

More SQL

Extended Relational Algebra
Outerjoins, Grouping/Aggregation

The Extended Algebra

δ = eliminate duplicates from bags.

τ = sort tuples.

γ = grouping and aggregation.

Outerjoin : avoids “dangling tuples” = tuples that do not join with anything.

Duplicate Elimination

- ◆ $R1 := \delta(R2)$.
- ◆ R1 consists of one copy of each tuple that appears in R2 one or more times.

Example: Duplicate Elimination

$R =$

A	B
1	2
3	4
1	2

$\delta(R) =$

A	B
1	2
3	4

Sorting

- ◆ $R1 := \tau_L (R2)$.
 - ◆ L is a list of some of the attributes of $R2$.
- ◆ $R1$ is the list of tuples of $R2$ sorted first on the value of the first attribute on L , then on the second attribute of L , and so on.
 - ◆ Break ties arbitrarily.
- ◆ τ is the only operator whose result is neither a set nor a bag.

Example: Sorting

$R =$

A	B
1	2
3	4
5	2

$$\tau_B(R) = [(5,2), (1,2), (3,4)]$$

Aggregation Operators

- ◆ Aggregation operators are not operators of relational algebra.
- ◆ Rather, they apply to entire columns of a table and produce a single result.
- ◆ The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R = (

A	B
1	3
3	4
3	2

)

$$\text{SUM}(A) = 7$$

$$\text{COUNT}(A) = 3$$

$$\text{MAX}(B) = 4$$

$$\text{AVG}(B) = 3$$

Grouping Operator

- ◆ $R1 := \gamma_L (R2)$. L is a list of elements that are either:
 1. Individual (*grouping*) attributes.
 2. $AGG(A)$, where AGG is one of the aggregation operators and A is an attribute.
 - An arrow and a new attribute name renames the component.

Applying $\gamma_L(R)$

- ◆ Group R according to all the grouping attributes on list L .
 - ◆ That is: form one group for each distinct list of values for those attributes in R .
- ◆ Within each group, compute $AGG(A)$ for each aggregation on list L .
- ◆ Result has one tuple for each group:
 1. The grouping attributes and
 2. Their group's aggregations.

Example: Grouping/Aggregation

$R =$ (

A	B	C
1	2	3
4	5	6
1	2	5

)

Then, average C
within groups:

A	B	X
1	2	4
4	5	6

$\Upsilon_{A,B,AVG(C)->X} (R) = ??$

First, group R by A and B :

A	B	C
1	2	3
1	2	5
4	5	6

Outerjoin

- ◆ Suppose we join $R \bowtie_C S$.
- ◆ A tuple of R that has no tuple of S with which it joins is said to be *dangling*.
 - ◆ Similarly for a tuple of S .
- ◆ Outerjoin preserves dangling tuples by padding them NULL.

Example: Outerjoin

R = (

A	B
1	2
4	5

S = (

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTER JOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Now --- Back to SQL

Each Operation Has a SQL
Equivalent

Outerjoins

- ◆ R OUTER JOIN S is the core of an outerjoin expression. It is modified by:
 1. Optional NATURAL in front of OUTER.
 2. Optional ON <condition> after JOIN.
 3. Optional LEFT, RIGHT, or FULL before OUTER.
 - ◆ LEFT = pad dangling tuples of R only.
 - ◆ RIGHT = pad dangling tuples of S only.
 - ◆ FULL = pad both; this choice is the default.
- Only one of these

Aggregations

- ◆ SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- ◆ Also, COUNT(*) counts the number of tuples.

Example: Aggregation

- ◆ From `Sells(bar, beer, price)`, find the average price of Bud:

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

Eliminating Duplicates in an Aggregation

- ◆ Use DISTINCT inside an aggregation.
- ◆ **Example:** find the number of *different* prices charged for Bud:

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

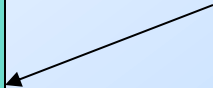
NULL's Ignored in Aggregation

- ◆ 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.

Example: Effect of NULL's

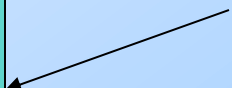
```
SELECT count(*)  
FROM Sells  
WHERE beer = 'Bud';
```

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

- ◆ We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- ◆ The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

Example: Grouping

- ◆ From `Sells(bar, beer, price)`, find the average price for each beer:

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

beer	AVG(price)
Bud	2.33
...	...

Example: Grouping

- ◆ From `Sells(bar, beer, price)` and `Frequents(drinker, bar)`, find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
```

```
FROM Frequents, Sells  
WHERE beer = 'Bud' AND  
      Frequents.bar = Sells.bar
```

```
GROUP BY drinker;
```

Compute all drinker-bar-price triples for Bud.

Then group them by drinker.

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

- ◆ You might think you could 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.

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.


Example: HAVING

- ◆ From `Sells(bar, beer, price)` and `Beers(name, manf)`, find the average price of those beers that are either served in at least three bars or are manufactured by Pete's.

Solution

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

Beer groups with at least 3 non-NULL bars and also beer groups where the manufacturer is Pete's.




```
HAVING COUNT(bar) >= 3 OR
```

```
beer IN (SELECT name
```

```
FROM Beers
```

```
WHERE manf = 'Pete"s');
```

Beers manufactured by Pete's.



Requirements on HAVING Conditions

- ◆ Anything goes in a subquery.
- ◆ Outside subqueries, they may refer to attributes only if they are either:
 1. A grouping attribute, or
 2. Aggregated(same condition as for SELECT clauses with aggregation).