Объединение результатов запроса с помощью предложения UNION

Предложение **UNION** используется для объединения результатов двух или более запросов. Предложение **UNION** производит слияние результатов первого и второго запросов, неявно удаляя повторяющиеся строки. В результате получается единственный набор, который состоит из всех строк, присутствующих в обоих таблицах.

Определять запросы, которые содержат предложение **UNION**, следует таким образом, чтобы они были совместимыми. Запросы должны иметь одинаковое количество столбцов и общий столбец, определенный в каждой таблице

Пример 1

SELECT badge FROM employees

UNION

SELECT badge FROM pays ORDER BY badge

Значение столбца bagde, который является общим для обеих таблиц, отображается с помощью двух инструкций **SELECT**, объединенных предложением **UNION**. Для упорядочивания итоговых результатов к запросу добавлено предложение **ORDER BY**, которое вступает в силу только после выполнения последней инструкции **SELECT**. Предложение UNION производит слияние результатов первого и второго запросов, неявно удаляя повторяющиеся строки:

Пример 2

SELECT name, badge FROM employees

WHERE department in ("Sales", ...)

UNION

SELECT name, badge FROM employees2

UNION

SELECT name, badge FROM employees3 WHERE department = "Sales" ORDER BY name, badge

Как с помощью предложения **UNION** объединить три таблицы. В первой таблице для задания нескольких вариантов написания одного отдела используется ключевое слово

IN. Во второй таблице нет предложения **WHERE**, поэтому выбираются все строки. В третьей таблице задано единственное имя отдела:

Использование функций

1. Функции для работы с данными таблиц базы данных

AVG, SUM, MIN, MAX, COUNT - функции агрегирования

Пример

SELECT AVG(pay_rate) FROM

В MS SQL добавлены дополнительные функции:

строковые - **CHAR, STR, ASCII**

арифметические - **ACOS, ASIN, ABS, RAND ...**

для работы с датой - **DATENAME, DATEPART, DATEADD ...**

2. Системные функции

Их используют для получения информации о компьютерных системах, пользователях, базах данных

HOST_NAME() - имя сервера

HOST_ID() - идентификатор сервера

USER_NAME() - имя пользователя

DB_NAME() - имя БД

Пример

SELECT HOST_NAME(), HOST_ID()

Индексы и ключи

Рассмотрим следующее назначение индексов.

Индексы используются для того, чтобы упростить и ускорить выборку данных. Они могут создаваться одновременно с таблицей или добавляться к таблице позднее.

Создание индексов различных типов

Имеется несколько типов индексов. Каждый индекс отслеживает поступающие данные и различными способами сохраняет их в таблице. Кроме этого, некоторые индексы влияют на физический способ хранения информации в БД.

Назначение и создание ключей

Строки данных содержат уникальные значения, которые можно использовать для идентификации каждой строки. Столбцы, которые содержат эти уникальные значения, называются ключевыми столбцами. Чтобы гарантировать уникальность ключевых столбцов, они объявляются при создании таблицы.

Чтобы получить ответ на любой запрос из таблицы, не имеющей индекса, SQL сервер вынужден сканировать таблицу, то есть считывать в ней каждую строку. Индексы обеспечивают механизм указателей на требуемые данные и работают аналогично справочнику.

Определение индексов

Индексы в SQL Server - это внутренний способ организации данных в таблице для их оптимальной выборки. Индексы представляют собой наборы уникальных для данной таблицы значений и соответствующий им список указателей на страницы данных, где эти значения находятся физически.

Правила выбора столбцов для определения индекса:

1. первичные и внешние ключи
2. запросы с большим объемом возвращаемых данных
3. поддержка сортировки и группировки**(ORDER BY** и **GROUP BY**)

Создание индекса

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX имя_индекса ON [[база_данных.]владелец.] имя_таблицы (имя_столбца)

UNIQUE . Если индекс создается с опцией **UNIQUE**(уникальный), SQL Server не допускает повторяющихся значений в индексе и поэтому запрещает добавлять в таблицу БД записи с повторяющимися значениями в столбце уникального индекса. Уникальные индексы не могут создаваться в столбцах с повторяющимися значениями - эти значения нужно удалить.

CLUSTERED. Кластерный индекс - это специальный индекс, который заставляет SQL Server физически хранить данные в файлах именно в том формате, который задает этот индекс. Использование этого индекса позволяет значительно сократить время доступа к данным.

NONCLUSTERED(по умолчанию).

Некластерный индекс создается по умолчанию. Это значит, что SQL Server будет создавать индекс, страницы индексированных данных которого содержат указатели на реальные страницы данных в базе.

Создание таблицы и индексов для нее

Пример

CREATE TABLE authors(

au_id varchar(40) NOT NULL,

au_lname varchar(40) NOT NULL,

au_fname varchar(40) NOT NULL,

phone char(12) NOT NULL,

address varchar(40) NULL,

city varchar(40)NULL,

state char(2) NULL,

zip char(5) NULL,

contact bit NOT NULL

)

CREATE INDEX aunmind ON dbo,authors2(au_lname, au_fname)

CREATE UNIQUE INDEX ON fk_au_id ON dbo.authors2(au_id)

Определение ключей

Ключи и индексы в БД часто имеют одно и то же значение, но в SQL Server они слегка отличаются между собой. Ключи могут быть определены для таблицы, а затем использоваться в качестве ограничений целостности ссылок, как это определено в структуре ANSI для SQL.

Первичный ключ - это столбец или группа столбцов с уникальными значениями, определяющими строки таблицы БД. В этом смысле первичный ключ в поддержании целостности таблицы выполняет роль уникального индекса. SQL Server допускает определение для таблицы только одного первичного ключа, а уникальных индексов может быть много.

Внешние ключи - это столбец таблицы, который соответствует первичному ключу в другой таблице.

Пример

CREATE TABLE table_a (

COLUMN_A smallint PRIMARY KEY

)

CREATE TABLE table_b (

COLUMN_B smallint CONSTRAINT PK_COLUMN_B PRIMARY KEY

)

Внешний ключ

CREATE TABLE table_c (

COLUMN_C smallint FOREIGN KEY (COLUMN_E) REFERENCES table_a (COLUMN_A)

)

Первичный ключ для нескольких столбцов

CREATE TABLE table_d (

COLUMN_D1 smallint CONSTRAINT PK_D_COLUMNS PRIMARY KEY(COLUMN_D1, COLUMN_D2),

COLUMN_D1 smallint

)