


 **enhorse** / **java-interview** Public[Code](#) [Issues 9](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) master ▾

...

java-interview / **sql.md**

KiselevichAndrey Update sql.md ...

 History 4 contributors 561 lines (392 sloc) | 40.7 KB

...

Вопросы для собеседования

SQL

- Что такое «SQL»?
- Какие существуют операторы SQL?
- Что означает `NULL` в SQL?
- Что такое «временная таблица»? Для чего она используется?
- Что такое «представление» (*view*) и для чего оно применяется?
- Каков общий синтаксис оператора `SELECT` ?
- Что такое `JOIN` ?
- Какие существуют типы `JOIN` ?
- Что лучше использовать `JOIN` или подзапросы?
- Для чего используется оператор `HAVING` ?
- В чем различие между операторами `HAVING` и `WHERE` ?
- Для чего используется оператор `ORDER BY` ?
- Для чего используется оператор `GROUP BY` ?
- Как `GROUP BY` обрабатывает значение `NULL` ?

- В чем разница между операторами `GROUP BY` и `DISTINCT` ?
- Перечислите основные агрегатные функции.
- В чем разница между `COUNT(*)` и `COUNT({column})` ?
- Что делает оператор `EXISTS` ?
- Для чего используются операторы `IN` , `BETWEEN` , `LIKE` ?
- Для чего применяется ключевое слово `UNION` ?
- Какие ограничения на целостность данных существуют в SQL?
- Какие отличия между ограничениями `PRIMARY` и `UNIQUE` ?
- Может ли значение в столбце, на который наложено ограничение `FOREIGN KEY` , равняться `NULL` ?
- Как создать индекс?
- Что делает оператор `MERGE` ?
- В чем отличие между операторами `DELETE` и `TRUNCATE` ?
- Что такое «*хранимая процедура*»?
- Что такое «*триггер*»?
- Что такое «*курсор*»?
- Опишите разницу типов данных `DATETIME` и `TIMESTAMP` .
- Для каких числовых типов недопустимо использовать операции сложения/вычитания?
- Какое назначение у операторов `PIVOT` и `UNPIVOT` в Transact-SQL?
- Расскажите об основных функциях ранжирования в Transact-SQL.
- Для чего используются операторы `INTERSECT` , `EXCEPT` в Transact-SQL?
- Напишите запрос...

Что такое «SQL»?

SQL, Structured query language («язык структурированных запросов») — формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД).

[к оглавлению](#)

Какие существуют операторы SQL?

операторы определения данных (Data Definition Language, DDL):

- CREATE создает объект БД (базу, таблицу, представление, пользователя и т. д.),
- ALTER изменяет объект,
- DROP удаляет объект;

операторы манипуляции данными (Data Manipulation Language, DML):

- SELECT выбирает данные, удовлетворяющие заданным условиям,
- INSERT добавляет новые данные,
- UPDATE изменяет существующие данные,
- DELETE удаляет данные;

операторы определения доступа к данным (Data Control Language, DCL):

- GRANT предоставляет пользователю (группе) разрешения на определенные операции с объектом,
- REVOKE отзывает ранее выданные разрешения,
- DENY задает запрет, имеющий приоритет над разрешением;

операторы управления транзакциями (Transaction Control Language, TCL):

- COMMIT применяет транзакцию,
- ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции,
- SAVEPOINT разбивает транзакцию на более мелкие.

[к оглавлению](#)

Что означает NULL в SQL?

NULL - специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных. NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения».

NULL означает отсутствие, неизвестность информации. Значение **NULL** не является значением в полном смысле слова: по определению оно означает отсутствие значения и не принадлежит ни одному типу данных. Поэтому **NULL** не равно ни логическому значению **FALSE**, ни *пустой строке*, ни **0**. При сравнении **NULL** с любым значением будет получен результат **NULL**, а не **FALSE** и не **0**. Более того, **NULL** не равно **NULL**!

[к оглавлению](#)

Что такое «временная таблица»? Для чего она используется?

Временная таблица — это объект базы данных, который хранится и управляется системой базы данных на временной основе. Они могут быть локальными или глобальными. Используется для сохранения результатов вызова хранимой процедуры, уменьшение числа строк при соединениях, агрегирование данных из различных источников или как замена курсоров и параметризованных представлений.

[к оглавлению](#)

Что такое «представление» (view) и для чего оно применяется?

Представление, View — виртуальная таблица, представляющая данные одной или более таблиц альтернативным образом.

В действительности **представление** — всего лишь результат выполнения оператора **SELECT**, который хранится в структуре памяти, напоминающей SQL таблицу. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления значительно расширяют возможности управления данными. Это способ дать публичный доступ к некоторой (но не всей) информации в таблице.

[к оглавлению](#)

Каков общий синтаксис оператора **SELECT**?

SELECT - оператор DML SQL, возвращающий набор данных (выборку) из базы данных, удовлетворяющих заданному условию. Имеет следующую структуру:

```
SELECT
    [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | column | formula}]
    [HAVING where_definition]
    [ORDER BY {unsigned_integer | column | formula} [ASC | DESC], ...]
```

[к оглавлению](#)

Что такое JOIN ?

JOIN - оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Предназначен для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор.

Особенностями операции соединения являются следующее:

- в схему таблицы-результата входят столбцы обеих исходных таблиц (таблиц-операндов), то есть схема результата является «сцеплением» схем операндов;
- каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда;
- при необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно).

```
SELECT
    field_name [,... n]
FROM
    Table1
    {INNER | {LEFT | RIGHT | FULL} OUTER | CROSS } JOIN
    Table2
    {ON <condition> | USING (field_name [,... n])}
```

[к оглавлению](#)

Какие существуют типы JOIN ?

(INNER) JOIN Результатом объединения таблиц являются **записи, общие для левой и правой таблиц**. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

LEFT (OUTER) JOIN Производит **выбор всех записей первой таблицы и соответствующих им записей второй таблицы**. Если записи во второй таблице не найдены, то вместо них подставляется пустой результат (`NULL`). Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

RIGHT (OUTER) JOIN **LEFT JOIN** с операндами, расставленными в обратном порядке. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

FULL (OUTER) JOIN Результатом объединения таблиц являются **все записи, которые присутствуют в таблицах**. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

CROSS JOIN (декартово произведение) При выборе **каждая строка одной таблицы объединяется с каждой строкой второй таблицы**, давая тем самым **все возможные сочетания строк двух таблиц**. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

[к оглавлению](#)

Что лучше использовать JOIN или подзапросы?

Обычно лучше использовать **JOIN**, поскольку в большинстве случаев он более понятен и лучше оптимизируется СУБД (но 100% этого гарантировать нельзя). Так же JOIN имеет заметное преимущество над подзапросами в случае, когда список выбора `SELECT` содержит столбцы более чем из одной таблицы.

Подзапросы лучше использовать в случаях, когда нужно вычислять **агрегатные значения** и использовать их для сравнений во внешних запросах.

[к оглавлению](#)

Для чего используется оператор HAVING ?

HAVING используется для фильтрации результата **GROUP BY** по заданным логическим условиям.

[к оглавлению](#)

В чем различие между операторами HAVING и WHERE ?

Основное отличие 'WHERE' от 'HAVING' заключается в том, что 'WHERE' сначала выбирает строки, а затем группирует их и вычисляет агрегатные функции (таким образом, она отбирает строки для вычисления агрегатов), тогда как 'HAVING' отбирает строки групп после группировки и вычисления агрегатных функций. Как следствие, предложение 'WHERE' не должно содержать агрегатных функций; не имеет смысла использовать агрегатные функции для определения строк для вычисления агрегатных функций. Предложение 'HAVING', напротив, всегда содержит агрегатные функции. (Строго говоря, вы можете написать предложение 'HAVING', не используя агрегаты, но это редко бывает полезно. То же самое условие может работать более эффективно на стадии 'WHERE'.)

[к оглавлению](#)

Для чего используется оператор ORDER BY ?

ORDER BY упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого. Возможно определять возрастание ASC или убывание DESC для каждого столбца. По умолчанию установлено - возрастание.

[к оглавлению](#)

Для чего используется оператор GROUP BY ?

GROUP BY используется для агрегации записей результата по заданным признакам-атрибутам.

[к оглавлению](#)

Как GROUP BY обрабатывает значение NULL ?

При использовании **GROUP BY** все значения **NULL** считаются равными.

[к оглавлению](#)

В чем разница между операторами GROUP BY и DISTINCT?

DISTINCT указывает, что для вычислений используются только уникальные значения столбца. **NULL** считается как отдельное значение. **GROUP BY** создает отдельную группу для всех возможных значений (включая значение **NULL**).

Если нужно удалить только дубликаты лучше использовать **DISTINCT**, **GROUP BY** лучше использовать для определения групп записей, к которым могут применяться агрегатные функции.

[к оглавлению](#)

Перечислите основные агрегатные функции.

Агрегатных функции - функции, которые берут группы значений и сводят их к одиночному значению.

SQL предоставляет несколько агрегатных функций:

COUNT - производит подсчет записей, удовлетворяющих условию запроса; **SUM** - вычисляет арифметическую сумму всех значений колонки; **AVG** - вычисляет среднее арифметическое всех значений; **MAX** - определяет наибольшее из всех выбранных значений; **MIN** - определяет наименьшее из всех выбранных значений.

[к оглавлению](#)

В чем разница между COUNT(*) и COUNT({column})?

COUNT (*) подсчитывает количество записей в таблице, не игнорируя значение **NULL**, поскольку эта функция оперирует записями, а не столбцами.

COUNT ({column}) подсчитывает количество значений в {column}. При подсчете количества значений столбца эта форма функции **COUNT** не принимает во внимание значение **NULL**.

[к оглавлению](#)

Что делает оператор EXISTS?

EXISTS берет подзапрос, как аргумент, и оценивает его как **TRUE**, если подзапрос возвращает какие-либо записи и **FALSE**, если нет.

[к оглавлению](#)

Для чего используются операторы **IN**, **BETWEEN**, **LIKE** ?

IN - определяет набор значений.

```
SELECT * FROM Persons WHERE name IN ('Ivan', 'Petr', 'Pavel');
```

BETWEEN определяет диапазон значений. В отличие от **IN**, **BETWEEN** чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

```
SELECT * FROM Persons WHERE age BETWEEN 20 AND 25;
```

LIKE применим только к полям типа **CHAR** или **VARCHAR**, с которыми он используется чтобы находить подстроки. В качестве условия используются *символы шаблонизации (wildkards)* - специальные символы, которые могут соответствовать чему-нибудь:

- **_** замещает любой одиночный символ. Например, **'b_t'** будет соответствовать словам **'bat'** или **'bit'**, но не будет соответствовать **'brat'**.
- **%** замещает последовательность любого числа символов. Например **'%p%t'** будет соответствовать словам **'put'**, **'posit'**, или **'opt'**, но не **'spite'**.

```
SELECT * FROM UNIVERSITY WHERE NAME LIKE '%o';
```

[к оглавлению](#)

Для чего применяется ключевое слово **UNION** ?

В языке SQL ключевое слово **UNION** применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих записей. Оба запроса должны возвращать одинаковое число столбцов и совместимые типы данных в соответствующих столбцах. Необходимо отметить, что **UNION** сам по себе не гарантирует порядок записей. Записи из второго запроса могут оказаться в начале, в конце или вообще перемешаться с записями из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать **ORDER BY**.

[к оглавлению](#)

Какие ограничения на целостность данных существуют в SQL?

PRIMARY KEY - набор полей (1 или более), значения которых образуют уникальную комбинацию и используются для однозначной идентификации записи в таблице. Для таблицы может быть создано только одно такое ограничение. Данное ограничение используется для обеспечения целостности сущности, которая описана таблицей.

CHECK используется для ограничения множества значений, которые могут быть помещены в данный столбец. Это ограничение используется для обеспечения целостности предметной области, которую описывают таблицы в базе.

UNIQUE обеспечивает отсутствие дубликатов в столбце или наборе столбцов.

FOREIGN KEY защищает от действий, которые могут нарушить связи между таблицами. **FOREIGN KEY** в одной таблице указывает на **PRIMARY KEY** в другой. Поэтому данное ограничение нацелено на то, чтобы не было записей **FOREIGN KEY**, которым не отвечают записи **PRIMARY KEY**.

[к оглавлению](#)

Какие отличия между ограничениями **PRIMARY** и **UNIQUE** ?

По умолчанию ограничение **PRIMARY** создает кластерный индекс на столбце, а **UNIQUE** - некластерный. Другим отличием является то, что **PRIMARY** не разрешает **NULL** записей, в то время как **UNIQUE** разрешает одну (а в некоторых СУБД несколько) **NULL** запись.

[к оглавлению](#)

Может ли значение в столбце, на который наложено ограничение FOREIGN KEY, равняться NULL ?

Может, если на данный столбец не наложено ограничение NOT NULL .

[к оглавлению](#)

Как создать индекс?

Индекс можно создать либо с помощью выражения CREATE INDEX :

```
CREATE INDEX index_name ON table_name (column_name)
```

либо указав ограничение целостности в виде уникального UNIQUE или первичного PRIMARY ключа в операторе создания таблицы CREATE TABLE .

[к оглавлению](#)

Что делает оператор MERGE ?

MERGE позволяет осуществить слияние данных одной таблицы с данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE , а если нет - INSERT . При этом изменять поля таблицы в секции UPDATE , по которым идет связывание двух таблиц, нельзя.

[к оглавлению](#)

В чем отличие между операторами DELETE и TRUNCATE ?

DELETE - оператор DML, удаляет записи из таблицы, которые удовлетворяют критерию WHERE при этом задействуются триггеры, ограничения и т.д.

TRUNCATE - DDL оператор (удаляет таблицу и создает ее заново. Причем если на эту таблицу есть ссылки FOREIGN KEY или таблица используется в репликации, то пересоздать такую таблицу не получится).

[к оглавлению](#)

Что такое «*хранимая процедура*»?

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных. В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным) и в дальнейшем её обработка осуществляется быстрее.

[к оглавлению](#)

Что такое «*триггер*»?

Триггер (trigger) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением, удалением или изменением данных в заданной таблице реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически и все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов `BEFORE` (триггер запускается до выполнения связанного с ним события) или `AFTER` (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись. Кроме того, триггеры могут быть привязаны не к таблице, а к представлению (`VIEW`). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова `BEFORE` и `AFTER` влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

[к оглавлению](#)

Что такое «курсор»?

Курсор — это объект базы данных, который позволяет приложениям работать с записями «по-одной», а не сразу с множеством, как это делается в обычных SQL командах.

Порядок работы с курсором такой:

- Определить курсор (`DECLARE`)
- Открыть курсор (`OPEN`)
- Получить запись из курсора (`FETCH`)
- Обработать запись...
- Закрыть курсор (`CLOSE`)
- Удалить ссылку курсора (`DEALLOCATE`). Когда удаляется последняя ссылка курсора, SQL освобождает структуры данных, составляющие курсор.

[к оглавлению](#)

Опишите разницу типов данных `DATETIME` и `TIMESTAMP`.

`DATETIME` предназначен для хранения целого числа: `YYYYMMDDHHMMSS`. И это время не зависит от временной зоны, настроенной на сервере. Размер: 8 байт

`TIMESTAMP` хранит значение равное количеству секунд, прошедших с полуночи 1 января 1970 года по усреднённому времени Гринвича. При получении из базы отображается с учётом часового пояса. Размер: 4 байта

[к оглавлению](#)

Для каких числовых типов недопустимо использовать операции сложения/вычитания?

В качестве операндов операций сложения и вычитания нельзя использовать числовой тип `BIT`.

[к оглавлению](#)

Какое назначение у операторов `PIVOT` и `UNPIVOT` в Transact-SQL?

`PIVOT` и `UNPIVOT` являются нестандартными реляционными операторами, которые поддерживаются Transact-SQL.

Оператор `PIVOT` разворачивает возвращающее табличное значение выражение, преобразуя уникальные значения одного столбца выражения в несколько выходных столбцов, а также, в случае необходимости, объединяет оставшиеся повторяющиеся значения столбца и отображает их в выходных данных. Оператор `UNPIVOT` производит действия, обратные `PIVOT`, преобразуя столбцы возвращающего табличное значение выражения в значения столбца.

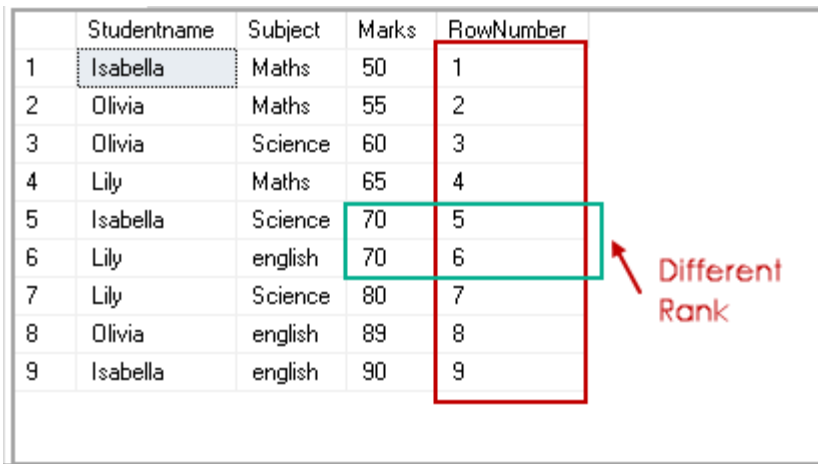
[к оглавлению](#)

Расскажите об основных функциях ранжирования в Transact-SQL.

Ранжирующие функции - это функции, которые возвращают значение для каждой записи группы в результирующем наборе данных. На практике они могут быть использованы, например, для простой нумерации списка, составления рейтинга или постраничной навигации.

К примеру, у нас имеется набор данных следующего вида:

	Studentname	Subject	Marks	RowNumber
1	Isabella	Maths	50	1
2	Olivia	Maths	55	2
3	Olivia	Science	60	3
4	Lily	Maths	65	4
5	Isabella	Science	70	5
6	Lily	english	70	6
7	Lily	Science	80	7
8	Olivia	english	89	8
9	Isabella	english	90	9



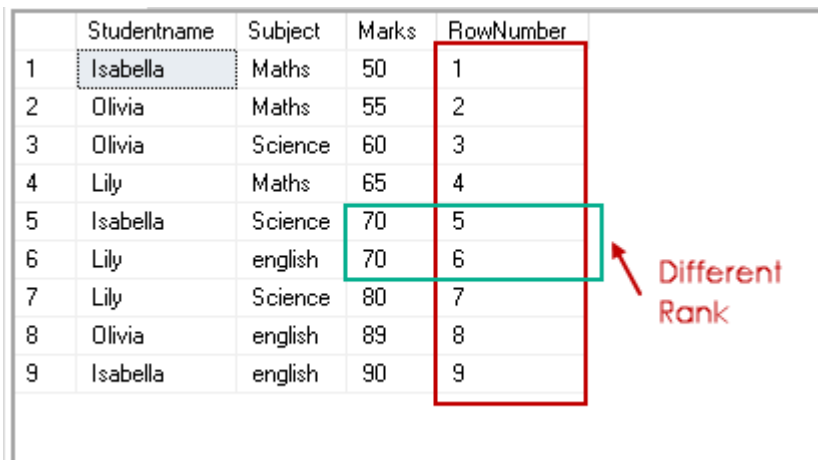
ROW_NUMBER – функция нумерации в Transact-SQL, которая возвращает просто номер записи.

Например, запрос

```
SELECT Studentname,  
       Subject,  
       Marks,  
       ROW_NUMBER() OVER(ORDER BY Marks) RowNumber  
FROM ExamResult;
```

Вернёт набор данных следующего вида:

	Studentname	Subject	Marks	RowNumber
1	Isabella	Maths	50	1
2	Olivia	Maths	55	2
3	Olivia	Science	60	3
4	Lily	Maths	65	4
5	Isabella	Science	70	5
6	Lily	english	70	6
7	Lily	Science	80	7
8	Olivia	english	89	8
9	Isabella	english	90	9



А запрос вида

```
SELECT Studentname,  
       Subject,  
       Marks,  
       ROW_NUMBER() OVER(ORDER BY Marks desc) RowNumber  
FROM ExamResult;
```

Вернёт набор

	Studentname	Subject	Marks	RowNumber
1	Isabella	english	90	1
2	Olivia	english	89	2
3	Lily	Science	80	3
4	Lily	english	70	4
5	Isabella	Science	70	5
6	Lily	Maths	65	6
7	Olivia	Science	60	7
8	Olivia	Maths	55	8
9	Isabella	Maths	50	9

RANK возвращает ранг каждой записи. В данном случае, в отличие от ROW_NUMBER , идет уже анализ значений и в случае нахождения одинаковых возвращает одинаковый ранг с пропуском следующего.

Например:

```
SELECT Studentname,  
       Subject,  
       Marks,  
       RANK() OVER(PARTITION BY Studentname ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY Studentname,  
       Rank;
```

Результат:

	Studentname	Subject	Marks	Rank
1	Isabella	english	90	1
2	Isabella	Science	70	2
3	Isabella	Maths	50	3
4	Lily	Science	80	1
5	Lily	english	70	2
6	Lily	Maths	65	3
7	Olivia	english	89	1
8	Olivia	Science	60	2
9	Olivia	Maths	55	3

partition

Rank

Ещё пример:

```
SELECT Studentname,  
       Subject,  
       Marks,  
       RANK() OVER(ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY Rank;
```

Результат:

	Studentname	Subject	Marks	Rank
1	Isabella	english	90	1
2	Olivia	english	89	2
3	Lily	Science	80	3
4	Lily	english	70	4
5	Isabella	Science	70	4
6	Lily	Maths	65	6
7	Olivia	Science	60	7
8	Olivia	Maths	55	8
9	Isabella	Maths	50	9

DENSE_RANK так же возвращает ранг каждой записи, но в отличие от RANK в случае нахождения одинаковых значений возвращает ранг без пропуска следующего.

Например:

```
SELECT Studentname,  
       Subject,  
       Marks,  
       DENSE_RANK() OVER(ORDER BY Marks DESC) Rank
```

```
FROM ExamResult
ORDER BY Rank;
```

Результат:

	Studentname	Subject	Marks	Rank
1	Isabella	english	90	1
2	Olivia	english	89	2
3	Lily	Science	80	3
4	Lily	english	70	4
5	Isabella	Science	70	4
6	Lily	Maths	65	5
7	Olivia	Science	60	6
8	Olivia	Maths	55	7
9	Isabella	Maths	50	8

Similar Rank

Ещё пример:

```
SELECT Studentname,
       Subject,
       Marks,
       DENSE_RANK() OVER(PARTITION BY Subject ORDER BY Marks DESC) Rank
FROM ExamResult
ORDER BY Studentname,
       Rank;
```

Результат:

	Studentname	Subject	Marks	Rank
1	Isabella	english	90	1
2	Olivia	english	89	2
3	Lily	english	70	3
4	Lily	Maths	65	1
5	Olivia	Maths	55	2
6	Isabella	Maths	50	3
7	Lily	Science	80	1
8	Isabella	Science	70	2
9	Olivia	Science	60	3

Ну, и напоследок, продемонстрируем разницу между DENSE_RANK и RANK :

```
SELECT Studentname,
       Subject,
```

```

Marks,
RANK() OVER(PARTITION BY StudentName ORDER BY Marks ) Rank
FROM ExamResult
ORDER BY Studentname,
Rank;

SELECT Studentname,
Subject,
Marks,
DENSE_RANK() OVER(PARTITION BY StudentName ORDER BY Marks ) Rank
FROM ExamResult
ORDER BY Studentname,
Rank;

```

	Studentname	Subject	Marks	Rank
1	Isabella	Maths	70	1
2	Isabella	Science	70	1
3	Isabella	english	90	2
4	Lily	Maths	65	1
5	Lily	english	70	2
6	Lily	Science	80	3
7	Olivia	Maths	55	1
8	Olivia	Science	60	2
9	Olivia	english	89	3

	Studentname	Subject	Marks	Rank
1	Isabella	Maths	70	1
2	Isabella	Science	70	1
3	Isabella	english	90	3
4	Lily	Maths	65	1
5	Lily	english	70	2
6	Lily	Science	80	3
7	Olivia	Maths	55	1
8	Olivia	Science	60	2
9	Olivia	english	89	3

NTILE – функция Transact-SQL, которая делит результирующий набор на группы по определенному столбцу.

Например:

```

SELECT *,
NTILE(2) OVER(
ORDER BY Marks DESC) Rank
FROM ExamResult
ORDER BY rank;

```

Результат:

	StudentName	Subject	Marks	Rank	
1	Isabella	english	90	1	Group 1
2	Olivia	english	89	1	
3	Lily	Science	80	1	
4	Isabella	Maths	70	1	
5	Isabella	Science	70	1	
6	Lily	english	65	2	Group 2
7	Lily	Maths	65	2	
8	Olivia	Science	60	2	
9	Olivia	Maths	55	2	

Пример 2:

```
SELECT *,  
        NTILE(3) OVER(  
            ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY rank;
```

Результат:

	StudentName	Subject	Marks	Rank	
1	Isabella	english	90	1	Group 1
2	Olivia	english	89	1	
3	Lily	Science	80	1	
4	Isabella	Maths	70	2	Group 2
5	Isabella	Science	70	2	
6	Lily	english	65	2	
7	Lily	Maths	65	3	Group 3
8	Olivia	Science	60	3	
9	Olivia	Maths	55	3	

Пример 3:

```
SELECT *,  
        NTILE(2) OVER(PARTITION BY subject ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY subject, rank;
```

Результат:

	StudentName	Subject	Marks	Rank	
1	Isabella	english	90	1	Group 1
2	Olivia	english	89	1	
3	Lily	english	65	2	
4	Isabella	Maths	70	1	Group 2
5	Lily	Maths	65	1	
6	Olivia	Maths	55	2	
7	Lily	Science	80	1	
8	Isabella	Science	70	1	
9	Olivia	Science	60	2	

[к оглавлению](#)

Для чего используются операторы INTERSECT , EXCEPT в Transact-SQL?

Оператор EXCEPT возвращает уникальные записи из левого входного запроса, которые не выводятся правым входным запросом.

Оператор INTERSECT возвращает уникальные записи, выводимые левым и правым входными запросами.

[к оглавлению](#)

Напишите запрос...

```
CREATE TABLE table (  
    id BIGINT(20) NOT NULL AUTO_INCREMENT,  
    created TIMESTAMP NOT NULL DEFAULT 0,  
    PRIMARY KEY (id)  
);
```

Требуется написать запрос, который вернет максимальное значение id и значение created для этого id :

```
SELECT id, created FROM table where id = (SELECT MAX(id) FROM table);
```

```
CREATE TABLE track_downloads (  
  download_id BIGINT(20) NOT NULL AUTO_INCREMENT,  
  track_id INT NOT NULL,  
  user_id BIGINT(20) NOT NULL,  
  download_time TIMESTAMP NOT NULL DEFAULT 0,  
  PRIMARY KEY (download_id)  
);
```

Напишите SQL-запрос, возвращающий все пары (download_count, user_count), удовлетворяющие следующему условию: user_count — общее ненулевое число пользователей, сделавших ровно download_count скачиваний 19 ноября 2010 года :

```
SELECT DISTINCT download_count, COUNT(*) AS user_count  
FROM (  
  SELECT COUNT(*) AS download_count  
  FROM track_downloads WHERE download_time="2010-11-19"  
  GROUP BY user_id)  
AS download_count  
GROUP BY download_count;
```

[к оглавлению](#)

Источники

- [Википедия](#)
- [Quizful](#)

[Вопросы для собеседования](#)