

Lecture 3

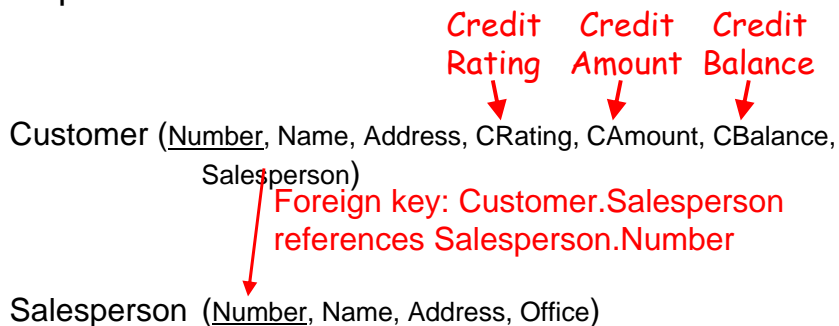
- Subqueries in the WHERE clause
 - Attribute compared to subquery
 - Attribute compared to SOME|ALL subquery
 - Correlated (and uncorrelated) subqueries
 - Attribute IN or NOT IN subquery
 - EXISTS, NOT EXISTS subquery
- Division operator in Relational Algebra
- Constraints and Triggers
- Views, Levels of Abstraction, Data Independence

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 1
Lecture 3

Recall our Sample Database

We use the following database for sample SQL queries:



CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 2
Lecture 3

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** or **subquery**.
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.Number <> ALL (SELECT C.Salesperson
                      FROM Customer C
                      WHERE C.CRating > 6);
```

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.
 - What would “= **ALL**” require?

Note: Single value on left, set value (table) on right

IN or NOT IN before a subquery

```

SELECT  C1.Number, C1.Name
FROM    Customer C1
WHERE   C1.Name IN
        (SELECT Name
         FROM Salesperson);
    
```

Any SQL query:

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

<attribute-name> IN (subquery)

or

<attribute-name> NOT IN (subquery)

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

Subquery is evaluated separately for each row in the outer query.

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   Salesperson S);
    
```

How about “NOT IN” and “<> SOME”?

Can you substitute <> SOME for NOT IN to make an equivalent query?

```
SELECT      C.Number, C.Name
FROM        Customer C
WHERE       C.address NOT IN
            (SELECT S.address
             FROM   Salesman S);
```

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

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 13
Lecture 3

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?

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 14
Lecture 3

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);
    
```

Predicates that can be applied to a subquery in SQL:

- EXISTS (subquery) - subquery answer not empty
- NOT EXISTS (subquery) –subquery answer is empty
- UNIQUE (subquery) –subquery answer has no duplicates
- NOT UNIQUE (subquery) –subquery answer has duplicates
 - Consider special case of 1-column subquery result

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 every type of account.

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 $R(A, B, C)$ and $S(B, C)$

R must have all attributes in S (B and C here).
Or have attributes with the same domains.

The query answer has the attributes of R (A here). The answer contains tuple $\langle a \rangle$ if tuple $\langle a, b, c \rangle$ is in R for every $\langle b, c \rangle$ tuple in S.

Hard to get the effect of divide in SQL

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

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

```

SELECT A.Owner
FROM Account A
WHERE NOT EXISTS
  ((SELECT T.Type
    FROM Account-types T)
  EXCEPT
  (SELECT A1.Type
    FROM Account A1
    WHERE A1.Owner = A.Owner))
    
```

Integrity Constraints (ICs)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are typically disallowed.
 - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)

Types of ICs

Domain constraints, primary key constraints, foreign key constraints, general constraints.

- *Domain constraints*: Field values must be of right type. Always enforced.

General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named, then dropped, suspended, etc.
- A very powerful mechanism

```
CREATE TABLE Account(...
    PRIMARY KEY Number,
    CHECK (CBalance <= CAmount))

CREATE TABLE Withdrawal(...
    PRIMARY KEY (Tran-id),
    CHECK(Amount <=
        SELECT Balance FROM ACCOUNT A
        WHERE A.Number = Account))
```

Domain Constraints

- You can create your own domains and use them in creating tables
- Joins are allowed between domains of the same BASE TYPE (Integer, in this case).
- SQL99 defines DISTINCT TYPES that can avoid this problem

```
CREATE DOMAIN AmtDom Integer
    CHECK(VALUE >=1000 AND
        VALUE <=100000)

CREATE TABLE Account
    ( ..., CAmount AmtDom,...)
```

Triggers

- Trigger: procedure that executes if specified changes occur to the DBMS
- Three parts:
 - Event (activates the trigger)
 - Condition (tests whether the triggers should run)
 - Action (what happens if the trigger runs)

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 25
Lecture 3

Triggers: Example

```
CREATE TRIGGER IncrCredit ON Deposit
AFTER INSERT AS
IF new.Amount >= 10000
  UPDATE Account A
  SET A.CRating = A.CRating+1
  WHERE A.Account = new.Number
```

Note the special variable **new** that represents the newly inserted row.

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 26
Lecture 3

Views, Levels of Abstraction and Data Independence

One database often supports multiple applications, which might have slightly different pictures of the world.

Views help accommodate this variation without storing redundant data.

Views

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

```
CREATE VIEW GStudent AS SELECT S.*  
FROM Student S WHERE s.gpa >= 2.5
```
- A view can be used like a *base table*, in a SELECT query or in another view. A kind of 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.

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 29
Lecture 3

Views for Security

Suppose schema is

Student(studID, name, address, major, gpa)

- This is a view of the Student table without the gpa field.

```
CREATE VIEW SecStudent AS  
SELECT studID, name, address, major  
FROM student
```

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 30
Lecture 3

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!

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 31
Lecture 3

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"

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 32
Lecture 3

But there's one problem with views

- Views cannot always be updated unambiguously
- Consider Student(StudID, gpa, major,...)

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

Majorgpa	major	gpa
	CS	3.5
	ECE	3.5

CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 33
Lecture 3

Updatability of views

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

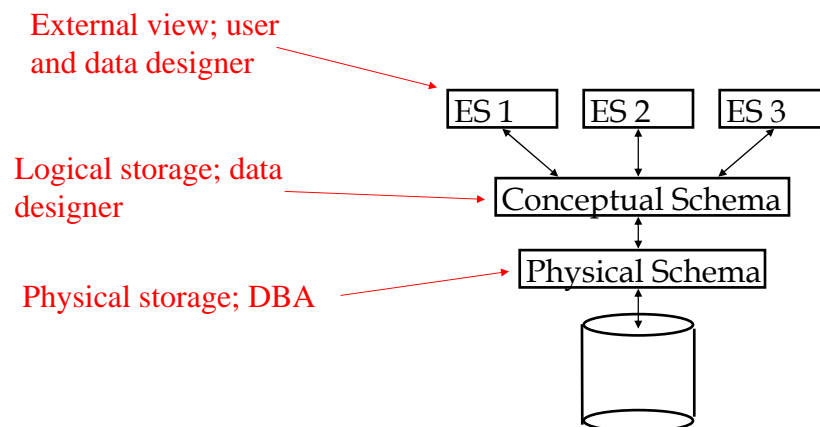
CS386 Intro. to Database Systems, © Lois Delcambre, David Maier 1999-2006
Some slides adapted from R. Ramakrishnan, with permission

Slide 34
Lecture 3

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



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 what data is in the database. 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 combination 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 protected from 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 not all views can be updated unambiguously
- Three levels of abstraction in a relational DBMS
 - Yields data independence: logical and physical