Maintaining Materialized Aggregate Views

Abhijeet Mohapatra and Michael Genesereth

STANFORD
UNIVERSITY

What are Views?

- Views are *virtual* tables

purchases

Name	Car	Year
John	Ferrari	2010
John	Mercedes	2010
Mary	Jaguar	2010
Mary	Ferrari	2010

v

Car
Ferrari
Mercedes
Jaguar

v(Car) :- *purchases*(Name, Car, Year)

Materialized Views

- Speed up the time taken to execute a query

purchases

Name	Car	Year
John	Ferrari	2010
John	Mercedes	2010
Mary	Jaguar	2010
Mary	Ferrari	2010

v

Car
Ferrari
Mercedes
Jaguar

v(Car) :- *purchases*(Name, Car, Year)

- However, there is a price to pay...

Maintaining Views

- When tables are updated, materialized views must be updated as well

purchases

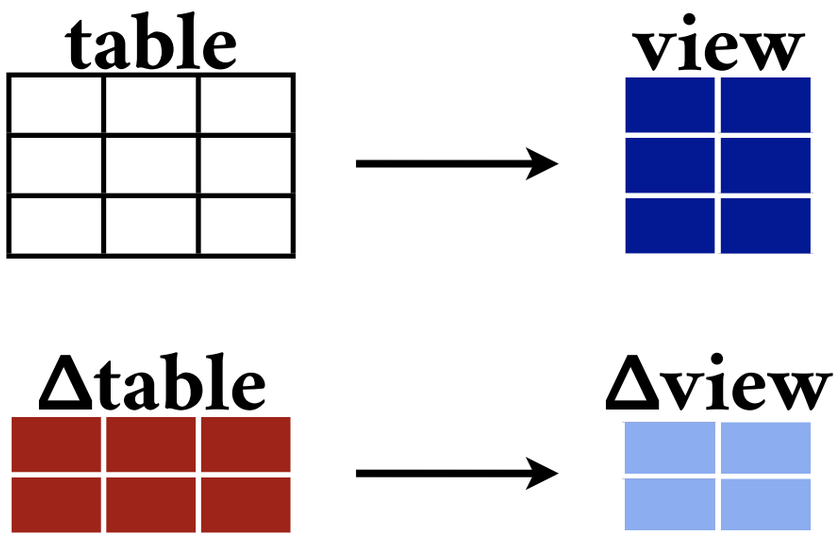
Name	Car	Year
John	Ferrari	2010
Mary	Jaguar	2010
Mary	Ferrari	2010

v

Car
Ferrari
Jaguar

- One can always recompute the view, but this defeats the purpose of materializing a view

Incremental Maintenance



View:

$v(X, Y) :- r(X, Z) \,\&\, s(Z, Y)$

Differential Rules:

$v^{+}(X, Y) :- r^{+}(X, Z) \,\&\, s(Z, Y)$

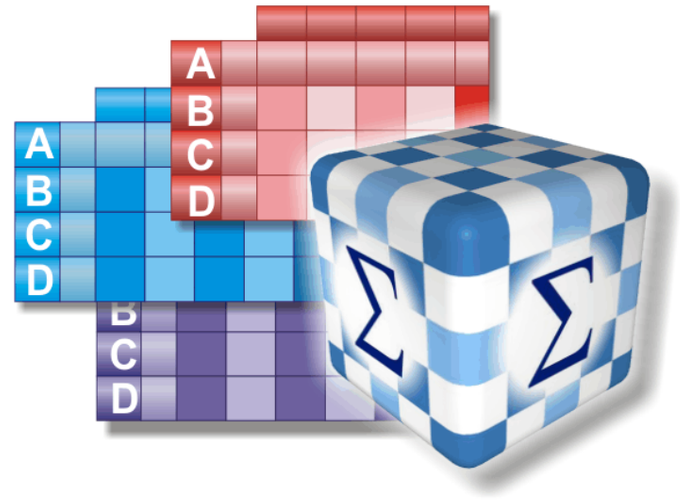
$v^{+}(X, Y) :- r(X, Z) \,\&\, s^{+}(Z, Y)$

$v^{+}(X, Y) :- r^{+}(X, Z) \,\&\, s^{+}(Z, Y)$

- **Differential Rules** (also called *delta rules*):
  [Blakeley '86, Gupta '93, Orman '98, Gupta '06]

What about Aggregate Views?

"*Aggregates can have a very significant effect on performance, in some cases speeding queries by a factor of one hundred or even one thousand.*" - ***Ralph Kimball***



purchases

Name	Car	Year
John	Ferrari	2010
John	Mercedes	2010
Mary	Jaguar	2010
Mary	Bugatti	2010
John	Jaguar	2013

query

Name	NCars	Year
John	2	2010
Mary	2	2010
John	1	2013

query(Name, NCars, Year)
:- Cars = **setof(**Car, *purchases*(Name, Car, Year)**)** & **count(**Cars, NCars**)**

Maintaining Aggregate Views

- Prior work [Griffin ’95, Gupta ’93 and ’06, Quass ’96]) handles only a fixed set of *built-in* aggregates- sum, count, min and max

We focus on *setof* aggregates[1] and adapt ***Differential Rules*** to incrementally update aggregate views

[1] *Aggregation in Datalog Under Set Semantics*
http://logic.stanford.edu/reports/LG-2012-01.pdf

Differential Rules

query(Name, Cars, Year)
:- Cars = **setof(**Car, *purchases*(Name, Car, Year)**)**

query

Name	Cars	Year
John	{Ferrari, Mercedes}	2010
Mary	{Jaguar, Bugatti}	2010
John	{Jaguar, Bugatti}	2013

purchases^{+}

Name	Car	Year
John	Bugatti	2013

query^{-}(Name, Cars, Year)
:- ΔCars = **setof(**Car, *purchases*$^{+}$(Name, Car, Year)**)** &
query(Name, Cars, Year)

query^{-}

Name	Cars	Year
John	{Jaguar}	2013

query^{+}(Name, Cars ∪ ΔCars, Year)
:- ΔCars = **setof(**Car, *purchases*$^{+}$(Name, Car, Year)**)** &
query(Name, Cars, Year)

query^{+}

Name	Cars	Year
John	{Jaguar, Bugatti}	2013

Problem with Differential Rules

purchases

Name	Car	Year
John	Ferrari	2010
John	Mercedes	2010
Mary	Jaguar	2010
Mary	Bugatti	2010
John	Jaguar	2013

purchases^{+}

Name	Car	Year
John	Bugatti	2013

query(Name, Cars)
:- Cars = **setof(**Car, *purchases*(Name, Car, Year)**)**

query

Name	Cars
John	{Ferrari, Mercedes}
Mary	{Jaguar, Bugatti}
John	{Jaguar}

update ambiguity!

John	Bugati

Solving the Problem

query(Name, Cars)
:- Cars = **setof(**Car, *purchases*(Name, Car, Year)**)**

Rewrite as:

$query_a$(Name, Cars, Year)
:- Cars = **setof(**Car, *purchases*(Name, Car, Year)**)**

We maintain $query_a$ using *Differential Rules* for sets

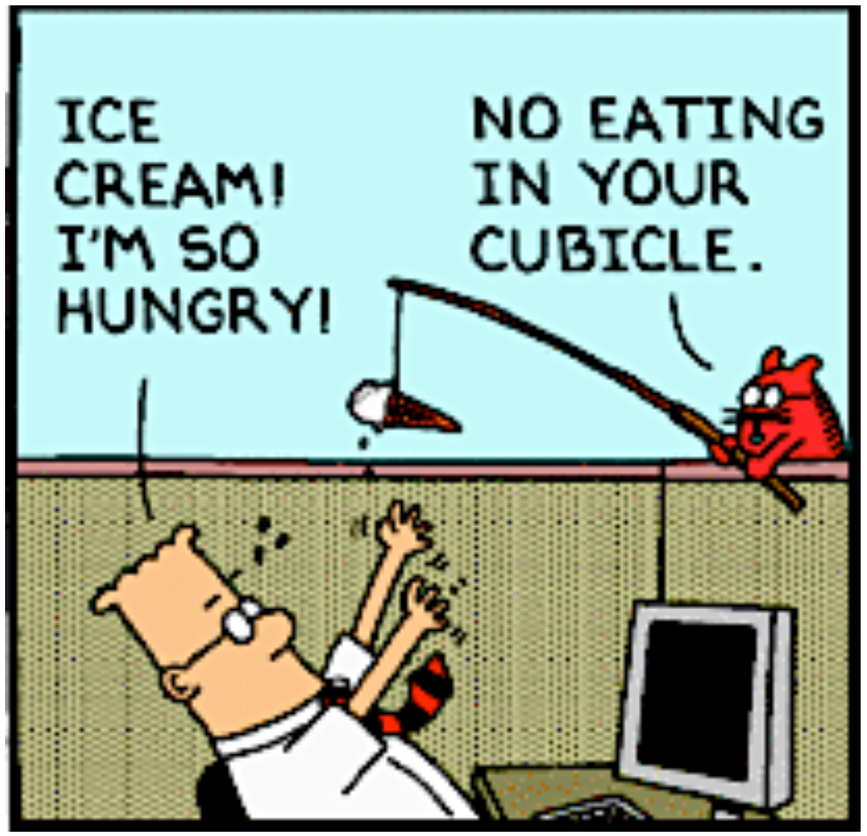
query(Name, Cars)
:- $query_a$(Name, Cars, Year)

We maintain *query* using the *Counting Algorithm* [Gupta '93]
Maintain the derivations of a tuple (Name, Car) $\in$ *query* from $query_a$

(-) more space required
(+) the *purchases* table is not accessed

CReaM Algorithm

- Input: A materialized aggregate view v, and the differentials of tables or views from which v is derived.



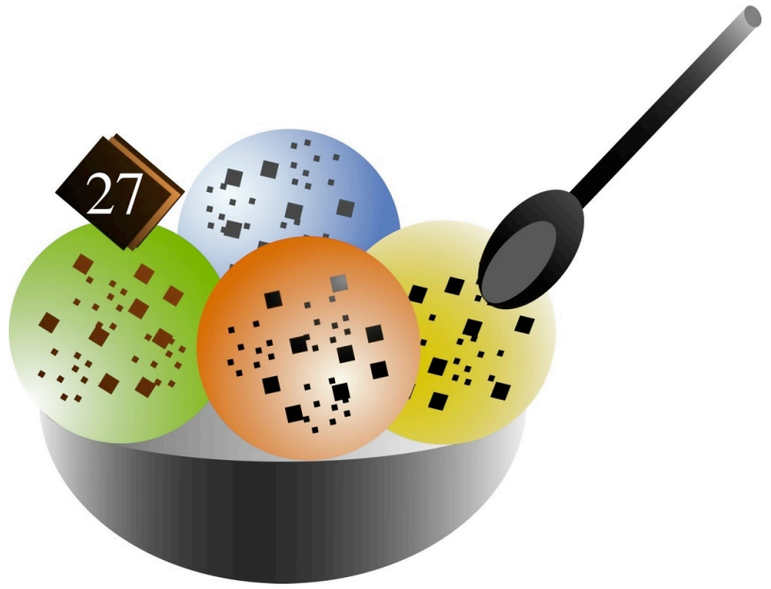
- *If all of the non-aggregated variables are passed to the head in the definition of v,*
  - Maintain v using differential rules.

- *Otherwise,*
  - Rewrite v as an auxiliary view v_a.
  - Maintain v_a using differential rules.
  - Maintain v using counting algorithm.

To summarize...

The Good: Incrementally maintainable

The Bad: More space is required
= linear in input size



The Awesome: If size of the input = O(n),
CReaM takes O(log n) time which is optimal

THANK YOU!