ITCS393 Database Systems Lab

Lab08: Nested Query

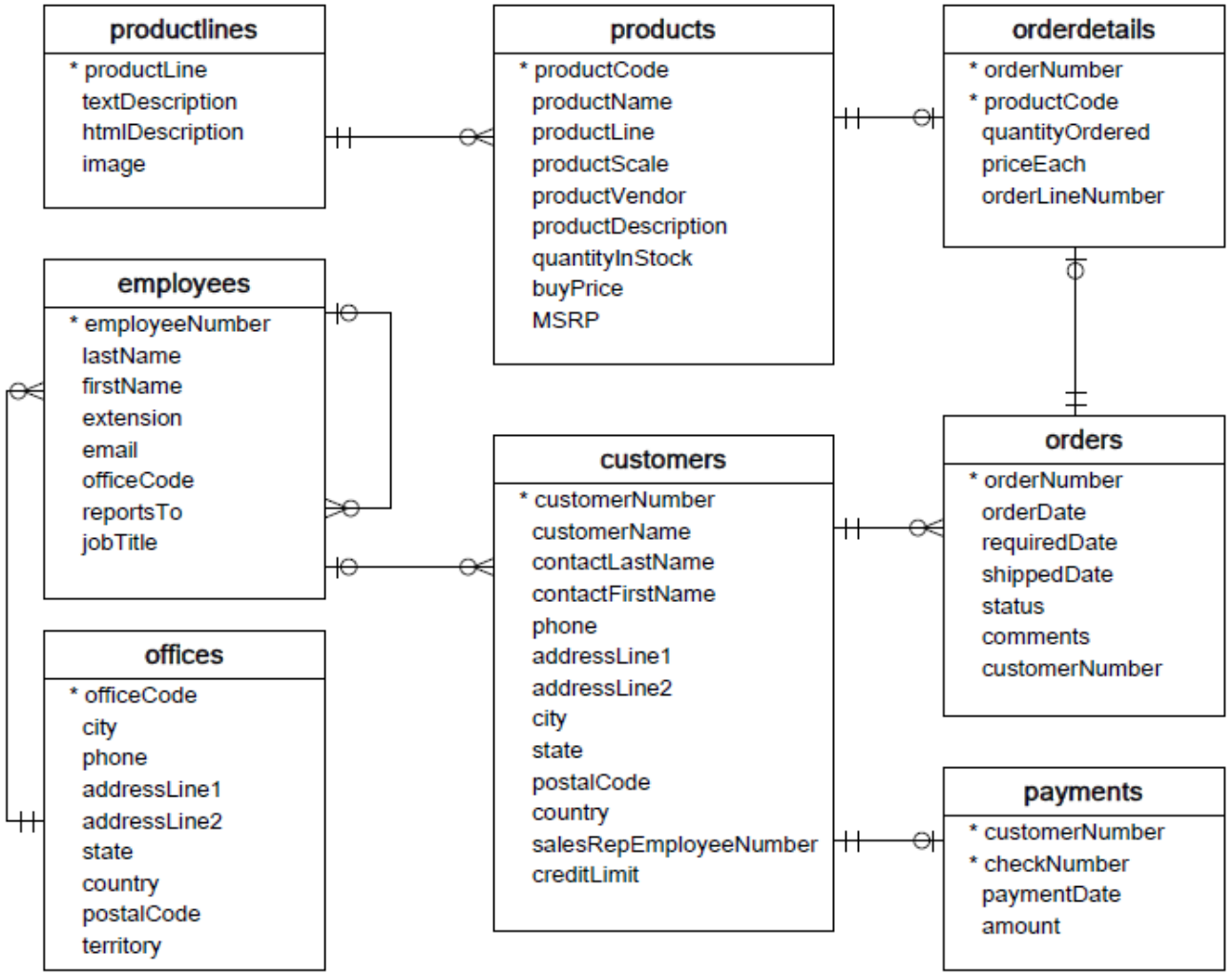
Dr. Petch Sajjacholapunt
Dr. Wudhichart Sawangphol
Dr. Jidapa Kraisangka

jidapa.kra@mahidol.edu

Class Objectives

- To learn how to apply nested query: self-contained subqueries and correlated subqueries.
- To learn how to use EXISTS predicate with subqueries and compare it with IN operators.
- To learn how to use subquries with INSERT, UPDATE, and DELETE statement

Database: classicmodels



- **Customers**: stores customer's data.
- **Products**: stores a list of scale model cars.
- **ProductLines**: stores a list of product line categories.
- **Orders**: stores sales orders placed by customers.
- **OrderDetails**: stores sales order line items for each sales order.
- **Payments**: stores payments made by customers based on their accounts.
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data.

Original ERD and SQL from https://www.mysqltutorial.org/mysql-sample-database.aspx

Subquery (or Nested Query)

- Subquery: a query inside another query by using parentheses to create an outer and an inner query
  - The inner query is executed first.
  - The output of an inner query is used as the input for the outer query.
- Subqueries can return single values or tables (with one or many rows and columns).

```
-- Outer query
SELECT …
FROM
      -- Inner query
      SELECT …
      FROM …
```

Self-contained Subquery

- The subquery which is completely independent and do not require any input from outer query.
- Self-Contained subquery is evaluated once for the outer query and its result is used with all records produced by outer query.

Example:

```
SELECT customerName, checkNumber, amount
FROM  Customers c INNER JOIN Payments p
      ON c.customerNumber = p.customerNumber
WHERE amount=(SELECT MAX(amount) FROM Payments);
```

Correlated Subquery

- Fully dependent on the outer query
- Require an input from its outer query.
- Cannot be executed without required attributes from outer query.
- This behavior increases the cost of the execution of the subquery as it needs to be executed for each of row of outer query

Example

```
SELECT *
FROM Employees e
WHERE NOT EXISTS
        (SELECT employeeNumber FROM Employees b
         WHERE b.employeeNumber = e.reportsTo)
```

Example of Subquery

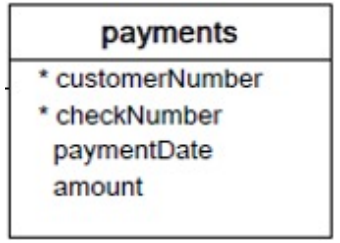
- You can include a subquery in
  - the SELECT clause, to specify a certain column.
  - the FROM clause, to specify a new table
  - the WHERE clause, to filter data.
  - the HAVING clause, as a group selector

Subquery in SELECT

EX1: Show the payment information and its maximum payment amount by the day of week (Sun, Mon, … , Sat)

Task
- T1: Calculate the maximum amount of perment per the weekday
- T2: Show payment info + the maximum amount per the weekday

Example



Payment information

Day of Week of the Payment Date

customerNumber	checkNumber	paymentDate	amount	dayOfWeek	maxOftheDay
103	HQ336336	2004-10-19	6066.78	Tuesday	63843.55
103	JM555205	2003-06-05	14571.44	Thursday	53959.21
103	OM314933	2004-12-18	1676.14	Saturday	101244.59
112	BO864823	2004-12-17	14191.12	Friday	120166.58
112	HQ55022	2003-06-06	32641.98	Friday	120166.58
112	ND748579	2004-08-20	33347.88	Friday	120166.58
114	GG31455	2003-05-20	45864.03	Tuesday	63843.55
114	MA765515	2004-12-15	82261.22	Wednesday	85024.46
114	NP603840	2003-05-31	7565.08	Saturday	101244.59

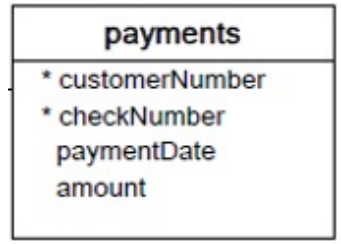
Max of the Day (e.g. Tue, Wed, …)

Subquery in SELECT

EX1: Show the payment information and its maximum payment amount by the day of week (Sun, Mon, … , Sat)

Task – T1: Calculate the maximum amount of perment per the weekday

```
SELECT DAYNAME(t1.paymentDate), MAX(amount)
FROM   Payments t1
GROUP BY 1;
```



DAYNAME(t1.paymentDate)	MAX(amount)
Tuesday	63843.55
Thursday	53959.21
Saturday	101244.59
Friday	120166.58
Wednesday	85024.46
Sunday	52151.81
Monday	85559.12

Note: this is the temporary result that is need to be related with T2

What is the condition to related them?

```
DAYNAME(t2.paymentDate) = DAYNAME(t1.paymentDate)
```

Subquery in SELECT

EX1: Show the payment information and its maximum payment amount by the day of week (Sun, Mon, … , Sat)

Task – T2: Show payment info + the maximum amount per the weekday

```
SELECT *, DAYNAME(t2.paymentDate) AS dayofWeek,
        (SELECT MAX(amount)
         FROM Payments t1
         WHERE DAYNAME(t1.paymentDate)= DAYNAME(t2.paymentDate)
         GROUP BY DAYNAME(t1.paymentDate) ) AS maxOftheDay
FROM Payments t2;
```

customerNumber	checkNumber	paymentDate	amount	dayOfWeek	maxOftheDay
103	HQ336336	2004-10-19	6066.78	Tuesday	63843.55
103	JM555205	2003-06-05	14571.44	Thursday	53959.21
103	OM314933	2004-12-18	1676.14	Saturday	101244.59
112	BO864823	2004-12-17	14191.12	Friday	120166.58
112	HQ55022	2003-06-06	32641.98	Friday	120166.58
112	ND748579	2004-08-20	33347.88	Friday	120166.58
114	GG31455	2003-05-20	45864.03	Tuesday	63843.55
114	MA765515	2004-12-15	82261.22	Wednesday	85024.46
114	NP603840	2003-05-31	7565.08	Saturday	101244.59
114	NR27552	2004-03-10	44894.74	Wednesday	85024.46

Note: The payment date's day of week from the outer query (`t2`) must determine the inner query (`t1`) to identify which `` `maxOfTheDay` `` to be displayed.

Subquery in `FROM`

EX2: What is the average of the total number of orders per months?

Task

T1: Calculate the total number of orders per month (COUNT)

T2: Find the average of those counts

Subquery in FROM

EX2: What is the average of the total number of orders per months?

Task

T1: Calculate the total number of orders per month (COUNT)

```
SELECT MONTHNAME(orderDate) AS `Month`,
COUNT(orderNumber) AS orNum
FROM Orders
GROUP BY 1;
```

Month	orNumsc
January	25
February	26
March	27
April	29
May	29
June	19
July	18
August	17
September	20
October	31
November	63
December	22

T2: Find the average of those counts

Subquery in FROM

EX2: What is the average of the total number of orders per months?

Task

T1: Calculate the total number of orders per month (COUNT)

```
SELECT MONTHNAME(orderDate) AS `Month`,
COUNT(orderNumber) AS orNum
FROM Orders
GROUP BY 1;
```

Month	orNumsc
January	25
February	26
March	27
April	29
May	29
June	19
July	18
August	17
September	20
October	31
November	63
December	22

T2: Find the average of those counts

```
SELECT AVG(o.orNum) AS avg. -- T2
FROM ( SELECT MONTHNAME(orderDate) AS, COUNT(orderNumber) AS orNum
       FROM Orders
       GROUP BY 1. -- T1) o;
```

avg
27.1667

Note: When creating a subquery as table, DBMS requires **an alias** of the table name

Subquery in WHERE

EX3: What is the customer name who paid the highest amounts of check?

Task

T1: Calculate the highest amount of check

```
SELECT MAX(amount)
FROM Payments);
```

MAX(amount)
120166.58

T2: Retrieve the customer name

```
SELECT customerName, checkNumber, amount
FROM    Customers c INNER JOIN Payments p
        ON c.customerNumber = p.customerNumber
WHERE amount =
        (SELECT MAX(amount) FROM Payments);
```

customerName	checkNumber	amount
Euro+ Shopping Channel	JE105477	120166.58

Subquery in WHERE

EX4: List customer name who are not located in the same country as any offices

Task

T1: Retrieve country lists of the offices

```
SELECT DISTINCT(country)
FROM Offices;
```

country
USA
France
Japan
Australia
UK

T2: Retrieve the customer name who are not in the T1 country list

```
SELECT customerName, city, country
FROM Customers c
WHERE country NOT IN
        (SELECT DISTINCT(country) FROM Offices);
```

customerName	city	country
Baane Mini Imports	Stavern	Norway
Havel & Zbyszek Co	Warszawa	Poland
Blauer See Auto, Co.	Frankfurt	Germany
Euro+ Shopping Channel	Madrid	Spain
Volvo Model Replicas, Co	Luleå	Sweden
Danish Wholesale Imports	Kobenhavn	Denmark
Dragon Souveniers, Ltd.	Singapore	Singapore
Handji Gifts& Co	Singapore	Singapore

Subquery in HAVING

EX5: List the month in the year 2005 that has the average amount of payment greater than the average amount of all payments in the year 2004

Task

T1: Calculate the average payment amount in the year 2004

```
SELECT AVG(amount)
FROM Payments
WHERE YEAR(paymentDate) = 2004
```

AVG(amount)
31715.648897

T2: Compare the monthly average payment of year 2005 with the avg amount from T1

```
SELECT MONTHNAME(paymentDate) AS `Month`, AVG(amount)
FROM Payments
WHERE YEAR(paymentDate) = 2005
GROUP BY 1      -- Group by the first column
HAVING AVG(amount) >
        (SELECT AVG(amount) FROM Payments
         WHERE YEAR(paymentDate) = 2004);
```

Month	AVG(amount)
March	48158.511250
April	36779.544000

EXISTS

- The EXISTS operator is used to test for the existence of any record in a subquery.
- Return TRUE if the subquery returns one or more records

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS -- Use with the subquery only
      (SELECT column_name FROM table_name WHERE condition);
```

Example: EXISTS

EX6: Find the customer name who are in the same city at the offices

```
SELECT customerName, city, country
FROM Customers c
WHERE EXISTS
      (SELECT DISTINCT city, country
       FROM Offices o
       WHERE c.country=o.country
             AND c.city = o.city);
```

customerName	city	country
Mini Wheels Co.	San Francisco	USA
Land of Toys Inc.	NYC	USA
Muscle Machine Inc	NYC	USA
La Corne D'abondance, Co.	Paris	France
Vitachrome Inc.	NYC	USA
Lyon Souveniers	Paris	France
Corporate Gift Ideas Co.	San Francisco	USA
Stylish Desk Decors, Co.	London	UK
Gifts4AllAges.com	Boston	USA
Auto Canal+ Petit	Paris	France
Classic Legends Inc.	NYC	USA
Microscale Inc.	NYC	USA
Double Decker Gift Stores, Ltd	London	UK
Diecast Collectables	Boston	USA

EXISTS **VS** IN

- IN operator will scan all records fetched from the inner query. On the other hand, the EXISTS operators will stop the scanning process as soon as it found a match

```
SELECT customerName, city, country
FROM Customers c
WHERE EXISTS ( SELECT DISTINCT city, country FROM Offices o
               WHERE c.country = o.country AND c.city = o.city);
```

```
SELECT customerName, city, country
FROM Customers c
WHERE (city,country) IN (SELECT DISTINCT city, country FROM Offices o);
```

- For the small queries, EXISTS and IN are not different in performance. However, the EXISTS clause is much faster than IN when the subquery results is very large.

Subquery with DML

- Subqueries can also be used with the following statements along with (the operators like `=, <, >, >=, <=, IN, BETWEEN`)
  - `INSERT`
  - `UPDATE`
  - `DELETE`

Subquery with INSERT

INSERT clause with SELECT allows to add the result of the query into a table

Example

Suppose we have created the table `Locations` for storing city, state, and country. The following code inserts a new data from the given query/subquery.

Table: locations

Columns:

locationID	smallint AI PK
city	varchar(50)
state	varchar(50)
country	varchar(50)

```
INSERT INTO Locations(city, state, country)
SELECT DISTINCT city, state, country
FROM Customers
WHERE (city, state, country) NOT IN
      (SELECT DISTINCT city, state, country
       FROM Offices) ;
```

locationID	city	state	country
1	Nantes	NULL	France
2	Las Vegas	NV	USA
3	Melbourne	Victoria	Australia
4	Stavern	NULL	Norway
5	San Rafael	CA	USA
6	Warszawa	NULL	Poland

```
91 row(s) affected Records: 91  Duplicates: 0  Warnings: 0
```

Subquery with UPDATE

Example: Update the state to have the same values as the city for the country without states.

```
UPDATE Locations L1
SET state =
        (SELECT DISTINCT L2.city
         FROM Customers L2
         WHERE  L1.city=L2.city
         AND L1.country=L2.country)
WHERE state IS NULL;
```

locationID	city	state	country
1	Nantes	Nantes	France
2	Las Vegas	NV	USA
3	Melbourne	Victoria	Australia
4	Stavern	Stavern	Norway
5	San Rafael	CA	USA
6	Warszawa	Warszawa	Poland
7	Frankfurt	Frankfurt	Germany
8	Madrid	Madrid	Spain
9	Luleå	Luleå	Sweden
10	Kobenhavn	Kobenhavn	Denmark
11	Lyon	Lyon	France
12	Singapore	Singapore	Singapore
13	Allentown	PA	USA
14	Burlingame	CA	USA

```
57 row(s) affected Rows matched: 57  Changed: 57  Warnings: 0
```

Subquery with DELETE

Example: Remove the locations that are outside the country having the offices

```
DELETE From Locations
WHERE (country) NOT IN
        (SELECT DISTINCT country
         FROM Offices);
```

```
52 row(s) affected, 39 rows left
```

locationID	city	state	country
1	Nantes	Nantes	France
2	Las Vegas	NV	USA
3	Melbourne	Victoria	Australia
5	San Rafael	CA	USA
11	Lyon	Lyon	France
13	Allentown	PA	USA
14	Burlingame	CA	USA
16	New Haven	CT	USA
18	Lille	Lille	France
19	Cambridge	MA	USA
20	Bridgewater	CT	USA
21	Kita-ku	Osaka	Japan
23	Manchester	Manc…	UK
25	Brickhaven	MA	USA
26	Liverpool	Liverp…	UK
28	Pasadena	CA	USA
29	Strasbourg	Stras…	France
32	Glendale	CA	USA