SQL

MOST IMPORTANT CONCEPTS

PLACEMENT PREPARATION

[EXCLUSIVE NOTES]

[SAVE AND SHARE]

Curated By- HIMANSHU KUMAR(LINKEDIN)

TOPICS COVERED-

PART-3 :-

- AND and OR operators
- Union Clause
- Join (Cartesian Join & Self Join)
- DROP, DELETE, TRUNCATE
- DROP, TRUNCATE
- Date functions
- EXISTS
- WITH clause
- NULL Values

HIMANSHU KUMAR(LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri

AND and OR operators-

In SQL, the AND & OR operators are used for filtering the data and getting precise results based on conditions. The SQL **AND** & **OR** operators are also used to combine multiple conditions. These two operators can be combined to test for multiple conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

When combining these conditions, it is important to use parentheses so that the database knows what order to evaluate each condition.

- The AND and OR operators are used with the WHERE clause.
- These two operators are called conjunctive operators.

AND Operator:-

This operator displays only those records where both the conditions condition1 and condition2 evaluates to True.

Syntax:

```
SELECT * FROM table_name WHERE condition1 AND condi
tion2 and ...conditionN;

table_name: name of the table

condition1,2,..N : first condition, second conditio
n and so on
```

OR Operator:

This operator displays the records where either one of the conditions condition1 and condition2 evaluates to True. That is, either condition1 is True or condition2 is True.

Syntax:

```
SELECT * FROM table_name WHERE condition1 OR condition2 OR... conditionN;

table_name: name of the table

condition1,2,..N : first condition, second condition and so on
```

Now, we consider a table database to demonstrate AND & OR operators with multiple cases:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

If suppose we want to fetch all the records from the Student table where Age is 18 and ADDRESS is Delhi. then the query will be:

Query:

```
SELECT * FROM Student WHERE Age = 18 AND ADDRESS = 'Delhi';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	18

Take another example, to fetch all the records from the Student table where NAME is Ram and Age is 18.

Query:

```
SELECT * FROM Student WHERE Age = 18 AND NAME = 'Ram';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

To fetch all the records from the Student table where NAME is Ram or NAME is SUJIT.

Query:

```
SELECT * FROM Student WHERE NAME = 'Ram' OR NAME = 'SUJIT';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20

To fetch all the records from the Student table where NAME is Ram or Age is 20.

Query:

```
SELECT * FROM Student WHERE NAME = 'Ram' OR Age = 20;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

Combining AND and OR:

We can combine AND and OR operators in the below manner to write complex queries.

Syntax:

```
SELECT * FROM table_name WHERE condition1 AND (condition2 OR condition3);
```

Take an example to fetch all the records from the Student table where Age is 18 NAME is Ram or RAMESH.

Query:

```
SELECT * FROM Student WHERE Age = 18 AND (NAME = 'Ram' OR NAME = 'RAMESH');
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18

Union Clause-

The Union Clause is used to combine two separate select statements and produce the result set as a union of both the select statements.

NOTE:

1. The fields to be used in both the select statements must be in same order, same number and same data type.
2. The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

Basic Syntax:-

```
SELECT column_name(s) FROM table1 UNION SELECT colu
mn_name(s) FROM table2;

Resultant set consists of distinct values.

SELECT column_name(s) FROM table1 UNION ALL SELECT
column_name(s) FROM table2;

Resultant set consists of duplicate values too.
```

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

Student_Details

ROLL_NO	Branch	Grade
1	Information Technology	O
2	Computer Science	E
3	Computer Science	O
4	Mechanical Engineering	A

Queries:-

- To fetch distinct ROLL_NO from Student and Student_Details table.

- ```
  SELECT ROLL_NO FROM Student UNION SELECT ROLL_NO FROM Student_Details;
  ```

Output:

ROLL_NO
1
2
3
  ```

ROLL_NO
4

- To fetch ROLL_NO from Student and Student_Details table including duplicate values.

```
SELECT ROLL_NO FROM Student UNION ALL SELECT ROLL_NO FROM Student_Details;
```

Output:

ROLL_NO
1
2
3
4
3
2

- To fetch ROLL_NO , NAME from Student table WHERE ROLL_NO is greater than 3 and ROLL_NO , Branch from Student_Details table WHERE ROLL_NO is less than 3 , including duplicate values and finally sorting the data by ROLL_NO.

```
SELECT ROLL_NO,NAME FROM Student WHERE ROLL_NO>3
UNION ALL
SELECT ROLL_NO,Branch FROM Student_Details WHERE ROLL_NO<3
ORDER BY 1;
Note:The column names in both the select statements can be different but the
 data type must be same.And in the result set the name of column used in the first
 select statement will appear.
```

Output:

ROLL_NO	NAME
1	Information Technology
2	Computer Science
4	SURESH

Join (Cartesian Join & Self Join)-

In this article, we will discuss about the remaining two JOINS:

- **CARTESIAN JOIN**
- **SELF JOIN**

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

CARTESIAN JOIN: The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.

- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

```
SELECT table1.column1 , table1.column2, table2.colu
mn1...
FROM table1
CROSS JOIN table2;
table1: First table.
table2: Second table
```

Example Queries(CARTESIAN JOIN):

- In the below query we will select NAME and Age from Student table and COURSE_ID from StudentCourse table. In the output you can see that each row of the table Student is joined with every row of the table StudentCourse. The total rows in the result-set = 4 * 4 = 16.

```
SELECT Student.NAME, Student.AGE, StudentCourse
.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;
```

Output:

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

2. **SELF JOIN**: As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some conditions. In other words we can say that it is a join between two copies of the same table.

Syntax:

```
SELECT a.coulmn1 , b.column2
FROM table_name a, table_name b
WHERE some_condition;
table_name: Name of the table.
some_condition: Condition for selecting the rows.
```

Example Queries(SELF JOIN):

```
SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;
```

Output:

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

DROP, DELETE, TRUNCATE-

DROP: DROP is used to delete a whole database or just a table.
TRUNCATE: Truncate statement is also used for same purpose.

DROP vs TRUNCATE

- Truncate is normally ultra-fast and its ideal for cleaning out data from a temporary table.
- It also preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.

Basic Syntax for deleting a table:

```
DROP TABLE table_name;
table_name: Name of the table to be deleted.
```

Basic Syntax for deleting a whole data base:

```
DROP DATABASE database_name;
database_name: Name of the database to be deleted.
```

Basic Syntax for truncating a table:

```
TRUNCATE TABLE  table_name;
table_name: Name of the table to be truncated.
```

NOTE: Table or Database deletion using DROP statement cannot be rolled back, so it must be used wisely. Suppose we have following tables in a database.

DATABASE name - student_data

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18
3	SUJIT	ROHTAK	9156253131	20
2	RAMESH	GURGAON	9652431543	18

Student_Details

ROLL_NO	Branch	Grade
1	Information Technology	O
2	Computer Science	E
3	Computer Science	O
4	Mechanical Engineering	A

Queries

- To delete Student table from student_data database.
- `DELETE TABLE Student;`

After running the above query we will be left with only Student_details table.

- To delete the whole database
- `DROP DATABASE student_data;`

After running the above query whole database will be deleted.

- To truncate Student_details table from student_data database.
- `TRUNCATE TABLE Student_details;`

After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for furter operations.

DROP, TRUNCATE-

DROP

DROP is used to delete a whole database or just a table.The DROP statement destroys the objects like an existing database, table, index, or view.
A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Syntax:

```
DROP object object_name
Examples:
DROP TABLE table_name;
table_name: Name of the table to be deleted.



DROP DATABASE database_name;
database_name: Name of the database to be deleted.
```

TRUNCATE

TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extents of a table for deallocation (empty for reuse). The result of this operation quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms.
The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).

Syntax:

```
TRUNCATE TABLE  table_name;
table_name: Name of the table to be truncated.
DATABASE name - student_data
```

DROP vs TRUNCATE-

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
- Table or Database deletion using DROP statement **cannot** be rolled back, so it must be used wisely.

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

Student_Details

ROLL_NO	Branch	Grade
1	Information Technology	O
2	Computer Science	E
3	Computer Science	O
4	Mechanical Engineering	A

Queries

- To delete the whole database
- `DROP DATABASE student_data;`

After running the above query whole database will be deleted.

- To truncate Student_details table from student_data database.

```
TRUNCATE TABLE Student_details;
```

After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

Date functions-

In SQL, dates are complicated for newbies, since while working with database, the format of the date in table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.
In MySql the default date functions are:

- **NOW():** Returns the current date and time. Example:

```
SELECT NOW();
```

Output:

```
2017-01-13 08:03:52
```

- **CURDATE()**: Returns the current date. Example:

```
SELECT CURDATE();
```

Output:

```
2017-01-13
```

- **CURTIME():** Returns the current time. Example:

- ```
  SELECT CURTIME();
  ```

Output:

```
08:05:15
```

- **DATE()**: Extracts the date part of a date or date/time expression. Example:
  For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

- ```
  SELECT Name, DATE(BirthTime) AS BirthDate FROM Test;
  ```
- **Output:**

Name	BirthDate
Pratik	1996-09-26

- **EXTRACT():** Returns a single part of a date/time. Syntax:

- ```
  EXTRACT(unit FROM date);
  ```

There are several units that can be considered but only some are used such as:
  ```

MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.
And 'date' is a valid date expression.

Example:
For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Queries

```
SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;
```

Output:

Name	BirthDay
Pratik	26

```
SELECT Name, Extract(YEAR FROM BirthTime) AS BirthYear FROM Test;
```

Output:

Name	BirthYear
Pratik	1996

```
SELECT Name, Extract(SECOND FROM BirthTime) AS BirthSecond FROM Test;
```

Output:

Name	BirthSecond
Pratik	581

- **DATE_ADD() :** Adds a specified time interval to a date

Syntax:

- `DATE_ADD(date, INTERVAL expr type);`

Where, date - valid date expression and expr is the number of interval we want to add.
and type can be one of the following:
MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.

Example:
For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Queries

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL 1 YEAR) AS BirthTimeModified FROM Test;
```

Output:

Name	BirthTimeModified
Pratik	1997-09-26 16:44:15.581

- ```
  SELECT Name, DATE_ADD(BirthTime, INTERVAL 30 DAY) AS BirthDayModified FROM Test;
  ```

Output:

Name	BirthDayModified
Pratik	1996-10-26 16:44:15.581

- ```
  SELECT Name, DATE_ADD(BirthTime, INTERVAL 4 HOUR) AS BirthHourModified FROM Test;
  ```

Output:

Name	BirthSecond
Pratik	1996-10-26 20:44:15.581

- **DATE_SUB():** Subtracts a specified time interval from a date. Syntax for DATE_SUB is same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.
- **DATEDIFF():** Returns the number of days between two dates.Syntax:

- ```
  DATEDIFF(date1, date2);
  ```
- ```
  date1 & date2- date/time expression
  ```

Example:

```
SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;
```

Output:

DateDiff
10

- **DATE_FORMAT():** Displays date/time data in different formats.Syntax:

- `DATE_FORMAT(date,format);`

date is a valid date and format specifies the output format for the date/time. The formats that can be used are:

  - %a-Abbreviated weekday name (Sun-Sat)
  - %b-Abbreviated month name (Jan-Dec)
  - %c-Month, numeric (0-12)
  - %D-Day of month with English suffix (0th, 1st, 2nd, 3rd)
  - %d-Day of month, numeric (00-31)
  - %e-Day of month, numeric (0-31)
  - %f-Microseconds (000000-999999)
  - %H-Hour (00-23)
  - %h-Hour (01-12)
  - %I-Hour (01-12)
  - %i-Minutes, numeric (00-59)
  - %j-Day of year (001-366)
  - %k-Hour (0-23)
  - %l-Hour (1-12)
  - %M-Month name (January-December)
  - %m-Month, numeric (00-12)

- %p-AM or PM
- %r-Time, 12-hour (hh:mm:ss followed by AM or PM)
- %S-Seconds (00-59)
- %s-Seconds (00-59)
- %T-Time, 24-hour (hh:mm:ss)
- %U-Week (00-53) where Sunday is the first day of week
- %u-Week (00-53) where Monday is the first day of week
- %V-Week (01-53) where Sunday is the first day of week, used with %X
- %v-Week (01-53) where Monday is the first day of week, used with %x
- %W-Weekday name (Sunday-Saturday)
- %w-Day of the week (0=Sunday, 6=Saturday)
- %X-Year for the week where Sunday is the first day of week, four digits, used with %V
- %x-Year for the week where Monday is the first day of week, four digits, used with %v
- %Y-Year, numeric, four digits
- %y-Year, numeric, two digits

Example:

```
DATE_FORMAT(NOW(),'%d %b %y')
```

Result:

```
13 Jan 17
```

EXISTS-

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a boolean value True or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
  (SELECT column_name(s)
   FROM table_name
   WHERE condition);
```

Examples:

Consider the following two relation "Customers" and "Orders".

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Queries

1. **Using EXISTS condition with SELECT statement** To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname
FROM Customers
WHERE EXISTS (SELECT * FROM Orders WHERE Customer
s.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

2. **Using NOT with EXISTS** Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *FROM Orders WHERE Custo
mers.customer_id = Orders.c_id);
```

Output:

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

3. **Using EXISTS condition with DELETE statement** Delete the record of all the customer from Order Table whose last name is 'Mehra'.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *FROM customers WHERE Custom
ers.customer_id = Orders.cid AND Customers.lname
= 'Mehra'); SELECT * FROM Orders;
```

Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

4. **Using EXISTS condition with UPDATE statement** Update the lname as 'Kumari' of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers
SET lname = 'Kumari'
WHERE EXISTS (SELECT *FROM Customers WHERE custom
er_id = 401);

SELECT * FROM Customers;
```

Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

WITH clause-

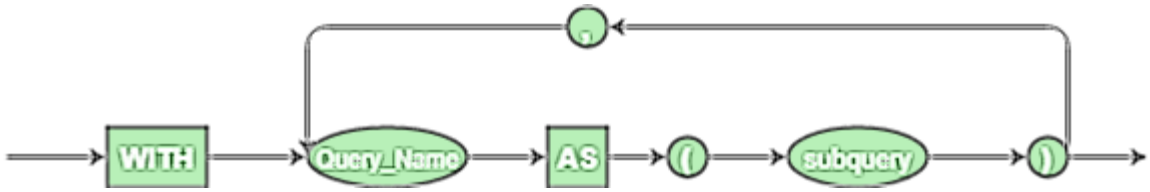
The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database. The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
- Queries that have an associated WITH clause can also be written using nested sub-queries but doing so add more complexity to read/debug the SQL query.
- WITH clause is not supported by all database system.
- The name assigned to the sub-query is treated as though it was an inline view or table

- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database.

Syntax:

```
WITH temporaryTable (averageValue) as(SELECT avg(Attr1)FROM Table) SELECT Attr1 FROM Table, temporaryTable WHERE Table.Attr1 > temporaryTable.averageValue;
```



In this query, WITH clause is used to define a temporary relation temporaryTable that has only 1 attribute averageValue. averageValue holds the average value of column Attr1 described in relation Table. The SELECT statement that follows the WITH clause will produce only those tuples where the value of Attr1 in relation Table is greater than the average value obtained from the WITH clause statement.

Note: When a query with a WITH clause is executed, first the query mentioned within the clause is evaluated and the output of this evaluation is stored in a temporary relation. Following this, the main query associated with the WITH clause is finally executed that would use the temporary relation produced.

Queries

Example 1: Find all the employee whose salary is more than the average salary of all employees.
Name of the relation: **Employee**

EmployeeID	Name	Salary
100011	Smith	50000
100022	Bill	94000
100027	Sam	70550
100845	Walden	80000
115585	Erik	60000
1100070	Kate	69000

SQL Query:

```
WITH temporaryTable(averageValue) as
    (SELECT avg(Salary)
    from Employee)
        SELECT EmployeeID,Name, Salary
        FROM Employee, temporaryTable
        WHERE Employee.Salary > temporaryTable.aver
ageValue;
```

Output:

EmployeeID	Name	Salary
100022	Bill	94000
100845	Walden	80000

Explanation: The average salary of all employees is 70591. Therefore, all employees whose salary is more than the obtained average lies in the output relation.

Example 2: Find all the airlines where the total salary of all pilots in that airline is more than the average of total salary of all pilots in the database.

Name of the relation: **Pilot**

EmployeeID	Airline	Name	Salary
70007	Airbus 380	Kim	60000
70002	Boeing	Laura	20000
10027	Airbus 380	Will	80050
10778	Airbus 380	Warren	80780
115585	Boeing	Smith	25000
114070	Airbus 380	Katy	78000

SQL Query:

```
WITH totalSalary(Airline, total) as
    (SELECT Airline, sum(Salary)
    FROM Pilot
    GROUP BY Airline),
    airlineAverage(avgSalary) as
    (SELECT avg(Salary)
    FROM Pilot )
    SELECT Airline
    FROM totalSalary, airlineAverage
    WHERE totalSalary.total > airlineAverage.avgSal
ary;
```

Output:

Airline
Airbus 380

Explanation: The total salary of all pilots of Airbus 380 = 298,830 and that of Boeing = 45000. Average salary of all pilots in the table Pilot = 57305. Since only the total salary of all pilots of Airbus 380 is greater than the average salary obtained, so Airbus 380 lies in the output relation.

Important Points:

- The SQL WITH clause is good when used with complex SQL statements rather than simple ones
- It also allows you to break down complex SQL queries into smaller ones which make it easy for debugging and processing the complex queries.
- The SQL WITH clause is basically a drop-in replacement to the normal sub-query.

NULL Values-

In SQL there may be some records in a table that do not have values or data for every field. This could be possible because at a time of data entry information is not available. So SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple. SQL places a NULL value in the field in the absence of a user-defined value. For example, the Apartment_number attribute of an address applies only to address that are in apartment buildings and not to other types of residences.

Importance of NULL value:

- It is important to understand that a NULL value is different from a zero value.
- A NULL value is used to represent a missing value, but that it usually has one of three different interpretations:

    - The value unknown (value exists but is not known)
    - Value not available (exists but is purposely withheld)

    - Attribute not applicable (undefined for this tuple)
- It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL.

Principles of NULL values:

- Setting a NULL value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is character.
- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, CHECK constraints will ignore by SQL.

In general, each NULL value is considered to be different from every other NULL in the database. When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN. Hence, SQL uses a three-valued logic with values **True, False**, and **Unknown.** It is, therefore, necessary to define the results of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

How to test for NULL Values?

SQL allows queries that check whether an attribute value is NULL. Rather than using = or to compare an attribute value to NULL, SQL uses **IS** and **IS NOT**. This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate. Now, consider the following Employee Table,

Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

Suppose if we find the Fname, Lname of the Employee having no Super_ssn then the query will be:

Query

```
SELECT Fname, Lname FROM Employee WHERE Super_ssn I
S NULL;
```

Output:

Fname	Lname
James	Borg

Now if we find the Count of the number of Employees having Super_ssn.

Query:

```
SELECT COUNT(*) AS Count FROM Employee WHERE Super_
ssn IS NOT NULL;
```

Output:

Count
7

HIMANSHU KUMAR(LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri

CREDITS- INTERNET

DISCLOSURE- THE DATA AND IMAGES ARE TAKEN FROM GOOGLE AND INTERNET.

CHECKOUT AND DOWNLOAD MY ALL NOTES

***LINK*- https://linktr.ee/exclusive_notes**