Lecture 3

- GROUP BY, HAVING, ORDER BY (final slides in Lecture 2)
- 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
- Views, Levels of Abstraction, Data Independence

Queries for the "Stock" sample data (an in-class exercise – see data on next page)

Process the data (in your envelope) to answer the following queries:

1. SELECT * FROM Stock
   WHERE Size = '16oz';
2. SELECT Store, Count(*)
   FROM Stock
   GROUP BY Store
   HAVING AVG(Size >= 16);
3. SELECT Store, Product
   FROM Stock
   WHERE Size = '16oz'
   GROUP BY Store, Product
4. Write an SQL query to find for each store, the product with the highest price. List the store and product in the final query answer.

Store	Beverage	Size	Price
Plaid Pantry	Whole Milk	16oz	1.15
Plaid Pantry	2% Milk	16oz	1.05
Plaid Pantry	Whole Milk	64oz	3.20
Plaid Pantry	Pepsi	12oz	.70
Plaid Pantry	Diet Pepsi	12oz	.70
Plaid Pantry	Pepsi	20 oz	.95
7-11	Whole Milk	8oz	.65
7-11	Chocolate Milk	8oz	.65
7-11	Whole Milk	16oz	1.10
7-11	2% Milk	16oz	1.00
7-11	Coke	12oz	.65
7-11	Diet Coke	12oz	.65
7-11	Coke	20oz	1.10
7-11	Diet Coke	20oz	1.10
7-11	Diet Caffeine-Free Cherry Low-Fizz Vitamin-Fortified Coke	20oz	1.10
Circle K	Whole Milk	8oz	.60
Circle K	Whole Milk	16oz	1.20
Circle K	Whole Milk	32oz	2.30
Circle K	Whole Milk	128oz	4.10
Circle K	Coke	20oz	1.15
Circle K	Diet Coke	20oz	1.15
Circle K	Coke	32oz	2.10
Circle K	Diet Coke	32oz	2.05
Safeway	Skim Milk	16oz	1.20
Safeway	Skim Milk	32oz	2.00
Safeway	2% Milk	16oz	1.25
Safeway	2% Milk	64oz	2.75
Safeway	Diet Pepsi	12oz	.60
Safeway	Diet Coke	12oz	.60

Review our Sample Database

We use the following database for sample SQL queries:

Credit Rating, Credit Amount, Credit Balance

Customer (Number, Name, Address, CRating, CAmount, CBalance, Salesperson)

Foreign key: Customer.Salesperson references Salesperson.Number

Salesperson (Number, Name, Address, Office)

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

Subqueries (in the where clause)

```
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.

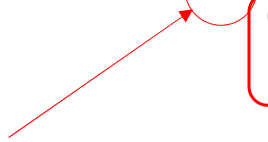
The circled expression is called the Inner Query. The rest is the Outer Query.

How would you evaluate this query?

Subqueries (in the where clause)

```
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.



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

ALL or SOME before a subquery

Syntax:

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

can appear in the WHERE clause

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

What will this query return?

ALL

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

What is the meaning of this query?

Meaning of SOME and ALL

```
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.

IN or NOT IN before a subquery

```
SELECT    C1.Number, C1.Name
FROM      Customer C1
WHERE     C1.Name IN
```

Any SQL query:

```
          (SELECT Name
          FROM  Salesman;);
```

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

<attribute-name> IN (subquery)

or

<attribute-name> NOT IN (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
                        WHERE      C.Name = S.Name);

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

Are these two queries equivalent?

"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;);
```


This is 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

You should look at the use of correlation names to figure out whether it is a correlated subquery.

Is this a correlated subquery?

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

What is the advantage of a subquery that is NOT correlated?

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

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 subquery answer not empty?
NOT EXISTS (subquery) – is subquery answer empty?
UNIQUE (subquery) – does subquery answer have just one row?
NOT UNIQUE (subquery) – does subquery answer have something besides just one row?

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.

$(\pi_{\text{Owner, Type}}$ Account) ÷ 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 ÷ S where R (r1, r2, r3, r4) and S(s1, s2)

Since S has two attributes, there must be two attributes in R (say r3 and r4) that are defined on the same domains, respectively, as s1 and s2. We could say that (r3, r4) is union-compatible with (s1, s2).

The query answer has the remaining attributes (r1, r2). And the answer has a tuple, (r1, r2), in the answer if the (r1, r2) value appears with *every* S tuple in R.

Write this SQL query in relational algebra

Customer (Number, Name, Address, CRating, CAmount, CBalance, Salesperson)

Salesperson (Number, Name, Address, Office)

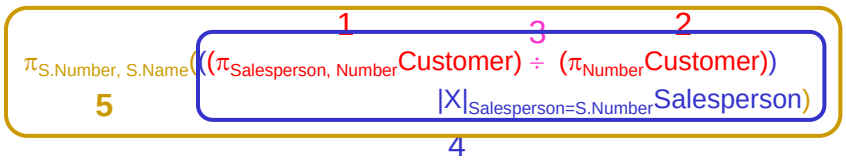
```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number = ALL      (SELECT C.Salesperson
                             FROM   Customer;);
```


Write this SQL query in relational algebra

Customer (Number, Name, Address, CRating, CAmount, CBalance, Salesperson)

Salesperson (Number, Name, Address, Office)

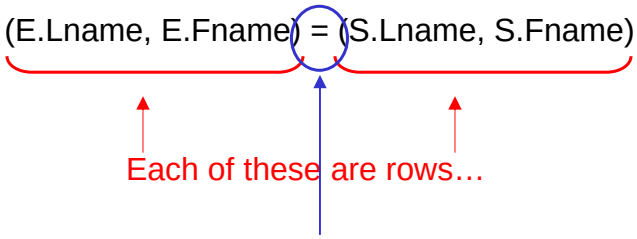
```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number = ALL      (SELECT C.Salesperson
                             FROM   Customer;);
```



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)



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

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 relational algebra?

- I'm not familiar with specific syntax for constructing row values or table values.
- However, if you need to have either a row or a relation, you can just define it and then use it in relational algebra.

  You could say something like:
  Let R = {("John", 5, "male"), ("Sue", 6, "female")
  and then use R in expressions … like R X Student
  …or whatever

Views, Levels of Abstraction and Data Independence

In our eagerness to get to SQL and relational algebra, we skipped some important topics in Chapters 1 and 3. We'll discuss those now.

Views

- A *view* is a query stored in the database
  - Think of it as a table definition for future use
- Example view definition:

CREATE VIEW *gstudents* AS SELECT S.*
FROM student S WHERE s.gpa >= 2.5

- Views can be used like *base tables*, in any query or in any other view. Like a Macro.

Example view use: simpler queries

- Completed(StudID, Course)
- Queries are often about “good” students only

SELECT S.name, S.phone
FROM gstudent S NATURAL JOIN completed C
WHERE C.course = ‘CS386’;

- Now it’s easier to write the query.

Views for Security

- This is the student relation without the gpa field.

```
CREATE VIEW sstudent AS
SELECT studID, name, address
FROM student
```

- Can you think of some other security examples?

Views for Extensibility

- A company's database includes a relation: Part (PartID: Char(4), weight:real,…)
- Weight is stored in pounds
- Company is purchased by a firm that uses metric weights
- Databases must be integrated and use Kg.
- But there's much old software using pounds.
- Solution: views!

Views for extensibility (ctd)

- Solution:

1. Base table with kilograms becomes NewParts, for integrated company.
2. CREATE VIEW Part AS
   SELECT PartID, 2.2046*weight, …(no other changes)…
   FROM NewParts
3. Old programs still call the table “Part”

But there’s one problem with views

- Views cannot always be updated unambiguously
- Consider Students(StudID, gpa, major,…)

```
CREATE VIEW majorgpa AS
SELECT major, AVG(gpa)
FROM Students
GROUP BY major
```

Majorgpa

major	gpa
CS	3.5
ECE	3.5

Updatability of views

- I want to change the GPA of CS majors from 3.5 to 3.6 .
- How can I do that?

The good news

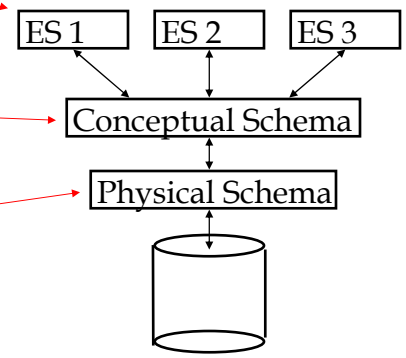
- A view can be updated if
  - It is defined on a single base table
  - Using only selection and projection
  - No aggregates
  - No DISTINCT

Levels of Abstraction

External view; user and data designer

Logical storage; data designer

Physical storage; DBA



Physical Schema

The *physical schema* is a description of how the data is physically stored in the database. It includes

- Where the data is located
- File structures
- Access methods
- Indexes

The physical schema is managed by the DBA.

Conceptual Schema

The conceptual schema is a logical description of how the data is stored. It consists of the schemas we have described with CREATE TABLE statements. It is managed by the data designer.

External Schemas

Each external schema is a combinaton of base tables and views, tailored to the needs of a single user. It is managed by the data designer and the user.

Data Independence

- A database model possesses *data independence* if application programs are immune to changes in the conceptual and physical schemas.
- Why is this important? Everything changes.
- How does the relational model achieve logical (conceptual) data independence?

Data Independence (ctd.)

- How does the relational model achieve physical data independence?
  1. Conceptual level contains no physical info
  2. SQL can program against the conceptual level
- Earlier DBMSs (network, hierarchical) did not have these properties.
  - Their languages had physical properties embedded in them.

Summary

- A view is a stored query definition
- Views can be very useful
  - Easier query writing, security, extensibility
- But views cannot be unambiguously updated
- Three levels of abstraction in a relational DBMS
  - Yields data independence: logical and physical