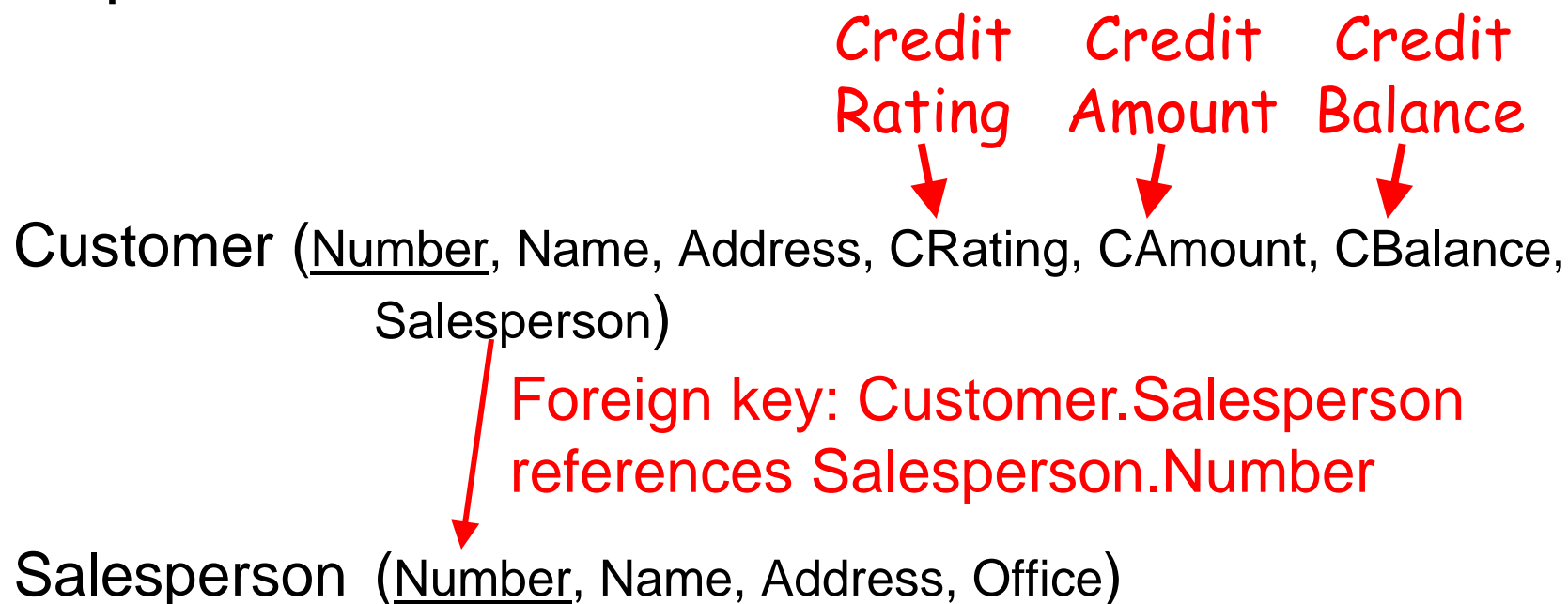


Lecture 3

- Subqueries in the WHERE clause
 - Attribute compared to subquery
 - Attribute compared to SOME|ALL subquery
 - Correlated subqueries vs. subqueries that are NOT correlated
 - Attribute IN or NOT IN subquery
 - EXISTS, NOT EXISTS, UNIQUE, NOT UNIQUE subquery
- Division operator in Relational Algebra
- A few other features in SQL

Review our Sample Database

We use the following database for sample SQL queries:



Sample Data

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	<null>

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	wei	bbb	26

Subquery in the where clause: Attribute compared to subquery

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating = (SELECT MAX (C2.CRating)
                          FROM   Customer C2);
```

This is a complete
SELECT..FROM..WHERE
query – inner query.

Outer query.

The circled expression is called the Inner Query. The rest is the Outer Query.

How could this work? What sort of query answer will the inner query produce? It better be JUST one row with JUST one attribute. We'll compare that with C1.CRating.

Subquery in the where clause: Attribute compared to subquery (cont.)

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating = (SELECT MAX (C2.CRating)
                          FROM    Customer C2);
```

How would you evaluate this query?

Just like we always do. Start with the FROM clause in the outer query. Take a row from the Customer table and evaluate the WHERE clause which includes evaluating the INNER query, in its entirety.

Evaluation of this Query: Attribute compared to subquery

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	<null>

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating = (SELECT MAX (C2.CRating)
                    FROM Customer C2);
```

First customer row (1, smith, xxx, 5, 1000, 1000, 101)

Evaluate the WHERE clause; first evaluate the inner query.

What is the max CRating in the Customer table? It is 10 (from row 3).

So, is the CRating from customer row 1 = to 10 (the answer from the inner query)? NO.

Ignore row 1.

How about row 2? NO. Ignore row 2.

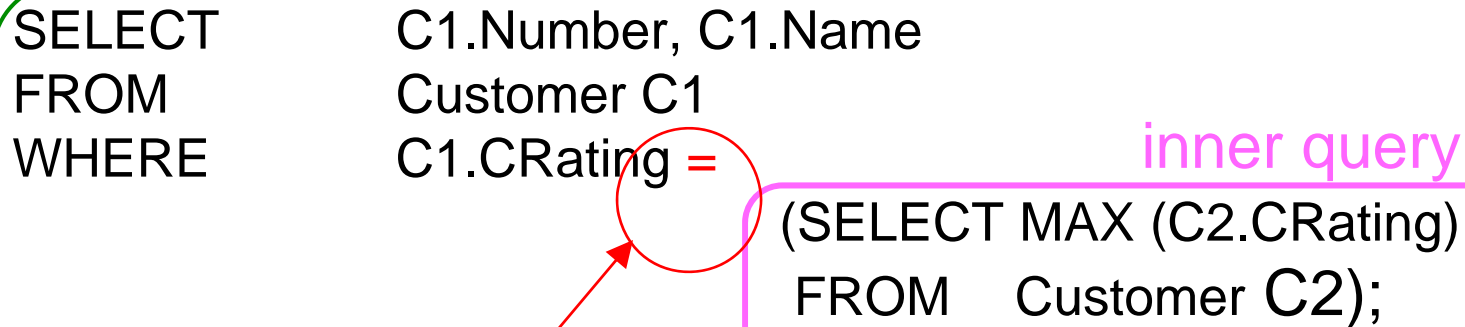
How about row 3? Yes; place row 3 in the intermediate query answer. Then drop columns.

Final query answer:

Number	Name
3	wei

Subqueries in the where clause: Attribute compared to subquery - syntax

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating = (SELECT MAX (C2.CRating)
                          FROM   Customer C2);
```



outer query

The comparator can be any of the six standard comparators: =, >, <, >=, <=, <> (not equal)

Note: inner query must return a single value!

row value constructors in SQL:1999

- A row value constructor allows you to create a row “on the fly” for example:

WHERE (E.Lname, E.Fname) = (S.Lname, S.Fname)



Each of these are rows...

and the equality comparison is doing a pair-wise comparison on the two rows.

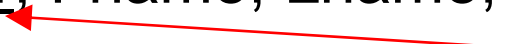
Row value constructors can be used with subqueries

A row value constructor allows you to create a row “on the fly” to compare to subquery:

Example:


```
SELECT T.Fname, T.Lname
FROM   Teacher T
WHERE  (T.Fname, T.Lname) =
        (SELECT S.Fname, S.Lname
         FROM Student S);
```

Teacher (Number, Fname, Lname, Office, Phone)
Student (Id, Fname, Lname, Major, Advisor)



Subqueries (inner queries) in the where clause that are NOT correlated to the outer query

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating =
            (SELECT MAX (C2.CRating)
             FROM   Customer C2);
```



The inner query doesn't mention any attributes from the outer query. That is, C1 is not mentioned in the inner query.

In this case, **you only need to evaluate the inner query once** – because nothing changes when each tuple from the outer query is evaluated. We say that this inner query is NOT correlated. (more about this later)

Subquery in the WHERE clause: attribute compared to SOME | ALL subquery

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE  S.Name = SOME (SELECT C.Name
                     FROM   Customer);
```

- For **SOME**, the expression must be true for **at least one row** in the subquery answer
 - **ANY** is an older form of **SOME**
- For **ALL**, the expression must be true for **all rows** in the subquery answer.

Subquery in the WHERE clause: attribute compared to SOME | ALL subquery - syntax

<attribute-name> <comparator> **SOME | ALL** <subquery>

can appear in the WHERE clause

We can use any of the standard comparators: =, >, <, >=, <=, <>

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE  S.Name = SOME (SELECT C.Name
                       FROM   Customer);
```

What will this query return? Is this subquery correlated?

Subquery in the WHERE clause:
attribute compared to SOME | ALL subquery.

Example using ALL


```
SELECT S.Name
FROM Salesperson S
WHERE S.Name = ALL (SELECT C.Salesperson
                    FROM Customer C
                    WHERE C.CRating = 3);
```

We want to know whether the S.Name is equal to all the values returned by the subquery.

State what this query computes in English.

Another example of a subquery that is NOT correlated

```
SELECT S.Name
FROM Salesperson S
WHERE S.Name = ALL (SELECT C.Salesperson
                    FROM Customer C
                    WHERE C.CRating = 3);
```



Notice that the inner query doesn't mention any attributes from the outer query. That is, S is not mentioned in the inner query.

In this case, you only need to evaluate the inner query once – because nothing changes when each tuple from the outer query is evaluated.

The inner query is NOT correlated.

Subquery in the WHERE clause: attribute IN | NOT IN subquery

SELECT
FROM
WHERE

C1.Number, C1.Name
Customer C1
C1.Name IN

(SELECT Name
FROM Salesman);

SQL query returns just one
attribute – that is comparable
To C1.Name; there could be
many rows.

Syntax: There are two operators, **IN** and **NOT IN**, that can appear in this form:

<attribute-name> IN (subquery)

attribute matches at least one value returned from subquery

or

<attribute-name> NOT IN (subquery)

attribute matches none of the values returned from subquery

Subquery with “IN” – can be equivalent to a join

```
SELECT      S.Number, S.Name
FROM        Salesperson S
WHERE       S.Number IN (SELECT      C.Salesperson
                        FROM        Customer C);
```

```
SELECT      DISTINCT S.Number, S.Name
FROM        Salesperson S, Customer C
WHERE       S.Number = C.Salesperson;
```

Are these two queries equivalent?

Do we need to use the DISTINCT clause in the second query in order for these two queries to be equivalent?

Is this subquery correlated?

“IN” and “SOME”

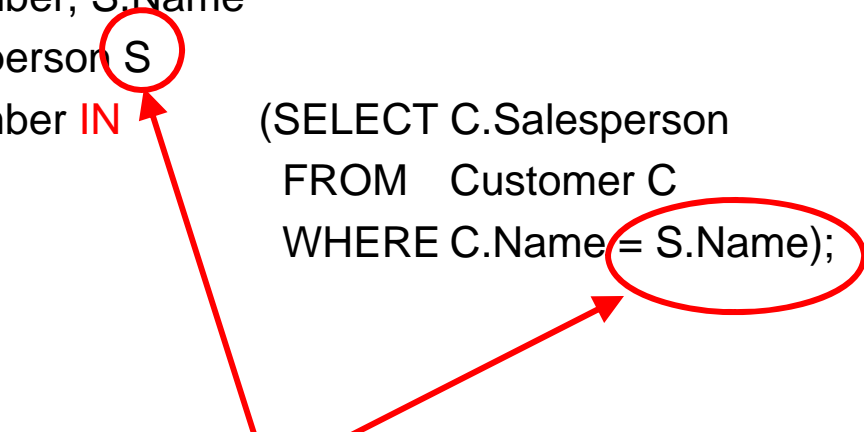
You can substitute = **SOME** for **IN**, and vice versa, to make an equivalent query, e.g.,

```
SELECT      C.Number, C.Name
FROM        Customer C
WHERE       C.address IN
            (SELECT S.address
             FROM   Salesman S);
```

```
SELECT      C.Number, C.Name
FROM        Customer C
WHERE       C.address = SOME
            (SELECT S.address
             FROM   Salesman S);
```

Example of a Correlated Subquery

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE  S.Number IN (SELECT C.Salesperson
                   FROM   Customer C
                   WHERE  C.Name = S.Name);
```



Because the subquery mentions an attribute from a table in the outer query, the subquery must be re-evaluated every time the WHERE clause from the outer query is evaluated.

Evaluation of a Correlated Subquery – first row in table from outer query

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	<null>

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	wei	bbb	26

```

SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.Salesperson
                   FROM Customer C
                   WHERE C.Name = S.Name);

```

Consider the first row of Salesperson (from the outer query); evaluate the WHERE clause from the outer query - you must evaluate the inner query with "johnson" substituted in for S.Name. Subquery returns empty answer; so ignore row 1.

Evaluation of a Correlated Subquery – second row from table in outer query

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	<null>

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	wei	bbb	26

```
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.Salesperson
                   FROM Customer C
                   WHERE C.Name = S.Name);
```

Consider the second row of Salesperson (from the outer query); evaluate the WHERE clause from the outer query with “wei” substituted in for S.Name in the inner query. Subquery returns <null> from Customer row 3). Does the Salesperson Number in row 2 of the Salesperson table match the salesperson number returned by the subquery? No. So ignore row 2 in the Salesperson table. We’re done. Answer is empty.


EXISTS and NOT EXISTS before a subquery

```
SELECT C.Name
FROM Customer C
WHERE EXISTS (SELECT *
              FROM Salesperson S
              WHERE S.Number = C.Salesperson
                  AND S.Name = C.Name);
```

If the answer to the subquery is not empty -
then the EXISTS predicate returns TRUE

Predicates before subqueries

```
SELECT C.Name
FROM Customer C
WHERE EXISTS (SELECT *
              FROM Salesperson S
              WHERE S.Number = C.Salesperson
                 AND S.Name = C.Name);
```



Four predicates that can be applied to a subquery in SQL:

EXISTS (subquery) - is the subquery answer non-empty?

NOT EXISTS (subquery) – is the subquery answer empty?

UNIQUE (subquery) – does the subquery answer have just one row?

NOT UNIQUE (subquery) – does the subquery answer have more than one row?

(repeated slide) row value constructors in SQL:1999

- A row value constructor allows you to create a row “on the fly” for example:

WHERE (E.Lname, E.Fname) = (S.Lname, S.Fname)



Each of these are rows...

and the equality comparison is doing a pair-wise comparison on the two rows.

(repeated slide) table value constructors in SQL:1999

- You can also create a table “on the fly”

VALUES row-value-expr, ..., row-value-expr

Example:

```
INSERT INTO movie_stars
```

```
VALUES (“Rocky Horror Picture Show”, 1977, “Curry, Tim”),  
      (“Rocky V”, 1984, “Stallone, Sylvester”);
```

What about row/table value constructors in relational algebra?

If you need to have either a row or a relation, you can just define it and then use it in relational algebra.

For row-values, you could say something like:

you could use (“John”, 5, “male”) directly in your query.

For table-values you could say something like:

Let $R = \{(\text{“John”}, 5, \text{“male”}), (\text{“Sue”}, 6, \text{“female”})\}$

or let R be the table

Name	Age	Gender
John	5	male
Sue	6	female

and then use R in expressions ... like $R \times \text{Student}$...or whatever

Relational Algebra: Divide Operator

Suppose we have this extra table, in the Bank database:

Account-types

Type
checking
savings

Suppose we would like to know which customers have at least one account of each type of account. That is, we want to know who has accounts of ALL the types.

We can use the Divide operator in Rel. Alg.

$(\pi_{\text{Owner, Type}} \text{Account}) \div \text{Account-types}$

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Account-types	Type
	checking
	savings

	Owner
	J. Smith

Find account owners
who have ALL types of
accounts.

Divide Operator

For $R \div S$ where $R (r_1, r_2, r_3, r_4)$ and $S(s_1, s_2)$

Since S has two attributes, there must be two attributes in R (say r_3 and r_4) that are defined on the same domains, respectively, as s_1 and s_2 . We could say that (r_3, r_4) is union-compatible with (s_1, s_2) .

The query answer has the remaining attributes (r_1, r_2) . And the answer has a tuple, (r_1, r_2) , in the answer if the (r_1, r_2) value appears with *every* S tuple in R .

How does divide work?

$(\pi_{\text{Owner, Type}} \text{Account}) \div \text{Account-types}$

$\pi_{\text{Owner, Type}} \text{Account}$

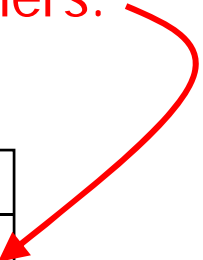
	Owner	Type
	J. Smith	checking
	W. Wei	checking
	J. Smith	savings
	M. Jones	checking
	H. Martin	checking

Can we find an owner where there are enough tuples in this table for that owner so that we can match EVERY tuple in Account-types?

List all such owners.

Account-types	Type
	checking
	savings

	Owner



Divide as a derived operator

- Division is another operator that can be **defined using the five basic operators.**
- Idea: For $R(x,y) \div S(y)$, we need to compute all x values that are not “**disqualified**” by some y value in S .
 - An x value is disqualified if for some y value in S , (x,y) is not in R

Disqualified x values: $\pi_x((\pi_x(R) \times S) - R)$

$R \div S = \pi_x(R) - \text{disqualified } x \text{ values}$

$R \div S = \pi_x(R) - \pi_x((\pi_x(R) \times S) - R)$

Division in SQL

Find account owners who own all types of accounts

```
SELECT A1.Owner  
FROM Account-owner A1  
WHERE NOT EXISTS (  
    SELECT T.Type  
    FROM Account-types T  
    WHERE NOT EXISTS (  
        SELECT A2.*  
        FROM Account-owner A2  
        WHERE A2.Type=T.Type  
        AND A2.Owner=A1.Owner))
```

} Find owners **A1** such that

} There is no account type **T**

} That **A1** does not possess

Temporary tables for intermediate results

```
SELECT * INTO loistemp ←  
FROM Agent  
WHERE ...
```

Creates a temporary table named “loistemp”

```
SELECT ...  
FROM Agent A, loistemp I  
WHERE ....
```

You can use the table, “loistemp”, in a subsequent query.

Temp tables in Postgres

1. Introdb_readonly does NOT have permission to create temp tables
2. But there is a second schema within the Spy database called temp. Introdb_readonly can create tables in temp.
3. `SELECT * into temp.lois3`
`FROM agent`
4. Look at the tables in the temp schema
5. Drop your temp tables when you're done!
6. Anyone (using introdb_readonly) can drop tables in the temp schema. Drop tables that are old/large!