Restriction on SELECT Lists With Aggregation

select
frome
where ...
Group by

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or s.t. Avg(price)
  2. An attribute on the GROUP BY list.

Gr. w/ one value

~~beer~~

a) select beer, avg(price)
from sells
group by beer;

bar?

Group

beer avg(..)

b) select avg(price)
from sells
where beer='bud'

⇒ 1 group ~~[illegible]~~ gr. attr.

Q: How about this query?

SQL

```
~~SELECT bar, MIN(price)~~
~~FROM Sells~~
~~WHERE beer = 'Bud';~~
```

one group

$\overset{\text{No}}{\Longrightarrow}$ gr. attr

$\Rightarrow$ (bar with min price, the price)

~~select bar, avg(price)~~
~~From sells~~

sel name, avg(GPA)
~ From Stu.

Q: How to do it right, then?

SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';

ind. only combined stuff

⇒ price < min (price)

Q: How to do it right?

① where price <= all (select price from sells ...)

② (select min(price) from ...)

MySQL impl is problematic!

x

- Select bar, avg(price) from Sells

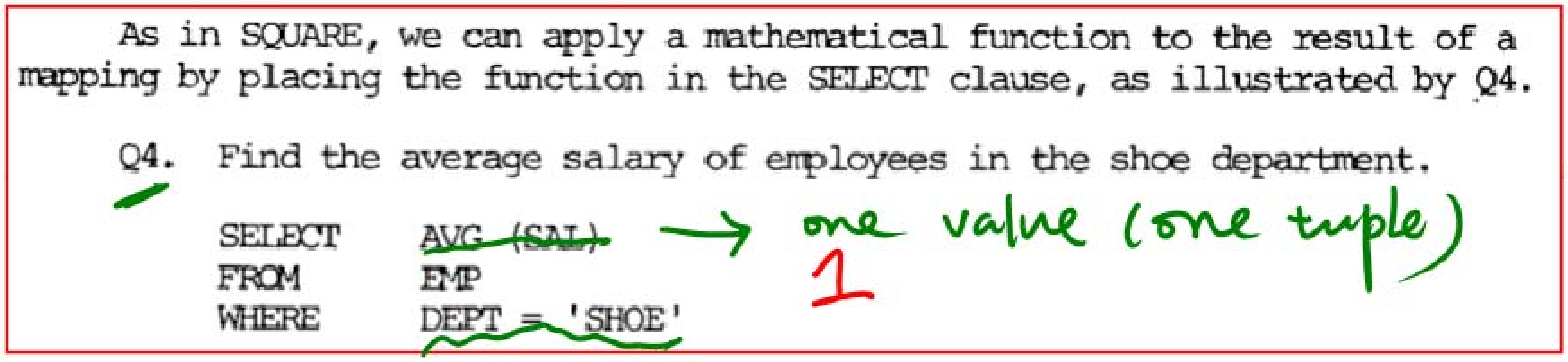
→ this is ~~NO~~T OK.

x

- Select bar, avg(price) from Sells Group By beer

→ this is OK!

Behind the Scene: First Aggregate Query



As in SQUARE, we can apply a mathematical function to the result of a mapping by placing the function in the SELECT clause, as illustrated by Q4.

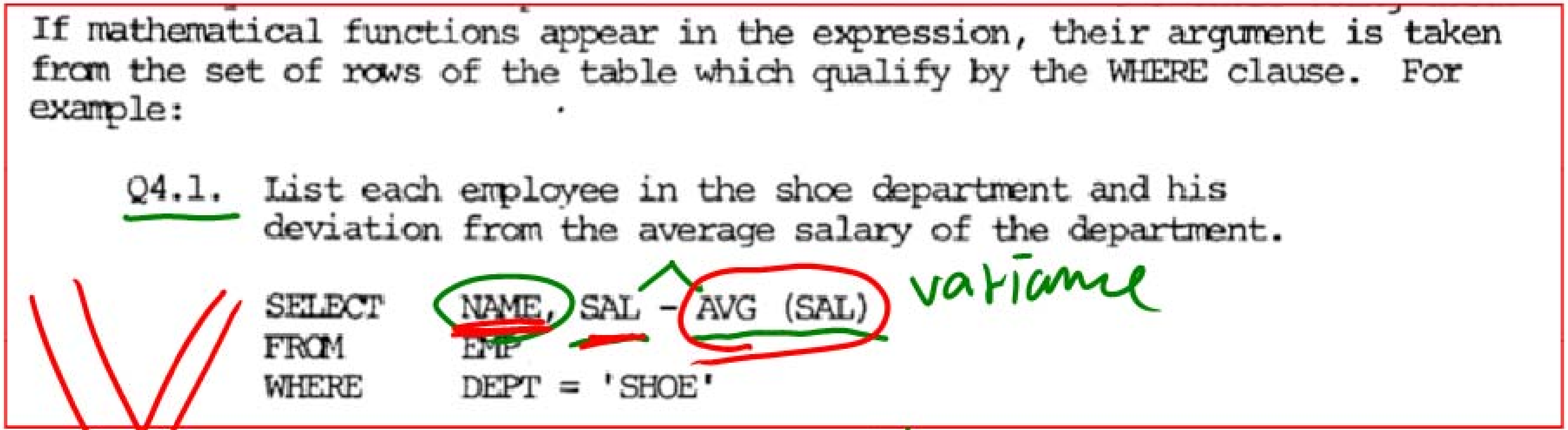
Q4. Find the average salary of employees in the shoe department.

```
SELECT    AVG (SAL)
FROM      EMP
WHERE     DEPT = 'SHOE'
```

→ one value (one tuple)

1

Behind the Scene: How about this one?



If mathematical functions appear in the expression, their argument is taken from the set of rows of the table which qualify by the WHERE clause. For example:

Q4.1. List each employee in the shoe department and his deviation from the average salary of the department.

```
SELECT    NAME, SAL - AVG (SAL)
FROM      EMP
WHERE     DEPT = 'SHOE'
```

variance

10 { John +$200
Alex -$2K
Mary
⋮ }

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.

Select
From
where
Group By
Having : count(*) ≥ 10

Requirements on HAVING Conditions

select
from Sells.
where
group by beers, bar

- These conditions may refer to any relation or tuple-variable in the FROM clause.
- They 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 → beers, bar
  2. Aggregated. → count(*), avg(price)

Example

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR beer = ‘michelob’;
```

General form of Grouping and Aggregation

```
SELECT     S
FROM       R_1,…,R_n
WHERE      C1          (subquery)
GROUP BY   a_1,…,a_k
HAVING     C2
```

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions or grouping attributes

General form of Grouping and Aggregation

```
SELECT    S
FROM      R_1,…,R_n
WHERE     C1
GROUP BY  a_1,…,a_k
HAVING    C2
```

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

$\boxed{\text{Write}}$

CS411
Database Systems

06c: SQL-3

DB Modification, Definition, Views

Why Do We Learn This?

Data is dynamic,

→ capture update,

Database Modification

Database Modifications

- A modification command does not return a result as a query does, but it changes the database in some way.
- There are three kinds of modifications:

1. *Insert* a tuple or tuples.
2. *Delete* a tuple or tuples.
3. *Update* the value(s) of an existing tuple or tuples.

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

```
INSERT INTO Likes
VALUES('Sally', 'Bud');
```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- There are two reasons to do so:

1. We forget the standard order of attributes for the relation.
2. We don’t have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>

  (<subquery>);

E.g., INSERT INTO Beers(name) values (..)

SELECT beer from Sells;

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally’s “potential buddies,” i.e., those drinkers who frequent at least one bar that Sally also frequents.

Solution

The other drinker

Pairs of Drinker tuples where the first is for Sally, the second is for someone else, and the bars are the same.

```
INSERT INTO PotBuddies
(SELECT d2.drinker
FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND
  d2.drinker <> 'Sally' AND
  d1.bar = d2.bar
);
```

Deletion

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
WHERE <condition>;
```

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = ‘Sally’ AND
      beer = ‘Bud’;
```

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Drop table Likes;

wri-es

Example: Delete Many Tuples

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b
WHERE EXISTS (
    SELECT name FROM Beers
    WHERE manf = b.manf AND
        name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Semantics of Deletion -- 1

- Suppose Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, When *b* is the tuple for Bud Lite, do we delete that tuple too?

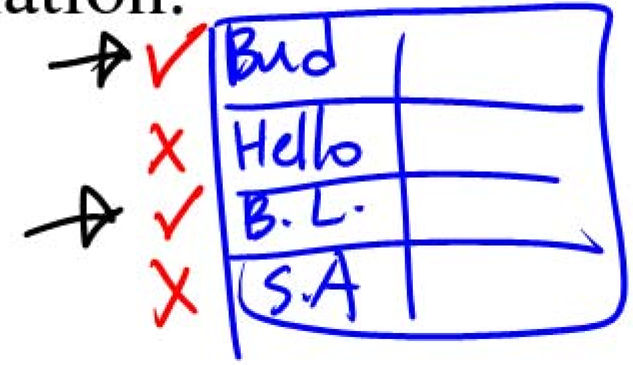
Semantics of Deletion -- 2

- The answer is that we *do* delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:

Eval Where before execut any change

1. Mark all tuples for which the WHERE condition is satisfied in the original relation.
2. Delete the marked tuples.

Actually do change



Updates

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

How change

what tuples to change

Example: Update

- Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

Example: Update Several Tuples

- Increase price that is cheap:

```
UPDATE Sells
SET price = price * 1.07
WHERE price < 3.0;
```

Defining a Database Schema

create (already!)

Views

Views

- A view is a "virtual table," a relation that is defined in terms of the contents of other tables and views.
- Declare by:

  CREATE VIEW <name> AS <query>;

  ANY return table

  select
  from stud.
  wher class='1341'
- In contrast, a relation whose value is really stored in the database is called a *base table*.

Example: View Definition

- CanDrink(drinker, beer) is a view "containing" the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

Example: Accessing a View

- You may query a view as if it were a base table.
  - There is a limited ability to modify views if the modification makes sense as a modification of the underlying base table.
- Example:

```
SELECT beer FROM CanDrink
WHERE drinker = ‘Sally’;
```