3. Прохоров, С.А. Паттерновое проектирование при создании комплекса программ для проведения вейвлет-анализа / С.А. Прохоров, А.А. Столбова // Известия СНЦ РАН. – Самара, 2015. – т. 17, № 2(5). – С. 1092-1096.

4. OpenMP [Электронный ресурс] // OpenMP Homepage. – Режим доступа: http://www.openmp.org.

5. Артамонов, Ю.С. Инструментальное программное обеспечение для разработки и поддержки исполнения приложений научных вычислений в кластерных системах / Ю.С.Артамонов, С.В. Востокин // Вестн. Сам. гос. техн. ун-та. Сер. Физ.-мат. науки, 19:4 (2015), 785–798.

М.В. Терёхин, Е.И. Чигарина

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ РАНЖИРОВАНИЯ В СИСТЕМАХ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

(Самарский университет)

В системах баз данных одной из основных задач является задача сокращения времени выполнения запросов при работе с данными с использованием минимального объема памяти, то есть задача эффективного выполнения запросов. В данной работе выполнен анализ эффективности реализации запросов с использованием функций ранжирования по сравнению с традиционными запросами без них.

Функции ранжирования возвращают ранг каждой записи внутри «окна». В общем случае рангом является число, отражающее положение или «вес» записи относительно других записей в том же наборе. Формируется «окно» с помощью группировки. Однако, поскольку результат работы функций ранжирования зависит от порядка обработки записей, то обязательно должен быть указан порядок записей внутри «окна» посредством конструкции ORDER BY. Функции ранжирования являются недетерминированными, то есть при одних и тех же входных значениях они могут возвращать разный результат [1].

Существует четыре функции ранжирования:

- ROW_NUMBER() – функция, выполняющая нумерацию записей в указанном порядке внутри «окна»;
- RANK() – функция, раздающая такие номера записям, что если встретятся несколько записей с одинаковым значением, то этим записям будет присвоен одинаковый ранг. Следующая запись с новым значением получит такой ранг, как будто бы предыдущие записи получили свой уникальный номер, то есть образуется пропуск в нумерации;
- DENSE_RANK() – функция, такая же, что и RANK(), но без пропусков ранга;
- NTILE() – функция, разделяющая записи на указанное количество групп.

Общая структура функций ранжирования имеет вид:

ФУНКЦИЯ_РАНЖИРОВАНИЯ ( )

OVER ([PARTITION BY столбец1,...] ORDER_BY столбец1,...)

Раздел PARTITION BY делит результирующий набор на секции, к которым применяется функция ранжирования, и определяет столбцы, по которым секционируется результирующий набор. Если PARTITION BY не указан, функция обрабатывает все строки результирующего набора запроса как одну группу. Раздел ORDER BY определяет сортировку в пределах указанной секции.

Для выполнения анализа использования функций ранжирования в запросах определен перечень параметров, влияющих на их использование. К таким параметрам относятся:

- время выполнения запроса;
- структура данных.

Альтернативой использования оконных функций могут выступать запросы, использующие самосоединение, вложенные запросы, курсоры, а также хранимые процедуры, функции, реализующие действия оконных функций.

В качестве СУБД для анализа работы функций использована СУБД MS SQL Server, имеющая встроенное приложение SQL Server Profiler, с помощью которого можно выполнить трассировку запроса.

Для выполнения анализа использована таблица «Студент», имеющая следующую структуру:

Имя столбца	**Тип данных**	**Назначение**
ID_record_book	bigint	№ зачетной книжки студента
FIO	varchar(max)	ФИО студента
Date_of_born	datetime	Дата рождения
ID_group	int	№ группы
Specialty	varchar(max)	Специальность
Stipend	money	Стипендия
Average_performance_score	real	Средний балл успеваемости

Рассмотрены варианты заполнения таблицы по 1000, 10000 и 100000 записей.

В работе составлены группы запросов и альтернативы к ним по следующей методике:

- первая группа запросов не использует предложение для секционирования PARTITION BY. Запросы отличаются столбцом сортировки в предложении ORDER BY. Используются столбцы ID_group, Specialty и year(Date_of_born) (от даты берется только год);
- вторая группа запросов использует предложение для секционирвоания PARTITION BY. Секционирование происходит по столбцам ID_group, Specialty и year(Date_of_born) (от даты берется только год). Сортировка ORDER BY не меняется и происходит по столбцу Stipend постоянно.

Разные столбцы используются для анализа влияния типа данных столбцов на вермя выполнения запросов.

После составления запросов, были проведены замеры их времени выполнения. На каждый запрос 10 попыток. Затем вычислено среднее значение вре-

мени выполнения для каждого варианта запроса. Поскольку MS SQL Server строит перед выполнением запроса его план для ускорения последующих похожих запросов, для чистоты эксперимента необходимо очищать кэш и план запросов перед каждым замером.

Для функции ROW_NUMBER() без секционирования альтернативным вариантом является использование временной таблицы. Альтернатива имеет более трудоемкую реализацию, что может оказаться неудобным для программиста. Единственная возможная альтернативная реализация запроса с секционированием PARTITION BY становится еще более трудоемкой и громоздкой, поскольку заменой такого запроса может быть только курсор. Коме того, среднее время выполнения такого запроса в 1000 раз больше, чем среднее время выполнения запроса с функцией ранжирования, и составляет около 58 секунд, и это для таблицы с числом записей 1000. Поэтому производить замеры выполнения такого запроса для таблицы с еще большим числом записей не имеет смысла.

В результате измерений была составлена сводная таблица со средними значениями времени выполнения запросов с использованием функций ранжирования и альтернативных вариантов. В этой и последующих таблицах указаны типы данных столбца, по которому выполнялась сортировка данных для запросов без секционирования, и тип данных столбца, по которому выполнялось секционирование, для соответствующих случаев. Время указано в милисекундах.

Число зап.	**Без секционирования**						**С секционированием**					
	int		**varchar(max)**		**datetime(year)**		**int**		**varchar(max)**		**datetime(year)**	
	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт
1'000	57	70,7	69,3	73,8	55,3	59,5	55,6	58763	61,8	57846	51,1	66247,3
10'000	132,6	168,2	240,6	281,8	126,7	161,9	129,2	326389	235,8	-	130,7	-
100'000	1495,6	2305,6	2392,3	3298,7	1480,3	2469,5	1512,5	-	2513,4	-	1533	-

Как видно из таблицы, для запросов без секционирования преимущество во времени выполнения запроса функций ранжирования становится заметным при увеличении числа записей. Для запросов с секционированием, поскольку альтернативой является только курсор, ясно, что функции ранжирования определенно выигрывают.

Функцию RANK() можно заменить используя вложенный запрос.

Число зап.	**Без секционирования**						**С секционированием**					
	int		**varchar(max)**		**datetime(year)**		**int**		**varchar(max)**		**datetime(year)**	
	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт
1'000	61,1	67,3	68,4	71,6	60,6	179,2	61,1	65	72,2	83,4	60,7	133,9
10'000	136,1	160,3	228,4	270,4	135,8	3028,3	131,3	172,1	247,6	353,5	141,6	5244,5
100'000	1499,2	1665,7	2238,1	2810,6	1475,2	30842,8	1587,3	1712,2	2528,8	3712	1551	80542,4

По данным из таблицы можно сделать вывод, что с увеличением числа записей в таблице, использование функции ранжирования позволяет сократить время выполнения запроса.

Для функции DENSE_RANK() были получены данные:

Число зап.	Без секционирования						С секционированием					
	int		varchar(max)		datetime(year)		int		varchar(max)		datetime(year)	
	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт
1'000	61,5	65,3	63,2	85,2	59,8	307,4	64,1	68,5	69,5	85	62,8	191,4
10'000	137,1	151,2	237,1	310,5	131,6	4599,1	137,7	180	243,9	404,3	141,1	8470,8
100'000	1467,2	1639,3	2423,1	3152,5	1509,7	46686,3	1506,7	1728,5	2525	4133,7	1524,2	236362,9

Из таблицы видно, что с увеличением числа записей в таблице выгоднее использовать функцию ранжирования, а не вложенный запрос.

Функция NTILE() имеет особенность, что ее можно выразить через функцию ROW_NUMBER() с помощью формулы:

NTILE(n) = (n * (ROW_NUMBER() – 1) / count(*)) + 1.

В результате замеров этой функции и ее альтернатив были получены данные:

Число зап.	Без секционирования						С секционированием					
	int		varchar(max)		datetime(year)		int		varchar(max)		datetime(year)	
	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт	ФР	Альт
1'000	67,5	71,7	71,6	67	66,6	68,7	61,2	71491,6	76,2	72424	61	71853,3
10'000	161,9	152,6	251,3	246,5	160,2	156,9	161,8	-	281,8	-	161,5	-
100'000	1722,6	1652,6	2570,9	2492,9	1780,5	1698,3	1738,6	-	2917,8	-	1748,9	-

В случае запроса без секционирования наблюдается преимущество альтернативного запроса, хоть и незначительное. Курсор будет выполняться быстрее функции ранжирования на всех рассмотренных типах данных, если секционирование не используется. Тем не менее, реализация альтернативного запроса без функции ранжирования является более трудоемкой для программиста.

Таким образом, исходя из произведенного исследования, можно сделать вывод о том, что использование функций ранжирования в системах реляционных баз данных позволяет сократить время выполнения запросов, а также текст запроса. Исключением является функция NTILE() без секционирования данных, для которой запрос без функции ранжирования является более производительным.

Литература

1. MS SQL 2005: оконные функции. Еще одно расширение T-SQL [Электронный ресурс]. – http://rsdn.ru/?article/db/WindowFunctions.xml

Д.А. Царёв, С.В. Востокин

ТЕХНОЛОГИЯ РАЗВЕРТЫВАНИЯ СКЕЛЕТНЫХ ПРОГРАММ ДЛЯ АВТОМАТИЗАЦИИ ВЫЧИСЛЕНИЙ НА СУПЕРКОМПЬЮТЕРЕ «СЕРГЕЙ КОРОЛЁВ»

(Самарский университет)

Задачи для высокопроизводительных вычислений на кластерных системах и суперкомпьютерах можно разделить на две категории. К первой категории относятся задачи математического моделирования, решаемые с помощью