SQL

Hexavarsity

Objective

- ORDER BY
- AGGREFATE FUNCTION
- GROUP BY
- INTRODUCTION TO JOINS
- JOIN QUERY
- STRING FUNCTION
- DATE FUNCTION
- MATHEMATICAL FUNCTION

SQL ORDER BY

SQL ORDER BY clause

- The ORDER BY is an optional clause of the SELECT statement.
- The ORDER BY clause allows you to sort the rows returned by the SELECT clause by one or more sort expressions in ascending or descending order.

Syntax:

```
SELECT column_list
FROM table1
ORDER BY sort_expression [ASC | DESC];
```


SQL ORDER BY clause

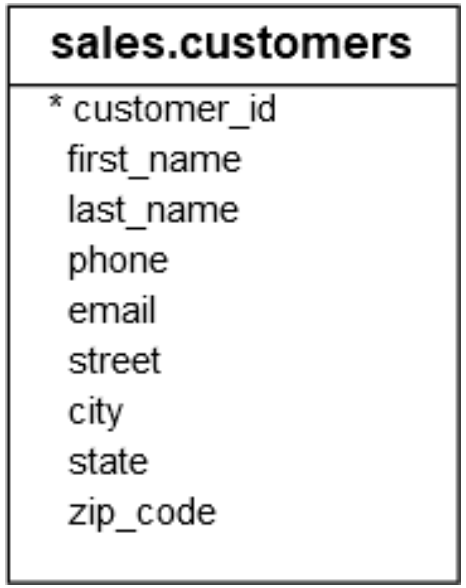
Example

- *SQL ORDER BY DESC clause*

```
SELECT firstname, lastname
FROM sales.customers
ORDER BY first_name DESC;
```

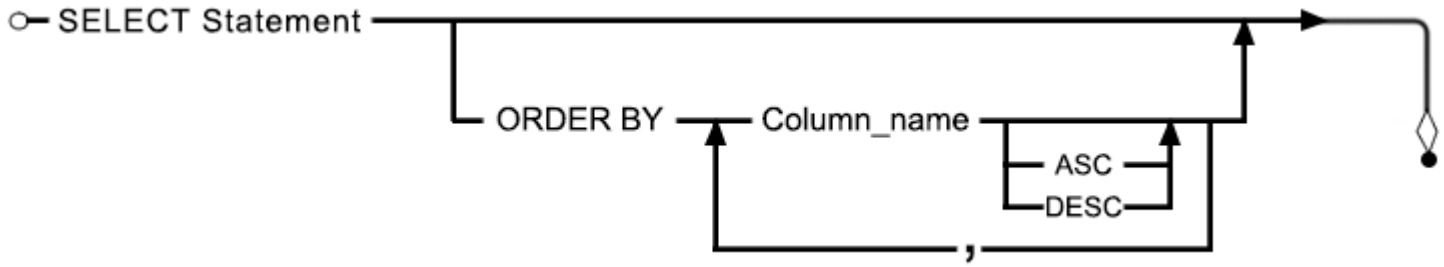
- *SQL ORDER BY clause*

```
SELECT firstname, lastname
FROM sales.customers
ORDER BY first_name;
```



Note:- ORDER BY clause sort first_name column in ascending order or use ASC Keyword

SQL ORDER BY clause




```
SELECT *
FROM employees
ORDER BY dep_id ASC,
job_name DESC;
```

employees

emp_id	emp_name	hire_date	salary	dep_id
68319	KAYLING	1991-11-18	6000	1001
66928	BLAZE	1991-05-01	2750	3001
67832	CLARE	1991-06-09	2550	1001
65646	JONAS	1991-04-02	2957	2001
64989	ADELYN	1991-02-20	1700	3001
65271	WADE	1991-02-22	1350	3001
66564	MADDEN	1991-09-28	1350	3001
68454	TUCKER	1991-09-08	1600	3001
68736	ADNRES	1997-05-23	1200	2001
69000	JULIUS	1991-12-03	1050	3001
69324	MARKER	1992-01-23	1400	1001
67858	SCARLET	1997-04-19	3100	2001
69062	FRANK	1991-12-03	3100	2001
63679	SANDRINE	1990-12-18	900	2001

job_name has been ordered descendingly

department_id has been ordered ascendingly

job_name has been ordered descendingly

department_id has been ordered ascendingly

emp_id	emp_name	job_name	manager_id	hire_date	salary	commission	dep_id
68319	KAYLING	PRESIDENT		1991-11-18	6000		1001
67832	CLARE	MANAGER	68319	1991-06-09	2550		1001
69324	MARKER	CLERK	67832	1992-01-23	1400		1001
65646	JONAS	MANAGER	68319	1991-04-02	2957		2001
63679	SANDRINE	CLERK	69062	1990-12-18	900		2001
68736	ADNRES	CLERK	67858	1997-05-23	1200		2001
67858	SCARLET	ANALYST	65646	1997-04-19	3100		2001
69062	FRANK	ANALYST	65646	1991-12-03	3100		2001
65271	WADE	SALESMAN	66928	1991-02-22	1350	600	3001
68454	TUCKER	SALESMAN	66928	1991-09-08	1600	0	3001
66564	MADDEN	SALESMAN	66928	1991-09-28	1350	1500	3001
64989	ADELYN	SALESMAN	66928	1991-02-20	1700	400	3001
66928	BLAZE	MANAGER	68319	1991-05-01	2750		3001
69000	JULIUS	CLERK	66928	1991-12-03	1050		3001

SQL LIMIT and OFFSET clause

- To limit the number of rows returned by a select statement, you use the LIMIT and OFFSET clauses.

Syntax:

```
SELECT column_list
FROM table1
ORDER BY column_list
LIMIT row_count OFFSET offset;
```

- The LIMIT row_count determines the number of rows (row_count) returned by the query.
- The OFFSET offset clause skips the offset rows before beginning to return the rows. The OFFSET clause is optional.

Demo

SQL GroupBy Clause

SQL GroupBy Clause

- The GROUP BY is an optional clause of the SELECT statement. The GROUP BY clause allows you to group rows based on values of one or more columns. It returns one row for each group.

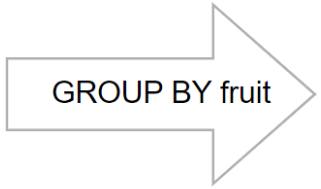
Syntax:

SELECT column1, column2, aggregate_function(column3)

FROM table_name

GROUP BY column1, column2;

id	fruit
1	Apple
2	Orange
3	Apple
4	Banana
5	Orange



fruit
Apple
Banana
Orange

SQL GroupBy Clause

Example

- *SQL GROUPBY clause*

```
SELECT customer_id, YEAR (order_date) order_year
FROM sales.orders
WHERE customer_id IN (1, 2)
GROUP BY customer_id, YEAR (order_date)
ORDER BY customer_id;
```

customer_id	order_year
1	2016
1	2018
2	2017
2	2018

- *SQL GROUPBY clause*

```
SELECT DISTINCT customer_id, YEAR (order_date) order_year
FROM sales.orders
WHERE customer_id IN (1, 2)
ORDER BY customer_id;
```

SQL GroupBy Clause Having

To filter groups based on condition, you use the HAVING clause.

- *SQL GROUPBY clause Having*

```
SELECT customer_id, YEAR (order_date), COUNT (order_id) order_count
FROM sales.orders
GROUP BY customer_id, YEAR (order_date)
HAVING COUNT (order_id) >= 2
ORDER BY customer_id;
```

customer_id	order_year	order_count
1	2018	2
2	2017	2
3	2018	3
4	2017	2
5	2016	2
6	2018	2
7	2018	2
9	2018	2
10	2018	2

HAVING vs. WHERE

- The WHERE clause applies the condition to individual rows before the rows are summarized into groups by the GROUP BY clause.

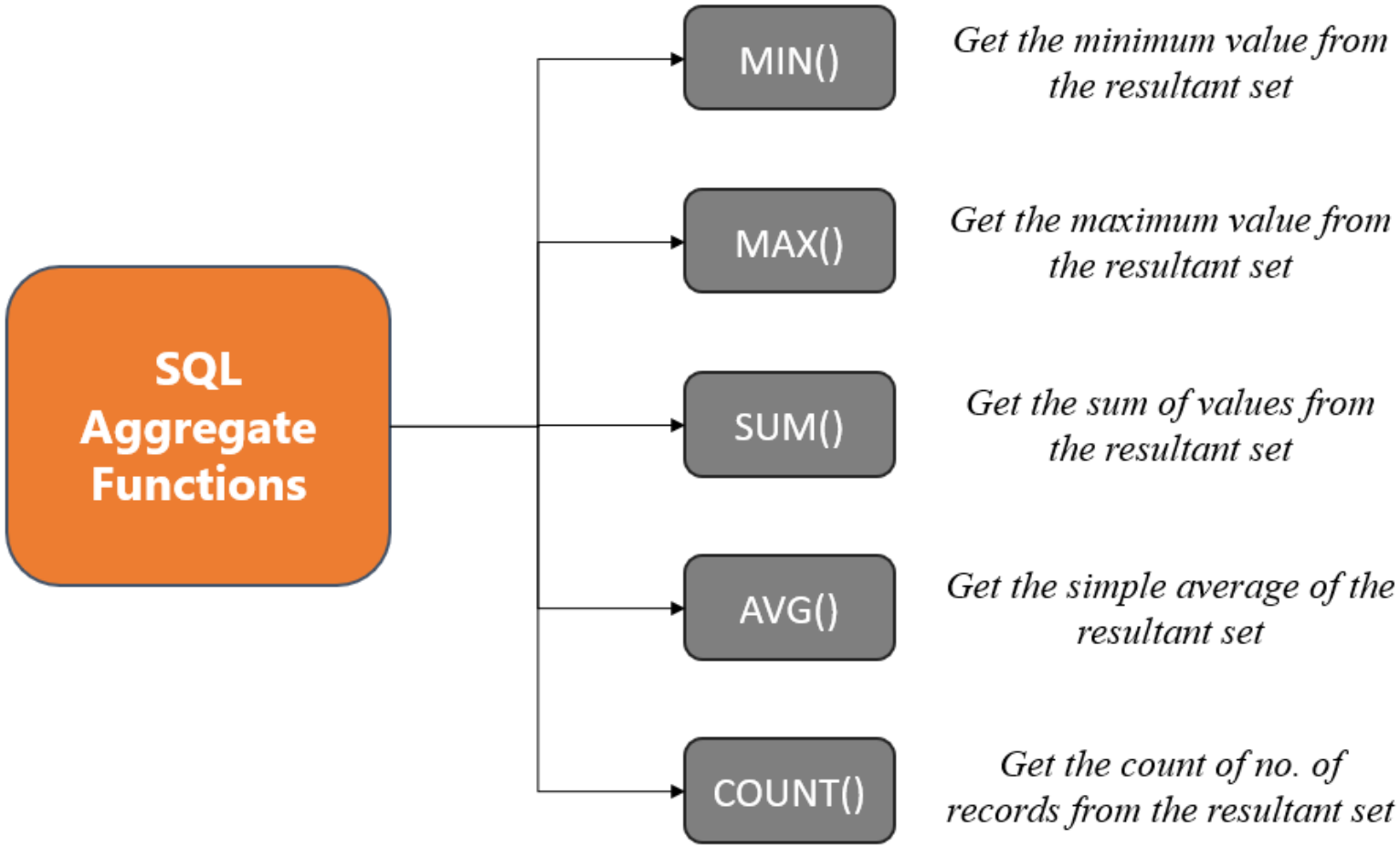
- The HAVING clause applies the condition to the groups after the rows are grouped into groups.

- Therefore, it is important to note that the HAVING clause is applied after whereas the WHERE clause is applied before the GROUP BY clause.

SQL Aggregate Functions

SQL Aggregate function

- Aggregate function is used to perform calculations on multiple values and return the result in a single value like the average, sum, maximum & minimum value among certain groups of values.



SQL Aggregate function

Example

- *SQL COUNT Function*

```
SELECT department_id,
        COUNT(employee_id) as headcount
FROM employees
GROUP BY department_id;
```

department_id	headcount
1	1
2	2
3	6
4	1
5	7
6	5
7	1
8	6
9	3
10	6
11	2

- *SQL SUM Function*

```
SELECT department_id,
        SUM(salary)
FROM employees
GROUP BY department_id;
```

department_id	SUM(salary)
1	4400
2	19000
3	24900
4	6500
5	41200
6	28800
7	10000
8	57700
9	58000
10	51600
11	20300

SQL Aggregate function

Example

- *SQL AVG Function*

```
SELECT department_id,
       AVG(salary)
FROM employees
GROUP BY department_id;
```

department_id	AVG(salary)
1	4400
2	9500
3	4150
4	6500
5	5885.714286
6	5760
7	10000
8	9616.666667
9	19333.33333
10	8600
11	10150

- *SQL SUM Function*

```
SELECT department_id,
       MAX(salary)
FROM employees
GROUP BY department_id;
```

department_id	MAX(salary)
1	4400
2	13000
3	11000
4	6500
5	8200
6	9000
7	10000
8	14000
9	24000
10	12000
11	12000

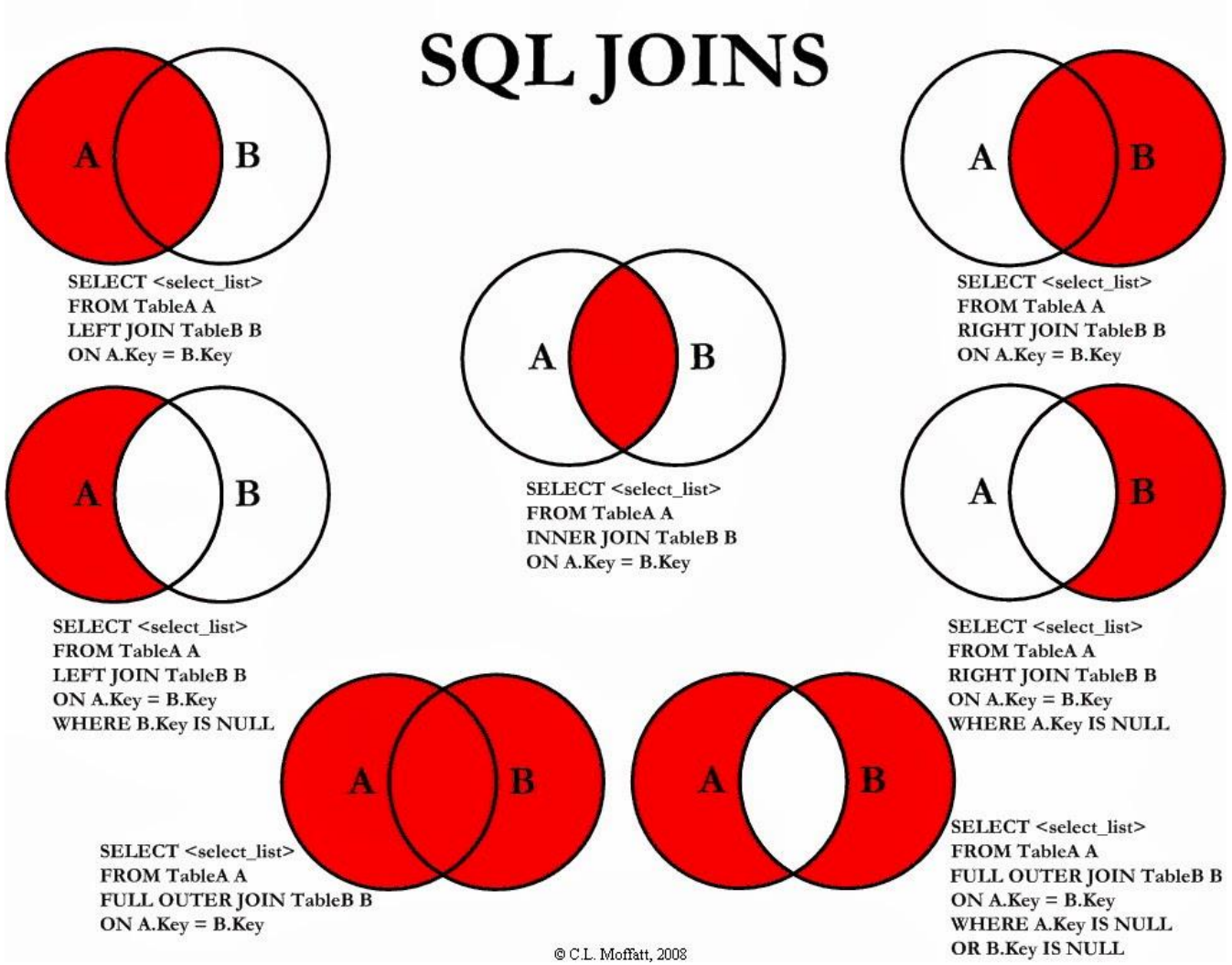
Write MIN function to get minimum salary

Demo

Join Query

SQL Join Query

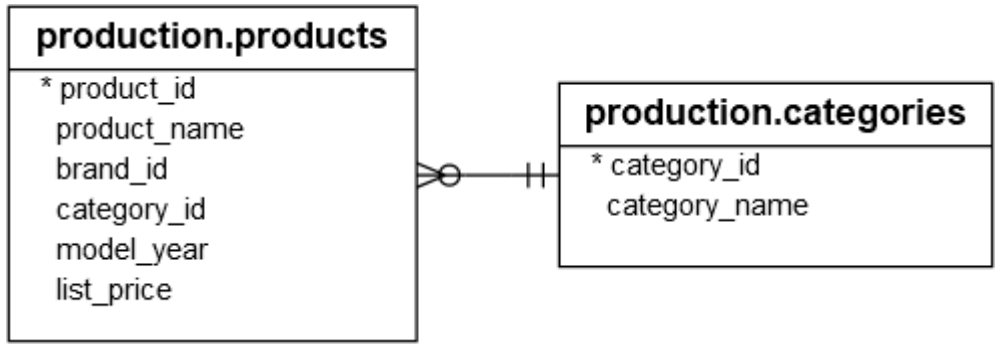
- MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.



Inner Join Query

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON Condition;
```

Syntax




```
SELECT product_name, category_name, list_price
FROM production.products p
INNER JOIN production.categories c
  ON c.category_id = p.category_id
ORDER BY
  product_name DESC;
```

Eg.

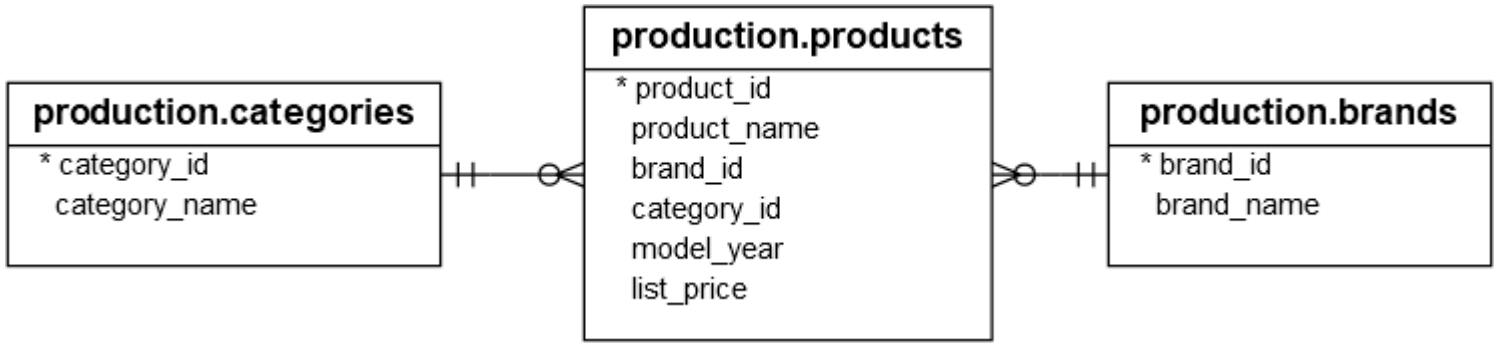
Result

product_name	category_name	list_price
Trek XM700+ Lowstep - 2018	Electric Bikes	3499.99
Trek XM700+ - 2018	Electric Bikes	3499.99
Trek X-Caliber Frameset - 2018	Mountain Bikes	1499.99
Trek X-Caliber 8 - 2018	Mountain Bikes	999.99
Trek X-Caliber 8 - 2017	Mountain Bikes	999.99
Trek X-Caliber 7 - 2018	Mountain Bikes	919.99
Trek Verve+ Lowstep - 2018	Electric Bikes	2299.99
Trek Verve+ - 2018	Electric Bikes	2299.99
Trek Ticket S Frame - 2018	Mountain Bikes	1469.99
Trek Superfly 24 - 2017/2018	Children Bicycles	489.99
Trek Superfly 20 - 2018	Children Bicycles	399.99
Trek Super Commuter+ 8S - 2018	Electric Bikes	4999.99
Trek Super Commuter+ 7 - 2018	Electric Bikes	3599.99
Trek Stache Carbon Frameset - 2018	Mountain Bikes	919.99

INNER JOIN 3 tables example

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON Condition;
INNER JOIN table3
ON Condition;
```

Syntax




```
SELECT product_name, category_name, brand_name, list_price
FROM production.products p
INNER JOIN production.categories c
ON c.category_id = p.category_id
INNER JOIN production.brands b
ON b.brand_id = p.brand_id
ORDER BY product_name DESC;
```

Eg.

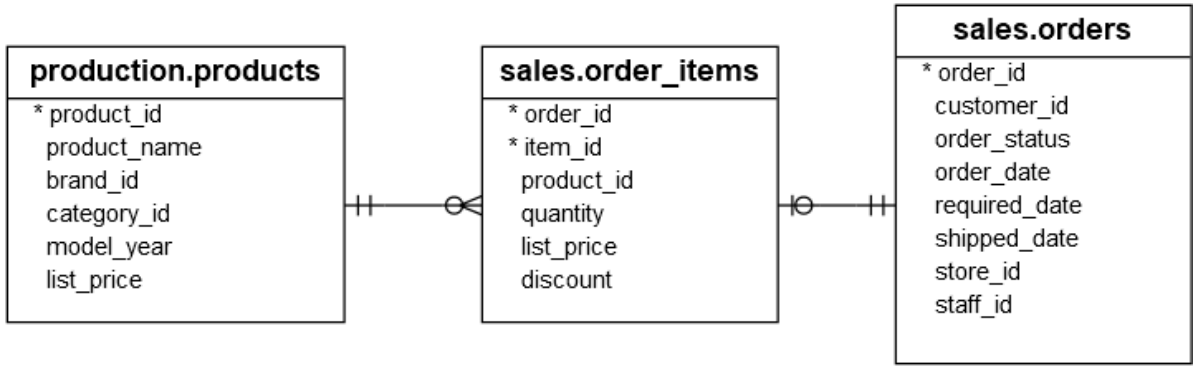
Result

product_name	category_name	brand_name	list_price
Trek XM700+ Lowstep - 2018	Electric Bikes	Trek	3499.99
Trek XM700+ - 2018	Electric Bikes	Trek	3499.99
Trek X-Caliber Frameset - 2018	Mountain Bikes	Trek	1499.99
Trek X-Caliber 8 - 2018	Mountain Bikes	Trek	999.99
Trek X-Caliber 8 - 2017	Mountain Bikes	Trek	999.99
Trek X-Caliber 7 - 2018	Mountain Bikes	Trek	919.99
Trek Verve+ Lowstep - 2018	Electric Bikes	Trek	2299.99
Trek Verve+ - 2018	Electric Bikes	Trek	2299.99
Trek Ticket S Frame - 2018	Mountain Bikes	Trek	1469.99
Trek Superfly 24 - 2017/2018	Children Bicycles	Trek	489.99

LEFT JOIN

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON Condition;
```

Syntax




```
SELECT p.product_name, o.order_id, i.item_id, o.order_date

FROM production.products p
        LEFT JOIN sales.order_items i
                ON i.product_id = p.product_id
        LEFT JOIN sales.orders o
                ON o.order_id = i.order_id

ORDER BY order_id;
```

Eg.

Result

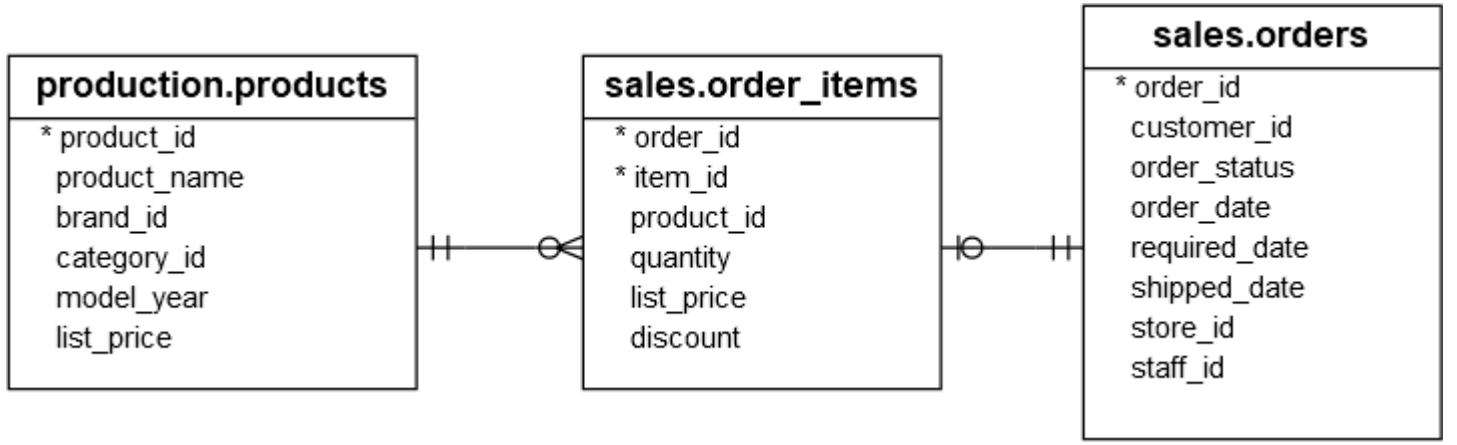
product_name	order_id	item_id	order_date
Electra Savannah 1 (20-inch) - Girl's - 2018	NULL	NULL	NULL
Electra Townie Go! 8i Ladies' - 2018	NULL	NULL	NULL
Trek Checkpoint ALR 5 Women's - 2019	NULL	NULL	NULL
Trek Checkpoint ALR Frameset - 2019	NULL	NULL	NULL
Trek Precaliber 12 Girl's - 2018	NULL	NULL	NULL
Surly Krampus Frameset - 2018	NULL	NULL	NULL
Trek Checkpoint SL 5 Women's - 2019	NULL	NULL	NULL
Trek 820 - 2016	NULL	NULL	NULL
Trek Checkpoint ALR 4 Women's - 2019	NULL	NULL	NULL
Trek Kids' Dual Sport - 2018	NULL	NULL	NULL
Trek Checkpoint ALR 5 - 2019	NULL	NULL	NULL
Electra Sweet Ride 1 (20-inch) - Girl's - 2018	NULL	NULL	NULL
Trek Domane SLR 6 Disc Women's - 2018	NULL	NULL	NULL
Trek Checkpoint SL 6 - 2019	NULL	NULL	NULL
Electra Townie Original 7D EQ - Women's - 2016	1	1	2016-01-01
Trek Remedy 29 Carbon Frameset - 2016	1	2	2016-01-01
Surly Straggler - 2016	1	3	2016-01-01
Electra Townie Original 7D EQ - 2016	1	4	2016-01-01
Trek Fuel EX 8 29 - 2016	1	5	2016-01-01

- non-matching rows in the right table are filled with the NULL values,
- you can apply the LEFT JOIN clause to miss-match rows between tables.

LEFT JOIN 3 tables example

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON Condition;
LEFT JOIN table3
ON Condition;
```

Syntax




```
SELECT p.product_name, o.order_id, i.item_id, o.order_date
FROM production.products p
        LEFT JOIN sales.order_items i
                ON i.product_id = p.product_id
        LEFT JOIN sales.orders o
                ON o.order_id = i.order_id
ORDER BY order_id;
```

Eg.

Result

product_name	order_id	item_id	order_date
Electra Savannah 1 (20-inch) - Girl's - 2018	NULL	NULL	NULL
Electra Townie Go! 8i Ladies' - 2018	NULL	NULL	NULL
Trek Checkpoint ALR 5 Women's - 2019	NULL	NULL	NULL
Trek Checkpoint ALR Frameset - 2019	NULL	NULL	NULL
Trek Precaliber 12 Girl's - 2018	NULL	NULL	NULL
Surly Krampus Frameset - 2018	NULL	NULL	NULL
Trek Checkpoint SL 5 Women's - 2019	NULL	NULL	NULL
Trek 820 - 2016	NULL	NULL	NULL
Trek Checkpoint ALR 4 Women's - 2019	NULL	NULL	NULL
Trek Kids' Dual Sport - 2018	NULL	NULL	NULL
Trek Checkpoint ALR 5 - 2019	NULL	NULL	NULL
Electra Sweet Ride 1 (20-inch) - Girl's - 2018	NULL	NULL	NULL
Trek Domane SLR 6 Disc Women's - 2018	NULL	NULL	NULL
Trek Checkpoint SL 6 - 2019	NULL	NULL	NULL
Electra Townie Original 7D EQ - Women's - 2016	1	1	2016-01-01
Trek Remedy 29 Carbon Frameset - 2016	1	2	2016-01-01
Surly Straggler - 2016	1	3	2016-01-01
Electra Townie Original 7D EQ - 2016	1	4	2016-01-01
Trek Fuel EX 8 29 - 2016	1	5	2016-01-01

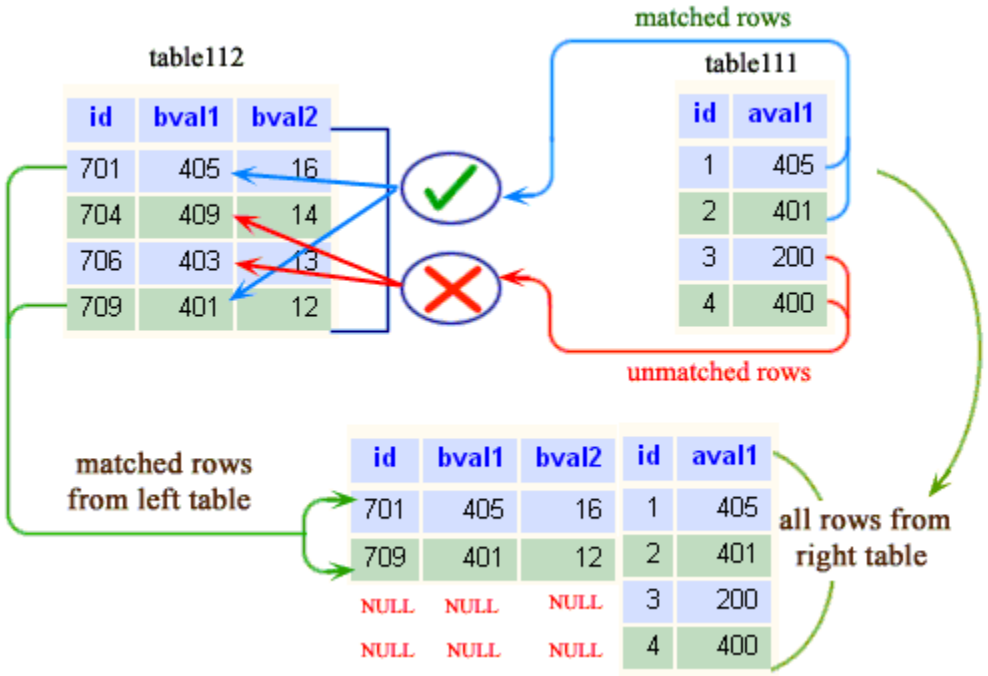
RIGHT JOIN

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON condition;
```

Syntax

```
SELECT product_name, order_id

FROM sales.order_items o

   RIGHT JOIN production.products p

      ON o.product_id = p.product_id

ORDER BY order_id;
```

Eg.



Cross JOIN

- CROSS JOIN produced a result set which is the product of rows of two associated tables when no WHERE clause is used with CROSS JOIN.

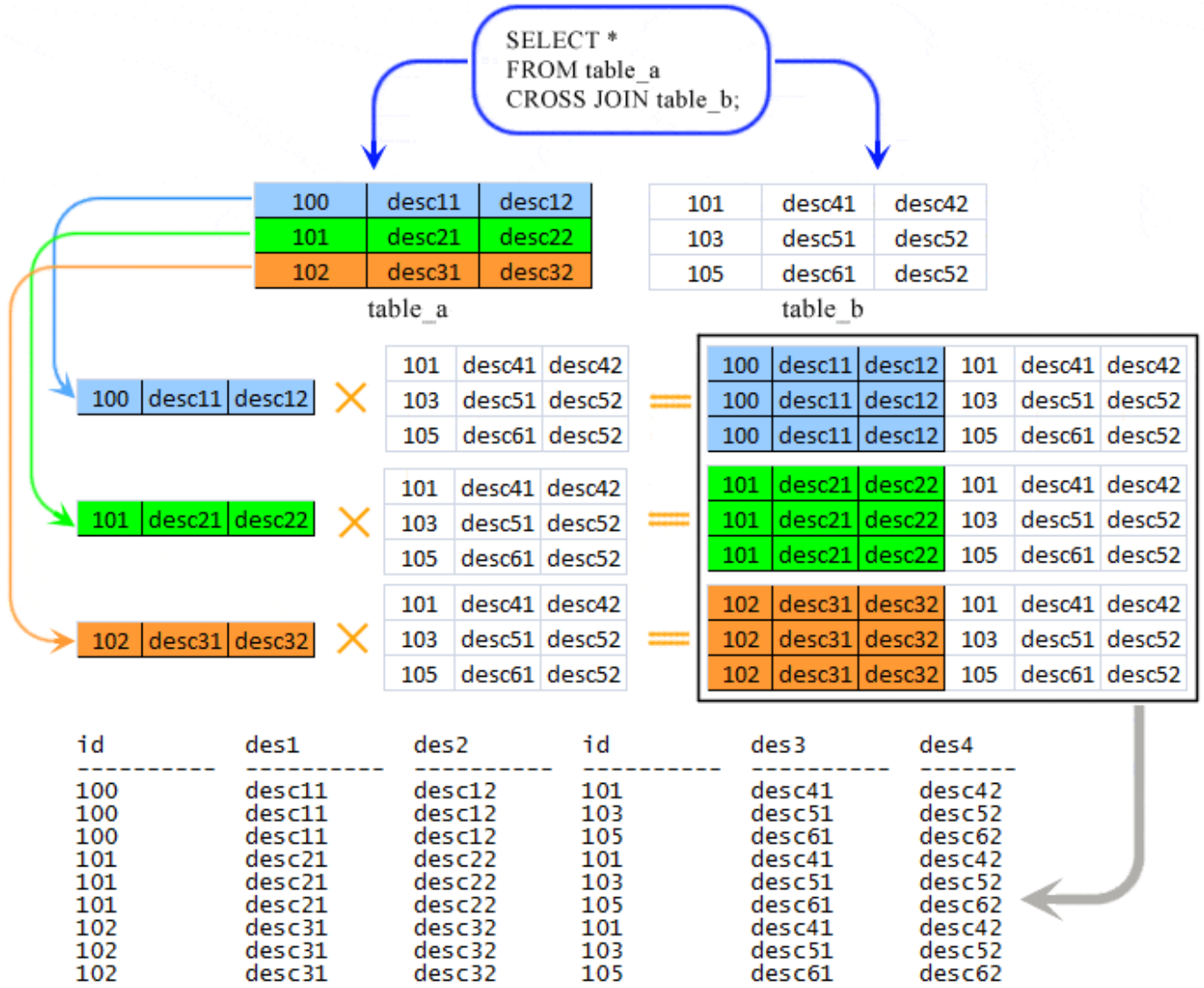
SELECT
product_id, product_name, store_id, 0 AS quantity

FROM production.products

CROSS JOIN sales.stores

ORDER BY product_name, store_id;

Eg.



SELF JOIN

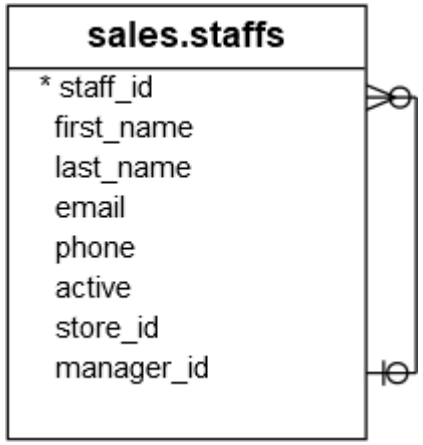
- join a table to itself. This type of join is known as the self-join.
- The manager_id column specifies the manager of an employee. The following statement joins the employees table to itself to query the information of who reports to whom.

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON Condition;
```

Syntax

```
SELECT
   e.first_name || ' ' || e.last_name AS employee,
   m.first_name || ' ' || m.last_name AS manager

FROM sales.staffs e

INNER JOIN sales.staffs m

ON m.staff_id = e.manager_id

ORDER BY manager;
```

Eg.



Result

employee	manager
Mireya Copeland	Fabiola Jackson
Jannette David	Fabiola Jackson
Kali Vargas	Fabiola Jackson
Marcelene Boyer	Jannette David
Venita Daniel	Jannette David
Genna Serrano	Mireya Copeland
Virgie Wiggins	Mireya Copeland
Layla Terrell	Venita Daniel
Bernardine Houston	Venita Daniel

Demo

String Function

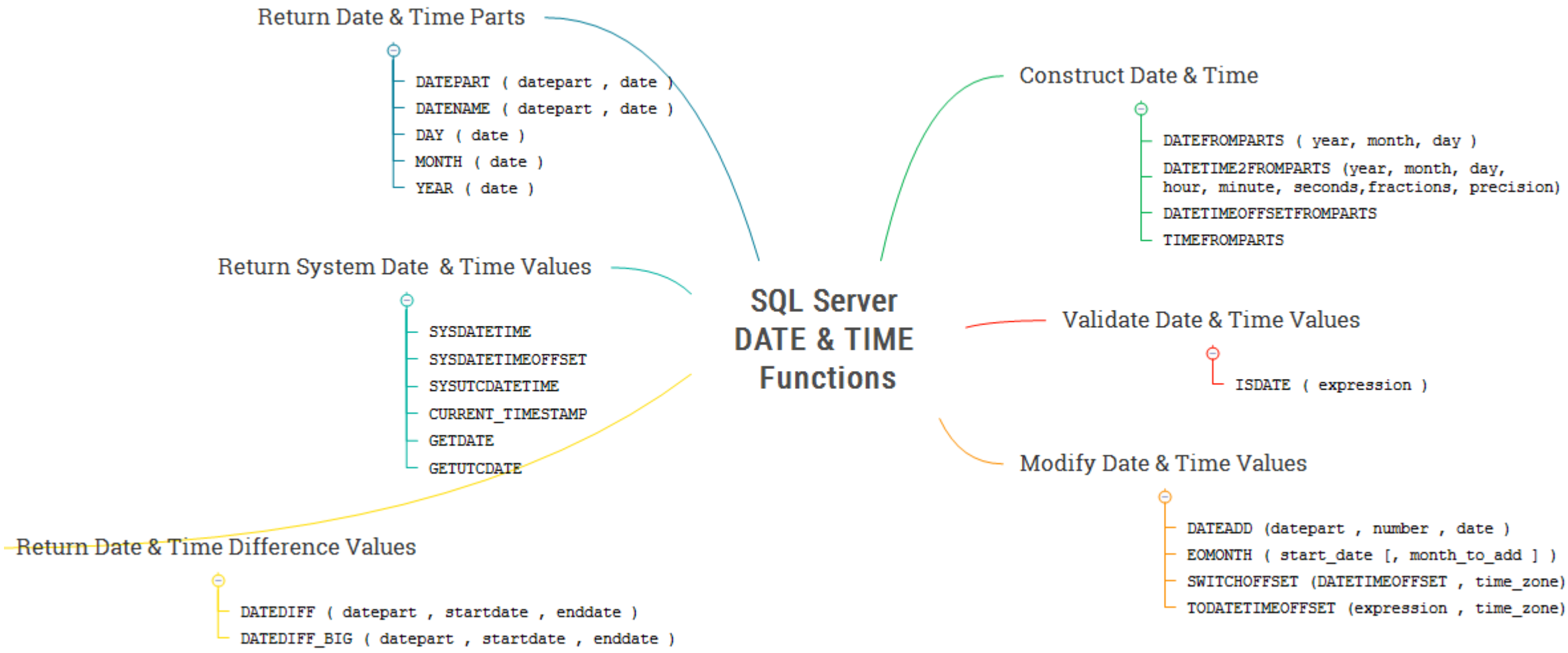
- A string function accepts a string value as an input and returns a string value regardless of the data type (string or numeric).

Category	Function	Description
Position	CHARINDEX	Find position of one or more characters in another value
	LEN	Return the number of characters
	PATINDEX	CHARINDEX on super vitamins...
Transformation	LEFT	Return beginning portion of value
	LOWER	Return value as all lower case characters
	LTRIM	Remove any beginning spaces
	QUOTENAME	Make the value legal for SQL code generation
	REPLACE	Replace one set of characters with another
	REPLICATE	Repeat characters
	REVERSE	Flip the value end to end
	RIGHT	Return the last portion of the value
	RTRIM	Remove any trailing spaces
	SPACE	Create a value of repeated spaces
	STR	Convert a number to a text value.
	STUFF	Insert characters inside another value
	SUBSTRING	Return a portion of a value, such as the middle.
	UPPER	Return value as all UPPER CASE characters
Character set	ASCII	Return the ASCII code for a character
	CHAR	Return the Character for the corresponding ASCII code
	NCHAR	Like CHAR but for UNICODE.
	UNICODE	Like ASCII but for UNICODE.
Soundex	DIFFERENCE	An interesting way to compare differences in strings.
	SOUNDEX	An interesting way to compare strings.

Mathematical Functions

Category	Function	Brief Description
Scientific and Trig Functions	ACOS	Arc Cosine (inverse cosine)
	ASIN	Arc Sine (inverse sine)
	ATAN	Arc Tangent (inverse tangent)
	ATN2	Arc Tangent (inverse tangent)
	COS	Cosine
	COT	Cotangent
	DEGREES	Degrees from Radians
	EXP	Exponent
	LOG	Natural logarithm (ln)
	LOG10	Log base 10
	PI	Apple pie? I think not!
	POWER	Power function
	RADIANS	Radians from Degrees
	SIN	Sine
	SQRT	Square Root
	SQUARE	Square
	TAN	Tangent
Rounding Functions	CEILING	Ceiling
	FLOOR	Floor
	ROUND	Round Number
Signs	ABS	Absolute Value
	SIGN	Determine Sign of Value
Random Numbers	RAND	Generate Random Number

DATE Functions



Demo

Quiz

1

Which of the following columns in a table cannot be updated?
A. DATE type columns in the table
B. Columns which allows NULL values in the table
C. A primary key column which also serves as foreign key reference in another table
D. All of the above

Columns which allows NULL values in the table

2

complete the query to update name as patrik whose age is 43.
update employee _____ name = ‘Patrik’ _____ age = 43

employee SET name = ‘Patrik’ AND age = 43

3

Fill in the blank to select the record with the smallest of the Price column as Small price.

SELECT _____, ProductName
FROM Products;

A. PRICE
B. MIN(PRICE)
C. MAX(PRICE)
D. All the above

Ans: MIN(PRICE)

4

Use the correct function to return the number of records that have the Price value set to 18.

SELECT _____(*)
FROM Products
_____ Price = 18;

A. MAX, WHERE
B. MIN, WHERE
C. COUNT, WHERE
D. AVG, WHERE

Ans: COUNT, WHERE

5

Choose the correct JOIN clause to select all records from the two tables where there is a match in both tables.

```
SELECT *
FROM Orders
_____________________
ON Orders.CustomerID=Customers.CustomerID;
```

A. INNER JOIN Customers
B. RIGHT JOIN Customers
C. FULL JOIN Customer
D. LEFT JOIN Customer

Ans: INNER JOIN Customers

Queries

References

1. https://powerbidocs.com/2019/12/25/sql-keys/
2. https://www.boardinfinity.com/blog/a-quick-guide-to-entities-in-dbms/
3. https://www.sqltutorial.org/
4. https://www.w3schools.com/mysql
5. https://www.w3resource.com/mysql

Thank you

Innovative Services

Passionate Employees

Delighted Customers