Subqueries

MAD 202

Relational Databases

- The key component of databases is being able to optimize the data and reduce errors and redundancy
- We create tables that hold information that is related
- The relationship between one table and another is represented by a foreign key
- A foreign key is another table's primary key
  - It allows one columns value to provide a lookup in another table to retrieve additional information.

Relationships

Foreign Key

Product Table

id	productName	price	supplierID	dateReceived
1	Bananas	3.07	7	2020-03-31
2	Wasabi Paste	6.68	8	2021-03-04
3	Brandy - Orange, Mc Guiness	4.16	3	2020-05-02
4	Amaretto	6.18	5	2020-09-23
5	Pork - Back, Short Cut, Boneless	10.80	8	2020-12-15
6	Red Pepper Paste	2.02	4	2020-07-02
7				
8				
9				
10				

Primary Key

id	companyName	purchasingManager	streetNumber	streetName	province	phoneNumber
1	Rodriguez LLC	Camile Nern	19	Judy	Québec	(768) 4199250
2	Powlowski Group	Mar Liff	1	Pennsylvania	Québec	(140) 2190327
3	Fay-Bernier	Nye McCusker	445	Mosinee	Québec	(657) 6159685
4	Marvin-Jacobs	Merrill Rydings	8	Blaine	Québec	(125) 5495312
5	Senger, Price and Hudson	Barde Adamovicz	0585	Buell	Québec	(880) 6080511
6	Morissette Group	Julia Hampshaw	7	Armistice	Québec	(224) 1163200
7	Hartmann, Dach and Emmerich	Olva Chotty	04260	Everett	Québec	(427) 2176481
8	Spinka Inc	Gratiana Goddert.sf	20192	Twin Pines	Québec	(402) 9372969
9	Moore-Russel	Othilia Glennie	903	Westend	Québec	(814) 6824989
10	Cummerata, Hagenes and Hickle	Kaylee Plumb	47	5th	Québec	(781) 1480137
11	Thompson Inc	Paule Voden	3642	Crownhardt	Ontario	(978) 7905853
12	Mertz, Murray and Hickle	Byrom Quinet	032	Lotheville	Québec	(638) 2442183

Supplier Table

Subqueries

- With subqueries, we use the results of one query – to filter another table's results.
- What if we wanted to know what supplier supplied us with Bread?
- We query one table (products) to get all of the suppliers that provide us with bread

```
SELECT supplierID
FROM products
WHERE productName LIKE '%bread%';
```

supplierID
8
2
6

Subqueries

- Important things to note – we asked for a single column of values. Just the supplierID

supplierID
8
2
6

- Do not create a query that returns multiple columns of values

```
SELECT supplierID, productName
FROM products
WHERE productName LIKE '%bread%';
```

Subqueries

- To ensure that you do not have duplicates – use DISTINCT

```
SELECT DISTINCT supplierID
FROM products
WHERE productName LIKE '%bread%';
```

Subqueries

- We could now use those ID's to look up the list of suppliers that have those specific companies

```
SELECT *
FROM suppliers
WHERE id IN (8,2,6);
```

supplierID
2
6
8

id	companyName	purchasingManager	streetNumber	streetName	province	phoneNumber
2	Powlowski Group	Mar Liff	1	Pennsylvania	Manitoba	(140) 2190327
6	Morissette Group	Julia Hampshaw	7	Armistice	Québec	(224) 1163200
8	Spinka Inc	Gratiana Goddert.sf	20192	Twin Pines	Ontario	(402) 9372969

Subqueries

- Subqueries simplify the process of creating two statements.

```
SELECT *
FROM suppliers
WHERE id IN ( SELECT DISTINCT supplierID
        FROM products
        WHERE productName LIKE '%bread%' );
```

id	companyName	purchasingManager	streetNumber	streetName	province	phoneNumber
2	Powlowski Group	Mar Liff	1	Pennsylvania	Manitoba	(140) 2190327
6	Morissette Group	Julia Hampshaw	7	Armistice	Québec	(224) 1163200
8	Spinka Inc	Gratiana Goddert.sf	20192	Twin Pines	Ontario	(402) 9372969

Testing Membership

- The IN keyword tests to see if the value is comparable to a ***series of values***
- Can also use the keyword NOT to reverse the results of the query

```
SELECT *
FROM suppliers
WHERE id IN ( SELECT DISTINCT supplierID
    FROM products
    WHERE productName LIKE '%bread%' );
```

This query returns more than a single value

Nesting subqueries

- A subquery can include a series of subqueries.
- The nested subqueries follow the same rules as a single sub query

Subquery Syntax

- Follows the same syntax as a normal SELECT statement with the following differences:
  - You can nest a subquery in a SELECT, FROM, WHERE, or HAVING clause
  - ALWAYS enclose a subquery in parentheses ( )
  - Do **NOT** terminate a subquery with a ;

Subquery Syntax

- Do **NOT** put an ORDER BY clause in a subquery (this has no value since you are returning intermediate information, so order is unimportant)
- Tables must be included in both the queries in order to have its columns appear in the final result
- The subquery may return an empty table

Subqueries and Aggregate Functions

- Subqueries are excellent for use with aggregate functions
- What is the cheapest product we have?

```
SELECT *
FROM products
WHERE price IN (SELECT  MIN(price) FROM products);
```

id	productName	price	supplierID	dateReceived
57	Chicken Breast Wing On	1.14	4	2020-07-30

Simple subqueries

- **Inner subquery** – the nested query
- **Outer subquery** – the statement containing the subquery
- A simple subquery evaluates the inner subquery **once** and provides its result into the outer query

```
SELECT *
FROM products
WHERE price in (SELECT MAX(price) FROM products)
```

Process this first – returns result and passes to the outer query

MAX(price)
12.95

id	productName	price	supplierID	dateReceived
39	Urban Zen Drinks	12.95	4	2020-04-14

Can also check for values that are NOT IN....

Select the products that do not come from Ontario.

```
SELECT *
FROM products
WHERE supplierID NOT IN (
        SELECT id
        FROM suppliers
        WHERE province = 'Ontario');
```

Qualifying column names

- Can explicitly declare column names in a subquery:

```
SELECT pub_name
FROM publishers
WHERE publishers.pub_id IN
(SELECT titles.pub_id
FROM titles WHERE type='biography)
```

The reason for qualifying is that the column name exists in both tables

Implicit assumptions are being made in the above case so explicitly declaring column names is not required – column names are qualified implicitly by the table referenced in the FROM clause at the same nesting level.

Subqueries as Column Expressions

- Subqueries that are used as column expressions MUST return a single value - (scalar subquery)
- This is accomplished through **aggregate functions** (these return a sum, an average, etc) or very restrictive WHERE conditions
- The subquery is placed in the position of a column expression in the SELECT clause
- ***LIST each product, the price and the average price***

```
SELECT productName, price, (SELECT AVG(price) FROM products)
FROM products;
```

Comparing a Subquery

- Subqueries can be used as filters in a WHERE or HAVING clause using any of the comparison operators (=,<>,<, etc)
- Rules:
  - The SELECT clause can only include a single expression or column name:
    - SELECT title FROM…
    - SELECT title, pages FROM….
  - The compared values must have the same data type or be implicitly convertible (string value to string value)
  - **The subquery MUST only return a single value**
    - Use an aggregate
    - Using a join with the outer query based on a key always returns a single value

Comparing a subquery

- ***Show the customers that live in the same province as Thompson Inc is located in***

Make note of the equal

```
SELECT * FROM customers
WHERE province =
(SELECT province
FROM suppliers WHERE companyName = 'Thompson Inc');
```

This only returns a SINGLE value