SQL notes

Select example:

SELECT firstname

FROM customer

Select statement sequences (logical sequence or syntax sequence):

* SELECT
* DISTINCT
* FROM
* WHERE
* GROUP BY
* HAVING
* ORDER BY

Processing sequence:

* FROM
* WHERE
* GROUP BY
* HAVING
* SELECT
* DISTINCT
* ORDER BY

To select multiple columns separate each column by a comma e.g.:

SELECT column1, column2, column3

FROM table1

To change column heads as shown in the output we can use aliases by include the keyword **AS** this improves readability e.g.:

SELECT phone\_number AS “Phone Number”

FROM customer

To select all columns in a table you can use an asterisk e.g.:

SELECT \*

FROM customer

Example of how to select multiple columns and use AS function at the same time:

SELECT firstname AS "First Name", lastname AS "Last Name", birth\_date AS "Date of Birth"

FROM customer

**WHERE** clause is used to retrieve specific rows e.g.:

SELECT \*

FROM product

Where name = ‘Chromecast’

*Boring note: = symbol is also known as a conditional operator*

Example of how to select multiple columns, use **AS** function and where:

SELECT firstname AS "First Name", lastname AS "Last Name", city AS "City"

FROM customer

WHERE city = 'Montpellier'

Operators can be used in SQL in much the same way as python e.g.:

SELECT name, description, price

FROM product

WHERE price >= 100

**WHERE** clauses can be used for multiple comparisons. You can use **AND** as well as **OR** clauses like in python e.g.:

SELECT name, price, available\_stock

FROM product

WHERE price < 10 OR available\_stock >= 3000;

Wildcards

Sometimes you might want matches which are similar, to do this we can use **LIKE** keyword in a **WHERE** clause. Use an underscore to substitute a single character. The example given would find a customer if their name was “Lisa” or “Liza”:

SELECT \*

FROM customer

WHERE firstname LIKE ‘Li\_a’

The **%** sign can be used to substitute for zero or more characters. It can be used with an **WHERE** clause to find all customers whose name begins with a certain character e.g.:

SELECT

FROM customer

WHERE firstname LIKE ‘J%’

The **%** sign can also be to find if specific characters exist e.g.:

SELECT name, price, description

FROM product

WHERE description LIKE '%HD%'

Square brackets are used to specify multiple possible character to match e.g.:

SELECT \*

FROM product

WHERE name LIKE ‘Kindle Fire [567]’

*This will return results for Kindle 5, Kindle Fire 6 and Kindle Fire 7*

Y

**^** circumflex character can be used to negate characters in square bracket

SELECT \*

FROM product

WHERE name LIKE ‘Kindle Fire [^1234]’

*This will return results for Kindle 5 – 9 NOT 1, 2, 3 or 4*

Wildcards can be combined within a single string e.g.:

SELECT \*

FROM product

WHERE name LIKE ‘[ABC]%’

*This will return all products with a name beginning with A, B or C*

**BETWEEN** can be used to find numbers that fall between two numbers e.g.:

SELECT name, description, price, available\_stock

FROM product

WHERE available\_stock BETWEEN 10 and 20 and price > 100

We can use the IN keyword to return matches with the things we specify within a list (enclosed in round brackets) e.g.:

SELECT firstname AS "First Name", lastname AS "Last Name", city AS "City"

FROM customer

WHERE city IN ('Tokyo', 'Beijing', 'Moscow')

**NULL** can also be use e.g.:

SELECT firstname AS "First Name", lastname AS "Last Name", address AS "Address", phone\_number AS "Phone Number"

FROM customer

WHERE address IS NULL OR phone\_number IS NULL

**TOP** allows you to get the top rows e.g.:

SELECT TOP 5 \*

FROM customer;

**ORDER BY** allows you to sort results alphabetically and on numbers. We can use (ASC) for ascending and (DESC) for descending, if we don’t specify it will use ascending by default e.g.

SELECT \*

FROM customer

ORDER BY lastname ASC, firstname ASC;

If there is a tie in the first column it is resolved based on the next column.

This example uses **TOP** and **ORDER BY**

SELECT TOP 5 name, description, price,

FROM product

ORDER BY price DESC

**DISTINCT** lets us remove duplicate rows from our queries e.g.:

SELECT DISTINCT city

FROM customer

We can use concatenation to connect values, SQL server uses +, where ANSI SQL uses || instead e.g.:

SELECT address ||', '|| city AS "Full Address"

FROM customer

WHERE address IS NOT null AND city IS NOT Null

String Functions

UPPER – turns everything UPPERCASE

LOWER – turn everything lowercase

TRIM – removes whitespace characts at start and end of string

LTRIM – removes whitespace at start of string

RTRIM – removes whites space at the end

LENGTH(text) – returns length of string

LEFT (text, N) – returns leftmost N characters(plural) of string

RIGHT(text, N) – returns rightmost N characters of a string

SUBSTRING(text, start\_index, length) – returns a number of characters equal to Length starting a start\_index from a string

POSITION(substring IN text) - returns the index of the first inistance of the character within the text, or 0 if that character is not within the text

e.g.:

SELECT LEFT(firstname, 1) ||'. '|| LEFT(lastname,1) || LOWER (SUBSTRING(lastname,2 , (LENGTH(lastname) - 1))) AS "Name"

FROM customer;

SELECT SUBSTRING(name, 1, POSITION(':' IN name) - 1) AS "Name", price AS "Price"

FROM product

WHERE POSITION(':' IN name)

DATE

DATEADD(Unit, N, date) – adds an amount of time but does not like decimals.

DATEDIFF(unit,date1, date2) – returns the difference between tow dates

YEAR(date) – extracts the years as an integer

MONTH(date) – extracts the month as an integer

DAY(date) – extracts the day as an integer

CASE – Allows you to affect individual entries instead of every entry. Each condition is specified using WHEN with the outcomes specified using THEN. It needs a ELSE and a END AS, to name the column. These are processed in order

*Don’t forget the comma before case*

SELECT name, description, price, available\_stock,

    CASE

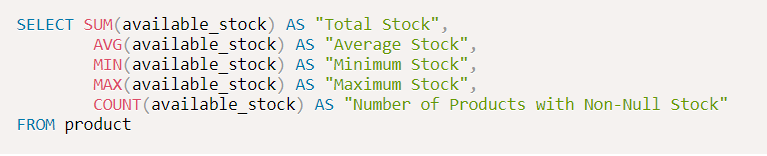
        WHEN available\_stock < 20 THEN 'Low Stock!'

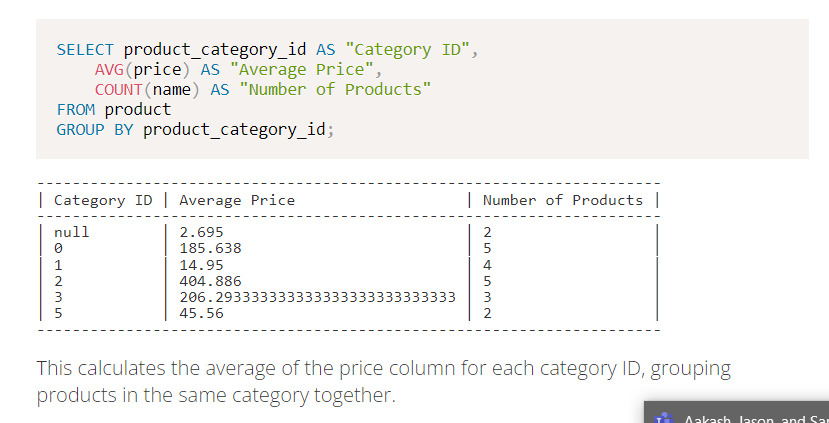
        WHEN available\_stock < 100 THEN 'Limited Stock'

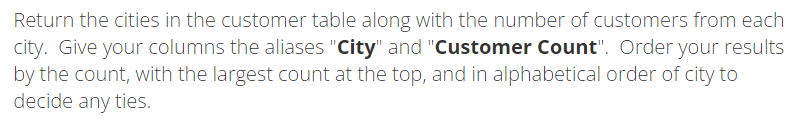
        ELSE 'Well-Stocked'

    END AS "STOCK LEVEL"

FROM product;







SELECT city AS "City",

    COUNT(customer\_id) AS "Customer Count"

FROM customer

GROUP BY city

ORDER BY "Customer Count" DESC;

The GROUP BY function lets group all the data by the parameters set, without it the information would produce multiple results instead of grouping them together by city.

HAVING is a clause used when we want to filter based on the result of aggregation. Do not confuse HAVING and WHERE. WHERE is used to filter rows in the original table. HAVING is used to filter based on aggregation. It is possible to use both together.

SELECT product\_category\_id AS "Category ID",

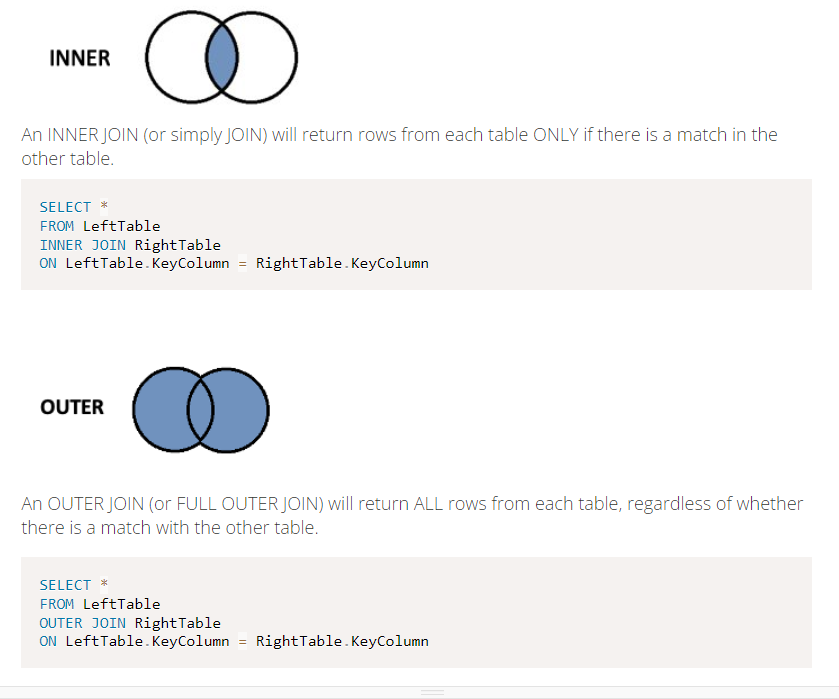
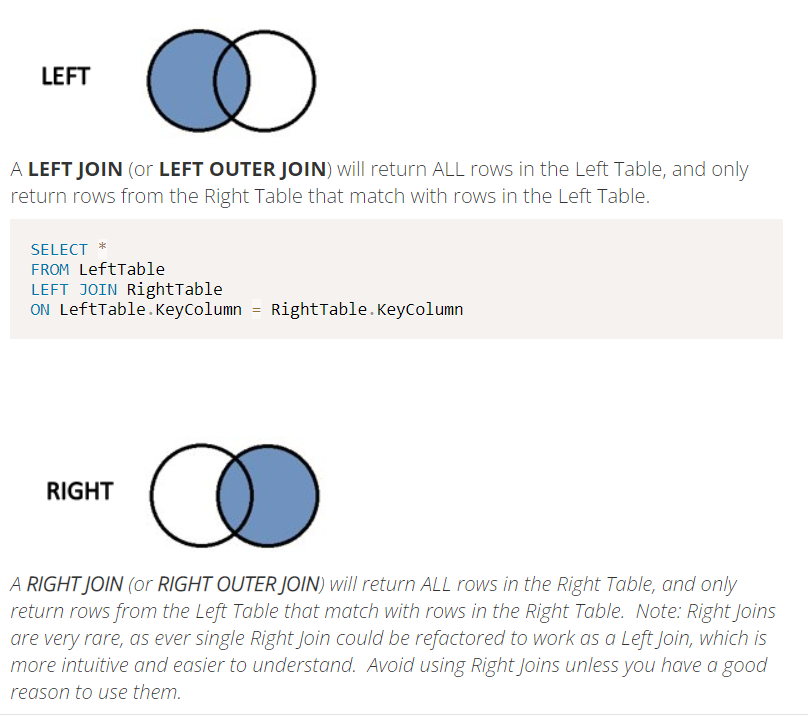
    MAX(price) AS "Maximum Price"

FROM product

GROUP BY product\_category\_id

HAVING MIN(available\_stock) < 100

ORDER BY "Category ID" ASC;

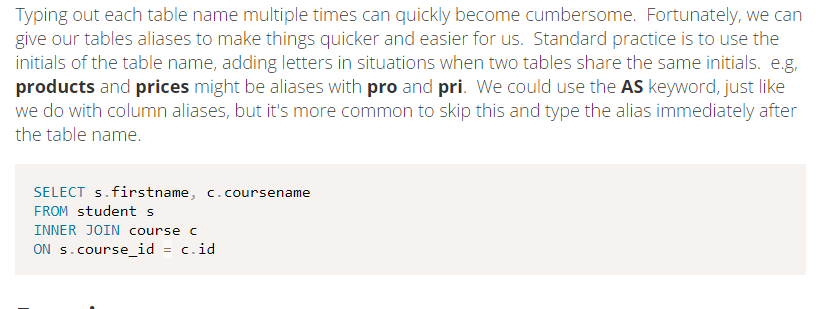


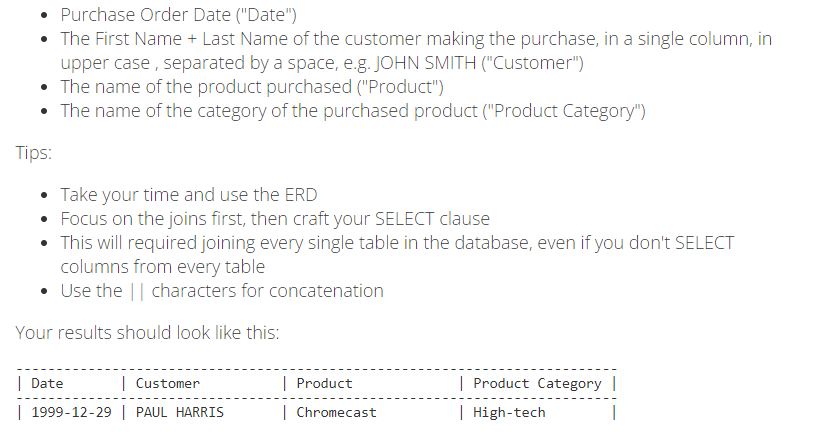
SELECT student.firstname, course.coursename

FROM student

INNER JOIN course

ON student.course\_id = course.id





SELECT po.date AS "Date",

UPPER(c.firstname) ||' '|| c.lastname AS "Customer",

p.name AS "Product",

pc.name AS "Product Category"

FROM purchase\_order po

JOIN customer c

    On c.customer\_id = po.customer\_id

JOIN order\_product op

    On op.order\_id = po.order\_id

JOIN product p

    on p.product\_id = op.product\_id

JOIN product\_category pc

    on pc.product\_category\_id = p.product\_category\_id

SELECT firstname AS "First Name", phone\_number AS "Phone Number"

FROM

(SELECT \*

FROM customer

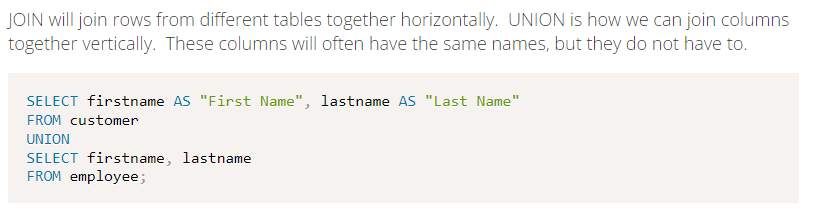
WHERE phone\_number IS NOT NULL)

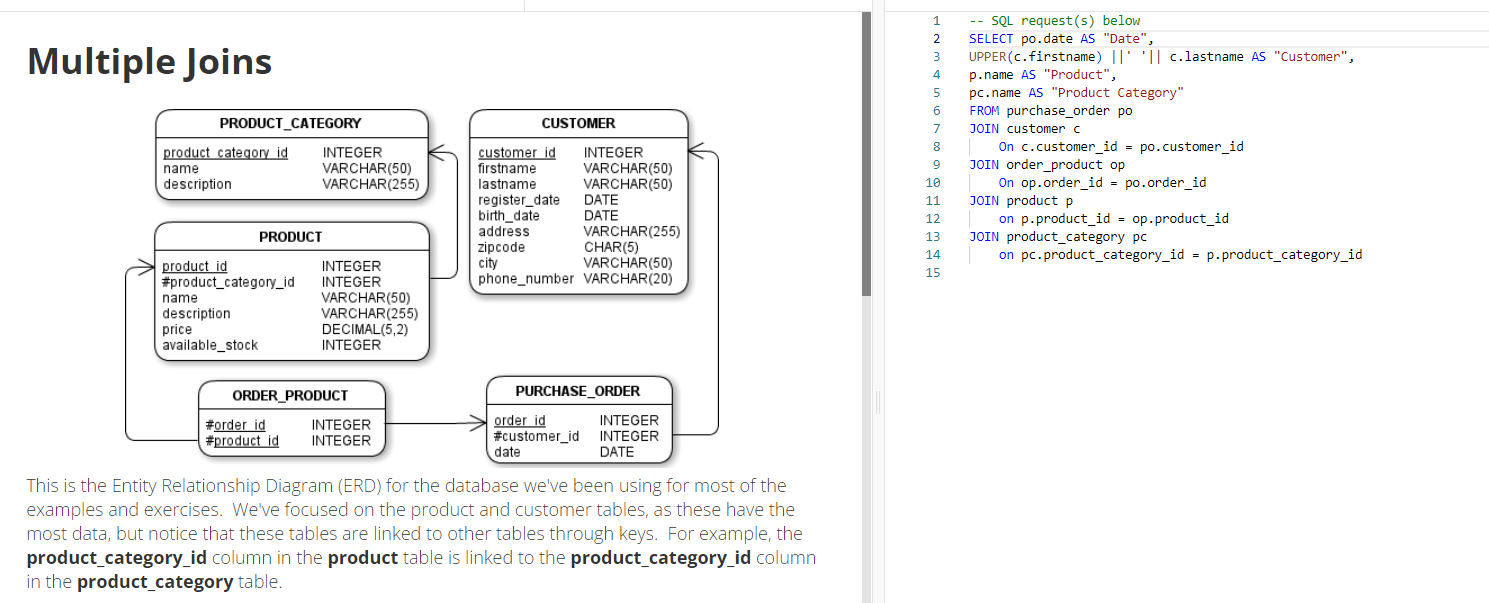
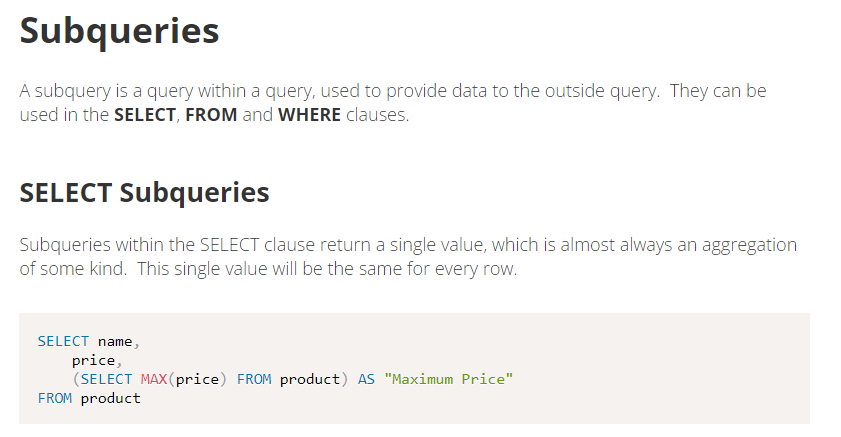
WHERE customer\_id NOT IN(

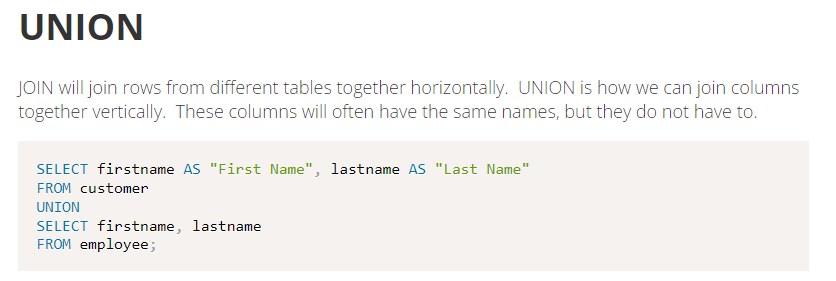
    SELECT DISTINCT customer\_id

    FROM purchase\_order

    );







Good examples

SELECT MAX(Salary) AS "SecondHighestSalary"

FROM Employee

WHERE salary < (SELECT MAX(SALARY) FROM Employee)