

Краткая информация по языку SQL

Подходит для людей с минимальным опытом в использовании баз, содержит примеры и описания всех функций, различные варианты для MSSQL, MySQL и Postgres. Также разбирается использование JOIN'ов и оконных функций.

Данный документ не является официальным учебным пособием, но призван помочь начинающим пользователям в освоении SQL и служить подсказкой и напоминанием для уже опытных пользователей.

Информация по обозначениям:

Название раздела

основной комментарий по разделу

* добавочный комментарий

код

* Разделитель ';' не обязателен во всех базах данных, но лучше писать команды с ним.

* Регистр команд можно не учитывать: select = SELECT.

Получить все данные из существующей таблицы:

```
select * from my_table;
```

Получить 5 или более верхних строчек из базы данных:

Для MSSQL

```
select top 5 * from my_table;
```

Для остальных баз

```
select * from continents limit 5;
```

Получить определенные колонки:

```
select column1, column2 from my_table;
```

Вычисления в колонках:

```
select person_name, population+10000 from my_table;
```

Чтобы задать определенное название колонки в выдаче, используем конструкцию:

```
select person_name as fio from my_table;
```

* 'as' в большинстве баз можно не писать

Операции сложения строк:

```
select first_name+last_name fio from my_table;
```

```
select first_name+' '+last_name as fio from my_table;
```

```
select concat(first_name, ' ', last_name) fio from my_table;
```

Математические функции:

```
select abs(-1000) # вернет 1000, только один аргумент принимает
```

```
select square(8) # возводим в квадрат
```

```
select power(8,2) # возведение 8 в степень 2
```

```
select sqrt(64) # извлекаем корень
```

```
select square(sqrt(64))
```

```
select pi()
```

```
select round(pi(),2) # округлим число pi до 2х знаков после запятой
```

```
select pi()*square(100) # считаем площадь круга по формуле pi r квадрат
```

```
select round(pi()*square(100),0)
```

Фильтрация строк, проверка на равенство:

```
select * from my_table where age=23;
```

```
select * from my_table where first_name='Ilon';  
select * from my_table where age>=23;  
select * from my_table where age>=23 and age<=23;
```

Фильтрация строк, логические операторы not/and/or:

```
select * from my_table where not age=23;  
select * from my_table where age=23 and gender='M';
```

Фильтрация строк, дополнительные операторы in/between:

```
select * from my_table where salary in (2000,2100,2200);  
select * from my_table where not salary in (2000,2100,2200);  
select * from my_table where salary between 1000 and 1900;
```

Паттерн like (% - любые символы в любом количестве, _ - один символ):

```
select * from my_table where manager_name like 'J%'; # находим все имена менеджеров,  
начинающиеся на J  
select * from my_table where manager_name like '_____'; # только имена менеджеров,  
состоящие из 5 букв
```

Сортировка в SQL:

```
select * from my_table order by salary; # сортировка по значению заработной платы. По  
умолчанию сортировка по возрастанию
```

Для изменения направления сортировки используются ключи:

ASC – сортировка по возрастанию (ascending)

DESC – сортировка по убыванию

```
select * from my_table order by salary desc;  
select * from my_table order by age ASC, male DESC;  
select * from my_table where gender='M' order by salary;  
select top 5 * from my_table where gender='F' order by age;  
select * from my_table where gender='F' order by age limit 5;
```

Функции для работы со строками (len/length, ltrim, rtrim, trim, upper, lower):

Определение длины строки

length – для MySQL

len – остальные базы

```
select len('это строка') # получим 10 символов
```

```
select first_name, length(first_name) from my_table;
```

ltrim – убираем пробелы в начале строки

rtrim – убираем пробелы в конце

trim – убираем пробелы с обеих сторон

в MSSQL ltrim может убрать пробелы в начале и в конце строки в зависимости от версии

```
select ltrim('  Моя строка  ') # получим 'Моя строка '  
select first_name from my_table where trim(first_name)='Ellis'
```

upper – перевод всей строки в верхний регистр

в MySQL можно еще использовать ucase

lower – перевод строки в нижний регистр

в MySQL можно еще lcase

```
select upper('строка') # получим 'СТРОКА'  
select lower('СТРОКА') # получим 'строка'
```

* Необходимо отметить, что база может быть чувствительна к регистру, т.е. 'AB' != 'ab' и наоборот не чувствительна к регистру, т.е. 'AB' = 'ab', для быстрой проверки данной ситуации можно использовать команду:

```
select case when 'A' = 'a' then 'Не чувствительна'  
else 'Чувствительна'  
end
```

Выделение подстроки:

```
select substring('мояСтрокамая',4,6) # выделяем строку, начиная с 4 символа, длиной 6  
символов. Получим 'Строка'
```

можно задавать длину строки формулой

```
select substring('мояСтрокамая',4,len(my_str)-3)  
select substring('мояСтрокамая',4,len(my_str)-len(my_str2))
```

ЗАДАЧА: сделать заглавной только первую букву имени.

Решение:

```
select concat(upper(substring(manager_name,1,1)),  
lower(substring(manager_name,2,len(manager_name)-1))) manager_name  
from my_table
```

Поиск наличия символа в строке:

в MSSQL используем charindex

```
select first_name from my_table where charindex('J', first_name)=1
```

в MySQL используем locate

```
select first_name from my_table where locate('J', first_name)=1
```

аналоги – instr, position

```
select last_name from my_table where instr(last_name, 'J')=1
```

```
select last_name from my_table where position('J' in last_name)=1
```

в Postgres есть position и strpos

```
select first_name from my_table where strpos(first_name, 'J')=1
```

* Есть универсальный способ, но он потребляет больше ресурсов и времени:

```
select first_name from my_table where first_name like 'J%'
```

В sql есть функция replace, меняющая одни символы в строке на другие:

```
select replace(first_name, 'opr', '') first_name  
from my_table where strpos(first_name, 'opr')>0
```

Работа с датой и временем

Получаем текущее время:

```
select CURRENT_TIMESTAMP;  
select getdate() # аналог, в MySQL это now()  
select getUTCdate() # нет в MySql
```

В MySQL и Postgres есть

```
select now()  
select localtime # в Postgres дает только время  
select localtimestamp  
select utc_timestamp # нет в Postgres
```

Конвертация (1й способ):

MSSQL

```
select cast(getdate() as date)  
select cast(getdate() as time)  
select cast(current_timestamp as time)
```

функция convert – имеет более расширенный функционал

```
select convert(Date, getdate())
```

в MySQL все аналогично, но вместо getdate() используем now(), а в convert другой порядок

```
select convert(now(), date)
```

для краткости можно сделать так

```
select date(now())
```

в Postgres есть cast(), но вместо convert используем ::

```
select now()::date
```

```
select now()::time
```

Конвертация (2й способ):

MSSQL

```
select current_timestamp time_default,  
format(current_timestamp, 'dd-mm-yyyy hh:mm:ss;') time_formated
```

можно в format использовать любые комбинации

```
select current_timestamp time_default,  
format(current_timestamp, 'yyyy') time_formated
```

MySQL, можно также разные комбинации

```
select current_timestamp time_default,  
date_format(current_timestamp, '%d-%m-%Y %H:%i:%s') time_formated
```

Postgres, можно также разные комбинации

```
select current_timestamp time_default,  
to_char(current_timestamp, 'dd-MM-yyyy hh:MI:ss') time_formated
```

Выделение отдельных частей времени:

MSSQL

```
select current_timestamp datetime_default,  
year(current_timestamp) year,  
month(current_timestamp) month,  
day(current_timestamp) day
```

```
select current_timestamp datetime_default,  
datepart(hour, current_timestamp) hour,  
datepart(minute, current_timestamp) minute,  
datepart(second, current_timestamp) second
```

#MySQL

```
select current_timestamp datetime_default,  
year(current_timestamp) year,  
month(current_timestamp) month,  
day(current_timestamp) day,  
hour(current_timestamp) hour,  
minute(current_timestamp) minute,  
second(current_timestamp) second
```

```
select current_timestamp datetime_default,  
extract(hour from current_timestamp) hour,  
extract(year from current_timestamp) year
```

в Postgress есть 2 универсальные функции

```
select current_timestamp datetime_default,  
date_part('hour', current_timestamp) as hour,  
date_part('year', current_timestamp) as year
```

Создание новых значений дат и времени:

самый простой способ – задаем в виде строки

```
select * from base where date_var='2020-11-12'
```

* Необходимо учитывать тип данных в колонке.

Даты можно сравнивать, используя > / >= / <= / <, приведя к одному формату

можно отбросить отдельные части даты (секунды, часы, дни и т.д.) для увеличения диапазона сравнения

* Сравнение дат с помощью like работает только в MySQL

try_convert(date, date_column)

str_to_date(date_string, '%Y%m%d') для MySQL

```
select * from my_table where  
str_to_date(date_bad_string, '%Y year %m month %d day')='2020-08-20'
```

в Postgres используется to_date

```
select * from my_table where  
TO_DATE(date_bad_string, 'YYYY "year" MM "month" DD "day")='2020-08-20'
```

MSSQL

```
select DATEFROMPARTS(2020,11,28)
select DATETIMEFROMPARTS(2020,11,28,13,48,50,0)
select cast(DATETIMEFROMPARTS(2020,11,28,13,48,50,0) as time)
```

Postgres

```
select make_date(2020,11,28)
select make_time(13,48,50)
select make_timestamp(2020,11,28,13,48,50)
```

MySQL только через date(concat(year,'-',month,'-',date))

Прибавление времени к текущей дате:

MSSQL

```
select current_timestamp,
dateadd(minute,1,current_timestamp),
dateadd(hour,3,current_timestamp)
```

получим строки, которые начинаются сегодня и умещаются в 15-дневный срок

```
select * from my_table_dates where
date_column>=current_timestamp
and
date_column<=dateadd(day,15,current_timestamp)
```

MySQL

```
select current_timestamp,
date_add(current_timestamp, interval 1 minute),
date_sub(current_timestamp, interval 1 month),
date_add(current_timestamp, interval -1 day)
```

получим строки, которые начинаются сегодня и умещаются в 15-дневный срок

```
select * from my_table_dates where
date_column>=current_timestamp
and
date_column<=date_add(current_timestamp, interval 15 day)
```

Postgres

```
select current_timestamp,
current_timestamp + interval '1 minute',
current_timestamp + interval '4 hour'
```

получим строки, которые начинаются сегодня и умещаются в 15-дневный срок


```
select * from my_table_dates where  
date_column>=current_timestamp  
and  
date_column<= current_timestamp + interval '15 day'  
# выбор дат, которые есть в заданном промежутке
```

```
select * from my_table_dates  
where birth_date in ('1987-10-22', '2000-08-10')
```

Агрегатные функции (есть еще оконные агрегатные функции, о них далее)

запрос количества строк

```
select count(*) from my_table  
select count(seller) from my_table
```

считаем сумму sum()

```
select sum(salary) from my_table
```

расчет среднего значений avg()

```
select avg(salary) from my_table
```

поиск минимального значения min() – можно также использовать с датой и строками

```
select min(salary) from my_table  
select min(date_column) from my_table
```

поиск максимального значения max()

```
select max(salary) from my_table  
select max(date_column) from my_table
```

использование агрегатных функций с фильтрами

```
select count(salary) salary, sum(salary) sum_salary  
from my_table where salary_cur='USD'
```

distinct дает только уникальные значения

```
select distinct salary_cur from my_table
```

distinct можно использовать для комбинации строк

```
select distinct salary, salary_cur from my_table
```

можно сосчитать количество уникальных строк

```
select count(distinct salary_cur) from my_table
```

вместо distinct можно использовать группировку group by по нужному столбцу

Группировка group by

```
Select count(*) from my_table group by salary_currency
```

```
select salary_currency, count(*) from my_table group by salary_currency  
select manager_id from my_table where manager_name = 'Jhon'  
group by manager_id
```

Фильтрация результатов группировки с использованием having

```
select payment_method, count(*) count from my_table  
group by payment_method, man_id having count(*)=4
```

```
select payment_method, count(*) count from my_table  
group by payment_method, man_id having sum(salary)>1500
```

Сортировка данных order by

```
select * from my_table  
group by payment_method order by date_column
```

после группировки group by в сортировке можно указывать новую колонку

```
select salary, manager_name, count(*) cnt  
from my_table group by salary_cur order by salary DESC
```

Объединение данных из разных таблиц

Объединение таблиц с использованием union, union all, intersect, except

важно, чтобы количество колонок было одинаково у обеих таблиц (названия колонок могут быть разными, но на выходе будут названия колонок из первой таблицы)

```
select * from my_table_april  
union  
select * from my_table_may;
```

union all – соединяет все строки, оставляя дубли; union – соединяет и убирает дубли

```
select * from my_table_april  
union all  
select * from my_table_may;
```

операция пересечения intersect – получаем строки, которые есть одновременно в пересекаемых таблицах, т.е. получаем строки, которые есть и там и там

```
select * from my_table_april  
intersect  
select * from my_table_may;
```

except – операция исключения; дает только те строки, которые есть только в первой таблице. Тут важен порядок таблиц

```
select * from my_table_april
```

```
except
```

```
select * from my_table_may;
```

в MySQL except и intersect не работает

Операции JOIN

Cross join – получение всех возможных пар значений (декартово произведение)

```
select * from suits cross join cards;
```

каждая строка одной таблицы соединяется с каждой строкой другой таблицы. Хороший пример: таб.1 – масти карт, таб.2 – номиналы карт. С помощью cross join получаем всю колоду

* Количество строк равно произведению сумм строк обеих таблиц

```
select table1.field1 from table1 cross join table2 cross join table3
```

```
select * from suits cross join cards where card_name='King' order by suit_name;
```

аналогичная cross join операция

```
select * from suits,cards
```

Inner – внутренние соединения, outer – внешние

inner – то же самое, что и cross join + f(x) (отбор строк по условию)

```
select * from my_table inner join my_table_categories
```

```
on my_table.manager_id = my_table_categories.manager_id
```

использование псевдонима таблицы (для удобства и сокращения)

```
Select * from my_table mt
```

inner join == join

но можно простой join превратить в cross join:

```
select * from my_table join my_table_categories on 1=1
```

можно вместо on использовать where

```
select * from my_table cross join my_table categories
```

```
where my_table.manager_id = my_table_categories.manager_id
```

```
select * from my_table_march
```

```
join my_table_persons on my_table_march.manager_id
```

```
join my_table_product on my_table_march.product_id= my_table_product.product_id
```

две равнозначных операции

```
select distinct manager_id from my_tab_september
```

```
except
```

```
select distinct manager_id from my_tab_manager
```

=====

```
select manager_id from my_tab_september
```

```
except
```

```
select manager_id from my_tab_manager
```

Left join – оставляет все строки первой таблицы (левой)

```
select e.name,age,r.age_range,gender,d.name department from employee e
```

```
inner join age_range r on e.age_range_id=r.age_range_id
```

```
inner join gender g on e.gender_id=g.gender_id
```

```
left join department d on e.department_id=d.department_id
```

```
where d.name in ('Marketing','IT') # можно сделать отрицание and not d.name in...
```

```
and age>=25
```

Full join – когда необходимо взять за основу данные обеих таблиц

для поиска изменений в двух таблицах

в MySQL вместо full join объединяем left и right

```
select * from my_table_september
```

```
left join my_table on my_table_september.man_id = my_table.man_id
```

```
union
```

```
select * from my_table_september
```

```
right join my_table on my_table_september.man_id = my_table.man_id;
```

То же, что и right join

```
select * from my_table_september
```

```
full join my_table
```

```
on my_table_september.man_id = my_table.man_id
```

```
where not my_table.man_id is null
```

* Типовая задача с использованием join – это раскрытие справочников. В таком случае удобнее основную таблицу размещать слева, а справочники – справа и далее использовать left join.

* Значения null не являются критичными ни для таблиц данных, ни для справочников.

* Еще одна типовая задача с join – поиск дубликатов, если в строке все одинаковое, кроме, например, id.

```
select * from managers_dup tb1
join managers_dup t2
on t1.first_name=t2.first_name
and t1.last_name=t2.last_name
```

добавим дополнительное условие для выведения строк, где все равно, кроме id

```
select * from managers_dup tb1
join managers_dup t2
on t1.first_name=t2.first_name
and t1.last_name=t2.last_name
where t1.manager_id != t2.manager_id
```

вернет все пары, но в двойном количестве. Дальше отсортируем так, чтобы не было дублей.

Правильной будем считать ту строку, у которой id больше

```
select * from managers_dup tb1
join managers_dup t2
on t1.first_name=t2.first_name
and t1.last_name=t2.last_name
where t1.manager_id > t2.manager_id
```

Еще пример использования join – объединение основной таблицы со справочниками.

```
select month(dt) month, product_category_name, sum(cnt) cnt, sum(cnt*price_usd) total_usd
from sales_march t1
left join product t3 on t1.product_id=t3.product_id
left join product_category t4
on t3.product_category_id=t4.product_category_id
group by month(dt), product_category_name
```

то же самое, но за 2 месяца

```
select month(dt) month, product_category_name, sum(cnt) cnt, sum(cnt*price_usd) total_usd
from sales_march t1
```

```
left join product t3 on t1.product_id=t3.product_id
left join product_category t4
on t3.product_category_id=t4.product_category_id
group by month(dt), product_category_name

union

select month(dt) month, product_category_name, sum(cnt) cnt, sum(cnt*price_usd) total_usd
from sales_april t1
left join product t3 on t1.product_id=t3.product_id
left join product_category t4
on t3.product_category_id=t4.product_category_id
group by month(dt), product_category_name
```

Подзапросы. Модификация данных и таблиц

Условия с подзапросом

```
select * from table_sales where
dt in (select dt from sales_april)
```

1. Вложенный запрос должен быть в круглых скобках

2. Результат вложенного запроса должен быть согласован с оператором сравнения in/=

```
select * from sales_april where
product_id = (select product_id from table_products where product_name='Shampoo')
```

типы данных тоже должны соответствовать

```
select first_name, last_name,
(select sum(t2.cnt) from sales_march t2 where t2.man_id=t1.man_id) cnt,
(select sum(t3.cnt) from sales_april t3 where t3.man_id=t1.man_id) cnt_april
from managers t1
```

```
select * from
(select * from sales_march
union all
select * from sales_april) a # без 'a' работать не будет
```

```
select top 5 dt, sum(cnt) * from
```

```
(select * from sales_march  
union all  
select * from sales_april) a  
group by dt  
order by sum(cnt) DESC
```

Создание и удаление таблиц, добавление данных

Создание

```
create table my_table(  
    manager_id int,  
    manager_name varchar(200),  
    manager_salary int,  
    salary_currency varchar(200),  
    price_usd float  
);
```

Удаление

```
drop table my_table  
drop table if exists my_table
```

Создание временных таблиц (действуют до конца сессии)

MSSQL

```
create table #my_table(  
    manager_id int,  
    manager_name varchar(200),  
    manager_salary int,  
    salary_currency varchar(200),  
    price_usd float  
);
```

MySQL, Postgress

```
create temporary table my_table(  
    manager_id int,  
    manager_name varchar(200),  
    manager_salary int,  
    salary_currency varchar(200),
```

```
price_usd float  
);
```

Показать скрипт создания таблицы

```
show create table my_table22  
show fields from my_table22
```

Добавление значений в таблицу (insert)

```
insert into my_table values  
(1,'Jhon','1000','RUB','65.5')
```

можно добавить сразу несколько строк

```
insert into my_table values  
(1,'Jhon',1000,'RUB',65.5),  
(2,'Patrick',1300,'RUB',65.5)
```

можно задать свой порядок данных для вставки

```
insert into my_table (manager_salary, manager_id, manager_name)  
values  
(3000,4,'Rebecca')
```

добавление данных из существующей таблицы в новую

```
insert into my_table_new_empty  
select * from my_table_old
```

```
insert into my_table_new_empty  
select * from my_table_old  
where city in ('Moscow', 'Kaluga')
```

Изменение и удаление данных

изменение

```
update my_table set price_usd=price_usd*1.5
```

```
update my_table set manager_id=0,  
salary = salary*1.6  
where manager_id=50  
update managers_tab set FIO = concat(first_name,' ',last_name)
```


удаление

```
delete from my_table where city='Moscow'
```

удаление всех строк

```
delete from my_table
```

Truncate используется для мгновенного удаления строк и приведения таблицы к начальному виду

```
truncate table my_table
```

Изменение структуры таблицы

добавление столбца/колонки

```
alter table manager_tb add new_column int
```

удаление столбца

```
alter table manager_tb drop old_column
```

переименование столбца

MSSQL

```
exec sp_rename 'my_table.old_column', 'new_column'
```

MySQL, Postgres

```
alter table my_table rename column old_column to new_column
```

Пример операции с изменением структуры таблицы

```
alter table managers add gender_2 int
```

```
update managers set gender_2=0
```

заполняем в начале все нулями - это женский пол

```
update managers set gender_2 = 1 where gender='M'
```

```
alter table managers drop column gender
```

```
alter table managers rename column gender_2 to gender
```

чтобы полностью скопировать структуру и данные одной таблицы в другую

MSSQL

```
select * into managers_new from managers_old
```

```
select * into managers_new from managers_old where id between 20 and 50
```

MySQL

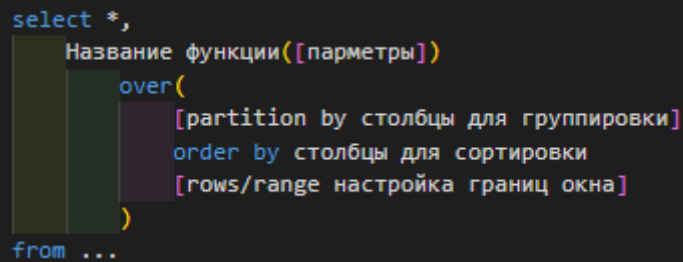
```
create table managers_new as select * from managers_old
```

сложный пример

```
select * from customer_suppliers where customers_suppliers_id in
(
  (select customer_id from deals
   intersect
   select supplier_id from deals)
 union
  select top 1 customer_id from deals order by cnt*price_usd desc
)
```

Оконные функции

Оконные функции – набор функций, имеющих особые способы применения. Основной синтаксис.



```
select *,
    Название функции([парметры])
    over(
        [partition by столбцы для группировки]
        order by столбцы для сортировки
        [rows/range настройка границ окна]
    )
from ...
```

Агрегатные функции `sum`, `count`, `avg`, `min/max`

считаем в общем по всей таблице

```
select manager, cnt,
count(cnt) over () count_result,
sum(cnt) over () sum_result,
avg(cnt) over () avg_result,
min(cnt) over () min_result,
max(cnt) over () max_result
from my_table
```

считаем для каждого менеджера

```
select manager, cnt,
count(cnt) over (partition by manager) count_result,
sum(cnt) over (partition by manager) sum_result,
avg(cnt) over (partition by manager) avg_result,
```

```
min(cnt) over (partition by manager) min_result,  
max(cnt) over (partition by manager) max_result  
from my_table  
order by manager, cnt
```

получили количества и суммы нарастающим итогом

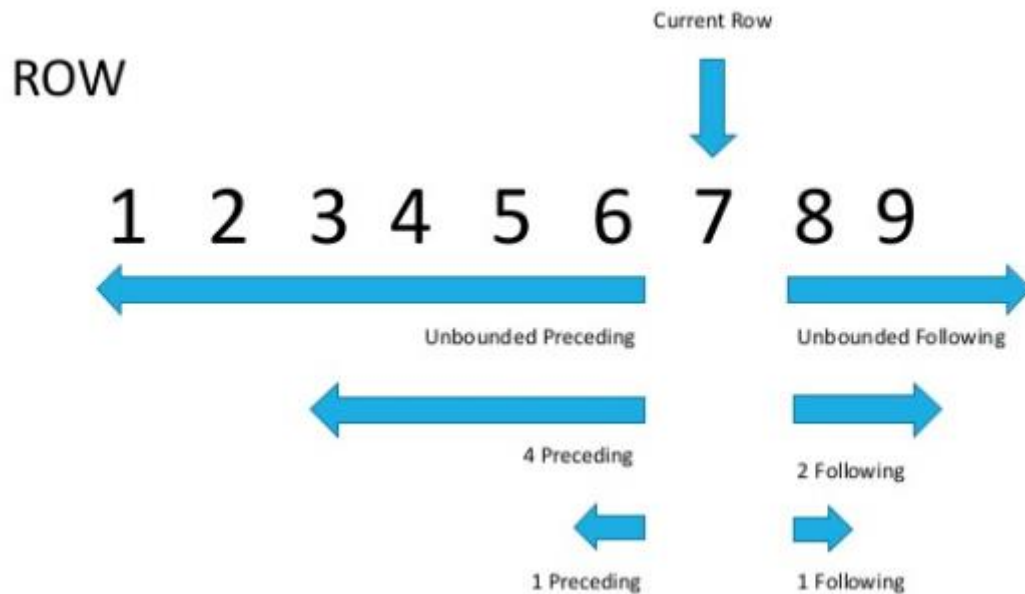
```
select manager,cnt,  
count(cnt) over (partition by manager order by order_date,order_time) count_result,  
sum(cnt) over (partition by manager order by order_date, order_time) sum_result  
from orders  
order by order_date, order_time
```

partition by – разбивка данных на отдельные окна

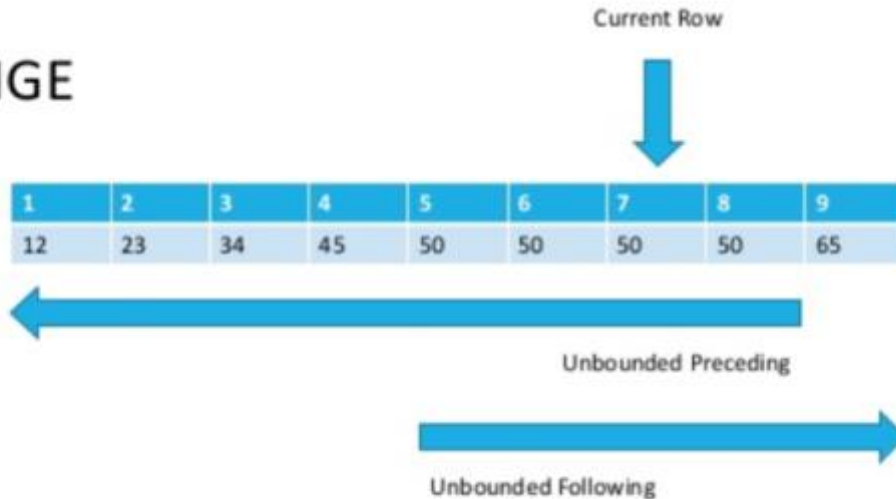
Разница между rows и between:

rows – опирается на конкретные номера строк, суммирует все значения от начала до текущей строки включительно

range – оперирует диапазоном



RANGE



1. Все, что до текущей строки/диапазона и само значение текущей строки

BETWEEN UNBOUNDED PRECEDING

BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

2. Текущая строка/диапазон и все, что после нее

BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

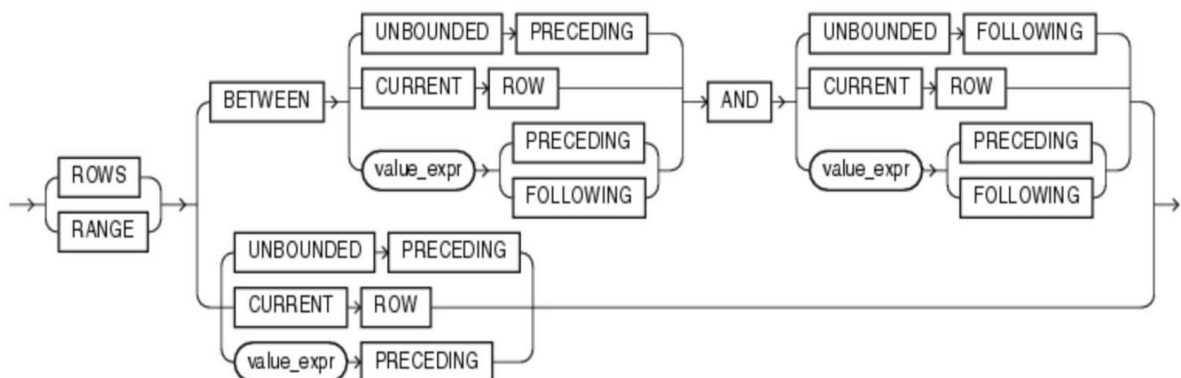
3. С конкретным указанием сколько строк до и после включать (не поддерживается для RANGE)

BETWEEN N Preceding AND N Following

BETWEEN CURRENT ROW AND N Following

BETWEEN N Preceding AND CURRENT ROW

Схема-подсказка из просторов интернета:



примеры

```
select manager,cnt,
count(cnt) over
```

```
(partition by manager
order by order_date,order_time
rows between unbounded preceding and current row) count_result,
sum(cnt) over
(partition by manager
order by order_date, order_time
rows between unbounded preceding and current row) sum_result
from orders
order by order_date, order_time

select manager,cnt,
count(cnt) over
(partition by manager
order by order_date,order_time
rows between current row and 1 following) count_result,
sum(cnt) over
(partition by manager
order by order_date, order_time
rows between current row and 1 following) sum_result
from orders
order by order_date, order_time
```

Ранжирующие функции row_number, rank, dense_rank, ntile

распределяем данные по группам разным способом. Как пример, необходимо для присвоения строками уникального цифрового идентификатора.

```
select *,
        row_number() over() num
from my_table
```

для MSSQL нужно обязательно сортировку указывать

```
select *,
        row_number() over(order by cnt) num
from my_table
```

```
select *,
    row_number() over(order by cnt) num1,
    row_number() over(order by cnt) num2 #влияет только на вычисление значений
внутри строки
from my_table
```

rank, dense_rank – дают ранги каждой строке таблицы. Разница – в rank ранги идут нарастающим итогом, в dense_rank ранг дается по-другому: 1224 (так как на 3м месте была 2ка, то 3ки не будет).

ntile – можно указать максимальное количество рангов

Функции смещения lag, lead, first_value, last_value

используются для сравнения текущего значения с одним из других:

lag – с предыдущим

lead – с последующим

first_value – с первым значением

last_value – с последним значением

```
select order_date, manager, price_usd, cnt, total, food_name
```

```
lag(food_name) over(partition by order_date
order by order_date, order_time) lag_val,
```

```
lead(food_name) over(partition by order_date
order by order_date, order_time) lead_val,
```

```
first_value(food_name) over(partition by order_date
order by order_date, order_time) first_val,
```

```
last_value(food_name) over(partition by order_date
order by order_date, order_time) last_val
```

```
from orders
```

```
order by order_date, order_time
```

Аналог if...else в SQL – case...end

```
select price_rank, count(*) from  
(select  
  case  
    when price<100 then 0  
    else 'more'  
  end price_rank  
from sales_semi) a  
group by price_rank order by price_rank
```

Конструкция WITH в SQL

Используется для предварительной корректировки исходной базы, с которой потом в последствии будет работать. CTE (Common Table Expressions) – общие табличные выражения. Также способствует красоте и понятности кода.

```
with cte_table as (  
  select t1.date1  
  , t1.date2  
  , t2.id_manager  
  from table1 t1  
  join table2 t2 on t1.date1 = t2.date_lock  
  order by t1.date1  
)  
  
select * from cte_table;
```