**Лабораторная работа 1. Расширенные возможности SELECT**

Предметная область БД, на которой необходимо производить упражнения, может быть любой. Рекомендуется использовать БД, которая была разработана в ходе лабораторного практикума по дисциплине «Управление данными». Структура таблиц должна включать в себя, как минимум, одну главную таблицу и две связанные с ней подчинённые таблицы. Например, «Продажи»-«Продавцы»-«Товары», или «Сотрудники»-«Отделы»-«Должности». Таблицы должны быть заполненными тестовыми данными, достаточными для демонстрации работы запросов (не менее 10 записей в каждой таблице).

**Задание:**

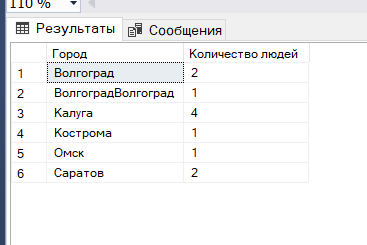
1 Продемонстрируйте работу простого запроса на группировку с отбором записей в исходном наборе до группировки и отбором получившихся групп в результирующем наборе.

Оператор GROUP BY группирует строки с одинаковыми значениями в одну строку.

SELECT Город, COUNT(\*) AS [Количество людей]

FROM Военнообязанные

GROUP BY Город

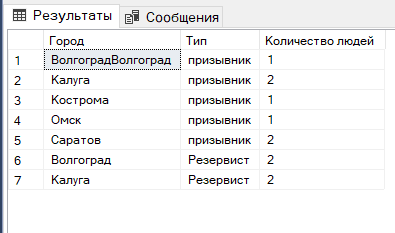


2 Повторите п.1, используя при этом группировку по двум-трём столбцам.

SELECT Город, Тип, COUNT(\*) AS [Количество людей]

FROM Военнообязанные

GROUP BY Город, Тип

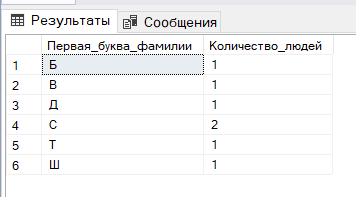


3 Повторите п.1, используя группировку по вычисляемому выражению.

SELECT SUBSTRING(Фамилия, 1, 1) AS Первая\_буква\_фамилии, COUNT(\*) AS Количество\_людей

FROM Сотрудники

GROUP BY SUBSTRING(Фамилия, 1, 1)



4 Продемонстрируйте работу простого запроса с использованием группировки по результату соединения (join) имеющихся таблиц. Покажите проблему группировки кортежей подчинённой таблицы по неуникальному полю одной из связанных таблиц. Продемонстрируйте более правильный вариант группировки.

SELECT

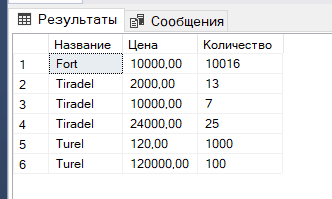
Товар.Название,

Товар.Цена,

sum([Закупка товаров].Количество\_товара) as Количество

FROM Товар JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY Товар.Название,Товар.Цена



Проблема заключается в том, что группировка выполняется по неуникальному полю "Название" из таблицы "Товар", разные товары могут иметь одинаковое название.

Более правильно будет использовать уникальный идентификатор товара (Id) вместо названия товара.

SELECT

Товар.Название,

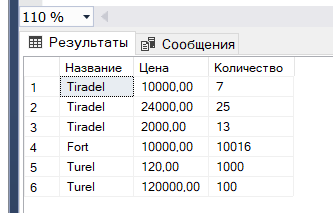
Товар.Цена,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY Товар.Id, Товар.Название, Товар.Цена



5 Модифицируйте запрос из п.4 так, чтобы в нём появились подитоги по иерархии значений в столбцах группировки. Используйте для этого оператор ROLLUP. При этом покажите использование различного количества столбцов в операторе ROLLUP.

SELECT

Товар.Название,

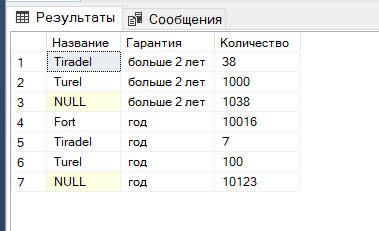
Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY Rollup(Товар.Название), Товар.Гарантия



Показывает общее количество всех товаров с одинаковой гарантией.

SELECT

Товар.Название,

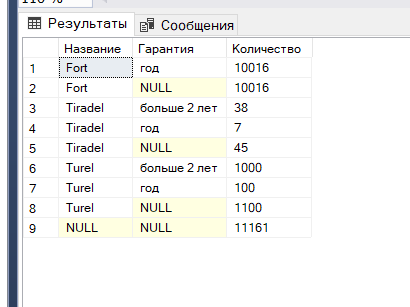
Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY Rollup(Товар.Название, Товар.Гарантия)



Показывает общее количество товаров с общим названием и в конце показывает количество закупленных товаров всего.

6 Модифицируйте запрос из п.4 так, чтобы в нём появились подитоги по комбинациям значений в столбцах группировки. Используйте для этого оператор CUBE. При этом покажите использование различного количества столбцов в операторе CUBE.

SELECT

Товар.Название,

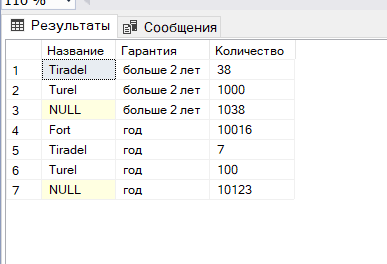
Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY CUBE(Товар.Название), Товар.Гарантия



Показывает общее количество товаров, если у них общая гарантия.

SELECT

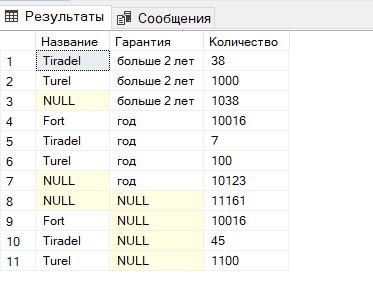
Товар.Название,

Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY CUBE(Товар.Название, Товар.Гарантия)

Показывает вообще все комбинации подитогов

7 Модифицируйте запрос из п.4. установив при помощи оператора GROUPING SETS произвольный набор конфигураций уровней блокирования.

SELECT

Товар.Название,

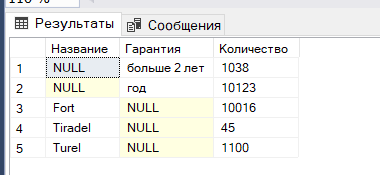
Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

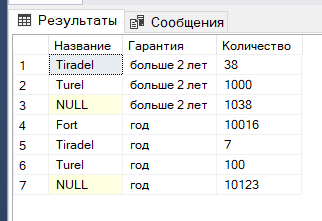
GROUP BY GROUPING SETS(Товар.Название, Товар.Гарантия)



В начале группирует по гарантии, затем по названию.

8 Продемонстрируйте примеры эквивалентных соотношений между операторами ROLLUP, CUBE и оператора GROUPING SETS.

1) эти два кода с CUBE и ROLLUP показывают одинаковый результат.



SELECT

Товар.Название,

Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY CUBE(Товар.Название), Товар.Гарантия

SELECT

Товар.Название,

Товар.Гарантия,

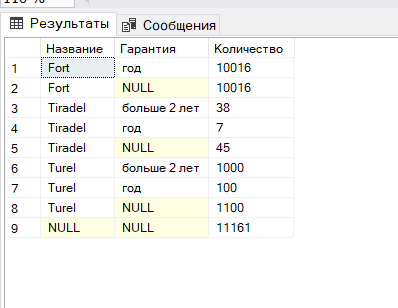
SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY RoLLUP(Товар.Название), Товар.Гарантия

2) Код с ROLLUP и GROUPING SETS выдает один результат.



SELECT

Товар.Название,

Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY GROUPING SETS(

(Товар.Название, Товар.Гарантия),

(Товар.Название),

()

)

SELECT

Товар.Название,

Товар.Гарантия,

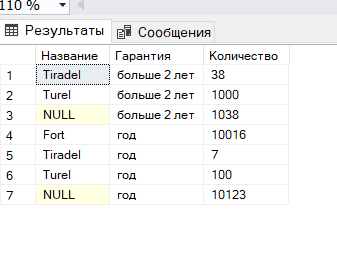
SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY Rollup(Товар.Название, Товар.Гарантия)

3) Эти коды с CUBE и GROUPING SETS выдают одинаковый результат.



SELECT

Товар.Название,

Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY CUBE(Товар.Название), Товар.Гарантия

SELECT

Товар.Название,

Товар.Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество]

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY GROUPING SETS(

(Товар.Название, Товар.Гарантия),

(Товар.Гарантия)

)

9 При помощи оператора GROUPING отделите в итоговом наборе запроса из п.4 значения NULL, показывающие исключение соответствующего атрибута из группирования от значений NULL, показывающие отсутствующие значения. Для этого замените первые - на строку «ВСЕ», а вторые – на строку «НЕИЗВЕСТНО».

SELECT

isNULL(Товар.Название,

CASE WHEN GROUPING(Товар.Название)=1 THEN 'ВСЕ' ELSE 'НЕИЗВЕСТНО' end)as Название,

isNULL(Товар.Гарантия,

Case when Товар.Гарантия IS NULL AND GROUPING(Товар.Гарантия)=1 THEN 'ВСЕ' else 'НЕИЗВЕСТНО' end) as Гарантия,

SUM([Закупка товаров].Количество\_товара) AS [Количество],

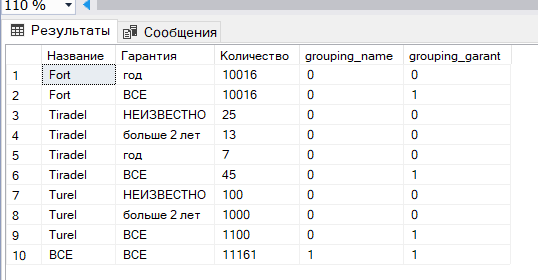
GROUPING(Товар.Название) as grouping\_name,

GROUPING(Товар.Гарантия) as grouping\_garant

FROM Товар

JOIN [Закупка товаров] ON Товар.Id = [Закупка товаров].Id\_товара

GROUP BY rollup(Товар.Название, Товар.Гарантия)



10 Продемонстрируйте работу ранжирующих функций RANK, DENSE\_RANK, ROW\_NUMBER и NTILE. Наглядно покажите разницу между ними.

DENSE\_RANK

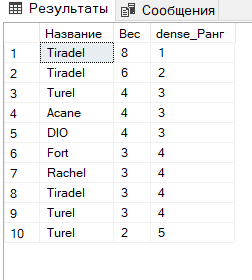
SELECT

Товар.Название,

Товар.Вес,

DENSE\_RANK() Over (ORDER BY Вес desc) as dense\_Ранг

FROM Товар;



Присваивает ранг каждой строке, если значения одинаковые, то присваивает им один и тот же ранг.

RANK

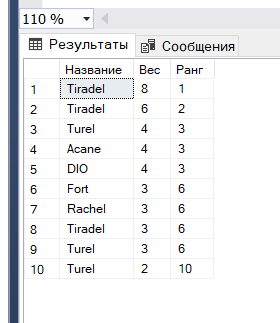
SELECT

Товар.Название,

Товар.Вес,

RANK() OVER (ORDER BY Вес desc) as Ранг

FROM Товар;



Присваивает ранг каждой строке, пропуская следующие значения если предыдущие были равны.

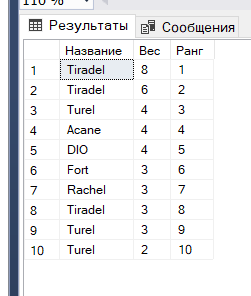
SELECT

Товар.Название,

Товар.Вес,

Row\_NUMBER() OVER (ORDER BY Вес desc) as Ранг

FROM Товар;



Просто нумерует строки по порядку без учета значений.

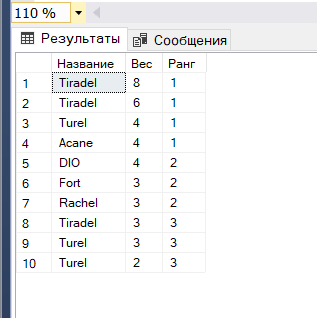
SELECT

Товар.Название,

Товар.Вес,

NTILE(3) OVER (ORDER BY Вес desc) as Ранг

FROM Товар;



делит результат на указанное количество групп, присваивая каждой строке номер группы.

11 Повторите пункт 10, но с применением оконных функций. В качестве критерия выделения окон можно выбрать отдел или должность для таблицы «Сотрудники» или категорию для таблицы «Товары».

SELECT

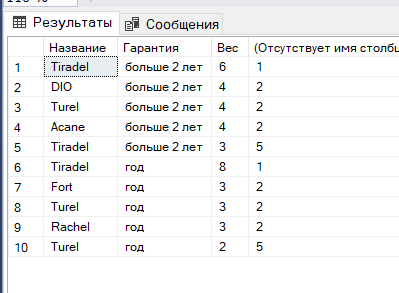
Товар.Название,

Товар.Гарантия,

Товар.Вес,

Rank() over(Partition by Гарантия Order by Вес desc)

From Товар



SELECT

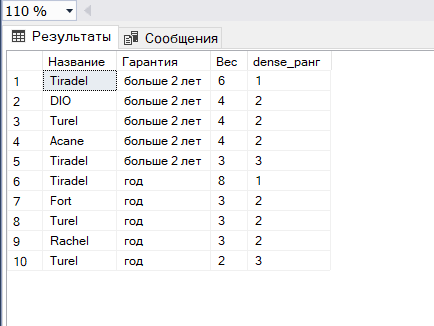
Товар.Название,

Товар.Гарантия,

Товар.Вес,

DENSE\_RANK() OVER (PARTITION by Гарантия order by вес desc) as dense\_ранг

From Товар



SELECT

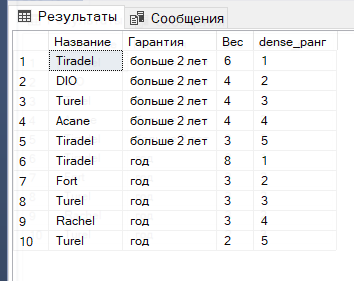
Товар.Название,

Товар.Гарантия,

Товар.Вес,

Row\_NUMBER() OVER (PARTITION by Гарантия order by вес desc) as dense\_ранг

From Товар



SELECT

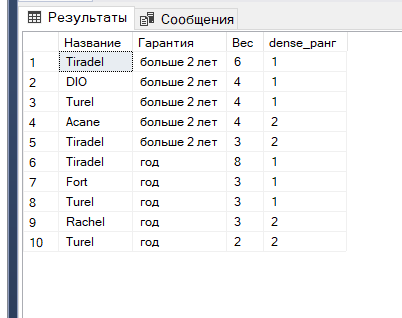
Товар.Название,

Товар.Гарантия,

Товар.Вес,

NTILE(2) OVER (PARTITION by Гарантия order by вес desc) as dense\_ранг

From Товар



12 Добавьте в одну из таблиц Вашей схемы атрибут, который будет являться внешним ключом, указывающим на записи этой же таблицы. Например, в таблицу «Сотрудники» добавьте информацию о руководителе для каждого сотрудника или для таблицы «Товары» - информацию о сопутствующем товаре, который прилагается к данному товару в подарок по рекламной акции. Составьте рекурсивное табличное выражение, в котором наглядно выведите записи вашей таблицы в порядке их иерархии. Предусмотрите визуальное отображение иерархии, например, при помощи отступов различной величины.

Alter table Товар

ADD id\_товара\_по\_акции int,

Foreign Key (id\_товара\_по\_акции) references Товар(Id)

with recursia(Id, Id\_товара\_по\_акции, Название, Уровень) as(

select Товар.Id,

Товар.id\_товара\_по\_акции,

Товар.Название,

0

from Товар

Where Товар.id\_товара\_по\_акции is Null

Union all

select Товар.Id,

Товар.id\_товара\_по\_акции,

Товар.Название,

recursia.Уровень+1

from Товар inner join recursia on Товар.id\_товара\_по\_акции= recursia.Id

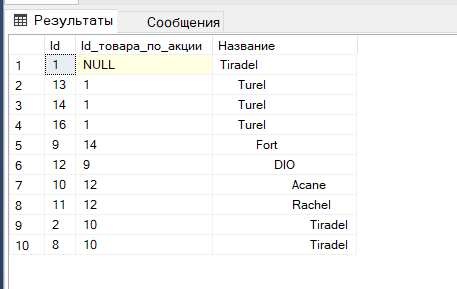
)

Select recursia.Id,

recursia.Id\_товара\_по\_акции,

(Space(recursia.Уровень \*6)+ recursia.Название) as Название

from recursia



13 Продемонстрируете владение оператором слияния наборов MERGE. Для этого организуйте слияние с существующей в Вашей схеме таблицы набора данных, полученного при помощи запроса к любой другой таблице. Например, в качестве источника строк можно использовать базу данных Adventure Work, входящую в комплект поставки MS SQL Server. В качестве критерия слияния выберите соответствие значений в столбцах, которые могут служить естественным идентифицирующим признаком для сущностей (например, серия и номер паспорта для сотрудника или артикул для товара).

merge into [Процесс прохождения мед комиссии] P

using (Select \* from Военнообязанные) V

on (P.Id\_военнообязанного =V.Id)

when matched then

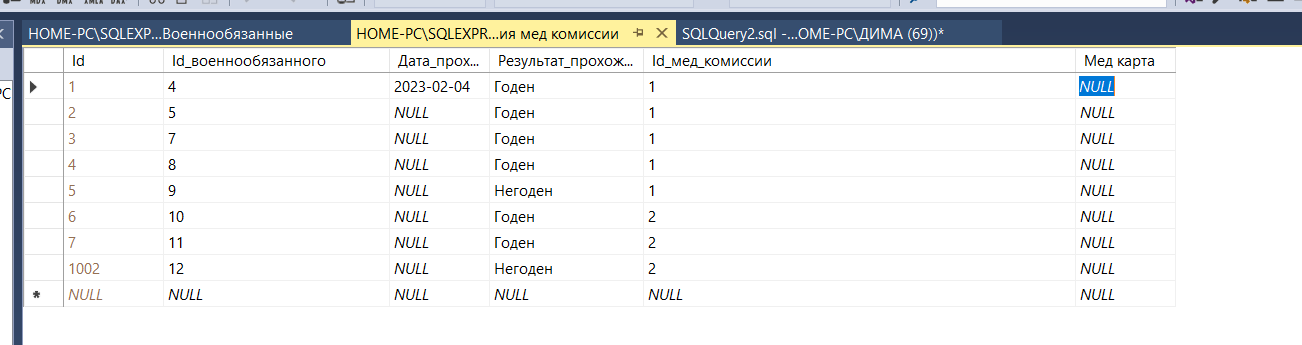
update set P.[Мед карта]=V.Здоровье

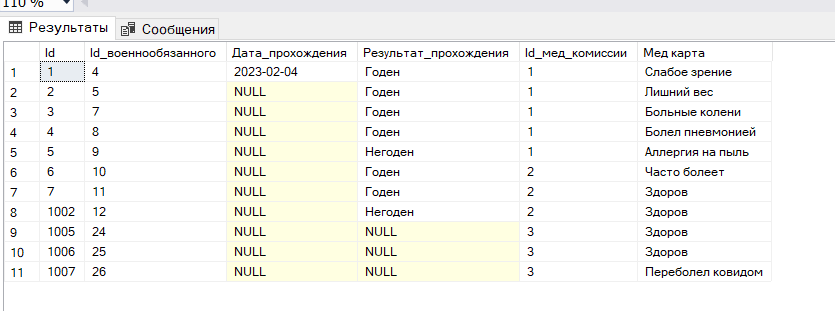
when not matched then

Insert (id\_военнообязанного,Дата\_прохождения,Результат\_прохождения,Id\_мед\_комиссии, [Мед карта])

values (V.Id,NULL,NULL,3,V.Здоровье);

select \* from [Процесс прохождения мед комиссии]





**Вопросы к отчёту:**

* Простая группировка данных. Операторы group by, having.

Оператор GROUP BY

Группировка данных позволяет объединить одинаковые значения в заданных полях в группы, а затем выполнять подсчеты для каждой группы.

Оператор HAVING определяет, какие группы будут включены в выходной результат, то есть выполняет фильтрацию групп.

Применение HAVING во многом аналогично применению WHERE. Только есть WHERE применяется к фильтрации строк, то HAVING используется для фильтрации групп.

* Ограничения оператора group by.

1. Столбцы в SELECT: Все столбцы, которые не являются агрегатными функциями, должны быть либо указаны в операторе GROUP BY, либо быть вложены в агрегатные функции.

И если в выражении SELECT производится выборка по одному или нескольким столбцам и также используются агрегатные функции, то необходимо использовать выражение GROUP BY

2. Для GROUP BY все значения NULL трактуются как равные, то есть при группировке по полю, содержащему NULL-значения, все такие строки попадут в одну группу.

* Группировка по вычисляемым выражениям.

В SQL можно использовать вычисляемые выражения для группировки данных. Например, можно создать новый столбец в запросе, используя математические операции или функции, и затем сгруппировать данные по этому вычисляемому выражению.

* Составная группировка по двум и более столбцам. В чём её смысл?

Составная группировка по двум и более столбцам в SQL позволяет сгруппировать данные на основе комбинации значений из нескольких столбцов. Это может быть полезно, когда нужно разделить данные на более детальные категории.

* Промежуточные подитоги. ROLLUP, CUBE и GROUPING SETS. Принципы работы, отличия, взаимоотношения.

\*\*Оператор CUBE\*\*:

- CUBE создает все возможные комбинации подитогов по всем столбцам, указанным в операторе CUBE.

- Например, если у нас есть столбцы A, B и C, то оператор CUBE(A, B, C) создаст подитоги для каждой комбинации этих столбцов (A, B, C, AB, AC, BC, ABC).

- CUBE генерирует значительно больше строк в результирующем наборе по сравнению с ROLLUP.

\*\*Оператор ROLLUP\*\*:

- ROLLUP создает иерархические подитоги по заданным столбцам.

- Например, если у нас есть столбцы A, B и C, то оператор ROLLUP(A, B, C) создаст подитоги для каждого уровня иерархии (A, AB, ABC).

- ROLLUP генерирует меньше строк в результирующем наборе по сравнению с CUBE, так как он строит иерархию подитогов.

GROUPING SETS позволяет явно указывать комбинации столбцов, по которым нужно сделать группировку и итоги. Он позволяет группировать данные по нескольким комбинациям столбцов одновременно. Например, можно указать группировку по столбцам A и B, а также отдельно по столбцу C.

Основное различие между операторами CUBE и ROLLUP заключается в том, что CUBE создает все возможные комбинации подитогов по всем столбцам, в то время как ROLLUP создает иерархические подитоги по заданным столбцам.GROUPING SETS позволяет явно указывать комбинации столбцов для группировки и итогов.

Взаимоотношения:

- ROLLUP является частным случаем CUBE, так как он создает все комбинации группировки, но без некоторых подмножеств.

- GROUPING SETS можно использовать как альтернативу ROLLUP или CUBE, позволяя указывать конкретные комбинации столбцов для группировки и итогов.

* Ранжирующие функции. Их возможности и сферы применения. Указание критериев ранжирования.

Ранжирующие функции в SQL позволяют присваивать каждой строке результата запроса ранг или порядковый номер в соответствии с определенными критериями. Это полезный инструмент для анализа данных

1. \*\*Основные ранжирующие функции\*\*:

- \*\*ROW\_NUMBER()\*\*: Присваивает каждой строке уникальный порядковый номер в пределах упорядоченного набора данных.

- \*\*RANK()\*\*: Присваивает одинаковый ранг для равных значений и пропускает следующий ранг.

- \*\*DENSE\_RANK()\*\*: Присваивает одинаковый ранг для равных значений, но не пропускает следующий ранг.

-\*\*NTILE\*\*: Эта функция присваивает каждой строке номер группы, к которой она относится. Количество групп определяется параметром, переданным в функцию NTILE.

2. \*\*Сферы применения\*\*:

- \*\*Аналитика данных\*\*: Ранжирующие функции часто используются для анализа данных и выявления топ-N записей или наименее значимых записей.

- \*\*Отчеты и дашборды\*\*: Ранжирующие функции помогают создавать отчеты с выделением наиболее важных данных или сегментов.

- \*\*Работа с временными рядами\*\*: Ранжирование данных по времени может быть полезно при анализе изменений во времени.

3. \*\*Критерии ранжирования\*\*:

- Ранжирование может осуществляться по одному или нескольким столбцам в результате запроса.

- Критерии ранжирования могут быть заданы в порядке возрастания (ASC) или убывания (DESC).

- При необходимости можно использовать дополнительные условия для определения критериев ранжирования.

* Оконные функции. Применение совместно с агрегатными или ранжирующими функциями.

Окно — это набор строк, определяемый пользователем. Оконная функция вычисляет значение для каждой строки в результирующем наборе, полученном из окна.

**OVER**

определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

**PARTITION BY**

Разделяет результирующий набор на секции. Оконная функция применяется к каждой секции отдельно, и вычисление начинается заново для каждой секции.

* Обобщённые табличные выражения. Сферы применения обобщённых табличных выражений.

Обобщенные табличные выражения (ОТВ) можно представить себе как временные результирующие наборы, определенные в области выполнения единичных инструкций SELECT, INSERT, UPDATE, DELETE или CREATE VIEW. ОТВ не сохраняются в базе данных в виде объектов, время их жизни ограничено продолжительностью запроса. При этом ОТВ могут ссылаться сами на себя, а на них один и тот же запрос может ссылаться несколько раз.

ОТВ предназначены для:

* Создания рекурсивных запросов.
* Замены представлений в тех случаях, когда использование представления не оправдано, то есть тогда, когда нет необходимости сохранять в метаданных базы его определение.
* Группирования по столбцу, производного от скалярного подзапроса выборки или функции, которая недетерминирована или имеет внешний доступ.
* Многократных ссылок на результирующую таблицу из одной и той же инструкции.

Применение ОТВ позволяет значительно повысить читаемость и упростить работу со сложными запросами, разбив его на отдельные логические строительные блоки.

* Организация рекурсивных запросов при помощи обобщённых табличных выражений.

ОТВ состоит из имени выражения, необязательного списка столбцов и определяющего ОТВ запроса. После определения ОТВ на него можно ссылаться из инструкций SELECT, INSERT, UPDATE и DELETE как на таблицу или представление.

* Оператор слияния наборов MERGE. Возможности и сферы его применения.

Оператор слияния наборов (MERGE) в SQL используется для выполнения операций вставки, обновления и удаления данных в таблице на основе сравнения с данными из другой таблицы или подзапроса. Оператор MERGE предоставляет удобный способ обновления данных в таблице, основываясь на условиях, что данные уже существуют или не существуют в целевой таблице.

Возможности и сферы применения оператора MERGE:

1. Обновление или вставка данных: Оператор MERGE позволяет одновременно проверить наличие данных в целевой таблице и выполнить операцию обновления или вставки в зависимости от результата проверки. Это удобно, когда нужно обновить существующие данные или вставить новые данные в таблицу.

2. Синхронизация данных: MERGE может использоваться для синхронизации данных между различными источниками данных или таблицами. Например, можно обновлять данные в целевой таблице на основе изменений в другой таблице.

3. Удаление данных: Оператор MERGE также позволяет выполнять операции удаления данных на основе условий. Это полезно, когда нужно удалить данные из целевой таблицы, которых нет в исходной таблице.

4. Избегание дублирования данных: MERGE помогает избежать дублирования данных при вставке новых записей, так как он автоматически проверяет наличие данных перед выполнением операции.

5. Эффективность: Оператор MERGE может быть более эффективным с точки зрения производительности, чем выполнение отдельных операций INSERT, UPDATE и DELETE.