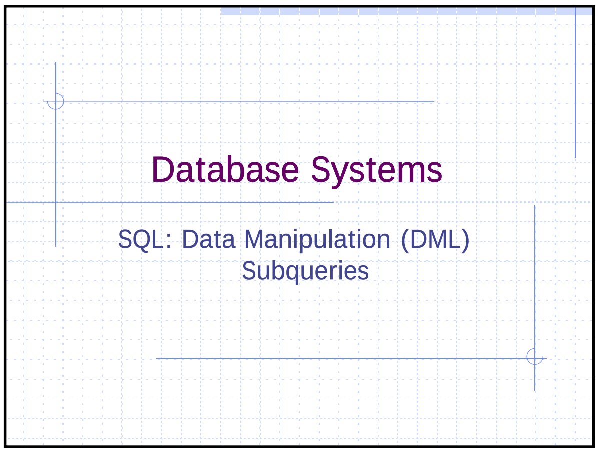
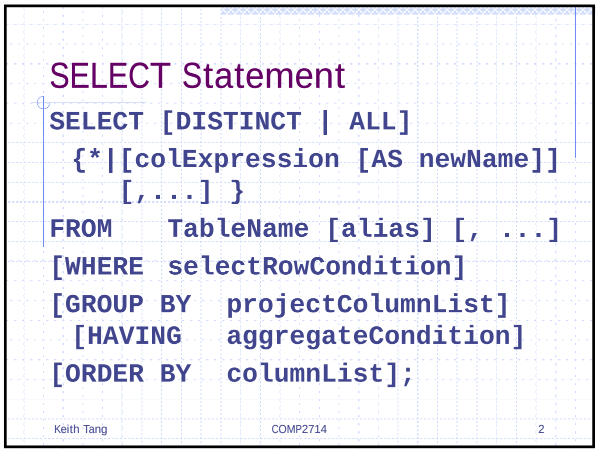


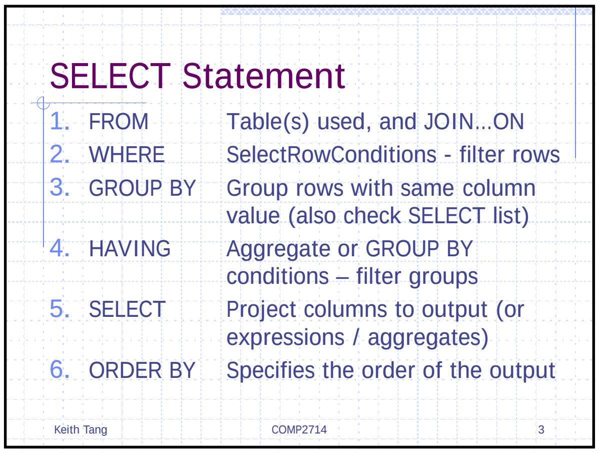
Database Systems

SQL: Data Manipulation (DML)
Subqueries



SELECT Statement

```
SELECT [DISTINCT | ALL]
 {*|[colExpression [AS newName]]
    [,...] }
FROM    TableName [alias] [, ...]
[WHERE  selectRowCondition]
[GROUP BY   projectColumnList]
 [HAVING    aggregateCondition]
[ORDER BY   columnList];
```



SELECT Statement

1.	FROM	Table(s) used, and JOIN...ON
2.	WHERE	SelectRowConditions - filter rows
3.	GROUP BY	Group rows with same column value (also check SELECT list)
4.	HAVING	Aggregate or GROUP BY conditions – filter groups
5.	SELECT	Project columns to output (or expressions / aggregates)
6.	ORDER BY	Specifies the order of the output

Books Database Schema

authors (au_id, au_fname, au_lname, phone, address, city, state, zip)

title_authors (title_id, au_id, au_order, royalty_share)

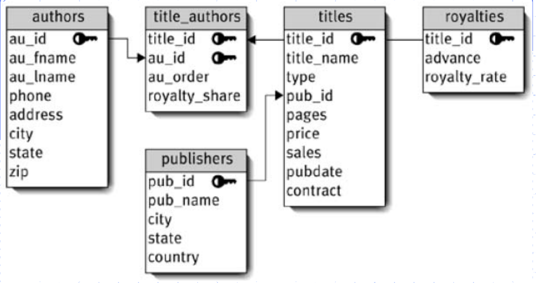
publishers (pub_id, pub_name, city, state, country)

royalties (title_id,advance,royalty_rate)

titles (title_id, title_name, type, pub_id, pages, price, sales, pubdate, contract)

http://www.fehily.com/books/SQL-Visual-QuickStart-Guide-3rd.html

Books Database Schema



Subqueries

- A SELECT embedded within a SELECT
- SQL-92 – Subqueries can be used:
  - SELECT clause
  - FROM clause
  - WHERE clause
  - HAVING clause
- Subqueries may also appear in INSERT, UPDATE, and DELETE
  - Most commonly used in WHERE clause

A Subquery Example

- Find publishers that publish 'biography' books:

-
  ```
  SELECT pub_id
  FROM titles
  WHERE type='biography';
  ```

```
pub_id
------
P01
P03
P01
P01
```

-
  ```
  SELECT pub_name
  FROM publishers
  WHERE pub_id IN
     ('P01','P03','P01','P01');
  ```

WHERE Subquery vs JOIN

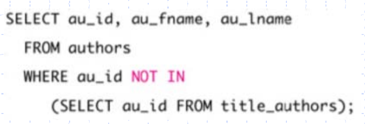
- Find publishers that publish 'biography' books:

-
  ```
  SELECT pub_name
  FROM publishers
  WHERE pub_id IN
    (SELECT pub_id    -- DISTINCT not req'd
     FROM titles
     WHERE type = 'biography');
  ```

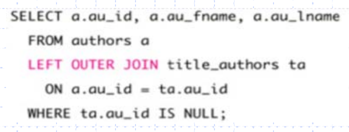
-
  ```
  SELECT pub_name, type
  FROM publishers p JOIN titles t
         ON p.pub_id = t.pub_id
  WHERE type = 'biography';
  ```
- Can use EXISTS instead of IN (more later)

WHERE Subquery vs JOIN

- Find authors who have not written a book:



```
SELECT au_id, au_fname, au_lname
  FROM authors
  WHERE au_id NOT IN
     (SELECT au_id FROM title_authors);
```



```
SELECT a.au_id, a.au_fname, a.au_lname
  FROM authors a
  LEFT OUTER JOIN title_authors ta
    ON a.au_id = ta.au_id
  WHERE ta.au_id IS NULL;
```

- Can also use NOT EXISTS instead of NOT IN

Common Subquery Types

- Scalar subquery : 1 column, 1 row
  - SELECT (subquery), ...
  - WHERE ... <logical op> (subquery)
- Column subquery : 1 column, >=1 rows
  - WHERE ... IN (subquery)
  - WHERE ... <logical op> ANY/ALL (subquery)
- Row subquery : 1 row, >=1 columns
  - WHERE ... = (subquery)
  - In SQL2 and Oracle9i
- Table subquery : >=1 columns
  - FROM (subquery) [AS] alias (>=1 rows)
  - WHERE [NOT] EXISTS (subquery) (>=0 rows)

Subquery with Aggregate

- List all books with a price greater than the average book price, and by how much:

-
  ```
  SELECT title_name, price,
  price-(SELECT AVG(price) FROM titles) pdiff
  FROM titles
  WHERE price > (SELECT AVG(price)
                 FROM titles);
  ```
- CANNOT write

  ~~WHERE price > AVG(price);~~

Subquery (SQL Server 2000)

- List all books with a price greater than the average book price, and by how much:

```
SELECT title_name, price,
       price – avgprice pdiff
FROM
 (SELECT title_name, price,
   (SELECT AVG(price) FROM titles) avgprice
  FROM titles
  WHERE price >
          (SELECT AVG(price)
           FROM titles)
) S1;
```

Subquery Rules

- Always enclosed with parentheses
- When subquery is an operand in a comparison, subquery must appear on right-hand side
- Subquery SELECT list must consist of a single column name or expression, **except** when using [NOT] EXISTS, or in FROM clause (Oracle supports multiple columns)
- ORDER BY may not be used in a subquery, only in outermost SELECT
- By default, column names in a subquery refer to tables inside the subquery; to refer to outer tables, use alias qualifier (See correlated subqueries later)

Simple Subqueries: use of IN

- List authors with books published by 'Peachpit':
-
```
SELECT au_id, au_fname, au_lname
FROM authors
WHERE au_id IN
  (SELECT au_id
   FROM title_authors
   WHERE title_id IN
     (SELECT title_id
      FROM titles
      WHERE pub_id IN
        (SELECT pub_id
         FROM publishers
         WHERE pub_name IN ('Peachpit')
         )
      )
   );
```

ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers
- With ALL, condition will only be true if it is satisfied by all values produced by subquery
- With ANY, condition will be true if it is satisfied by any values produced by subquery
- If subquery is empty
  ALL returns true, ANY returns false
- SOME may be used in place of ANY

Use of ANY

- Find the books with a price that is greater than the price of at least one other book:

-
  ```
  SELECT title_name, price, type
  FROM titles
  WHERE price > ANY
     (SELECT price
      FROM titles
     );
  ```

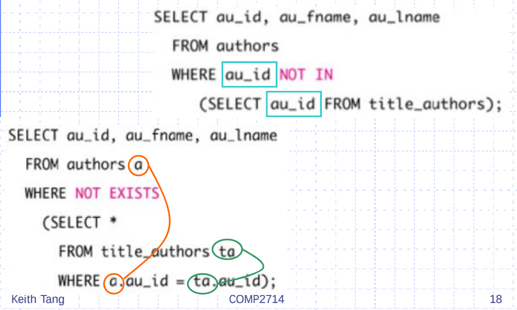
Use of ALL

- Find the most expensive books (similar to using the MAX aggregate function):

-
  ```
  SELECT title_name, type, price
  FROM titles
  WHERE price >= ALL
     (SELECT price
      FROM titles
      );
  ```

Correlated Subqueries Example

- Find authors who have not written a book:



```
SELECT au_id, au_fname, au_lname
  FROM authors
  WHERE au_id NOT IN
    (SELECT au_id FROM title_authors);
```

```
SELECT au_id, au_fname, au_lname
  FROM authors a
  WHERE NOT EXISTS
    (SELECT *
      FROM title_authors ta
      WHERE a.au_id = ta.au_id);
```

Correlated Subqueries

- Offers a more powerful data retrieval mechanism than simple subqueries.
- Order of execution starts with the outer query and it executes repeatedly once for each candidate row selected by the outer query.
- Cannot be executed independently of its outer query; needs the outer query for its values.
- Always refers to the table in the FROM clause of the outer query – Correlated.

Correlated Subqueries Example

- Find the authors that are in the same city and state of a publisher:

-
  ```
  SELECT *
  FROM authors
  WHERE (city, state) IN     -- Oracle supported
    (SELECT city, state      -- multi-columns
     FROM publishers);
  ```

-
  ```
  SELECT *
  FROM authors a
  WHERE city IN              -- If only single-
    (SELECT city             -- column supported
     FROM publishers p
     WHERE a.state = p.state);
  ```

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries
- Produce a simple true/false result
- True if and only if there exists at least one row in result table returned by subquery
- False if subquery returns an empty result table
- NOT EXISTS is the opposite of EXISTS

EXISTS and NOT EXISTS

- As [NOT] EXISTS check only for existence or non-existence of rows in the subquery result table, the columns SELECTed in the subquery is irrelevant
- It is common for subqueries following [NOT] EXISTS to be of form:
  (SELECT * ...)
  or
  (SELECT 1 ...)

Query using EXISTS

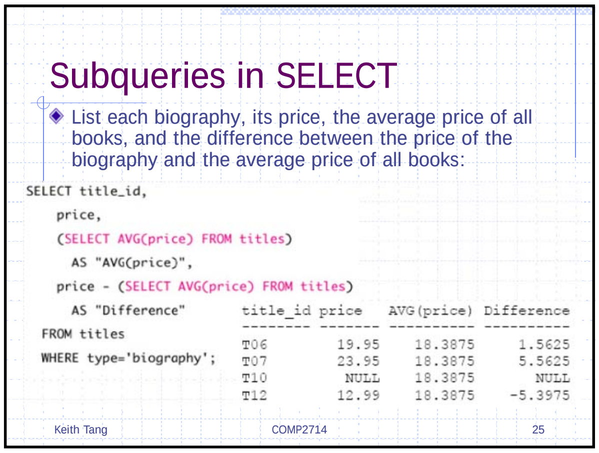
- List the authors who wrote (or co-wrote three or more books:

```
SELECT au_id, au_fname, au_lname
  FROM authors a
  WHERE EXISTS
    (SELECT *
      FROM title_authors ta
      WHERE ta.au_id = a.au_id
      HAVING COUNT(*) >= 3);
```

Query using NOT EXISTS

- List the cities in which an author lives but a publisher is not located:

```
SELECT DISTINCT city
  FROM authors a
  WHERE NOT EXISTS
    (SELECT *
      FROM publishers p
      WHERE p.city = a.city);
```

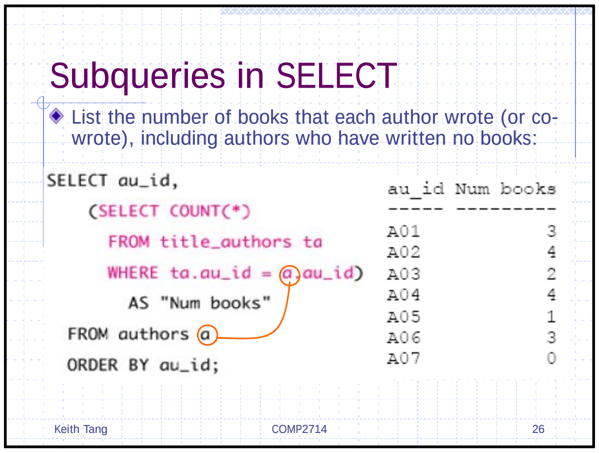


Subqueries in SELECT

- List each biography, its price, the average price of all books, and the difference between the price of the biography and the average price of all books:

```
SELECT title_id,
    price,
    (SELECT AVG(price) FROM titles)
      AS "AVG(price)",
    price - (SELECT AVG(price) FROM titles)
      AS "Difference"
  FROM titles
  WHERE type='biography';
```

```
title_id price   AVG(price) Difference
-------- ------- ---------- ----------
T06        19.95    18.3875     1.5625
T07        23.95    18.3875     5.5625
T10         NULL    18.3875       NULL
T12        12.99    18.3875    -5.3975
```



Subqueries in SELECT

- List the number of books that each author wrote (or co-wrote), including authors who have written no books:

```
SELECT au_id,
    (SELECT COUNT(*)
      FROM title_authors ta
      WHERE ta.au_id = a.au_id)
        AS "Num books"
  FROM authors a
  ORDER BY au_id;
```

```
au_id Num books
----- ---------
A01           3
A02           4
A03           2
A04           4
A05           1
A06           3
A07           0
```