CPSC 421
Database Management Systems

Lecture 4:
More SQL and Relational Algebra

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Today’s Agenda

• Go over last week’s quiz
• New assignments!
• More on SQL queries and relational algebra
More SQL Query Constructs

1. Extensions include SUM, COUNT, MIN, MAX, AVG, etc.

SELECT ...
FROM ...
WHERE ...
(SELECT ... FROM ... WHERE ...)
UNION
(SELECT ... FROM ... WHERE ...)

2. Extensions include various kinds of JOINs

ORDER BY ...
GROUP BY ...
HAVING ...

3. Additional comparators, e.g., EXISTS, IN, ANY

4. Operators that take two or more complete SQL queries as arguments, e.g., UNION and INTERSECT

5. Several additional clauses, e.g., ORDER BY, GROUP BY, and HAVING
Sample Database

- We’ll use this database in our examples

\[
\text{Customer(}\text{Number, Name, Address, CRating, CAmount, CBalance, Salesperson)}\]

\[
\text{Salesperson(}\text{Number, Name, Address, Office)}\]

Foreign key: Customer.Salesperson \(\rightarrow\) Salesperson.Number

SQL SELECT

Aggregate operators: \(\text{COUNT, SUM, MIN, MAX, and AVG}\)

\[
\text{SELECT MIN(Cbalance), MAX(Cbalance), AVG(Cbalance) FROM Customer;}
\]

\[
\text{SELECT MIN(Cbalance), MAX(Cbalance), AVG(Cbalance) FROM Customer WHERE age > 35;}
\]

If one aggregate operator appears in the SELECT clause, then ALL of the entries in the select clause must be an aggregate operator
- Unless the query includes a \text{GROUP BY} clause (covered later)
Applying Aggregates

SELECT Name, Phone, AVG(Age)
FROM Student
WHERE Major = "CS";

• This SQL query is *not* well-formed, and not allowed
• What would/should the query answer be?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Phone</th>
<th>Age</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>123</td>
<td>24</td>
<td>CS</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>456</td>
<td>28</td>
<td>CS</td>
</tr>
<tr>
<td>3</td>
<td>Arun</td>
<td>789</td>
<td>32</td>
<td>CS</td>
</tr>
<tr>
<td>4</td>
<td>John</td>
<td>999</td>
<td>18</td>
<td>English</td>
</tr>
</tbody>
</table>

SQL SELECT

What is the difference between these two queries?

SELECT COUNT(Name)  SELECT DISTINCT Name
FROM Customer       FROM Customer

When will these two queries return the same answer?
– that is, for what kind of DB instance would these return the same answer
SQL SELECT

• What is the implication of using DISTINCT when computing the SUM or AVG of an attribute?

SUM(DISTINCT(age)) vs. SUM(age)

• What is the implication of using DISTINCT when computing the MIN or MAX of an attribute?

MIN(DISTINCT(age)) vs. MIN(age)
SQL SELECT

• The SELECT clause list can also include simple arithmetic expressions using +, -, *, and /

```
SELECT (CAmount – CBalance) AS AvailableCredit, Name
FROM Customer
WHERE CAmount > 0
```

• This query computes the available credit for those Customers that have CAmount > 0

More SQL Query Constructs

```
SELECT ... FROM ... WHERE ...
```

2. Extensions include various kinds of JOINs
There are a number of join types that can be expressed in the WHERE clause

- inner join (the regular join)
- cross join
- natural join
- left outer join
- right outer join
- full outer join

These can be expressed using basic SELECT-FROM-WHERE queries --- they act as “syntactic sugar”
SQL FROM

There are a number of join types that can be expressed in the WHERE clause

- inner join (the regular join)
- cross join
- natural join
- left outer join
- right outer join
- full outer join

These can be expressed using basic SELECT-FROM-WHERE queries
--- they act as “syntactic sugar”

These are new operators

These two queries are equivalent

```
SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number
WHERE C.CreditRating < 6;
```

```
SELECT C.Name, S.Name
FROM Customer C, Salesperson S
WHERE C.Salesperson = S.Number AND C.CreditRating < 6;
```
SQL FROM

SQL and equivalent relational algebra queries

SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number
WHERE C.CreditRating < 6;

\[ \pi_{C.Name, S.Name}(\sigma_{C.CreditRating < 6}(Customer \bowtie C.Salesperson = S.Number Salesperson)) \]

SELECT C.Name, S.Name
FROM Customer C, Salesperson S
WHERE C.Salesperson = S.Number AND C.CreditRating < 6;

\[ \pi_{C.Name, S.Name}(\sigma_{C.CreditRating < 6 \land C.Salesperson = S.Number}(Customer \times Salesperson)) \]

---

SQL FROM

JOIN with USING clause when attributes in the 2 tables have the same name

Course(CNumber, CName, Description)
Teacher(TNumber, TName, Phone)
Offering(CNumber, TNumber, Time, Days, Room)

These two queries are equivalent

SELECT C.CNumber, C.CName, Room
FROM Course C JOIN Offering USING(CNumber);

SELECT C.CNumber, C.Name, Room
FROM Course C JOIN Offering O ON C.CNumber=O.CNumber;

USING clause doesn’t need (and can’t have) a correlation name
SQL FROM

• Basic Join ≡ INNER JOIN

For the INNER JOIN

```
SELECT C.Name, S.Name
FROM Customer C INNER JOIN Salesperson S ON
    C.Salesperson = S.Number;
```

the query answer includes all “matches” but excludes:

– Customer rows that do not have a Salesperson
– Salesperson rows that are not assigned to any customers

SQL FROM

• Basic Join ≡ INNER JOIN
• The keyword “INNER” is optional …

This INNER JOIN query

```
SELECT C.Name, S.Name
FROM Customer C INNER JOIN Salesperson S ON
    C.Salesperson = S.Number;
```

is the same as

```
SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number;
```
SQL FROM

• A CROSS JOIN is a cross product

These queries are equivalent

SELECT *
FROM Customer, Salesperson

SELECT *
FROM Customer CROSS JOIN Salesperson;

SQL SELECT

• Equi-Join vs. Natural Join
  
  – Equi-join: Account $^=\_Account$ Deposit

• When the join is based on equality we always have two identical attributes (columns) in the answer

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
<th>Account</th>
<th>TransId</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000</td>
<td>checking</td>
<td>102</td>
<td>1</td>
<td>10/22/09</td>
<td>500</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000</td>
<td>checking</td>
<td>102</td>
<td>2</td>
<td>10/29/09</td>
<td>200</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000</td>
<td>checking</td>
<td>104</td>
<td>3</td>
<td>10/29/09</td>
<td>1000</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10000</td>
<td>checking</td>
<td>105</td>
<td>4</td>
<td>11/2/09</td>
<td>10000</td>
</tr>
</tbody>
</table>

• The natural join eliminates the duplicate column for joins based on equality
SQL FROM

**Natural join** requires attributes with the same name in the two relations

- For the previous example, we would need to rename “Account” to “Number” to use natural join

- If you don’t specify which attributes to join on, natural join will join on *all attributes with the same name*

SQL FROM

- **NATURAL JOIN** is like a “macro” that joins tables with an equality condition for all attributes *with the same name*

- Consider the following database

```sql
Course(CNumber, CName, Description)
Teacher(TNumber, TName, Phone)
Offering(CNumber, TNumber, Time, Days, Room)
```
SQL FROM

NATURAL JOIN drops columns automatically

• With any join based on equality, there will always be pairs of identical columns (one for each column joined)

• The NATURAL JOIN eliminates one of the duplicate columns

SQL FROM

List the course and teacher name for all course offerings

This query can be expressed with the NATURAL JOIN or with an INNER JOIN

• These two queries are equivalent

```
SELECT CName, TName
FROM Course C, Offering O, Teaching T
WHERE C.CNumber = O.CNumber AND O.TNumber = T.Tnumber
```

```
SELECT CName, TName
FROM Course NATURAL JOIN Offering NATURAL JOIN Teacher;
```

• This works because the join attributes have the same attribute names
SQL FROM

Natural Join can be “risky” …

What if we had different attribute names?

\[
\begin{align*}
\text{Course} & : (CNumber, \text{Name}, \text{Description}) \\
\text{Teacher} & : (TNumber, \text{Name}, \text{Phone}) \\
\text{Offering} & : (CNumber, TNumber, Time, Days, Room)
\end{align*}
\]

\[
\text{SELECT CName, TName}
\]

\[
\text{FROM Course NATURAL JOIN Offering NATURAL JOIN Teacher;}
\]

• What else could lead to a problem for natural join?

---

SQL FROM

INNER JOIN vs. OUTER JOIN

For the INNER JOIN

\[
\text{SELECT C.Name, S.Name}
\]

\[
\text{FROM Customer INNER JOIN Salesperson ON}
\]

\[
\text{C.Salesperson = S.Number}
\]

the query answer does not include

- a customer that does not have a salesperson
- a salesperson that is not assigned to any customers
SQL FROM

Customer without a salesperson
Salesperson without a customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>123 X St</td>
<td>700</td>
<td>10000</td>
<td>9000</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>222 Y St</td>
<td>700</td>
<td>8000</td>
<td>3750</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
<td>111 Z St</td>
<td>700</td>
<td>11000</td>
<td>9000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Miller</td>
<td>555 A St</td>
<td>100</td>
</tr>
<tr>
<td>65</td>
<td>Rojas</td>
<td>555 A St</td>
<td>101</td>
</tr>
<tr>
<td>75</td>
<td>Martin</td>
<td>777 B St</td>
<td>200</td>
</tr>
</tbody>
</table>

SQL FROM

An INNER (regular) JOIN includes only those customers that have salespersons (only the matches)

```
SELECT C.Name, S.Name
FROM Customer INNER JOIN Salesperson ON C.Salesperson = S.Number
```

A LEFT OUTER JOIN will include all matches plus all
– customers that do not have a Salesperson

A RIGHT OUTER JOIN will include all matches plus all
– salespersons that are not assigned to any customers

A FULL OUTER JOIN will include all of these
SQL FROM

INNER JOIN on C.Salesperson = S.Number gives:
  1 Smith 123 X St 700 10000 9000 55 55 Miller 555 A St 100
  2 Jones 222 Y St 700 8000 3750 65 65 Rojas 555 A St 101

LEFT OUTER JOIN on C.Salesperson = S.Number gives us:
  1 Smith 123 X St 700 10000 9000 55 55 Miller 555 A St 100
  2 Jones 222 Y St 700 8000 3750 65 65 Rojas 555 A St 101
  3 Wei  111 Z St 700 11000 9000 NULL NULL NULL NULL

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>123 X st</td>
<td>700</td>
<td>10000</td>
<td>9000</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>222 Y st</td>
<td>700</td>
<td>8000</td>
<td>3750</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
<td>111 Z st</td>
<td>700</td>
<td>11000</td>
<td>9000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Miller</td>
<td>555 A St</td>
<td>100</td>
</tr>
<tr>
<td>65</td>
<td>Rojas</td>
<td>555 A St</td>
<td>101</td>
</tr>
<tr>
<td>75</td>
<td>Martin</td>
<td>777 B St</td>
<td>200</td>
</tr>
</tbody>
</table>

SQL FROM

INNER JOIN on C.Salesperson = S.Number gives:
  1 Smith 123 X St 700 10000 9000 55 55 Miller 555 A St 100
  2 Jones 222 Y St 700 8000 3750 65 65 Rojas 555 A St 101

RIGHT OUTER JOIN on C.Salesperson = S.Number gives us:
  1 Smith 123 X St 700 10000 9000 55 55 Miller 555 A St 100
  2 Jones 222 Y St 700 8000 3750 65 65 Rojas 555 A St 101
  NULL NULL NULL NULL NULL NULL NULL 75 Martin 777 B St 200

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>123 X st</td>
<td>700</td>
<td>10000</td>
<td>9000</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>222 Y st</td>
<td>700</td>
<td>8000</td>
<td>3750</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
<td>111 Z st</td>
<td>700</td>
<td>11000</td>
<td>9000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Miller</td>
<td>555 A St</td>
<td>100</td>
</tr>
<tr>
<td>65</td>
<td>Rojas</td>
<td>555 A St</td>
<td>101</td>
</tr>
<tr>
<td>75</td>
<td>Martin</td>
<td>777 B St</td>
<td>200</td>
</tr>
</tbody>
</table>
## SQL FROM

**FULL OUTER JOIN** on `C.Salesperson = S.Number` gives us:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>123 X St</td>
<td>700</td>
<td>10000</td>
<td>9000</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>222 Y St</td>
<td>700</td>
<td>8000</td>
<td>3750</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
<td>111 Z St</td>
<td>700</td>
<td>11000</td>
<td>9000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>123 X st</td>
<td>700</td>
<td>10000</td>
<td>9000</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>222 Y st</td>
<td>700</td>
<td>8000</td>
<td>3750</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>Wei</td>
<td>111 Z st</td>
<td>700</td>
<td>11000</td>
<td>9000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Miller</td>
<td>555 A St</td>
<td>100</td>
</tr>
<tr>
<td>65</td>
<td>Rojas</td>
<td>555 A St</td>
<td>101</td>
</tr>
<tr>
<td>75</td>
<td>Martin</td>
<td>777 B St</td>
<td>200</td>
</tr>
</tbody>
</table>

## SQL FROM

You can put a complete query expression in the FROM clause (a form of "subquery")

```sql
SELECT ...
FROM Employee E, (SELECT ... FROM ...) WHERE ...
```

- The parentheses are important here
Relational Algebra

Eight standard operators:

- \( \pi \) project
- \( \sigma \) select
- \( \cup \) union
- \( \cap \) intersect
- \( - \) difference
- \( \times \) cross product
- \( \bowtie \) join
- \( \div \) divide
- \( \rho \) renaming

These 4 you have seen already
Relational Algebra

Eight standard operators:
- $\pi$ project
- $\sigma$ select
- $\cup$ union
- $\cap$ intersect
- $-$ difference
- $\times$ cross product
- $\bowtie$ join
- $\div$ divide
- $\rho$ renaming

These 3 are from set theory

These operators can only be used with relations that are “union-compatible”
Relational Algebra

• Two relations are union-compatible if
  – They have the same arity (the same number of attributes)
  – The corresponding attribute have the same domains

• These relations are union compatible
  – Checking(CNum: int, COwner: string, CBalance: float)
  – Savings(SNum: int, SOwner: string, SBalance: float)

Union, intersection, and different all require union-compatible relations

Relational Algebra

• U union

CheckingAccount U SavingsAccount

Note that attributes are from the first relation in the query
Relational Algebra

• \( \cap \) intersection

CheckingAccount \( \cap \) SavingsAccount
  – What is the answer to this query?

\( \pi_{COwner}(\text{CheckingAccount}) \cap \pi_{SOwner}(\text{SavingsAccount}) \)
  – What is the answer to this query?

It is empty – no tuples appear in both relations.

Smith – the only owner in SavingsAccount

Relational Algebra

• \( \cap \) intersection

CheckingAccount \( \cap \) SavingsAccount
  – What is the answer to this query?

\( \pi_{COwner}(\text{CheckingAccount}) \cap \pi_{SOwner}(\text{SavingsAccount}) \)
  – What is the answer to this query?

Smith – the only owner in SavingsAccount
Relational Algebra

- difference

CheckingAccount − SavingsAccount

Find all tuples that are in the CheckingAccount relation but are not in the SavingsAccount relation

\[ \pi_{COwner}(\text{CheckingAccount}) - \pi_{SOwner}(\text{SavingsAccount}) \]

- Everyone in CheckingAccount except Smith

More SQL Query Constructs

SELECT ...
FROM ...
WHERE ...

(\text{SELECT...FROM...WHERE...}) \text{ UNION } (\text{SELECT...FROM...WHERE...})

4. Operators that take two or more complete SQL queries as arguments, e.g., UNION and INTERSECT
**SQL UNION**

(SELECT C.Name  
FROM Customer C  
WHERE C.Name LIKE "B%")

UNION

(SELECT S.Name  
FROM Salesperson S  
WHERE S.Name LIKE "B");

• Two complete queries with the UNION operator in between

**SQL INTERSECTION**

(SELECT C.Name  
FROM Customer C)  
INTERSECT  
(SELECT S.Name  
FROM Salesperson S);

• Two complete queries with the INTERSECT operator in between
**SQL EXCEPT (i.e., DIFFERENCE)**

```
(SELECT C.Name
FROM Customer C)
EXCEPT
(SELECT S.Name
FROM Salesperson S);
```

- Two complete queries with the INTERSECT operator in between

- MySQL doesn’t support EXCEPT (difference)
  - Inconvenient, but can be simulated using other operators

**SQL ALL**

- Dealing with bags in UNION, INTERSECT, EXCEPT
  - UNION vs UNION ALL
  - INTERSECT vs INTERSECT ALL
  - EXCEPT vs EXCEPT ALL

- If you don’t specify ALL, the answer is computed on sets:
  - Eliminate duplicates from first operand
  - Eliminate duplicates from second operand
  - Compute operation
  - Eliminate duplicates from answer

*Why do this and not just apply operator and eliminate duplicates from the result?*
For Next Tuesday

• Reading
  – Ch 4: Intro, 4.1, 4.2
  – Ch 5: 5.5
  – On reserve in the library

• Be sure to know:
  – Relational algebra, basic aggregates, and joins

• Homework
  – Homework 2 due next Thursday
  – Tuesday we’ll do group by, having, order by (you’ll need this for homework 2)
  – First part of project assigned … start early!