SUBQUERY

What is subquery in SQL?

- A subquery is a SQL query nested inside a larger query.
- A subquery may occur in :
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

Contd...

- You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:
- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.

SYNTAX

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT    select_list
                     FROM      table);
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

SQL Subqueries Example :

- We have the following two tables 'student' and 'marks' with common field 'StudentID'.

StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

student

StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

marks

Write a query to identify all students who get better marks than that of the student who's StudentID is 'V002'.

we require two queries.

- **One query** returns the marks (stored in Total_marks field) of 'V002' and
- **Second query** identifies the students who get better marks than the result of the first query.

Contd...

First query:

```sql
SELECT *
FROM `marks`
WHERE studentid = 'V002';
```

Query result:

StudentID	Total_marks
V002	80

The result of the query is 80.

Using the result of this query, here we have written another query to identify the students who get better marks than 80.

Second query:

```
1  SELECT a.studentid, a.name, b.total_marks
2  FROM student a, marks b
3  WHERE a.studentid = b.studentid
4  AND b.total_marks >80;
```

Query result:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Above two queries identified students who get the better number than the student who's StudentID is 'V002' (Abhay).

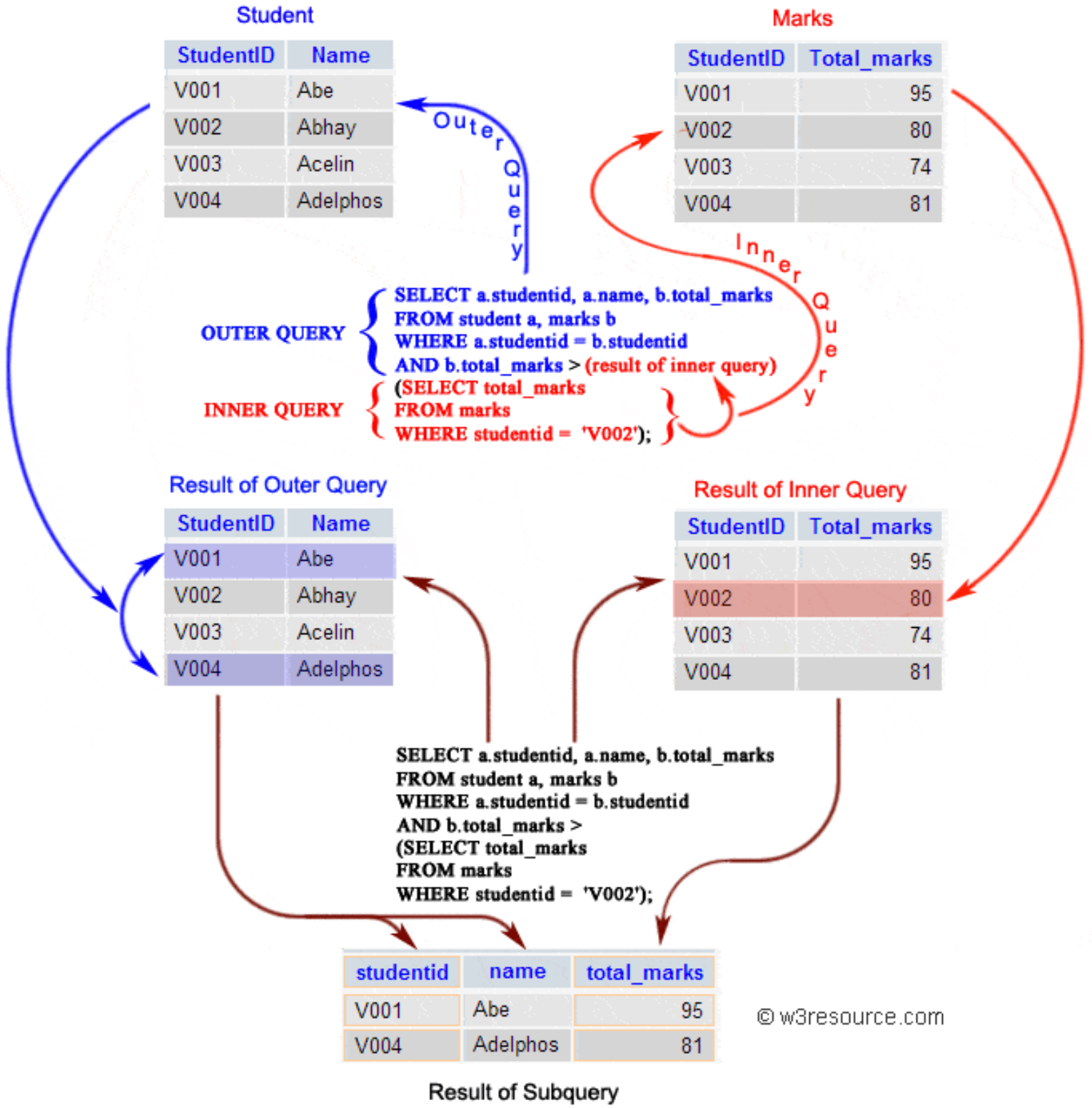
You can combine the above two queries by placing one query inside the other. The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

SQL Code:

```
1  SELECT a.studentid, a.name, b.total_marks
2  FROM student a, marks b
3  WHERE a.studentid = b.studentid AND b.total_marks >
4  (SELECT total_marks
5  FROM marks
6  WHERE studentid =  'V002');
```

Query result:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81


Student
StudentID Name
V001 Abe
V002 Abhay
V003 Acelin
V004 Adelphos
Marks
StudentID Total_marks
V001 95
V002 80
V003 74
V004 81
Outer Query
Inner Query
OUTER QUERY
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks > (result of inner query)
INNER QUERY
(SELECT total_marks
FROM marks
WHERE studentid = 'V002');
Result of Outer Query
StudentID Name
V001 Abe
V002 Abhay
V003 Acelin
V004 Adelphos
Result of Inner Query
StudentID Total_marks
V001 95
V002 80
V003 74
V004 81
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid = 'V002');
studentid name total_marks
V001 Abe 95
V004 Adelphos 81
© w3resource.com
Result of Subquery

Subqueries: Guidelines

There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
- Use single-row operators with single-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Type of Subqueries

- **Single row subquery :** Returns zero or one row.
- **Multiple row subquery :** Returns one or more rows.
- **Multiple column subqueries :** Returns one or more columns.
- **Correlated subqueries :** Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
- **Nested subqueries :** Subqueries are placed within another subquery.

Subqueries with INSERT statement

INSERT statement can be used with subqueries. Here are the syntax and an example of subqueries using INSERT statement.

Syntax:

INSERT INTO table_name [(column1 [, column2])]

SELECT [*|column1 [, column2]

FROM table1 [, table2]

[WHERE VALUE OPERATOR];

If we want to insert those orders from 'orders' table which have the advance_amount 2000 or 5000 into 'neworder' table the following SQL can be used:

Sample table: orders

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200114	3500	2000	15-AUG-08	C00002	A008	
200122	2500	400	16-SEP-08	C00003	A004	
200118	500	100	20-JUL-08	C00023	A006	
200119	4000	700	16-SEP-08	C00007	A010	
200121	1500	600	23-SEP-08	C00008	A004	
200130	2500	400	30-JUL-08	C00025	A011	
200134	4200	1800	25-SEP-08	C00004	A005	
200108	4000	600	15-FEB-08	C00008	A004	
200103	1500	700	15-MAY-08	C00021	A005	
200105	2500	500	18-JUL-08	C00025	A011	

SQL Code:

```
1  INSERT INTO neworder
2  SELECT * FROM  orders
3  WHERE advance_amount in(2000,5000);
```

Subqueries with UPDATE statement

In a UPDATE statement, you can set new column value equal to the result returned by a single row subquery. Here are the syntax and an example of subqueries using UPDATE statement.

Syntax:

UPDATE table SET column_name = new_value

[WHERE OPERATOR [VALUE]

(SELECT COLUMN_NAME

FROM TABLE_NAME)

[WHERE)]

- If we want to update that ord_date in 'neworder' table with '15-JAN-10' which have the difference of ord_amount and advance_amount is less than the minimum ord_amount of 'orders' table the following SQL can be used:

Sample table: neworder

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200114	3500	2000	15-AUG-08	C00002	A008	
200122	2500	400	16-SEP-08	C00003	A004	
200118	500	100	20-JUL-08	C00023	A006	
200119	4000	700	16-SEP-08	C00007	A010	
200121	1500	600	23-SEP-08	C00008	A004	
200130	2500	400	30-JUL-08	C00025	A011	
200134	4200	1800	25-SEP-08	C00004	A005	
200108	4000	600	15-FEB-08	C00008	A004	
200103	1500	700	15-MAY-08	C00021	A005	
200105	2500	500	18-JUL-08	C00025	A011	

SQL Code:

```
1  UPDATE neworder
2  SET ord_date='15-JAN-10'
3  WHERE ord_amount-advance_amount<
4  (SELECT MIN(ord_amount) FROM orders);
```

Output:

```
7 row(s) updated.

0.06 seconds
```

Subqueries with DELETE statement

DELETE statement can be used with subqueries. Here are the syntax and an example of subqueries using DELETE statement.

Syntax:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

If we want to delete those orders from 'neworder' table which advance_amount are less than the maximum advance_amount of 'orders' table, the following SQL can be used:

Sample table: neworder

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200114	3500	2000	15-AUG-08	C00002	A008	
200122	2500	400	16-SEP-08	C00003	A004	
200118	500	100	20-JUL-08	C00023	A006	
200119	4000	700	16-SEP-08	C00007	A010	
200121	1500	600	23-SEP-08	C00008	A004	
200130	2500	400	30-JUL-08	C00025	A011	
200134	4200	1800	25-SEP-08	C00004	A005	
200108	4000	600	15-FEB-08	C00008	A004	
200103	1500	700	15-MAY-08	C00021	A005	
200105	2500	500	18-JUL-08	C00025	A011	

SQL Code:

```
1  DELETE FROM neworder
2  WHERE advance_amount<
3  (SELECT MAX(advance_amount) FROM orders);
```

Write a query to display all the orders from the orders table issued by the salesman 'Paul Adam'

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen	San Jose	0.12
5007	Paul Adam	Rome	0.13

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample Solution:

```
SELECT *
   FROM orders
   WHERE salesman_id =
              (SELECT salesman_id
               FROM salesman
               WHERE name='Paul Adam');
```

Sample Output:

ord_no	purch_amt	ord_date	customer_id	salesman_id
70011	75.29	2012-08-17	3003	5007

Write a query to display all the orders for the salesman who belongs to the city London.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen	San Jose	0.12
5007	Paul Adam	Rome	0.13

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample Solution:

```
SELECT *
FROM orders
WHERE salesman_id =
    (SELECT salesman_id
     FROM salesman
     WHERE city='London');
```

Sample Output:

```
ord_no  purch_amt       ord_date        customer_id     salesman_id
70009   270.65          2012-09-10      3001            5005
```

Write a query to find all the orders issued against the salesman who works for customer whose id is 3007.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen	San Jose	0.12
5007	Paul Adam	Rome	0.13

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample Solution:

```
1  SELECT *
2  FROM orders
3  WHERE salesman_id =
4      (SELECT DISTINCT salesman_id
5       FROM orders
6       WHERE customer_id =3007);
```

Sample Output:

```
ord_no  purch_amt       ord_date        customer_id     salesman_id
70002   65.26           2012-10-05      3002            5001
70005   2400.60         2012-07-27      3007            5001
70008   5760.00         2012-09-10      3002            5001
70013   3045.60         2012-04-25      3002            5001
```

Write a query to display all the orders which values are greater than the average order value for 10th October 2012.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen	San Jose	0.12
5007	Paul Adam	Rome	0.13

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample Solution:

```
1  SELECT *
2  FROM orders
3  WHERE purch_amt >
4      (SELECT  AVG(purch_amt)
5       FROM orders
6       WHERE ord_date ='10/10/2012');
```

Sample Output:

```
ord_no  purch_amt       ord_date        customer_id     salesman_id
70005   2400.60         2012-07-27      3007            5001
70008   5760.00         2012-09-10      3002            5001
70003   2480.40         2012-10-10      3009            5003
70013   3045.60         2012-04-25      3002            5001
```

Write a query to count the customers with grades above New York's average.

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT grade, COUNT (DISTINCT customer_id)
2  FROM customer
3  GROUP BY grade
4  HAVING grade >
5      (SELECT AVG(grade)
6       FROM customer
7       WHERE city = 'New York');
```

Sample Output:

```
grade   count
200             3
300             2
```

Write a query to find the name and numbers of all salesmen who had more than one customer.

Sample table: Orders

70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.4	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.6	2012-04-25	3002	5001

Sample table: customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample table: salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample Solution:

```
1  SELECT salesman_id,name
2  FROM salesman a
3  WHERE 1 <
4      (SELECT COUNT(*)
5       FROM customer
6       WHERE salesman_id=a.salesman_id);
```

Sample Output:

```
salesman_id      name
5001             James Hoog
5002             Nail Knite
```

Write a query to find the sums of the amounts from the orders table, grouped by date, eliminating all those dates where the sum was not at least 1000.00 above the maximum order amount for that date.

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London	100	5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT ord_date, SUM (purch_amt)
2  FROM orders a
3  GROUP BY ord_date
4  HAVING SUM (purch_amt) >
5      (SELECT 1000.00 + MAX(purch_amt)
6       FROM orders b
7       WHERE a.ord_date = b.ord_date);
```

Sample Output:

```
ord_date            sum
2012-09-10          6979.15
2012-10-10          4463.83
```

Write a query to extract the data from the customer table if and only if one or more of the customers in the customer table are located in London.

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT customer_id,cust_name, city
2  FROM customer
3  WHERE EXISTS
4     (SELECT *
5      FROM customer
6      WHERE city='London');
```

Sample Output:

```
customer_id     cust_name                 city
3002            Nick Rimando              New York
3007            Brad Davis                New York
3005            Graham Zusi               California
3008            Julian Green              London
3004            Fabian Johnson            Paris
3009            Geoff Cameron             Berlin
3003            Jozy Altidor              Moscow
3001            Brad Guzan                London
```

Write a query to find all the salesmen who worked for only one customer.

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample Solution:

```
SELECT *
FROM salesman
WHERE salesman_id IN (
   SELECT DISTINCT salesman_id
   FROM customer a
   WHERE NOT EXISTS (
      SELECT * FROM customer b
      WHERE a.salesman_id=b.salesman_id
      AND a.cust_name<>b.cust_name));
```

Sample Output:

```
salesman_id     name            city    commission
5005            Pit Alex        London  0.11
5006            Mc Lyon         Paris   0.14
5007            Paul Adam       Rome    0.13
5003            Lauson Hen      San Jose0.12
```

Write a query to find all the salesmen for whom there are customers that follow them.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT *
2  FROM salesman
3  WHERE city IN
4      (SELECT city
5       FROM customer);
```

Sample Output:

```
salesman_id     name            city            commission
5001            James Hoog      New York        0.15
5002            Nail Knite      Paris           0.13
5005            Pit Alex        London          0.11
5006            Mc Lyon         Paris           0.14
```

Write a query to display the salesmen which name are alphabetically lower than the name of the customers.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
SELECT *
FROM salesman a
WHERE EXISTS
   (SELECT *
    FROM CUSTOMER b
    WHERE  a.name  < b.cust_name);
```

Sample Output:

```
salesman_id     name            city            commission
5001            James Hoog      New York        0.15
5002            Nail Knite      Paris           0.13
5006            Mc Lyon         Paris           0.14
5003            Lauson Hen      San Jose        0.12
```

Write a query to find all orders with an amount smaller than any amount for a customer in London.

Sample table: Orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London	100	5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT *
2  FROM orders
3  WHERE purch_amt < ANY
4     (SELECT purch_amt
5      FROM orders a, customer b
6      WHERE  a.customer_id=b.customer_id
7      AND b.city='London');
```

Sample Output:

ord_no	purch_amt	ord_date	customer_id	salesman_id
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70011	75.29	2012-08-17	3003	5007
70001	150.50	2012-10-05	3005	5002
70012	250.45	2012-06-27	3008	5002

Write a query to display only those customers whose grade are, in fact, higher than every customer in New York.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT *
2  FROM customer
3  WHERE grade > ALL
4     (SELECT grade
5      FROM customer
6      WHERE city='New York');
```

Sample Output:

```
customer_id     cust_name              city    grade   salesman_id
3008            Julian Green    London  300             5002
3004            Fabian Johnson  Paris   300             5006
```

Write a query to get all the information for those customers whose grade is not as the grade of customer who belongs to the city London.

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT *
2  FROM customer
3  WHERE grade <> ANY
4     (SELECT grade
5      FROM customer
6      WHERE city='London');
```

Sample Output:

```
customer_id     cust_name           city          grade   salesman_id
3002            Nick Rimando        New York      100     5001
3007            Brad Davis          New York      200     5001
3005            Graham Zusi         California    200     5002
3009            Geoff Cameron       Berlin        100     5003
3003            Jozy Altidor        Moscow        200     5007
```

Write a query to find all those customers whose grade are not as the grade, belongs to the city Paris.

Sample table: Salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Sample table: Customer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002

Sample Solution:

```
1  SELECT *
2  FROM customer
3  WHERE grade NOT IN
4     (SELECT grade
5      FROM customer
6      WHERE city='Paris');
```

Sample Output:

```
customer_id     cust_name           city            grade   salesman_id
3002            Nick Rimando        New York        100     5001
3007            Brad Davis          New York        200     5001
3005            Graham Zusi         California      200     5002
3008            Julian Green        London          300     5002
3004            Fabian Johnson      Paris           300     5006
3009            Geoff Cameron       Berlin          100     5003
3003            Jozy Altidor        Moscow          200     5007
3001            Brad Guzan          London                  5005
```

Write a SQL query to display the name of each company, price for their most expensive product along with their Name.

Sample table: company_mast

COM_ID	COM_NAME
11	Samsung
12	iBall
13	Epsion
14	Zebronics
15	Asus
16	Frontech

Sample table: item_mast

PRO_ID	PRO_NAME	PRO_PRICE	PRO_COM
101	Mother Board	3200	15
102	Key Board	450	16
103	ZIP drive	250	14
104	Speaker	550	16
105	Monitor	5000	11
106	DVD drive	900	12
107	CD drive	800	12
108	Printer	2600	13
109	Refill cartridge	350	13
110	Mouse	250	12

Sample Solution:

```
SELECT P.pro_name AS "Product Name",
       P.pro_price AS "Price",
       C.com_name AS "Company"
   FROM item_mast P, company_mast C
   WHERE P.pro_com = C.com_id
     AND P.pro_price =
     (
       SELECT MAX(P.pro_price)
         FROM item_mast P
         WHERE P.pro_com = C.com_id
     );
```

Output :

Product Name	Price	Company
Monitor	5000	Samsung
DVD drive	900	iBall
Printer	2600	Epsion
ZIP drive	250	Zebronics
Mother Board	3200	Asus
Speaker	550	Frontech

Write a query in SQL to find the names of departments with more than two employees are working.

Sample table: emp_department

DPT_CODE	DPT_NAME	DPT_ALLOTMENT
57	IT	65000
63	Finance	15000
47	HR	240000
27	RD	55000
89	QC	75000

Sample table: emp_details

EMP_IDNO	EMP_FNAME	EMP_LNAME	EMP_DEPT
127323	Michale	Robbin	57
526689	Carlos	Snares	63
843795	Enric	Dosio	57
328717	Jhon	Snares	63
444527	Joseph	Dosni	47
659831	Zanifer	Emily	47
847674	Kuleswar	Sitaraman	57
748681	Henrey	Gabriel	47
555935	Alex	Manuel	57
539569	George	Mardy	27

Sample Solution:

```
1  SELECT dpt_name FROM emp_department
2    WHERE dpt_code IN
3    (
4      SELECT emp_dept
5        FROM emp_details
6        GROUP BY emp_dept
7        HAVING COUNT(*) >2
8    );
```

Output:

DPT_NAME
IT
HR
Finance

In most cases JOINs are faster than sub-queries and it is very rare for a sub-query to be faster.

- In JOINs RDBMS can create an execution plan that is better for your query and can predict what data should be loaded to be processed and save time, unlike the sub-query where it will run all the queries and load all their data to do the processing.
- The good thing in sub-queries is that they are more readable than JOINs: that’s why most new SQL people prefer them; it is the easy way; but when it comes to performance, JOINS are better in most cases even though they are not hard to read too.

SQL Correlated Subqueries

- **SQL Correlated Subqueries** are used to select data from a table referenced in the outer query.
- The subquery is known as a correlated because the subquery is related to the outer query.
- In this type of queries, a table alias (also called a correlation name) must be used to specify which table reference is to be used.

Using EXISTS with a Correlated Subquery

We have already used the EXISTS operator to check the existence of a result of a subquery. EXISTS operator can be used in correlated subqueries also. Using EXISTS the following query display the employee_id, manager_id, first_name and last_name of those employees who manage other employees.

SQL Code:

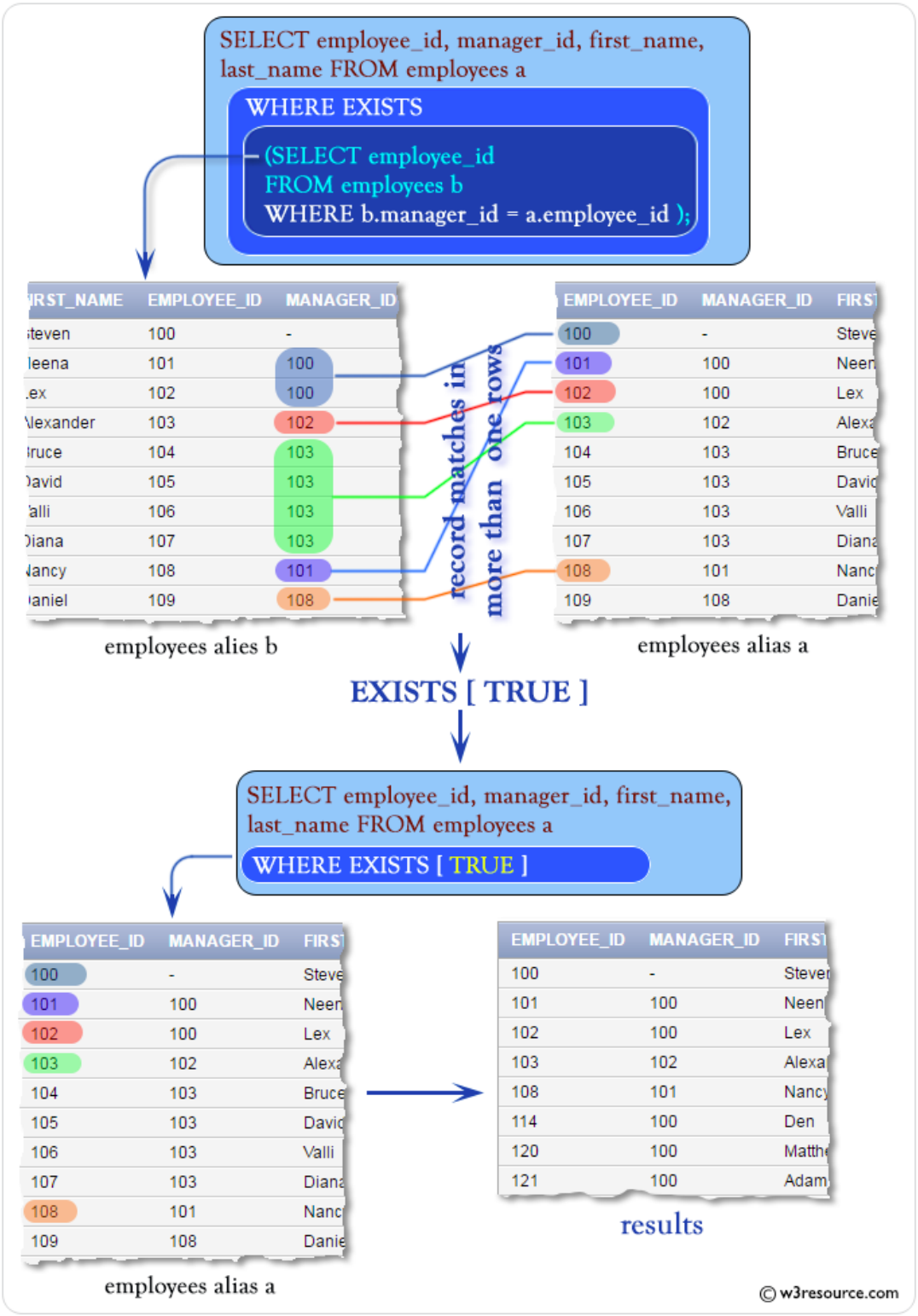
```
1  SELECT employee_id, manager_id, first_name, last_name
2  FROM employees a
3  WHERE EXISTS
4  (SELECT employee_id
5  FROM employees b
6  WHERE b.manager_id = a.employee_id)
```

Sample table: employees

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary
100	Steven	King	SKING	515.123.4567	6/17/1987	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	6/18/1987	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	6/19/1987	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	6/20/1987	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4568	6/21/1987	IT_PROG	6000
105	David	Austin	DAUSTIN	590.423.4569	6/22/1987	IT_PROG	4800
106	Valli	Pataballa	VPATABAL	590.423.4560	6/23/1987	IT_PROG	4800
107	Diana	Lorentz	DLORENTZ	590.423.5567	6/24/1987	IT_PROG	4200
108	Nancy	Greenberg	NGREENBE	515.124.4569	6/25/1987	FI_MGR	12000

Output:

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME
100		Steven	King
101	100	Neena	Kochhar
102	100	Lex	De Haan
103	102	Alexander	Hunold
108	101	Nancy	Greenberg
114	100	Den	Raphaely
120	100	Matthew	Weiss
121	100	Adam	Fripp
122	100	Payam	Kaufling
123	100	Shanta	Vollman
124	100	Kevin	Mourgos
145	100	John	Russell
146	100	Karen	Partners
147	100	Alberto	Errazuriz
148	100	Gerald	Cambrault
149	100	Eleni	Zlotkey
201	100	Michael	Hartstein
205	101	Shelley	Higgins


SELECT employee_id, manager_id, first_name, last_name FROM employees a
WHERE EXISTS
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id);
EMPLOYEE_ID MANAGER_ID
100 -
101 100
102 100
103 102
104 103
105 103
106 103
107 103
108 101
109 108
record matches in more than one rows
employees alies b
employees alias a
EXISTS [TRUE]
SELECT employee_id, manager_id, first_name, last_name FROM employees a
WHERE EXISTS [TRUE]
EMPLOYEE_ID MANAGER_ID
100 -
101 100
102 100
103 102
104 103
105 103
106 103
107 103
108 101
109 108
employees alias a
EMPLOYEE_ID MANAGER_ID
100 -
101 100
102 100
103 102
108 101
114 100
120 100
121 100
results
© w3resource.com

Using NOT EXISTS with a Correlated Subquery

NOT EXISTS is logically opposite of EXISTS operator. NOT EXISTS is used when we need to check if rows do not exist in the results returned by a subquery. Using NOT EXISTS the following query display the employee_id, manager_id, first_name and last_name of those employees who have no manager status. This query is opposite to the previous one.

SQL Code:

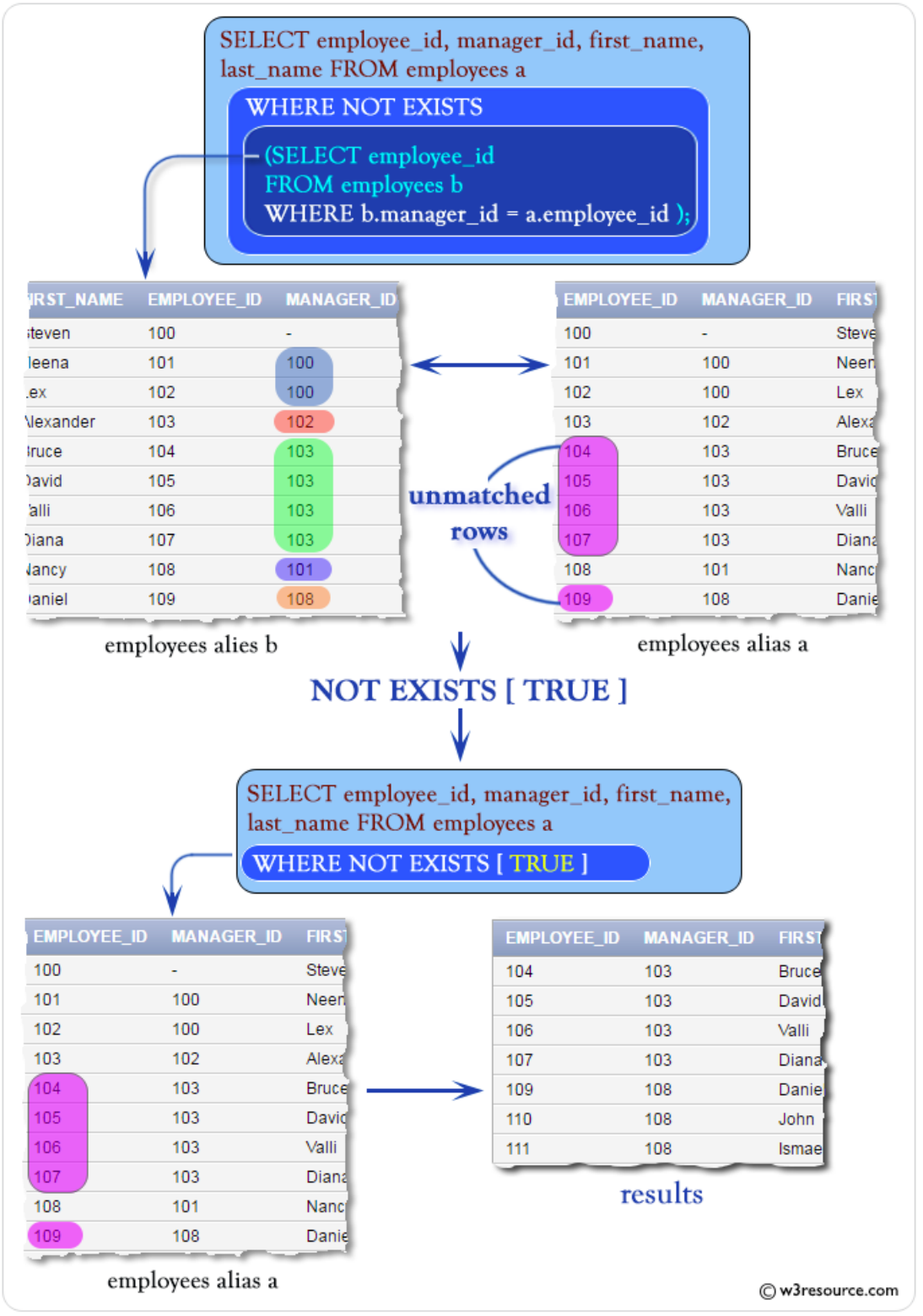
```sql
SELECT employee_id, manager_id, first_name, last_name
FROM employees a
WHERE NOT EXISTS
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id);
```

Sample table: employees

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary
100	Steven	King	SKING	515.123.4567	6/17/1987	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	6/18/1987	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	6/19/1987	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	6/20/1987	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4568	6/21/1987	IT_PROG	6000
105	David	Austin	DAUSTIN	590.423.4569	6/22/1987	IT_PROG	4800
106	Valli	Pataballa	VPATABAL	590.423.4560	6/23/1987	IT_PROG	4800
107	Diana	Lorentz	DLORENTZ	590.423.5567	6/24/1987	IT_PROG	4200
108	Nancy	Greenberg	NGREENBE	515.124.4569	6/25/1987	FI_MGR	12000

Output:

```
EMPLOYEE_ID MANAGER_ID FIRST_NAME           LAST_NAME
----------- ---------- -------------------- --------------
        104        103 Bruce                Ernst
        105        103 David                Austin
        106        103 Valli                Pataballa
        107        103 Diana                Lorentz
        109        108 Daniel               Faviet
        110        108 John                 Chen
        111        108 Ismael               Sciarra
        112        108 Jose Manuel          Urman
        113        108 Luis                 Popp
        115        114 Alexander            Khoo
        116        114 Shelli               Baida
        117        114 Sigal                Tobias
        118        114 Guy                  Himuro
        119        114 Karen                Colmenares
        125        120 Julia                Nayer
        126        120 Irene                Mikkilineni
        127        120 James                Landry
        128        120 Steven               Markle
        129        121 Laura                Bissot
        130        121 Mozhe                Atkinson
        131        121 James                Marlow
.........
```


SELECT employee_id, manager_id, first_name, last_name FROM employees a
WHERE NOT EXISTS
(SELECT employee_id
FROM employees b
WHERE b.manager_id = a.employee_id);
unmatched rows
employees alies b
employees alias a
NOT EXISTS [TRUE]
SELECT employee_id, manager_id, first_name, last_name FROM employees a
WHERE NOT EXISTS [TRUE]
employees alias a
results
© w3resource.com