Advanced Data Manipulation Language
Database 2 - Lecture 1

1. Queries within Queries

We will examine how we can nest one query inside another. SQL allows queries within queries, or subqueries, which are SELECT statements inside SELECT statements.

A subquery's syntax is just the same as a normal SELECT query's syntax. As with a normal SELECT statement, a subquery can contain joins, WHERE clauses, HAVING clauses, and GROUP BY clauses.

2. Subquery Terminology

You'll notice references to the outer and inner subqueries. The outer query is the main SELECT statement, and you could say that so far all of your SELECT statements have been outer queries. Shown below is a standard query:

```
SELECT MemberId FROM Members;
```

Using the standard query, you can nest—that is, place inside the outer query—a subquery, which is termed the inner query:

```
SELECT MemberId FROM MemberDetails
WHERE MemberId = (SELECT MAX(FilmId) FROM Films);
```

A WHERE clause is added to the outer query, and it specifies that MemberId must equal the value returned by the nested inner query, which is contained within brackets in the preceding example. It's also possible to nest a subquery inside the inner query. Consider the following example:

```
SELECT MemberId FROM MemberDetails
WHERE MemberId = (SELECT MAX(FilmId) FROM Films
WHERE FilmId IN (SELECT LocationId FROM Location));
```

In the preceding example, a subquery is added to the WHERE clause of the inner query.

3. Subqueries in a SELECT List

You can include a subquery as one of the expressions returning a value in a SELECT query, just as you can include a single column. However, the subquery must return just one record in one expression, in what is known as a scalar subquery. The subquery must also be enclosed in brackets. An example makes things a bit clearer:

```
SELECT Category,
(SELECT MAX(DVDPrice) FROM Films WHERE Films.CategoryId =
Category.CategoryId),
CategoryId
FROM Category;
```

The SELECT query starts off by selecting the Category column, much as you've already seen a hundred times before in the book. However, the next item in the list is not another column but rather a subquery. This query inside the main query returns the maximum price of a DVD. An

aggregate function returns only one value, complying with the need for a subquery in a SELECT statement to be a scalar subquery. The subquery is also linked to the outer SELECT query using a WHERE clause. Because of this link, MAX(DVDPrice) returns the maximum price for each category in the Category table.
If you execute the whole query, you get the following results:

Category	DVDPrice	Category.CategoryId
Thriller	12.99	1
Romance	12.99	2
Horror	9.99	3
War	12.99	4
Sci-fi	12.99	5
Historical	15.99	6
Comedy	NULL	7
Film Noir	NULL	9

There are no films with a value in the DVDPrice columns for the Comedy or Film Noir categories, so NULL is returned.

4. Subqueries in the WHERE Clause

The syntax and form that subqueries take in WHERE clauses are identical to how they are used in SELECT statements, with one difference: now subqueries are used in comparisons. It's a little clearer with an example. Imagine that you need to find out the name or names, if there are two or more of identical value, of the cheapest DVDs for each category. You want the results to display the category name, the name of the DVD, and its price.

```
SELECT Category, FilmName, DVDPrice
FROM Category INNER JOIN Films
ON Category.CategoryId = Films.CategoryId
WHERE Films.DVDPrice =
(SELECT MIN(DVDPrice) FROM Films WHERE Films.CategoryId =
Category.CategoryId);
```

The results are as follows:

Category	FilmName	DVDPrice
Thriller	The Maltese Poodle	2.99
Romance	On Golden Puddle	12.99
Horror	One Flew over the Crow's Nest	8.95
War	Planet of the Japes	12.99
Sci-fi	Soylent Yellow	12.99
Historical	The Good, the Bad, and the Facially Challenged	8.95

5. Operators in Subqueries

So far, all the subqueries you've seen have been scalar subqueries—that is, queries that only return only one row. If more than one row is returned, you end up with an error. In this and the following sections, you learn about operators that allow you to make comparisons against a multirecord results set.

5.1 Revisiting the IN Operator

You can also use the IN operator with subqueries. Instead of providing a list of literal values, a SELECT query provides the list of values. For example, if you want to know which members were born in the same year that a film in the Films table was released, you'd use the following SQL query.

```
SELECT FirstName, LastName, YEAR(DateOfBirth)
FROM MemberDetails
WHERE YEAR(DateOfBirth) IN (SELECT YearReleased FROM Films);
```

Ex2:

```
SELECT FirstName, LastName, YEAR(DateOfBirth)
FROM MemberDetails
WHERE YEAR(DateOfBirth) NOT IN (SELECT YearReleased FROM Films);
```

5.2 Using the ANY, SOME, and ALL Operators

The IN operator allows a simple comparison to see whether a value matches one in a list of values returned by a subquery. The ANY, SOME, and ALL operators not only allow an equality match but also allow any comparison operator to be used.

5.2.1 ANY and SOME Operators

First off, ANY and SOME are identical; they do the same thing but have a different name. For ANY to return true to a match, the value being compared needs to match any one of the values returned by the subquery. You must place the comparison operator before the ANY keyword.

For example, the following SQL uses the equality (=) operator to find out whether any members have the same birth year as the release date of a film in the Films table:

```
SELECT FirstName, LastName, YEAR(DateOfBirth)
FROM MemberDetails
WHERE YEAR(DateOfBirth) = ANY (SELECT YearReleased FROM Films);
```

We note that the results from = ANY are the same as using the IN operator but we can use any comparison operator rather than =, for example :

```
SELECT FirstName, LastName
FROM MemberDetails
WHERE DateOfJoining < ANY (SELECT MeetingDate FROM Attendance);
```

5.2.2 ALL Operator

The ALL operator requires that every item in the list (all the results of a subquery) comply with the condition set by the comparison operator used with ALL.

Put the ALL operator into an example where you select MemberIds that are less than all the values returned by the subquery (SELECT FilmId FROM Films WHERE FilmId > 5):

```
SELECT MemberId
FROM MemberDetails
WHERE MemberId < ALL (SELECT FilmId FROM Films WHERE FilmId > 5);
```

5.3 Using the EXISTS Operator

The EXISTS operator is unusual in that it checks for rows and does not compare columns. So far you've seen lots of clauses that compare one column to another. On the other hand, EXISTS simply checks to see whether a subquery has returned one or more rows. If it has, then the clause returns true; if not, then it returns false.

Ex:

```
SELECT City
FROM Location
WHERE EXISTS (SELECT * FROM MemberDetails WHERE MemberId < 5);
```

You can reverse the logic of EXISTS by using the NOT operator, essentially checking to see whether no results are returned by the subquery. Modify the second example described previously, where the subquery returns no results, adding a NOT operator to the EXISTS keyword:

```
SELECT City
FROM Location
WHERE NOT EXISTS (SELECT * FROM MemberDetails WHERE MemberId > 99);
```