### Лекция 8

1. Задача. Сгенерировать таблицу с числами от 0 до 999.
2. Задача. Из таблицы с натуральными числами (от 1 до 1000) вывести только простые числа.
3. Объединение таблиц: UNION
4. INTERSECT, EXCEPT
5. Использование подзапросов. Предикаты EXISTS, IN, ANY и т.д.

**Задача**

У Вас есть таблица с числами от 0 до 9, как сгенерировать таблицу с числами от 0 до 999?

Возможный вариант:

DECLARE @T TABLE (val int)

INSERT INTO @T

VALUES (0), (1), (2), (3), (4), (5), (6), (7), (8), (9)

SELECT T1.val\*100 + T2.val\*10 + T3.val new

FROM @T T1, @T T2, @T T3

ORDER BY new asc

Если Вы включите опцию «Client Statistics» (посмотрите в панели инструментов, недалеко от Execution Plan), то получите некую информацию по скорости работы запроса на 10 запусках:

У меня получилось так (на разных запусках по-разному, но в среднем числа похожие):

|  |  |
| --- | --- |
|  | Avg on 10 Trials, ms |
| Client processing time | 9.2 |
| Total execution time | 30.7 |
| Wait time on server replies | 21.5 |

Для сравнения, давайте попробуем использовать SQL не как SQL. И получим эту таблицу в цикле:

DECLARE @new TABLE (val int)

DECLARE @i int

SET @i = 0

SET NOCOUNT ON

WHILE @i < 1000

BEGIN

INSERT INTO @new

VALUES (@i)

SET @i = @i + 1

END

SET NOCOUNT OFF

SELECT \*

FROM @new

Данные из client statistics (у Вас могут быть другие значения, но обратите внимание на то, что в среднем время выполнения дольше)

|  |  |
| --- | --- |
|  | Avg on 10 Trials, ms |
| Client processing time | 30.8 |
| Total execution time | 64.7 |
| Wait time on server replies | 33.9 |

Обратите внимание, что этот запрос будет работать дольше.

Если написать аналогичные запросы для 100 000, то таблица по скорости:

Для декартового произведения

|  |  |
| --- | --- |
|  | Avg on 10 Trials, ms |
| Client processing time | 216.8 |
| Total execution time | 344.2 |
| Wait time on server replies | 127.4 |

То есть менее секунды

Для цикла:

|  |  |
| --- | --- |
|  | Avg on 10 Trials, ms |
| Client processing time | 2116.7 |
| Total execution time | 2147.4 |
| Wait time on server replies | 30.7 |

Уже 2 секунды и более.

**Пример.** Что выведет запрос?

SELECT \*

FROM Номенклатура

WHERE Масса NOT IN (1 , 2 , NULL)

(Задача с собеседования) Подумайте, а как из таблицы с натуральными числами (от 1 до 1000) вывести только простые числа?

Для получения остатка от деления можно использовать оператор: %

Возможный вариант решения:

DECLARE @T TABLE (val int)

DECLARE @i int

SET @i = 0

WHILE @i < 1000

BEGIN

SET @i = @i + 1

INSERT INTO @T

VALUES (@i)

END

SELECT T1.val, SUM(CASE WHEN T1.val%T2.val = 0 THEN 1 ELSE 0 END)

FROM @T T1, @T T2

WHERE T1.val > T2.val

GROUP BY T1.val

HAVING SUM(CASE WHEN T1.val%T2.val = 0 THEN 1 ELSE 0 END) = 1

ORDER BY T1.val

Вместо “,” можно использовать CROSS JOIN.

**Объединение таблиц (или запросов):**

Объединение множеств А и В – множество элементов, каждый из которых принадлежит либо одному из множеств, либо им обоим одновременно. Для таблиц справедливо следующее: множество строк, каждая из которых принадлежит либо одной из таблиц, либо им обеим.

Отсюда получаем требования на объединяемые таблицы:

* одинаковое количество соединяемых атрибутов;
* одинаковые типы данных в соединяемых столбцах (i - й столбец первой таблицы такого же типа, что и i - й столбец второй таблицы).

**Синтаксис:**

SELECT поле1, поле2…, полеN

FROM А

WHERE …

UNION

SELECT поле1, поле2…, полеN

FROM B

WHERE …

Особенностью UNION является то, что при использовании UNION нет дублирующихся строк в выводе. То есть на выходе UNION дает уникальные строки, несмотря на то, что строки могли дублироваться в любом из запросов, либо одна строка выводится и из первого, и из второго.

Если все-таки на выходе хочется иметь дублирующиеся строки, то следует использовать UNION ALL.

Если в данных, которые Вам нужно объединить нет дубликатов, то есть результат UNION и UNION ALL ожидается одинаковым, то лучше использовать UNION ALL, так как в случае использования UNION ядро БД может потратить время на попытку удаления дубликатов.

**UNION:**

1. Объединение таблиц из Архива с таблицами из неархива.
2. Пример подсчета оборота по месяцам с учетом аннулирования документов.

Пример:

В Вашей схеме документооборота можно аннулировать документы, в случае, если покупатель отказался от покупки.

Например, 20.10 Покупатель Иванов купил товар за 5000 руб., 21.10.20 Иванов вернулся с претензиями, документ от 20.10 был аннулирован (товар и деньги возвращены) и введен новый (например, с новым товаром) на сумму 2000 руб.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ндок | Дата | Покупатель | Сумма | Аннулирован | аДата |
| 1 | 20.10.20 | Иванов | 5000 | True | 21.10.20 |
| 2 | 21.10.20 | Иванов | 2000 | False |  |

Таким образом, выручка Вашего магазина по дням (денег в кассе на конец дня) должна быть равна:

|  |  |
| --- | --- |
| Дата | Оборот, руб |
| 20.10.20 | 5000 |
| 21.10.20 | -3000 |
|  |  |

Требуется написать запрос, который по дням будет правильно вычислять обороты (чтобы при запуске запроса на следующий день, обороты за предыдущие дни не менялись).

**Пример составления сводных отчетов:**

Постановка: Есть две таблицы, в каждой из которых по покупателям известно, что они купили и что вернули через брак. При этом если покупатель ничего не покупал. То упоминания о нем в таблице Продажи не будет, и если покупатель ничего не сдавал в брак, то упоминания в таблице Брак так же не будет. Нужно написать запрос, который по каждому покупателю покажет его обороты: закупочный и оборот брака.

SELECT Покупатель, sum(Отпущено), sum(Брак)

FROM (

SELECT Покупатель, sum(Колво) as Отпущено, 0 as Брак

FROM Продажи

GROUP BY Покупатель

UNION

SELECT Покупатель, 0, sum(Колво)

FROM Брак

GROUP BY Покупатель

) T

GROUP BY Покупатель

Запрос можно написать без использования UNION, учитывая то, что у нас есть также отдельная таблица с покупателями.

Переписать запрос (из union (пример составления агрегированных отчетов)), без использования UNION, учитывая то, что имеется таблица со всеми покупателями.

SELECT Покупатели.Покупатель, isNULL(Отпущено, 0) Продажи, isNULL(Брак, 0) Брак

FROM Покупатели

LEFT JOIN

(

SELECT Покупатель, sum(Колво) as Отпущено

FROM Продажи

GROUP BY Покупатель

) as Продажи ON Покупатели.Покупатель = Продажи.Покупатель

LEFT JOIN

(

SELECT Покупатель, sum(Колво) as Брак

FROM Брак

GROUP BY Покупатель

) as Брак ON Покупатели.Покупатель = Брак.Покупатель

**В MSSQL есть вспомогательные операторы Except и Intersect.**

Оба оператора возвращают уникальные значения, получающиеся в результате сравнения двух запросов.

EXCEPT возвращает все уникальные строки из левого запроса (верхнего), которые не найдены в правом.

INTERSECT возвращает уникальные строки, которые возвращают оба запроса.

Основные правила синтаксиса аналогичны условиям для UNION: типы полей должны быть совместимы, кол-во столбцов, возвращаемых обоими запросами, должно быть одинаково. NULL значения считаются равными (то есть возвращаются по одному разу).

Имена колонок используемых в ORDER BY должны ссылаться на названия колонок, возвращаемых левой частью запроса.

В случае использования подряд нескольких INTERSECT и EXCEPT нужно помнить, что INTERSECT приоритетнее, чем EXCEPT. В остальном они выполняются в порядке следования.

**Примеры:**

**INTERSECT**

1. Повторные значения:

DECLARE @A TABLE (id1 int, id2 int)

INSERT INTO @A

VALUES (1, 2), (1, 2)

DECLARE @B TABLE (id1 int, id2 int)

INSERT INTO @B

VALUES (1, 4), (1, 3), (2, 5)

SELECT id1

FROM @A

INTERSECT

SELECT id1

FROM @B

Ответ: 1 единожды

1. NULL значения

DECLARE @A TABLE (id1 int, id2 int)

INSERT INTO @A

VALUES (1, 2), (1, 3), (NULL, NULL)

DECLARE @B TABLE (id1 int, id2 int)

INSERT INTO @B

VALUES (1, 4), (1, 3), (NULL, 1)

SELECT id1

FROM @A

INTERSECT

SELECT id1

FROM @B

Ответ: 1, NULL

**EXCEPT**

1. Повторные значения

DECLARE @A TABLE (id1 int, id2 int)

INSERT INTO @A

VALUES (1, 2), (1, 2)

DECLARE @B TABLE (id1 int, id2 int)

INSERT INTO @B

VALUES (1, 4), (1, 3), (2, 5)

SELECT id2

FROM @A

EXCEPT

SELECT id2

FROM @B

Ответ: 2 (1 раз)

1. NULL-значения

DECLARE @A TABLE (id1 int, id2 int)

INSERT INTO @A

VALUES (1, 2), (1, 3), (NULL, NULL)

DECLARE @B TABLE (id1 int, id2 int)

INSERT INTO @B

VALUES (1, 4), (1, 3), (NULL, 1)

SELECT id2

FROM @A

EXCEPT

SELECT id2

FROM @B

Ответ: NULL, 2

**Порядок выполнения:**

1. EXCEPT по порядку:

DECLARE @A TABLE (id1 int)

INSERT INTO @A

VALUES (3)

DECLARE @B TABLE (id1 int)

INSERT INTO @B

VALUES (3), (2)

DECLARE @C TABLE (id1 int)

INSERT INTO @C

VALUES (3), (1)

SELECT id1

FROM @A

EXCEPT

SELECT id1

FROM @B

EXCEPT

SELECT id1

FROM @C

Ответ: 0 строк. Сначала из А вычитается B.

Если было бы наоборот, то вывелось бы 3.

1. Сначала INTERSECT, после EXCEPT

DECLARE @A TABLE (id1 int)

INSERT INTO @A

VALUES (3)

DECLARE @B TABLE (id1 int)

INSERT INTO @B

VALUES (3), (2)

DECLARE @C TABLE (id1 int)

INSERT INTO @C

VALUES (3), (1)

SELECT id1

FROM @B

EXCEPT

SELECT id1

FROM @A

INTERSECT

SELECT id1

FROM @C

Ответ: 2. Иначе было бы пусто.

Если есть еще UNION, то UNION и EXCEPT выполняются в порядке появления в запросе слева направо. При этом первый приоритет у INTERSECT.

**Вопрос**: как получить разницу двух таблиц при помощи EXCEPT?

DECLARE @A TABLE (id1 int, id2 int, id3 int)

INSERT INTO @A

VALUES

(1, 2, 3), (4, 5, 6), (1, 2, 3), (NULL, 1, 2), (NULL, NULL, NULL)

DECLARE @B TABLE (id1 int, id2 int, id3 int)

INSERT INTO @B

VALUES (4, 5, 6), (7, 8, 9), (NULL, 1, 2)

Перед тем как писать запрос, подумайте, а что, по Вашему мнению, является разницей двух таблиц?

Давайте вспомним, как мы получали разницу двух таблиц при помощи LEFT JOIN?

SELECT A.\*

FROM @A A LEFT JOIN @B B ON

A.id1 = B.id1 AND

A.id2 = B.id2 AND

A.id3 = B.id3

WHERE B.id1 is NULL

|  |  |  |
| --- | --- | --- |
| id1 | id2 | id3 |
| 1 | 2 | 3 |
| 1 | 2 | 3 |
| NULL | 1 | 2 |
| NULL | NULL | NULL |

Соответствует ли это Вашим ожиданиям?

Посмотрите на выдачу у EXCEPT:

SELECT \*

FROM @A

EXCEPT

SELECT \*

FROM @B

|  |  |  |
| --- | --- | --- |
| id1 | id2 | id3 |
| NULL | NULL | NULL |
| 1 | 2 | 3 |

Подумайте, что изменится, если Вы добавите в таблицу B строку с NULL значениями во всех полях?

PS: обратите внимание, что, если у Вас есть NULL значения, то писать запрос на разницу таблиц при помощи LEFT JOIN нужно аккуратно. В результате нужно всегда перед написанием запроса думать, а что Вам нужно, нужны ли дублирующие строки, нужно ли выводить NULL значения и т.д.

**Задача:** вывести все товары, которые покупатель с идентификатором = 1 не покупал после 01.11.20 (включая и 01.11.20)

Пример данных:

DECLARE @Товары TABLE (good\_id int primary key)

INSERT INTO @Товары

VALUES (1), (2), (3)

DECLARE @Продажи TABLE (doc\_id int, Data datetime, IDC int, good\_id int, Qty int, primary key (doc\_id, good\_id))

INSERT INTO @Продажи VALUES

(1, '20200901', 1, 2, 1),

(2, '20200901', 1, 3, 1),

(3, '20201101', 1, 3, 1),

(4, '20201101', 2, 1, 1)

Видим, что 1-ого покупателя подходящие товары – это 1 и 2. 1 – он не покупал вообще (взаимоотношения 2-ого покупателя и 1-ого товара нас не интересуют), 2 – покупал до 01.09.20.

Варианты запросов:

LEFT JOIN с условиями

SELECT Товары.good\_id

FROM @Товары Товары LEFT JOIN @Продажи Продажи ON

Товары.good\_id = Продажи.good\_id AND

Продажи.IDC = 1 AND

Продажи.Data >= '20201101'

WHERE Продажи.doc\_id is NULL

Подумайте, что случится, если условие на идентификатор покупателя и дату продажи перенести в WHERE?

EXCEPT:

SELECT good\_id

FROM @Товары

EXCEPT

SELECT good\_id

FROM @Продажи

WHERE IDC = 1 AND Data >= '20201101'

LEFT JOIN и подзапрос:

SELECT Товары.good\_id

FROM @Товары Товары LEFT JOIN

(

SELECT good\_id

FROM @Продажи

WHERE IDC = 1 AND Data >= '20201101'

) ТоварыПокупателя ON

Товары.good\_id = ТоварыПокупателя.good\_id

WHERE ТоварыПокупателя.good\_id is NULL

### Использование подзапросов:

SQL позволяет использовать результаты одного запроса в другом, то есть вкладывать запросы друг в друга. Таким образом, используя подзапросы, можно конструировать более сложные запросы.

Подчиненные запросы могут использоваться для:

* проверки наличия результатов подчиненного запроса (используются зарезервированные слова EXISTS или NOT EXISTS);
* поиска значений в основном запросе, которые равны, превышают или меньше значений, возвращаемых подчиненным запросом (используются зарезервированные слова ANY, IN или ALL);
* создания подчиненных запросов внутри подчиненных запросов (вложенные запросы).

Подзапросы могут быть независимыми (self-contained) и зависимыми (correlated), могут возвращать скалярное значение (single valued), много значений (multi valued) или таблицу (table-valued).

Разберем сначала использование подзапросов в WHERE, хотя подзапросы можно использовать практически во всех предложениях: SELECT (single-valued), FROM, HAVING (single, multi).

**Использование в WHERE:**

Есть таблица Номенклатура с полями: **Товар\_ID**, Наименование, Масса. Ключевое поле – Товар\_ID.

Вы хотите получить все товары, масса которых больше, чем масса товара, задаваемого параметром. Возможный пример такого запроса:

SELECT Товар\_ID, Наименование, Масса

FROM Номенклатура

WHERE Масса > (SELECT Масса FROM Номенклатура WHERE Товар\_ID = @Товар)

Это пример независимого подзапроса (self-contained), возвращающего одно значение (single valued). Независимого – так как подзапрос не ссылается на внешний запрос и может быть выполнен отдельно от внешнего запроса (поэтому независимые запросы легко исследовать на правильность работы (debug)). Одно значение – так как в таблице номенклатура у нас ровно одно значение массы для каждого товара.

В целом, подзапросы, которые возвращают одно значение можно помещать в любую часть запроса, где возможно использование скаляров: SELECT, FROM, WHERE, HAVING.

Следующий запрос выполнить не удастся в том случае, когда параметром будет задано значение массы меньшее, чем значение массы для нескольких артикулов.

SELECT Товар\_ID, Наименование, Масса

FROM Номенклатура

WHERE Товар\_ID = (SELECT Товар\_ID FROM Номенклатура WHERE Масса >@Масса)

В результате выполнения подзапроса выведется более одного артикула, а реляционные операторы задают сравнения строго между двумя значениями.

То есть при использовании реляционных операторов подзапрос должен выбирать одно поле, а тип данных этого поля должен совпадать с тем значением, с которым он будет сравниваться в предикате.

Пример подзапроса выше – это независимый многозначный запрос (self-contained multi valued подзапрос).

Как можно обеспечить единственное значение в подзапросе? Для примера выше можно заменить «=» на in.

Следующие запросы будут написаны к такой схеме данных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Документы** |  | **Документы\_данные** |  | **Остатки** |
| **ндок** |  | **ндок** |  | **Дата** |
| Дата |  | **Товар\_ID** |  | **Товар\_ID** |
| Покупатель |  | Колво |  | Остаток |
| Сумма |  | Цена |  |  |

1. ***Использование distinct***

Допустим, цена на любой товар устанавливается в начале дня и в течение дня не меняется. Пусть нам необходимо найти товары, цена продажи на которые была меньше, чем цена на товар, заданный параметром, в заданный также параметром день.

SELECT DISTINCT Документы\_данные.Товар\_ID

FROM Документы INNER JOIN Документы\_данные ON

Документы.ндок = Документы\_данные.ндок

WHERE CAST([Документы].[Дата] as date) = @День

AND Документы\_данные.Цена <

(

SELECT DISTINCT Цена

FROM Документы INNER JOIN Документы\_данные ON

Документы.ндок = Документы\_данные.ндок

WHERE Товар\_ID = [Товар] AND

CAST(Документы.Дата as date) = @День

);

1. ***Использование агрегатных функций***

Агрегатные функции автоматически производят единственное значение для любого числа строк.

Пример:

1. Получение документа с максимальным оборотом.

SELECT Покупатель, Сумма, ндок

FROM Документы as Продажи

WHERE Сумма =

(

SELECT MAX(Сумма)

FROM Документы

)

1. Получение документа с максимальным оборотом по каждому покупателю:

Подумайте, почему следующие запросы не подойдут?

|  |  |
| --- | --- |
| SELECT Покупатель, Сумма, ндок  FROM Документы as Продажи  WHERE Сумма =  (  SELECT MAX(Сумма)  FROM Документы  GROUP BY Покупатель  ) | SELECT Покупатель, Сумма, ндок  FROM Документы as Продажи  WHERE Сумма IN  (  SELECT MAX(Сумма)  FROM Документы  GROUP BY Покупатель  ) |

В первом ошибка синтаксиса: так как «равно» предполагает наличие одного значения. Второй запрос синтаксически правильный, но в подзапросе получаем максимальное значение для каждого покупателя, значит, во внешнем запросе выведем всех покупателей, которые купили на такую сумму, но это не значит, что это обязательно их максимальная сумма покупки.

Пример: покупатель 1 купить на 100 и 200 у.е, покупатель 2 купил на 100 у.е. В результате второго запроса Вы выведете все 3 покупки, а правильно вывести только две.

Правильный запрос:

SELECT Покупатель, Сумма, ндок

FROM Документы as Продажи

WHERE Сумма =

(

SELECT MAX(Сумма)

FROM Документы

WHERE Документы.Покупатель = Продажи.Покупатель

)

Заметим, что если у одного контрагента имеется несколько документов на одну максимальную сумму, то в результат запроса попадут все эти документы. Кроме того, заметим, что в подзапросе можно использовать поле из внешнего запроса. Это пример зависимого (связанного,correlated) подзапроса.

Как понять, какой результат выведет SQL Server при наличии зависимого (связанного) подзапроса? Логическая последовательность обработки следующая:

* Выбирается строка из таблицы, имя которой указано во внешнем запросе.
* Выполняется подзапрос и полученное значение применяется для анализа этой строки в условии предложения WHERE внешнего запроса.
* По результату оценки этого условия принимается решение о включении или не включении строки в состав выходных данных.
* Процедура повторяется для следующей строки таблицы внешнего запроса.

Еще возможный вариант:

SELECT Продажи.Покупатель, Продажи.Сумма, ндок

FROM Документы as Продажи INNER JOIN

(

SELECT Max(Сумма) as MaxSumm, Покупатель as Пок

FROM Документы

GROUP BY Покупатель

) as MaxT ON

Продажи.Покупатель = MaxT.Пок AND

Продажи.Сумма = MaxT.MaxSumm

***Использование подчиненных запросов в предложении HAVING.***

Пример:

Подсчет количества товаров с группировкой по массе, причем таких, что масса больше средней массы по всей номенклатуре.

SELECT Масса, COUNT (Товар\_ID)

FROM Номенклатура

GROUP BY Масса

HAVING Масса > (SELECT Avg(Масса) FROM Номенклатура)

1. ***Использование предикатов***

**Предикат EXISTS().**

Назначение предиката Exists состоит в проверке того, что множество не является пустым.

Предикат генерирует значения TRUE или FALSE в зависимости от выходных данных подзапроса. Используя подзапросы в качестве аргумента, этот оператор оценивает результат выполнения подзапроса как истинный, если этот подзапрос генерирует выходные данные, то есть в случае *существования (возврата)* хотя бы одного найденного значения. В противном случае результат подзапроса ложный.

Пример: поиск товара, который не отпускался в течение месяца.

SELECT Товар

FROM Номенклатура

WHERE not exists

(

SELECT \*

FROM Документы INNER JOIN Документы\_данные ON

Документы.ндок = Документы\_данные.ндок

WHERE Документы\_данные.Товар = Номенклатура.Товар

AND Документы.Дата > ‘20201101’

)

В принципе этот же запрос можно написать другими способами (поэкспериментируйте с большими объемами данных, какая версия сработает лучше?):

1. Запрос с использованием предиката IN.

SELECT Товар

FROM Номенклатура

WHERE Товар not in

(

SELECT Товар

FROM Документы INNER JOIN Документы\_данные ON

Документы.ндок = Документы\_данные.ндок

WHERE Документы.Дата > ‘20201101’

)

1. Запрос с использованием LEFT JOIN

SELECT Номенклатура.Товар

FROM Номенклатура LEFT JOIN

(SELECT distinct Товар

FROM Документы INNER JOIN Документы\_данные ON

Документы.ндок = Документы\_данные.ндок

WHERE Документы.Дата > ‘20201101’

) as Продажи ON Продажи.Товар = Номенклатура.Товар

WHERE Продажи.Товар is NULL

**Пример с EXISTS:**

У Вас есть таблица с натуральными числами T. Некоторые числа пропущены, Вам нужно найти минимальное пропущенное число.

CREATE TABLE #T (num INT NOT NULL PRIMARY KEY CHECK(num > 0));

INSERT INTO #T VALUES (3), (4), (6), (7)

-- Вариант с EXISTS

SELECT CASE WHEN EXISTS(SELECT \* FROM #T WHERE num = 1) THEN missing ELSE 1 END

FROM

(

SELECT MIN(A.num) + 1 as missing

FROM #T AS A

WHERE NOT EXISTS

(

SELECT \* FROM #T AS B

WHERE B.num = A.num + 1

)

) T

-- Вариант с LEFT JOIN

SELECT CASE WHEN (SELECT MIN(num) FROM #T) = 1 THEN missing ELSE 1 END

FROM

(

SELECT MIN(A.num + 1) missing

FROM #T A LEFT JOIN #T B ON

A.num + 1 = B.num

WHERE B.num is NULL

) T

-- Вариант решения проблемы с недостающей единицей

DECLARE @T TABLE (num int)

INSERT INTO @T

SELECT num

FROM #T

INSERT INTO @T

VALUES (0)

SELECT MIN(A.num + 1) missing

FROM @T A LEFT JOIN @T B ON

A.num + 1 = B.num

WHERE B.num is NULL

DROP TABLE #T

**Операторы сравнения с множеством значений IN, ANY, ALL**

|  |  |
| --- | --- |
| IN | Равно любому из значений, полученных во внутреннем запросе |
| NOT IN | Не равно ни одному из значений, полученных во внутреннем запросе |
| = ANY | То же, что и IN. Соответствуют логическому оператору OR |
| >ANY, >=ANY | Больше, чем (либо больше или равно) любое (какого либо) полученное число. Эквивалентно > или >= для самого меньшего полученного числа. |
| <ANY, <=ANY | Меньше, чем (либо меньше или равно) любое (какого либо) полученное число. Эквивалентно < или <= для самого большого полученного числа. |
| =ALL | Равно всем полученным значениям. Эквивалентно логическому оператору AND. |
| >ALL, >=ALL | Больше, чем (либо больше или равно) все полученные числа. Эквивалент > или > = для самого большего полученного числа. |
| <ALL, <=ALL | Меньше, чем (либо меньше или равно) все полученные числа. Эквивалент < или < = для самого меньшего полученного числа. |

Примеры запросов:

DECLARE @T1 TABLE (num int)

INSERT INTO @T1 VALUES (2), (4), (6)

DECLARE @T2 TABLE (num int)

INSERT INTO @T2 VALUES (1), (3), (5)

SELECT \*

FROM @T1

WHERE num > any (SELECT num FROM @T2)

SELECT \*

FROM @T1

WHERE num > all (SELECT num FROM @T2)

**Особенности применения операторов ANY, ALL, Exists при обработке пустых значений (NULL).**

Когда правильный подзапрос не генерирует никаких выходных данных, оператор ALL автоматически принимает значение TRUE, а оператор ANY - FALSE.

**Дополнение**: подумайте, что выведет следующий запрос:

SELECT \*

FROM Товары

WHERE Масса NOT IN (1, 3, NULL)

Как получить сотрудников, которые не являются начальниками?

DECLARE @Empl TABLE (empl\_id int)

INSERT INTO @Empl VALUES (1), (2), (3)

DECLARE @Deps TABLE (empl\_id int, boss\_id int)

INSERT INTO @Deps VALUES

(1, 2), (2, 3), (3, NULL)

Вариант, который требует «допиливания»:

SELECT \*

FROM @Empl E

WHERE empl\_id not in

(SELECT boss\_ID FROM @deps)

Как его нужно исправить, чтобы этот вариант работал верно?

SELECT \*

FROM @Empl E

WHERE NOT Exists (SELECT \* FROM @deps WHERE boss\_id = E.empl\_id)

***Использование single-valued запросов в SELECT.***

Задача: для каждого факта продажи товара проставить остаток товара на этот момент по таблице Остатки:

Возможный вариант:

SELECT Док.ндок, Док.Дата, Док.Покупатель, Док.Сумма, Днн.Товар\_ID, Днн.Колво, Днн.Цена,

(SELECT TOP 1 Остаток FROM Остатки WHERE Остатки.Товар\_ID = Днн.Товар\_ID AND Остатки.Дата <= Док.Дата ORDER BY Остатки.Дата desc) Остаток

FROM Документы Док INNER JOIN Документы\_данные Днн ON

Док.ндок = Днн.ндок

***Комментарий:***

Осторожно обращайтесь с названиями полей. Лучше всего в случае зависимого (связанного) запроса обращаться к названиям полей с указанием таблицы.

**Пример: Что выдаст запрос?**

DECLARE @Покупатели TABLE (Пок\_ID int primary key, ФИО nvarchar(max))

INSERT INTO @Покупатели VALUES (1, 'test1'), (2, 'test2')

DECLARE @Документы TABLE (ндок int primary key, ПокID int)

INSERT INTO @Документы VALUES (1, 1)

SELECT \*

FROM @Покупатели

WHERE Пок\_ID in (SELECT distinct Пок\_ID FROM @Документы)

Дополнительные задачи:

1. Как для каждого акта продажи товара получить, сколько этого товара было продано за месяц до?

SELECT Док.ндок, Док.Дата, Днн.Товар\_ID, Днн.Колво,

(

SELECT SUM(ДннВн.Колво)

FROM Документы ДокВн INNER JOIN Документы\_данные ДннВн ON

ДокВн.ндок = ДннВн.ндок

WHERE ДокВн.Дата >= dateadd(m, -1, Док.Дата) AND ДокВн.Дата < Док.Дата AND ДннВн.Товар\_ID = Днн.Товар\_ID

) ПродажиПредМесяц

FROM Документы Док INNER JOIN Документы\_данные Днн ON

Док.ндок = Днн.ндок

1. Как вывести рейтинг каждого покупателя, используя подзапрос в SELECT?

DECLARE @Покупатели TABLE (id int, s float)

INSERT INTO @Покупатели

VALUES (1, 100), (2, 50), (3, 50), (4, 10)

SELECT id, s, (SELECT COUNT(\*) FROM @Покупатели П2 WHERE П1.s <= П2.s) rk

FROM @Покупатели П1

**Литература:**

1. И.Ф.Астахова, А.П.Толстобров, В.М.Мельников «SQL в примерах и задачах»
2. Т.М.Дадашев, В.Ю.Рубаев, О.Л.Белоусов, Д.Р.Гончар «Введение в реляционные базы данных и язык SQL»
3. Help в Access
4. MSDN