**SubQuery**

Salespeople(snum,sname,city,comm)

Customer(Cnum,Cname,city,Rating,snum)

Orders(onum,Amount,Odate,cnum,snum)

Inner Query geneates values that are tested in the predicate of the outer query

1) Suppose you have name and not snum of salesperson ‘XYZ’ and want to extract all his orders

Select \* from orders where snum =( select snum from salesperson where

Sname like ‘XYZ’);

Subquery must select one and only one column, & the data type of this column must match that of the value to which it is being comapared in the predicate.

Distinct with Subquery:

2) Find all orders credited to the same salesperson who sevices Hoffman(cnum=2)

Select \* from from orders where snum =

(select DISTINT snum from orders where cnum=2);

Aggregate functions in SubQueries:

3) Find all orders that are greater than the average for October 4th

Select \* from orders where amt > ( select avg(amount)

From orders where odate = ’10-OT-90’);

Subqueries that produce multiple rows

4) Find all orders attributed to salespeople in London

Select \* from orders where snum IN( select snum from salespeople

Where city like ‘London’);

Alternate

Select onum,amount,odate,orders.snum from orders,salespeople

Where orders.snum=salespeople.snum and salespeople.city like ‘London’

// Much greater complexity

5) Find the commission of all salespeople servicing customers in London

Select comm. From salesperson where snum IN ( select snum from customers where city like ‘London’);

Expressions in SubQuery

6) Find all customers whose cnum is 1000 above the snum of ‘XYZ’

Select \* from customers where cnum=( select snum+1000 from salespeople where sname like ‘XYZ’);

Salespeople(snum,sname,city,comm)

Customer(Cnum,Cname,city,Rating,snum)

Orders(onum,Amount,Odate,cnum,snum)

SubQueries in Having

These can use their own aggregate functions as long as they do not produce multiple values or use group by or having themselves,

7) Count the customers with ratings above SanJose’ average

**Select rating, count (DISTINCT cnum) from customers**

**group by rating having rating > ( select avg(rating) from customers**

**where city like ‘SanJose’);**

8) Find the total amount in orders for each salesman for whom their total is greater than the amount of the largest order in the table.

**Select snum,sum(amount) from orders group by snum**

**Having sum(amount)> (select max(amount) from orders);**

SubQueries and Join

9) Write a query that produces the names and ratings of all customers who have above-average orders.

Select DISTINCT cname,rating from customers,orders where

Amount> (select avg(amount) from orders) and orders.cnum=customers.cnum;

Select DISTINCT cname,rating from customers natural join orders on customers.cnum=orders.cnum where

Amount> (select avg(amount) from orders) ;

Correlated SubQuery

Can refer in the inner query to the table in the form clause of the outer query. The subquery is exercised repeatedly, once for each row of the main query’s table.

10) Find all customers with orders on October 3rd

**Select \* from customers outer where ’10-oct-03’ IN ( select odate**

**From orders inner where outer.cnum=inner.cnum);**

HOW THE CORRELATED SUBQUERY WORKS?

In the above example, "inner" and "outer" are, of course,

aliases. We chose these names for the sake of clarity; they refer to values from the inner and outer queries, respectively.

Because the value in the cnum field of the outer query varies, the inner query must be executed separately for each row of the outer query. The row of the outer query for which the inner query is being executed at any time is called the **current candidate row**. Therefore, the procedure to evaluate a correlated subquery is this:

1. Select a row from the table named in the outer query.

This will be the current candidate row.

2. Store the values from this candidate row in the alias

named in the FROM clause of the outer query.

3. Perform the subquery. Wherever the alias given for the

outer query is found (in this case "outer"), use the value

for the current candidate row. The use of a value from

the outer query's candidate row in a subquery is called

an **outer reference**.

4. Evaluate the predicate of the outer query on the basis of the

results of the subquery performed in step 3. This will **determine**

whether the candidate row is selected for output.

5. Repeat the procedure for the next candidate row of the

table, and so on until all the rows of the table have been

tested.

Explanation of the example:

1. It selects the row of Hoffman from the Customers table.

2. It stores this row as the current candidate row under the

alias "outer".

3. It then performs the subquery. The subquery goes

through the entire Orders table to find rows where the

cnum field is the same as outer, cnum, which currently is

2001, the cnum of Hoffman's row. It then extracts the

odate field from each row of the Orders table for which

this is true, and builds a set of the resulting odate values.

4. Having formed a set of all odate values where the cnum

is 2001, it tests the predicate of the main query to see if

October 3 is in this set. If it is (and it is), it selects Hoffman's

row for output from the main query.

5. It repeats the entire procedure using Giovanni's row as

the candidate row, and then keeps repeating until every

row of the Customers table has been tested.

Alternate:

SELECT \* FROM Customers first, Orders second

WHERE first.cnum = second.cnum AND second.odate = 10/03/1990;

//This will produce duplicate records

11) Find the names and numbers of all salespeople who have more than one customer

**SELECT snum, sname FROM Salespeople main WHERE 1 <**

**(SELECT COUNT (\*) FROM Customers WHERE snum = main.snum);**

12) Find all orders with above average amounts for their customers

**SELECT \* FROM Orders outer WHERE amt > (SELECT AVG (amt)**

**FROM Orders inner WHERE inner.cnum = outer.cnum);**

Salespeople(snum,sname,city,comm)

Customer(Cnum,Cname,city,Rating,snum)

Orders(onum,Amount,Odate,cnum,snum)

Correlation with itself

13) Find all orders with above – average amounts for their customers

**Select \* from orders outer where amount > ( select avg(amount) from orders inner where inner.cnum=outer.cnum);**

Correlated Subquery in Having

14) Find the sums of amounts from the orders table, grouped by date, eliminating all those dates where the sum was not at least 2000 above the maximum amount.

**Select odate,sum(amount) from orders a**

**Group by odate having sum(amount) >**

**(select 2000 + max(amount) from orders b**

**Where a.odate=b.odate);**

// The sub query calculates the MAX value for all rows with the same date as the current aggregate group of the main query. This must be done as above with a where clause. The subquery itself must not use a group by or having clause.

Exists Operator

**Exists is an operator** that produces a **true or false value**, ie. A Boolean Expression. This means it can stand alone in a predicate or be combined with other Boolean Expressions using Boolean operators **AND/OR or NOT**. It takes a subquery as an argument and evaluates to TRUE if it **produces any output** or FALSE if it does not.

15) Extract some data from customers table **if and only if** one or more of the customers are located in London

**Select cnum,cname,city from customers where EXISTS ( select \* from customers where city like ‘London’);**

Combining Exists with Correlated Sub query

16) Find salespeople who have multiple customers

**Select DISTINCT snum from customers outer where EXISTS ( select \* from customers inner where inner.snum = outer.snum and**

**inner.cnum <> outer.cnum);**

Combining Exists and Joins.

17) Repeat the previous query but print, snum,sname and city

**Select DISTINCT first.snum,sname.first.city from salesperson first, customers second where EXISTS ( select \* from customers third where**

**Second.snum = third.snum and second.cnum<>third.cnum and**

**First.snum=second.snum);**

Using not Exists

18) Find all salespeople with only one customer

**Select DISTINCT snum from customers outer where NOT EXISTS ( select \* from customers inner where inner.snum = outer.snum and inner.cnum<>outer.cnum);**

Salespeople(snum,sname,city,comm)

Customer(Cnum,Cname,city,Rating,snum)

Orders(onum,Amount,Odate,cnum,snum)

ANY, ALL

ANY, ALL are similar to EXISTS on that they take subqueries as arguments; they differ in that they are used in conjuction with relational operators.

19) Find salespeople with customers located in their cities

**Select \* from salespeople where city = ANY ( select city from customers);**

Alternate:

**Select \* from salespeople where city IN ( select city from customers);**

ANY can take other logical operators where IN takes only equal to operator. The ANY operator takes all values produced by the subquery, in this case all city values in the Customers table, and evaluates to true if ANY of them equal the city value of the current row of the outer query. This means that the subquery must select values of the same type as those they are being compared to in the main predicate. This is in contrast to EXISTS, which simply determines if a subquery produces results or not and does not actually use the results.

20) Find all salespeople for whom there are customers that follow them in alphabetical order

**SELECT \* FROM Salespeople WHERE sname < ANY (SELECT cname**

**FROM Customers);**

Equivalent to the following EXISTS query:

**SELECT \* FROM Salespeople outer WHERE EXISTS (SELECT \***

**FROM Customers inner WHERE outer.sname < inner.cname);**

21) Select all orders that had amounts that were greater than at least one of the orders from October 6th:

**SELECT \* FROM Orders WHERE amt > ANY (SELECT amt**

**FROM Orders WHERE odate = 10/06/1990);**

22) Find all orders with amounts smaller than any amount for a customer in San Jose

**SELECT \* FROM Orders WHERE amt < ANY (SELECT amt FROM Orders a, Customers b WHEREa.cnum = b.cnum AND b.city = 'San Jose');**

Even though the smallest order in the table was for a customer in San Jose, so was the second largest; therefore almost all the rows were selected. An easy thing to remember is that < ANY means less than the largest value selected, and > ANY means greater than the smallest value selected.

THE SPECIAL OPERATOR ALL

With ALL, the predicate is true if every value selected by the subquery satisfies the condition in the predicate of the outer query.

23) Find only those customers whose ratings are greater than every customer in Rome.

**SELECT \* FROM Customers WHERE rating > ALL (SELECT rating**

**FROM Customers WHERE city = 'Rome');**

This statement examined the rating values of all customers in Rome. It then found those customers with a higher rating than every one of the Rome customers. The highest rating in Rome is Giovanni, with a value of 200. Therefore, only those with a value higher than 200 were selected.

Just as with ANY, we can use EXISTS to produce an alternative

formulation of the same query:

**SELECT \* FROM Customers outer WHERE NOT EXISTS (SELECT \***

**FROM Customers inner WHERE outer.rating < = inner.rating**

**AND inner.city = 'Rome');**

ALL is used primarily with inequalities rather than equalities because a value can be "equal to all" of the results of a subquery only if all of the said results are identical.

UNITING MULTIPLE QUERIES AS ONE

You can put multiple queries together and combine their output using the UNION clause. The UNION clause merges the output of two or more SQL queries into a single set of rows and columns.

24) Find all salespeople and customers located in London output as a single body

**SELECT snum, sname FROM Salespeople WHERE city = 'London'**

**UNION**

**SELECT cnum, cname FROM Customers WHERE city = 'London';**

But arity must be same for union to execute. Union will automatically eliminate duplicate rows.

25) Suppose you have to make a report of which salespeople produce

the largest and smallest orders on each date. We could unite the two queries, inserting text to distinguish the two cases.

**SELECT a.snum, sname, onum, 'Highest on', odate FROM Salespeople a, Orders b WHERE a.snum = b.snum AND b.amt = (SELECT MAX (amt)**

**FROM Orders c WHERE c.odate = b.odate)**

**UNION**

**SELECT a.snum, sname, onum, 'Lowest on', odate FROM Salespeople a, Orders b WHERE a.snum = b.snum AND b.amt = (SELECT MIN (amt)**

**FROM Orders c WHERE c.odate = b.odate);**