

Schedule

- Today: Jan. 24 (TH)
 - ◆ Subqueries, Grouping and Aggregation.
 - ◆ Read Sections 6.3-6.4. Project Part 2 due.
- Jan. 29 (T)
 - ◆ Modifications, Schemas, Views.
 - ◆ Read Sections 6.5-6.7. Assignment 3 due.
- Jan. 31 (TH)
 - ◆ Constraints.
 - ◆ Read Sections 7.1-7.3, 7.4.1. Project Part 3 due.
- Feb. 5 (T)
 - ◆ Triggers, PL/SQL.
 - ◆ Read Sections 7.4, 8.2. Assignment 4 due.
- **Reminder: Midterm is Feb. 14 (TH)**

Union, Intersection, Difference

- “(subquery) UNION (subquery)” produces the union of the two relations.
- Similarly for INTERSECT, EXCEPT = intersection and set difference.
 - ◆ But: in Oracle set difference is MINUS, not EXCEPT.

Example

Find the drinkers and beers such that the drinker likes the beer and frequents a bar that serves it.

```
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

(SELECT * FROM Likes)
  INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```

Forcing Set/Bag Semantics

- Default for select-from-where is bag; default for union, intersection, and difference is set.
 - ◆ Why? Saves time of not comparing tuples as we generate them.
 - ◆ But we need to sort anyway when we take intersection or difference. (Union seems to be thrown in for good measure!)
- Force set semantics with `DISTINCT` after `SELECT`.
 - ◆ But make sure the extra time is worth it.

Example

Find the different prices charged for beers.

Sells(bar, beer, price)

SELECT DISTINCT price

FROM Sells;

- Force bag semantics with ALL after UNION, etc.

Join-Based Expressions

A number of forms are provided.

- Can be used either stand-alone (in place of a select-from-where) or to define a relation in the FROM-clause.

R NATURAL JOIN *S*

R JOIN *S* ON condition

e.g., condition: *R.B=S.B*

R CROSS JOIN *S*

R OUTER JOIN *S*

- Outerjoin can be modified by:

1. Optional NATURAL in front.
2. Optional ON condition at end.
3. Optional LEFT, RIGHT, or FULL (default) before OUTER.
 - ◆ LEFT = pad (with NULL) dangling tuples of *R* only; RIGHT = pad dangling tuples of *S* only.

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.

Class Problem

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.

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)
Frequent(drinker, bar)

SELECT drinker, AVG(price)
FROM Frequent, Sells
WHERE beer = 'Bud' AND

      Frequent.bar = Sells.bar
GROUP BY drinker;
```

- Note: grouping occurs after the $\times$ and σ operations.

Restriction on SELECT Lists With Aggregation

If any aggregation is used, then *each* element of a SELECT clause must either be aggregated or appear in a group-by clause.

Example

- The following might seem a tempting way to find the bar that sells Bud the cheapest:

```
Sells(bar, beer, price)
```

```
SELECT bar, MIN(price)
```

```
FROM Sells
```

```
WHERE beer = 'Bud' ;
```

- **But it is illegal in SQL.**

Problem

How would we find that bar?

HAVING Clauses

HAVING clauses are selections on groups, just as WHERE clauses are selections on tuples.

- Condition can use the tuple variables or relations in the FROM and their attributes, just like the WHERE can.
- ◆ But the tuple variables range only over the group.
- ◆ And the attribute better make sense within a group; *i.e.*, be one of the grouping attributes.

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'
      );
```

DB Modifications

- *Modification* = insert + delete + update.

Insertion of a Tuple

INSERT INTO relation VALUES (list of values).

- Inserts the tuple = list of values, associating values with attributes in the order the attributes were declared.
 - ◆ Forget the order? List the attributes as arguments of the relation.

Example

Likes(drinker, beer)

Insert the fact that Sally likes Bud.

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

Insertion of the Result of a Query

INSERT INTO relation (subquery).

Example

Create a (unary) table of all Sally's potential buddies, i.e., the people who frequent bars that Sally also frequents.

Frequents(drinker, bar)

```
CREATE TABLE PotBuddies(  
    name char(30)  
);
```

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

Deletion

DELETE FROM relation WHERE condition.

- Deletes all tuples satisfying the condition from the named relation.

Example

Sally no longer likes Bud.

Likes(drinker, beer)

DELETE FROM Likes

WHERE drinker = 'Sally' AND
beer = 'Bud' ;

Example

Make the Likes relation empty.

DELETE FROM Likes;

Example

- Delete all beers for which there is another beer by the same manufacturer.

```
Beers(name, manf)
```

```
DELETE FROM Beers b
```

```
WHERE EXISTS
```

```
  (SELECT name
```

```
    FROM Beers
```

```
   WHERE manf = b.manf AND
```

```
        name <> b.name
```

```
);
```

- Note alias for relation from which deletion occurs.

- Semantics is tricky. If A.B. makes Bud and BudLite (only), does deletion of Bud make BudLite *not* satisfy the condition?
- SQL semantics: all conditions in modifications must be evaluated by the system before any mods due to that mod command occur.
- ◆ In Bud/Budlite example, we would first identify both beers a targets, and then delete both.

Updates

UPDATE relation SET list of assignments WHERE condition.

Example

Drinker Fred's phone number is 555-1212.

Drinkers(name, addr, phone)

```
UPDATE Drinkers
```

```
SET phone = '555-1212'
```

```
WHERE name = 'Fred';
```

Example

Make \$4 the maximum price for beer.

- Updates many tuples at once.

Sells(bar, beer, price)

```
UPDATE Sells
```

```
SET price = 4.00
```

```
WHERE price > 4.00;
```