# МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

DOI 10.17223/20710410/24/5

УДК 004.94

## АНАЛИЗ УСЛОВИЙ ПРЕДОСТАВЛЕНИЯ И ПОЛУЧЕНИЯ ПРАВ ДОСТУПА В МОДЕЛИ УПРАВЛЕНИЯ ДОСТУПОМ MS SQL SERVER

В. Ю. Смольянинов

*Учебно-методическое объединение по информационной безопасности, г. Москва, Россия*

**E-mail:** VladimirSmall@gmail.com

Рассматривается модель управления доступом MS SQL Server, построенная на основе СУБД ДП-модели. Для учёта особенностей управления доступом в СУБД MS SQL Server 2012 в модель добавлены роли, права доступа к учётным записям пользователей и ролям, цепочки владения, а также возможность олицетворения пользователей и активации триггеров и процедур от имени заданных учётных записей пользователей. Доказывается утверждение об эквивалентности возможности выполнения произвольного SQL-кода от имени заданной учётной записи и возможности получения к ней права олицетворения. Обосновываются необходимые и достаточные условия получения и предоставления прав доступа к сущностям при отсутствии кооперации между субъект-сессиями.

**Ключевые слова:** *компьютерная безопасность, модель управления доступом MS SQL Server, система управления базами данных.*

### Введение

Современные реляционные системы управления базами данных (СУБД) зарекомендовали себя как надёжное и эффективное решение задач поиска, хранения и обработки информации. Разработчики автоматизированных информационных систем используют СУБД не только для решения этих задач, но и для защиты информации и, в частности, для управления доступом к обрабатываемым данным. Существующие формальные модели логического управления доступом [1, 2] ориентированы в основном на применение в ОС и не учитывают некоторых существенных особенностей функционирования автоматизированных информационных систем, построенных с использованием реляционных СУБД, которые могут быть использованы нарушителем для реализации запрещённых доступов и информационных потоков. Таким образом, теоретический анализ условий, при которых возможно несанкционированное распространение прав доступа в автоматизированных информационных системах, использующих штатные механизмы управления доступом СУБД, является важной и актуальной задачей.

В [3] предпринята попытка построения СУБД ДП-модели, в которой учитывается наследование прав доступа к дочерним сущностям, существование прав на предоставление прав доступа, а также возможность автоматического выполнения SQL-кода при реализации доступа к данным и изменения учётной записи пользователя при активации хранимых процедур и триггеров. Тем не менее данная модель не учитывает

в полной мере некоторых существенных особенностей организации управления доступом в новой версии СУБД MS SQL Server 2012. В частности, игнорируются роли, наличие прав доступа к учётным записям пользователей и ролям, цепочки владения, возможность олицетворения и выполнения триггеров и процедур от имени заданных учётных записей.

В настоящей работе на базе СУБД ДП-модели [3] построена модель управления доступом MS SQL Server, в которой учитываются перечисленные особенности управления доступом и проведено необходимое уточнение правил преобразования состояний. Описываются элементы модели, предназначенной для анализа безопасности управления доступом в СУБД MS SQL Server. Построенная модель может быть в дальнейшем адаптирована для анализа безопасности других реляционных СУБД.

## 1. Определения и обозначения

В предлагаемой модели управления доступом *MS SQL Server* структура множества сущностей задаётся следующим образом:

- $C$ — множество контейнеров;
- $O_t$ — множество триггеров;
- $O_p$ — множество процедур;
- $U$ — множество учётных записей пользователей;
- $R$ — множество ролей;
- $P = U \cup R$ — множество принципалов, при этом считается, что $U \cap R = \varnothing$;
- $E = C \cup O_p \cup P$ — множество сущностей, при этом считается, что множества процедур, триггеров, контейнеров и принципалов попарно не пересекаются.

Множество контейнеров реальных реляционных СУБД включает такие сущности, как экземпляр СУБД, экземпляры баз данных, схемы и таблицы. В рамках модели не рассматривается управление доступом к отдельным записям таблиц и поэтому в модель не включено множество записей. Причиной этому служит отсутствие в СУБД MS SQL Server штатных механизмов контроля доступа на уровне отдельных записей. Кроме того, согласно [4], присутствующие в СУБД MS SQL Server 2012 штатные механизмы управления доступом к отдельным полям таблиц не рекомендованы к использованию и, возможно, будут исключены в будущем.

Элементы множества триггеров соответствуют триггерам таблиц в СУБД, которые автоматически срабатывают при осуществлении операций вставки, удаления или обновления записей таблиц, с которыми они связаны. Элементы множества процедур соответствуют хранимым процедурам СУБД, выполнение которых может быть инициировано в рамках взаимодействия с системой.

Считается, что триггеры и процедуры содержат SQL-код, выполнение которого приводит к реализации преобразований системы, задаваемых правилами, описываемыми в п. 2. В реальных системах процедуры могут принимать формальные параметры. При вызове хранимой процедуры ей передаются фактические значения формальных параметров, что приводит к выполнению некоторой фиксированной последовательности операторов. В рамках модели считается, что каждой хранимой процедуре, принимающей входные параметры, соответствует множество процедур. При этом каждой допустимой комбинации значений фактических параметров соответствует отдельная процедура, содержащая SQL-код, являющийся результатом их подстановки в операторы хранимой процедуры. Если внутри хранимой процедуры имеются операторы вызова других хранимых процедур, принимающих параметры, то каждый такой вызов

заменяется оператором активации соответствующей процедуры, полученной подстановкой фактических параметров.

Большинство современных СУБД реализуют, как минимум, дискреционное управление доступом учётных записей пользователей к контейнерам и процедурам. В СУБД MS SQL Server 2012 пользователи представлены на двух уровнях [4]: в виде логинов (`logins`) на уровне экземпляра СУБД и в виде пользователей (`users`) на уровне экземпляра базы данных. Для логинов могут быть заданы права к экземпляру СУБД, а для пользователей — права доступа к сущностям, находящимся в базе данных. Каждому пользователю базы данных соответствует единственный логин. В рамках модели различия между логинами и пользователями игнорируются и считается, что каждый пользователь имеет уникальную учётную запись.

Для повышения гибкости управления правами доступа СУБД MS SQL Server реализует элементы ролевого управления доступом. Роли являются совокупностью прав доступа к сущностям. В СУБД MS SQL Server 2012 роли присутствуют на двух уровнях [4]: на уровне экземпляра СУБД (серверные роли, `server roles`) и на уровне базы данных (роли базы данных, `database roles`). В рамках модели различия между серверными ролями и ролями базы данных игнорируются. В СУБД роли бывают двух видов: фиксированные роли (`fixed roles`) и гибкие роли (`flexible roles`). Отличие фиксированных ролей от гибких состоит в том, что у последних можно управлять не только членством в роли, но и совокупностью предоставленных ей прав доступа к сущностям. Для управления авторизацией на фиксированную роль требуется наличие авторизации на неё у учётной записи инициировавшего операцию пользователя. В рамках модели игнорируется отличие между фиксированными и гибкими ролями.

Для упрощения процедуры администрирования в СУБД поддерживается управление правами доступа с учётом иерархических отношений между сущностями, а также выполняется неявное предоставление всех возможных прав доступа владельцам сущностей. В СУБД обладать правами доступа к сущностям и быть их владельцами могут не только учётные записи пользователей, но и роли. Таким образом, множество принципалов $P$ задаёт участников безопасности, которым могут быть предоставлены права доступа к сущностям. В СУБД права доступа могут быть заданы не только в отношении контейнеров и процедур, но и в отношении самих принципалов, и, следовательно, множество сущностей задаётся как $E = C \cup O_p \cup P$. Заметим, что в рамках модели триггеры не рассматриваются как сущности, так как в СУБД MS SQL Server отсутствуют механизмы управления доступом к ним и, значит, они не являются объектами доступа.

Обозначим $E_H = E \cup O_t$, элементы которого будем называть сущностями (технически это не совсем корректно, так как триггеры не считаются сущностями; $E \cap O_t = \varnothing$). В дальнейшем будем использовать термин «сущность» как в широком смысле (в отношении элементов множества $E_H$), так и в узком (в отношении элементов множества $E$); использование термина будет понятно из контекста. Тогда принципал владельца сущности задаётся с помощью функции $owner\colon E_H \to P$. По определению будем считать, что владельцем учётной записи является она сама, то есть для каждого $u \in U$ верно $owner(u) = u$.

Заметим, что в СУБД MS SQL Server триггеры не имеют владельцев. Тем не менее в рамках модели удобно считать, что у любого триггера есть владелец, совпадающий с владельцем таблицы, к которой он относится. У каждой сущности задан единственный владелец, который определяется в момент её создания. Для простоты в модели игнорируется присутствующая в СУБД возможность задания принципала владельца

сущности с использованием SQL-выражения *ALTER AUTHORIZATION ON* (для баз данных, схем, таблиц и процедур), и поэтому выполняется следующее предположение.

**Предположение 1.** Для любой сущности $e$ учётная запись пользователя владельца $owner(e)$ после создания сущности $e$ не изменяется.

В СУБД MS SQL Server владельцами сущностей могут являться не только учётные записи пользователей, но и роли. Для учёта данной особенности функционирования СУБД областью значений функции *owner* является множество принципалов $P$.

Функция *owner* введена в модель в связи с тем, что в современных СУБД владелец сущности получает к ней все права доступа. Если же сущность является контейнером, то владелец неявно получает все возможные права доступа не только к самому контейнеру, но и ко всем сущностям, которые в нём содержатся.

Используем также следующие обозначения и определения:

- $S$ — множество субъект-сессий пользователей, при этом считается, что $S \cap E = \varnothing$;
- $user\colon S \to U$ — функция, задающая для каждой субъект-сессии учётную запись пользователя, от имени которой она выполняется;
- $U^*$ — множество всех конечных последовательностей учётных записей пользователей;
- $user\_stack\colon S \to U^*$ — функция, задающая для каждой субъект-сессии последовательность учётных записей пользователей, от имени которых она выполнялась.

Субъект-сессии соответствуют сессиям пользователей с СУБД, которые, в отличие от процессов ОС, не являются объектами доступа и потому не считаются сущностями. Субъект-сессии инициируют выполнение SQL-кода, что приводит к реализации заданных в нём преобразований состояний системы.

Субъект-сессии применяют правила преобразования состояний системы от имени учётных записей пользователей из множества $U$. В реальных СУБД для создания новой сессии требуется указать в строке подключения данные, аутентифицирующие подключающегося пользователя. Изначально субъект-сессия начинает применять правила от имени учётной записи пользователя, инициировавшей её создание. Однако при переходе системы из состояния в состояние допускается изменение значений функции *user*, что связано с наличием в реальных СУБД возможности выполнения кода хранимых процедур и триггеров от имени учётной записи, не являющейся инициатором их выполнения. Отметим, что возможность изменения значений функции *user* является существенным отличием от моделей ролевого управления доступом и ДП-моделей, в которых предполагается неизменность её значений.

В СУБД пользователь имеет возможность с применением оператора $REVERT$ восстановить учётную запись пользователя, от имени которой субъект-сессия выполнялась до момента предыдущего изменения контекста выполнения. Функция $user\_stack$ добавлена в модель для обеспечения возможности восстановления субъект-сессией $s$ всех предыдущих значений функции $user(s)$.

Функция $cmode\colon C \to \{\texttt{creator}, \texttt{parent}\}$ задаёт для каждого контейнера режим установки владельца вновь создаваемых дочерних сущностей. Пусть для некоторого контейнера $c \in C$ в рамках сессии $s \in S$ было инициировано создание дочерней сущности контейнера $c$. Тогда, если значение $cmode(c) = \texttt{creator}$, то владельцем вновь созданной дочерней сущности становится учётная запись пользователя $user(s)$. В противном случае, если значение $cmode(c) = \texttt{parent}$, то владельцем вновь созданной дочерней сущности становится принципал $owner(c)$.

Режим установки владельца определяется в момент создания контейнера и не изменяется при выполнении преобразований системы, а значит, верно предположение 2.

**Предположение 2.** Для любого контейнера $c \in C$ режим установки владельца вновь создаваемых дочерних сущностей $cmode(c)$ после его создания не изменяется.

Необходимость во введении функции *cmode* связана с тем, что в СУБД MS SQL Server владельцем таблиц и процедур является владелец схемы, которой они принадлежат. С другой стороны, владельцем базы данных становится учётная запись создавшего её пользователя.

Для учёта возможности изменения учётной записи пользователя при выполнении триггеров и процедур введена функция $execute\_as\colon O_p \cup O_t \to U \cup \{\texttt{as\_caller}\}$, значение которой задаёт режим выполнения процедур и триггеров, а также определяет учётную запись пользователя, от имени которой субъект-сессии будут выполнять их SQL-код. Отметим, что случай $execute\_as(e) = u$, где $u = owner(e)$ и $u \in U$, соответствует механизму повышения полномочий $SUID$ в ОС семейства UNIX, а случай $execute\_as(e) = \texttt{as\_caller}$ — механизму имперсонации (олицетворения) в ОС семейства Windows. Если некоторая субъект-сессия $s \in S$ инициировала выполнение процедуры или триггера $e \in O_p \cup O_t$ и $execute\_as(e) = \texttt{as\_caller}$, то код сущности $e$ будет выполняться от имени учётной записи пользователя $user(s)$, от имени которой $s$ выполнялась непосредственно до запуска SQL-кода сущности $e$. В противном случае, если $execute\_as(e) = u$, где $u \in U$, то код сущности $e$ будет выполняться субъект-сессией $s$ от имени учётной записи пользователя $u$.

Заметим, что в модели не рассматриваются два присутствующих в СУБД режима выполнения триггеров и процедур: от имени владельца (*AS OWNER*) и от имени учётной записи пользователя, последним изменившим их SQL-код (*AS SELF*). Оба этих режима могут рассматриваться как частные случаи выполнения SQL-кода процедур и триггеров от имени заданной учётной записи пользователя. Заметим, что в СУБД режим выполнения от имени владельца может быть задан только для триггеров и процедур, владельцем которых являются учётные записи, а не роли.

Выше уже отмечалось, что СУБД MS SQL Server поддерживает базовые механизмы ролевого управления доступом, для учёта которых в модель добавлен ряд элементов, в частности функция $UA\colon U \to 2^R$ — функция авторизованных ролей учётных записей пользователей, задающая для каждой учётной записи пользователя множество ролей, на которые она может быть авторизована.

В дальнейшем будем считать, что выполняется следующее предположение.

**Предположение 3.** Для каждой роли $r \in R$ существует учётная запись пользователя $u$, такая, что $r \in UA(u)$. Таким образом, для каждой роли существуют авторизованные на неё пользователи.

Будем считать, что существует выделенная роль $public \in R$, такая, что для каждого $u \in U$ верно $public \in UA(u)$, то есть на роль *public* авторизованы учётные записи всех пользователей.

В СУБД MS SQL Server учётные записи пользователей нельзя лишить авторизации на роль *public*. Роль *public* считается фиксированной, однако в отличие от других фиксированных ролей существует возможность управлять совокупностью предоставленных ей прав доступа. Именно поэтому в рамках модели данная роль считается гибкой и отнесена с указанными выше ограничениями к множеству ролей $R$.

Будем считать, что на множестве ролей $R$ задано отношение частичного порядка, которое удовлетворяет следующему определению.

**Определение 1.** Иерархией ролей называется заданное на множестве ролей $R$ отношение частичного порядка $\leqslant$. При этом по определению выполняется: если для пользователя $u \in U$ роли $r, r' \in R$ такие, что $r \in UA(u)$ и $r' \leqslant r$, то $r' \in UA(u)$.

В случае, когда для двух ролей $r_1, r_2 \in R$ выполняются условия $r_1 \leqslant r_2$ и $r_1 \neq r_2$, будем использовать обозначение $r_1 < r_2$.

Будем считать, что существует выделенная роль $sysadmin \in R$, такая, что для каждой роли $r \in R$ выполняется $r \leqslant sysadmin$.

В СУБД MS SQL Server фиксированная роль $sysadmin$ считается ролью системных администраторов. Пользователи, авторизованные на данную роль, могут выполнять произвольные действия с любой сущностью СУБД.

В рамках модели считается, что субъект-сессия, выполняемая от имени некоторой учётной записи пользователя, авторизована на все её роли. Следовательно, можно считать, что учётная запись пользователя обладает правами доступа всех ролей, на которые она может быть авторизована. Так как на множестве ролей задана иерархия, учётная запись пользователя обладает правами доступа всех ролей, «иерархически подчиненных» роли, на которую она может быть авторизована.

В СУБД MS SQL Server иерархия на множествах гибких и фиксированных ролей задаётся путём предоставления «членства в роли»: считается, что если роль $r_2$ является членом роли $r_1$, то $r_1 \leqslant r_2$. Отметим, что авторизация пользователей на роль также осуществляется путём предоставления пользователю «членства в роли». Рассмотрим следующий пример. Пусть роль $Hackers$ является членом роли $Users$ и роли $Users$ предоставлено право доступа на чтение к таблице $Table$, а роли $Hackers$ предоставлено к ней право доступа на запись. Тогда если пользователь $Alice$ является членом группы $Users$, то ей предоставлено право доступа на чтение к таблице $Table$, в то время как члену группы $Hackers$ пользователю $Bob$ предоставлено право доступа к таблице $Table$ как на чтение, так и на запись.

В современных СУБД на множестве контейнеров неявно задаётся иерархия сущностей. Как правило, с экземпляром СУБД связаны экземпляры баз данных, которые, в свою очередь, содержат схемы данных, являющиеся контейнерами для хранимых процедур, а также таблиц вместе с содержащимися в них записями и триггерами. Для задания иерархических отношений сущностей введём следующие определения.

**Определение 2.** Определим $H\colon C \to 2^{E_H}$ — функцию иерархии сущностей, сопоставляющую каждому контейнеру $c \in C$ множество $H(c) \subset E_H$ и удовлетворяющую следующим условиям.

У с л о в и е 1. Пусть $Pr(e) = \{c \in C : e \in H(c)\}$ — множество родительских контейнеров сущности $e$. Тогда существует единственный корневой контейнер $c_r \in C$, такой, что $|Pr(c_r)| = 0$, $owner(c_r) = sysadmin$ и для каждой сущности $e \in E_H \setminus \{c_r\}$ выполняется $|Pr(e)| = 1$.

У с л о в и е 2. Для каждой сущности $e \in E_H$ либо $e = c_r$, либо существуют контейнеры $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, такие, что $c_1 = c_r$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $e \in H(c_n)$, при этом если $e \in P$, то $n = 1$.

У с л о в и е 3. Пусть $T = \{c \in C : \exists o_t \in O_t\,(o_t \in H(c))\}$ — множество таблиц. Тогда для каждой таблицы $t \in T$ по определению выполняется $H(t) \subseteq O_t$ и для каждого $o_t \in H(t)$ верно $owner(o_t) = owner(t)$.

У с л о в и е 4. Для каждой сущности $e \in O_p \cup T$ выполняется равенство $cmode(Pr(e)) = \texttt{parent}$.

**Определение 3.** Если для сущности $e \in E_H$ существует контейнер $c \in C$, такой, что $e \in H(c)$, то будем говорить, что контейнер $c$ содержит сущность $e$, а сущность $e$ содержится в контейнере $c$; также будем говорить, что $c$ является родительским контейнером сущности $e$, а $e$ — дочерней сущностью контейнера $c$.

Будем считать, что единственный корневой контейнер $c_r$, у которого отсутствуют родительские контейнеры, соответствует экземпляру СУБД. Владельцем корневого контейнера $c_r$ является роль *sysadmin*.

Согласно условию 2 определения 2, родительским контейнером всех принципалов является корневой контейнер $c_r$. В реальных СУБД роли не считаются частью иерархии сущностей, тем не менее владелец экземпляра СУБД обладает всеми правами доступа к принципалам. Следовательно, удобно считать, что все принципалы являются дочерними сущностями $c_r$.

Помимо корневого контейнера $c_r$, в рамках модели выделяется особый вид контейнеров — таблицы. Согласно условию 3 определения 2, для каждой таблицы $t \in T$ выполняется $H(t) \subseteq O_t$, из чего следует, что таблицы не могут иметь дочерних контейнеров и содержат только триггеры. В реальных СУБД триггеры связаны с таблицами, а не содержатся в них, при этом наличие права доступа к таблице на изменение её способа задания (*ALTER*) позволяет также изменять способ задание её триггеров. Следовательно, в рамках модели удобно считать триггеры частью иерархии и дочерними сущностями таблиц.

Выше отмечалось, что в СУБД MS SQL Server владельцем таблиц и процедур является владелец схемы, в которой они содержатся. Для того чтобы модель соответствовала данному ограничению, в определение 2 введено условие 4: режим установки владельца для контейнеров, содержащих таблицы и процедуры, должен быть равен `parent`.

Наконец отметим, что в отличие от реальных СУБД модель не накладывает ограничений на число уровней вложенности в иерархии.

В СУБД принципалы, имеющие права доступа к контейнерам, получают также права ко всем сущностям, расположенным ниже по иерархии. Для учёта этой особенности управления доступом введём определение отношения иерархической подчинённости сущностей.

**Определение 4.** Будем говорить, что сущность $e \in E_H$ иерархически подчинена контейнеру $c \in C$, если существуют контейнеры $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, такие, что $c_1 = c$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $e \in H(c_n)$.

В случае, если сущность $e \in E_H$ иерархически подчинена контейнеру $c \in C$, будем использовать обозначение $e < c$. В случае, если сущность $e \in E_H$ иерархически подчинена контейнеру $c \in C$ или равна ему, будем использовать обозначение $e \leqslant c$.

В зависимости от контекста будет понятно, используется ли «$\leqslant$» для обозначения иерархической подчинённости роли контейнеру или иерархических отношений на множестве ролей.

**Замечание 1.** Из определений 2 и 4 следует, что для любой сущности $e \in E_H \setminus \{c_r\}$ существует контейнер $c \in C$, такой, что $e < c$. Очевидно, что каждая сущность либо является корневой, либо иерархически подчинена корневой сущности $c_r$ и содержится ровно в одном из её дочерних контейнеров.

Итак, для каждой сущности, за исключением корневой, существует единственный контейнер, в котором она содержится. Кроме того, в соответствии с условием 3 определения 2 в множество таблиц включаются все контейнеры, содержащие хотя бы один

триггер. Следовательно, для каждого триггера $o_t \in O_t$ существует ровно одна таблица $t \in T$, в которой он содержится, то есть $o_t \in H(t)$.

**Замечание 2.** Легко показать, что если для некоторой сущности $e \in E_H$ существуют контейнеры $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, такие, что $c_1 = c_r$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $e \in H(c_n)$, и контейнеры $c'_1, \ldots, c'_{n'} \in C$, где $n' \geqslant 1$, такие, что $c'_1 = c_r$, $c'_{i+1} \in H(c'_i)$ для $i = 1, \ldots, n'-1$ и $e \in H(c'_{n'})$, то $n = n'$ и $c'_i = c_i$ для $i = 1, \ldots, n$. Это означает, что для любой некорневой сущности $e \in E_H$ существует единственная последовательность, начинающаяся с $c_r$ и заканчивающаяся контейнером, содержащим $e$, такая, что каждый её элемент является родительским для последующего.

**Определение 5.** Функция $holders\colon E_H \to 2^P$ задаёт для каждой сущности принципалы её иерархических владельцев, при этом для каждой сущности $e \in E_H$ по определению $p \in holders(e)$ тогда и только тогда, когда либо $p = owner(e)$, либо существует контейнер $c \in Pr(e)$, такой, что $p \in holders(c)$.

Принципалы иерархических владельцев сущности обладают к ней всеми возможными правами доступа. Заметим, что значения функции $holders$ однозначно определяются функцией иерархии сущностей $H$ и функцией задания владельцев сущностей $owner$, и поэтому она не рассматривается в качестве самостоятельного элемента состояния системы.

Введём множество видов прав доступа к сущностям $R_r$, которые соответствуют по смыслу и назначению традиционным для СУБД разрешениям *SELECT*, *UPDATE*, *INSERT*, *DELETE*, *ALTER*, *EXECUTE* и *IMPERSONATE*:

$$R_r = \{select_r, insert_r, update_r, delete_r, alter_r, execute_r, impersonate_r\}.$$

Считается, что наличие у учётной записи пользователя прав доступа $select_r$, $insert_r$, $update_r$ или $delete_r$ к таблице позволяет реализовать к ней соответствующее право доступа (то есть выполнить операции выборки, вставки, обновления и удаления соответственно). Обладание правом доступа $alter_r$ к процедуре позволяет задавать её SQL-код и режим выполнения. Право доступа $alter_r$ к таблице позволяет создавать в ней триггеры, а также изменять SQL-код связанных с ней триггеров и режим их выполнения. Наличие права доступа $execute_r$ к процедуре позволяет выполнять её SQL-код. Наличие права доступа $impersonate_r$ к учётной записи пользователя позволяет инициировать выполнение SQL-кода от её имени. Заметим, что право доступа $impersonate_r$ неприменимо к ролям, и поэтому область значений функции $execute\_as$ задана как $U \cup \{\texttt{as\_caller}\}$, а не как $P \cup \{\texttt{as\_caller}\}$. Кроме того, наличие права доступа $alter_r$ к процедуре $o_p \in O_p$ и $impersonate_r$ к учётной записи $u \in U$ позволяет установить режим выполнения процедур $o_p$ от имени учётной записи пользователя $u$. Аналогично, для задания режима выполнения триггеров таблицы $t \in T$ от имени учётной записи пользователя $u \in U$ требуется наличие права доступа $alter_r$ к таблице, а также права доступа $impersonate_r$ к учётной записи пользователя $u$.

В СУБД MS SQL Server существует разрешение *TAKE OWNERSHIP*, позволяющее стать владельцем сущности. По умолчанию, разрешение *TAKE OWNERSHIP* к сущности может быть предоставлено только её владельцем и администраторами СУБД. В рамках модели возможность изменения владельцев сущностей игнорируется.

Введём $R_d \subseteq P \times E \times R_r$ — множество непосредственно заданных прав доступа принципалов к сущностям. Права доступа к сущностям добавляются в множество $R_d$ в результате явного применения правил преобразования системы, а не в результате обладания ролью или правами доступа к контейнеру, которому иерархически подчинена сущность.

Отметим, что способ задания множества $R_d$ учитывает, что большинство современных СУБД предоставляют возможность устанавливать права доступа к контейнерам и процедурам, не позволяя определять их на уровне записей таблиц и триггеров.

Введём $R_{own} = \{(p, e, \alpha_r) : e \in E, p = owner(e), \alpha_r \in R_r\}$ — множество прав доступа принципалов владельцев к сущностям. В соответствии со способом задания $R_{own}$ принципалы владельцев сущностей обладают к ним всеми правами доступа.

Для учёта того, что принципалы, имеющие право доступа к контейнеру, также обладают им ко всем его иерархически подчинённым сущностям, вводится множество иерархических прав доступа принципалов к сущностям:

$$R_H = \{(p, e, \alpha_r) : p \in P, e \in E, \alpha_r \in R_r, \exists c \in C((p, c, \alpha_r) \in R_d \cup R_{own} \,\&\, e < c)\}.$$

Обозначим $PR = R_d \cup R_{own} \cup R_H$. Введём $R_e$ — множество действующих прав доступа с учётом того, что на множестве ролей задана иерархия и каждая субъект-сессия авторизована на все роли, на которые авторизована учётная запись пользователя, от имени которой она выполняется. Заметим, что для удобства множество $R_e$ задано таким образом, чтобы роли имели права доступа всех иерархически подчинённых им ролей:

$$\begin{gathered} R_e = \{(u, e, \alpha_r) : u \in U, e \in E, \alpha_r \in R_r((u, e, \alpha_r) \in PR)\} \cup \\ \cup\{(u, e, \alpha_r) : u \in U, e \in E, \alpha_r \in R_r, \exists r \in R((r, e, \alpha_r) \in PR \,\&\, r \in UA(u))\} \cup \\ \cup\{(r, e, \alpha_r) : r \in R, e \in E, \alpha_r \in R, \exists r' \in R((r', e, \alpha_r) \in PR, r' \leqslant r)\}. \end{gathered}$$

При определении возможности применения правил преобразования состояний учитываются права доступа, содержащиеся в множестве $R_e$. Полагается, что субъект-сессии $s$, функционирующей от имени учётной записи пользователя $u = user(s)$, разрешено реализовать доступ вида $\alpha_r$ к сущности $e$ в случае, если $(u, e, \alpha_r) \in R_e$, то есть у учётной записи пользователя $u$ есть действующее право доступа $\alpha_r$ к сущности $e$ (далее в этом случае будем говорить о наличии права доступа).

Отметим, что множество $R_e$ включает права доступа учётных записей пользователей с учётом ролей, на которые они авторизованы.

Очевидно, что из способа задания множеств $R_H$ и $R_e$ следует

**Замечание 3.** Если $u \in U$, $e, e' \in C \cup O_p$, $\alpha_r \in R_r$, $e \leqslant e'$ и $(u, e', \alpha_r) \in R_e$, то $(u, e, \alpha_r) \in R_e$.

Множества прав доступа $R_{own}$, $R_H$ и $R_e$ однозначно определяются множеством непосредственно заданных прав доступа $R_d$, функцией иерархии сущностей $H$, функцией авторизованных ролей пользователей $UA$, а также функцией задания владельцев сущностей *owner* и поэтому не рассматриваются в качестве самостоятельных элементов состояния системы.

Как правило, в ОС, реализующих механизмы дискреционного управления доступом, назначать права доступа к сущности может только её владелец и, возможно, пользователь, обладающий привилегиями администратора. Указанные пользователи могут предоставить произвольное право доступа к сущности любому другому пользователю системы. В то же время в современных СУБД присутствуют штатные механизмы, позволяющие делегировать предоставление прав доступа к сущности заданному для неё кругу учётных записей пользователей. При этом СУБД позволяют ограничить для каждой учётной записи из этого круга набор прав доступа к сущности, которые она может предоставлять.

Обозначим:

— $Gr_d \subseteq P \times E \times R_r$ — множество непосредственно заданных прав принципалов на предоставление прав доступа к сущностям, удовлетворяющее условию $Gr_d \subseteq R_d$;

— $Gr_{hld} = \{(p, e, \alpha_r) : \alpha_r \in R_r, e \in E, p \in holders(e)\}$ — множество прав доступа принципалов на предоставление прав доступа иерархическими владельцами сущностей;

— $PGr = Gr_d \cup Gr_{hld}$. Тогда, с учётом того, что учётная запись пользователя обладает правами всех ролей, на которые она может быть авторизована, а также с учётом того, что роль обладает правами всех иерархически подчинённых ей ролей, введём множество действующих прав доступа учётных записей пользователей на предоставление прав доступа к сущностям:

$$Gr_e = \{(u, e, \alpha_r) : u \in U, e \in E, \alpha_r \in R_r((u, e, \alpha_r) \in PGr)\} \cup$$
$$\cup\{(u, e, \alpha_r) : u \in U, e \in E, \alpha_r \in R_r, \exists r \in R((r, e, \alpha_r) \in PGr \,\&\, r \in UA(u))\} \cup$$
$$\cup\{(r, e, \alpha_r) : r \in R, e \in E, \alpha_r \in R, \exists r' \in R((r', e, \alpha_r) \in PGr, r' \leqslant r)\}.$$

Множество $Gr_e$ включает права доступа на предоставление прав доступа учётных записей пользователей, с учётом ролей, на которые они могут быть авторизованы. Заметим, что для удобства множество $Gr_e$ задано таким образом, чтобы роли имели возможность предоставлять права доступа к сущностям, к которым могут предоставлять права их иерархически подчинённые роли.

Множества $Gr_{hld}$ и $Gr_e$ однозначно определяются множеством непосредственно заданных прав доступа учётных записей пользователей на предоставление прав доступа к сущностям $Gr_d$, функцией иерархии сущностей $H$, функцией авторизованных ролей пользователей $UA$, а также функцией задания владельцев сущностей *owner* и поэтому не рассматриваются в качестве самостоятельных элементов состояния системы.

Для удобства введём обозначение $Gr(e, \alpha_r) = \{p \in P : (p, e, \alpha_r) \in Gr_e\}$ — множество принципалов, которые могут предоставлять право доступа $\alpha_r \in R_r$ к сущности $e \in E$. В рамках модели полагается, что субъект-сессии $s$, функционирующей от имени учётной записи пользователя $u = user(s)$, разрешено предоставить право доступа вида $\alpha_r$ к сущности $e$, если $u \in Gr(e, \alpha_r)$, то есть учётная запись пользователя $u$ обладает действующим правом доступа на предоставление права доступа $\alpha_r$ к сущности $e$ (далее в этом случае будем говорить о наличии права доступа на предоставление права доступа).

В современных СУБД для того чтобы пользователь имел возможность предоставить право доступа к сущности, требуется, чтобы он сам обладал к ней этим правом доступа. Обоснуем, что в рамках модели необходимым условием наличия у учётной записи пользователя $u \in U$ права на предоставление права доступа $\alpha_r$ к сущности $e \in E$ является наличие у $u$ права доступа $\alpha_r$ к $e$.

**Утверждение 1.** $Gr_e \subseteq R_e$.

***Доказательство.*** Заметим, что, согласно способу задания множеств $R_e$ и $Gr_e$, достаточно показать, что $PGr \subseteq PR$. По способу задания $PGr = Gr_d \cup Gr_{hld}$ и $Gr_d \subseteq R_d \subseteq PR$. Покажем, что $Gr_{hld} \subseteq PR$. Пусть $(p, e, \alpha_r) \in Gr_{hld}$, где $e \in E$, $\alpha_r \in R_r$ и $p \in holders(e)$. В случае $p = owner(e)$, согласно способу задания множества $R_{own}$, выполняется $(p, e, \alpha_r) \in R_{own} \subseteq PR$. Пусть $p \neq owner(e)$. Согласно условию 3 определения 2, для каждой сущности $e \in E$ либо $e = c_r$, либо существуют контейнеры $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, такие, что $c_1 = c_r$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $e \in H(c_n)$. Согласно замечанию 2, такая последовательность единственна. Следовательно, в соответствии со способом задания множество $holders(e)$ состоит из элемен-

тов $owner(c_1), \ldots, owner(c_n), owner(e)$. Так как $p \neq owner(e)$, существует $i \in \{1, \ldots, n\}$, такое, что $p = owner(c_i)$ и $e < c_i$. Так как $p = owner(c_i)$, по способу задания множества $R_{own}$ выполняется $(p, c_i, \alpha_r) \in R_{own}$. Таким образом, выполняется $p \in P$, $e \in E$, $\alpha_r \in R_r$ и существует контейнер $c_i \in C$, такой, что $(p, c_i, \alpha_r) \in R_{own}$ и $e < c_i$. Тогда по способу задания множества $R_H$ выполняется $(p, e, \alpha_r) \in R_H \subseteq PR$. ■

Используем следующие обозначения:

— $OP_{\text{int}}$ — множество внутренних правил преобразования состояний, заданных в табл. 1;
— $OP_{\text{ext}}$ — множество внешних правил преобразования состояний, заданных в табл. 2;
— $OP = OP_{\text{int}} \cup OP_{\text{ext}}$ — множество правил преобразования состояний;
— $OP^*_{\text{ext}}$ — множество всех конечных последовательностей внешних правил преобразования состояний;
— $OP^*$ — множество всех конечных последовательностей правил преобразования состояний;
— $tr \in OP^*$ — последовательность всех применённых правил, из которой исключены правила, условия применения которых не были выполнены;
— $assoc_f(s)\colon S \to O_p \cup O_t \cup \{\varnothing\}$ — функция, задающая для каждой субъект-сессии $s \in S$ функционально ассоциированную с ней сущность. Для сущностей, функционально-ассоциированных с субъект-сессией $s$, используется обозначение $[s]$;
— $operations\colon O_p \cup O_t \to OP^*_{\text{ext}}$ — функция, задающая для каждой процедуры и триггера соответствующие им последовательности внешних правил преобразования состояний, реализуемых при выполнении субъектами-сессиями их $SQL$-кода.

Таблица 1

**Внутренние правила преобразования состояний**

| Правило | Исходное состояние $G$ | Результирующее состояние $G'$ |
|---|---|---|
| $do\_switch(s, o, u)$ | $s \in S, o \in O_p \cup O_t \cup \{\varnothing\}, u \in U$ | $G'\{user', assoc'_f, tr'\} \sim G$, $user'\{s \mapsto u\} \sim user$, $assoc'_f\{s \mapsto o\} \sim assoc_f$, $tr' = (tr, do\_switch(s, o, u))$ |
| $execute(s, o)$ | $s \in S, o \in O_p \cup O_t, e = [s]$, $operations(o) = (op_1, \ldots, op_k)$, $k \geqslant 0$, $\forall i \in \{1, \ldots, k\}(op_i \in OP_{\text{ext}})$ | $G'' = G(do\_switch(s, o, u), op_1, \ldots, op_k, do\_switch(s, e, user(s)))$, где $p = \begin{cases} user(s) \text{ при } execute\_as(o) = \texttt{as\_caller}, \\ \text{иначе } execute\_as(o) \end{cases}$ $G'\{user\_stack'\} \sim G''$, где $user\_stack' = user\_stack$ |

Таблица 2

**Внешние правила преобразования**

| Правило | Исходное состояние $G$ | Результирующее состояние $G'$ |
|---|---|---|
| $create\_session(s, u)$ | $u \in U, s \notin S$ | $G'\{S', user', user\_stack', assoc'_f, tr'\} \sim G$, $S' = S \cup \{s\}$, $user'\{s \mapsto u\} \sim user$, $user\_stack'\{s \mapsto (u)\} \sim user\_stack$, $assoc'_f\{s \mapsto \varnothing\} \sim assoc_f$, $tr' = (tr, create\_session(s, u))$ |
| $switch(s, u)$ | $s \in S$, $u \in U$, $(user(s), u, impersonate_r) \in R_e$ | $G'' = G(do\_switch(s, [s], u))$, $G'\{user\_stack', tr'\} \sim G''$, $user\_stack'\{s \mapsto (user\_stack(s), u)\} \sim user\_stack''$, $tr' = (tr'', switch(s, u))$ |
| $revert(s)$ | $s \in S$, $user\_stack(s) = (u_1, \ldots, u_{k-1}, u_k)$, $k \geqslant 1$ | $G'\{user', user\_stack', tr'\} \sim G$, $user'\{s \mapsto \mapsto u_{\max(1,k-1)}\} \sim user$, $user\_stack'\{s \mapsto (u_1, \ldots, u_{\max(1,k-1)})\} \sim user\_stack$, $tr' = (tr, revert(s))$ |

Окончание таблицы 2

| Правило | Исходное состояние $G$ | Результирующее состояние $G'$ |
|---|---|---|
| $grant\_right$ $(s, p, e, \alpha_r,$ $with\_grant)$ | $s \in S, p \in P, e \in E, \alpha_r \in R_r,$ $user(s) \in Gr(e, \alpha_r),$ $with\_grant \in \{\texttt{yes}, \texttt{no}\}$ | $G'\{R'_d, Gr'_d, tr'\} \sim G$, $R'_d = R_d \cup \{(p, e, \alpha_r)\}$, $Gr'_d = \begin{cases} Gr_d \cup \{(p, e, \alpha_r)\} \text{ при } with\_grant = \texttt{yes}, \\ Gr_d \text{ при } with\_grant = \texttt{no} \end{cases}$ $tr' = (tr, grant\_right(s, p, e, \alpha_r, with\_grant))$ |
| $add\_member$ $(s, r, u)$ | $s \in S$, $r \in R$, $u \in U$ $(user(s), r, alter_r) \in R_e$ | $G'\{UA', tr'\} \sim G$, $UA'\{u \mapsto UA(u) \cup \{r' \in R : r' \leqslant r\}\} \sim UA$, $tr' = (tr, add\_role\_member(s, r, u))$ |
| $create\_container$ $(s, c_p, c, mode)$ | $s \in S$, $c_p \in C$, $c \notin C$, $(user(s), c_p, alter_r) \in R_e$ $mode \in \{\texttt{creator}, \texttt{parent}\}$ | $G'\{E'_H, H', owner', cmode', tr'\} \sim G$, $E'_H = C' \cup O_p \cup P \cup O_t$, $C' = C \cup \{c\}$, $H'\{c \mapsto \varnothing, c_p \mapsto H(c_p) \cup \{c\}\} \sim H$, $owner'\{c \mapsto p\} \sim owner$, где $p = \begin{cases} user(s) \text{ при } cmode(c_p) = \texttt{creator}, \\ owner(c_p) \text{ при } cmode(c_p) = \texttt{parent} \end{cases}$ $cmode'\{c \mapsto mode\} \sim cmode$, $tr' = (tr, create\_container(s, c_p, c, mode))$ |
| $create\_procedure$ $(s, c, o_p, md,$ $op_1, \ldots, op_k)$ | $s \in S, c \in C \setminus T$, $o_p \notin O_p$, $cmode(c) = \texttt{parent}$, $md \in U \cup \{\texttt{as\_caller}\}$, $k \geqslant 0$, $\forall i \in \{1, \ldots k\}(op_i \in OP_{\text{ext}})$, $(user(s), c, alter_r) \in R_e$; если $md \in U$, то $(user(s),$ $md, impersonate_r) \in R_e$ | $G'\{E'_H, H', owner', execute\_as', operations',$ $tr'\} \sim G$, $E'_H = C \cup O'_p \cup P \cup O_t$, $O'_p = O_p \cup \{o_p\}$, $H'\{c \mapsto H(c) \cup \{o_p\}\} \sim H$, $owner'\{o_p \mapsto owner(c)\} \sim owner$, $execute\_as'\{o_p \mapsto md\} \sim execute\_as$, $operations'\{o_p \mapsto (op_1, \ldots, op_k)\} \sim operations$, $tr' = (tr, create\_procedure(s, c, o_p, md, op_1, \ldots, op_k))$ |
| $alter\_procedure$ $(s, o_p, md,$ $op_1, \ldots, op_k)$ | $s \in S$, $o_p \in O_p$, $md \in U \cup \{\texttt{as\_caller}\}$, $k \geqslant 0$, $\forall i \in \{1, \ldots k\}(op_i \in OP_{\text{ext}})$, $(user(s), o_p, alter_r) \in R_e$; если $md \in U$, то $(user(s),$ $md, impersonate_r) \in R_e$ | $G'\{execute\_as', operations', tr'\} \sim G$, $execute\_as'\{o_p \mapsto m\} \sim execute\_as$, $operations'\{o_p \mapsto (op_1, \ldots, op_k)\} \sim operations$, $tr' = (tr, alter\_procedure(s, o_p, md, op_1, \ldots, op_k))$ |
| $create\_trigger$ $(s, o_t, t, \alpha_r, md,$ $op_1, \ldots, op_k)$ | $s \in S$, $t \in T$, $o_t \notin O_t$, $\alpha_r \in \{insert_r, update_r, delete_r\}$, $md \in U \cup \{\texttt{as\_caller}\}$, $k \geqslant 0$, $\forall i \in \{1, \ldots k\}(op_i \in OP_{\text{ext}})$, $(user(s), t, alter_r) \in R_e$; если $md \in U$, то $(user(s),$ $md, impersonate_r) \in R_e$ | $G'\{E'_H, H', owner', execute\_as', triggers',$ $operations', tr'\} \sim G$, $E'_H = E_H \cup O'_t$, $O'_t = O_t \cup \{o_t\}$, $H'\{t \mapsto H(t) \cup \{o_t\}\} \sim H$, $owner'\{o_t \mapsto owner(t)\} \sim owner$, $execute\_as'\{o_t \mapsto md\} \sim execute\_as$, $triggers'\{(t, \alpha_r) \mapsto (triggers(t, \alpha_r), o_t)\} \sim$ $\sim triggers$, $operations'\{o_t \mapsto (op_1, \ldots, op_k)\} \sim$ $\sim operations$, $tr' = (tr, create\_trigger(s, o_t, t, \alpha_r,$ $md, op_1, \ldots, op_k))$ |
| $alter\_trigger$ $(s, o_t, t, md,$ $op_1, \ldots, op_k)$ | $s \in S, t \in T, o_t \in H(t)$, $md \in U \cup \{\texttt{as\_caller}\}$, $k \geqslant 0$, $\forall i \in \{1, \ldots k\}(op_i \in OP_{\text{ext}})$, $(user(s), t, alter_r) \in R_e$; если $md \in U$, то $(user(s),$ $md, impersonate_r) \in R_e$ | $G'\{execute\_as', operations', tr'\} \sim G$, $execute\_as'\{o_t \mapsto md\} \sim execute\_as$, $operations'\{o_t \mapsto (op_1, \ldots, op_k)\} \sim operations$, $tr' = (tr, alter\_trigger(s, t, o_t, md, op_1, \ldots, op_k))$ |
| $execute\_procedure$ $(s, o_p)$ | $s \in S, o_p \in O_p$, $(user(s), o_p, execute_r) \in R_e$ или $[s] \neq \varnothing$ и $owner([s]) =$ $= owner(o_p)$ | $G' = G(execute(s, o_p))$ |
| $access(s, t, \alpha_r)$ | $s \in S$, $t \in T$, $\alpha_r \in \{insert_r,$ $update_r, delete_r\}$, $triggers(t, \alpha_r) =$ $= (o_{t,1}, \ldots, o_{t,k})$, $k \geqslant 0$, $\forall i \in \{1, \ldots, k\}(o_{t,i} \in O_t)$, $(user(s), t, \alpha_r) \in R_e$ или $[s] \neq \varnothing$ и $owner([s]) =$ $= owner(t)$ | $G' = G(execute(s, o_{t,1}), \ldots, execute(s, o_{t,k}))$ |

Необходимость во введении нового обозначения $assoc_f(s)$ обусловлена тем, что в отличие от других моделей полагается, что в каждый момент времени множество функционально-ассоциированных с субъект-сессией $s$ сущностей $assoc_f(s)$ либо содержит ровно один выполняемый им в текущий момент времени триггер или процедуру, либо не содержит элементов. Последнее означает, что субъект-сессия выполняет SQL-код, который не относится ни к одной процедуре или триггеру. В последнем случае считается, что выполнение такого SQL-кода эквивалентно непосредственному применению субъект-сессией правил преобразования состояний из множества $OP_{\text{ext}}$. В дальнейшем будем полагать, что по определению выполняется равенство $[s] = assoc_f(s)$.

SQL-код может либо относиться к одной из процедур или триггеров, либо не относиться ни к одной из них. Считается, что SQL-код содержит инструкции, выполнение которых непосредственно приводит к применению внешних правил из множества $OP_{\text{ext}}$. При этом предполагается, что SQL-код не может содержать инструкции, выполнение которых непосредственно приводит к применению внутренних правил из множества $OP_{\text{int}}$. Выполнение внутренних правил инициируется опосредованно как составная часть преобразований системы, задаваемых внешними правилами. Например, выполнение SQL-кода триггеров осуществляется только при применении внешних правил, связанных со вставкой, удалением и обновлением записей таблиц.

В рамках модели предполагается, что триггеры и процедуры не содержат динамически генерируемого SQL-кода и условий, а их выполнение приводит к применению фиксированной последовательности правил, задаваемой соответствующим значением функции *operations*.

В СУБД MS SQL Server существует специальный режим проверки прав доступа при наличии «цепочки владения» (*ownership chaining*). При выполнении DML-инструкций (Data Manipulation Language) SQL-кода процедуры не выполняется проверка прав доступа при обращении к сущностям, владелец которых совпадает с владельцем выполняемой процедуры. Аналогичные проверки не производятся и в отношении DML-инструкций SQL-кода триггеров. Модель учитывает данную особенность функционирования СУБД при задании условий применения внешних правил преобразования состояний системы.

**Определение 6.** Обозначим через $O_t^*$ множество всех конечных последовательностей триггеров. Определим $triggers\colon T \times \{insert_r, update_r, delete_r\} \to O_t^*$ — функцию, задающую для каждой таблицы связанную с ней последовательность триггеров и удовлетворяющую следующим условиям.

У с л о в и е 1. Для каждой таблицы $t \in T$ и каждого права доступа $\alpha_r \in \{insert_r, update_r, delete_r\}$ выполняется $triggers(t, \alpha_r) \subseteq H(t)$.

У с л о в и е 2. Для каждого триггера $o_t \in O_t$ существует единственная таблица $t \in T$ и единственное право доступа $\alpha_r \in \{insert_r, update_r, delete_r\}$, такое, что $o_t \in triggers(t, \alpha_r)$.

Если существуют таблица $t \in T$, право доступа $\alpha_r \in \{insert_r, update_r, delete_r\}$ и триггер $o_t \in triggers(t, \alpha_r)$, то будем говорить, что триггер $o_t$ имеет тип $\alpha_r$ и связан с таблицей $t$.

Каждый триггер имеет тип, задаваемый функцией *triggers* и соответствующий определённому виду доступа. Реализация субъект-сессией права доступа вида $\alpha_r \in \{insert_r, update_r, delete_r\}$ к таблице $t \in T$ с применением правила $access(s, t, \alpha_r)$ (см. табл. 2) автоматически инициирует выполнение ею SQL-кода всех триггеров из последовательности $triggers(t, \alpha_r)$.

Введём следующие обозначения:

— $G = (E_H,\ H,\ UA,\ S,\ R_d,\ Gr_d,\ user,\ user\_stack,\ assoc_f,\ owner,\ cmode,\ execute\_as,\ triggers, operations, tr)$ — состояние системы;
— $\Sigma(G^*, OP)$ — система, где $G^*$ — множество всех возможных состояний;
— $\Sigma(G^*, OP, G_0)$ — система $\Sigma(G^*, OP)$ с начальным состоянием $G_0$.

Отметим, что из состояния системы исключены элементы, которые не изменяются либо могут быть определены на основании других элементов.

В дальнейшем будем использовать следующее предположение.

**Предположение 4.** В начальном состоянии системы $\Sigma(G^*, OP, G_0)$ отсутствуют субъект-сессии, то есть $S_0 = \varnothing$, и последовательность $tr_0$ не содержит элементов.

## 2. Правила преобразования состояний

Будем использовать запись вида $G'\{I'_1, \ldots, I'_k\} \sim G$ для обозначения того, что все элементы состояния $G'$ равны соответствующим элементам состояния $G$, за исключением элементов $I'_1, \ldots, I'_k$ состояния $G'$. Будем также использовать запись вида $f'\{x_1, \ldots, x_k\} \sim f$ для обозначения того, что значения функции $f'(x)$ равны значениям функции $f(x)$ для всех аргументов $x$, за исключением $x_1, \ldots, x_k$. Кроме того, будем использовать запись вида $f'\{x_1 \mapsto y_1, \ldots, x_k \mapsto y_k\} \sim f$ для обозначения того, что значения функции $f'(x)$ равны значениям функции $f(x)$ для всех аргументов $x$, кроме $x_1, \ldots, x_k$, и при этом $f'(x_1) = y_1$, …, $f'(x_k) = y_k$. Очевидно, что если выполняется $G'\{I'_1, \ldots, I'_k\} \sim G$, то также верно и $G\{I_1, \ldots, I_k\} \sim G'$.

Используем обозначение: $G \vdash_{op} G'$ — переход системы $\Sigma(G^*, OP)$ из состояния $G$ в состояние $G'$ с применением правила преобразования состояний $op \in OP$, при этом если условия применения правила $op$ не выполняются, то по определению справедливо равенство $G' = G$.

Запись вида $G' = G(op_1, \ldots, op_k)$ означает, что состояние $G'$ есть результат последовательного применения правил преобразования состояний $op_1, \ldots, op_k$, начиная с состояния $G$, то есть существуют состояния $G_1, \ldots, G_k$, такие, что $G \vdash_{op_1} G_1 \vdash_{op_2} \ldots \vdash_{op_k} G_k$ и $G' = G_k$.

В модели определены приведённые в табл. 1 и 2 правила преобразования состояний из множества $OP$, в которых учтены особенности управления доступом СУБД MS SQL Server 2012.

**Замечание 4.** Будем считать, что все правила из табл. 1 и 2, за исключением правила $create\_session(s, u)$, получают в качестве первого параметра субъект-сессию, инициировавшую их выполнение.

Будем использовать обозначение $op\langle s\rangle$ в случае, если применение правила инициировано субъект-сессией $s$. Будем считать, что применение правила $create\_session(s, u)$ также инициировано субъект-сессией $s$.

**Определение 7.** Будем говорить, что применение правила $op \in OP$ инициировано учётной записью пользователя $u \in U$, если либо $op = create\_session(s, u)$, либо $op$ имеет в качестве первого параметра субъект-сессию $s$, такую, что $u = user(s)$.

Поясним приведённые в табл. 1 условия и результаты применения внутренних правил преобразования системы.

В отличие от внешних, внутренние правила применяются субъект-сессиями опосредованно в ходе выполнения SQL-кода как составная часть преобразований системы, задаваемых внешними (см. описание правил $execute\_procedure$, $access$, $switch$ в табл. 2) или другими внутренними правилами (см. описание правила $execute$).

В результате применения внутреннего правила $do\_switch(s, o, u)$ в качестве учётной записи, от имени которой функционирует субъект-сессия $s$, устанавливается $u$, а $o$ становится функционально-ассоциированной с $s$ сущностью. Запись вида $(tr, op)$, использованная при определении нового значения $tr'$, означает, что в возможно пустую последовательность $tr$ добавляются данные о применяемом правиле $op$.

Внутреннее правило $execute(s, o)$ задаёт состояние системы после выполнения SQL-кода, содержащегося в процедуре или триггере $o \in O_p \cup O_t$ субъект-сессией $s \in S$. Перед выполнением SQL-кода сущности $o$ производится изменение учётной записи пользователя, от имени которой субъект-сессия $s$ будет последовательно применять внешние правила $op_1, \ldots, op_k$. Учётная запись пользователя, от имени которой функционирует субъект-сессия $s$, устанавливается в соответствии с режимом выполнения кода процедуры или триггера $o$, задаваемых значением функции $execute\_as(o)$. Кроме того, в результате применения правила $do\_switch$ перед выполнением SQL-кода сущности $o$ в качестве функционально-ассоциированной с субъект-сессией $s$ сущностью устанавливается сущность $o$. После завершения описанных подготовительных действий производится выполнение SQL-кода сущности $o$, что приводит к реализации преобразований состояний системы, задаваемых последовательностью правил $operations(o) = (op_1, \ldots, op_k)$. После выполнения кода с использованием правила $do\_switch$ производится восстановление исходной функционально-ассоциированной сущности с субъект-сессией $s$, а также учётной записи пользователя, от имени которой она выполнялась до применения внутреннего правила $execute(s, o)$. После завершения описанных действий производится восстановление исходной последовательности учётных записей, от имени которых выполнялась субъект-сессия $s$, до применения правила $execute(s, o)$.

В табл. 2 приведены внешние правила преобразования состояний из множества $OP_{\text{ext}}$, которые могут быть непосредственно применены субъект-сессиями. Поясним условия и результаты применения внешних правил преобразования состояний.

В результате применения правила $create\_session(s, u)$ создаётся новая субъект-сессия $s$, которая в дальнейшем может выполнять действия от имени учётной записи пользователя $u \in U$. Заметим, что после создания субъект-сессии множество функционально-ассоциированных с ней сущностей пусто. Отметим, что изначально в последовательность учётных записей, от имени которых выполнялась субъект-сессия, заносится учётная запись пользователя $u$, инициировавшего создание субъект-сессии $s$. В реальных СУБД при подключении к базе данных пользователь должен указать информацию, аутентифицирующую его учётную запись.

**Определение 8.** Пусть $G \vdash_{op} G'$ — переход системы $\Sigma(G^*, OP)$ из состояния $G$ в состояние $G'$. Назовём переход $G \vdash_{op} G'$ инициированным извне, если $op \in OP_{\text{ext}}$ и либо $op = create\_session(s, u)$, либо правило $op$ получает в качестве первого параметра субъект-сессию $s$, такую, что $[s] = \varnothing$. Если переход $G \vdash_{op} G'$ инициирован извне, то будем называть правило $op$ инициированным извне, при этом если $user(s) = u$, то будем использовать обозначение $op[u]$.

В дальнейшем будем считать, что выполняется следующее предположение.

**Предположение 5.** Применение правила $create\_session(s, u)$ может быть инициировано только извне.

В результате применения правила $switch(s, u)$ в качестве учётной записи, от имени которой функционирует субъект-сессия $s$, устанавливается $u$. Необходимым условием применения данного правила является наличие у учётной записи, от имени кото-

рой функционирует субъект-сессия $s$, права доступа $impersonate_r$ к учётной записи пользователя $u$. Запись вида $(user\_stack(s), u)$, использованная при определении нового значения $user\_stack'(s)$, означает, что в возможно пустую последовательность $user\_stack(s)$ добавляется учётная запись пользователя $u$. Отметим, что в результате применения правила $switch(s, u)$ функционально ассоциированная с $s$ сущность не изменяется. Заметим, что правилу $switch$ соответствует SQL-оператор *EXECUTE AS*, используемый в реальных СУБД для выполнения действий от имени учётной записи заданного пользователя.

Правило $revert(s)$ предназначено для восстановления учётной записи, от имени которой функционировала субъект-сессия $s$ до момента предыдущего применения ею правила $switch$. Заметим, что восстановление организовано по принципу *LIFO* (*Last In — First Out*). Единственное отличие состоит в том, что учётная запись пользователя, от имени которого была создана субъект-сессия $s$, никогда не удаляется из последовательности $user\_stack(s)$. В СУБД для восстановления контекста выполнения до применения оператора *EXECUTE AS* используется оператор *REVERT*.

**Замечание 5.** Из способа задания внешних правил в табл. 2 видно, что при применении правил учитываются права учётной записи пользователя, от имени которой в текущем состоянии выполняется субъект-сессия $s$, а не права учётной записи пользователя, инициировавшего создание субъект-сессии $s$. Следовательно, наличие права $impersonate_r$ к учётной записи $u$ позволяет с использованием $switch$ инициировать применение любого внешнего правила преобразования состояний системы с правами учётной записи пользователя $u$. Это означает, что наличие права $impersonate_r$ к учётной записи $u$ равносильно обладанию информации, аутентифицирующей учётную запись пользователя $u$. Кроме того, можно считать, что наличие права $impersonate_r$ к учётной записи $u$ равносильно функционированию в кооперации с субъект-сессией, создание которой было инициировано ею.

В результате применения правила $grant\_right(s, p, e, \alpha_r, with\_grant)$ принципалу $p$ предоставляется право доступа $\alpha_r$ к сущности $e$. Необходимым условием является наличие у учётной записи пользователя, от имени которой выполняется субъект-сессия $s$, права давать право доступа $\alpha_r$ к сущности $e$, то есть $user(s) \in Gr(e, \alpha_r)$. Если параметр $with\_grant$ равен `yes`, то принципалу $p$ даётся также право предоставлять право $\alpha_r$ к сущности $e$. Правило $grant\_right$ соответствует SQL-оператору *GRANT*, позволяющему предоставлять права пользователям и ролям к сущностям базы данных.

Назначение правила $add\_member(s, r, u)$ состоит в предоставлении учётной записи пользователя $u$ авторизации на роль $r$, а также на все роли, которые «иерархически подчинены» роли $r$. Необходимым условием применения данного правила является наличие у учётной записи, от имени которой выполняется субъект-сессия $s$, права доступа $alter_r$ к роли $r$.

**Замечание 6.** Исходя из способа задания внутренних и внешних правил преобразования состояний, можно сделать вывод, что единственным способом изменения множества действующих прав доступа учётных записей пользователей на предоставление прав доступа к сущностям $Gr_e$ является применение правила $grant\_right$. В то же время изменение множества действующих прав доступа $R_e$ возможно с применением либо правила $grant\_right$, либо $add\_member$.

Правило $create\_container(s, c_p, c, mode)$ позволяет субъект-сессии $s$ создать новый контейнер $c$, включив его в существующий контейнер $c_p$. Необходимым условием применения данного правила является наличие у учётной записи пользователя $user(s)$

права доступа $alter_r$ к контейнеру $c_p$. Владельцем вновь созданного контейнера $c$ назначается в зависимости от режима установки владельца вновь создаваемых дочерних сущностей контейнера $c_p$ либо учётная запись пользователя, инициировавшая применение правила (режим $cmode(c_p) = \texttt{creator}$), либо принципал владельца контейнера $c_p$ (режим $cmode(c_p) = \texttt{parent}$). Режим установки владельца создаваемого контейнера устанавливается равным параметру $mode$.

В СУБД MS SQL Server способ установки владельца экземпляра СУБД и экземпляров баз данных соответствует режиму `creator`, то есть владельцем становится пользователь, инициировавший их создание. С другой стороны, у схем способ установки соответствует режиму `parent`. Таким образом, владельцем создаваемых таблиц и триггеров становится владелец схемы, в которой выполняется операция создания.

В результате применения правила $create\_procedure(s, c, o_p, md, op_1, \ldots, op_k)$ в контейнере $c$, не являющемся таблицей, создаётся новая процедура $o_p$, содержащая SQL-код, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$, при этом владельцем вновь созданной процедуры $o_p$ становится принципал владельца контейнера $c$, а режим выполнения $o_p$ устанавливается равным $md$. Необходимым условием выполнения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к контейнеру $c$. Для назначения режима выполнения процедуры от имени заданной учётной записи пользователя требуется наличие к ней у $user(s)$ права доступа $impersonate_r$.

В результате применения правила $alter\_procedure(s, o_p, md, op_1, \ldots, op_k)$ производится изменение режима выполнения существующей процедуры $o_p$, а также замена её SQL-кода на новый, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$. Необходимым условием выполнения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к процедуре $o_p$. Для назначения режима выполнения процедуры от имени учётной записи пользователя требуется наличие к ней права доступа $impersonate_r$. Отметим, что, в отличие от правила $create\_procedure$, владелец процедуры $o_p$ не изменяется.

В результате применения правила $create\_trigger(s, t, o_t, \alpha_r, md, op_1, \ldots, op_k)$ в таблице $t$ создаётся новый триггер $o_t$, содержащий SQL-код, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$, при этом владельцем вновь созданного триггера $o_t$ становится принципал владельца таблицы $t$, а режим выполнения $o_t$ задаётся равным $md$. Созданный триггер активизируется только при реализации к таблице $t$ права доступа $\alpha_r$ с использованием правила $access(s, t, \alpha_r)$. Необходимым условием выполнения правила $create\_trigger$ является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к таблице $t$. Для назначения режима выполнения триггера от имени заданной учётной записи пользователя требуется наличие к ней у $user(s)$ права доступа $impersonate_r$.

В результате применения правила $alter\_trigger(s, t, o_t, md, op_1, \ldots, op_k)$ в таблице $t$ производится модификация режима выполнения существующего триггера $o_t$, а также замена его SQL-кода на новый, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$. Необходимым условием выполнения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к таблице $t$. Для назначения режима выполнения триггера от имени учётной записи пользователя требуется наличие к ней у $user(s)$ права доступа $impersonate_r$. Отметим, что, в отличие от правила $create\_trigger$, не выполняется изменение учётной записи владельца триггера $o_t$, а также вида права доступа, при реализации которого активизируется триггер $o_t$.

**Замечание 7.** В результате применения правил *create_container*, *create_procedure* и *create_trigger* не выполняется добавление в множества $R_d$ и $Gr_d$ прав доступа учётной записи владельца к вновь созданной сущности. Это связано с тем, что при задании учётной записи владельца в соответствии с правилами задания в множества $R_e$ и $Gr_e$ включаются права доступа всех её иерархических владельцев.

Будем использовать обозначение $op \prec expression$ в случае, если операция $op$ имеет вид $expression$. Например, если операция $op$ имеет вид $create\_session(s, \ldots)$, то будем использовать обозначение $op \prec create\_session(s, \ldots)$. Будем использовать обозначение $op \not\prec expression$ в случае, если операция $op$ не имеет вид $expression$.

**Предположение 6.** Будем считать, что выполнение SQL-кода триггеров и процедур не приводит к применению правил *create_container*, *create_procedure*, *alter_procedure*, *create_trigger* или *alter_trigger*, то есть перечисленные правила могут быть применены только извне. Следовательно, при применении правила $op \prec create\_procedure(\ldots, op_1, \ldots, op_k)$, либо $op \prec alter\_procedure(\ldots, op_1, \ldots, op_k)$, либо $op \prec create\_trigger(\ldots, op_1, \ldots, op_k)$, либо $op \prec alter\_trigger(\ldots, op_1, \ldots, op_k)$ для каждого $i = 1, \ldots, k$ правило $op_i \not\prec create\_procedure$, $op_i \not\prec alter\_procedure$, $op_i \not\prec create\_trigger$ и $op_i \not\prec alter\_trigger$.

**Замечание 8.** Заметим, что если $o \in O_p \cup O_t$, $operations(o) = (op_1, \ldots, op_k)$, где $k \geqslant 0$, то $op_i \not\prec create\_session$ для каждого $i = 1, \ldots, k$. Предположим противное: существует $i \in \{1, \ldots, k\}$, такое, что $op_i \prec create\_session$. В соответствии со способом задания внутреннего правила $execute(s, o)$ на время применения правил сущности $o$, она становится функционально-ассоциированной с субъект-сессией $s$. Это противоречит предположению о том, что применение правила $op_i \prec create\_session$ может быть инициировано только извне. Следовательно, правила, связанные с процедурами и триггерами, не могут включать правила вида *create_session*. Кроме того, при применении правил вида $create\_procedure(\ldots, op_1, \ldots, op_k)$, $alter\_procedure(\ldots, op_1, \ldots, op_k)$, $create\_trigger(\ldots, op_1, \ldots, op_k)$ и $alter\_trigger(\ldots, op_1, \ldots, op_k)$ для каждого $i = 1, \ldots, k$ выполняется $op_i \not\prec create\_session$.

Допустим, что $o \in O_p \cup O_t$, $operations(o) = (op_1, \ldots, op_k)$, где $k \geqslant 0$. Будем использовать обозначение $op \in operations(o)$ в случае, если существует $i \in \{1, \ldots, k\}$, такое, что $op = op_i$.

**Замечание 9.** Пусть $o \in O_p \cup O_t$ и $op \in operations(o)$. В соответствии с предположением 6 и замечанием 8, правило $op$ не может иметь вид *create_container*, *create_procedure*, *alter_procedure*, *create_trigger*, *alter_trigger* и *create_session*. Учитывая это, а также то, что $op \in OP_{\text{ext}}$, по определению функции *operations* получаем, что либо $op \prec grant\_right$, либо $op \prec add\_member$, либо $op \prec execute\_procedure$, либо $op \prec access$.

Для упрощения анализа условий несанкционированного распространения прав доступа будем считать, что выполняется следующее предположение.

**Предположение 7.** Считается, что в начальном состоянии $\Sigma(G^*, OP, G_0)$ для каждого $o \in O_{p_0} \cup O_{t_0}$ и каждого $op \in operations_0(o)$ выполняется $op \not\prec grant\_right$ и $op \not\prec add\_member$.

Это предположение должно соблюдаться разработчиками автоматизированных информационных системы, использующих данную модель. Таким образом, предоставления прав доступа и управление членством в группах должно производиться вне процедур и триггеров.

Правило $execute\_procedure(s, o_p)$ позволяет субъект-сессии $s$ выполнить SQL-код процедуры $o_p$. Необходимым условием применения данного правила является либо наличие у учётной записи пользователя, от имени которой выполняется $s$, права доступа $execute_r$ к процедуре $o_p$, либо совпадение принципала владельца процедуры $o_p$ и принципала владельца процедуры или триггера, в рамках выполнения SQL-кода которых предпринята попытка применения правила $execute\_procedure(s, o_p)$. Последний случай соответствует специальному режиму проверки прав доступа при наличии «цепочки владения». Отметим, что при выполнении SQL-кода процедуры может осуществляться активация других процедур и триггеров.

Применение правила $access(s, t, \alpha_r)$ приводит к активации триггеров типа $\alpha_r$ таблицы $t$ субъект-сессией $s$. Необходимым условием применения данного правила является либо наличие у учётной записи пользователя, от имени которой выполняется $s$, права доступа $\alpha_r$ к таблице $t$, либо совпадение принципала владельца таблицы $t$ и принципала владельца процедуры или триггера, в рамках выполнения SQL-кода которых предпринята попытка применения правила $access(s, t, \alpha_r)$. Отметим, что при выполнении SQL-кода триггера может осуществляться активация других процедур и триггеров.

**Замечание 10.** Заметим, что в реальных СУБД активация триггеров таблицы выполняется только при внесении в неё изменений.

**Замечание 11.** Будем считать, что при активации правил преобразования состояний, связанных с процедурами и триггерами, не возникает циклов, являющихся результатом повторного применения правил ранее активированных сущностей.

**Замечание 12.** В соответствии со способом задания применение внешних и внутренних правил не приводит к удалению элементов из множеств, являющихся элементами состояния системы $\Sigma(G^*, OP)$. Следовательно, все правила системы $\Sigma(G^*, OP)$ являются монотонными.

Учитывая монотонность правил системы, а также то, что в условиях применения правил $grant\_right$, $add\_member$ $alter\_procedure$, $alter\_trigger$, $execute\_procedure$, $access$ нет проверок на отсутствие элементов в состоянии системы $\Sigma(G^*, OP)$, отметим, что верно следующее замечание.

**Замечание 13.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$ и $op_1, \ldots, op_k \in OP$, где $k \geqslant 1$, $G' = G(op_1, \ldots, op_k)$, $op$ является одним из перечисленных выше правил. Пусть в состоянии $G$ выполнены условия применения правила $op$ субъект-сессией, функционирующей от имени учётной записи пользователя $u \in U$. Тогда в состоянии $G'$ также будут выполнены условия применения правила $op$ субъект-сессией, функционирующей от имени учётной записи пользователя $u$.

## 3. Функция *vestige*

В соответствии со способом задания правил преобразования состояний в результате применения правил $execute\_procedure$ и $access$ может быть инициировано выполнение правил триггеров и процедур. Введём функцию $vestige(G, op_1, \ldots, op_k)$, значением которой является последовательность всех правил преобразований состояний системы, применение которых является результатом последовательного применения правил $op_1, \ldots, op_k$ в состоянии $G$.

**Определение 9.** Пусть заданы состояния системы $G, G_1, \ldots, G_k$, где $k \geqslant 1$, и правила преобразования состояний $op_1, \ldots, op_k$, такие, что $G \vdash_{op_1} G_1 \vdash_{op_2} \ldots \vdash_{op_k} G_k$ и для каждого $i = 1, \ldots, k$ переход $G_{i-1} \vdash_{op_i} G_i$ инициирован извне и $tr_k = (tr, \widehat{op}_1, \ldots, \widehat{op}_m)$,

где $m \geqslant 0$. Обозначим через $vestige(G, op_1, \ldots, op_k)$ последовательность правил преобразования состояний $(\widehat{op}_1, \ldots, \widehat{op}_m)$.

**Замечание 14.** Заметим, что последовательность правил $vestige(G, op_1, \ldots, op_k)$ однозначно определяется состоянием системы $G$ и последовательностью инициированных извне правил $op_1, \ldots, op_k$, поэтому в дальнейшем $vestige$ будет рассматриваться как функция $G^* \times OP^*_{\text{ext}} \to OP^*$.

**Замечание 15.** Отметим, что в соответствии со способом задания перехода системы из состояния в состояние последовательность правил преобразования состояний $vestige(G, op_1, \ldots, op_k)$ не включает правила, условия применения которых не выполняются.

Пусть заданы состояния системы $G_0, \ldots, G_k$, где $k \geqslant 1$, и правила преобразования состояний $op_1, \ldots, op_k$, такие, что $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \ldots \vdash_{op_k} G_k$ и для каждого $i = 1, \ldots, k$ переход $G_{i-1} \vdash_{op_i} G_i$ инициирован извне и $vestige(G_0, op_1, \ldots, op_k) = (\widehat{op}_1, \ldots, \widehat{op}_m)$. Пусть состояния $\widehat{G}_0, \widehat{G}_1, \ldots, \widehat{G}_m$ таковы, что $\widehat{G}_0 = G_0$ и $\widehat{G}_0 \vdash_{\widehat{op}_1} \widehat{G}_1 \vdash_{\widehat{op}_2} \ldots \vdash_{\widehat{op}_m} \widehat{G}_m$. Функция $vestige$ обладает следующими свойствами.

**Свойство 1.** $G_k = \widehat{G}_m$.

**Свойство 2.** $vestige(G_0, op_1, \ldots, op_k) = (vestige(G_0, op_1, \ldots, op_{k-1}), vestige(G_{k-1}, op_k)) = (vestige(G_0, op_1), vestige(G_1, op_2), \ldots, vestige(G_{k-1}, op_k))$ (для удобства будем считать, что запись $((\widehat{op}_1, \ldots, \widehat{op}_m), (\widehat{op}_{m+1}, \ldots, \widehat{op}_n))$ эквивалентна последовательности $(\widehat{op}_1, \ldots, \widehat{op}_m, \widehat{op}_{m+1}, \ldots, \widehat{op}_n)$).

**Свойство 3.** Если в состоянии $G_0$ не выполнены условия применения правила $op \in OP_{\text{ext}}$, то $vestige(G_0, op) = ()$.

**Свойство 4.** Если выполнены условия применения правила $op \in OP_{\text{ext}}$ в состоянии $G_0$ и $op \not\prec access$ и $op \not\prec execute\_procedure$, то $vestige(G, op) = (op)$.

**Свойство 5.** Для каждого $i = 1, \ldots, m$ либо $\widehat{op}_i \prec switch$, либо $\widehat{op}_i \prec revert$, либо $\widehat{op}_i \prec create\_session$, либо $\widehat{op}_i \prec create\_procedure$, либо $\widehat{op}_i \prec alter\_procedure$, либо $\widehat{op}_i \prec create\_trigger$, либо $\widehat{op}_i \prec alter\_trigger$, либо $\widehat{op}_i \prec grant\_right$, либо $\widehat{op}_i \prec add\_member$.

**Свойство 6.** Для каждого $i = 1, \ldots, m$ выполняется $\widehat{op}_i \not\prec access$ и $\widehat{op}_i \not\prec execute\_procedure$.

Выполнение указанных свойств следует из определения функции $vestige$ и способа задания внутренних и внешних правил преобразования состояний в табл. 1 и 2.

## 4. Анализ условий выполнения SQL-кода от имени заданной учётной записи

Несанкционированное выполнение нарушителем произвольного SQL-кода от имени другой учётной записи является одной из главных угроз безопасности автоматизированной информационной системы. Наличие у нарушителя возможности выполнения произвольного SQL-кода от имени административной учётной записи может привести к компрометации всей системы в целом. Следовательно, необходимо провести всесторонний теоретический анализ условий, при выполнении которых возможно выполнение пользователем произвольного SQL-кода от имени заданной учётной записи. В рамках модели возможность выполнение произвольного $SQL$-кода от имени заданной учётной записи рассматривается как возможность применения от её имени произвольного правила.

Введём определение возможности применения правила от имени заданной учётной записи.

**Определение 10.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $u, u' \in U$, $op \in OP_{\text{ext}}$, $s \notin S$. Пусть в состоянии $G$ выполнены условия применения внешнего правила $op$ субъект-сессией, функционирующей от имени учётной записи пользователя $u'$. Определим предикат $can\_execute\_as(u, u', op, G)$, истинный тогда и только тогда, когда существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $vestige(G, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$ и существует $i \in \{2, \ldots, m\}$, что $\widehat{op}_i = op$ и $\widehat{user}_{i-1}(s) = u'$.

Для теоретического анализа условий выполнения SQL-кода от имени заданной учётной записи введём следующее определение.

**Определение 11.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $u, u' \in U$. Определим предикат $can\_act\_as(u, u', G)$, истинный тогда и только тогда, когда истинен предикат $can\_execute\_as(u, u', op, G)$ для любого внешнего правила $op \in OP_{\text{ext}}$, для которого в состоянии $G$ выполнены условия его применения субъект-сессией, функционирующей от имени учётной записи пользователя $u'$.

В соответствии с замечанием 5 обладание учётной записью пользователя $u$ правом доступа $impersonate_r$ к учётной записи пользователя $u'$ позволяет применить правило $switch$ и в дальнейшем выполнить любое внешнее правило преобразования состояния системы от её имени. Для анализа возможности получения права $impersonate_r$ введём следующее определение.

**Определение 12.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $u, u' \in U$, $s \notin S$. Определим предикат $can\_become(u, u', G)$, истинный тогда и только тогда, когда существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$ и $(u, u', impersonate_r) \in R_{e_k}$.

В случае истинности предиката $can\_become(u, u', G)$ субъект-сессия $s$, созданная пользователем $u$, без кооперации с другими субъект-сессиями может получить право доступа $impersonate_r$ к учётной записи пользователя $u'$.

### 4.1. Эквивалентность возможности выполнения произвольного SQL-кода от имени учётной записи пользователя и возможности получения к ней права олицетворения

Справедливо следующее утверждение о связи между возможностью выполнения произвольного SQL-кода от имени заданной учётной записи и возможностью получения к ней права доступа $impersonate_r$.

**Утверждение 2.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $u, u' \in U$. Тогда предикат $can\_act\_as(u, u', G)$ истинен тогда и только тогда, когда истинен предикат $can\_become(u, u', G)$.

***Доказательство.*** Пусть $s \notin S$.

Пусть истинен предикат $can\_act\_as(u, u', G)$. В соответствии со способом задания множества $Gr_e$ владелец сущности может предоставить к ней любые права доступа. Отсюда, учитывая, что $owner(u') = u'$, следует, что $u' \in Gr(u', impersonate_r)$. Это означает, что в состоянии $G$ выполнены условия применения $op = grant\_right(s, u, u', impersonate_r, \texttt{yes})$ в случае, если субъект-сессия $s$ функционирует от имени учётной записи пользователя $u'$. В этом случае, по определению 11, истинен предикат $can\_execute\_as(u, u', op, G)$. Значит, существуют инициированные извне прави-

ла $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $vestige(G, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$, и существует $i \in \{2, \ldots, m\}$, что $\widehat{op}_i = op$ и $\widehat{user}_{i-1}(s) = u'$. В соответствии с замечанием 15 последовательность $vestige(G, op_1\langle s\rangle \ldots, op_k\langle s\rangle)$ не включает правила, условия применения которых не выполняются. Значит, выполнены условия применения правила $op = grant\_right(s', u, u', impersonate_r, \texttt{yes})$, откуда $(u, u', impersonate_r) \in \widehat{R}_{e_i}$. Из монотонности правил преобразования состояний системы следует, что $(u, u', impersonate_r) \in R_{e_k}$, а значит, по определению истинен предикат $can\_become(u, u', G)$.

Пусть истинен предикат $can\_become(u, u', G)$. По определению 12 существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$ и $(u, u', impersonate_r) \in R_{e_k}$. Пусть для правила $op \in OP_{\text{ext}}$ в состоянии $G$ выполнены условия его применения субъект-сессией, функционирующей от имени учётной записи пользователя $u'$. Легко заметить, что в результате применения внешних правил преобразований значение функции $user\_stack$ не изменяется. Таким образом, $user\_stack_k(s) = user\_stack_1(s) = (u)$, откуда $user_k(s) = u$. Так как $(u, u', impersonate_r) \in R_{e_k}$, в состоянии $G_k$ выполнены условия применения правил $op_{k+1}\langle s\rangle = switch(s, u')$ и $user_{k+1}(s) = u'$.

Покажем, что в состоянии $G_{k+1}$ выполнены условия применения правила $op$.

Во всех внешних правилах, за исключением $create\_session$, $create\_trigger$, $create\_procedure$ и $create\_container$, нет проверок на отсутствие элементов в состоянии системы. Отсюда, в силу монотонности правил преобразования состояний, в состоянии $G_{k+1}$ выполнены условия применения $op$, а следовательно, по определению истинен предикат $can\_execute\_as(u, u', op, G)$.

Согласно предположению 5, применение правила $create\_session$ не может быть инициировано ранее созданной субъект-сессией, а следовательно, в рамках определения 10 правило $op$ не может иметь вид $create\_session$.

Пусть $op \prec create\_container(\ldots, c, \ldots)$. Если $c \notin C_{k+1}$, то в силу монотонности правил в состоянии $G_{k+1}$ выполнены условия применения правила $op$. Пусть $c \in C_{k+1}$. Тогда, согласно предположению 6, существует $i \in \{1, \ldots, k\}$, такое, что $op_i\langle s\rangle \prec create\_container(\ldots, c, \ldots)$. Заменим в правилах преобразования состояний $op_i\langle s\rangle, \ldots, op_{k+1}\langle s\rangle$ все случаи использования контейнера $c$ на $c' \notin C_{i-1}$. Очевидно, что в этом случае $(u, u', impersonate_r) \in R_{e_{k+1}}$ и $c \notin C_{k+1}$, тогда в силу монотонности правил в состоянии $G_{k+1}$ выполнены условия применения правила $op$, а следовательно, истинен предикат $can\_execute\_as(u, u', op, G)$.

Проводя аналогичные рассуждения для случаев $create\_procedure$ и $create\_trigger$, можно показать, что предикат $can\_execute\_as(u, u', op, G)$ также будет истинен.

Следовательно, предикат $can\_execute\_as(u, u', op, G)$ истинен для любого правила $op \in OP_{\text{ext}}$, для которого в состоянии $G$ выполнены условия применения субъект-сессией, функционирующей от имени учётной записи пользователя $u'$, а значит, по определению истинен предикат $can\_act\_as(u, u', G)$. ■

Следовательно, для определения условий истинности предиката $can\_act\_as(u, u', G)$ необходимо и достаточно обосновать условия истинности предиката $can\_become(u, u', G)$.

### 4.2. Достаточные условия выполнения *SQL*-кода от имени заданной учётной записи пользователя

Покажем, что если в начальном состоянии системы $G_0$ в результате применения извне правил субъект-сессией, созданной пользователем $u$, были выполнены правила

вида $grant\_right$ и $add\_member$ от имени учётной записи пользователя $u'$, то предикат $can\_act\_as(u, u', G_0)$ истинен.

**Утверждение 3.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*, OP)$, $s \notin S$, $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, $op_1\langle s\rangle = create\_session(s, u)$, $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$ и существует $i \in \{2, \ldots, m\}$, что $\widehat{op}_i \prec grant\_right$ или $\widehat{op}_i \prec add\_member$ и $\widehat{user}_{i-1}(s) = u'$. Тогда истинен предикат $can\_act\_as(u, u', G_0)$.

***Доказательство.*** Возможны два случая: 1) $\widehat{[s]}_{i-1} = \varnothing$ и 2) $\widehat{[s]}_{i-1} = o$, где $o \in O_{p_{i-1}} \cup O_{t_{i-1}}$.

Рассмотрим первый случай. Если $\widehat{[s]}_{i-1} = \varnothing$, то применение правила $\widehat{op}_i$ было инициировано извне субъект-сессией $s$. Следовательно, существует $j \in \{1, \ldots, k\}$, такое, что $op_j\langle s\rangle = \widehat{op}_i$. Так как $\widehat{user}_{i-1}(s) = u'$, имеем $user_{j-1}(s) = u'$. Положим $op_j\langle s\rangle = grant\_right(s, u, u', impersonate_r, \texttt{no})$. Так как $(u', u', impersonate_r) \in R_{e_{j-1}}$, в состоянии $G_{j-1}$ выполнены условия применения правила $op_j\langle s\rangle$ субъект-сессией, функционирующей от имени учётной записи пользователя $u'$, откуда $(u, u', impersonate_r) \in R_{e_j}$. Тогда по определению истинен предикат $can\_become(u, u', G_0)$ и, в соответствии с утверждением 2, истинен предикат $can\_act\_as(u, u', G_0)$.

Рассмотрим второй случай. Возможны следующие два варианта: 2.1) $o \in O_{p_0} \cup O_{t_0}$ и 2.2) $o \notin O_{p_0} \cup O_{t_0}$.

Рассмотрим случай 2.1. В соответствии с предположением 7 в начальном состоянии $\Sigma(G^*, OP, G_0)$ для каждого $o \in O_{p_0} \cup O_{t_0}$ и каждого $op \in operations_0(o)$ выполняется $op \not\prec grant\_right$ и $op \not\prec add\_member$. Следовательно, существует $i' \in \{2, \ldots, i-1\}$, такое, что либо $\widehat{op}_{i'} \prec alter\_procedure$, либо $\widehat{op}_{i'} \prec alter\_trigger$. Выберем значение $i'$ максимальным. В соответствии с предположением 6 применение правил $alter\_procedure$ и $alter\_trigger$ возможно только извне. Следовательно, существует $j \in \{1, \ldots, k\}$, такое, что $op_j\langle s\rangle = \widehat{op}_{i'}$. Возможны два случая: $o \in O_{p_0}$ и $o \in O_{t_0}$. В первом случае $op_j\langle s\rangle = alter\_procedure(\widetilde{op}_1, \ldots, \widetilde{op}_l)$ и существует $j' \in \{1, \ldots, l\}$, такое, что $\widetilde{op}_{j'} = \widehat{op}_i$. Заменим в правиле $op_j\langle s\rangle$ в последовательности $(\widetilde{op}_1, \ldots, \widetilde{op}_l)$ правило $\widetilde{op}_{j'}$ на $grant\_right(s, u, u', impersonate_r, \texttt{no})$. Из максимальности выбора $i'$ следует, что в состоянии $\widehat{G}_{j-1}$ верно равенство $\widehat{operations}_{j-1}(o) = (\widetilde{op}_1, \ldots, \widetilde{op}_{j'-1}, grant\_right(s, u, u', impersonate_r, \texttt{no}), \widetilde{op}_{j'+1}, \ldots, \widetilde{op}_l)$, откуда $\widehat{op}_i \prec grant\_right(s, u, u', impersonate_r, \texttt{no})$ и $\widehat{user}_{i-1}(s) = u'$. Так как $(u', u', impersonate_r) \in \widehat{R}_{e_{i-1}}$, в состоянии $\widehat{G}_{i-1}$ выполнены условия применения правила $\widehat{op}_i$ и $(u, u', impersonate_r) \in \widehat{R}_{e_i}$, а значит, в силу монотонности правил преобразования состояний, $(u, u', impersonate_r) \in R_{e_k}$. Тогда по определению истинен предикат $can\_become(u, u', G_0)$ и, по утверждению 2, истинен предикат $can\_act\_as(u, u', G_0)$. Случай $o \in O_{t_0}$ обосновывается аналогично.

Рассмотрим случай 2.2. В соответствии с предположением 6 применение правил $create\_procedure$ и $create\_trigger$ возможно только извне. Так как $o \notin O_{p_0} \cup O_{t_0}$, существует $j \in \{1, \ldots, k\}$, такое, что $o \notin O_{p_{j-1}} \cup O_{t_{j-1}}$, $o \notin O_{p_j} \cup O_{t_j}$ и либо $op_j\langle s\rangle \prec create\_procedure$, либо $op_j\langle s\rangle \prec create\_trigger$. Возможны следующие два случая: 1) $\widehat{operations}_{i-1}(o) \neq operations_j(o)$; 2) $\widehat{operations}_{i-1}(o) = operations_j(o)$. Если $\widehat{operations}_{i-1}(o) \neq operations_j(o)$, то, проводя рассуждения, аналогичные случаю 2.1, можно показать, что истинен предикат $can\_act\_as(u, u', G_0)$. Рассмотрим второй случай: $\widehat{operations}_j(o) = (\widetilde{op}_1, \ldots, \widetilde{op}_l)$, тогда существует $j' \in \{1, \ldots, l\}$, такое, что $\widetilde{op}_{j'} = \widehat{op}_i$. Заменим в правиле $op_j\langle s\rangle$ в последовательности $(\widetilde{op}_1, \ldots, \widetilde{op}_l)$ правило $\widetilde{op}_{j'}$ на $grant\_right(s, u, u', impersonate_r, \texttt{no})$. Так как $\widehat{operations}_{i-1}(o) = operations_j(o)$, получим $\widehat{op}_i \prec grant\_right(s, u, u', impersonate_r, \texttt{no})$ и $\widehat{user}_{i-1}(s) = u'$. В соответствии

со способом задания $(u', u', impersonate_r) \in \widehat{R}_{e_{i-1}}$, и значит, в состоянии $\widehat{G}_{i-1}$ выполнены условия применения правила $\widehat{op}_i$ и $(u, u', impersonate_r) \in \widehat{R}_{e_i}$, откуда, в силу монотонности правил преобразования состояний, $(u, u', impersonate_r) \in R_{e_k}$. Тогда по определению истинен предикат $can\_become(u, u', G_0)$ и в соответствии с утверждением 2 истинен предикат $can\_act\_as(u, u', G_0)$. ■

## 5. Ролевые цепочки получения прав доступа

Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $p \in P$, $r \in R$, $e \in E$, $\alpha_r \in R_r$. Будем использовать обозначение $(p, r, e, \alpha_r) \lessdot G$ в случае, если $(p, r, alter_r) \in R_e$ и $(r, e, \alpha_r) \in R_e$.

Пусть $u \in U$, тогда будем использовать обозначение $(u, e, \alpha_r) \lhd G$ в случае, если существуют роли $r_1, \ldots, r_k \in R$, где $k \geqslant 1$, такие, что $(u, r_1, r_2, alter_r) \lessdot G$, для каждого $i = 1, \ldots, k-2$ выполняется $(r_i, r_{i+1}, r_{i+2}, alter_r) \lessdot G$ и $(r_{k-1}, r_k, e, \alpha_r) \lessdot G$. В последнем случае будем использовать также обозначение $(u, r_k, e, \alpha_r) \lhd G$. В случае $(u, e, \alpha_r) \lhd G$ будем говорить, что учётная запись пользователя $u$ обладает «ролевой цепочкой» получения права доступа $\alpha_r$ к сущности $e$. Если $(u, r, e, \alpha_r) \lhd G$, то будем говорить, что учётная запись пользователя $u$ обладает «ролевой цепочкой» получения права доступа $\alpha_r$ к сущности $e$ через роль $r$.

Будем использовать обозначение $(u, e, \alpha_r) \blacktriangleleft G$ в случае, если существуют роли $r_1, \ldots, r_k \in R$, где $k \geqslant 1$, такие, что $(u, r_1, r_2, alter_r) \lessdot G$, для каждого $i = 1, \ldots, k-2$ выполняется $(r_i, r_{i+1}, r_{i+2}, alter_r) \lessdot G$ и $(r_{k-1}, r_k, alter_r) \in R_e$ и $r_k \in Gr(e, \alpha_r)$. Если $(u, e, \alpha_r) \blacktriangleleft G$, то будем говорить, что учётная запись пользователя $u$ обладает «ролевой цепочкой» предоставления права доступа $\alpha_r$ к сущности $e$. Очевидно, что если $(u, e, \alpha_r) \blacktriangleleft G$, то $(u, e, \alpha_r) \lhd G$.

В силу монотонности правил преобразования состояний справедливы следующие замечания.

**Замечание 16.** Если $G$ — состояние системы $\Sigma(G^*, OP)$, $p \in P$, $r \in R$, $e \in E$, $\alpha_r \in R_r$, $(p, r, e, \alpha_r) \lessdot G$, $op_1, \ldots, op_k \in OP_{\text{ext}}$, где $k \geqslant 1$, $G_k = G(op_1, \ldots, op_k)$, то $(p, r, e, \alpha_r) \lessdot G_k$.

**Замечание 17.** Если $G$ — состояние системы $\Sigma(G^*, OP)$, $u \in U$, $e \in E$, $\alpha_r \in R_r$, $(u, e, \alpha_r) \lhd G$, $op_1, \ldots, op_k \in OP_{\text{ext}}$, где $k \geqslant 1$, $G_k = G(op_1, \ldots, op_k)$, то $(u, e, \alpha_r) \lhd G_k$.

**Замечание 18.** Если $G$ — состояние системы $\Sigma(G^*, OP)$, $u \in U$, $e \in E$, $\alpha_r \in R_r$, $(u, e, \alpha_r) \blacktriangleleft G$, $op_1, \ldots, op_k \in OP_{\text{ext}}$, где $k \geqslant 1$, $G_k = G(op_1, \ldots, op_k)$, то $(u, e, \alpha_r) \blacktriangleleft G_k$.

**Утверждение 4.** Если $G$ — состояние системы $\Sigma(G^*, OP)$, $u \in U$, $r \in R$, $e \in E$, $\alpha_r \in R_r$, $(u, r, e, \alpha_r) \lessdot G$, $s \in S$, $user(s) = u$, $G_1 = G(add\_member(s, r, u))$, то $(u, e, \alpha_r) \in \in R_{e_1}$.

***Доказательство.*** Следует из того, что в состоянии $G$ выполнены условия применения правила $add\_member(s, r, u)$ от имени учётной записи $u$ и при этом $(r, e, \alpha_r) \in R_e$. ■

Докажем, что если в результате применения извне правил субъект-сессией, созданной пользователем $u$, у принципала $p$ появится право доступа $alter_r$ к роли $r$, то в начальном состоянии $G_0$ существует учётная запись пользователя $u'$, обладающая ролевой цепочкой получения права доступа $\alpha_r$ к сущности $e$ через роль $r$, и $u$ имеет возможность выполнения от её имени произвольного SQL-кода.

**Утверждение 5.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*, OP)$, $u \in U$, $p \in P$, $r \in R$, $e \in E_0$, $\alpha_r \in R_r$, $(r, e, \alpha_r) \in R_{e_0}$, $s \notin S_0$, $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G_0(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$, $(p, r, alter_r) \notin R_{e_0}$ и

$(p, r, alter_r) \in R_{e_k}$. Тогда существует $u' \in U$, такое, что предикат $can\_act\_as(u, u', G_0)$ истинен и $(u', r, e, \alpha_r) \lhd G_0$.

***Доказательство.*** Пусть $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$.

Так как $(p, r, alter_r) \notin R_{e_0}$ и $(p, r, alter_r) \in R_{e_k}$, существует $i \in \{1, \ldots, m\}$, такое, что $(p, r, alter_r) \notin \widehat{R}_{e_{i-1}}$ и $(p, r, alter_r) \in \widehat{R}_{e_i}$. Заметим, что в соответствии с замечанием 15 в состоянии $\widehat{G}_{i-1}$ выполнены условия применения правила $\widehat{op}_i$. Пусть $l$ — количество правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_i)$. Доказательство проведём индукцией по $l$.

Пусть $l = 1$. Обозначим $\widehat{user}_{i-1}(s)$ как $u'$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u', G_0)$. Так как $l = 1$, $\widehat{op}_i$ — единственное правило вида $grant\_right$ и $add\_member$, откуда $\widehat{R}_{e_{i-1}} = R_{e_0}$.

Если $\widehat{op}_i \prec grant\_right$, то $\widehat{op}_i \prec grant\_right(s, \ldots, e', alter_r, \ldots)$, где $e' \in \{r, c_r\}$ и $u' \in \widehat{Gr}_{i-1}(e', alter_r)$, а значит, $(u', e', alter_r) \in \widehat{R}_{e_{i-1}}$. Заметим, что $r \leqslant e'$, откуда $(u', r, alter_r) \in \widehat{R}_{e_{i-1}}$. Ранее было показано, что $\widehat{R}_{e_{i-1}} = R_{e_0}$, а значит, $(u', r, alter_r) \in R_{e_0}$. По условию $(r, e, \alpha_r) \in R_{e_0}$, следовательно, $(u', r, e, \alpha_r) \lhd G_0$.

Пусть $\widehat{op}_i \prec add\_member$, тогда $p \in U$, $\widehat{op}_i = add\_member(s, r', p)$, где $r' \in R$, $(u', r', alter_r) \in \widehat{R}_{e_{i-1}}$ и $(r', r, alter_r) \in \widehat{R}_{e_{i-1}}$. Так как $\widehat{R}_{e_{i-1}} = R_{e_0}$, имеем $(u', r', alter_r) \in$ $\in R_{e_0}$ и $(r', r, alter_r) \in R_{e_0}$. По условию $(r, e, \alpha_r) \in R_{e_0}$, а значит, $(u', r, e, \alpha_r) \lhd G_0$.

Пусть условия утверждения выполняются для всех $l < N$, где $N > 1$. Покажем, что оно верно и при $l = N$. Обозначим $\widehat{user}_{i-1}(s)$ как $u''$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u'', G_0)$.

Если $\widehat{op}_i \prec grant\_right$, то $\widehat{op}_i \prec grant\_right(s, \ldots, e', alter_r, \ldots)$, где $e' \in \{r, c_r\}$ и $u'' \in \widehat{Gr}_{i-1}(e', alter_r)$, а значит, $(u'', e', alter_r) \in \widehat{R}_{e_{i-1}}$. Заметим, что $r \leqslant e'$, откуда $(u'', r, alter_r) \in \widehat{R}_{e_{i-1}}$. Возможны случаи: 1) $(u'', r, alter_r) \in R_{e_0}$; 2) $(u'', r, alter_r) \notin R_{e_0}$.

С л у ч а й 1. Положим $u' = u''$. По условию $(r, e, \alpha_r) \in R_{e_0}$, откуда $(u', r, e, \alpha_r) \lhd G_0$.

С л у ч а й 2. Так как $(u'', r, alter_r) \notin R_{e_0}$ и $(u'', e', alter_r) \in \widehat{R}_{e_{i-1}}$, существует $j \in \{1, \ldots, i-1\}$, что $(u'', r, alter_r) \notin \widehat{R}_{e_{j-1}}$ и $(u'', r, alter_r) \in \widehat{R}_{e_j}$. Так как $\widehat{op}_i \prec grant\_right$ и $j < i$, для $u''$, $r$ и $e$ выполнено предположение индукции с количеством правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$, меньшим чем $N$. Тогда по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и $(u', r, e, \alpha_r) \lhd G_0$.

Пусть $\widehat{op}_i \prec add\_member$. Тогда $p \in U$, $\widehat{op}_i = add\_member(s, r', p)$, где $r' \in R$, $(u'', r', alter_r) \in \widehat{R}_{e_{i-1}}$ и $(r', r, alter_r) \in \widehat{R}_{e_{i-1}}$. Возможны случаи: 1) $(u'', r', alter_r) \in R_{e_0}$ и $(r', r, alter_r) \in R_{e_0}$; 2) $(u'', r', alter_r) \notin R_{e_0}$ и $(r', r, alter_r) \in R_{e_0}$; 3) $(r', r, alter_r) \notin R_{e_0}$.

С л у ч а й 1. Положим $u' = u''$. По условию $(r, e, \alpha_r) \in R_{e_0}$, следовательно, $(u', r, e, \alpha_r) \lhd G_0$.

С л у ч а й 2. Так как $(u'', r', alter_r) \notin R_{e_0}$ и $(u'', r', alter_r) \in \widehat{R}_{e_{i-1}}$, существует $j \in \{1, \ldots, i-1\}$, такое, что $(u'', r', alter_r) \notin \widehat{R}_{e_{j-1}}$ и $(u'', r', alter_r) \in \widehat{R}_{e_j}$. Так как $\widehat{op}_i \prec add\_member$ и $(r', r, alter_r) \in R_{e_0}$, для $u''$, $r'$ и $r$ выполнено предположение индукции с количеством правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$, меньшим чем $N$. Тогда по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и $(u', r', r, alter_r) \lhd G_0$. По условию $(r, e, \alpha_r) \in R_{e_0}$, откуда $(u', r, e, \alpha_r) \lhd G_0$.

С л у ч а й 3. Так как $(r', r, alter_r) \notin R_{e_0}$ и $(r', r, alter_r) \in \widehat{R}_{e_{i-1}}$, существует $j \in \{1, \ldots, i-1\}$, такое, что $(r', r, alter_r) \notin \widehat{R}_{e_{j-1}}$ и $(r', r, alter_r) \in \widehat{R}_{e_j}$. Так как $\widehat{op}_i \prec add\_member$, для $r'$, $r$ и $e$ выполнено предположение индукции с количеством

правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$, меньшим чем $N$. Тогда по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и $(u', r, e, \alpha_r) \lhd G_0$. ■

## 6. Необходимые и достаточные условия предоставления прав доступа

Введём предикат $can\_grant\_right(u, e, \alpha_r, G)$, истинный тогда, когда в результате применения правил субъект-сессией, созданной пользователем $u$, учётная запись пользователя $u$ получает возможность предоставлять право доступа $\alpha_r$ к сущности $e$.

**Определение 13.** Пусть $G$ — состояние системы $\Sigma(G^*, OP)$, $u \in U$, $e \in E$, $\alpha_r \in \in R_r$, $s \notin S$. Определим предикат $can\_grant\_right(u, e, \alpha_r, G)$, истинный тогда и только тогда, когда существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$ и $u \in Gr_k(e, \alpha_r)$.

Покажем, что если в результате применения извне правил субъект-сессией, созданной пользователем $u$, у принципала $p$ появляется право на предоставление права доступа $\alpha_r$ к сущности $e$, то в начальном состоянии $G_0$ существует учётная запись пользователя $u'$, имеющая право на предоставление этого права к $e$, и $u$ имеет возможность выполнения от её имени произвольного SQL-кода.

**Утверждение 6.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*, OP)$, $u \in U$, $e \in E_0$, $\alpha_r \in R_r$, $s \notin S_0$, $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G_0(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$, $p \in P$, $p \notin Gr_0(e, \alpha_r)$ и $p \in Gr_k(e, \alpha_r)$. Тогда существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и либо $u' \in Gr_0(e, \alpha_r)$, либо $(u', e, \alpha_r) \blacktriangleleft G_0$.

***Доказательство.*** Пусть $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$.

Так как $p \notin Gr_0(e, \alpha_r)$ и $p \in Gr_k(e, \alpha_r)$, существует $i \in \{1, \ldots, m\}$, такое, что $p \notin \widehat{Gr}_{i-1}(e, \alpha_r)$ и $p \in \widehat{Gr}_i(e, \alpha_r)$. Пусть $l$ — количество правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_i)$. Доказательство проведём индукцией по $l$.

Пусть $l = 1$. Обозначим $\widehat{user}_{i-1}(s)$ как $u'$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u', G_0)$. Заметим, что в соответствии с замечанием 15 в состоянии $\widehat{G}_{i-1}$ выполнены условия применения правила $\widehat{op}_i$. Так как $l = 1$, $\widehat{op}_i$ — единственное правило вида $grant\_right$ и $add\_member$, откуда $\widehat{R}_{e_{i-1}} = R_{e_0}$ и $\widehat{Gr}_{e_{i-1}} = Gr_{e_0}$.

Пусть $\widehat{op}_i \prec grant\_right$, тогда $\widehat{op}_i = grant\_right(s, p, e, \alpha_r, \texttt{yes})$ и $u' \in \widehat{Gr}_{i-1}(e, \alpha_r)$. Так как $\widehat{Gr}_{e_{i-1}} = Gr_{e_0}$, получаем $u' \in Gr_0(e, \alpha_r)$, а значит, выполнено условие утверждения.

Пусть $\widehat{op}_i \prec add\_member$, тогда $p \in U$, $\widehat{op}_i = add\_member(s, r, p)$, где $r \in R$, $(u', r, alter_r) \in \widehat{R}_{e_{i-1}}$ и $r \in \widehat{Gr}_{i-1}(e, \alpha_r)$. Так как $\widehat{R}_{e_{i-1}} = R_{e_0}$ и $\widehat{Gr}_{e_{i-1}} = Gr_{e_0}$, имеем $(u', r, alter_r) \in R_{e_0}$ и $r \in Gr_0(e, \alpha_r)$. Следовательно, $(u', e, \alpha_r) \blacktriangleleft G_0$, а значит, выполнено условие утверждения.

Пусть условия утверждения выполняются для всех $l < N$, где $N > 1$. Покажем, что оно верно и при $l = N$. Обозначим $\widehat{user}_{i-1}(s)$ как $u''$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u'', G_0)$.

Пусть $\widehat{op}_i \prec grant\_right$, тогда $\widehat{op}_i \prec grant\_right(s, p, e, \alpha_r, \texttt{yes})$ и $u'' \in \widehat{Gr}_{i-1}(e, \alpha_r)$. Возможны два случая: 1) $u'' \in Gr_0(e, \alpha_r)$; 2) $u'' \notin Gr_0(e, \alpha_r)$.

С л у ч а й 1. Если $u'' \in Gr_0(e, \alpha_r)$, то, положив $u' = u''$, видим, что выполнены условия утверждения.

С л у ч а й 2. Так как $u'' \notin Gr_0(e,\alpha_r)$ и $u'' \in \widehat{Gr}_{i-1}(e,\alpha_r)$, существует $j \in \{1, \ldots, i-1\}$, такое, что $u'' \notin \widehat{Gr}_{j-1}(e,\alpha_r)$ и $u'' \in \widehat{Gr}_j(e,\alpha_r)$. Так как $\widehat{op}_i \prec grant\_right$ и $j < i$, в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$ количество правил вида $grant\_right$ и $add\_member$ меньше $N$. Значит, по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и либо $u' \in Gr_0(e,\alpha_r)$, либо $(u', e, \alpha_r) \blacktriangleleft G_0$.

Пусть $\widehat{op}_i \prec add\_member$, тогда $p \in U$ и $\widehat{op}_i \prec add\_member(s, r, p)$, где $r \in R$, $(u'', r, alter_r) \in \widehat{R}_{e_{i-1}}$ и $r \in \widehat{Gr}_{i-1}(e,\alpha_r)$. Возможны три случая: 1) $(u'', r, alter_r) \in R_{e_0}$ и $r \in Gr_0(e,\alpha_r)$; 2) $r \notin Gr_0(e,\alpha_r)$; 3) $(u'', r, alter_r) \notin R_{e_0}$ и $r \in Gr_0(e,\alpha_r)$.

С л у ч а й 1. Легко заметить, что $(u'', e, \alpha_r) \blacktriangleleft G_0$. Положив $u' = u''$, видим, что выполнены условия утверждения.

С л у ч а й 2. Так как $r \notin Gr_0(e,\alpha_r)$ и $r \in \widehat{Gr}_{i-1}(e,\alpha_r)$, существует $j \in \{1, \ldots, i-1\}$, что $r \notin \widehat{Gr}_{j-1}(e,\alpha_r)$ и $r \in \widehat{Gr}_j(e,\alpha_r)$. Так как $\widehat{op}_i \prec add\_member$ и $j < i$, в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$ количество правил вида $grant\_right$ и $add\_member$ меньше $N$. Значит, по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и либо $u' \in Gr_0(e,\alpha_r)$, либо $(u', e, \alpha_r) \blacktriangleleft G_0$.

С л у ч а й 3. Так как $r \in Gr_0(e,\alpha_r)$, верно $(r, e, \alpha_r) \in R_{e_0}$. Кроме того, так как $(u'', r, alter_r) \in \widehat{R}_{e_{i-1}}$, в силу монотонности правил преобразования состояний, $(u'', r, alter_r) \in R_{e_k}$. Заметим, что так как $(u'', r, alter_r) \notin R_{e_0}$, $(u'', r, alter_r) \in R_{e_k}$ и $(r, e, \alpha_r) \in R_{e_0}$, выполнены условия утверждения 5. Следовательно, существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и $(u', r, e, \alpha_r) \lhd G_0$. Легко заметить, что ввиду $(u', r, e, \alpha_r) \lhd G_0$ и $r \in Gr_0(e,\alpha_r)$ выполнено $(u', e, \alpha_r) \blacktriangleleft G_0$. ■

Покажем, что справедливы следующие необходимые и достаточные условия истинности предиката $can\_grant\_right$.

**Теорема 1.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*, OP)$, $u \in U$, $e \in E_0$, $\alpha_r \in R_r$. Предикат $can\_grant\_right(u, e, \alpha_r, G_0)$ истинен тогда и только тогда, когда существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и выполнено одно из условий:

*Условие 1.* $u' \in Gr_0(e,\alpha_r)$.

*Условие 2.* $(u', e, \alpha_r) \blacktriangleleft G_0$.

***Доказательство.*** Обоснуем достаточность условий утверждения.

Пусть выполнено условие 1. Положим $op = grant\_right(s, u, e, \alpha_r, \texttt{yes})$. Так как $u' \in Gr_0(e,\alpha_r)$, то в $G_0$ выполнены условия применения правила $op$ субъект-сессией, функционирующей от имени учётной записи пользователя $u'$. Тогда в соответствии с определением 11 из истинности предиката $can\_act\_as(u, u', G_0)$ следует, что истинен предикат $can\_execute\_as(u, u', op, G_0)$. Отсюда по определению 10 существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$ и $s \notin S_0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$, и существует $i \in \{2, \ldots, m\}$ такое, что $\widehat{op}_i = op$ и $\widehat{user}_{i-1}(s) = u'$. Заметим, что в соответствии с замечанием 15 последовательность $\widehat{op}_1, \ldots, \widehat{op}_m$ не включает правила, условия применения которых не выполняются, поэтому выполнены условия применения правила $\widehat{op}_i$ в состоянии $\widehat{G}_{i-1}$, а значит, $u \in \widehat{Gr}_i(e,\alpha_r)$. Пусть $G_k = G_0(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$. В силу монотонности правил преобразования состояний $u \in Gr_k(e,\alpha_r)$. Значит, по определению 13 истинен предикат $can\_grant\_right(u, e, \alpha_r, G_0)$.

Пусть выполнено условие 2. Согласно утверждению 2, из истинности предиката $can\_act\_as(u, u', G_0)$ следует истинность предиката $can\_become(u, u', G_0)$. Отсюда, по определению 12, существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$,

где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s,u)$, $G_k = G_0(op_1\langle s\rangle,\ldots,op_k\langle s\rangle)$ и $(u,u',impersonate_r) \in R_{e_k}$. Положим $op_{k+1}\langle s\rangle = switch(s,u')$. Заметим, что в состоянии $G_k$ выполнены условия применения правила $op_{k+1}\langle s\rangle$ и $user_{k+1}(s) = u'$. В соответствии с замечанием 18 $(u',e,\alpha_r) \blacktriangleleft G_{k+1}$. Значит, существуют роли $r_1,\ldots,r_n \in R$, где $n \geqslant 1$, такие, что $(u',r_1,r_2,alter_r) \lessdot G_{k+1}$, для каждого $i = 1,\ldots,n-2$ выполняется $(r_i,r_{i+1},r_{i+2},alter_r) \lessdot G_{k+1}$, и $(r_{n-1},r_n,alter_r) \in R_{e_{k+1}}$, и $r_n \in Gr_{k+1}(e,\alpha_r)$. Положим $op_{k+2}\langle s\rangle = add\_member(s,r_1,u'),\ldots,\ op_{k+n}\langle s\rangle = add\_member(s,r_{n-1},u')$. Пусть $G_{k+n} = G_{k+1}(op_{k+2}\langle s\rangle,\ \ldots,op_{k+n}\langle s\rangle)$. Так как $user_{k+1}(s) = u'$, то, учитывая, что $(u',r_1,r_2,alter_r) \lessdot G_{k+1}$, в соответствии с утверждением 4 $(u',r_2,alter_r) \in R_{e_{k+2}}$. В силу замечания 16, $(r_1,r_2,r_3,\ alter_r) \lessdot G_{k+2}$, откуда $(r_2,r_3,alter_r) \in R_{e_{k+2}}$, а значит, $(u',r_2,r_3,alter_r) \lessdot G_{k+2}$. Проводя аналогичные рассуждения, легко показать, что $(u',r_n,alter_r) \in R_{e_{k+n}}$. Положим $op_{k+n+1}\langle s\rangle = add\_member(s,r_n,u)$. Пусть $G_{k+n+1} = G_{k+n}(op_{k+n+1}\langle s\rangle)$. Заметим, что в состоянии $G_{k+n}$ выполнены условия применения правила, а значит, $r_n \in UA_{k+n+1}(u)$. Так как $r_n \in Gr_{k+1}(e,\alpha_r)$, то в силу монотонности правил преобразования состояний $r_n \in Gr_{k+n+1}(e,\alpha_r)$. Тогда $u \in Gr_{k+n+1}(e,\alpha_r)$, а значит, по определению 13 истинен предикат $can\_grant\_right(u,e,\alpha_r,G_0)$.

Обоснуем необходимость выполнения условий утверждения. Пусть истинен предикат $can\_grant\_right(u,e,\alpha_r,G_0)$, тогда существуют инициированные извне правила $op_1\langle s\rangle,\ldots,op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s,u)$, $G_k = G(op_1\langle s\rangle,\ldots,op_k\langle s\rangle)$ и $u \in Gr_k(e,\alpha_r)$. Заметим, что так как $(u,u,impersonate_r) \in R_{e_0}$, истинен предикат $can\_become(u,u,G_0)$, а значит, согласно утверждению 2, истинен предикат $can\_act\_as(u,u,G_0)$. Если $u \in Gr_0(e,\alpha_r)$, то так как истинен предикат $can\_become(u,u,G_0)$, выполнено условие 1 утверждения. Если $u \notin Gr_0(e,\alpha_r)$, то выполнены условия утверждения 6 и существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u,u',G_0)$ и либо $u' \in Gr_0(e,\alpha_r)$, либо $(u',e,\alpha_r) \blacktriangleleft G_0$. ■

Из утверждения 6 и теоремы 1 очевидна справедливость следующего следствия.

**Следствие 1.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*,OP)$, $u \in U$, $e \in E_0$, $\alpha_r \in R_r$, $s \notin S_0$, $op_1\langle s\rangle,\ldots,op_k\langle s\rangle \in OP_{\text{ext}}$, где $op_1\langle s\rangle = create\_session(s,u)$, $G_k = G_0(op_1\langle s\rangle,\ldots,op_k\langle s\rangle)$, $p \in P$, $p \notin Gr_0(e,\alpha_r)$ и $p \in Gr_k(e,\alpha_r)$. Тогда истинен предикат $can\_grant\_right(u,e,\alpha_r,G_0)$.

## 7. Необходимые и достаточные условия получения прав доступа

Введём предикат $can\_get\_right(u,e,\alpha_r,G)$, который будет истинен, когда в результате применения правил субъект-сессией, созданной пользователем $u$, учётная запись пользователя $u$ может получить право доступа $\alpha_r$ к сущности $e$.

**Определение 14.** Пусть $G$ — состояние системы $\Sigma(G^*,OP)$, $u \in U$, $e \in E$, $\alpha_r \in R_r$, $s \notin S$ и $(u,e,\alpha_r) \notin R_e$. Определим предикат $can\_get\_right(u,\ e,\alpha_r,G)$, который будет истинным тогда и только тогда, когда существуют инициированные извне правила $op_1\langle s\rangle,\ldots,op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s,u)$, $G_k = G(op_1\langle s\rangle,\ldots,op_k\langle s\rangle)$ и $(u,e,\alpha_r) \in R_{e_k}$.

Обоснуем необходимые и достаточные условия получения учётной записью пользователя прав доступа к сущности в начальном состоянии системы $G_0$.

**Теорема 2.** Пусть $G_0$ — начальное состояние системы $\Sigma(G^*,OP)$, $u \in U$, $e \in E_0$, $\alpha_r \in R_r$ и $(u,e,\alpha_r) \notin R_{e_0}$. Предикат $can\_get\_right(u,e,\alpha_r,G_0)$ истинен тогда и только тогда, когда существуют $u' \in U$, $e' \in E_0$, такие, что предикат $can\_act\_as(u,u',G_0)$ истинен, $e \leqslant e'$ и выполнено одно из следующих условий.

У с л о в и е 1. $u' \in Gr_0(e',\alpha_r)$.

У с л о в и е 2. $(u', e', \alpha_r) \lhd G_0$.

***Доказательство.*** Достаточность.

Пусть выполнено условие 1. Положим $op = grant\_right(s, u, e', \alpha_r, \mathtt{no})$. Так как $u' \in Gr_0(e', \alpha_r)$, в $G_0$ выполнены условия применения правила $op$ субъект-сессией, функционирующей от имени учётной записи пользователя $u'$. Тогда в соответствии с определением 11 из истинности предиката $can\_act\_as(u, u', G_0)$ следует, что истинен предикат $can\_execute\_as(u, u', op, G_0)$. Значит, существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $s \notin S_0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$ и существует $i \in \{2, \ldots, m\}$, что $\widehat{op}_i = op$ и $\widehat{user}_{i-1}(s) = u'$. Заметим, что в соответствии с замечанием 15 последовательность $\widehat{op}_1, \ldots, \widehat{op}_m$ не включает правила, условия применения которых не выполняются, поэтому выполнены условия применения правила $\widehat{op}_i$ в состоянии $\widehat{G}_{i-1}$, а значит, $(u, e', \alpha_r) \in \widehat{R}_{e_i}$. Пусть $G_k = G_0(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$. В силу монотонности правил преобразования состояний, $(u, e', \alpha_r) \in R_{e_k}$. Отсюда, так как $e \leqslant e'$, в соответствии с замечанием 3 $(u, e, \alpha_r) \in R_{e_k}$. Значит, по определению 14 истинен предикат $can\_get\_right(u, e, \alpha_r, G)$.

Пусть выполнено условие 2. Согласно утверждению 2, из истинности предиката $can\_act\_as(u, u', G_0)$ следует истинность предиката $can\_become(u, u', G_0)$. Значит, по определению 12 существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k > 0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$ и $(u, u', impersonate_r) \in R_{e_k}$. Положим $op_{k+1}\langle s\rangle = switch(s, u')$. Заметим, что в состоянии $G_k$ выполнены условия применения правила $op_{k+1}\langle s\rangle$ и, следовательно, $user_{k+1}(s) = u'$. В соответствии с замечанием 17 $(u', e', \alpha_r) \lhd G_{k+1}$. Значит, существуют роли $r_1, \ldots, r_n \in R$, где $n \geqslant 1$, такие, что $(u', r_1, r_2, alter_r) \lessdot G_{k+1}$, для каждого $i = 1, \ldots, n-2$ выполняется $(r_i, r_{i+1}, r_{i+2}, alter_r) \lessdot G_{k+1}$ и $(r_{n-1}, r_n, e', \alpha_r) \lessdot G_{k+1}$. Положим $op_{k+2}\langle s\rangle = add\_member(s, r_1, u'), \ldots, op_{k+n}\langle s\rangle = add\_member(s, r_{n-1}, u')$. Пусть $G_{k+n} = G_k(op_{k+2}\langle s\rangle, \ldots, op_{k+n}\langle s\rangle)$. Так как $user_{k+1}(s) = u'$, учитывая $(u', r_1, r_2, alter_r) \lessdot G_{k+1}$, в соответствии с утверждением 4 получим $(u', r_2, alter_r) \in R_{e_{k+2}}$. В силу замечания 16 $(r_1, r_2, r_3, alter_r) \lessdot G_{k+2}$, откуда $(r_2, r_3, alter_r) \in R_{e_{k+2}}$, а значит, $(u', r_2, r_3, alter_r) \lessdot G_{k+2}$. Проводя аналогичные рассуждения, легко показать, что $(u', r_n, alter_r) \in R_{e_{k+n}}$. Положим $op_{k+n+1}\langle s\rangle = add\_member(s, r_n, u)$. Пусть $G_{k+n+1} = G_{k+n}(op_{k+n+1}\langle s\rangle)$. Заметим, что в состоянии $G_{k+n}$ выполнены условия применения правила, а значит, $r_n \in UA_{k+n+1}(u)$. Так как $(r_{n-1}, r_n, e', \alpha_r) \lessdot G_{k+1}$, в силу замечания 16 имеем $(r_{n-1}, r_n, e', \alpha_r) \lessdot G_{k+n+1}$, а значит, $(r_n, e', \alpha_r) \in R_{e_{k+n+1}}$, и тогда $(u, e', \alpha_r) \in R_{e_{k+n+1}}$. Отсюда, так как $e \leqslant e'$, в соответствии с замечанием 3 $(u, e, \alpha_r) \in R_{e_{k+n+1}}$. Значит, по определению 14 истинен предикат $can\_get\_right(u, e, \alpha_r, G_0)$.

Необходимость.

Пусть истинен предикат $can\_get\_right(u, e, \alpha_r, G_0)$. Тогда, согласно определению 14, существуют инициированные извне правила $op_1\langle s\rangle, \ldots, op_k\langle s\rangle \in OP_{\text{ext}}$, где $k \geqslant 0$ и $s \notin S_0$, такие, что $op_1\langle s\rangle = create\_session(s, u)$, $G_k = G_0(op_1\langle s\rangle, \ldots, op_k\langle s\rangle)$ и $(u, e, \alpha_r) \in R_{e_k}$. Пусть $vestige(G_0, op_1\langle s\rangle, \ldots, op_k\langle s\rangle) = (\widehat{op}_1, \ldots, \widehat{op}_m)$.

Так как $(u, e, \alpha_r) \notin R_{e_0}$ и $(u, e, \alpha_r) \notin R_{e_k}$, существует $i \in \{1, \ldots, m\}$, что $(u, e, \alpha_r) \notin \widehat{R}_{e_{i-1}}$ и $(u, e, \alpha_r) \notin \widehat{R}_{e_i}$. Пусть $l$ — количество правил вида $grant\_right$ и $add\_member$ в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_i)$. Доказательство проведём индукцией по $l$.

Пусть $l = 1$. Обозначим $\widehat{user}_{i-1}(s)$ как $u'$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u', G_0)$. Заметим, что в соответствии с замечанием 15 в состо-

янии $\widehat{G}_{i-1}$ выполнены условия применения правила $\widehat{op}_i$. Так как $l = 1$, $\widehat{op}_i$ — единственное правило вида $grant\_right$ и $add\_member$, откуда $\widehat{R}_{e_{i-1}} = R_{e_0}$ и $\widehat{Gr}_{e_{i-1}} = Gr_{e_0}$.

Пусть $\widehat{op}_i \prec grant\_right$, тогда $\widehat{op}_i = grant\_right(s, \ldots, e', \alpha_r, \ldots)$, $e' \in \widehat{E}_{i-1}$, $e \leqslant e'$ и $u' \in \widehat{Gr}_{i-1}(e', \alpha_r)$. В системе отсутствуют правила, приводящие к изменению родительских контейнеров сущностей, поэтому $e' \in E_0$. Так как $\widehat{Gr}_{e_{i-1}} = Gr_{e_0}$, выполняется $u' \in Gr_0(e', \alpha_r)$, а значит, выполнено условие 1 утверждения.

Если $\widehat{op}_i \prec add\_member$, то $\widehat{op}_i = add\_member(s, r, u)$, $r \in R$, $(u', r, alter_r) \in \widehat{R}_{e_{i-1}}$ и $(r, e', \alpha_r) \in \widehat{R}_{e_{i-1}}$, где $e' \in \widehat{E}_{i-1}$ и $e \leqslant e'$. В системе отсутствуют правила, приводящие к изменению родительских контейнеров сущностей, поэтому $e' \in E_0$. Так как $\widehat{R}_{e_{i-1}} = R_{e_0}$, имеем $(u', r, alter_r) \in R_{e_0}$ и $(r, e', \alpha_r) \in R_{e_0}$. Следовательно, $(u', e', \alpha_r) \lhd G_0$, а значит, выполнено условие 2 утверждения.

Пусть условия утверждения выполняются для всех $l < N$, где $N > 1$. Покажем, что они верны и при $l = N$. Обозначим $\widehat{user}_{i-1}(s)$ как $u''$. В соответствии с утверждением 3 истинен предикат $can\_act\_as(u, u'', G_0)$.

Пусть $\widehat{op}_i \prec grant\_right$, тогда $\widehat{op}_i \prec grant\_right(s, \ldots, e', \alpha_r, \ldots)$, $e' \in \widehat{E}_{i-1}$, $e \leqslant e'$ и $u'' \in \widehat{Gr}_{i-1}(e', \alpha_r)$. В системе отсутствуют правила, приводящие к изменению родительских контейнеров сущностей, поэтому $e' \in E_0$. Возможны следующие два случая: 1) $u'' \in Gr_0(e', \alpha_r)$; 2) $u'' \notin Gr_0(e', \alpha_r)$.

С л у ч а й 1. Если $u'' \in Gr_0(e', \alpha_r)$, то, положив $u' = u''$, видим, что выполнено условие 1 утверждения.

С л у ч а й 2. Так как $u'' \in \widehat{Gr}_{i-1}(e', \alpha_r)$, в силу монотонности правил преобразования состояний $u'' \in Gr_k(e', \alpha_r)$. Так как $u'' \notin Gr_0(e', \alpha_r)$ и $u'' \in Gr_k(e', \alpha_r)$, в соответствии с утверждением 6 существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и либо $u' \in Gr_0(e', \alpha_r)$, либо $(u', e', \alpha_r) \blacktriangleleft G_0$. Учитывая, что в случае $(u', e', \alpha_r) \blacktriangleleft G_0$ также выполняется $(u', e', \alpha_r) \lhd G_0$, видим, что выполнены условия 1 и 2 утверждения.

Пусть $\widehat{op}_i \prec add\_member$, тогда $\widehat{op}_i \prec add\_member(s, r, u)$, где $r \in R$, $(u'', r, alter_r) \in \widehat{R}_{e_{i-1}}$ и $(r, e', \alpha_r) \in \widehat{R}_{e_{i-1}}$, где $e' \in \widehat{E}_{i-1}$ и $e \leqslant e'$. В системе отсутствуют правила, приводящие к изменению родительских контейнеров сущностей, поэтому $e' \in E_0$. Возможны три случая: 1) $(u'', r, alter_r) \in R_{e_0}$ и $(r, e', \alpha_r) \in R_{e_0}$; 2) $(r, e', \alpha_r) \notin R_{e_0}$; 3) $(u'', r, alter_r) \notin R_{e_0}$ и $(r, e', \alpha_r) \in R_{e_0}$.

С л у ч а й 1. Легко заметить, что $(u'', e', \alpha_r) \lhd G_0$. Положив $u' = u''$, видим, что выполнено условие 2 утверждения.

С л у ч а й 2. Так как $(r, e', \alpha_r) \notin R_{e_0}$ и $(r, e', \alpha_r) \in \widehat{R}_{e_{i-1}}$, существует $j \in \{1, \ldots, i-1\}$, такое, что $(r, e', \alpha_r) \notin \widehat{R}_{e_{j-1}}$ и $(r, e', \alpha_r) \in \widehat{R}_{e_j}$. Так как $\widehat{op}_i \prec add\_member$ и $j < i$, в последовательности $(\widehat{op}_1, \ldots, \widehat{op}_j)$ количество правил вида $grant\_right$ и $add\_member$ меньше $N$. Значит, по предположению индукции существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и либо $u' \in Gr_0(e', \alpha_r)$, либо $(u', e', \alpha_r) \lhd G_0$.

С л у ч а й 3. Так как $(u'', r, alter_r) \in \widehat{R}_{e_{i-1}}$, в силу монотонности правил преобразования состояний получим $(u'', r, alter_r) \in R_{e_k}$. Заметим, что ввиду $(r, e', \alpha_r) \in R_{e_0}$, $(u'', r, alter_r) \notin R_{e_0}$ и $(u'', r, alter_r) \in R_{e_k}$, в соответствии с утверждением 5 существует $u' \in U$, такое, что истинен предикат $can\_act\_as(u, u', G_0)$ и $(u', r, e', \alpha_r) \lhd G_0$. Отсюда следует, что $(u', e', \alpha_r) \lhd G_0$, а значит, выполнено условие 2 утверждения. ■

## Заключение

Таким образом, для обеспечения возможности теоретического анализа безопасности СУБД построена формальная модель, в которую добавлены новые элементы, поз-

воляющие более полно учесть особенности управления доступом в MS SQL Server. В рамках модели определены необходимые и достаточные условия предоставления и получения прав доступа. Обоснована эквивалентность возможности выполнения SQL-кода от имени заданной учётной записи и получения к ней права олицетворения. В дальнейшем планируется определение необходимых и достаточных условий выполнения SQL-кода от имени заданной учётной записи, а также проведение анализа условий, при которых пользователь может активизировать процедуру или триггер от имени учётной записи заданного пользователя.

## ЛИТЕРАТУРА

1. *Девянин П. Н.* Ролевая ДП-модель управления доступом и информационными потоками в операционных системах семейства Linux // Прикладная дискретная математика. 2012. № 1(15). С. 69–90.
2. *Колегов Д. Н.* Дискреционная модель безопасности управления доступом и информационными потоками в компьютерных системах с функционально или параметрически ассоциированными сущностями: дис. ... канд. техн. наук. Томск, 2009.
3. *Смольянинов В. Ю.* Правила преобразования состояний СУБД ДП-модели // Прикладная дискретная математика. 2013. № 1(18). С. 50–68.
4. *Bruchez R.* Microsoft SQL Server 2012 Security Cookbook. Pact Publishing, 2012. 307 с.