Chapter 6 The database Language SQL –as a tutorial

- **About SQL**

SQL is a standard database language, adopted by many commercial systems.

ANSI SQL, SQL-92 or SQL2, SQL99 or SQL3 extends SQL2 with object-relational features. SQL2003 is the collection of extensions to SQL3.

- **How to query the database**
- How to make modifications on database
- Transactions in SQL

Subqueries

Simplest Case：Returns a Single, Unary Tuple

Find bars that serve Miller at the same price Joe charges for Bud.

Sells(bar, beer, price)

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND price =
          (SELECT price
          FROM Sells
          WHERE bar = 'Joe''s Bar' AND
                    beer = 'Bud');
```

- **Notice the *scoping rule*: an attribute refers to the most closely nested relation with that attribute.**
- **Parentheses around subquery are essential.**

The IN Operator

"Tuple IN relation" is true iff the tuple is in the relation.

Find the name and manufacturer of beers that Fred likes.

Beers(name, manf)
Likes(drinker, beer)

```
SELECT *
FROM Beers
WHERE name IN
    (SELECT beer
    FROM Likes
    WHERE drinker = 'Fred');
```

- Also: NOT IN.

The Exists Operator

- **EXISTS(<relation>) is true if and only if the <relation> is not empty.**

Beers(name, manf),

- Example: find those beers that are the unique beer by their manufacturer.

Example Query with EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS(
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

- A subquery that refers to values from a surrounding query is called a ***correlated subquery***.

The Operator ANY

- *x* = ANY(<relation>) is a boolean condition meaning that *x* equals at least one tuple in the relation.

- Example: *x* >= ANY(<relation>) means *x* is not smaller than all tuples in the relation.
  - Note tuples must have one component only.

The Operator ALL

- **Similarly, *x* <> ALL(<relation>) is true if and only if for every tuple *t* in the relation, *x* is not equal to *t*.**
  - **That is, *x* is not a member of the relation.**
- **Example: *x* >= ALL(<relation>) means there is no tuple larger than *x* in the relation.**

Quantifiers

ANY and ALL behave as existential and universal quantifiers, respectively.

Example

Find the beer(s) sold for the highest price.

Sells(bar, beer, price)

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must not be less than any price.

Conditions Involving Relations

- **EXISTS R: true if and only if R is not empty.**
- **s IN R: true if and only if s is equal to one of the values in R.**
- **s > ALL R: true if and only if s is greater than every value in unary R.**
- **s > ANY R: true if and only if s is greater than at least one value in unary R**

Classroom exercise

```
Q1:  select a from R
     Where b>=
      ANY (select d from S   where c>10);
Q2:  select a from R
     Where b>=
     ALL (select d from S where c>10);
```

a) Q1 and Q2 produce the same answer.

b) The answer to Q1 is contained in the answer to Q2

c) The answer to Q2 is contained in the answer to Q1

d) Q1 and Q2 produce different answers.

Think about when the subquery is empty, what is the result?

Answer:

if the subquery is empty, Q1 is null, Q2 is all the list of R.

Where b>= ANY (empty) is not true,

there does not exist an element than which b are bigger.

Where b>=ALL (empty) is true

Aggregations

Sum, avg, min, max, and count apply to attributes/columns. Also, count(*) applies to tuples.

- Use these in lists following SELECT.

Example

Find the average price of Bud.

Sells(bar, beer, price)

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

- Counts each tuple (presumably each bar that sells Bud) once.

What would we do if Sells were a bag?

Eliminating Duplicates Before Aggregation

Find the number of different prices at which Bud is sold.

Sells (bar, beer, price)

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

- DISTINCT may be used in any aggregation, but typically only makes sense with COUNT.

NULL's Ignored in Aggregation

- **NULL never contributes to a sum, average, or count and can never be the minimum or maximum of a column.**
- **But if there are no non-NULL values in a column, then the result of the aggregation is NULL.**
- Exception: **COUNT of an empty set is 0**.

Examples: About count()

Select count(*)

From Sells

counts the number of tuples in Sells.

Select count(bar)

From Sells

counts the number of values(non-NULL) in the bar column. Duplicates values are not eliminated.

Select count (distinct bar)

From Sells

counts the number of different values in the bar column, no matter how many kinds of beers bars sold.

Example: count(*) vs. count(column)

```
SELECT count(*)
FROM Sells
WHERE beer = ‘Bud’;
```

The number of bars that sell Bud.

```
SELECT count(price)
FROM Sells
WHERE beer = ‘Bud’;
```

The number of bars that sell Bud at a known price.

Grouping

Follow select-from-where by GROUP BY and a list of attributes.

- The relation that is the result of the FROM and WHERE clauses **is grouped** according to the values of these attributes, and **aggregations** take place only **within a group**.

Example

Find the average sales price for each beer.

Sells (bar, beer, price)

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

Example

Find, for each drinker, the average price of Bud at the bars they frequent.

Sells(bar, beer, price)

Frequents(drinker, bar)

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
        Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute drinker-bar-price of Bud tuples first, then group by drinker

- **Note: grouping occurs after the × and σ operations.**

Restriction on SELECT Lists With Aggregation

- **If any aggregation is used, then each element of the SELECT list must be either:**

1. **Aggregated, or**
2. **An attribute on the GROUP BY list.**

Illegal Query Example

- **Find the bar that sells Bud the cheapest by:**

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = ‘Bud’;
```

- **But this query is illegal in SQL.**
  - **Why? Note bar is neither aggregated nor on the GROUP BY list.**

HAVING Clauses

- **HAVING <condition> may follow a GROUP BY clause.**
- **If so, the condition applies to each group, and groups not satisfying the condition are eliminated.**

Requirements on HAVING **Conditions**

- May refer to any relation or tuple-variable in the FROM clause.
- May refer to attributes of those relations, as long as the attribute makes sense within a group; i.e., it is either:
  1. A grouping attribute, or
  2. Aggregated.

Example

Find the average price of those beers that are either served in at least 3 bars or manufactured by Anheuser-Busch.

Beers(name, manf)

Sells(bar, beer, price)

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(*) >= 3 OR
                beer IN (
                        SELECT name
                        FROM Beers
                        WHERE manf = 'Anheuser-Busch’
                        );
```

Rules for having clause

- Anything goes in a subquery.
- Outside subqueries, they may refer to attributes only if they are either:
    - A grouping attribute, or
    - Aggregated

Grouping, Aggregation and Null

- **The value NULL is ignored in any aggregation.**
- **NULL is treated as an ordinary value in a grouped attribute.**

Select a, avg(b) from R

Group by a

Result will be:

R(a,b)

a	avg(b)
2	4
3	9
null	4

Grouping, Aggregation and Null (cont.)

R(a,b)=(null,null)

- **Select a, count(b) from R group by a → ?**
- **Select a, sum(b) from R group by a →?**

Classroom Exercises

Use aggregation function, subqueries to find:

- **the highest grade of each courses.**
- **how many students failed for each course**

Subqueries in From and Where clause.

Query： the highest grade of each courses.

- select cid, max(grade) from sc where grade is not null group by cid;

- select cid,grade

```
from sc C1
where C1.grade is not NULL and not exists  (select * from sc C2
            where C2.grade > C1.grade and C2.cid= C1.cid);
```

How many students failed in the examination?

```
select cid,count(*) as numberOffailed
from sc
where sc.grade <60
group by cid;
```

Subquery can be in a From clause

Choose the highest grade of each courses, the highest grade is greater than 80.

```
 select *
 from  (select cid,max(grade)
as X from sc group by cid ) G
  where G.X>80;
```