More Relational Algebra
More SQL
More cowbell

Extended Relational Algebra
Outerjoins,
Grouping/Aggregation
Insert/Delete/Update

Skip for skills
Section 6.3.6+ in the Textbook.

The Extended Algebra
\( \delta = \) eliminate duplicates from bags.
\( \tau = \) sort tuples.
\( \gamma = \) grouping and aggregation.
Outerjoin: avoids “dangling tuples” = tuples that do not join with anything.

Duplicate Elimination
\( R_1 := \delta(R_2). \)
\( R_1 \) consists of one copy of each tuple that appears in \( R_2 \) one or more times.

Example: Duplicate Elimination
\[
R = \begin{bmatrix}
1 & 2 \\
3 & 4 \\
1 & 2
\end{bmatrix}
\]
\( \delta(R) = \begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}\)

Sorting
\( R_1 := \tau_L(R_2). \)
\( L \) is a list of some of the attributes of \( R_2 \).
\( R_1 \) is the list of tuples of \( R_2 \) sorted first on the value of the first attribute on \( L \), then on the second attribute of \( L \), and so on.
Break ties arbitrarily.
\( \tau \) is the only operator whose result is neither a set nor a bag.
**Example: Sorting**

\[ R = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 2 \end{pmatrix} \]

\[ \tau_s (R) = \{(5,2), (1,2), (3,4)\} \]

**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**

\[ R = \begin{pmatrix} A & B \\ 1 & 3 \\ 3 & 2 \end{pmatrix} \]

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
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**

First, group \( R \) by \( A \) and \( B \):

\[ \begin{array}{ccc} A & B & \gamma_A, \gamma_B \rightarrow X \\ 1 & 2 & 4 \\ 4 & 5 & 6 \\ 1 & 2 & 5 \end{array} \]

Then, average \( C \) within groups:

\[ \begin{array}{ccc} A & B & X \\ 1 & 2 & 4 \\ 4 & 5 & 6 \end{array} \]
Suppose we join $R \bowtie S$.

- 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

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

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

$R \text{ OUTERJOIN } S =$

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>6</td>
<td>NULL</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Outerjoins

- $R \text{ OUTER JOIN } S$ 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.

Example: Aggregation

- From $\text{Sells(bar, beer, price)}$, find the average price of Bud:
  
  ```sql
  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:
  
  ```sql
  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

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

The number of bars that sell Bud.

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

- From `Sells(bar, beer, price)` and `Frequents(drinker, bar)`, find for each drinker the average price of Bud at the bars they frequent:
  ```sql
  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:
  ```sql
  SELECT bar, MIN(price)
  FROM Sells
  WHERE beer = 'Bud';
  ```
- But this query is illegal in SQL.

GRAB A LAPTOP

- Log into storm
- Grab your book
- Turn to page 52, Exercises 2.4.14, #2.4.1
  - Today is "Exam 1 prep."
  - You should be savvy with Ex. 2.4.1 BOTH Rel. Alg. AND SQL.
- Take your Rel. Alg. Solutions to 2.4.1 and deploy SQL solutions to each on storm.cs.uni.edu.
  - Working in groups is encouraged.
  - Queries collected at the end of class for in-class points.

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

```sql
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR beer IN (SELECT name
FROM Beers
WHERE manf = '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).

Database Modifications

◆ A modification command does not return a result (as a query does), but changes the database in some way.
◆ 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.
◆ 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’);

Adding Default Values

◆ In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
◆ When an inserted tuple has no value for that attribute, the default will be used.
Example: Default Values

```
CREATE TABLE Drinkers (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50) DEFAULT '123 Sesame St.',
    phone CHAR(16)
);
```

Example: Default Values

```
INSERT INTO Drinkers(name)
VALUES('Sally');
Resulting tuple:

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>123 Sesame St</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

Inserting Many Tuples

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

```
INSERT INTO <relation> ( <subquery> );
```

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

```
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.

### Example: Delete Some 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 Anheuser-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)

- **Answer:** we do delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied.
  2. Delete the marked tuples.

### Updates

- To change certain attributes in certain tuples of a relation:
  
  ```
  UPDATE <relation>
  SET <list of attribute assignments>
  WHERE <condition on tuples>;
  ```
Example: Update

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

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

Example: Update Several Tuples

- Make $4 the maximum price for beer:

  ```sql
  UPDATE Sells
  SET price = 4.00
  WHERE price > 4.00;
  ```