Data statistics for query optimization & Views: maintenance and use

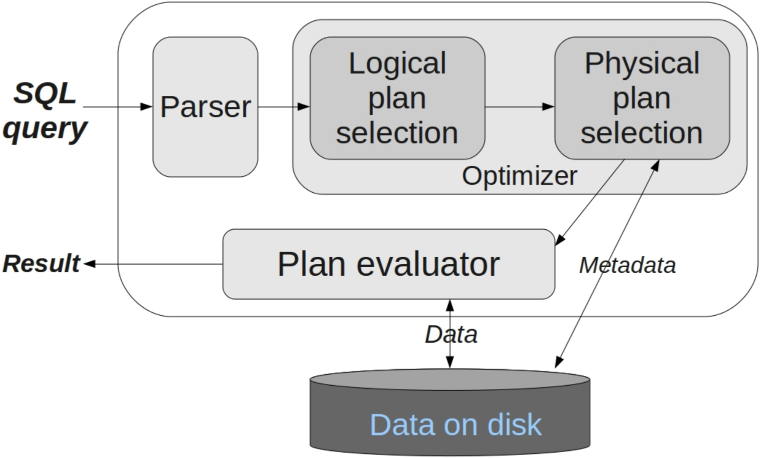
Lecture 6
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

8 May 2015

The life of a query

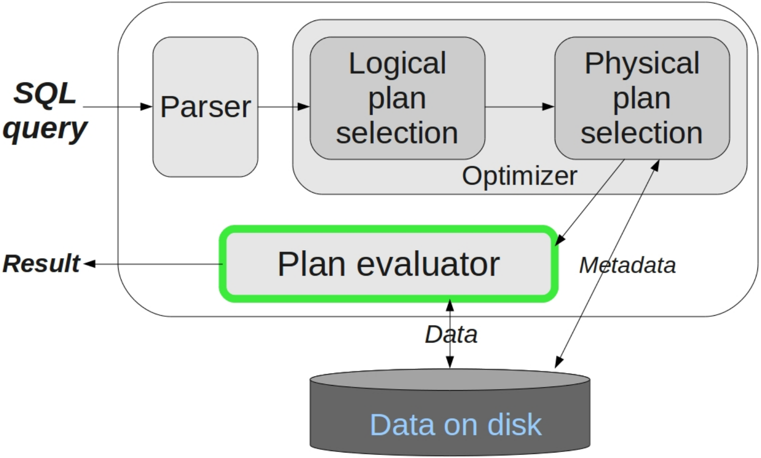


Where we've been

query evaluation

- physical processing of relational operators

The life of a query: evaluation



Where we're headed

query *optimization*

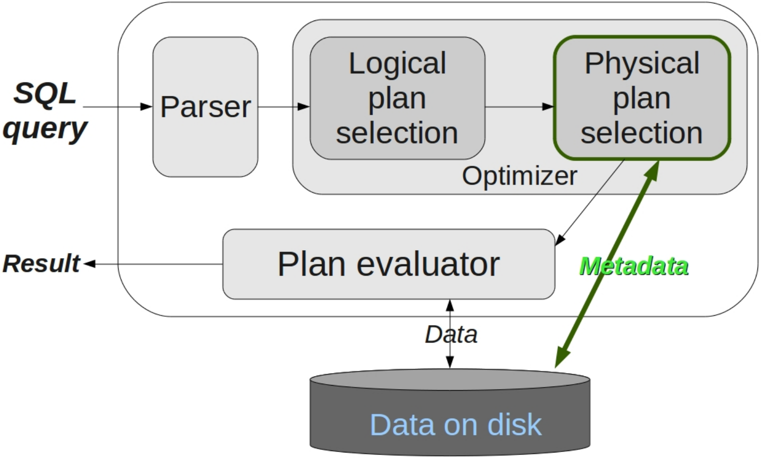
- logical optimization
- physical optimization

Where we're headed

query *optimization*

- logical optimization
- physical optimization
    - today: cost estimation using data statistics

The life of a query: cost estimation



The life of a query: cost estimation

- A plan is a tree of physical operators
  - i.e., operators which access and manipulate physical data

The life of a query: cost estimation

- A plan is a tree of physical operators
  - i.e., operators which access and manipulate physical data
- physical plan indicates
  - algorithm for each node
  - the way stored data is obtained (i.e., access paths)
  - the way in which data is passed between nodes
  - the order in which nodes are performed

The life of a query: cost estimation

- two parts to estimating the cost of a plan
  - for each node, estimate the cost of performing the operation
  - for each node, estimate the result size, and any properties it might have (e.g., sorted)

The life of a query: cost estimation

- two parts to estimating the cost of a plan
  - for each node, estimate the cost of performing the operation
  - for each node, estimate the result size, and any properties it might have (e.g., sorted)
- the overall estimate is obtained by combining these local estimates
  - note, however, that errors in estimates propagate exponentially ...

The life of a query: cost estimation

- two parts to estimating the cost of a plan
  - for each node, estimate the cost of performing the operation
  - for each node, estimate the result size, and any properties it might have (e.g., sorted)
- the overall estimate is obtained by combining these local estimates
  - note, however, that errors in estimates propagate exponentially ...
- keep in mind:
  - cost estimates are truly approximations
  - goal is really to just avoid the *worst* plans

The life of a query: cost estimation

reduction factor:

- the fraction of tuples which satisfy a condition C is called the reduction factor of C

The life of a query: cost estimation

reduction factor:

- the fraction of tuples which satisfy a condition C is called the reduction factor of C
- collectively, the reduction factor of $C_1 \wedge \cdots \wedge C_m$ is $rf_1 \times \cdots \times rf_m$ (assuming statistical independence)

The life of a query: cost estimation

```
SELECT A1, ..., Ak
FROM R1, ..., Rn
WHERE C1 AND ... AND Cm
```

The life of a query: cost estimation

```
SELECT A1, ..., Ak
FROM R1, ..., Rn
WHERE C1 AND ... AND Cm
```

max size:

$$T_{R_1} \times \cdots \times T_{R_n}$$

The life of a query: cost estimation

```
SELECT A1, ..., Ak
FROM R1, ..., Rn
WHERE C1 AND ... AND Cm
```

max size:

$$T_{R_1} \times \cdots \times T_{R_n}$$

estimate of actual size:

$$(rf_1 \times \cdots \times rf_m) \times (T_{R_1} \times \cdots \times T_{R_n})$$

The life of a query: cost estimation

- for each RA implementation, we discussed cost

The life of a query: cost estimation

- for each RA implementation, we discussed cost
- the parameters to cost were
  - input relation size (pages and/or tuples), and
  - available buffer space

The life of a query: cost estimation

- for each RA implementation, we discussed cost
- the parameters to cost were
  - input relation size (pages and/or tuples), and
  - available buffer space
- let's consider "quick-and-dirty" heuristics for estimating result size (under the assumption of uniform distribution of values)

Result size estimation: π

For projection, actually computable

Result size estimation: π

For projection, actually computable

- reduction factor is $\frac{\text{new tuple size}}{\text{old tuple size}}$

Result size estimation: π

For projection, actually computable

- reduction factor is $\frac{\text{new tuple size}}{\text{old tuple size}}$
- impacts the number of pages in output

Result size estimation: σ

selection

- let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R

Result size estimation: σ

selection

- let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R
  - by default, we choose $V(R, A) = 10$
    - some systems keep actual counts
  - if there is an index on $R.A$, then $V(R, A)$ is equal to the number of keys

Result size estimation: σ

selection

- let $V(R, A)$ denote the number of distinct values appearing in attribute A of relation R
  - by default, we choose $V(R, A) = 10$
    - some systems keep actual counts
  - if there is an index on $R.A$, then $V(R, A)$ is equal to the number of keys
- then, the size estimate of $\sigma_{A=c}(R)$ is

$$\frac{1}{V(R, A)} T_R$$

Result size estimation: σ

For the size estimate of $\sigma_{A>c}(R)$

- if A is not an arithmetic type, then

$$\frac{1}{3}T_R$$

Result size estimation: σ

For the size estimate of $\sigma_{A>c}(R)$

- if A is not an arithmetic type, then

$$\frac{1}{3} T_R$$

- else

$$\frac{highVal(A) - c}{highVal(A) - lowVal(A)} T_R$$

Result size estimation: σ

- for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

Result size estimation: σ

- for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

- for the size estimate of $\sigma_{C_1 \vee C_2}(R)$

$$\min \{ T_r, \mathit{size}(\sigma_{C_1}(R)) + \mathit{size}(\sigma_{C_2}(R)) \}$$

Result size estimation: σ

- for the size estimate of $\sigma_{A \neq c}(R)$

$$T_R$$

- for the size estimate of $\sigma_{C_1 \vee C_2}(R)$

$$\min \{ T_r, \mathit{size}(\sigma_{C_1}(R)) + \mathit{size}(\sigma_{C_2}(R)) \}$$

- for the size estimate of $\sigma_{C_1 \wedge C_2}(R)$

$$\mathit{rf}_{C_1} \times \mathit{rf}_{C_2} \times T_r$$

Result size estimation: $\cup$

$$|R \cup S| \approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2}$$

Result size estimation: $\cup$

$$
\begin{aligned}
|R \cup S| &\approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2} \\
&= \frac{(2 \times \text{larger}) + \text{smaller}}{2}
\end{aligned}
$$

Result size estimation: $\cup$

$$
\begin{aligned}
|R \cup S| &\approx \frac{(T_R + T_S) + \max\{T_R, T_S\}}{2} \\
&= \frac{(2 \times \text{larger}) + \text{smaller}}{2} \\
&= \text{larger} + \frac{\text{smaller}}{2}
\end{aligned}
$$

Result size estimation: $\cap$

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.

Result size estimation: $\cap$

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.
Hence,

$$|R \cap S| \approx \frac{0 + \min\{T_R, T_S\}}{2}$$

Result size estimation: $\cap$

For intersection $R \cap S$, the minimum result size is 0, and the max size is $\min\{T_R, T_S\}$.
Hence,

$$|R \cap S| \approx \frac{0 + \min\{T_R, T_S\}}{2} = \frac{\text{smaller}}{2}$$

Result size estimation: $-$

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R.

Result size estimation: $-$

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R.
Hence,

$$|R - S| \approx \frac{T_R + (T_R - T_S)}{2}$$

Result size estimation: $-$

For difference $R - S$, the minimum result size is $T_R - T_S$, and the max size is T_R.
Hence,

$$
\begin{aligned}
|R - S| &\approx \frac{T_R + (T_R - T_S)}{2} \\
&= T_R - \frac{1}{2} T_S
\end{aligned}
$$

Result size estimation: $\bowtie$

For natural join $R \bowtie S$ on attribute Y:

$$|R \bowtie S| \;\approx\; \min\left\{ T_R \frac{T_S}{V(S,Y)},\, T_S \frac{T_R}{V(R,Y)} \right\}$$

Result size estimation: $\bowtie$

For natural join $R \bowtie S$ on attribute Y:

$$|R \bowtie S| \approx \min\left\{ T_R \frac{T_S}{V(S,Y)}, T_S \frac{T_R}{V(R,Y)} \right\}$$

since,

- if $V(R,Y) = V(S,Y)$, then each tuple of R joins with approximately $\frac{T_S}{V(S,Y)}$ tuples of R
  - and, each tuple of S joins with approximately $\frac{T_R}{V(R,Y)}$ tuples of S

Result size estimation: $\bowtie$

For natural join $R \bowtie S$ on attribute Y:

$$|R \bowtie S| \;\approx\; \min\left\{ T_R \frac{T_S}{V(S,Y)}, T_S \frac{T_R}{V(R,Y)} \right\}$$

since,

- if $V(R,Y) = V(S,Y)$, then each tuple of R joins with approximately $\frac{T_S}{V(S,Y)}$ tuples of R
    - and, each tuple of S joins with approximately $\frac{T_R}{V(R,Y)}$ tuples of S
- and if not, then we minimize the contribution of dangling tuples

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

- values across columns are not correlated
  - this assumption is hard to lift (active area of research)

Result size estimation: assumptions made

We have made a few assumptions in these estimates:

- values across columns are not correlated
  - this assumption is hard to lift (active area of research)
- values in a single column are uniformly distributed
  - this assumption can be lifted with better statistics

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Provides approximation of value distribution of an attribute in a relation instance

- *small:* typically fit on one disk page
- *accurate:* typically, less than 5% error

Result size estimation: histograms

Histograms: simple data structures for more refined computation of reduction factors

Provides approximation of value distribution of an attribute in a relation instance

- *small:* typically fit on one disk page
- *accurate:* typically, less than 5% error

two basic types:

- equi-width
- equi-depth

Result size estimation: histograms

Equi-width, on column A

- divide range of values appearing in A into equal sized sub-ranges
- compute and store total number of tuples falling into each of these "buckets"

Result size estimation: histograms

Equi-width, on column A

- divide range of values appearing in A into equal sized sub-ranges
- compute and store total number of tuples falling into each of these "buckets"
- to estimate the output cardinality of a range query on A, find starting bucket, and scan forward until ending bucket is identified
- sum number of tuples seen, assuming uniform distribution of values within buckets

Result size estimation: histograms

Equi-depth, on column A

- divide range of values appearing in A into sub-ranges, such that each bucket contains the same number of tuples

Result size estimation: histograms

Equi-depth, on column A

- divide range of values appearing in A into sub-ranges, such that each bucket contains the same number of tuples
- use the same algorithm to approximate the number of tuples falling in a range query over A

Result size estimation: histograms

Example. Consider a "Sales" attribute with the following actual value distribution:

Number of tuples	Sales value
10	0.5 mil
10	1 mil
10	2 mil
5	5 mil
5	7 mil
5	15 mil
2	40 mil
1	50 mil
1	70 mil
1	100 mil

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of "sales $\leq$ 10 million" for equi-width is $\frac{23}{50} = 46\%$ and for equi-depth is $\frac{40}{50} = 80\%$

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of "sales $\leq$ 10 million" for equi-width is $\frac{23}{50} = 46\%$ and for equi-depth is $\frac{40}{50} = 80\%$ which is the actual selectivity!

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of "sales $\leq$ 5 million" for equi-width is $\frac{12}{50} = 24\%$ and for equi-depth is $\frac{35}{50} = 70\%$

Result size estimation: histograms

Suppose we have enough space to store histograms with five buckets

equi-width		equi-depth	
value range	tuple count	value range	tuple count
0-20	45	0-0.5	10
20-40	2	0.5-1	10
40-60	1	1-2	10
60-80	1	2-7	10
80-100	1	7-100	10

The selectivity estimate of "sales $\leq$ 5 million" for equi-width is $\frac{12}{50} = 24\%$ and for equi-depth is $\frac{35}{50} = 70\%$ which is again the actual selectivity!

Result size estimation: histograms

Equi-depth:

- more accurate than equi-width, since buckets with frequently occurring values correspond to smaller ranges, hence giving finer approximations
- effective, simple approach to selectivity estimation, and hence quite common

Result size estimation: histograms

Exercise. Consider a “Friends” attribute with the following actual value distribution:

Friends value	Number of tuples
0	1
1	3
2	6
3	10
4	3
5	2
6	2
7	2
8	1

Construct equi-depth and equi-width histograms over this attribute, using three buckets. What estimates do they give for the count of tuples with “friends $\geq$ 4”?

Wrap up

- Cost estimation, towards choosing a good physical plan

Wrap up

- Cost estimation, towards choosing a good physical plan
- After the break
  - Answering queries using views

Views

Where we've been

cost estimation using data statistics

Where we're headed

query optimization:

- logical optimization
- physical optimization

Where we're headed

query optimization:

- logical optimization
- physical optimization
- next: the creation, maintenance, and use of "views"

Views

Virtual/derived relations, providing alternative logical schemata

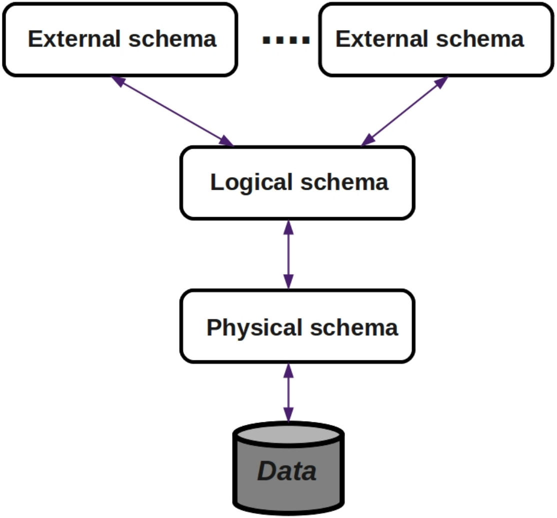
Views

Virtual/derived relations, providing alternative logical schemata

Often desirable to provide

- security
- efficiency
- logical data independence

Data independence



Views

We've already studied a few special cases:

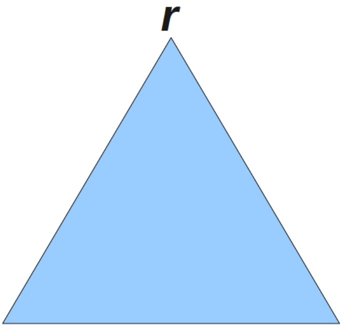
- Histograms/statistics as views

Views

We've already studied a few special cases:

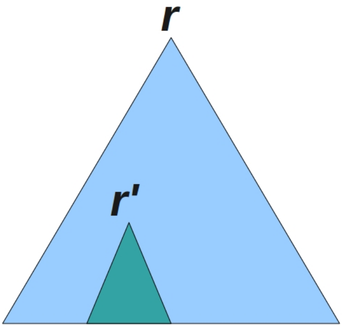
- Histograms/statistics as views
- Indexes as (family of) views

Indexes as views



$predicate(r)$

Indexes as views



$predicate(r)$ vs. $predicate(r')$

Views

We've already studied a few special cases:

- Histograms/statistics as views
- Indexes as (family of) views
  - trees as hierarchical histograms: annotate nodes with actual or approximate counts of items in subtrees

Views

We've already studied a few special cases:

- Histograms/statistics as views
- Indexes as (family of) views
  - trees as hierarchical histograms: annotate nodes with actual or approximate counts of items in subtrees
- GMAPs

Views

In SQL, views are created with the CREATE VIEW statement

- CREATE VIEW *view_name* AS *expression*

Views

In SQL, views are created with the CREATE VIEW statement

- CREATE VIEW *view_name* AS *expression*
  - for example
    CREATE VIEW mng_view AS
    SELECT name, address, phone
    FROM emp
    WHERE title=‘manager’

Views

Basic issues with views:

Views

Basic issues with views:

- creation and implementation
- maintenance under updates
- answering queries using views: query containment

Updating views

Views

$$I_B$$

"Base" instance I_B and view v

Views

$$\begin{tikzcd} I_V \\ I_B \arrow[u, "v"] \end{tikzcd}$$

"Base" instance I_B and view v

Views

- We'd like to allow users to treat the view instance I_V just like any other (base) relation.

Views

- We'd like to allow users to treat the view instance I_V just like any other (base) relation.
- in particular, it would be nice to support both queries and updates on I_V

Updating a view

$$\begin{tikzcd} I_V \arrow[r, "\delta_V"] & I'_V \\ I_B \arrow[u, "v"] & \end{tikzcd}$$

Given update δ_V on view instance I_V, find appropriate update δ_B on the base data I_B

Updating a view

$$
\begin{tikzcd}
I_V \arrow[r, "\delta_V"] & I'_V \\
I_B \arrow[u, "v"] \arrow[r, "\delta_B"] & I'_B \arrow[u, "v"']
\end{tikzcd}
$$

Given update δ_V on view instance I_V, find appropriate update δ_B on the base data I_B

Updating a view

Unfortunately, not all views are directly updateable

- INSERT INTO mng_view
  VALUES (Fred, Eindhoven, 1234567)

Updating a view

Unfortunately, not all views are directly updateable

- INSERT INTO mng_view
  VALUES (Fred, Eindhoven, 1234567)
- how to update the EMP table, if it has a salary field? What should Fred's salary be?

Updating a view

Another example

Suppose we have base table $edge(x, y)$, defining a directed graph, and the view

$$hop(x, y) = edge \bowtie edge$$

(i.e., paths of length 2)

Updating a view

Another example

Suppose we have base table $edge(x, y)$, defining a directed graph, and the view

$$hop(x, y) = edge \bowtie edge$$

(i.e., paths of length 2)

How should we handle an insert on *hop*? How should we materialize this in *edge*?

Updating a view

this is an active research topic!

- Can couple all view definitions with an appropriate "update policy", using some formalism (e.g., so-called lenses)

Updating a view

this is an active research topic!

- Can couple all view definitions with an appropriate "update policy", using some formalism (e.g., so-called lenses)
- DBA can build trigger on view, to enforce "reasonable" behavior

Updating a view

this is an active research topic!

- Can couple all view definitions with an appropriate "update policy", using some formalism (e.g., so-called lenses)
- DBA can build trigger on view, to enforce "reasonable" behavior

```
CREATE TRIGGER mng_trigger
INSTEAD OF INSERT ON mng_view
BEGIN
 INSERT INTO emp VALUES
 (NEW.name, NEW.address, NEW.phone,
  'manager', $0);
END;
```

Updating a view

Can proceed with simple restrictions

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- FROM clause has only one relation,

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- FROM clause has only one relation,
- SELECT clause contains only attributes (no expressions, etc.),

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- FROM clause has only one relation,
- SELECT clause contains only attributes (no expressions, etc.),
- any attribute not listed in SELECT can be set to NULL, and

Updating a view

Can proceed with simple restrictions

A view is *updateable* if

- FROM clause has only one relation,
- SELECT clause contains only attributes (no expressions, etc.),
- any attribute not listed in SELECT can be set to NULL, and
- no GROUP BY or HAVING clauses.

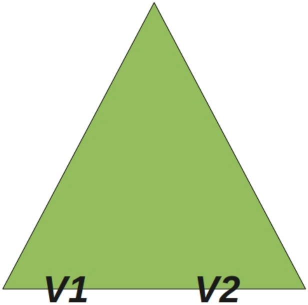
Implementing views

Implementing views

Two alternatives for view implementation

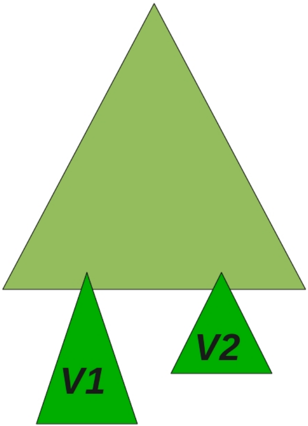
- virtual views: unfold any use of views in a query

Virtual view unfolding



expression parse tree, using views V_1 and V_2

Virtual view unfolding



expression parse tree, with V_1 and V_2 replaced with their definitions

Virtual view unfolding

For example

$$\sigma_{address=Eindhoven}(MngView)$$

Virtual view unfolding

For example

$$\sigma_{address=Eindhoven}(MngView)$$

becomes

$$\sigma_{address=Eindhoven}(\pi_{address,name,phone}(\sigma_{title=Manager}(Emp)))$$

Views

Two alternatives for implementation

- virtual views: unfold any use of views in a query

Views

Two alternatives for implementation

- virtual views: unfold any use of views in a query
- materialization

Materialized views

- precompute and store view results

Materialized views

- precompute and store view results
- more efficient for frequently used and/or expensive views
    - ex: the ATM view of your bank account

Materialized views

- precompute and store view results
- more efficient for frequently used and/or expensive views
  - ex: the ATM view of your bank account
  - ex: view which denormalizes (i.e., incurs costly joins)

Materialized views

Apps

- query optimization

Materialized views

Apps

- query optimization
- data warehousing
    - integration of data
    - OLAP

Materialized views

Apps

- query optimization
- data warehousing
  - integration of data
  - OLAP
- data replication/archiving

Materialized views

Apps

- query optimization
- data warehousing
  - integration of data
  - OLAP
- data replication/archiving
- data visualization

Materialized views

Apps

- query optimization
- data warehousing
  - integration of data
  - OLAP
- data replication/archiving
- data visualization
- caching in networked devices

Materialized views

Apps

- query optimization
- data warehousing
  - integration of data
  - OLAP
- data replication/archiving
- data visualization
- caching in networked devices
- ....

Maintaining materialized views

Materialized views

Issue: we must keep the view up-to-date, as base data evolves ...

View maintenance

$$\begin{tikzcd} I_V & \\ I_B \arrow[u, "v"] \arrow[r, "\delta_B"] & I'_B \end{tikzcd}$$

Given update δ_B on base instance I_B, find appropriate update δ_V on materialized view I_V

View maintenance

$$\begin{tikzcd}
I_V \arrow[r, "\delta_V"] & I'_V \\
I_B \arrow[u, "v"] \arrow[r, "\delta_B"] & I'_B \arrow[u, "v"]
\end{tikzcd}$$

Given update δ_B on base instance I_B, find appropriate update δ_V on materialized view I_V

Materialized views: maintenance

- incremental vs. complete
- immediate vs. deferred

Materialized views: maintenance

- manual code

Materialized views: maintenance

- manual code
- triggers on base relations

Materialized views: maintenance

- manual code
- triggers on base relations

```
CREATE TRIGGER mng_update
INSERT ON emp
BEGIN
  INSERT INTO mng_view VALUES
    (NEW.name, NEW.address, NEW.phone);
END;
```

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- keep track of the number of derivations of a tuple in the view
  - essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- keep track of the number of derivations of a tuple in the view
  - essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query
- calculate update differential δ_V for the view
  - e.g., for view $V = R \bowtie S$, and update δ_R^+ on R, we have $\delta_V = \delta_R^+ \bowtie S$, and

$$V' = V \cup \delta_V$$

Materialized views: maintenance

The Counting Algorithm for incremental maintenance

- keep track of the number of derivations of a tuple in the view
  - essentially, this is the number of duplicates of the tuple in bag-evaluation of the view query
- calculate update differential δ_V for the view
  - e.g., for view $V = R \bowtie S$, and update δ_R^+ on R, we have $\delta_V = \delta_R^+ \bowtie S$, and

$$V' = V \cup \delta_V$$

  - rules for other algebra operators given in our textbook

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
  - how many ways is $hop(a, c)$ derivable?

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
  - how many ways is $hop(a, c)$ derivable?
  - how many ways is $hop(a, e)$ derivable?

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
  - how many ways is $hop(a, c)$ derivable?
  - how many ways is $hop(a, e)$ derivable?
  - now, suppose $\delta^{-}_{edge} = \{(a, b)\}$.
  - then, $\delta^{-}_{hop} = \{(a, c), (a, e)\}$.

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
  - how many ways is $hop(a, c)$ derivable?
  - how many ways is $hop(a, e)$ derivable?
  - now, suppose $\delta^-_{edge} = \{(a, b)\}$.
  - then, $\delta^-_{hop} = \{(a, c), (a, e)\}$.
- we know that after the update, $hop = \{(a, c)\}$.
- how to incrementally apply δ_{hop} to get the correct hop?

Materialized views: maintenance

The Counting Algorithm

- example: for $edge = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$, we have $hop = \{(a, c), (a, e)\}$
  - how many ways is $hop(a, c)$ derivable?
  - how many ways is $hop(a, e)$ derivable?
  - now, suppose $\delta^-_{edge} = \{(a, b)\}$.
  - then, $\delta^-_{hop} = \{(a, c), (a, e)\}$.
- we know that after the update, $hop = \{(a, c)\}$.
- how to incrementally apply δ_{hop} to get the correct hop?
- check the counts of the elements of the view!

Materialized views: maintenance

The Counting Algorithm

- keep track of the number of derivations of a tuple in the view
- calculate update differential δ_V for the view

Materialized views: maintenance

The Counting Algorithm

- keep track of the number of derivations of a tuple in the view
- calculate update differential δ_V for the view
- for insertions (i.e., δ^+), increment counts; for deletions (i.e., δ^-), decrement counts
- if the count of a tuple goes to zero, remove it from the view

Materialized views: maintenance

The Counting Algorithm

- keep track of the number of derivations of a tuple in the view
- calculate update differential δ_V for the view
- for insertions (i.e., δ^+), increment counts; for deletions (i.e., δ^-), decrement counts
- if the count of a tuple goes to zero, remove it from the view

in our example $hop = \{(a, c) : 2, (a, e) : 1\}$, and $\delta_{hop}(hop) = \{(a, c) : 1\}$

Using views

Query containment & equivalence

When is a view useful for a given query?

Query containment & equivalence

When is a view useful for a given query?

- more generally, given two queries Q_1 and Q_2, is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I?

Query containment & equivalence

When is a view useful for a given query?

- more generally, given two queries Q_1 and Q_2, is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I?
- in this case we say Q_1 is contained in Q_2, denoted $Q_1 \subseteq Q_2$

Query containment & equivalence

When is a view useful for a given query?

- more generally, given two queries Q_1 and Q_2, is it the case that $Q_1(I) \subseteq Q_2(I)$, for any database instance I?
- in this case we say Q_1 is contained in Q_2, denoted $Q_1 \subseteq Q_2$
- if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$, then we say Q_1 and Q_2 are equivalent

Query containment & equivalence

- recall the *hop* view, which we now write as

$$hop(x, y) \leftarrow edge(x, z), edge(z, y)$$

  in familiar datalog notation

Query containment & equivalence

- recall the *hop* view, which we now write as

$$hop(x, y) \leftarrow edge(x, z), edge(z, y)$$

  in familiar datalog notation
- next, consider

$$hop'(v, w) \leftarrow edge(v, u), edge(u, w), edge(w, w)$$

Query containment & equivalence

- recall the *hop* view, which we now write as

$$hop(x, y) \leftarrow edge(x, z), edge(z, y)$$

  in familiar datalog notation
- next, consider

$$hop'(v, w) \leftarrow edge(v, u), edge(u, w), edge(w, w)$$

- is it the case that $hop \subseteq hop'$?
- is it the case that $hop' \subseteq hop$?

Query containment & equivalence

- recall the *hop* view, which we now write as

$$hop(x, y) \leftarrow edge(x, z), edge(z, y)$$

  in familiar datalog notation
- next, consider

$$hop'(v, w) \leftarrow edge(v, u), edge(u, w), edge(w, w)$$

- is it the case that $hop \subseteq hop'$?
- is it the case that $hop' \subseteq hop$?
- how can we prove this?
  - we will restrict our discussion now to conjunctive queries (containment is undecidable for FO ...)

Query containment & equivalence

Homomorphisms

- a mapping from the variables of Q_2 to the variables of Q_1, such that
  - the head of Q_2 becomes the head of Q_1
  - each subgoal of Q_2 becomes some subgoal of Q_1

Query containment & equivalence

Homomorphisms

- a mapping from the variables of Q_2 to the variables of Q_1, such that
  - the head of Q_2 becomes the head of Q_1
  - each subgoal of Q_2 becomes some subgoal of Q_1
- it isn't necessary for every subgoal of Q_1 to be mapped onto

Query containment & equivalence

for example, the homomorphism φ defined as

$$
\begin{aligned}
x &\rightarrow v, \\
y &\rightarrow w, \\
z &\rightarrow u
\end{aligned}
$$

Query containment & equivalence

for example, the homomorphism φ defined as

$$\begin{aligned} x &\rightarrow v, \\ y &\rightarrow w, \\ z &\rightarrow u \end{aligned}$$

maps

$$hop(x, y) \leftarrow edge(x, z), edge(z, y)$$

onto

$$hop'(v, w) \leftarrow edge(v, u), edge(u, w), edge(w, w)$$

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof: **(if)** Let $\varphi : Q_2 \to Q_1$ be a homomorphism, and let I be a database. Every tuple $t \in Q_1(I)$ is produced by some substitution ψ_t on the variables of Q_1 that make all of $Q_1's$ subgoals true in I. Then $\psi_t \circ \varphi$ is a substitution for variables of Q_2 such that $t \in Q_2(I)$. Hence, $Q_1 \subseteq Q_2$.

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof, continued: **(only if)** Create a fresh unique atom for each variable of Q_1, and let I_{Q_1} be the database instance consisting of all the subgoals of Q_1, with the chosen atoms substituted for variables. Now, note that $Q_1(I_{Q_1})$ contains the "atom-head" t of Q_1. Since $Q_1 \subseteq Q_2$, it must also be that $t \in Q_2(I_{Q_1})$.

Query containment & equivalence

Theorem

$Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1

proof, continued: Let ψ_t be the substitution of constants from I_{Q_1} for the variables of Q_2 that makes each subgoal of Q_2 a tuple of the instance I_{Q_1} and yields t as the head; and, let φ be the substitution that maps constants of I_{Q_1} to their unique corresponding variable of Q_1. Then $\varphi \circ \psi_t$ is a homomorphism from Q_2 to Q_1. □

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

1. given a mapping m from Q_2 to Q_1, we can check if m is indeed a homomorphism in polynomial time

Query containment & equivalence

unfortunately, checking containment for conjunctive queries is an NP-complete problem

1. given a mapping m from Q_2 to Q_1, we can check if m is indeed a homomorphism in polynomial time
2. Let $G = (V, E)$ be a graph and k be an integer. Consider, for set C of k new distinct variables,

$$\begin{aligned} Q_2 &= out() \leftarrow \bigwedge_{(u,v)\in E} E(u,v) \\ Q_1 &= out() \leftarrow \bigwedge_{u,v\in C, u\neq v} E(u,v) \end{aligned}$$

Then $Q_1 \subseteq Q_2$ iff G has a k-coloring. □

Query containment & equivalence

Fortunately, queries are often quite small, especially with respect to the size of data

Furthermore, checking query containment for acyclic conjunctive queries is *tractable* (i.e., computable in polynomial time). More on this in a later lecture ...

Exercise: answering queries with views

Consider the following conjunctive query.

$$Q: \quad \mathit{result}(A) \quad \leftarrow \quad r(A,B), r(A,C), s(B,D,E), s(B,F,F)$$

Minimize Q. In other words, give a query Q' that (i) has the smallest possible body and (ii) is equivalent to Q. Demonstrate that your query satisfies both of these properties.

Wrap Up

Wrap up

- Cost estimation, towards choosing a good physical plan
- the construction, maintenance, and use of views

Wrap up

- Cost estimation, towards choosing a good physical plan
- the construction, maintenance, and use of views
- Next time
  - putting everything together for query optimization

Wrap up

- Cost estimation, towards choosing a good physical plan
- the construction, maintenance, and use of views
- Next time
  - putting everything together for query optimization
- Reminder: team project report due by Wednesday 13 May

Credits

Ullman 1999