# Subqueries

* **What is a subquery?**
  + They are nested queries.
  + Sometimes you need to create a query that is based on the results or another query.
  + A nested query consists of a main query and one or more subqueries. The **main query** is the first query that appears in the SELECT command. A **subquery** retrieves values that the main query's search condition must match.
  + In a nested subquery, the DBMS executes the subquery first then the main query
  + Subqueries can be inside the WHERE or HAVING clause of another SELECT statement
  + They can also be used as search conditions in INSERT, UPDATE or DELETE statement
  + We will never put them in the FROM or SELECT sections. This is usually the wrong thing to do and is outside the scope of this class.
* **Syntax:**

Beginning of outer

SELECT statement

SELECT [DISTINCT] select\_list

FROM table\_list

WHERE

{expression { [NOT] IN | comparison\_operator [ANY | ALL]} | [NOT]EXISTS}

**(SELECT [DISTINCT] subquery\_select\_list**

**FROM table\_list**

Subquery, enclosed in parentheses

**WHERE conditions**

**[GROUP BY group\_by\_list]**

**[HAVING conditions]) – cannot have order by in the subquery**

[GROUP BY group\_by\_list]

Optional – rest of outer SELECT statement

[HAVING conditions][(Subquery here too maybe)]

[ORDER BY order\_by\_list]

* Subquery operators:
  + IN/NOT IN
  + Relational operator { = | != | > | >= | < | <= } [ ANY | ALL ]
  + EXISTS/NOT EXISTS – used almost exclusively with correlated subqueries
* **Types:**
  + **Simple subqueries** or non-correlated subqueries – the inner query is executed first and the outer query takes an action based on the results of the inner query
  + **Subquery Returning a Single Value**
    - **Example 0:**
      * Find all the Authors that work in the same city as Livia Karsen
      * Steps:
        1. Find the city Livia works in
        2. Send the name of the city to another query to find all the people who work in 'Oakland'
        3. Put the 2 together
  + **Subquery Retuning Multiple Values**
    - **Example 1:** 
      * Find the names of all publishers that publish business books.
      * Steps:
        1. Where is the type of book stored?

In the type field in the Title table

now find all the books that are of this type

* + - * 1. Look at the query. What could I use to find out the names of the publishers?
        2. Lets create the query to find the publishers names
        3. Put the 2 together
        4. Looking at it another way:

SELECT \*

FROM Publisher

WHERE pub\_id IN

(SELECT pub\_id

FROM Titles

WHERE type = ‘business’);

Inner query is independent, gets evaluated first, and passes results to outer query.

* + - **Example 2:** 
      * List the books published by Binnet & Hardley that have a year to date sales of more than $15,000.
      * Steps:
        1. Need to find Binnet & Hardley's pubID?
        2. Now use that in the main query
        3. Put the 2 together
    - **Example 3:** 
      * List the books that have a price greater or equal to the price of the book Straight Talk About Computers - titleID is BU7832.
      * Steps:
        1. Find "Straight Talk About Computers" price.
        2. Find all the books with a price of >= 29.99
    - **Example 4:** 
      * What if I didn't know that BU7832 was the titleID of Straight Talk About Computers?
      * Change the subquery in Example 3
    - **Example 5:** 
      * List the books that have a price less than the average price of all books.
      * Steps:
        1. Find the average book price.
        2. Find the books with a price less than the average
    - **Example 6:** 
      * Give the names of the authors that live in states where more than one Author is listed?
      * Steps:
        1. Find how many Authors are in each state.
        2. Find which are greater than 1
        3. Get the names of the authors that live in those states
  + **Correlated subqueries**
    - The outer query provides values that the inner query uses in its evaluation.
    - Within a correlated subquery the subquery results are integrated with the outer query on a row-by-row basis, allowing the comparison to be even more dynamic.
    - We did this as a simple subquery.

**Example 1:**

* + **Correlated subqueries**
    - The outer query provides values that the inner query uses in its evaluation.
      * Display a list of publisher's names that have business books?

SELECT pubName

FROM Publisher p

WHERE EXISTS

(SELECT \*

FROM Title t

WHERE t.pubID = p.pubID

AND type = ‘business’);

Inner query needs pub\_id values from outer query, passes results to outer query.

**EXISTS is almost exclusively used with correlate subqueries.**

## Correlated subquery steps:

1. The outer query finds the first name in the publishers table (for example, New Age Books)

SELECT pubName, pubID

FROM Publisher p;

- which is 0736

1. The inner query joins the associated Publishers.pub\_id (0736) to Titles.pubID to find qualifying rows in the Titles table (it finds six).

SELECT title

FROM Title

WHERE pubID = '0736';

Only one of the six is a business book.

SELECT title

FROM Titles

WHERE pubID = ‘0736’

AND type = ‘business’;

- one row matches, so this publisher will be returned to be displayed in the outer query's results: New Age Books qualifies.

1. The outer join then goes to work again, this time passing the pub\_id of the second row in publishers (Binnet & Hardley, 0877) to the inner query.

SELECT pubID, title

FROM Title

WHERE pubID='0877' AND type='business';

- no rows selected, so nothing to return

1. Using this value, the inner query finds no rows with the correct type and signals “no rows found” to the outer query. Binnet & Hardley does not qualify.
2. The subquery runs the third time with a Publishers.pub\_id of 1389 (Algodata Infosystems)

SELECT pubID, title

FROM Title

WHERE pubID='0877' AND type='business';

- one row matches, so this publisher will be returned to be displayed in the outer query's results: Algodata Infosystems qualifies.

1. Of the 3 Publishers passed in to the inner query only 2 of them qualify and will be displayed in the result.

**Simple vs Correlated Performance Issues**

* + The simple subquery traversed the titles Table once.
  + The correlated subquery accesses the Titles table once for every qualified row in the Publishers table, or a total of 3 times.
  + In most cases it is more efficient to use simple subqueries.
  + However, if you need to look at the data row by row, a correlated subquery is the answer.

**Example 2:**

Retrieve the name, publisher ID, and ytdSales, of any book whose ytdSales is above average for that book's publisher.

**Example 3:**

Change the query in Example 2 to display the publisher's name instead of the ID.

**Joins or subqueries?**

* Many SQL solutions can be done a number of ways: as a join query, as a simple subquery, or as a correlated subquery.
* They can often be used interchangeably to solve a given problem
* Some SQL users prefer a subquery while other prefers a join.
* But each has advantages and disadvantages:
  + Subqueries useful when an intermediate result is needed for comparison
  + Joins useful when you want to display data from more than one table
  + Subquery Example: consider the problem of listing all books with prices equal to minimum book price.
    - Without a subquery, you’d do the job in two steps:

SELECT MIN(price)

FROM Titles;

* + - Take the result from this query and place in

SELECT title, price

FROM Titles

WHERE price = 12.99;

* + - Using a subquery it would only take one statement

SELECT title, price

FROM Titles

WHERE price =

(SELECT MIN(price)

FROM Titles);

* + - The ability to calculate an aggregate value on the fly and feed it back to the outer query for comparison is easier done with a subquery.
  + Join Example: Search for the names of publishers located in the same city as an author. The queries below both perform the same task.
    - What if you wanted to display the publisher’s name and the author’s name.

**Notes**

* + Usually the IN operator compares one column to one column. Oracle does have a syntax to allow comparison of multiple columns to multiple columns:

SELECT au\_lname, au\_fname

FROM authors

WHERE (city, state) IN

(SELECT city, state

FROM editors

WHERE ed\_lname = 'DeLongue');

* + What actually appears in the SELECT clause of a subquery used with EXISTS is pretty much irrelevant: we are just testing for the existence or non-existence of rows. The book suggests you use SELECT \* … – in other places, you’ll see things like SELECT 'x' …
  + Subquery results cannot be ordered. (No ORDER BY clause in the subquery.) Ordering of the internal rows is irrelevant.

Use the **7 step process** in creating your queries using joins

1. What columns do you want to display? (SELECT)
2. Identify the tables involved. (FROM)
3. Look at the join conditions and types of joins. (FROM & ON)
4. Look at any restrictions on the rows. (WHERE)  
   Look at any restrictions on the groups. (HAVING)
   1. Decide whether these restrictions should be derived from a subquery.
5. Is there any grouping required? (GROUP BY) – is distinct necessary?
6. Is there any ordering required? (ORDER BY)
7. Do you need to combine result sets (UNION, INTERSECT, MINUS).

**Example:** What publishers have published books that cost more than $35.00.

1. What columns do you want to display?
2. Identify the tables involved.
3. Look at the join conditions and types of joins.
4. Look at any restrictions on the rows.
5. Is there any grouping or ordering required?

**Solve as a Join**

**Solve as a Subquery**

Note: Tables listed in the outer FROM clause dictate which columns are available for display.

**More Examples:**

# 1. What is the name of the book with the highest price?

1. Write inner query first
2. Write outer query next

**2. Would like to know the names of the books that Eleanore Himmel edited.**

1. Identify the tables involved.
2. Look at the join conditions and types of joins.
3. Look at any restrictions on the rows.
4. What columns do you want to display?
5. Is there any grouping or ordering required?

As a join:

As a subquery - start with the inner most subquery

**Testing Nonexistence**

**Find the names of all the publishers that do not publish business books**

- Does not matter what you select – this example uses ‘t’ for True because it is evaluating each row to true or false and not anything it is returning.

SELECT pubName

FROM Publisher p

WHERE NOT EXISTS

(SELECT \*

FROM Title t

WHERE t.pubID = p.pubID

AND type = 'business');

Step

1. Display the publisher & type

2. Display the publisher & type

Doesn't matter what you use in the SELECT clause (SELECT 't')

We are checking for the rows existence or not; not the value in the row.

An other way to write this query. This way is very inefficient because for the number of rows in the table it has to process every row and count them

//count the number of books a business has, if the count is 0, then keep them, same as not exists

SELECT pub\_id, pub\_name

FROM Publishers P\_Out

WHERE 0 =

(SELECT count(\*)

FROM Titles t\_In

WHERE type = 'business'

AND t\_In.pub\_id = p\_Out.pub\_id);

**Subqueries using the IN operator**

The result of the inner subquery is a list of zero or more values.

Find the names of all second authors who live in California and receive less than 30 percent of the royalties on the books they coauthor.

SQL Solution Strategy:

* What needs to be determined?
* Identify the tables involved.
* What restrictions are there?
* Any ordering conditions?
* Any grouping conditions?
* How should the tables be joined?
* Look for statements that suggest a subquery. Try solving these queries in parts – the inner query, the outer query – then put them together.
* Look for cases that may require the DISTINCT clause or set operations (UNION, INTERSECT, MINUS).
* Work in small pieces – start with one table and gradually build to the full query