Structured Query Language (SQL) II

CEE412/CEE522

Transportation Data Management And Visualization

WINTER 2020

Announcement

Assignment 2 solution is posted

- Q 1: list all required items
- Q 3: did not form the loop properly, misplaced certain things in E/R diagram, and duplicated relationships
- Q 4: missed the pilot relationship

- Please come to the TA office hours or send emails to us if you have any questions.

Project 1

- Database will be accessible by today

Next Wednesday: Midterm 1

Today's Outline

- Grouping and Aggregation in SQL
- Subqueries
- Union, Intersection, and Difference
- Constraints in SQL

Aggregation Operators

SQL supports five aggregation operators

- SUM
- MIN
- MAX
- AVG
- COUNT
- ~~Median~~

These aggregations apply to a single attribute or value, except COUNT:

- COUNT(*) can be used to count all tuples.

Aggregation in SQL

Product

PName	Price	Category	Manufacturer
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

```sql
SELECT AVG(price)
  FROM product
 WHERE manufacturer = 'GizmoWorks'
```

(No column name)
24.99

```sql
SELECT COUNT(*) AS ProductCount
  FROM product
 WHERE manufacturer = 'GizmoWorks'
```

ProductCount
2

Aggregation in SQL - Quick Note about COUNT

Product

PName	Price	Category	Manufacturer
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

- COUNT applies to duplicates, unless otherwise stated:

```
SELECT COUNT(category) AS CategoryCt
  FROM product
```

CategoryCt
4

```
SELECT COUNT(DISTINCT category) AS CategoryCt
  FROM product
```

CategoryCt
3

Aggregation in SQL

Sale

Product	Date	Price	Quantity
Banana	2016-10-19	0.52	17
Bagel	2016-10-20	0.85	20
Bagel	2016-10-21	0.85	15
Banana	2016-10-22	0.52	7

- Example 1: find total sales

```
SELECT SUM(price * quantity) AS TotalSale
  FROM sale
```

TotalSale
42.23

- Example 2: find total sales of bagels

```
SELECT SUM(price * quantity) AS BagelSale
  FROM sale
 WHERE product = 'bagel'
```

BagelSale
29.75

Grouping and Aggregation in SQL

Usually, we want aggregations on certain parts of the relation.

Sale(product, date, price, quantity)

- Example 3: find total sales per product.

```
SELECT Product, SUM(price * quantity) AS TotalSale
  FROM sale
 GROUP BY Product
```

Product	TotalSale
Bagel	29.75
Banana	12.48

Grouping and Aggregation in SQL

Procedure of grouping and aggregation in SQL:

1. Compute the FROM and WHERE clauses.
2. Separate the table for every combination of GROUP BY attributes.
3. Apply aggregation and return one tuple for each sub-table.

When aggregation is used, SELECT can only have two types of expressions:

- Attributes in the GROUP BY clause
- Aggregations

Grouping and Aggregation in SQL

How GROUP BY makes things easy?

- Using GROUP BY

```
SELECT Product, SUM(price * quantity) AS TotalSale
  FROM sale
 GROUP BY Product
```

- Compared to not using GROUP BY (subquery / inner query / nested query)

```
SELECT DISTINCT x.Product,
        (SELECT SUM(price * quantity) FROM sale AS y
          WHERE x.product = y.product) AS TotalSale
  FROM sale AS x
```

Grouping and Aggregation in SQL

Multiple aggregations

- For each product, what is the total sales and the max quantity sold?

```
SELECT Product, SUM(price * quantity) AS TotalSale,
       MAX(quantity) AS MaxQuantity
  FROM sale
 GROUP BY Product
```

Product	TotalSale	MaxQuantity
Bagel	29.75	20
Banana	12.48	17

HAVING Clause

HAVING <conditions> may follow a GROUP BY clause.

The conditions applies to each group, and groups not satisfying <conditions> are eliminated.

The conditions in HAVING clause may refer to attributes as long as the attribute makes sense within a group; i.e., it is either:

- Attributes in the GROUP BY clause
- Aggregations

HAVING Clause

- Find the product name and total sales for each product sold after 10/1/2016 and with a total sale quantity more than 30.

```
SELECT Product, SUM(price * quantity) AS TotalSale
  FROM sale
 WHERE date > '2016-10-1'
 GROUP BY Product
HAVING SUM(quantity) > 30
```

Sale

Product	Date	Price	Quantity
Banana	2016-10-19	0.52	17
Bagel	2016-10-20	0.85	20
Bagel	2016-10-21	0.85	15
Banana	2016-10-22	0.52	7

Product	TotalSale
Bagel	29.75

Grouping and Aggregation in SQL

General form of Grouping and Aggregation

```
SELECT S
  FROM R1,…,Rn
 WHERE C1
 GROUP BY a1,…,ak
HAVING C2
```

S: may contain attributes a1,…,ak and/or corresponding aggregates but NO OTHER ATTRIBUTES

C1: any condition on the attributes in R1,…,Rn

C2: any condition on aggregate expressions

Aggregation Examples

Author(AuthorID, Name)

Write(PaperName, AuthorID)

- Find all authors who have wrote at least 10 papers

```
SELECT a.name
  FROM author AS a, write AS w
 WHERE a.authorid = w.authorid
 GROUP BY a.name
HAVING COUNT(w.papername) > 10
```

Subqueries

A parenthesized SELECT-FROM-WHERE statement (subquery) can be used in a number of places, including FROM and WHERE clauses

- In place of a relation in the FROM clause, we can place another query, and then query its result
- You can use a query that is guaranteed to return a single value in the place of a value

Subqueries

Product

PName	Price	Category	Manufacturer
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

- Find names and stock prices for all companies that produce some product in the "gadgets" category

```
SELECT DISTINCT c.cname, stockprice
  FROM (SELECT cname, stockprice
          FROM company, product
         WHERE cname = manufacturer
               AND product.category = 'gadgets'
        ) AS c
```

Name your subquery when using as a relation.

Subqueries Returning Relations

Product (pname, price, category, manufacturer)

Purchase (buyer, seller, store, product)

Company (cname, stockPrice, country)

- Find the stock prices of companies that made some products bought by Joe

```
SELECT stockprice
  FROM company, product
 WHERE cname = manufacturer
       AND pname IN (SELECT product
                       FROM purchase
                      WHERE buyer = 'Joe')
```

Subqueries Returning Relations

Similar job can be done without subquery:

- Find the stock prices of companies that made some products bought by Joe

```
SELECT stockPrice
  FROM company, product, purchase
 WHERE cname = manufacturer
       AND  pname = purchase.product
       AND  buyer = 'Joe'
```

Is this query equivalent to the previous one?

- Duplicates can make them different.

Subqueries Returning Relations

The following two queries will return exactly the same results:

```
SELECT DISTINCT stockprice
  FROM company, product
 WHERE cname = manufacturer
       AND pname IN (SELECT product
                       FROM purchase
                      WHERE buyer = 'Joe')
```

```
SELECT DISTINCT stockPrice
  FROM company, product, purchase
 WHERE cname = manufacturer
       AND  pname = purchase.product
       AND  buyer = 'Joe'
```

Which one is more interpretable?

Subqueries Returning Relations

ALL and ANY with comparison operators

- S > ALL <set>: returns TRUE if S is larger than all values in the set
- S > ANY <set>: returns TRUE if S is larger than any single value in the set

- Example: find products that are more expensive than all those produced by the company named "GizmoWorks".

```
SELECT pname
  FROM product
 WHERE price > ALL (SELECT price
                      FROM product
                     WHERE manufacturer = 'GizmoWorks')
```

Correlated Queries

A correlated query references a table outside the subquery using a named table

Student(FirstName, LastName, Gender, Age)

- Find the first name that is used by more than one student.

```
SELECT DISTINCT firstname
  FROM student AS s
 WHERE lastname <> ANY (SELECT lastname
                          FROM student
                         WHERE firstname = s.firstname)
```

Correlated

Can this be done in single query statement?

```
SELECT DISTINCT s1.firstname
  FROM student AS s1, student AS s2
 WHERE s1.firstname = s2.firstname
       AND s1.lastname <> s2.lastname1
```

Subqueries Returning One Tuple

If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.

- Usually, the tuple has one attribute.
- A run-time error occurs if there is no tuple or more than one tuple.

Subqueries Returning One Tuple

Players(Name, Salary, Height, Weight, Team)

- Question: find the name of the player with the highest salary.

This is a bit tricky, but lets do this step-by-step:

- First, find the highest salary in my table

```
SELECT MAX(salary)
  FROM player
```

(No column name)
15000000.00

Subqueries Returning One Tuple

Players(Name, Salary, Height, Weight, Team)

- Question: find the name of the player with the highest salary.

```
SELECT name, salary
  FROM player
 WHERE salary = (SELECT MAX(salary)
                   FROM player)
```

name	salary
Peyton Manning	15000000.00

Views and Temporary Tables

Views are relations, except that they are not physically stored.

- A virtual table which stores a shot-cut of a query statement.
- The query will not be run or processed unless you are accessing the view from another query.
- Results will be regenerated every time you access the view.
- Good for organize your code.

Players(Name, Salary, Height, Weight, Team)

- Example: create a view that stores Seahawks player information

```
CREATE VIEW Seahawks AS
SELECT *
  FROM player
 WHERE team = 'Seahawks'
```

→

```
SELECT MAX(salary)
  FROM Seahawks
```

Views and Temporary Tables

Temporary tables store the results in tempDB.

- Good for store some intermediate results that need to be retrieved multiple times in the future

Local temporary tables (defined by #tablename)

- Only available to the current connection and current login.
- Dropped when the connection is closed.

Global temporary tables (defined by ##tablename)

- Available to any connection upon their creation
- Dropped when the last connection using them is closed.

Views and Temporary Tables

Players(Name, Salary, Height, Weight, Team)

- Example: create a temporary table that stores Seahawks player information

```
SELECT *
  INTO #Seahawks
  FROM player
 WHERE team = 'Seahawks'
```

- You can refer to the temporary table in other queries

```
SELECT MAX(salary)
  FROM #Seahawks
```

- Remember to drop your temporary table after finishing using them

```
DROP TABLE #Seahawks
```

Union, Intersection, and Difference

Purchase(buyer, seller, store, product)

Person(pername, phone number, city)

- Example: find names of people who live in Seattle or who are buyers at GAP.

```
(SELECT pername
   FROM person
  WHERE city = 'Seattle')
  UNION
(SELECT pername
   FROM person, purchase
  WHERE buyer = pername
        AND store = 'GAP')
```

Outputs from two tables must have the same attribute names!

Union, Intersection, and Difference

If you want to preserve duplicates, use the ALL keyword.

```
(SELECT pername
   FROM person
  WHERE city = 'Seattle')
  UNION ALL
(SELECT pername
   FROM person, purchase
  WHERE buyer = pername
        AND store = 'GAP')
```

Similarly, you can use INTERSECT and EXCEPT.

Insertions

Often, you will use a query to replace the VALUES in the INSERT command.

Students(Name, StudentID, Gender, Age, Major, Phone)

Freshmen(Name, StudentID, Gender, Age, Major, Phone)

```
INSERT INTO students
SELECT *
  FROM freshmen
```

Select Into

INSERT INTO: Insert into an existing table

SELECT INTO: Create a new table containing these values

- Note: table created will have the columns contained in the select list with the same data types as the source data.

- Create a table that include information of CEE students

```
SELECT *
  INTO CEEStudents
  FROM students
 WHERE major = 'CEE'
```

Deletions

General form:

```
DELETE FROM R
 WHERE <conditions>
```

Example:

```
DELETE FROM students
 WHERE name = ‘Kris'
```

Updates

General form:

```
UPDATE R SET <new-value assignments>
WHERE <conditions>
```

Example:

```
UPDATE students
   SET phone = '111-222-3333'
 WHERE studentid = 1234567
```

```
UPDATE students
   SET age = age + 1
```

Updates from Another Table

General form:

```
UPDATE R
   SET a.attribute = b.attribute
  FROM a JOIN b
    ON <conditions>
 WHERE <conditions>
```

Example:

```
UPDATE accident
   SET a.roadseg = r.segmentid
  FROM accident AS a JOIN road AS r
    ON a.roadnumber = r.roadnumber
       AND a.milepost BETWEEN r.begmp AND r.endmp
```

Constraints in SQL

A constraint is a relationship among data elements that the DBMS is required to enforce.

The system will enforce the constraint by taking some actions:

- forbid an update, or
- perform compensating updates

Constraints in SQL

Different types of constraints:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints

The more complex the constraint, the harder it is to check and to enforce.

Define the Primary Key

The following two queries are equal:

```
CREATE TABLE Person(
    name      VARCHAR(100),
    ssn       INT PRIMARY KEY,
    age       SMALLINT,
    city      VARCHAR(30),
    gender    CHAR(1),
    birthdate DATE
)
```

```
CREATE TABLE Person(
    name      VARCHAR(100),
    ssn       INT,
    age       SMALLINT,
    city      VARCHAR(30),
    gender    CHAR(1),
    birthdate DATE,
    PRIMARY KEY (ssn)
)
```

Define the Primary Key

Define the multi-attribute key:

```
CREATE TABLE Person(
    firstname VARCHAR(100),
    lastname  VARCHAR(100),
    ssn       INT,
    age       SMALLINT,
    city      VARCHAR(30),
    gender    CHAR(1),
    birthdate DATE,
    PRIMARY KEY (firstname, lastname)
)
```

Uniqueness Constraints

Uniqueness constraint for other candidate keys:

```
CREATE TABLE Person(
    firstname VARCHAR(100),
    lastname  VARCHAR(100),
    ssn       INT,
    age       SMALLINT,
    city      VARCHAR(30),
    gender    CHAR(1),
    birthdate DATE,
    PRIMARY KEY (firstname, lastname),
    UNIQUE (ssn)
)
```

Foreign Key Constraints

A foreign key (FK) is a column or combination of columns used to establish and enforce a link between the data in two tables.

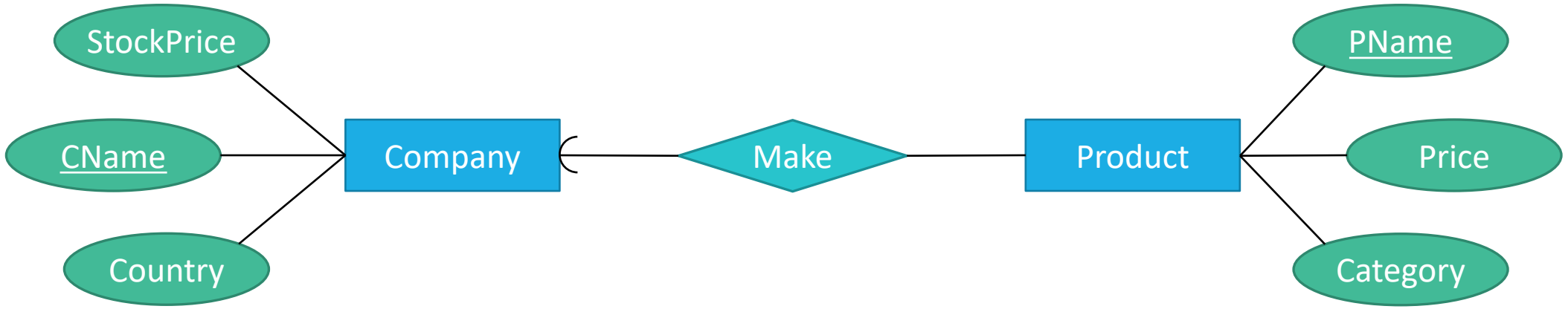
A link is created between two tables by adding one table's primary key (or unique) values to the other table. This becomes a foreign key in the second table.

What generates a foreign key when converting an E/R diagram to relational schema?

Referential Integrity Constraint, not a many-one relationship.

Foreign Key Constraints

Example:



Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

Foreign Key

PName	Price	Category	CName
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

Foreign Key Constraints

Declare a foreign key:

```
CREATE TABLE Product(
    Pname    CHAR(30) PRIMARY KEY,
    Category CHAR(30),
    Price    FLOAT,
    CName    CHAR(30) REFERENCES Company(CName)
)
```

What can we infer from the above table definition?

- CName is a foreign key in Product to Company(CName)
- CName must be a key in Product, but not necessarily the primary key.

Foreign Key Constraints

Another way of declaring a foreign key:

```
CREATE TABLE Product(
    Pname    CHAR(30) PRIMARY KEY,
    Category CHAR(30),
    Price    FLOAT,
    CName    CHAR(30),
    FOREIGN KEY (Cname) REFERENCES Company(CName)
)
```

Foreign Key Constraints

What happens during updates?

Two violations are possible:

- An insert or update to Product introduces values not found in Company.
- A deletion or update to Company causes some tuples of Product to "dangle."

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

PName	Price	Category	CName
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

"dangling tuples" = tuples that do not join with anything

Foreign Key Constraints

What happens during updates?

An insert or update to Product that introduces a nonexistent Company must be rejected.

A deletion or update to Company table that removes a CName found in some tuples of Product can be handled in three ways:

- Default: Reject the modification
- Cascade: Make the same changes in Product
  - Deleted CName: delete corresponding Product
  - Updated CName: change value in Product
- Set NULL: Change the CName in Product to NULL

What Happens During Updates?

- CASCADE independently for deletions and update. When we declare a foreign key, we may choose policies from SET NULL to CASCADE.

- Follow the foreign-key declaration by:

```
ON [UPDATE, DELETE][SET NULL, CASCADE]
```

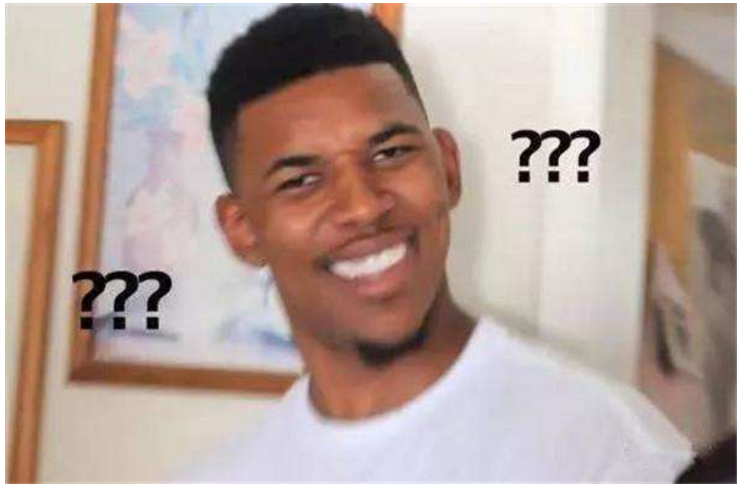
- Two such clauses may be used, one for update and one for delete.
- Otherwise, the default (reject) is used.

What Happens During Updates?

SQL Example:

```
CREATE TABLE Product(
    PName    CHAR(30) PRIMARY KEY,
    Category CHAR(30),
    Price    FLOAT,
    CName    CHAR(30),
    FOREIGN KEY (CName) REFERENCES Company(CName)
    ON DELETE SET NULL
    ON UPDATE CASCADE
)
```

SQL Practice This Friday



Concepts (abstract)

Practice (concrete)

Like any new interface or language…

Practice is the best way to learn.

Test Preparation

- Remind you that we have a test on Wednesday next week
- Very specific SQL syntax questions, such as:
- SQL Example:

  Write a query based on the given schema that returns the total number of sales of products in the chocolate category made by each salesperson. Include in your results only the salespeople that have sold at least 9 items

Test Preparation

- E/R diagrams – Interpret
- Converting E/R diagrams to relational schemas
- Conceptual questions, it is recommended that you have completed the assigned readings
- Definitions covered in the class slides, including data models, normal forms, database elements, design issues.

Tips for Success

- Write SQL code!!
- Make sure you go through the lectures on SQL and know about all of the topics and functions we covered.
- Complete the readings (only two).
- Go through the presentations.
- You are allowed to bring one double sided cheat sheet.

Other Notes About the Test

- 3 Sections: multiple choice; T/F; short answer.
- No electronic devices other than calculators (no phones).
- Graduate tests will be larger, NO extra credit for undergrads who complete the larger version.

Sample Questions

Which best describes **Atomicity** in the ACID properties of database transactions?

a) Database constraints and rules are not violated

b) The result of concurrent operations are the same as if transactions were executed serially

c) That committed transactions are permanently stored to disk

d) That a transaction must be completed in entirety or not at all

Sample Questions

Vehicles

Make	Model	Year
Ford	F150	1999
Dodge	Dart	2013
Dodge	Neon	1996
Honda	Accord	1989
Toyota	Prius	2002
Honda	Accord	2014

Manufacturers

Name	City	Country
Ford	Detroit, MI	USA
Dodge	Auburn Hills, MI	USA
Honda	Minato, Tokyo	Japan
Toyota	Toyota, Aichi	Japan

- Given the relations above, what is the result of the following query?

```
SELECT make, COUNT(*)
  FROM vehicles JOIN manufacturers
    ON vehicles.make = manufacturers.name
 WHERE manufacturers.country = 'USA'
 GROUP BY make
HAVING COUNT(*) > 2
```

Write queries to answer:

What is the average capacity of the stadium where players who make over $10 million play?

What stadiums are not associated with any teams in the Teams relation?

Stadiums

Name	City	State	Capacity	SID
Sports Authority	Denver	Colorado	77160	1001
LP Field	Nashville	Tennessee	69143	1002
CenturyLink Field	Seattle	Washington	72000	1003
TCF Bank Stadium	Minneapolis	Minnesota	52525	1007
Soldier Field	Chicago	Illinois	62871	1004

Players

Name	Salary	Height	Weight	Team
Sidney Rice	8500000	6.33	202	Seahawks
Peyton Manning	15000000	6.42	230	Broncos
Champ Baily	9500000	6	192	Broncos
Russel Okung	7060000	6.42	310	Seahawks
Wesley Woodward	3000000	6	233	Titans
Jared Allan	14280612	6	270	Vikings

Teams

Name	SID	Record	AvgTicket
Broncos	1001	13-3	89
Seahawks	1003	13-3	81
Titans	1002	7-9	67
Vikings	1007	5-10	89