CMPT 321
Fall 2017

SQL queries

review

By *Marina Barsky*

SQL makes these queries simpler

1. Finding min/max
2. At least k
3. Every

Suppliers (<u>sid</u>, sname, address)
Parts (<u>pid</u>, pname, color)
Catalog (<u>sid</u>, <u>pid</u>, cost)
Substitute (<u>pid</u>, <u>subst id</u>)

Finding min/max:
find part(s) with a minimum price

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
SELECT MIN(cost)
FROM Catalog;
```

- It is easy to implement full-table aggregates using a single accumulator variable and scanning the table by comparing value in each row to value of the accumulator

```
SELECT pid FROM catalog
WHERE cost = (SELECT MIN(cost) FROM catalog);
```

- From here it is easy to find part names

Groups and aggregates

Suppliers (<u>sid</u>, sname, address)
Parts (<u>pid</u>, pname, color)
Catalog (<u>sid</u>, <u>pid</u>, cost)
Substitute (<u>pid</u>, <u>subst id</u>)

- Finding average cost for each part

```
SELECT pid, AVERAGE (cost)
FROM catalog
GROUP BY pid;
```

- Finding min cost for each part

```
SELECT pid, MIN (cost)
FROM catalog
GROUP BY pid;
```

- Finding number of different colors for each part

```
SELECT pid, COUNT (color)
FROM parts
GROUP BY pid;
```

At least k:
find part(s) offered in at least 4 colors

Suppliers (<u>sid</u>, sname, address)
Parts (<u>pid</u>, pname, color)
Catalog (<u>sid</u>, <u>pid</u>, cost)
Substitute (<u>pid</u>, <u>subst id</u>)

- Use GROUP BY and HAVING

```
SELECT pid, COUNT (color)
FROM parts
GROUP BY pid
HAVING COUNT (color) >=4;
```

Every color: find parts that are offered in **every** color

- Idea is the same as in Relational Algebra.
- We can use a subquery with NOT EXISTS:

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
CREATE VIEW product AS
SELECT pid, color
FROM
(SELECT pid FROM  parts),
(SELECT DISTINCT color FROM parts)

CREATE VIEW notevery AS
SELECT * FROM product
EXCEPT SELECT pid, color FROM parts;

SELECT pid FROM parts outer
WHERE NOT EXISTS
(SELECT 1 FROM notEvery inner
WHERE inner.pid=outer.pid);
```

Cartesian Product – when no join condition is specified

For each pid, execute subquery and find if the result is empty

5 more queries of interest

1. Top k
2. Expanding self-relationships
3. Above/below average
4. Mode (most frequent value)
5. Custom groups

Top 3: find top 3 suppliers based on the total number of distinct parts offered

- First, create groups with counts
- Then, use ORDER BY and LIMIT

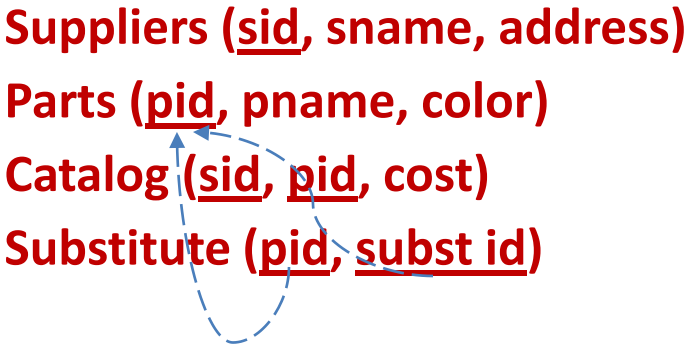
Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
CREATE VIEW supplier_groups AS
SELECT sid, COUNT(pid) part_counts
FROM catalog
GROUP BY sid;

SELECT sid FROM supplier_groups
ORDER BY part_counts DESC
LIMIT 3;
```

Self-relationships: for each part of supplier A, give a substitute (pname, sname) pair: 1/3

```
--find all pids for supplier A
CREATE VIEW sup_A_parts AS
SELECT pid
FROM Catalog NATURAL JOIN Suppliers
WHERE sname = 'A';
```




```
--find all subst part ids for pids in the view above
CREATE VIEW A_subst AS SELECT A.pid, S.subst_id
FROM sup_A_parts A NATURAL JOIN substitute S;
```

A_SUBST	
pid	Subst_id
1	2
1	3
2	1

Self-relationships: for each part of supplier A, give a substitute (pname, sname) pair: 2/3

```
--Expand pids with pname
CREATE VIEW pid_names AS
SELECT pid, pname
FROM parts p JOIN A_SUBST A
ON p.pid = A.pid;

--Expand subst_ids with pname
CREATE VIEW substid_names AS
SELECT subst_id, pname as subst_name
FROM parts p JOIN A_SUBST A
ON p.pid = A.subst_id;
```

Suppliers (sid, sname, address)

Parts (pid, pname, color)

Catalog (sid, pid, cost)

Substitute (pid, subst id)

Pid_names		
pid	Subst_id	pname
1	2	A
1	3	A
2	1	B

Subst_names		
pid	Subst_id	Subst_name
1	2	B
1	3	C
2	1	A

Self-relationships: for each part of supplier A, give a substitute (pname, sname) pair: 2/3

```
--finally, join both to get
---a full list
SELECT pname, subst_name
FROM pid_names p, subst_names s
WHERE p.id = s.id
AND p.subst_id = s.subst_id
ORDER BY pname;
```

Of course this all can be done with a single join, but the main thing is that *pname* has to be renamed for the substitute part name

Suppliers (sid, sname, address)

Parts (pid, pname, color)

Catalog (sid, pid, cost)

Substitute (pid, subst id)

Pid_names		
pid	Subst_id	pname
1	2	A
1	3	A
2	1	B

Subst_names		
pid	Subst_id	Subst_name
1	2	B
1	3	C
2	1	A

Above average: find parts that are charged above their average price

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

- First, for each part compute

its average price:

```
CREATE VIEW average_cost AS
SELECT pid, AVERAGE (cost) as ave
FROM catalog
GROUP BY pid;
```

- Now use correlated subquery to compare each part to the corresponding average:

```
SELECT pid FROM catalog
WHERE cost >
(SELECT ave FROM average_cost
WHERE average_cost.pid = catalog.pid);
```

Computing mode – most frequent value: for each part what is its mode (most frequent) color

- The value for mode should be discrete

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
CREATE VIEW color_counts AS
SELECT  pid,  color,  count(*) as cnt
FROM parts p
GROUP BY pid, color;

SELECT  pid, color as mode
FROM color_counts outer
WHERE cnt =
(SELECT MAX(cnt)
FROM color_counts inner
WHERE outer.pid = inner.pid);
```

Computing mode – using join:
for each part what is its mode (most frequent) color

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
CREATE VIEW color_counts AS
SELECT  pid,  color,  count(*) as cnt
FROM parts
GROUP BY pid, color;


SELECT pid, color as mode
FROM color_counts c1
JOIN
(SELECT pid, MAX(cnt) max_cnt
FROM color_counts c2 GROUP BY pid)
ON c1.pid = c2.pid AND c1.cnt = c2.max_cnt;
```

Custom groups: all combinations – for each supplier count how many parts are in each of 3 cost groups

- When there are no discrete groups, we can create them using CASE

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
SELECT sid,
CASE
 WHEN cost BETWEEN 0 AND 100 THEN 'low_price'
 WHEN cost BETWEEN 100 AND 200 THEN 'ave_price'
 ELSE 'high_price'
END AS price_group,
COUNT(pid) AS num_parts
FROM catalog
GROUP BY sid, price_group
ORDER BY 1,2;
```

Custom groups: new columns– for each supplier count how many parts are in each of 3 cost groups

- We can create a new column

for each group

Suppliers (sid, sname, address)
Parts (pid, pname, color)
Catalog (sid, pid, cost)
Substitute (pid, subst id)

```
SELECT sid,
SUM(CASE WHEN cost BETWEEN 0 AND 100 THEN 1
ELSE 0 END) as low_price,
SUM(CASE WHEN cost BETWEEN 100 AND 200 THEN 1
else 0 END) as ave_price,
SUM(CASE WHEN cost > 200 THEN 1 ELSE 0 END) as
high_price,
FROM catalog
GROUP BY sid;
```