SQL Guide

Table of Contents

[The Hierarchy of query in SQL 2](#_Toc68951253)

[Selecting a two columns, naming them, ordering them in ascending order and showing top 30 2](#_Toc68951254)

[Where function 2](#_Toc68951255)

[Floor Ceiling and Rounding 3](#_Toc68951256)

[Best practise name your tables and reference this in the column 3](#_Toc68951257)

[STRINGS. Concat, lower, upper 3](#_Toc68951258)

[DATES 4](#_Toc68951259)

[Finding Nulls 5](#_Toc68951260)

[When and Else Satements 5](#_Toc68951261)

[If Statements 5](#_Toc68951262)

[Wildcards 6](#_Toc68951263)

[Creating databases, tables and rows 6](#_Toc68951264)

[Group BY & Having 6](#_Toc68951265)

[WINDOWS 7](#_Toc68951266)

[Creating views and temporary tables 9](#_Toc68951267)

[Joining Tables 10](#_Toc68951268)

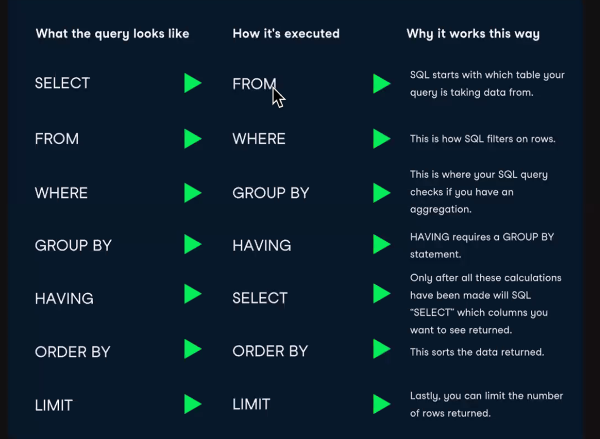
[Self join (Lookup in a duplicate of itself) 11](#_Toc68951269)

[Sub Query 12](#_Toc68951270)

[Terminology about data Architecture 14](#_Toc68951271)

**One site for SQL:** [**https://www.w3schools.com/sql/sql\_operators.asp**](https://www.w3schools.com/sql/sql_operators.asp)

## The Hierarchy of query in SQL



## Selecting a two columns, naming them, ordering them in ascending order and showing top 30

1. Using table ‘bank’. From the district table (which we shall now call ‘d’), select ‘district name’ and A3 (now renamed as region). Order the results in ascending order with a max of 30 rows

use bank;

select d.A2 as district\_name, d.A3 as region

from district d

order by district\_name ASC

limit 30

## Where function

1. Using table ‘bank’. From the loan table (which we shall now call ‘l’), list amount by status and then amount in descending order, Where status is not B or D.

use bank;

select \* from loan l

where l.status not in ('B','D')

order by l.status, l.amount Desc

1. Filter by an amount.

use bank;

select \* from order o

where o.amount > 1000 AND o.k\_symbol=’SIPO’

order by o.amount DESC

1. Searching for bad debts

##what are the top ten biggest unpaid debts (B = debt period finished and not fully paid)

use bank;

select loan\_id, amount, payments

from loan l

where l.status='B' and l.amount > 10000

order by l.amount DESC

limit 10

## Floor Ceiling and Rounding

1. Working out urban population

use bank;

select round(A4\*(A10/100) ,2) as urbanpop, A2

from district

1. Floor and ceiling. – rounds up or down to integers

use bank;

select floor(A4\*(A10/100)) as urbanpop, A2

from district

## Best practise name your tables and reference this in the column

Good idea to reference the table you are looking at every time. (because a lot of tables have same solumn names meaning the different things).   
This is why we give the table a 1 letter name e.g. ‘l’. Then reference it as l.orders, l.type etc

## STRINGS. Concat, lower, upper

1. Strings. Take left 3 digits from k\_symbol and Concat with id string and make it all lower case

#strings

use bank;

select \*, concat(lower(left(k\_symbol,3)),' ',account\_id) as symbolid

from order1

1. Random allocation and playing with strings

use bank;

select concat(lower(A2),upper(A3),left(A2,5))

from district

## DATES

Today () – returns today’s date

1. Converting date to readable

use bank;

select account\_id, date, convert(date,datetime)

from trans

1. This gives you a nice date format

use bank;

select account\_id, date, date\_format(convert(date,datetime), '%D-%M') as day\_n\_month

from trans

1. Substring\_index (value, ‘index around this thing’, what part?). So here it is using the space as the split in the value then taking the 1st part.

use bank;

select \*,card\_id, date\_format(convert(substring\_index(issued, ' ',1),datetime), 'Date is %D-%M\_%Y') as date23

from card

where type = 'gold'

1. Another way using left() – just takes first 6 characters from the left

use bank;

select \*,card\_id, date\_format(convert(left(issued, 6),datetime), 'Date is d of %M of %Y') as date23

from card

where type = 'gold'

1. Finding first year that gold cards were issued

use bank;

select date\_format(convert(left(c.issued,6),date),'%Y') as year

from card c

where type = 'gold'

order by year ASC

limit 1;

## Finding Nulls

select isnull(k\_symbol) from bank.trans;

## When and Else Satements

1. Printing statements depending on status

select loan\_id, account\_id,

case

when status = 'A' then 'Good - Contract Finished'

when status = 'B' then 'Defaulter - Contract Finished'

when status = 'C' then 'Good - Contract Running'

else 'In Debt - Contract Running'

end as 'Status\_Description'

from bank.loan;

## If Statements

1. An if statement to do the same thing. If(this, then this, if not this) and call it decr:

select \*,

If(type='PRIJEM','credit','withdrwal') as descr

from bank.trans;

Note the **IF statements are not so often used**. Probably use a WHEN statement instead. Where you do sue it it’s in one line, perhaps to do a true, false statement like these:

1. Is it city of country?

SELECT \*, if(A10>80, 'URBAN', 'COUNTRY') as locale

From district;

1. Or on a sum result true or false

SELECT \*, if(sum(l.payments/l.amount) > 0.5, 'Looking Good', 'Not much paid back') as how\_going\_so\_far

from loan l;

## Wildcards

https://www.w3schools.com/sql/sql\_wildcards.asp

WHERE City LIKE ‘L\_n\_on’; = find cities with names structured like this.

Also find: BETWEEN, IS, OR, AND, REGEXP

## Creating databases, tables and rows

1. Create database

create database if not exists bank\_demo

1. Table

create table bank\_demo.account(

account\_id int(11) UNIQUE NOT NULL,

district\_id int(11) DEFAULT NULL

)

1. Insert a row

insert into bank\_demo.account values

(1,4),(2,4),(3,4);

## Group BY & Having

1. Group by status and duration. – When you use Group By must aggregate the results so that you get one result for the aggregation:

use bank;

SELECT l.duration, l.status, ,round(avg(l.amount-l.payments),2) as balance

from loan l

Group by l.duration, l.status

order by l.duration, l.status DESC;

1. Where Statements. Same as above but with a more complex where statement

use bank;

SELECT l.duration, l.status, l.account\_id,round(l.amount-l.payments,2) as balance

from loan l

where status <> 'A' AND (duration = 12 OR duration = 36)

Group by l.duration, l.status

order by l.duration, l.status DESC;

**HAVING**

1. HAVING happens after GROUP BY, so in this instance it shows all districts with over 100 customers. A where statement would not to this because it comes before GROUP BY (ie before aggregation)

select c.district\_id, count(\*) as num\_customers

from bank.client c

group by c.district\_id

having count(\*) > 100

order by num\_customers desc;

1. HAVING is all about using the grouping from GROUP BY to filter again. So here the Where statement will deliver individual records over 1000, HAVING perfroms same filter but after the GROUP BY – sou you get over 22,000 by aggregate group ids.

select account\_id, sum(round(amount-payments,2)) as bal

from loan

where round(amount-payments,2) > 10000

group by account\_id

having sum(round(amount-payments,2)) > 22000

order by sum(round(amount-payments,2))

## WINDOWS

**Graphical user interface, application

Description automatically generated**

Window Functions perform a function without aggregating into one row.

SUM, COUNT, RANK etc are examples

**Use the OVER word – because the function is applied to = Over**

1. Here we get average balance by each duration. Average is the window function over partition by duration.

select loan\_id, account\_id, amount, payments, duration,

amount-payments as "Balance",

#window function

avg(amount-payments) over (partition by duration) as "Avg\_Balance"

#this is averaging the balance only by different durations.

from bank.loan

where amount > 100000

order by duration, balance desc;

1. Rank is the most common Window function

select \*, rank() over (partition by status order by amount DESC) as 'rank'

FROM bank.loan;

#Here rank is just giving each entry a rank number based on amount. Status is also splitting and then creating this rank for each status.

#over = what we chose to apply this function to

#note - here 'order by' comes inline - but this is an exception. Only for rank and a couple of other window functions.

1. Other window functions are available. They hang on the key word ‘over’. Here is a piece of coding using row\_number functions

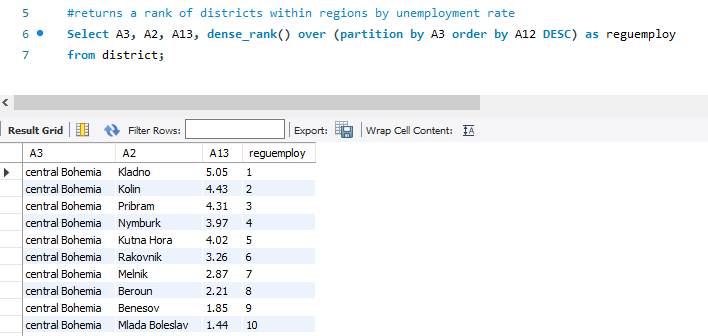
select title, length, row\_number() over (order by length ASC)

from film

1. Use partitions to group by a common variable. So this is creating a rank based on length but divided (grouped) into ratings.

select title, length, rating, RANK() over (partition by rating order by length DESC)

from film



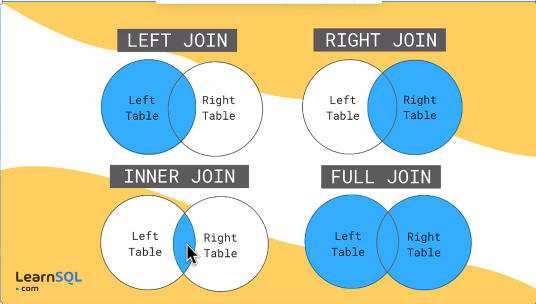
## Creating views and temporary tables

Create view this as:

CREATE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'Brazil'OR  
Create temporary this1 as:

Code

## Joining Tables



1. This is an inner join

SELECT \* from bank.account a

Join bank.loan l

on a.account\_id = l.account\_id

Need to identify Primary Keys and Foreign Keys  
Primary Key is unique to each table and   
Foreign Key is common across tables and it is what we use to join tables.

1. Join is just a query through two tables. Like this:

#Q: show all data for all accounts which have any loans of a 12 month duration

select a.account\_id, a.district\_id, a.frequency from bank.account a

join bank.loan l

on a.account\_id = l.account\_id

where l.duration = 12

1. Left Join delivers all the rows in the primary (first mentioned table) + the rows from the secondary table that share the common identified. The issue is that many tables will not have the common id across all its entries. – So this is a way of prioritising tables. – **“print all of left and what you can identify from right”.** Right join does the opposite.

select a.account\_id, a.district\_id, a.frequency from bank.account a

left join bank.loan l

on a.account\_id = l.account\_id

Note: Full Join is not supported in Mysql.

**Joining multiple tables**

1. **Here we join 3**

#joining 3 tables (well 2 tables and the view you created).

Select \* from client c

Join district d

on c.district\_id = d.A1

Join region\_unemployment\_breakdown v

on v.A2 = d.A2;

1. **Joining 3 tables – another example:**

# total loans and average loans per district.

Select d.A2, sum(l.amount) as da, floor(avg(l.amount)) as avgloan, count(l.loan\_id) from loan l

Join account a

on l.account\_id = a.account\_id

Join district d

on a.district\_id = d.A1

GROUP BY d.A1

Order by avgloan desc;

## Self join (Lookup in a duplicate of itself)

Same sytax as a join but it performs a very different function. Vlookup and create new column. (vlookup + conditions)

1. Harry potter example – the table joins to itself.

SELECT m1.title as movietitle , m1.released as movieyear , m2.title as sequeltitle, m2.released as sequelyear

FROM Harry.movies m1

left join Harry.movies m2

on m1.sequel\_id = m2.movie\_id

1. Checking for fraud

use bank;

#This creates a list of account ids with both of the 'type'. SO we are looking for owners and disonents of the loach (could be fruadulent)

SELECT

d1.client\_id, d1.account\_id, d1.type

FROM

disp d1

JOIN

disp d2 ON d1.account\_id = d2.account\_id

WHERE

d1.type <> d2.type;

**Cross Joins**

Where you have two columns and you want to display all possible combinations of the two. Very rarely used.

## Sub Query

Sub queries are in brackets, they execute first and are often used as part of a Where statement.

1. Sub query gets the average loan amount to filter against. This shows customers with bigger than average amounts.

Select \*

from loan

Where amount > (select avg(amount) from loan);

1. Using a sub query to work out the average of a count:

use bank;

#Find out the average number of transactions by account. Get those accounts that have more transactions than the average.

select account\_id, count(trans\_id) as numtrans

from trans

group by account\_id

having numtrans > (

select floor(avg(numtrans)) from

(select account\_id, count(trans\_id) as numtrans

from trans

group by account\_id) q1

)

#Note. to save complexity you might just work out the average (192) and hard code it into a simple look up against county of transactions

**Nice piece of code to check the values across tables to make sure you have the same count of eg. Districts:**

select count(distinct d.A1) from district d;

select count(distinct c.district\_id) from client c;

## Terminology about data Architecture

1. Data Lake = non-structured data (pool of stuff). Collect loads of data from different places. Some sort of blending? But don’t necessarily need to join data together.   
   **A managed lake** – is more catalogued and refined + delivers useable formats for other tools.
2. Data Warehouse = Copy of transactional data structured for querying and reporting. Supports normalized and denormalized data (de-normalized data = joined up data. Makes it easier to report against). Mostly used for data mining and dashboards. Is resource and cost heavy.
3. Online Transactions Processing (OLTP) is a store of transactional information. It stores current transactional data – there are no redundancies.  
   Online Analytical Processing (OLAP). System designed for relatively small no of transactions. Stores historic data. Queries are more complex. Support different schemas – snowflake   
   Real Time Analysis Processing (RTAP). Uses cloud. Real time data. More flexibility. Real time data and historic data. Most businesses demand real time data today. EG Google Cloud. Data Bricks.

Different Schemas

