11 = 111

101 + 10 = 111

110 = 110 ?

Iteration = Merge ?

Oracle has an optimizer. It optimizes!

100 + 11 = 111

101 + 10 = 111

Operation Cost
Heuristics
Cardinalities
Skew

Oracle has an optimizer. It optimizes!

What about repeated agg calls?

Simplest Aggregation (Revisited)

```
SELECT ELEMENT_COUNT(CHARACTER_NAME),
    ABS(:COUNTY_FAIR_GUESS - (ELEMENT_COUNT(CHARACTER_NAME)))
    AS GUESS_DELTA
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Simplest Aggregation (Revisited)

```
SELECT ELEMENT_COUNT(CHARACTER_NAME),
    ABS(:COUNTY_FAIR_GUESS - (ELEMENT_COUNT(CHARACTER_NAME)))
    AS GUESS_DELTA
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
1	50	0	1

Side-Note: Built-In Advantage

```
SELECT
    COMICS_UNIVERSE,
    CHARACTER_ALIGNMENT,
    ELEMENT_COUNT(1)
FROM COMICS_CHARACTER
GROUP BY CUBE(COMICS_UNIVERSE, CHARACTER_ALIGNMENT);
```

Side-Note: Built-In Advantage

```
SELECT
    COMICS_UNIVERSE,
    CHARACTER_ALIGNMENT,
    COUNT(*)
FROM COMICS_CHARACTER
GROUP BY CUBE(COMICS_UNIVERSE, CHARACTER_ALIGNMENT);
```

Side-Note: Built-In Advantage

```
SELECT
    COMICS_UNIVERSE,
    CHARACTER_ALIGNMENT,
    COUNT(*)
FROM COMICS_CHARACTER
GROUP BY CUBE(COMICS_UNIVERSE, CHARACTER_ALIGNMENT);
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Side-Side Note: Careful with CUBE...

2^n * c

50 -> 100 -> 200 -> 400

What about Analytics?

Simplest Analytic

```
SELECT
    CHARACTER_NAME,
     ELEMENT_COUNT(1) OVER
       (PARTITION BY NULL
        ROWS BETWEEN
        UNBOUNDED PRECEDING
        AND UNBOUNDED FOLLOWING)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Simplest Analytic

CHARACTER_NAME	CHARACTER_COUNT
JOHN CONSTANTINE	50
JUBILEE	50
POISON IVY	50
PREDATOR	50
RAVEN	50

Simplest Analytic

```
SELECT
    CHARACTER_NAME,
     ELEMENT_COUNT(1) OVER
       (PARTITION BY NULL
        ROWS BETWEEN
        UNBOUNDED PRECEDING
        AND UNBOUNDED FOLLOWING)
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
1	50	0	**51**

This is off too. Why **51**?

Partitioned Analytic

```
SELECT
    CHARACTER_NAME,
     ELEMENT_COUNT(1) OVER
       (PARTITION BY COMICS_UNIVERSE
        ROWS BETWEEN
        UNBOUNDED PRECEDING
        AND UNBOUNDED FOLLOWING) AS COUNT_IN_UNIVERSE
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Partitioned Analytic

CHARACTER_NAME	COUNT_IN_UNIVERSE
ALIEN	5
HELLBOY	5
PREDATOR	5
T-800	5
T-1000	5
PHANTOM	22

Partitioned Analytic

```
SELECT
    CHARACTER_NAME,
     ELEMENT_COUNT(1) OVER
       (PARTITION BY COMICS_UNIVERSE
        ROWS BETWEEN
        UNBOUNDED PRECEDING
        AND UNBOUNDED FOLLOWING) AS COUNT_IN_UNIVERSE
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
3	50	0	51

PARTITION == GROUP BY

Not just logically, but mechanically

Windowed Analytic

```
SELECT CHARACTER_NAME,
 ELEMENT_COUNT(1)
   OVER (PARTITION BY NULL
   ROWS BETWEEN
   10 PRECEDING AND 10 FOLLOWING) AS CHARACTER_COUNT
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Windowed Analytic

CHARACTER_NAME	CHARACTER_COUNT
JOHN CONSTANTINE	11
JUBILEE	12
JUGGERNAUT	13
...	...
POISON IVY	21
BANSHEE	21
...	...
SCARECROW	16
STARFIRE	15

Windowed Analytic

```
SELECT CHARACTER_NAME,
 ELEMENT_COUNT(1)
   OVER (PARTITION BY NULL
   ROWS BETWEEN
   10 PRECEDING AND 10 FOLLOWING) AS CHARACTER_COUNT
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
40	**785**	0	**51**

785? Ouch!! What happened?

785? Ouch!! What happened?

10 preceding
\+ Current Row
\+ 10 following

21-Element Window

785? Ouch!! What happened?

ROW#	PRECEDING	CURRENT_ROW	FOLLOWING	WINDOW	NEW_ELEM	TOTAL
1	0	1	10	11	11	11

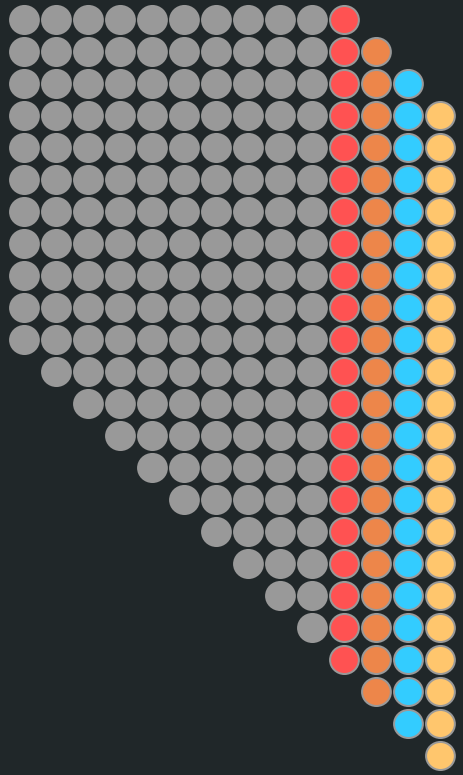
785? Ouch!! What happened?

ROW#	PRECEDING	CURRENT_ROW	FOLLOWING	WINDOW	NEW_ELEM	TOTAL
1	0	1	10	11	11	11
2	1	1	10	12	1	12
3	2	1	10	13	1	13
...	...	...	...	...	...	...
...	...	...	...	...	...	...
10	9	1	10	20	1	20
11	**10**	1	10	**21**	1	21

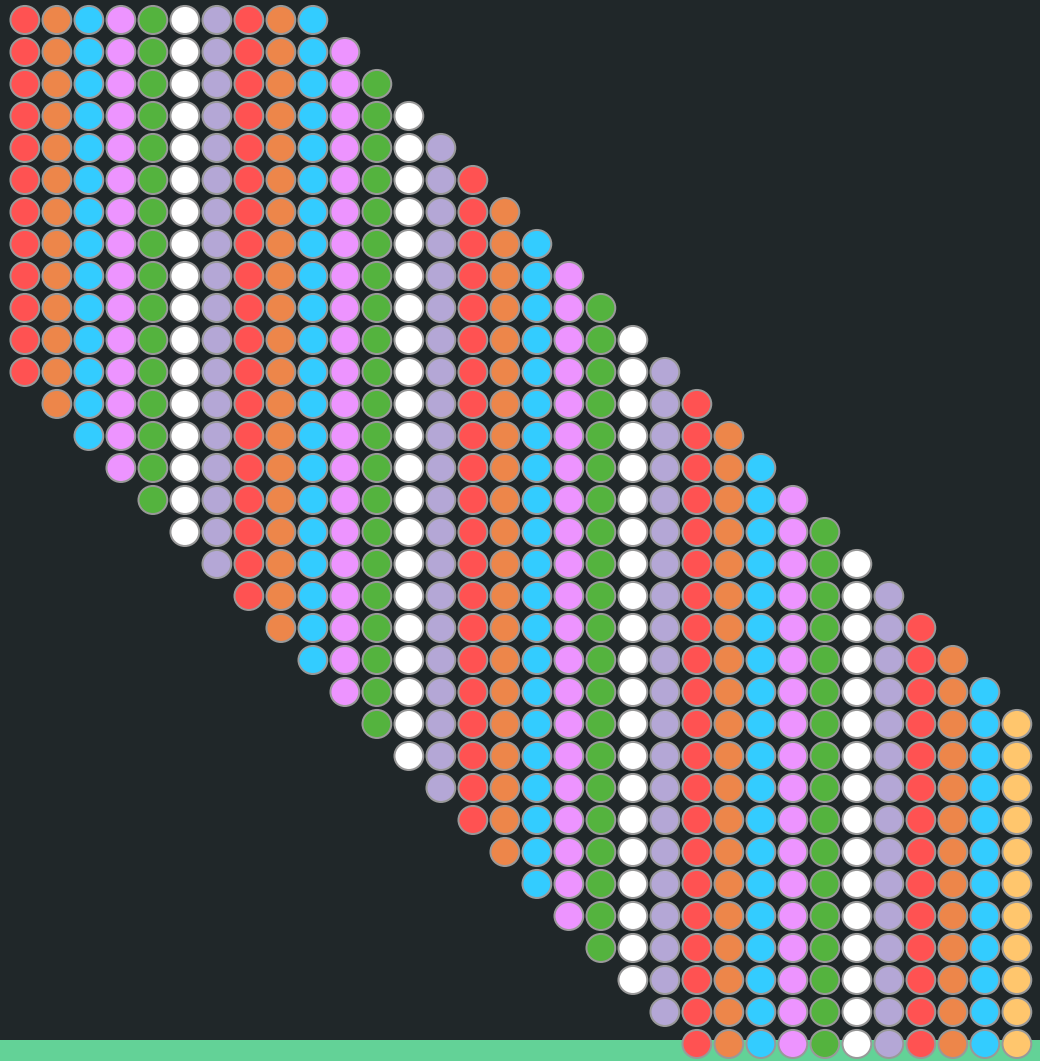
That doesn’t seem so bad…

ROW#	PRECEDING	CURRENT_ROW	FOLLOWING	WINDOW	NEW_ELEM	TOTAL
...	...	...	...	...	...	...
10	9	1	10	20	1	20
11	10	1	10	21	1	21
12	10	1	10	21	**21**	**42**

ROW#	PRECEDING	CURRENT_ROW	FOLLOWING	WINDOW	NEW_ELEM	TOTAL
...	...	...	...	...	...	...
10	9	1	10	20	1	20
11	10	1	10	21	1	21
12	10	1	10	21	21	42
13	10	1	10	21	**21**	**63**



ROW#	PRECEDING	CURRENT_ROW	FOLLOWING	WINDOW	NEW_ELEM	TOTAL
...	...	...	...	...	...	...
10	9	1	10	20	1	20
11	10	1	10	21	1	21
12	10	1	10	21	21	42
13	10	1	10	21	21	63
...	...	...	...	...	...	...
40	10	1	10	21	21	630
41	10	1	9	20	20	650
42	10	1	8	19	19	669
...	...	...	...	...	...	...
50	10	1	**0**	11	**11**	**785**



Nearly 1K function calls for just 50 rows…
How about 1M rows?

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
??	??	??	??

Nearly 1K function calls for just 50 rows...
How about **1M** rows?

INITIALIZATIONS	ITERATIONS	MERGES	TERMINATIONS
999990	**20,999,735**	0	1000001

But that’s terrible!
There must be some way around

You’re in luck.

IF the aggregate is **reversible**, great things happen.

Don’t re-calculate the full window, just:

- Delete one distal element
- Register a new to take its place

Aggregation-Context VS Aggregator

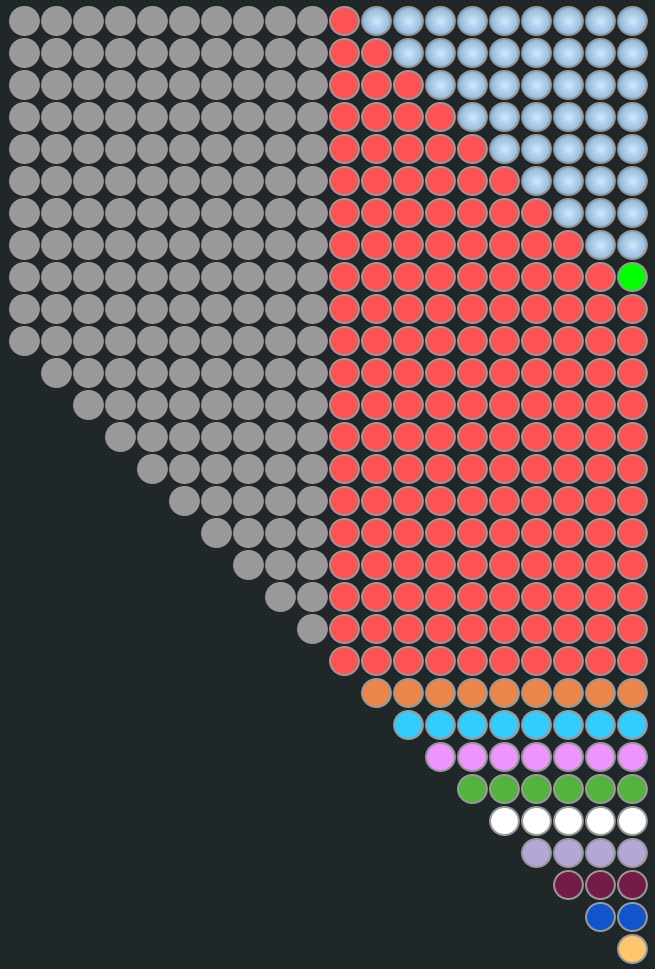
- Multiple participants carry state

Let’s make it ***reversible***, and try again.

Windowed **REVERSIBLE** Analytic

```
SELECT CHARACTER_NAME,
 ELEMENT_COUNT(1)
   OVER (PARTITION BY NULL
   ROWS BETWEEN
   10 PRECEDING AND 10 FOLLOWING) AS CHARACTER_COUNT
FROM COMICS_CHARACTER;
```

INITIALIZATIONS	DELETIONS	ITERATIONS	MERGES	TERMINATIONS
1	**39**	**50**	0	**51**



So reversible is ***always*** better??

Windowed take-away

An irreversible windowed analytic is vastly more expensive

A reversible windowed analytic is more costly than full-window, but is reasonable

Part 3: User-Defined Aggregates

How to implement a UDF Aggregate

We've already seen it!

INITIALIZE

ITERATE

MERGE

TERMINATE

DELETE

How to implement a UDF Aggregate

INITIALIZE => ODCIAggregateInitialize

ITERATE => ODCIAggregateIterate

MERGE => ODCIAggregateMerge

TERMINATE => ODCIAggregateTerminate

**DELETE* => ODCIAggregateDelete*

***DELEGATE** => ODCIAggregateWrapContext*

How to implement a UDF Aggregate

- CREATE TYPE <<AGGREGATOR>> ... <<ODCIAggregate Interfaces>> ...

- CREATE TYPE BODY <<AGGREGATOR>> <<ODCIAggregate Implementation>>

- CREATE FUNCTION <<FUNCTION>>(<<PARAM>>) RETURN <<TYPE>> PARALLEL_ENABLE AGGREGATE USING <<AGGREGATOR>>;

WHY to create a UDF Aggregate?

Same reasons as for any function!

- Encapsulation
- Reusability
- Productivity
- Readability
- Performance?

Compare

```
SELECT
    CHARACTER_NAME AS RANDOM_NAME
FROM COMICS_CHARACTER
 ORDER BY DBMS_RANDOM.VALUE(1,2123456789)
 OFFSET 0 ROWS FETCH FIRST 1 ROW ONLY;
```

Compare

```
SELECT
    RANDOM_ELEMENT(CHARACTER_NAME) AS RANDOM_NAME
FROM COMICS_CHARACTER;
```

Not worth it? How about:
random character ***with a random alignment***?

Compare: non-aggregate?

```
SELECT
    CHARACTER_NAME AS RANDOM_NAME,
    (SELECT
        COMICS_CHARACTER.CHARACTER_ALIGNMENT
      FROM COMICS_CHARACTER
        ORDER BY DBMS_RANDOM.VALUE(1,2123456789)
        OFFSET 0 ROWS FETCH FIRST 1 ROW ONLY) AS
RANDOM_ALIGNMENT
FROM COMICS_CHARACTER
    ORDER BY DBMS_RANDOM.VALUE(1,2123456789)
    OFFSET 0 ROWS FETCH FIRST 1 ROW ONLY;
```

Or maybe:

```
SELECT
    MAX(CHARACTER_NAME) KEEP (DENSE_RANK FIRST ORDER BY
DBMS_RANDOM.VALUE(1,2123456789) ASC)  AS RANDOM_NAME,
    MAX(CHARACTER_ALIGNMENT) KEEP (DENSE_RANK FIRST ORDER BY
DBMS_RANDOM.VALUE(1,2123456789) ASC)  AS RANDOM_ALIGNMENT
FROM COMICS_CHARACTER;
```

To

```
SELECT
  RANDOM_ELEMENT(CHARACTER_NAME) AS RANDOM_NAME,
  RANDOM_ELEMENT(CHARACTER_ALIGNMENT) AS RANDOM_ALIGNMENT
FROM COMICS_CHARACTER;
```

Or this?

```
SELECT
  LISTAGG(DISTINCT CHARACTER_NAME,',')
    WITHIN GROUP(ORDER BY COMICS_UNIVERSE ASC,
CHARACTER_NAME ASC)
FROM COMICS_CHARACTER;
```

ORA-30482: DISTINCT option not allowed for this function *

*** **Side-Note: Oracle 19! Yaya!!!!** ***

Or this?

```
SELECT
  LISTAGG(CHARACTER_NAME,',')
    WITHIN GROUP(ORDER BY COMICS_UNIVERSE ASC,
CHARACTER_NAME ASC)
FROM COMICS_CHARACTER;
```

ORA-01489: result of string concatenation is too long

Or this?

Compare

```
SELECT
  COMICS_UNIVERSE,
  MIN(CHARACTER_WEIGHT) AS MIN_WEIGHT,
  MAX(CHARACTER_WEIGHT) AS MAX_WEIGHT,
  AVG(CHARACTER_WEIGHT) AS AVG_WEIGHT,
  MEDIAN(CHARACTER_WEIGHT) AS MED_WEIGHT,
  SUM(CHARACTER_WEIGHT) AS TOTAL_WEIGHT,
  COUNT(CHARACTER_WEIGHT) AS ITEM_COUNT,
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE;
```

VS

```
SELECT
   COMICS_UNIVERSE,
   COMSTAT(CHARACTER_WEIGHT) AS COMSTAT
FROM COMICS_CHARACTER
GROUP BY COMICS_UNIVERSE;
```

```
SELECT MAX(CHARACTER_NAME)
    KEEP (DENSE_RANK FIRST
        ORDER BY CHARACTER_HEIGHT ASC)  AS SHORTEST_CHARACTER
FROM COMICS_CHARACTER;
```

SHORTEST_CHARACTER
RAVEN

VS

```
SELECT MIN(CHARACTER_NAME)
    KEEP (DENSE_RANK FIRST
        ORDER BY CHARACTER_HEIGHT ASC)  AS SHORTEST_CHARACTER
FROM COMICS_CHARACTER;
```

SHORTEST_CHARACTER
JUBILEE

```
SELECT HIGHLANDER(CHARACTER_NAME)
    KEEP (DENSE_RANK FIRST
        ORDER BY CHARACTER_HEIGHT ASC)  AS SHORTEST_CHARACTER
FROM COMICS_CHARACTER;
```

ORA-20476: There can be only One!

Or this????????

```
SELECT
    LING_MAX(CHARACTER_NAME) AS MAX_NAME
FROM COMICS_CHARACTER;
```

MAX_NAME
YMIR

```
SELECT
    NLSMAX(NLSSORTER(CHARACTER_NAME,'BINARY_AI')) AS MAX_NAME
FROM COMICS_CHARACTER;
```

Summary

Aggregator vs Aggregation-Context

Four (+ 1) (…. +2) aggregation functions used

To create a User-Defined Aggregate, just implement the 4(/5/6) interfaces

The sql engine is amazingly efficient! The optimizer has all its tricks

Be confident with ***multi-attribute*** aggregation! Cost increase is very, very low

Be confident with ***multi-dimensional*** aggregation! Cost increase is very, very low

Be confident with ***full-partition*** analytics. (**PARTITION** vs **GROUP**)

Be confident with (**REVERSIBLE**) ***windowed*** analytics (**Delete + Add**)

Be confident with (**IRREVERSIBLE**) ***windowed*** analytics and CUBEs
-- By knowing their costs, one plan accordingly.