Announcements

• HW 1 is due today (23:59PM)
• HW 2 has been “published”
  • may take some time to receive an invitation.
• No TA office hours tomorrow (school is closed)
• Project Track 1 and 2 Stage 0 is due this Thursday.
Last Week!

• Single- and multi-relation SQL queries
• Advanced SQL
  • Subqueries
  • Set operations
  • Aggregation
• Null Values in SQL
Today’s Lecture

- Inner and Outer Joins
- Updating Instance
- Updating Schema
- Views

Based on assigned reading:
- Textbook, Ch 8.1-8.2
- Supplemental reading: "SQL for Web Nerds” Ch. 7 (Views)
### Example Database

**Employee table**

<table>
<thead>
<tr>
<th>LastName</th>
<th>DepartmentID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>John</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Department table**

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>DepartmentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Sales</td>
</tr>
<tr>
<td>33</td>
<td>Engineering</td>
</tr>
<tr>
<td>34</td>
<td>Clerical</td>
</tr>
<tr>
<td>35</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

© 2018 A. Alawini & A. Parameswaran
### Cross Product

**Department** × **Employee**

#### SELECT * FROM Department, Employee

<table>
<thead>
<tr>
<th>Department table</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DepartmentID</td>
<td>DepartmentName</td>
</tr>
<tr>
<td>31</td>
<td>Sales</td>
</tr>
<tr>
<td>33</td>
<td>Engineering</td>
</tr>
<tr>
<td>34</td>
<td>Clerical</td>
</tr>
<tr>
<td>35</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee table</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>DepartmentID</td>
</tr>
<tr>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>John</td>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department.DepartmentName</th>
<th>Department.DepartmentID</th>
<th>Employee.LastName</th>
<th>Employee.DepartmentID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>31</td>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Sales</td>
<td>31</td>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Sales</td>
<td>31</td>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Sales</td>
<td>31</td>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>Sales</td>
<td>31</td>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Sales</td>
<td>31</td>
<td>John</td>
<td>NULL</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Engineering</td>
<td>33</td>
<td>John</td>
<td>NULL</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Clerical</td>
<td>34</td>
<td>John</td>
<td>NULL</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>Rafferty</td>
<td>31</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>Jones</td>
<td>33</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>Steinberg</td>
<td>33</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>Smith</td>
<td>34</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>Robinson</td>
<td>34</td>
</tr>
<tr>
<td>Marketing</td>
<td>35</td>
<td>John</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Equijoin

**Employee** ✖️ **Department**

Employee.DeptID = Department.DeptID

```
SELECT *
FROM Employee emp JOIN Department dept
ON emp.DeptID = dept.DeptID
```
Natural Join
Employee \(\bowtie\) Department

```sql
SELECT *
FROM Employee emp NATURAL JOIN Department dept
```

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Employee.LastName</th>
<th>Department.DepartmentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>Smith</td>
<td>Clerical</td>
</tr>
<tr>
<td>33</td>
<td>Jones</td>
<td>Engineering</td>
</tr>
<tr>
<td>34</td>
<td>Robinson</td>
<td>Clerical</td>
</tr>
<tr>
<td>33</td>
<td>Steinberg</td>
<td>Engineering</td>
</tr>
<tr>
<td>31</td>
<td>Rafferty</td>
<td>Sales</td>
</tr>
</tbody>
</table>
Nulls and Joins

• Sometimes need special variations of joins:
  • I want to see all employees and their departments
  • ... But what if there’s a department with no employees?
  • Or what if an employee has not been assigned to a department?
• Outer join:
  • Most common is *left outer join*
Outer Joins

• Left outer join:
  • Include the left tuple even if there’s no match

• Right outer join:
  • Include the right tuple even if there’s no match

• Full outer join:
  • Include both the left and right tuples even if there’s no match
Left Outer Join

**Employee** ⋈ **Department**

\[ \text{Employee}.\text{DeptID} = \text{Department}.\text{DeptID} \]

**SELECT** *
**FROM** Employee emp **LEFT OUTER JOIN** Department dept **ON** emp.DeptID = dept.DeptID
Right Outer Join

Employee $\bowtie$ Department
Employee.DeptID = Department.DeptID

```
SELECT *
FROM Employee emp RIGHT OUTER JOIN Department dept
ON emp.DeptID = dept.DeptID
```
### Full Outer Join

**Employee** $\bowtie$ **Department**

Employee.DeptID = Department.DeptID

**SQL Query:**

```sql
SELECT *
FROM Employee FULL OUTER JOIN Department dept
ON emp.DeptID = dept.DeptID
```

<table>
<thead>
<tr>
<th>Employee.Last_Name</th>
<th>Employee.DeptID</th>
<th>Department.DepartmentName</th>
<th>Department.DeptID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>34</td>
<td>Clerical</td>
<td>34</td>
</tr>
<tr>
<td>Jones</td>
<td>33</td>
<td>Engineering</td>
<td>33</td>
</tr>
<tr>
<td>Robinson</td>
<td>34</td>
<td>Clerical</td>
<td>34</td>
</tr>
<tr>
<td>John</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Steinberg</td>
<td>33</td>
<td>Engineering</td>
<td>33</td>
</tr>
<tr>
<td>Rafferty</td>
<td>31</td>
<td>Sales</td>
<td>31</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>Marketing</td>
<td>35</td>
</tr>
</tbody>
</table>
Outline

- Inner and Outer Joins
  - Updating Instance
  - Updating Schema
  - Views
So far, we’ve been talking about queries...

Now onto Database Modification
Database Modifications

• A modification command does not return a result the way a query does, but it changes the database in some way.

• There are two kinds of database modifications:

  1. Instance Modifications
     • Insert a tuple or tuples.
     • Delete a tuple or tuples.
     • Update the value(s) of an existing tuple or tuples.

  2. Schema Modifications
     • Add/Drop/Modify column
     • Drop Table/View
Insertion

• To insert a single tuple:
  INSERT INTO <relation>
  VALUES ( <list of values> );

• Example: add to Likes (customer, drink) the fact that Sally likes Latte.
  INSERT INTO Likes
  VALUES(‘Sally’, ‘Latte’);
Specifying Attributes in Insert

\[
\text{INSERT INTO Likes} \\
\text{VALUES (‘Sally’, ‘Latte’);} \\
\]

• BUT: this assumes that we remember the order of attributes of Likes

• Instead: we can be explicit: to insert Sally into Likes (customer, drink):

\[
\text{INSERT INTO Likes(drink, customer)} \\
\text{VALUES (‘Latte’, ‘Sally’);} \\
\]

• Can also add multiple tuples separated by commas
Specifying Attributes in INSERT

Overall, two reasons to specify attributes in the INSERT statement:

1. We may have forgotten 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.

    simply omit the ones you don’t want to insert
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 Drinks(name)
    SELECT drink FROM Sells;
```
Example: Insert a Subquery

• Using Frequent(customer, cafe), enter into the new relation PotBuddies(name) all of Sally’s “potential buddies,” i.e., those customers who frequent at least one cafe that Sally also frequents.
Deletion

• To delete tuples satisfying a condition from some relation:
  
  DELETE FROM <relation>
  WHERE <condition>;

Example: Deletion

- Delete from Likes (customer, drink) the fact that Sally likes Latte:

  ```sql
  DELETE FROM Likes
  WHERE customer = 'Sally' AND drink = 'Latte';
  ```
Example: Delete all Tuples

• Make the relation Likes empty:

    DELETE FROM Likes;

• Note no WHERE clause needed.
• Table is not deleted: use the DROP TABLE statement instead
Example: Delete Many Tuples

- Delete from Drinks(name, manf) all drinks manufactured by Starbucks

```
DELETE FROM Drinks
WHERE name = 'Starbucks';
```
Another Example: Delete Many Tuples

- Delete from Drinks (name, manf) all drinks for which there is another drink by the same manufacturer.
Semantics of Deletion -- 1

DELETE FROM Drinks
WHERE name  IN (
    SELECT b1.name
    FROM Drinks b1, Drinks b2
    WHERE b1.manf = b2.manf AND
    b1.name <> b2.name);

<table>
<thead>
<tr>
<th>Name</th>
<th>Manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latte</td>
<td>Seattle's Best Coffee</td>
</tr>
<tr>
<td>Mocha</td>
<td>Seattle's Best Coffee</td>
</tr>
</tbody>
</table>

How would we end up with each case?
Semantics of Deletion -- 2

• Semantics: we *delete both*.
• The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied in the original relation.
  2. Delete the marked tuples.

This makes the deletion process deterministic
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 Fred’s phone number to 555-1212:
  UPDATE Customer
  SET phone = '555-1212'
  WHERE name = 'Fred';

• Add area code ‘217’ to Fred’s phone number:
  UPDATE Customer
  SET phone = '(217)' || phone
  WHERE name = 'Fred';
Example: Update Several Tuples

• Increase price that is cheap:
  
  UPDATE Sells
  SET price = price * 2
  WHERE price < 3.0;

  This could have the same issue as the deletion problem: why?

  How would we solve it?
What about Multiple Users?

• What happens if multiple users update + delete at the same time?

• This requires management of concurrent operations

• We’ll talk about **concurrency control** later in the term
Outline

✓ Inner and Outer Joins
✓ Updating Instance
  • Updating Schema
  • Views
ALTER TABLE Command

Add column

```
ALTER TABLE Sells
ADD size real NULL
```

Drop column

```
ALTER TABLE Sells
DROP COLUMN size
```

Some database systems don't allow deleting a column
ALTER TABLE Command

Change column type

ALTER TABLE Sells
MODIFY COLUMN size decimal

Add primary key

ALTER TABLE Sells
ADD CONSTRAINT sells_pk PRIMARY KEY (cafe, drink, size)

Drop constraints

ALTER TABLE Sells
DROP CONSTRAINT std_pk
DROP Command

• Drop table
  
  DROP TABLE Sells

• Drop view
  
  DROP VIEW CanDrink
Outline

✓ Inner and Outer Joins
✓ Updating Instance
✓ Updating Schema
• 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>;

• Views are not stored in the database, but can be queried as if they existed.
  • We’ll talk about an exception later

• In contrast, a relation whose value is really stored in the database is called a base table.
Example: View Definition

- CanDrink (customer, drink) is a view “containing” the customer-drink pairs such that the customer frequents at least one cafe that serves the drink on relations Frequents (customer, cafe) and Sells (cafe, drink, price)

CREATE VIEW CanDrink AS
SELECT customer, drink
FROM Frequents, Sells
WHERE Frequents.cafe = Sells.cafe;
Example: Accessing a View

• You may query a view as if it were a base table.
• Example:

  SELECT drink FROM CanDrink
  WHERE customer = 'Sally';
What’s Useful about Views

1. Can be used as relations in other queries
   • Allows the user to query things that make more sense
   • Can be stored (materialized) as appropriate
   • Sometimes can even be updated!
What’s Useful about Views

2. Can facilitate security/access control
   • We can assign users permissions on different views
   • Can select or project so we only reveal what we want!

3. Describe *transformations* or *mappings* from one schema (the base relations) to another (the output of the view)
   • The basis of converting from different data models or representations
   • Incredibly useful for logical data independence
Materialized Views

A **materialized view** is one that is computed once and its results are stored as a table

- Think of this as a cached answer
- These are incredibly useful!
- Techniques exist for using materialized views to answer other queries
- Materialized views are the basis of relating tables in different schemas

CREATE MATERIALIZED VIEW AS ...
Views Should Stay Fresh

• Views (sometimes called *intensional relations*) behave, from the perspective of a query language, exactly like base relations (*extensional relations*)

• But there’s an association that should be maintained:
  • If tuples change in the base relation, they should change in the view (whether it’s materialized or not)
  • If tuples change in the view, that should reflect in the base relation(s)
View Maintenance and Updates

• There exist algorithms to incrementally recompute a materialized view when the base relations change
• We can try to propagate view changes to the base relations
  • However, there are lots of views that aren’t easily updatable:

```
R   S   R \times S
A B C
1 2 4
2 2 3
delete?
```
The good news

We can ensure views are updatable by enforcing certain constraints:

• It is defined on a single base table
• Using only selection and projection
• No aggregates
• No DISTINCT

...but this limits the kinds of views we can have
That’s all folks!

✓ Joins
✓ Updating Instance
✓ Updating Schema
✓ Views

Next lecture
• Constraints
• Project 1 + Survey Overview