Лексический порядок выглядит так:

SELECT [ DISTINCT ]

FROM

WHERE

GROUP BY

HAVING

UNION

ORDER BY

Порядок выполнения в зависимости от выбора оптимизатора):

FROM

WHERE

GROUP BY

HAVING

SELECT

DISTINCT

UNION

ORDER BY

5) Exists

Оператор ANY возвращает true, если какое-либо из значений подзапроса соответствует условию. Оператор ALL возвращает true, если все значения подзапроса удовлетворяют условию.

8) Выражение CASE имеет два формата:

простое выражение CASE для определения результата сравнивает выражение с набором простых выражений;

поисковое выражение CASE для определения результата вычисляет набор логических выражений.

**Простое выражение CASE**

Простое выражение CASE сравнивает первое выражение с выражением в каждом предложении WHEN. Если эти выражения эквивалентны, то возвращается выражение в предложении THEN.

Допускается только проверка равенства.

В указанном порядке сравнивает значения выражений input\_expression и when\_expression для каждого предложения WHEN.

Возвращает выражение *result\_expression*, соответствующее первой операции *input\_expression* = *when\_expression*, равной TRUE.

Если ни одна из операций *input\_expression* = *when\_expression* не дает значения TRUE, Компонент SQL Server Database Engine возвращает выражение *else\_result\_expression*, если указано предложение ELSE, или значение NULL, если предложение ELSE не указано.

**Поисковое выражение CASE**

Вычисляет в указанном порядке выражения *Boolean\_expression* для каждого предложения WHEN.

Возвращает выражение *result\_expression*, соответствующее первому выражению *Boolean\_expression*, которое имеет значение TRUE.

Если ни одно выражение *Boolean\_expression* не равно TRUE, Компонент Database Engine возвращает выражение *else\_result\_expression*, если указано предложение ELSE, или значение NULL, если предложение ELSE не указано.

SQL Server допускает применение в выражениях CASE не более 10 уровней вложенности.

**11) Временные таблицы**

В дополнение к табличным переменным можно определять временные таблицы. Такие таблицы могут быть полезны для хранения табличных данных внутри сложного комплексного скрипта.

Временные таблицы **существуют на протяжении сессии базы данных**. Если такая таблица создается в редакторе запросов (Query Editor) в SQL Server Management Studio, то таблица будет существовать пока открыт редактор запросов. Таким образом, к временной таблице можно обращаться из разных скриптов внутри редактора запросов.

После создания все временные таблицы сохраняются в таблице **tempdb**, которая имеется по умолчанию в MS SQL Server.

Если необходимо удалить таблицу до завершения сессии базы данных, то для этой таблицы следует выполнить команду **DROP TABLE**.

Название временной таблицы начинается со знака решетки #. Если используется один **знак #,** то создается **локальная таблица**, которая доступна в течение текущей сессии. Ели используются два **знака ##,** то создается **глобальная временная таблица**. В отличие от локальной **глобальная временная таблица доступна всем открытым сессиям базы данных.**

И с этой таблицей можно работать в большей степени как и с обычной таблицей - получать данные, добавлять, изменять и удалять их. Только **после закрытия редактора запросов эта таблица перестанет существовать.**

Во временных таблицах **нельзя использовать внешние ключи**. Нет, ошибки не будет, и при попытке создать таблицу, она будет создана корректно, но попытки создать внешний ключ будут игнорироваться. Видимо, это связано с тем, что из-за внешнего ключа усложняется удаление таблиц и нужно следовать определенной последовательности.

**12) Коррелирующий подзапрос ??????????????????**

**Коррелирующий подзапрос** — это подзапрос, который содержит ссылку на столбцы из включающего его запроса (назовем его основным). Таким образом, коррелирующий подзапрос будет выполняться для каждой строки основного запроса, так как значения столбцов основного запроса будут меняться.

Поэтому можно сказать, что **коррелированный подзапрос зависит от внешнего запроса.** Это - главное различие между коррелированным и простым подзапросом. **Простой подзапрос не ссылается на внешний запрос, он может быть выполнен независимо от него.**

**Производная таблица** (derived table) - это табличное выражение, входящее в предложение FROM запроса. Производные таблицы можно применять в тех случаях, когда использование псевдонимов столбцов не представляется возможным, поскольку транслятор SQL обрабатывает другое предложение до того, как псевдоним станет известным.

13) ???????

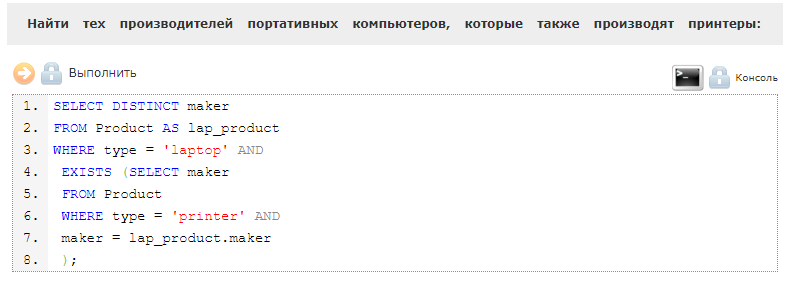
придуман **оператор IN**, позволяющий указать сразу несколько значений.

SELECT \*

FROM customers

WHERE city IN('Краснодар', 'Белгород', 'Москва');

Обычно предикат **EXISTS**используется в зависимых (коррелирующих) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе. Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому предикат **EXISTS**может иметь разные значения для разных строк основного запроса.



В подзапросе выбираются производители принтеров и сравниваются с производителем, значение которого передается из основного запроса. В основном же запросе отбираются производители портативных компьютеров. Таким образом, для каждого производителя портативных компьютеров проверяется, возвращает ли подзапрос строки (которые говорят о том, что этот производитель также выпускает принтеры).

14) консолидирующая данные с помощью предложения GROUP BY, но без предложения HAVING.

**Консолидация** — укрепление, объединение, интеграция, сплочение чего-либо.

15) Команда **HAVING** позволяет фильтровать результат группировки, сделанной с помощью команды [group by](http://old.code.mu/sql/group-by.html).

Предложение HAVING было добавлено в SQL, поскольку ключевое слово **WHERE не могло использоваться с агрегатными функциями** (SUM, COUNT, MAX, MIN и т. д)

**WHERE** выполняется до формирования групп **GROUP BY**. Это нужно для того, чтобы можно было оперировать как можно меньшим количеством данных и сэкономить ресурсы сервера и время пользователя.

Следующим этапом формируются группы, которые указаны в **GROUP BY**. После того как сформированы группы, можно накладывать условия на результаты агрегатных функций. И тут как раз наступает очередь **HAVING**: выполняются условия, которые вы задали.

Главное отличие **HAVING** от **WHERE** в том, что в **HAVING** можно наложить условия на результаты группировки, потому что порядок исполнения запроса устроен таким образом, что на этапе, когда выполняется **WHERE**, ещё нет групп, а **HAVING** выполняется уже после формирования групп.

ORDER после HAVING.

17) ??????????????????

22) простое обобщенное табличное выражение

Задается временно именованный результирующий набор, называемый обобщенным табличным выражением (ОТВ). Он получается при выполнении простого запроса и определяется в области выполнения одиночной инструкции SELECT, INSERT, UPDATE, DELETE или MERGE.

**Common Table Expression** (*CTE*) или **обобщенное табличное выражение** (*OTB*) – это временные результирующие наборы (*т.е. результаты выполнения SQL запроса*), которые не сохраняются в базе данных в виде объектов, но к ним можно обращаться.

Главной особенностью обобщенных табличных выражений является то, что с помощью них можно писать **рекурсивные запросы**.

В каких случаях нам могут пригодиться OTB, в общем, для чего они предназначены:

* Основной целью OTB является написание рекурсивных запросов, можно сказать для этого они, и были созданы;
* OTB можно использовать также и для замены представлений (VIEW), например, в тех случаях, когда нет необходимости сохранять в базе SQL запрос представления, т.е. его определение;
* Обобщенные табличные выражения повышают читаемость кода путем разделения запроса на логические блоки, и тем самым упрощают работу со сложными запросами;
* Также OTB предназначены и для многократных ссылок на результирующий набор из одной и той же SQL инструкции.

Обобщенное табличное выражение определяется с помощью **конструкции WITH**

## Какие бывают обобщенные табличные выражения?

Они бывают простые и рекурсивные.

Простые не включают ссылки на самого себя, а рекурсивные соответственно включают.

Рекурсивные ОТВ используются для возвращения иерархических данных, например, отображение сотрудников в структуре организации.

**Нерекурсивную форму OTB можно использовать в качестве альтернативы производным таблицам и представлениям.** Обычно OTB определяется посредством предложения WITH и дополнительного запроса, который ссылается на имя, используемое в предложении WITH.

24) Оконные функции

Сразу проясним, что оконные функции — это не то же самое, что GROUP BY. Они не уменьшают количество строк, а возвращают столько же значений, сколько получили на вход. Во-вторых, в отличие от GROUP BY, OVER может обращаться к другим строкам. И в-третьих, они могут считать скользящие средние и кумулятивные суммы.

оператор OVER создаёт оконную функцию

PARTITION BY позволяет сгруппировать строки по значению определённого столбца. Это полезно, если данные логически делятся на какие-то категории и нужно что-то сделать с данной строкой с учётом других строк той же группы (скажем, сравнить теннисиста с остальными теннисистами, но не с бегунами или пловцами). Этот оператор работает только с оконными функциями типа LAG, LEAD, RANK и т. д.

**Оконная функция** – это функция, которая соответственно работает с окном, т.е. набором строк, и возвращает значение на основе неких вычислений.

Как я уже отметил, оконные функции используют в аналитических отчетах, например, для вычисления каких-то статистических значений (суммы, скользящие средние, промежуточные итоги и так далее) для каждой строки результирующего набора данных.

### Упрощенный синтаксис инструкции OVER

***Оконная функция*** (столбец для вычислений) ***OVER*** (

[***PARTITION BY*** столбец для группировки]

[***ORDER BY*** столбец для сортировки]

[***ROWS*** или ***RANGE*** выражение для ограничения строк в пределах группы])

Они **не уменьшают и не увеличивают количество строк**, как GROUP BY или JOIN, а добавляют действительно новую функциональность, например:

- Посчитать сумму с накоплением (нарастающий итог)  
- Посчитать общее кол-во записей без учета limit, offset  
- Получить номер строки выборки. А если показалось легко, то усложните до — получить номер строки сотрудника, среди его отдела с обратной сортировкой по имени. И все в одной таблице

25)

**TOP** (**top**\_value) — **удалит** верхнее число строк в результирующем наборе на основе **top**\_value