Advanced T-SQL

Table OF Contents

1. SQL: GROUP BY CLAUSE

1.1 DESCRIPTION

The SQL GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

1.2 SYNTAX

The syntax for the SQL GROUP BY clause is:

```
SELECT expression1, expression2, ... expression_n,
      aggregate_function (expression)
FROM tables
WHERE conditions
GROUP BY expression1, expression2, ... expression_n;
```

1.3 PARAMETERS OR ARGUMENTS

- expression1, expression2, ... expression_n are expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY Clause.
- aggregate_function can be a function such as SUM function, COUNT function, MIN function, or MAX function.
- tables are the tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.
- conditions are conditions that must be met for the records to be selected.

1.4 EXAMPLE - USING SUM FUNCTION

Let's look at a SQL GROUP BY query example that uses the SQL SUM function.
This GROUP BY example uses the SUM function to return the name of the department and the total sales (for the department).

```
SELECT department, SUM(sales) AS "Total sales"
FROM order_details
GROUP BY department;
```

Because you have listed one column (the department field) in your SQL SELECT statement that is not encapsulated in the SUM function, you must use the GROUP BY Clause. The department field must, therefore, be listed in the GROUP BY clause.

1.5 EXAMPLE - USING COUNT FUNCTION

Let's look at how we could use the GROUP BY clause with the SQL COUNT function.
This GROUP BY example uses the COUNT function to return the department and the number of employees (in the department) that make over $25,000 / year.

```
SELECT department, COUNT(*) AS "Number of employees"
FROM employees
WHERE salary > 25000
GROUP BY department;
```

1.6 EXAMPLE - USING MIN FUNCTION

Let's next look at how we could use the GROUP BY clause with the SQL MIN function.
This GROUP BY example uses the MIN function to return the name of each department and the minimum salary in the department.

```
SELECT department, MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department;
```

1.7 EXAMPLE - USING MAX FUNCTION

Finally, let's look at how we could use the GROUP BY clause with the SQL MAX function.
This GROUP BY example uses the MAX function to return the name of each department and the maximum salary in the department.

```
SELECT department, MAX(salary) AS "Highest salary"
FROM employees
GROUP BY department;
```

2. SQL: HAVING CLAUSE

2.1 DESCRIPTION

The SQL HAVING Clause is used in combination with the GROUP BY Clause to restrict the groups of returned rows to only those whose the condition is TRUE.

2.2 SYNTAX

The syntax for the SQL HAVING Clause is:

```
SELECT expression1, expression2, ... expression_n,
       aggregate_function (expression)
FROM tables
WHERE conditions
GROUP BY expression1, expression2, ... expression_n
HAVING condition;
```

2.3 PARAMETERS OR ARGUMENTS

- aggregate_function can be a function such as SQL SUM function, SQL COUNT function, SQL MIN function, or SQL MAX function.
- expression1, expression2, ... expression_n are expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY Clause.
- condition is the condition that is used to restrict the groups of returned rows. Only those groups whose condition evaluates to TRUE will be included in the result set.

2.4 EXAMPLE - USING SUM FUNCTION

Let's look at a SQL HAVING clause example that uses the SQL SUM function.
You could also use the SQL SUM function to return the name of the department and the total sales

(in the associated department). The SQL HAVING clause will filter the results so that only departments with sales greater than $1000 will be returned.
SELECT department, SUM(sales) AS "Total sales"
FROM order_details
GROUP BY department
HAVING SUM(sales) > 1000;

2.5 EXAMPLE - USING COUNT FUNCTION

Let's look at how we could use the HAVING clause with the SQL COUNT function.
You could use the SQL COUNT function to return the name of the department and the number of employees (in the associated department) that make over $25,000 / year. The SQL HAVING clause will filter the results so that only departments with more than 10 employees will be returned.
SELECT department, COUNT(*) AS "Number of employees"
FROM employees
WHERE salary > 25000
GROUP BY department
HAVING COUNT(*) > 10;

2.6 EXAMPLE - USING MIN FUNCTION

Let's next look at how we could use the HAVING clause with the SQL MIN function.
You could also use the SQL MIN function to return the name of each department and the minimum salary in the department. The SQL HAVING clause will return only those departments where the minimum salary is greater than $35,000.
SELECT department, MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department
HAVING MIN(salary) > 35000;
EXAMPLE - USING MAX FUNCTION
Finally, let's look at how we could use the HAVING clause with the SQL MAX function.
For example, you could also use the SQL MAX function to return the name of each department and the maximum salary in the department. The SQL HAVING clause will return only those departments whose maximum salary is less than $50,000.
SELECT department, MAX(salary) AS "Highest salary"
FROM employees
GROUP BY department
HAVING MAX(salary) < 50000;

3. SQL: IN CONDITION

3.1 DESCRIPTION

The SQL IN condition is used to help reduce the need for multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

3.2 SYNTAX

The syntax for the SQL IN condition is:
expression IN (value1, value2, ... value_n);

3.3 PARAMETERS OR ARGUMENTS

- expression is a value to test.
- value1, value2..., or value_n are the values to test against expression.

NOTE
- The SQL IN condition will return the records where expression is value1, value2..., or value_n.
- The SQL IN condition is also called the SQL IN operator.

3.4 EXAMPLE - WITH CHARACTER

Let's look at an IN condition example using character values.
The following is a SQL SELECT statement that uses the IN condition to compare character values:

```
SELECT *
FROM suppliers
WHERE supplier_name IN ('IBM', 'Hewlett Packard', 'Microsoft');
```

This SQL IN condition example would return all rows where the supplier_name is either IBM, Hewlett Packard, or Microsoft. Because the * is used in the select, all fields from the suppliers table would appear in the result set.
This IN condition example is equivalent to the following SQL statement:

```
SELECT *
FROM suppliers
WHERE supplier_name = 'IBM'
OR supplier_name = 'Hewlett Packard'
OR supplier_name = 'Microsoft';
```

As you can see, using the SQL IN condition makes the statement easier to read and more efficient.

3.5 EXAMPLE - WITH NUMERIC

Next, let's look at an IN condition example using numeric values.
For example:

```
SELECT *
FROM orders
WHERE order_id IN (10000, 10001, 10003, 10005);
```

This SQL IN condition example would return all orders where the order_id is either 10000, 10001, 10003, or 10005.
This IN condition example is equivalent to the following SQL statement:

```
SELECT *
FROM orders
WHERE order_id = 10000
OR order_id = 10001
OR order_id = 10003
OR order_id = 10005;
```

3.6 EXAMPLE - USING NOT OPERATOR

Finally, let's look at an IN condition example using the NOT operator.
For example:
SELECT *
FROM suppliers
WHERE supplier_name NOT IN ('IBM', 'Hewlett Packard', 'Microsoft');
This SQL IN condition example would return all rows where the supplier_name is neither IBM, Hewlett Packard, or Microsoft. Sometimes, it is more efficient to list the values that you do not want, as opposed to the values that you do want.

4. SQL: JOINS

4.1 DESCRIPTION

SQL JOINS are used to retrieve data from multiple tables. A SQL JOIN is performed whenever two or more tables are joined in a SQL statement.

There are 4 different types of SQL joins:

- SQL INNER JOIN (or sometimes called simple join)
- SQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- SQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
- SQL FULL OUTER JOIN (or sometimes called FULL JOIN)

So let's discuss SQL JOIN syntax, look at visual illustrations of SQL JOINS, and explore SQL JOIN examples.

4.2 SQL INNER JOIN (SIMPLE JOIN)

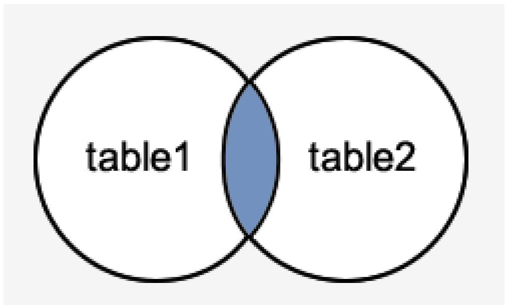
Chances are, you've already written a SQL statement that uses an SQL INNER JOIN. It is the most common type of SQL join. SQL INNER JOINS return all rows from multiple tables where the join condition is met.

4.2.1 SYNTAX

The syntax for the SQL INNER JOIN is:
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;

4.2.2 VISUAL ILLUSTRATION

In this visual diagram, the SQL INNER JOIN returns the shaded area:



The SQL INNER JOIN would return the records where table1 and table2 intersect.

4.2.3 EXAMPLE

Here is an example of a SQL INNER JOIN:

```
SELECT s.supplier_id, s.supplier_name, od.order_date
FROM suppliers AS s
INNER JOIN order_details AS od
ON s.supplier_id = od.supplier_id;
```

This SQL INNER JOIN example would return all rows from the suppliers and orders tables where there is a matching supplier_id value in both the suppliers and orders tables.

Let's look at some data to explain how the INNER JOINS work:
We have a table called suppliers with two fields (supplier_id and supplier_ name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have another table called orders with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10004	2003/05/14

If we run the SQL statement (that contains an INNER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13

The rows for Microsoft and NVIDIA from the supplier table would be omitted, since the supplier_id's 10002 and 10003 do not exist in both tables. The row for 500127 (order_id) from the orders table would be omitted, since the supplier_id 10004 does not exist in the suppliers table.

4.2.4 OLD SYNTAX

As a final note, it is worth mentioning that the SQL INNER JOIN example above could be rewritten using the older implicit syntax as follows (but we still recommend using the INNER JOIN keyword syntax):
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id;

4.3 SQL LEFT OUTER JOIN

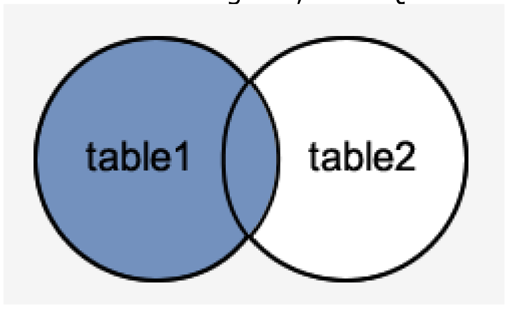
Another type of join is called a LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

4.3.1 SYNTAX

The syntax for the SQL LEFT OUTER JOIN is:
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
In some databases, the LEFT OUTER JOIN keywords are replaced with LEFT JOIN.

4.3.2 VISUAL ILLUSTRATION

In this visual diagram, the SQL LEFT OUTER JOIN returns the shaded area:



The SQL LEFT OUTER JOIN would return the all records from table1 and only those records from table2 that intersect with table1.

4.3.3 EXAMPLE

Here is an example of a SQL LEFT OUTER JOIN:
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
This LEFT OUTER JOIN example would return all rows from the suppliers table and only those rows from the orders table where the joined fields are equal.
If a supplier_id value in the suppliers table does not exist in the orders table, all fields in the

orders table will display as <null> in the result set.

Let's look at some data to explain how LEFT OUTER JOINS work:
We have a table called suppliers with two fields (supplier_id and name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have a second table called orders with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13

If we run the SQL statement (that contains a LEFT OUTER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	supplier_name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13
10002	Microsoft	<null>
10003	NVIDIA	<null>

The rows for Microsoft and NVIDIA would be included because a LEFT OUTER JOIN was used. However, you will notice that the order_date field for those records contains a <null> value.

4.3.4 OLD SYNTAX

As a final note, it is worth mentioning that the LEFT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the LEFT OUTER JOIN keyword syntax):

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id(+);
```

4.4 SQL RIGHT OUTER JOIN

Another type of join is called a SQL RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

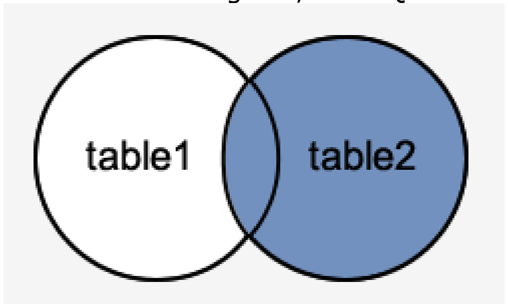
4.4.1 SYNTAX

The syntax for the SQL RIGHT OUTER JOIN is:

SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
In some databases, the RIGHT OUTER JOIN keywords are replaced with RIGHT JOIN.

4.4.2 VISUAL ILLUSTRATION

In this visual diagram, the SQL RIGHT OUTER JOIN returns the shaded area:



The SQL RIGHT OUTER JOIN would return the all records from table2 and only those records from table1 that intersect with table2.

4.4.3 EXAMPLE

Here is an example of a SQL RIGHT OUTER JOIN:
SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
This RIGHT OUTER JOIN example would return all rows from the orders table and only those rows from the suppliers table where the joined fields are equal.
If a supplier_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

Let's look at some data to explain how RIGHT OUTER JOINS work:
We have a table called suppliers with two fields (supplier_id and name). It contains the following data:

supplier_id	supplier_name
10000	Apple
10001	Google

We have a second table called orders with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10002	2013/08/14

If we run the SQL statement (that contains a RIGHT OUTER JOIN) below:
SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;
Our result set would look like this:

order_id	order_date	supplier_name
500125	2013/08/12	Apple
500126	2013/08/13	Google
500127	2013/08/14	<null>

The row for 500127 (order_id) would be included because a RIGHT OUTER JOIN was used. However, you will notice that the supplier_name field for that record contains a <null> value.

4.4.4 OLD SYNTAX

As a final note, it is worth mentioning that the RIGHT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the RIGHT OUTER JOIN keyword syntax):

```
SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers, orders
WHERE suppliers.supplier_id(+) = orders.supplier_id;
```

4.5 SQL FULL OUTER JOIN

Another type of join is called a SQL FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

4.5.1 SYNTAX

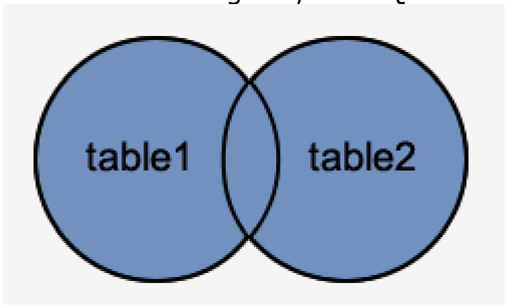
The syntax for the SQL FULL OUTER JOIN is:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

In some databases, the FULL OUTER JOIN keywords are replaced with FULL JOIN.

4.5.2 VISUAL ILLUSTRATION

In this visual diagram, the SQL FULL OUTER JOIN returns the shaded area:



The SQL FULL OUTER JOIN would return the all records from both table1 and table2.

4.5.3 EXAMPLE

Here is an example of a SQL FULL OUTER JOIN:

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
FULL OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

This FULL OUTER JOIN example would return all rows from the suppliers table and all rows from the orders table and whenever the join condition is not met, <nulls> would be extended to those fields in the result set.
If a supplier_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set. If a supplier_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

Let's look at some data to explain how FULL OUTER JOINS work:
We have a table called suppliers with two fields (supplier_id and name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have a second table called orders with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10004	2013/08/14

If we run the SQL statement (that contains a FULL OUTER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
FULL OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	supplier_name	order_date
10000	IBM	2013/08/12
10001	Hewlett Packard	2013/08/13
10002	Microsoft	<null>
10003	NVIDIA	<null>
<null>	<null>	2013/08/14

The rows for Microsoft and NVIDIA would be included because a FULL OUTER JOIN was used. However, you will notice that the order_date field for those records contains a <null> value.
The row for supplier_id 10004 would be also included because a FULL OUTER JOIN was used. However, you will notice that the supplier_id and supplier_name field for those records contain a <null> value.

4.5.4 OLD SYNTAX

As a final note, it is worth mentioning that the FULL OUTER JOIN example above could not have been written in the old syntax without using a UNION query.

5. SQL: LIKE CONDITION

5.1 DESCRIPTION

The SQL LIKE condition allows you to use wildcards to perform pattern matching. The LIKE condition is used in the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement.

5.2 SYNTAX

The syntax for the SQL LIKE Condition is:
expression LIKE pattern [ESCAPE 'escape_character']

5.3 PARAMETERS OR ARGUMENTS

- expression is a character expression such as a column or field.
- pattern is a character expression that contains pattern matching. The patterns that you can choose from are:
  - % allows you to match any string of any length (including zero length)
  - _ allows you to match on a single character
  - escape_character is optional. It allows you to test for literal instances of a wildcard character such as % or _.

5.4 EXAMPLE - USING % WILDCARD (PERCENT SIGN WILDCARD)

The first SQL LIKE example that we will look at involves using the % wildcard (percent sign wildcard).
Let's explain how the % wildcard works in the SQL LIKE condition. We want to find all of the suppliers whose name begins with 'Hew'.

```
SELECT supplier_name
FROM suppliers
WHERE supplier_name LIKE 'Hew%';
```

You can also using the % wildcard multiple times within the same string. For example,

```
SELECT supplier_name
FROM suppliers
WHERE supplier_name LIKE '%bob%';
```

In this SQL LIKE condition example, we are looking for all suppliers whose name contains the characters 'bob'.

5.5 EXAMPLE - USING _ WILDCARD (UNDERSCORE WILDCARD)

Next, let's explain how the _ wildcard (underscore wildcard) works in the SQL LIKE condition. Remember that _ wildcard is looking for only one character.
For example:

```
SELECT last_name
```

```
FROM customers
WHERE last_name LIKE 'Sm_th';
```

This SQL LIKE condition example would return all customers whose last_name is 5 characters long, where the first two characters is 'Sm' and the last two characters is 'th'. For example, it could return customers whose last_name is 'Smith', 'Smyth', 'Smath', 'Smeth', etc.
Here is another example:

```
SELECT *
FROM suppliers
WHERE account_number LIKE '12317_';
```

You might find that you are looking for an account number, but you only have 5 of the 6 digits. The example above, would retrieve potentially 10 records back (where the missing value could equal anything from 0 to 9). For example, it could return suppliers whose account numbers are: 123170, 123171, 123172, 123173, 123174, 123175, 123176, 123177, 123178, 123179

5.6 EXAMPLE - USING THE NOT OPERATOR

Next, let's look at how you would use the SQL NOT Operator with wildcards.
Let's use the % wilcard with the NOT Operator. You could also use the SQL LIKE condition to find suppliers whose name does not start with 'T'.
For example:

```
SELECT supplier_name
FROM suppliers
WHERE supplier_name NOT LIKE 'T%';
```

By placing the NOT Operator in front of the SQL LIKE condition, you are able to retrieve all suppliers whose supplier_name does not start with 'T'.

5.7 EXAMPLE - USING ESCAPE CHARACTERS

It is important to understand how to "Escape Characters" when pattern matching. These examples deal specifically with escaping characters in Oracle.
Let's say you wanted to search for a % or a _ character in the SQL LIKE condition. You can do this using an Escape character.
Please note that you can only define an escape character as a single character (length of 1).
For example:

```
SELECT *
FROM suppliers
WHERE supplier_name LIKE '!%' escape '!';
```

This SQL LIKE condition example identifies the ! character as an escape character. This statement will return all suppliers whose name is %.
Here is another more complicated example using escape characters in the SQL LIKE condition.

```
SELECT *
FROM suppliers
WHERE supplier_name LIKE 'H%!%' escape '!';
```

This SQL LIKE condition example returns all suppliers whose name starts with H and ends in %. For example, it would return a value such as 'Hello%'.
You can also use the escape character with the _ character in the SQL LIKE condition.
For example:

```
SELECT *
FROM suppliers
WHERE supplier_name LIKE 'H%!_' escape '!';
```

This SQL LIKE condition example returns all suppliers whose name starts with H and ends in _. For example, it would return a value such as 'Hello_'.

FREQUENTLY ASKED QUESTIONS

Question: How do you incorporate the Oracle UPPER function with the SQL LIKE condition? I'm trying to query against a free text field for all records containing the word "test". The problem is that it can be entered in the following ways: TEST, Test, or test.
Answer: To answer this question, let's look at an example.
Let's say that we have a suppliers table with a field called supplier_name that contains the values TEST, Test, or test.
If we wanted to find all records containing the word "test", regardless of whether it was stored as TEST, Test, or test, we could run either of the following SQL SELECT statements:

```
SELECT *
FROM suppliers
WHERE UPPER(supplier_name) LIKE ('TEST%');
OR
SELECT *
FROM suppliers
WHERE UPPER(supplier_name) LIKE UPPER('test%')
```

These SQL SELECT statements use a combination of the Oracle UPPER function and the SQL LIKE condition to return all of the records where the supplier_name field contains the word "test", regardless of whether it was stored as TEST, Test, or test.

5.8 PRACTICE EXERCISE #1:

Based on the employees table populated with the following data, find all records whose employee_name ends with the letter "h".

```
CREATE TABLE employees
( employee_number number(10) not null,
  employee_name varchar2(50) not null,
  salary number(6),
  CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1001, 'John Smith', 62000);
INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1002, 'Jane Anderson', 57500);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1003, 'Brad Everest', 71000);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1004, 'Jack Horvath', 42000);
```

5.8.1 SOLUTION FOR PRACTICE EXERCISE #1:

The following SQL SELECT statement uses the SQL LIKE condition to return the records whose employee_name ends with the letter "h".

```
SELECT *
FROM employees
WHERE employee_name LIKE '%h';
```

It would return the following result set:

EMPLOYEE_NUMBER	EMPLOYEE_NAME	SALARY

1001	John Smith	62000
1004	Jack Horvath	42000

5.9 PRACTICE EXERCISE #2:

Based on the employees table populated with the following data, find all records whose employee_name contains the letter "s".

```
CREATE TABLE employees
( employee_number number(10) not null,
  employee_name varchar2(50) not null,
  salary number(6),
  CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1001, 'John Smith', 62000);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1002, 'Jane Anderson', 57500);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1003, 'Brad Everest', 71000);

INSERT INTO employees (employee_number, employee_name, salary)
VALUES (1004, 'Jack Horvath', 42000);
```

5.9.1 SOLUTION FOR PRACTICE EXERCISE #2:

The following SQL SELECT statement would use the SQL LIKE condition to return the records whose employee_name contains the letter "s".

```
SELECT *
FROM employees
WHERE employee_name LIKE '%s%';
```

It would return the following result set:

EMPLOYEE_NUMBER	EMPLOYEE_NAME	SALARY
1002	Jane Anderson	57500
1003	Brad Everest	71000

5.10 PRACTICE EXERCISE #3:

Based on the suppliers table populated with the following data, find all records whose supplier_id is 4 digits and starts with "500".

```
CREATE TABLE suppliers
( supplier_id varchar2(10) not null,
  supplier_name varchar2(50) not null,
  city varchar2(50),
  CONSTRAINT suppliers_pk PRIMARY KEY (supplier_id)
);

INSERT INTO suppliers(supplier_id, supplier_name, city)
VALUES ('5008', 'Microsoft', 'New York');
```

```
INSERT INTO suppliers (supplier_id, supplier_name, city)
VALUES ('5009', 'IBM', 'Chicago');

INSERT INTO suppliers (supplier_id, supplier_name, city)
VALUES ('5010', 'Red Hat', 'Detroit');

INSERT INTO suppliers (supplier_id, supplier_name, city)
VALUES ('5011', 'NVIDIA', 'New York');
```

5.10.1 SOLUTION FOR PRACTICE EXERCISE #3:

The following SQL SELECT statement would use the SQL LIKE condition to return the records whose supplier_id is 4 digits and starts with "500".

```
SELECT *
FROM suppliers
WHERE supplier_id LIKE '500_';
```

It would return the following result set:

SUPPLIER_ID	SUPPLIER_NAME	CITY
5008	Microsoft	New York
5009	IBM	Chicago

6. SQL: ORDER BY CLAUSE

6.1 DESCRIPTION

The SQL ORDER BY clause is used to sort the records in the result set for a SELECT statement.

6.2 SYNTAX

The syntax for the SQL ORDER BY clause is:

```
SELECT expressions
FROM tables
WHERE conditions
ORDER BY expression [ ASC | DESC ];
```

6.3 PARAMETERS OR ARGUMENTS

- expressions are the columns or calculations that you wish to retrieve.
- tables are the tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.
- conditions are conditions that must be met for the records to be selected.
- ASC is optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).
- DESC is optional. It sorts the result set in descending order by expression.

NOTE

- If the ASC or DESC modifier is not provided in the ORDER BY clause, the results will be sorted by expression in ascending order (which is equivalent to "ORDER BY expression ASC").

6.4 EXAMPLE - SORTING WITHOUT USING ASC/DESC ATTRIBUTE

The SQL ORDER BY clause can be used without specifying the ASC or DESC value. When this attribute is omitted from the SQL ORDER BY clause, the sort order is defaulted to ASC or ascending order.
For example:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city;
```

This SQL ORDER BY example would return all records sorted by the supplier_city field in ascending order and would be equivalent to the following SQL ORDER BY clause:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city ASC;
```

Most programmers omit the ASC attribute if sorting in ascending order.

6.5 EXAMPLE - SORTING IN DESCENDING ORDER

When sorting your result set in descending order, you use the DESC attribute in your ORDER BY clause as follows:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

This SQL ORDER BY example would return all records sorted by the supplier_city field in descending order.

6.6 EXAMPLE - SORTING BY RELATIVE POSITION

You can also use the SQL ORDER BY clause to sort by relative position in the result set, where the first field in the result set is 1. The next field is 2, and so on.
For example:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY 1 DESC;
```

This SQL ORDER BY would return all records sorted by the supplier_city field in descending order, since the supplier_city field is in position #1 in the result set and would be equivalent to the following SQL ORDER BY clause:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

6.7 EXAMPLE - USING BOTH ASC AND DESC ATTRIBUTES

When sorting your result set using the SQL ORDER BY clause, you can use the ASC and DESC attributes in a single SQL SELECT statement.
For example:

```
SELECT supplier_city, supplier_state
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC, supplier_state ASC;
```

This SQL ORDER BY would return all records sorted by the supplier_city field in descending order, with a secondary sort by supplier_state in ascending order.

7. SQL: DISTINCT CLAUSE

7.1 DESCRIPTION

The SQL DISTINCT clause is used to remove duplicates from the result set of a SELECT statement.

7.2 SYNTAX

The syntax for the SQL DISTINCT clause is:

```
SELECT DISTINCT expressions
FROM tables
WHERE conditions;
```

7.3 PARAMETERS OR ARGUMENTS

- expressions are the columns or calculations that you wish to retrieve.
- tables are the tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.
- conditions are conditions that must be met for the records to be selected.

NOTE

- When only one expression is provided in the DISTINCT clause, the query will return the unique values for that expression.
- When more than one expression is provided in the DISTINCT clause, the query will retrieve unique combinations for the expressions listed.

7.4 EXAMPLE - WITH SINGLE FIELD

Let's look at the simplest SQL DISTINCT query example. We can use the SQL DISTINCT clause to return a single field that removes the duplicates from the result set.
For example:

```
SELECT DISTINCT city
FROM suppliers;
```

This SQL DISTINCT example would return all unique city values from the suppliers table.

7.5 EXAMPLE - WITH MULTIPLE FIELDS

Let's look at how you might use the SQL DISTINCT clause to remove duplicates from more than one field in your SQL SELECT statement.
For example:

```
SELECT DISTINCT city, state
FROM suppliers;
```

This SQL DISTINCT clause example would return each unique city and state combination. In this case, the DISTINCT applies to each field listed after the DISTINCT keyword.