УДК 681.3

А.И. Баранчиков, П.А. Баранчиков

ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ ДИСКРЕЦИОННО-РОЛЕВОГО ДОСТУПА НА ЧТЕНИЕ К ЗАПИСЯМ БД

Рассматривается вычислительная эффективность различных алгоритмов, реализующих методы доступа к отдельным записям таблиц БД на примере нескольких высокопроизводительных СУБД различного класса. Рассматриваются реляционные БД. При построении всех реализаций ограничения доступа в работе использовался аппарат реляционной алгебры.

Ключевые слова: *базы данных, доступ, модель доступа, оптимизация запросов.*

Введение. Расширение областей применения различных систем автоматизации, основанных на реляционных БД [1] привело к необходимости разработки более гибких методов контроля доступа пользователей к данным в БД.

Использование дискреционно-ролевой модели доступа к записям таблиц БД описано изначально в [2]. Суть модели заключается в том, что каждой записи назначены владелец и группа. Владелец имеет полный доступ к записи, в том числе может изменять настройки доступа этой записи.

Помимо владельца, к записи имеют доступ пользователи, принадлежащие к заданной группе. В БД принято называть группы ролями (по названию модели доступа). Пользователю, входящему в группу, предоставлена соответствующая роль.

У каждой записи установлена только одна группа (роль), члены которой имеют доступ к этой записи. Каждый пользователь может входить в несколько различных групп. Пользователю для предоставления доступа к необходимой ему совокупности записей предоставляется соответствующая роль, обладающая доступом к этим записям.

При доступе к таблицам в БД говорят о привилегиях на выборку, добавление, удаление и изменение записей. Данная статья посвящена только доступу на чтение информации, так как при ограничении модификации данных используются иные механизмы (триггеры, ограничения представлений). Потому доступ на update, insert или delete не рассматривается. В рамках данного исследования присутствуют только 2 варианта доступа: доступ на чтение записи предоставлен или доступ на чтение записи не предоставлен.

Примем следующие упрощения.

1. Группы между собой не инкапсулированы. Это упрощение выполняется с целью увеличения скорости запросов к БД. Иерархическую структуру инкапсуляции ролей (групп) довольно сложно и трудоемко организовать на основе реляционной БД.

2. Доступ остальных пользователей, не входящих в группу, не описывается. В данном случае, по умолчанию, доступ другим пользователям не предоставлен.

3. Рассмотрим только запросы, вычисляющие количество записей, доступ к которым предоставлен пользователю. Это сделано для минимизации времени передачи информации от сервера к клиенту. При этом сложность вычислений остается прежней, а временем передачи одного числа от сервера к клиенту можно пренебречь.

Исследуем различные варианты организации такого доступа.

Введем служебные отношения пользователей *users* и групп *groups*:

$$\begin{aligned} users &= \{id, username\} \\ groups &= \{id, groupname\}. \end{aligned} \qquad (1)$$

Здесь *id* – числовой идентификатор пользователя или группы, *username* и *groupname* – символьные (текстовые) идентификаторы пользователя и группы соответственно.

Защищаемая таблица, доступ к записям которой необходимо ограничить, – *publication*, к которой в тексте будем обращаться как *r(R)*.

Рассмотренные в [2] реализации ограничения доступа могут быть оптимизированы различным образом [3]. Существует необходимость выбора оптимальных структур запросов.

Интерфейс пользовательского доступа к данным может быть реализован как посредством простого запроса, так и с помощью представлений пользователя в зависимости от потребностей системы и вычислительной мощности аппарат-

ных средств [4].

Символьные идентификаторы владельца и группы (роли). В исходное отношение *r* добавляются атрибуты *username* и *groupname*, определяющие имена владельца и группы записи.

Пользователи и группы связаны друг с другом как многие-ко-многим, так как один пользователь может находится в нескольких группах, а в одной группе содержится множество пользователей. Служебное отношение *users_groups*, выполняющее функцию связи многие-ко-многим будет иметь вид:

$$users\_groups = \{username, groupname\}. \quad (2)$$

Для реализации распределения доступа к защищаемому отношению *r* добавим атрибуты *username* и *groupname*, являющиеся символьными идентификаторами владельца и группы соответственно:

$$R' = \{A_1 \ldots A_n username groupname\}. \quad (3)$$

Здесь A_i – атрибуты защищаемого отношения *r(R)*.

В таком случае, запрос на выборку данных, доступных пользователю, примет следующий вид:

```
SELECT count(*)
FROM "publications"
WHERE "username" = 'pavel'
    OR "groupname" IN
    (SELECT "groupname" FROM "users_groups" WHERE "username" = 'pavel');
```

В запросе стоит текстовая константа 'pavel', которая замещает функцию определения имени пользователя. Такая замена связана с тем, что в различных СУБД функция, определяющая имя пользователя, имеет различное наименование. Запросы составлены таким образом, чтобы их можно было выполнить на различных СУБД.

Предполагаемые недостатки такого метода заключаются в длительной обработке текстовых полей при расчете выборки. Использование индексов для столбцов *usename* и *groupname* может сократить время выборки.

Другим вариантом извлечения пользовательской выборки является соединение таблиц с последующим удалением дублирующихся записей посредством distinct. Дублирование записей появляется при декартовом соединении защищаемого отношения и подзапроса пользовательских групп в результате того, что пользователь может сам быть владельцем записи, и одной из групп, в которые он включен, также будет предоставлен доступ к этой записи.

```
select count(*) from
    (
        SELECT distinct t.*
        FROM "publications" t, "people" s,
            ( SELECT ug."group_id"
          FROM "users_groups" ug, "people" p
WHERE ug."user_id" = p."id" and p."login"='pavel'
) u
        WHERE t."user_id" = s."id"
            and (
                (s."login" = 'pavel')
                or (t."group_id" = u."group_id")
            )
        ) t;
```

При больших объемах данных запрос с опцией distinct предположительно, должен выполняться довольно долго.

Числовые идентификаторы. Заменим имена владельца и группы на их числовые идентификаторы. При этом должна увеличиться скорость вычисления за счет отказа от длительных обработок текстовых данных. Отношение *users_groups* в таком случае тоже должно содержать числовые идентификаторы пользователя и группы. И его схема примет вид:

$$users\_groups = \{user\_id group\_id\}. \quad (1)$$

Схема защищаемого отношения примет вид:

$$R' = \{A_1 \ldots A_n user\_id group\_id\}. \quad (2)$$

При выполнении соединений и расчетов в подзапросах типа *WHERE <значение> IN*, возможно, будет сильно увеличена производительность запросов при индексировании идентификаторов пользователей и групп в защищаемом отношении.

В таком случае запрос на строки, доступные пользователю, будет выглядеть так:

```
SELECT count(*)
FROM "publications" t, "people" s
WHERE t."user_id" = s."id"
    and (
            (s."login" = 'pavel')
            or t."group_id" IN
                    ( SELECT "group_id"
                    FROM "users_groups" ug,
"people" p
                    WHERE ug."user_id" =
p."id" and p."login"='pavel' )
        );
```

В запросе реализовано декартово произведение двух таблиц: *publications* и *people*. СУБД из-за различий в оптимизиаторах запросов вычисляют такие запросы с различной эффективностью при использовании связи *join* в явном виде. Пользовательская выборка будет осуществляться следующим запросом:

```
SELECT count(*)
FROM "publications" t left join "people" s on(t."user_id" = s."id")
WHERE
    (s."login" = 'pavel')
    or t."group_id" IN
            ( SELECT "group_id"
            FROM "users_groups" ug, "people" p
            WHERE ug."user_id" = p."id" and p."login"='pavel' );
```

Отношение с привилегиями. Вариант с вынесением всей информации о доступе в отдельное отношение подходит для рассмотрения при выборе оптимального [2]. Отношение привилегий *permissions* будет иметь вид:

```
CREATE TABLE permissions (
   id integer NOT NULL,
   group_id integer,
   user_id integer,
);
```

Отношение *r′* будет дополнено атрибутом *permission_id*. Здесь при работе с полем в защищаемом отношении возможно увеличение производительности при использовании индекса.

```
SELECT count(*) from "publications" p
WHERE
p."permission_id" in (
SELECT pm."id" from "permissions" pm left join "people" pl on(pm."user_id" = pl."id")
WHERE
        pl."name"='pavel'
        or
        (pm."group_id" IN
                ( SELECT "group_id"
                FROM "users_groups" ug, "people" p
                WHERE ug."user_id" = p."id" and p."login"='pavel' )
        )
);
```

Доступ к разрешенным данным через введенное отношение можно организовать иными методами. Реализуем подзапрос, находящийся в секции *where,* через представление пользователя.

```
select count(*) from "publications" where "permission_id" in (select "id" from "user_permissions_direct");
```

Представление *user_permission_direct* можно реализовать как непосредственную выборку данных из *users_groups*:

```
CREATE VIEW user_permissions_direct AS
    SELECT pm.id, pm.group_id, pm.user_id, pm.group_can_delete, pm.tablename, pm.group_can_update
    FROM (permissions pm LEFT JOIN people pl ON ((pm.user_id = pl.id)))
    WHERE (((pl.name) = 'pavel') OR (pm.group_id IN
            (SELECT ug.group_id
            FROM users_groups ug, people p
            WHERE ((ug.user_id = p.id)
                    AND ((p.login)::text = = 'pavel')))));
```

Это же представление можно реализовать посредством использования еще одного представления *users_groups*, выбирающего все группы, в которые входит пользователь:

```
CREATE VIEW user_groups AS
    SELECT ug.group_id
    FROM users_groups ug, people pl
    WHERE ((ug.user_id = pl.id)
            AND ((pl.login) = 'pavel'));
```

Обозначим такое «косвенное» отношение как user_permissions:

```
CREATE VIEW user_permissions AS
    SELECT pm.id, pm.group_id, pm.user_id, pm.group_can_delete, pm.tablename, pm.group_can_update
    FROM (permissions pm LEFT JOIN people pl ON ((pm.user_id = pl.id)))
    WHERE (((pl.name)::text = 'pavel'::text)
            OR (pm.group_id IN (SELECT user_groups.group_id FROM user_groups)));
```

Тестовое отношение. Для тестирования производительности рассмотренных вариантов запросов добавим к защищаемому отношению следующие атрибуты:

– username — имя владельца записи;

– username2 — имя владельца записи (индексированное);

– groupname — имя группы записи;

– groupname2 — имя группы записи (индексированное);

– user_id — числовой идентификатор владельца записи;

– user_id2 — числовой идентификатор владельца записи (индексированный);

– group_id — числовой идентификатор группы записи (индексированный);

– group_id2 — числовой идентификатор группы записи (индексированный);

– permission_id — идентификатор привилегий записи;

– permission_id2 — идентификатор привиле-

гий записи (индексированный).

Согласно этим добавленным столбцам изменим указанные выше запросы.

Для тестирования защищаемая таблица *publications* была заполнена 618897 записями с произвольными значениями. Было проведено 354 запроса для каждой из СУБД для каждого запроса. Это позволяет существенно снизить риск ошибочной оценки времени выполнения запроса.

Для тестирования производительности использованы 3 различные СУБД: MySQL 5.0.67, PostgreSQL 8.3.5 и Oracle Database 10g Enterprise Edition Release 10.2.0.1.0.

Таблица пользователей содержит 1744 записи. Таблица групп содержит 29357 записей. Таблица групп пользователей содержит 1951 запись.

Оценка производительности проведена в автоматическом режиме, с запуском всех запросов во всех СУБД поочередно. Данные по замерам длительности выполнения запросов внесены в СУБД PostgreSQL для статистической обработки. В результате получены следующие данные, представленные в таблице.

Столбец «описание запроса» словесно описывает один из приведенных выше запросов. На пересечении строки каждого запроса со столбцом СУБД выставлено время выполнения запроса, измеренное в секундах. Описания запросов по обозначениям можно посмотреть в таблице.

Заключение. Проанализировав полученные в результате тестирования производительности данные, можно сделать следующие выводы.

1. Для всех СУБД наиболее приемлемым является выделение привилегий в отдельное отношение.

2. При этом наличие индекса по внешнему ключу защищаемой таблицы необходимо для mySQL, так как это улучшает производительность примерно в 7 раз.

3. При реализации доступа с хранением числовых идентификаторов владельца и группы в защищаемом отношении использование *join* незначительно замедляет трудоемкость выборки в mySQL и Oracle.

4. Использование доступа с хранением числовых идентификаторов владельца и группы является менее предпочтительным по сравнению с идентификатором привилегии, однако он также может быть использован, хотя снижает производительность примерно в 1.2 в PostgreSQL. Нежелательно использование на mySQL и Oracle, так как дает снижение производительности около 3-х раз.

5. Использование запросов с distinct нерационально на всех 3-х СУБД и показывает самые низкие результаты.

Таблица – Результаты тестирования производительности

Описание запроса	My-SQL	Postgre-SQL	Oracle
Запрос без ограничения доступа	0,19	0,72	0,05
Выборка по числовым идентификаторам пользователей с индексированием	40,26	27,81	3,18
Выборка по числовым идентификаторам пользователей без индексирования	12,07	27,83	3,19
Выборка по числовым идентификаторам в подзапросе в WHERE с индексированием	3,08	1,23	1,28
Выборка по числовым идентификаторам в подзапросе в WHERE с индексированием и join	3,94	1,24	1,34
Выборка по числовым идентификаторам в подзапросе в WHERE без индексирования, с использованием join	4,44	1,24	1,33
Выборка по числовым идентификаторам в подзапросе в WHERE без индексирования	4,44	1,24	1,3
Запрос через permissions непосредственно, без представлений, с идексированием	1,48	0,92	0,45
Запрос через представление user_permissions	8,21	0,92	0,46
Запрос через представление user_permissions_direct	8,17	0,92	0,46
Запрос через представление user_permissions_direct с индексированием	1,53	0,92	0,47
Запрос через представление user_permissions с индексированием	1,53	0,93	0,46
Выборка по тестовым идентификаторам пользователей с индексированием	24,34	0,97	3,72
Выборка по текстовым полям без индексации	9,54	0,95	3,65

Библиографический список:

1. *Мейер Д.* Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.

2. *Баранчиков А.И., Баранчиков П.А.* Организация доступа к записям таблиц БД по аналогии с

POSIX-совместимыми файловыми системами / Проблемы передачи и обработки информации в сетях и системах телекоммуникаций: материалы 15-й международной науч.-техн. конф., часть 1. – Рязань: Рязанский государственный радиотехнический университет, 2008. – 148 с.

3. *Chaudhuri Surajit, Shim Kyuseok.* Optimization of queries with user-defined predicates. ACM Trans. Database Syst. 2, 1999. Т. 24. С. 177–228.

4. *Баранчиков А.И., Баранчиков П.А.* Организация доступа к записям таблиц в базах данных // Вестник РГРТА. Вып. 20. – Рязань, 2007.