2013 Математические основы компьютерной безопасности №1(19)
DOI 10.17223/20710410/19/5
УДК 004.94

ПРАВИЛА ПРЕОБРАЗОВАНИЯ СОСТОЯНИЙ СУБД ДП-МОДЕЛИ

В. Ю. Смольянинов

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

E-mail: VladimirSmall@gmail.com

Рассматривается ДП-модель управления доступом в реляционных системах управления базами данных (кратко СУБД ДП-модель). Для учёта особенностей управления доступом в СУБД в модель включены сущности-процедуры, сущности-триггеры, специфичные для СУБД права доступа, наследование прав доступа, возможность динамического изменения учётной записи пользователя при активации сущностей-процедур и сущностей-триггеров. Обосновывается утверждение о предоставления прав доступа к сущностям-процедурам и контейнерам.

Ключевые слова: *компьютерная безопасность, СУБД ДП-модель, система управления базами данных.*

Введение

В настоящее время при проектировании, разработке и эксплуатации автоматизированных информационных систем особое значение стали приобретать вопросы обеспечения информационной безопасности. Практика показала, что одной из наиболее важных задач является обеспечение корректности управления доступом, а также анализ условий, приводящих к реализации запрещённых информационных потоков.

Для обеспечения возможности поиска, хранения и обработки информации разработчики автоматизированных информационных систем традиционно используют реляционные системы управления базами данных (СУБД). Современные СУБД предоставляют разработчикам прикладных приложений развитые средства управления доступом, которые имеют отличия от механизмов, применяемых в операционных системах (ОС).

В соответствии с «Критериями оценки безопасности информационных технологий» [1] использование формальных математических моделей является обязательным для анализа безопасности управления доступом и информационными потоками в компьютерных системах, начиная с уровня доверия ОУД 5. Существующие формальные модели логического управления доступом [2, 3] 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. В частности, ими игнорируется 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.

Следовательно, теоретический анализ условий, при которых возможно несанкционированное распространение прав доступа, а также возникновение запрещённых информационных потоков в автоматизированных информационных системах, использу-

ющих штатные механизмы управления доступом в СУБД, является важной и актуальной задачей.

Далее описываются элементы СУБД ДП-модели, предназначенной для анализа безопасности управления доступом в СУБД *Microsoft SQL Server*. Построенная модель может быть адаптирована для анализа безопасности других реляционных СУБД.

1. Определения и обозначения

В предлагаемой СУБД ДП-модели, построенной на основе РОСЛ ДП-модели [4], используются следующие обозначения и определения:

O_d — множество сущностей-данных;

O_p — множество сущностей-процедур;

O_t — множество сущностей-триггеров;

$O = O_d \cup O_p \cup O_t$ — множество сущностей-объектов, при этом считается, что множества сущностей-данных, сущностей-процедур и сущностей-триггеров попарно не пересекаются, то есть $O_d \cap O_p = \varnothing$, $O_d \cap O_t = \varnothing$ и $O_p \cap O_t = \varnothing$;

C — множество контейнеров;

$E = O \cup C$ — множество сущностей, при этом считается, что $O \cap C = \varnothing$, то есть контейнеры не являются сущностями-объектами.

Определение 1. Определим $H\colon C \to 2^E$ — функцию иерархии сущностей, сопоставляющую каждому контейнеру $c \in C$ множество $H(c) \subset E$ и удовлетворяющую следующим условиям.

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

У с л о в и е 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 $e \in H(c_n)$.

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

Определение 2. Если для сущности $e \in E$ 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.

Определение 3. Будем говорить, что сущность $e \in E$ 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 $e \in E$ иерархически подчинена контейнеру $c \in C$, будем использовать обозначение $e < c$. В случае, если сущность $e \in E$ иерархически подчинена контейнеру $c \in C$ или равна ему, будем использовать обозначение $e \leqslant c$.

Замечание 1. Из определений 1 и 3 следует, что для любой сущности $e \in E \setminus \{c_r\}$ существует контейнер $c \in C$, такой, что $e < c$. Кроме того, очевидно, что каждая сущность либо является корневой, либо иерархически подчинена корневой сущности c_r и содержится ровно в одном из её дочерних контейнеров. Будем считать необходимым условием равенства сущностей их принадлежность одному контейнеру.

Замечание 2. Легко показать, что если для некоторой сущности $e \in E$ 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 $i = 1, \ldots, n$, то есть для любой некорневой сущности $e \in E$ существует единственная последовательность, начинающаяся с c_r и заканчивающаяся контейнером, содержащим e, такая, что каждый её элемент является родительским для последующего.

Используем также следующие обозначения и определения:

U — множество учётных записей пользователей, при этом считается, что учётные записи пользователей по определению не являются сущностями, то есть $E \cap U = \varnothing$;

S — множество субъект-сессий пользователей, при этом считается, что $S \cap U = \varnothing$ и $S \cap E = \varnothing$;

$user\colon S \to U$ — функция, задающая для каждой субъект-сессии учётную запись пользователя, от имени которой она выполняется;

$assoc_f(s)\colon S \to O_p \cup O_t \cup \{\varnothing\}$ — функция, задающая для каждой субъект-сессии $s \in S$ функционально ассоциированную с ней сущность;

$owner\colon E \to U$ — функция, задающая для каждой сущности учётную запись её владельца, при этом для каждой таблицы $t \in T$ и каждой сущности $e \in H(t)$ по определению выполняется равенство $owner(e) = owner(t)$;

$owners\colon E \to 2^U$ — функция, задающая для каждой сущности учётные записи её иерархических владельцев, при этом для каждой сущности $e \in E$ по определению $u \in owners(e)$ тогда и только тогда, когда либо $u = owner(e)$, либо существует контейнер $c \in P(e)$, такой, что $u \in owners(c)$;

$execute\_as\colon O_p \cup O_t \to \{as\_caller, as\_owner\}$ — функция, задающая для каждой сущности-процедуры и сущности-триггера режим их выполнения субъект-сессиями (as_caller соответствует режиму выполнения сущности от имени учётной записи пользователя, инициировавшей её запуск, а as_owner — от имени учётной записи владельца сущности);

$R_r = \{read_r, write_r, append_r, delete_r, alter_r, execute_r\}$ — множество видов прав доступа к контейнерам и сущностям-процедурам;

O_t^* — множество всех конечных последовательностей сущностей-триггеров.

Определение 4. Определим $triggers\colon T \times \{write_r, append_r, delete_r\} \to O_t^*$ — функцию, задающую для каждой таблицы связанную с ней последовательность сущностей-триггеров и удовлетворяющую следующим условиям.

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

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

Если существуют таблица $t \in T$, право доступа $\alpha_r \in \{write_r, append_r, delete_r\}$ и сущность-триггер $o_t \in triggers(t, \alpha_r)$, то будем говорить, что сущность-триггер o_t имеет тип α_r и связана с таблицей t.

Введём также следующие обозначения:

$R \subseteq U \times (C \cup O_p) \times R_r$ — множество непосредственно заданных прав доступа учётных записей пользователей к контейнерам и сущностям-процедурам;

$R_{own} = \{(u, e, \alpha_r) : \alpha_r \in R_r, e \in C \cup O_p, u = owner(e)\}$ — множество прав доступа учётных записей пользователей — владельцев контейнеров и сущностей-процедур;

$R_H = \{(u, e, \alpha_r) : u \in U, e \in C \cup O_p, \alpha_r \in R_r, \exists c \in C((u, c, \alpha_r) \in R \cup R_{own} \,\&\, e < c)\}$ — множество иерархических прав доступа учётных записей пользователей к контейнерам и сущностям-процедурам;

$R_e = R \cup R_{own} \cup R_H$ — множество действующих прав доступа учётных записей пользователей к контейнерам и сущностям-процедурам;

$Gr_e = Gr \cup Gr_{own}$ — множество действующих прав на предоставление прав доступа к контейнерам и сущностям-процедурам, при этом множества Gr и Gr_{own} задаются следующим образом:

$Gr \subseteq U \times (C \cup O_p) \times R_r$ — множество непосредственно заданных прав на предоставление прав доступа к контейнерам и сущностям-процедурам, удовлетворяющее условию $Gr \subseteq R_e$;

$Gr_{own} = \{(u, e, \alpha_r) : u \in U, \alpha_r \in R_r, e \in C \cup O_p, u \in owners(e)\}$ — множество прав доступа на предоставление прав доступа иерархическими владельцами контейнеров и сущностей-процедур;

$Gr(e, \alpha_r) = \{u \in U : (u, e, \alpha_r) \in Gr_e\}$, где $e \in C \cup O_p$, $\alpha_r \in R_r$ — множество учётных записей пользователей, которые могут предоставлять право доступа $\alpha_r \in R_r$ к сущности $e \in C \cup O_p$;

OP_{int} — множество внутренних правил преобразования состояний, заданных в табл. 1;

OP_{ext} — множество внешних правил преобразования состояний, заданных в табл. 2;

$OP = OP_{int} \cup OP_{ext}$ — множество правил преобразования состояний;

OP_{ext}^* — множество всех конечных последовательностей внешних правил преобразования состояний;

OP^* — множество всех конечных последовательностей правил преобразования состояний;

$operations\colon O_p \cup O_t \to OP_{ext}^*$ — функция, задающая для каждой сущности-процедуры и сущности-триггера соответствующие им последовательности внешних правил преобразования состояний, реализуемых при выполнении субъектами-сессиями их SQL-кода;

$time \in \mathbb{N}_0$ — счётчик моментов времени, значение которого увеличивается на единицу при реализации каждого правила $op \in OP$;

$tr \in OP^*$ — последовательность всех применённых правил, начиная с нулевого момента времени, непосредственная реализация которых привела к изменению значения счётчика времени;

$G = (U, S, E, R, Gr, H, user, assoc_f, owner, execute\_as, triggers, operations, time, tr)$ — 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.

В дальнейшем будем использовать следующее предположение.

Предположение 1. Считается, что в начальном состоянии системы $\Sigma(G^*, OP, G_0)$ отсутствуют субъект-сессии, то есть $S_0 = \varnothing$; кроме того, значение счётчика моментов времени $time_0$ в начальном состоянии равно 0 и последовательность tr_0 не содержит элементов.

Поясним введённые обозначения и определения.

Сущности-данные модели соответствуют записям таблиц СУБД. В отличие от реальных СУБД считается, что у записей таблиц отсутствует внутренняя структура, то есть нет отдельных полей. Причина подобного подхода к рассмотрению записей состоит в том, что в большинстве современных СУБД отсутствуют механизмы контроля доступа не только на уровне полей записей таблиц, но и на уровне отдельных записей таблиц. Следовательно, построение формальных моделей управления доступом

к отдельным полям записей таблиц для современных СУБД не является актуальной задачей.

Таблица 1

Внутренние правила преобразования состояний СУБД ДП-модели

Правило	Исходное состояние G	Результирующее состояние G'
$switch(s, o, u, op)$	$s \in S, u \in U,$ $o \in O_p \cup O_t \cup \{\varnothing\}$, $op \in OP_{ext}$	$G'\{user', assoc'_f, time', tr'\} \sim G',$ $user'\{s \mapsto u\} \sim user$, $assoc'_f\{s \mapsto o\} \sim assoc_f,$ $time' = time + 1$, $tr' = (tr, switch(s, o, u, op))$
$revert(s, u, o)$	$s \in S, u \in U,$ $o \in O_p \cup O_t \cup \{\varnothing\}$	$G'\{user', assoc'_f, time', tr'\} \sim G',$ $user'\{s \mapsto u\} \sim user$, $assoc'_f\{s \mapsto o\} \sim assoc_f,$ $time' = time + 1$, $tr' = (tr, revert(s, u, o))$
$execute(s, o, op)$	$s \in S, o \in O_p \cup O_t,$ $op \in OP_{ext}$, $operations(o) =$ $= (op_1, \ldots, op_k)$, $k \geqslant 0,$ $\forall i \in \overline{1,k}\ op_i \in OP_{ext}$	$G' = G(switch(s, o, u, op), op_1, \ldots, op_k,$ $revert(s, user(s), [s])),$ $u = \begin{cases} owner(o) \text{ при } execute\_as(o) = as\_owner, \\ user(s) \text{ при } execute\_as(o) = as\_caller \end{cases}$
do_insert $(s, t_1, o_{d_1}, t_2, o_{d_2})$	$s \in S, t_1, t_2 \in T, u = user(s),$ $o_{d_1} \notin O_d, o_{d_2} \in H(t_2)$	$G'\{E', H', time', tr'\} \sim G,$ $E' = O' \cup C,$ $O' = O \cup \{o_{d_1}\}$, $H'\{t_1 \mapsto H(t_1) \cup \{o_{d_1}\}\} \sim H,$ $time' = time + 1,$ $tr' = (tr, do\_insert(s, t_1, o_{d_1}, t_2, o_{d_2}))$
do_update $(s, t_1, o_{d_1}, t_2, o_{d_2})$	$s \in S, t_1, t_2 \in T, u = user(s),$ $o_{d_1} \in H(t_1)$, $o_{d_2} \in H(t_2)$	$G'\{time', tr'\} \sim G$, $time' = time + 1,$ $tr' = (tr, do\_update(s, t_1, o_{d_1}, t_2, o_{d_2}))$
$do\_delete(s, t, o_d)$	$s \in S, t \in T, u = user(s),$ $o_d \in H(t)$	$G'\{E', H', time', tr'\} \sim G,$ $E' = O' \cup C,$ $O' = O \setminus \{o_d\}$, $H'\{t \mapsto H(t) \setminus \{o_d\}\} \sim H,$ $time' = time + 1$, $tr' = (tr, do\_delete(s, t, o_d))$

Таблица 2

Внешние правила преобразования состояний СУБД ДП-модели

Правило	Исходное состояние G	Результирующее состояние G'
1	2	3
$create\_session$ (u, s)	$u \in U, s \notin S$	$G'\{S', user', assoc'_f, time', tr'\} \sim G, S' = S \cup \{s\},$ $user'\{s \mapsto u\} \sim user$, $assoc'_f\{s \mapsto \varnothing\} \sim assoc_f,$ $time' = time + 1$, $tr' = (tr, create\_session(u, s))$
$grant\_right$ $(s, u, e, \alpha_r,$ $with\_grant)$	$s \in S$, $u \in U$, $e \in C \cup O_p,$ $\alpha_r \in R_r$, $user(s) \in Gr(e, \alpha_r),$ $with\_grant \in \{\texttt{yes}, \texttt{no}\}$	$G'\{R', Gr', time', tr'\} \sim G, R' = R \cup \{(u, e, \alpha_r)\},$ $Gr' = \begin{cases} Gr \cup \{(u, e, \alpha_r)\} \text{ при } with\_grant = \texttt{yes}, \\ Gr \text{ при } with\_grant = \texttt{no} \end{cases}$ $time' = time + 1$, $tr' = (tr, grant\_right(s, u, e,$ $\alpha_r, with\_grant))$
$create\_container$ (s, c_p, c)	$s \in S, c_p \in C, c \notin C,$ $(user(s), c_p, alter_r) \in R_e$	$G'\{E', H', owner', time', tr'\} \sim G,$ $E' = O \cup C',$ $C' = C \cup \{c\}$, $H'\{c \mapsto \varnothing, c_p \mapsto H(c_p) \cup \{c\}\} \sim H,$ $owner'\{c \mapsto user(s)\} \sim owner$, $time' = time + 1,$ $tr' = (tr, create\_container(s, c_p, c))$
$create\_procedure$ $(s, c, o_p, md,$ $op_1, \ldots, op_k)$	$s \in S, c \in C \setminus T$, $o_p \notin O_p,$ $md \in \{as\_caller, as\_owner\},$ $k \geqslant 0$, $\forall i \in \overline{1,k}\ op_i \in OP_{ext},$ $(user(s), c, alter_r) \in R_e$	$G'\{E', H', owner', execute\_as', operations',$ $time', tr'\} \sim G,$ $E' = O' \cup C,$ $O' = O \cup \{o_p\},$ $H'\{c \mapsto H(c) \cup \{o_p\}\} \sim H,$ $owner'\{o_p \mapsto user(s)\} \sim owner,$ $execute\_as'\{o_p \mapsto md\} \sim execute\_as,$ $operations'\{o_p \mapsto (op_1, \ldots, op_k)\} \sim operations,$ $time' = time + 1$, $tr' = (tr, create\_procedure(s, c,$ $o_p, md, op_1, \ldots, op_k)$

Окончание табл. 2

1	2	3
$alter\_procedure$ $(s, o_p, md,$ $op_1, \ldots, op_k)$	$s \in S$, $o_p \in O_p$, $md \in \{as\_caller, as\_owner\}$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ op_i \in OP_{ext}$, $(user(s), o_p, alter_r) \in R_e$	$G'\{execute\_as', operations', time', tr'\} \sim G$, $execute\_as'\{o_p \mapsto m\} \sim execute\_as$, $operations'\{o_p \mapsto (op_1, \ldots, op_k)\} \sim operations$, $time' = time+1$, $tr' = (tr, alter\_procedure(s, o_p,$ $md, op_1, \ldots, op_k))$
$create\_trigger$ $(s, t, o_t, \alpha_r, md,$ $op_1, \ldots, op_k)$	$s \in S, t \in T, o_t \notin O_t, \alpha_r \in R_r$, $md \in \{as\_caller, as\_owner\}$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ op_i \in OP_{ext}$, $(user(s), t, alter_r) \in R_e$	$G'\{E', H', owner', execute\_as', triggers',$ $operations', time', tr'\} \sim G$, $E' = O' \cup C$, $O' = O \cup \{o_t\}$, $H'\{t \mapsto H(t) \cup \{o_t\}\} \sim H$, $owner'\{o_t \mapsto user(s)\} \sim owner$, $execute\_as'\{o_t \mapsto md\} \sim execute\_as$, $triggers'\{(t, \alpha_r) \mapsto (triggers(t, \alpha_r), o_t)\} \sim$ $\sim triggers$, $operations'\{o_t \mapsto (op_1, \ldots, op_k)\} \sim$ $\sim operations$, $time' = time + 1$, $tr' = (tr,$ $create\_trigger(s, t, o_t, \alpha_r, md, op_1, \ldots, op_k))$
$alter\_trigger$ $(s, t, o_t, md,$ $op_1, \ldots, op_k)$	$s \in S, t \in T, o_t \in O_t$, $md \in \{as\_caller, as\_owner\}$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ op_i \in OP_{ext}$, $(user(s), t, alter_r) \in R_e$, $o_t \in H(t)$	$G'\{execute\_as', operations', time', tr'\} \sim G$, $execute\_as'\{o_t \mapsto md\} \sim execute\_as$, $operations'\{o_t \mapsto (op_1, \ldots, op_k)\} \sim operations$, $time' = time + 1$, $tr' = (tr, alter\_trigger(s, t, o_t,$ $md, op_1, \ldots, op_k))$
$execute\_procedure$ (s, o_p)	$s \in S, o_p \in O_p$, $(user(s), o_p, execute_r) \in R_e$	$G' = G(execute(s, o_p, execute\_procedure(s, o_p)))$
$access\_insert$ $(s, t_1, o_{d_1}, t_2, o_{d_2})$	$s \in S, t_1, t_2 \in T, u = user(s)$, $(u, t_1, write_r) \in R_e$, $(u, t_2, read_r) \in R_e$, $triggers(t_1, append_r) =$ $= (o_{t,1}, \ldots, o_{t,k})$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ o_{t,i} \in O_t$	$G' = G(do\_insert(s, t_1, o_{d_1}, t_2, o_{d_2}),$ $execute(s, o_{t,1}, access\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})),$ $\ldots$ $execute(s, o_{t,k}, access\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})))$
$access\_update$ $(s, t_1, o_{d_1}, t_2, o_{d_2})$	$s \in S, t_1, t_2 \in T, u = user(s)$, $(u, t_1, write_r) \in R_e$, $(u, t_2, read_r) \in R_e$, $triggers(t_1, write_r) =$ $= (o_{t,1}, \ldots, o_{t,k})$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ o_{t,i} \in O_t$	$G' = G(do\_update(s, t_1, o_{d_1}, t_2, o_{d_2}),$ $execute(s, o_{t,1}, access\_update(s, t_1, o_{d_1}, t_2, o_{d_2})),$ $\ldots$ $execute(s, o_{t,k}, access\_update(s, t_1, o_{d_1}, t_2, o_{d_2})))$
$access\_delete$ (s, t, o_d)	$s \in S, u = user(s)$, $(u, t, delete_r) \in R_e$, $triggers(t, delete_r) =$ $= (o_{t,1}, \ldots, o_{t,k})$, $k \geqslant 0$, $\forall i \in \overline{1,k}\ o_{t,i} \in O_t$	$G' = G(do\_delete(s, t, o_d),$ $execute(s, o_{t,1}, access\_delete(s, t, o_d)),$ $\ldots$ $execute(s, o_{t,k}, access\_delete(s, t, o_d)))$

Множество контейнеров реальных реляционных СУБД включает такие сущности как экземпляр СУБД, экземпляры баз данных, схемы и таблицы. При этом иерархия сущностей на этом множестве контейнеров задаётся неявно. 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.

Будем считать, что корневой контейнер c_r, у которого отсутствуют родительские контейнеры, соответствует экземпляру СУБД. В отличие от реальных СУБД, модель не накладывает ограничений на число уровней вложенности в иерархии.

Помимо корневого контейнера c_r, в рамках модели выделяется особый вид контейнеров — таблицы. Согласно условию 3 определения 1, для каждой таблицы $t \in T$ выполняется $H(t) \subseteq O_d \cup O_t$, из чего следует, что таблицы не могут иметь дочерних контейнеров и содержат только сущности-данные и сущности-триггеры.

Обоснуем следующее утверждение.

Утверждение 1. Если $|C| > 1$, то $c_r \notin T$.

Доказательство. Предположим противное, то есть $c_r \in T$.

Из условия утверждения следует, что существует $c \in C$, $c \neq c_r$. Тогда по условию 2 определения 1 существуют контейнеры $c_1, \ldots, c_n$, где $n \geqslant 1$, такие, что $c_1 = c_r$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $c \in H(c_n)$. Это означает, что существует контейнер $c' \in C$, такой, что $c' \in H(c_r)$, причём $c' \notin O_d \cup O_t$, так как $O \cap C = \varnothing$.

По предположению корневой контейнер c_r является таблицей, следовательно, по условию 3 определения 1 выполняется $H(c_r) \subseteq O_d \cup O_t$.

Отсюда одновременно имеет место $c' \in H(c_r) \subseteq O_d \cup O_t$ и $c' \notin O_d \cup O_t$. Пришли к противоречию, следовательно, корневой контейнер c_r не является таблицей. ■

В замечании 1 отмечается, что для каждой сущности за исключением корневой существует единственный контейнер, в котором она содержится. Кроме того, в соответствии с условием 3 определения 1 в множество таблиц включаются все контейнеры, содержащие хотя бы одну сущность-данные или сущность-триггер. Следовательно, для каждой сущности-данные и сущности-триггера $e \in O_d \cup O_t$ существует ровно одна таблица $t \in T$, в которой она содержится, то есть $e \in H(t)$.

Сущности-триггеры соответствуют триггерам таблиц в СУБД, SQL-код которых автоматически выполняется при осуществлении операций вставки, удаления и обновления записей таблиц, с которыми они связаны. Каждая сущность-триггер имеет тип, задаваемый функцией $triggers$ и соответствующий определённому виду доступа. Реализация субъект-сессией доступа вида $\alpha_r \in \{write_r, append_r, delete_r\}$ к таблице $t \in T$ автоматически инициирует выполнение ею SQL-кода всех триггеров из последовательности $triggers(t, \alpha_r)$.

Сущности-процедуры соответствуют хранимым процедурам СУБД, выполнение SQL-кода которых может быть инициировано в рамках взаимодействия с системой. В реальных СУБД хранимые процедуры могут находиться только в схемах данных. В рамках модели считается, что сущности-процедуры могут находиться в любом контейнере, не являющемся таблицей.

Считается, что сущности-триггеры и сущности-процедуры содержат SQL-код, выполнение которого приводит к реализации преобразований системы, задаваемых правилами, описываемыми в п. 2. В рамках модели предполагается, что сущности-триггеры и сущности-процедуры не содержат динамически генерируемого SQL-кода и условий, а их выполнение приводит к применению фиксированной последовательности правил, задаваемой соответствующим значением функции $operations$. В реальных системах сущности-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 сущностей-процедур. При этом для каждой допустимой комбинации значений фактических параметров задаётся отдельная сущность-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 сущности-процедуры, полученной подстановкой фактических параметров.

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.

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

Традиционно для сущностей, функционально-ассоциированных с субъект-сессией s, используется обозначение $[s]$. 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 одну выполняемую им в текущий момент времени сущность-триггер или сущность-процедуру, либо не содержит элементов. Последнее означает, что субъект-сессия выполняет SQL-код, который не относится ни к одной сущности-процедуре или сущности-триггеру. В последнем случае считается, что выполнение такого SQL-кода эквивалентно непосредственному применению субъект-сессией правил преобразования состояний из множества OP_{ext}. В дальнейшем будем полагать, что по определению выполняется равенство $[s] = assoc_f(s)$.

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 внешние 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 пользователя, не являющегося 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.

Функция $owner$ введена в модель в связи с тем, что в современных СУБД владельцы сущностей-контейнеров неявно получают все возможные права доступа как к самому контейнеру, так и ко всем его иерархически подчиненным сущностям. Из определения функции $owner$ следует, что владельцем сущностей-данных и сущностей-триггеров является владелец содержащей их таблицы. Это отражает тот факт, что в реальных СУБД у записей таблиц и триггеров не существует непосредственно задаваемой учётной записи владельца.

Предположение 2. Для любой сущности e учётная запись пользователя владельца $owner(e)$ после создания сущности e не изменяется.

Функция $owners$ задаёт для каждой сущности учётные записи всех её иерархических владельцев, которые обладают к ней всеми возможными правами доступа. При этом справедливо следующее

Утверждение 2. Для каждой сущности $e \in E$ выполняется равенство

$$owners(e) = \{u \in U : \exists c \in C\ (e < c\ \&\ u = owner(c))\} \cup \{owner(e)\}.$$

Доказательство. Обозначим

$$owners'(e) = \{u \in U : \exists c \in C\ (e < c\ \&\ u = owner(c))\} \cup \{owner(e)\}.$$

Покажем, что для каждой $e \in E$ верно равенство $owners'(e) = owners(e)$. Возможны два случая: $e = c_r$ и $e \neq c_r$.

Если $e = c_r$, то по определению 1 выполняется $|P(c_r)| = 0$, а значит, из определения 3 следует, что не существует контейнера $c \in C$, такого, что $c_r < c$. Значит, в данном случае $owners'(e) = \{owner(e)\}$. Кроме того, из способа задания функции $owners$ видно, что в случае $e = c_r$ выполняется равенство $owners(e) = \{owner(e)\}$. Следовательно, в случае $e = c_r$ верно равенство $owners'(e) = owners(e)$.

Если $e \neq c_r$, то, согласно замечанию 2, существует единственная последовательность контейнеров $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, таких, что $c_1 = c_r$, $c_{i+1} \in H(c_i)$ для $i = 1, \ldots, n-1$ и $e \in H(c_n)$.

Покажем, что не существует контейнера $c \in C$, такого, что $e < c$ и $c \neq c_j$, $j = 1, \ldots, n$. Предположим противное. Так как $c_1 = c_r$ и по предположению $c \neq c_1$, то, согласно замечанию 2, существует единственная последовательность контейнеров $c'_1, \ldots, c'_k \in C$, где $k \geqslant 1$, таких, что $c'_1 = c_r$, $c'_{i+1} \in H(c'_i)$ для $i = 1, \ldots, k-1$ и $c \in H(c'_k)$. Кроме того, так как $e < c$, то по определению 3 существуют контейнеры $c'_{k+1}, \ldots, c'_{n'} \in C$, где $n' \geqslant k+1$, такие, что $c'_{k+1} = c$, $c'_{i+1} \in H(c'_i)$ для $i = k+1, \ldots, n'-1$ и $e \in H(c'_{n'})$. Следовательно, существует последовательность контейнеров $c'_1, \ldots, c'_{n'}$, таких, что $c'_1 = c_r$, $c'_{i+1} \in H(c'_i)$ для $i = 1, \ldots, n'-1$ и $e \in H(c'_{n'})$. Так как по замечанию 2 существует лишь единственная последовательность контейнеров, удовлетворяющая данным свойствам, то $n' = n$ и $c'_i = c_i$, $i = 1, \ldots, n$. Следовательно, существует $j = k+1$, $j \in \{1, \ldots, n\}$ и $c = c'_j = c_j$, а по предположению $c \neq c_j$ для всех $j = 1, \ldots, n$. Пришли к противоречию.

Отсюда имеем, что $\{c \in C : e < c\} = \{c_1, \ldots, c_n\}$. Следовательно, верно равенство

$$owners'(e) = \Big(\bigcup_{i=1}^{n} owner(c_i) \Big) \bigcup \{owner(e)\}. \tag{1}$$

Доказательство для случая $e \neq c_r$ проведем индукцией по n — количеству элементов в последовательности контейнеров $c_1, \ldots, c_n \in C$, где $n \geqslant 1$, таких, 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 $n = 1$ из (1) следует выполнение равенства

$$owners'(e) = owner(c_1) \cup owner(e). \tag{2}$$

Пусть $u \in owners(e)$. Из способа задания функции $owners$ следует, что возможны два случая: либо $u = owner(e)$, либо существует контейнер $c \in P(e)$, такой, что $u \in owners(c)$. Если $u = owner(e)$, то из (2) следует, что $u \in owners'(e)$. Рассмотрим второй случай. Пусть существует контейнер $c \in P(e)$, такой, что $u \in owners(c)$. Так как $e \in H(c_1)$, то $c = c_1$, а следовательно, $u \in owners(c_1)$. В этом случае из (2) следует, что $u \in owners'(e)$.

Пусть $u \in owners'(e)$. Из (2) следует, что возможны два случая: $u = owner(e)$ и $u = owner(c_1)$. В случае $u = owner(e)$ из способа задания функции $owners$ следует, что $u \in owners(e)$, а если $u = owner(c_1)$, то $u \in owners(c_1)$. Таким образом, существует контейнер $c_1 \in P(e)$, такой, что $u \in owners(c_1)$, а следовательно, $u \in owners(e)$.

Итак, в случае $n = 1$ выполняется равенство $owners'(e) = owners(e)$.

Пусть $n > 1$ и равенство $owners'(e) = owners(e)$ выполняется для всех $k < n$. Заметим, что из (1) следует $owners'(e) = owners'(c_n) \cup \{owner(e)\}$. Число элементов в последовательности $c_1, \ldots, c_{n-1}$ меньше, чем n, и, следовательно, по предположению индукции $owners'(c_n) = owners(c_n)$. Таким образом, верно равенство

$$owners'(e) = owners(c_n) \cup \{owner(e)\}. \tag{3}$$

Пусть $u \in owners(e)$. Из способа задания функции $owners$ следует, что возможны два случая: либо $u = owner(e)$, либо существует контейнер $c \in P(e)$, такой, что $u \in owners(c)$. Если $u = owner(e)$, то из (3) следует $u \in owners'(e)$. Рассмотрим случай $u \neq owner(e)$. Так как $c \in P(e)$ и $e \in H(c_n)$, то $c = c_n$, а следовательно, $u \in owners(c_n)$. Тогда из равенства (3) видно, что $u \in owners'(e)$.

Пусть $u \in owners'(e)$. Из (3) следует, что возможны два случая: $u = owner(e)$ и $u \in owners(c_n)$. Если $u = owner(e)$, то из способа задания функции $owners$ следует $u \in owners(e)$. Рассмотрим случай $u \in owners(c_n)$. Ввиду $e \in H(c_n)$ имеем $c_n \in P(e)$. Таким образом, $c_n \in P(e)$ и $u \in owners(c_n)$, откуда, по способу задания функции $owners$, верно $u \in owners(e)$. Следовательно, для случая $k = n$ индуктивный шаг доказан и $owners'(e) = owners(e)$. ■

Таким образом, для каждой сущности $e \in E$ множество $owners(e)$ состоит из учётной записи владельца сущности e, а также учётных записей владельцев контейнеров, которым иерархически подчинена сущность e. Заметим, что значения функции $owners$ однозначно определяются функцией иерархии сущностей H и функцией задания владельцев сущностей $owner$, и поэтому она не рассматривается в качестве самостоятельного элемента состояния системы.

Значение функции $execute\_as$ задаёт режим выполнения сущностей-процедур и сущностей-триггеров, определяющий учётную запись пользователя, от имени которой субъект-сессии будут выполнять их SQL-код. Если некоторая субъект-сессия $s \in S$ инициировала выполнение сущности-процедуры или сущности-триггера $e \in O_p \cup O_t$ и $execute\_as(e) = as\_caller$, то код сущности e будет выполняться от имени учётной записи $user(s)$, в противном случае, если $execute\_as(e) = as\_owner$, то код сущности e будет выполняться субъект-сессией s от имени учётной записи владельца сущности e, то есть от имени учётной записи пользователя $owner(e)$. Отметим, что режим выполнения as_owner соответствует механизму повышения полномочий $SUID$ в ОС семейства UNIX, а режим as_caller — механизму имперсонации (олицетворения) в ОС семейства Windows.

Замечание 3. Присутствующий в реальных СУБД режим выполнения сущностей-процедур и сущностей-триггеров от имени фиксированной учётной записи в рамках модели не рассматривается.

Виды прав доступа из множества R_r соответствуют по смыслу и назначению традиционным для СУБД правам доступа $SELECT$, $UPDATE$, $INSERT$, $DELETE$, $ALTER$ и $EXECUTE$.

Большинство современных СУБД реализуют, как минимум, дискреционное управление доступом учётных записей пользователей к контейнерам и сущностям-процеду-

рам. При этом для упрощения процедуры администрирования прав доступа в СУБД поддерживается управление правами учётных записей пользователей с учётом иерархических отношений между сущностями, а также выполняется неявное предоставление всех возможных прав доступа учётным записям владельцев сущностей.

При определении возможности применения правил преобразования состояний в модели учитываются права доступа, содержащиеся в множестве действующих прав доступа учётных записей пользователей R_e. Полагается, что субъект-сессии s, функционирующей от имени учётной записи пользователя $user(s) = u$, разрешено реализовать доступ вида α_r к сущности e в случае, если $(u, e, \alpha_r) \in R_e$. Множество действующих прав доступа R_e включает множество непосредственно заданных прав доступа R, множество прав доступа учётных записей владельцев R_{own}, а также множество иерархических прав доступа R_H. Из способа задания множества R_{own} следует, что учётная запись владельца сущности обладает к ней всеми правами доступа. Кроме того, из способа задания множества R_H следует, что учётная запись пользователя, имеющего непосредственно заданное право доступа α_r к контейнеру c либо являющегося его владельцем, также обладает этим правом доступа и к любому его иерархически подчинённому контейнеру и сущности-процедуре. Таким образом, верно следующее замечание.

Замечание 4. 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 $(u, e, \alpha_r) \in R_e$.

Заметим, что множества прав доступа R_{own}, R_H и R_e однозначно определяются множеством прав доступа R, функцией иерархии сущностей H и функцией задания владельцев сущностей $owner$ и поэтому не рассматриваются в качестве самостоятельных элементов состояния системы.

Считается, что наличие у учётной записи прав доступа $read_r$, $write_r$, $append_r$ или $delete_r$ к таблице позволяет выполнять над иерархически подчинёнными ей сущностями-данными операции выборки, обновления, вставки и удаления соответственно. Наличие права доступа $delete_r$ к контейнеру позволяет удалять его дочерние контейнеры и сущности-процедуры. Обладание правом доступа $alter_r$ к сущности-процедуре позволяет задавать её SQL-код и режим выполнения. Право доступа $alter_r$ к таблице позволяет создавать в ней сущности-триггеры, а также изменять SQL-код связанных с ней сущностей-триггеров и режим их выполнения. Наличие права доступа $execute_r$ на сущность-процедуру позволяет выполнять её SQL-код.

При введении множества непосредственно заданных прав доступа R для простоты игнорируется, что права доступа $read_r$, $write_r$ и $append_r$ могут применяться только к контейнерам, а право доступа $execute_r$ неприменимо к таблицам. Отметим, что способ задания множества R учитывает, что большинство современных СУБД предоставляют возможность устанавливать права доступа к контейнерам и сущностям-процедурам, не позволяя определять их на уровне отдельных сущностей-данных и сущностей-триггеров.

Как правило, в ОС, реализующих механизмы дискреционного управления доступом, управлять предоставлением прав к сущности может только её владелец и, возможно, пользователь, обладающий привилегиями администратора. Указанные пользователи могут дать произвольное право доступа к сущности любому другому пользователю системы. 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.

Для учёта данной особенности в модель вводится множество действующих прав на предоставление прав доступа к контейнерам и сущностям-процедурам Gr_e. Полагается, что если $(u, e, \alpha_r) \in Gr_e$, то субъект-сессия s, функционирующая от имени учётной записи пользователя $user(s) = u$, может дать любой другой учётной записи пользователя право доступа α_r к сущности e. Помимо множества непосредственно заданных прав на предоставление прав Gr, в Gr_e включаются права иерархических владельцев сущностей, заданные множеством Gr_{own}. При этом субъект-сессия, функционирующая от имени одного из иерархических владельцев сущности, может дать любые права доступа к ней. Заметим, что множества Gr_e и Gr_{own} однозначно определяются множеством Gr, функцией иерархии сущностей H и функцией задания владельцев сущностей $owner$ и поэтому не рассматриваются в качестве самостоятельных элементов состояния системы.

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 α_r к контейнеру или сущности-процедуре $e \in C \cup O_p$ является наличие у u права доступа α_r к e.

Утверждение 3. $Gr_e \subseteq R_e$.

Доказательство. По способу задания $Gr_e = Gr \cup Gr_{own}$ и $Gr \subseteq R_e$. Покажем, что $Gr_{own} \subseteq R_e$. Пусть $e \in C \cup O_p$, $\alpha_r \in R_r$, $(u, e, \alpha_r) \in Gr_{own}$, где $u \in owners(e)$. В случае $u = owner(e)$, согласно способу задания множества R_{own}, выполняется $(u, e, \alpha_r) \in R_{own} \subseteq R_e$. Если же $u \neq owner(e)$, то, согласно утверждению 2, существует контейнер $c \in C$, такой, что $e < c$ и $u = owner(c)$. Так как $u = owner(c)$, то по способу задания множества R_{own} выполняется $(u, c, \alpha_r) \in R_{own}$. Таким образом, выполняется $u \in U$, $e \in C \cup O_p$, $\alpha_r \in R_r$ и существует контейнер $c \in C$, такой, что $(u, c, \alpha_r) \in R_{own}$ и $e < c$. Следовательно, согласно способу задания множества R_H, выполняется $(u, e, \alpha_r) \in R_H \subseteq R_e$. ■

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 $op_1, \ldots, op_k$ в начальном состоянии 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 СУБД ДП-модели определены приведённые в табл. 1 и 2 правила преобразования состояний из множества OP, в которых учтены особенности управления доступом в современных СУБД.

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

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

Запись вида (tr, op), использованная при определении нового значения tr', означает, что в возможно пустую последовательность tr добавляются данные о применяемом правиле op.

В результате применения внутреннего правила $switch(s, o, u, op)$ в качестве учётной записи, от имени которой функционирует субъект-сессия s, устанавливается u, а o становится функционально-ассоциированной с s сущностью. Параметр $op \in OP_{ext}$ задаёт внешнее правило, в результате применения которого было инициировано выполнения правила $switch(s, o, u, op)$.

Правило $revert(s, u, o)$ предназначено для восстановления состояния субъект-сессии s до активации ею сущности-триггера или сущности-процедуры. В результате применения правила $revert(s, u, o)$ в качестве учётной записи, от имени которой функционирует субъект-сессия s, устанавливается u, а o становится функционально-ассоциированной с s сущностью.

Внутреннее правило $execute(s, o, op)$ задаёт состояние системы после выполнения SQL-кода, содержащегося в сущности-процедуре или сущности-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 устанавливается в соответствии с режимом выполнения кода сущности-процедуры или сущности-триггера o, задаваемой значением функции $execute\_as(o)$. Кроме того, 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 производится восстановление исходной функционально-ассоциированной сущности с субъект-сессией s, а также учётной записи, от имени которой она выполнялась до применения внутреннего правила $execute(s, o, op)$. Параметр $op \in OP_{ext}$ задаёт внешнее правило, в результате применения которого было инициировано выполнение правила $execute(s, o, op)$.

Применение внутреннего правила $do\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})$ приводит к добавлению сущности-данные o_{d_1} в таблицу t_1, при этом содержимое сущности-данные o_{d_2} копируется во вновь создаваемую сущность-данные o_{d_1}.

Применение внутреннего правила $do\_update(s, t_1, o_{d_1}, t_2, o_{d_2})$ приводит к замене содержимого сущности-данные o_{d_1} на содержимое сущности-данные o_{d_2}.

Внутреннее правило $do\_delete(s, t, o_d)$ позволяет активировавшей его субъект-сессии s выполнить удаление сущности-данные o_d из содержащей её таблицы t.

В табл. 2 приведены внешние правила преобразования состояний из множества OP_{ext}, которые могут быть непосредственно применены субъект-сессиями.

Поясним приведённые в табл. 2 условия и результаты применения внешних правил преобразования состояний.

В результате применения правила $create\_session(u, s)$ создаётся новая субъект-сессия s, которая в дальнейшем может выполнять действия от имени учётной записи пользователя $u \in U$.

Предположение 3. Будем считать, что все правила из табл. 1 и 2, за исключением правила $create\_session(u, s)$, получают в качестве первого параметра субъект-сессию, инициировавшую выполнение данного правила. В рамках вновь созданной субъект-сессии повторное применение правила $create\_session(u, s)$ не допускается.

Определение 5. Будем говорить, что применение правила $op \in OP$ инициировано учётной записью пользователя $u \in U$, если либо $op = create\_session(u, s)$, либо op имеет в качестве первого параметра субъект-сессию s, такую, что $u = user(s)$.

Отметим, что в реальных СУБД учётная запись пользователя, от имени которой инициировано выполнение SQL-кода, также не может быть задана явным образом.

Определение 6. 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 $op \in OP_{ext}$ и либо op имеет вид $create\_session(u, s)$, 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.

Замечание 5. Согласно предположению 3, применение правила $create\_session(u, s)$ не может быть инициировано ранее созданной субъект-сессией. Следовательно, правило $create\_session(u, s)$ может быть инициировано только извне.

Допустим, заданы состояния системы $G_0, \ldots, G_k$, где $k \geqslant 1$, и внешние правила преобразования состояний $op_1, \ldots, op_k \in OP_{ext}$, 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 учётной записью пользователя $u \in U_0$, причем $op_1 = create\_session(u, s_1)$. Предположим, что в результате применения указанных правил были созданы субъект-сессии $s_1, \ldots, s_m$, где $m \geqslant 1$. Согласно замечанию 5, субъект-сессии могут быть созданы только в результате применения правил, инициированных извне. Следовательно, для каждого $i \in \{1, \ldots, m\}$ существует $j \in \{1, \ldots, k\}$, такой, что $op_j = create\_session(u, s_i)$. Обозначим через $op'_1, \ldots, op_{k'}$, где $k' \geqslant 1$, последовательность правил, полученную из $op_1, \ldots, op_k$ путём исключения всех правил вида $create\_session(u, s_i)$, $i \geqslant 2$, и заменой всех случаев использования s_i на s_1. Пусть состояния системы $G_0, \ldots, G_{k'}$ таковы, что $G_0 \vdash_{op'_1} G'_1 \vdash_{op'_2} \ldots \vdash_{op'_{k'}} G'_{k'}$. Заметим, что состояние $G'_{k'}$ удовлетворяет соотношению $G'_{k'}\{S'_{k'}\} \sim G_k, S'_{k'} = S_k \backslash \{s_2, \ldots, s_m\}$. Следовательно, без ограничения общности, будем считать, что выполняется следующее предположение.

Предположение 4. В процессе перехода системы из состояния в состояние для каждой учётной записи пользователя создаётся не более чем одна соответствующая ей субъект-сессия. Будем считать, что в начальном состоянии G_0 системы $\Sigma(G^*, OP)$ для каждой учётной записи пользователя $u \in U$ существует единственная соответствующая ей субъект-сессия $s \in S$.

Замечание 6. Как видно из табл. 2, в результате применения внешних правил производится изменение значения счётчика времени. В большинстве правил значение этого счётчика увеличивается на единицу. Однако в результате выполнения SQL-кода

сущностей-процедур и сущностей-триггеров (при применении правил *access_insert*, *access_update*, *access_delete* и *execute_procedure*) значение счётчика времени увеличивается на величину, большую чем единица.

Определение 7. 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 $tr_k = (tr_0, \widehat{op}_1, \ldots, \widehat{op}_m)$, 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 7. 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 $G^* \times OP^*_{ext} \to OP^*$.

Замечание 8. Отметим, что в соответствии со способом задания перехода системы из состояния в состояние последовательность правил преобразования состояний $vestige(G_0, op_1, \ldots, op_k)$ 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 $\widehat{G}_0 \vdash_{\widehat{op}_1} \widehat{G}_1 \vdash_{\widehat{op}_2} \ldots \vdash_{\widehat{op}_m} \widehat{G}_m$. Заметим, используя приведённые выше обозначения, что функция *vestige* обладает следующими свойствами.

Свойство 1. $G_k = \widehat{G}_m$.

Свойство 2. Для каждого $i = 1, \ldots, m$ выполняется $\widehat{time}_i = \widehat{time}_{i-1} + 1$.

Свойство 3. $vestige(G_0, op_1, \ldots, op_k) = (vestige(G_0, op_1, \ldots, op_{k-1}), vestige(G_{k-1}, op_k)) = (vestige(G_0, op_1), vestige(G_1, op_2), \ldots, vestige(G_{k-1}, op_k))$.

Свойство 4. Если в состоянии G_0 не выполнены условия применения правила $op \in OP_{ext}$, то $vestige(G_0, op) = ()$.

Свойство 5. Если выполнены условия применения правила $op \in OP_{ext}$ в состоянии G_0 и *op* не является правилом вида *access_insert*, *access_update*, *access_delete* и *execute_procedure*, то $vestige(G, op) = (op)$.

Свойство 6. $vestige(G_0, op_1, \ldots, op_k)$ не может включать правила преобразования состояний следующего вида: *access_insert*, *access_update*, *access_delete*, *execute* и *execute_procedure* — и может включать правила следующего вида: *switch*, *revert*, *do_insert*, *do_update*, *do_delete*, *create_session*, *create_procedure*, *alter_procedure*, *create_trigger*, *alter_trigger* или *grant_right*.

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

В результате применения правила $grant\_right(s, u, e, \alpha_r, with\_grant)$ учётной записи пользователя u предоставляется право доступа α_r к сущности e. Необходимым условием является наличие у учётной записи, от имени которой выполняется субъект-сессия s, права давать право доступа α_r к сущности e, то есть $user(s) \in Gr(e, \alpha_r)$. В случае если параметр *with_grant* равен `yes`, то самой учётной записи пользователя u даётся право предоставлять право α_r на сущность e.

Правило $create\_container(s, c_p, c)$ позволяет субъект-сессии s создать новый контейнер c, включив его в существующий контейнер c_p, при этом владельцем вновь со-

зданного контейнера становится учётная запись пользователя $user(s)$. Необходимым условием применения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к контейнеру c_p.

Замечание 9. В результате применения правил $alter\_procedure$, $alter\_trigger$ и $create\_container$ в множества R и Gr не выполняется добавление 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.

В результате применения правила $create\_procedure(s, c, o_p, md, op_1, \ldots, op_k)$ в контейнере c, не являющемся таблицей, создаётся новая сущность-процедура o_p, содержащая SQL-код, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$, при этом владельцем вновь созданной сущности-процедуры o_p становится учётная запись пользователя $user(s)$, а режим выполнения o_p устанавливается равным m. Необходимым условием выполнения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к контейнеру c.

В результате применения правила $alter\_procedure(s, o_p, md, op_1, \ldots, op_k)$ выполняется изменение режима выполнения существующей сущности-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 применения данного правила является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к сущности-процедуре o_p. Отметим, что в отличие от правила $create\_procedure$ владелец сущности-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 новая сущность-триггер o_t, содержащая SQL-код, выполнение которого приводит к реализации правил преобразования состояний $op_1, \ldots, op_k$, при этом владельцем вновь созданной сущности-триггера o_t становится учётная запись пользователя $user(s)$, а режим выполнения o_t задаётся равным m. Созданная сущность-триггер активизируется только при реализации к таблице t права доступа α_r. Если создаваемая сущность-триггер имеет тип $\alpha_r = append_r$, то она активизируется при применении в отношении таблицы t внешнего правила $access\_insert$. В случае если o_t имеет тип $\alpha_r = write_r$, то o_t активизируется при применении в отношении таблицы t внешнего правила $access\_update$. Если же $\alpha_r = delete_r$, то сущность-триггер o_t активизируется при применении к таблице t внешнего правила $access\_delete$. Необходимым условием применения правила $create\_trigger$ является наличие у учётной записи пользователя $user(s)$ права доступа $alter_r$ к таблице t.

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

Предположение 5. Будем считать, что выполнение SQL-кода сущностей-триггеров и сущностей-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. Из данного предпо-

ложения следует, что при применении правил $create\_procedure(\ldots, op_1, \ldots, op_k)$, $alter\_procedure(\ldots, op_1, \ldots, op_k)$, $create\_trigger(\ldots, op_1, \ldots, op_k)$ и $alter\_trigger(\ldots, op_1, \ldots, op_k)$ ни одно из правил $op_1, \ldots, op_k$ не должно быть равно $create\_procedure$, $alter\_procedure$, $create\_trigger$ или $alter\_trigger$.

Замечание 10. Заметим, что если $o \in O_p \cup O_t$, $operations(o) = (op_1, \ldots, op_k)$, где $k \geqslant 0$, то для каждого $i = 1, \ldots, k$ выполняется неравенство $op_i \neq \neq create\_session$. Предположим противное: существует $i \in \{1, \ldots, k\}$, такое, что $op_i = create\_session$. Согласно способу задания правил преобразования состояний системы, выполнение правил сущностей-триггеров и сущностей-процедур o осуществляется ранее созданной субъект-сессией $s \in S$. Следовательно, применение правила $op_i = create\_session$ инициируется субъект-сессией s, что противоречит предположению 3 о том, что в рамках вновь созданной субъект-сессии повторное применение правила $create\_session$ не допускается. Следовательно, правила, связанные с сущностями-процедурами и сущностями-триггерами, не могут включать правила вида $create\_session$. Кроме того, при применении правил $create\_procedure(s, c, o_p, md, op_1, \ldots, op_k)$, $alter\_procedure(s, o_p, md, op_1, \ldots, op_k)$, $create\_trigger(s, t, o_t, m, op_1, \ldots, op_k)$ и $alter\_trigger(s, t, o_t, md, op_1, \ldots, op_k)$ ни одно из правил $op_1, \ldots, op_k$ не должно иметь вид $create\_session$.

Замечание 11. Допустим, что $o \in O_p \cup O_t$, $operations(o) = (op_1, \ldots, op_k)$, где $k \geqslant 0$, $op \in operations(o)$, то есть существует $i \in \{1, \ldots, k\}$, такое, что $op = op_i$. Следовательно, в соответствии с предположением 5 и замечанием 10 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 $op \in OP_{ext}$ по определению функции $operations$, получаем, что в соответствии с табл. 2 op может иметь вид $grant\_right$, $execute\_procedure$, $access\_insert$, $access\_update$ или $access\_delete$.

Замечание 12. Из способа задания внешних правил видно, что владельцем созданной сущности (сущности-триггера, сущности-процедуры или контейнера) становится учётная запись пользователя, от имени которой выполнялась субъект-сессия, инициировавшая её создание. В соответствии с предположением 2 в системе отсутствуют внешние и внутренние правила, позволяющие изменять учётную запись владельца сущности.

Правило $execute\_procedure(s, o_p)$ позволяет вызывающей субъект-сессии s при наличии права доступа $execute_r$ к сущности-процедуре o_p выполнить её SQL-код. При этом результирующее состояние G' есть результат выполнения внутреннего правила $execute(s, o_p, execute\_procedure(s, o_p))$. Отметим, что при выполнении SQL-кода сущности-процедуры может осуществляться активация других сущностей-процедур и сущностей-триггеров.

Замечание 13. Будем считать, что при активации правил преобразования состояний, связанных с сущностями-процедурами и сущностями-триггерами, не возникает циклов, являющихся результатом повторного применения правил ранее активированных сущностей.

Применение правила $access\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})$ приводит к добавлению сущности-данные o_{d_1} в таблицу t_1, при этом содержимое сущности-данные o_{d_2} копируется во вновь создаваемую сущность-данные o_{d_1}. При применении правила $access\_insert$ выполняется активация всех сущностей-триггеров типа $append_r$, связанных с таблицей t_1. Необходимым условием применения данного правила является наличие у учётной записи пользователя $user(s)$ прав доступа на чтение к таблице t_2, а также права

доступа на запись к целевой таблице t_1, в которую выполняется добавление сущности-данные o_{d_1}.

В результате применения правила $access\_update(s, t_1, o_{d_1}, t_2, o_{d_2})$ выполняется обновление сущности-данные o_{d_1} с использованием данных, содержащихся в сущности-данные o_{d_2}. При применении правила $access\_update$ выполняется активация сущностей-триггеров типа $write_r$ таблицы, содержащей сущность-данные o_{d_1}. Необходимым условием применения данного правила является наличие у учётной записи пользователя $user(s)$ прав доступа на чтение к таблице t_2, а также права доступа на запись к целевой таблице t_1.

Правило $access\_delete(s, t, o_d)$ позволяет активировавшей его субъект-сессии s выполнить удаление сущности-данные o_d из содержащей её таблицы. При применении правила $access\_delete$ выполняется активация всех сущностей-триггеров типа $delete_r$, связанных с таблицей t. Необходимым условием применения данного правила является наличие у учётной записи пользователя $user(s)$ прав доступа на удаление к таблице t.

Замечание 14. Заметим, что условия применения правил $access\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})$, $access\_update(s, t_1, o_{d_1}, t_2, o_{d_2})$ и $access\_delete(s, t, o_d)$ ослаблены по сравнению с соответствующими внутренними правилами $do\_insert(s, t_1, o_{d_1}, t_2, o_{d_2})$, $do\_update(s, t_1, o_{d_1}, t_2, o_{d_2})$ и $do\_delete(s, t, o_d)$. Отличие в условиях применения указанных правил состоит в отсутствии требований, связанных с наличием или отсутствием сущностей-данных в соответствующих таблицах. Необходимым условием применения перечисленных правил остаётся только наличие соответствующих прав доступа к таблицам. Отметим, что указанные ослабления условий применения правил $access\_insert$, $access\_update$ и $access\_delete$ потенциально могут привести к появлению дополнительных путей распространения прав доступа в системе. Если в дальнейшем будут определены достаточные условия, при выполнении которых невозможно распространение прав доступа, то эти достаточные условия будут применимы и при наличии дополнительных ограничений на условия применения правил.

Замечание 15. В соответствии со способом задания применение внешних правил (за исключением $access\_delete$) не приводит к удалению элементов из множеств, являющихся элементами состояния системы $\Sigma(G^*, OP)$. Учитывая, что в условиях применения правил $grant\_right$, $alter\_procedure$, $alter\_trigger$, $execute\_procedure$, $access\_insert$, $access\_update$ и $access\_delete$ нет проверок на отсутствие элементов в состоянии системы, отметим, что верно следующее замечание. Если в состоянии G_0 системы $\Sigma(G^*, OP)$ выполнялись условия применения правила $op \in OP$, являющегося одним из перечисленных выше правил, и существуют правила преобразования состояний $op_1, \ldots, op_k \in OP$, где $k \geqslant 0$, то условия применения правила $op \in OP_{ext}$ будут выполняться и в состоянии $G' = G(op_1, \ldots, op_k)$.

3. Условия передачи прав доступа

При отсутствии ограничений на кооперацию субъект-сессий верно следующее утверждение о передаче прав доступа учётными записям пользователей к контейнерам и сущностям-процедурам.

Утверждение 4. Пусть G_0 — состояние системы $\Sigma(G^*, OP)$, $u \in U_0$, $e \in C_0 \cup O_{p_0}$ и пусть $\alpha_r \in R_r$ — некоторое право доступа. Тогда существуют состояния $G_1, \ldots, G_k$, где $k \geqslant 0$, правила преобразования состояний $op_1, \ldots, op_k$, такие, что $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \ldots \vdash_{op_k} G_k$, $(u, e, \alpha_r) \in R_{e_k}$ и для каждого $i = 1, \ldots, k$ переход $G_{i-1} \vdash_{op_i} G_i$ инициирован извне.

Доказательство. Если $(u, e, \alpha_r) \in R_{e_0}$, то $k = 0$ и условия утверждения выполнены. Пусть $(u, e, \alpha_r) \notin R_{e_0}$. Заметим, что ввиду $Gr_{own_0} \subseteq Gr_0$ и способа задания множеств Gr_{own_0} и $Gr_0(e, \alpha_r)$ выполняется соотношение

$$\{(u, e, \alpha_r) : u \in owners_0(e)\} \subseteq Gr_0(e, \alpha_r). \tag{4}$$

Так как множество $owners_0(e)$ включает, как минимум, учётную запись пользователя $owner(e)$, то из (4) следует, что множество $Gr_0(e, \alpha_r)$ не пусто и существует пользователь $u' \in Gr_0(e, \alpha_r)$. Рассмотрим следующую последовательность внешних правил: $op_1 = create\_session(u', s)$, $op_2 = grant\_right(s, u, e, \alpha_r, no)$. Тогда существуют состояния G_1 и G_2, такие, что $G_0 \vdash_{op_1} G_1 \vdash_{op_2} G_2$. Заметим, что выполнены условия применения правила op_2, так как $u' \in Gr_0(e, \alpha_r)$, а следовательно, $user_1(s) = u' \in \in Gr_1(e, \alpha_r) = Gr_0(e, \alpha_r)$. Отсюда $(u, e, \alpha_r) \in R_{e_2}$ и все переходы $G_0 \vdash_{op_1} G_1$, $G_1 \vdash_{op_2} G_2$ инициированы извне. ■

Из данного утверждения следует, что при отсутствии ограничений на кооперацию субъект-сессий иерархические владельцы сущности могут предоставить к ней произвольное право доступа любой учётной записи.

Заключение

Таким образом, для обеспечения возможности теоретического анализа безопасности СУБД построена формальная модель, в которую по сравнению с РОСЛ ДП-моделью добавлены несколько новых существенных элементов, позволяющих полнее учесть особенности управления доступом в СУБД. В дальнейшем планируется развитие построенной модели по следующим направлениям: анализ условий получения прав доступа при отсутствии кооперации между субъект-сессиями, определение условий активации сущностей-процедур и сущностей-триггеров от имени заданных учётных записей, а также уточнение правил преобразования состояний для определения условий реализации информационных потоков в случаях наличия или отсутствия кооперации субъект-сессий.

ЛИТЕРАТУРА

1. Руководящий документ. Безопасность информационных технологий. Критерии оценки безопасности информационных технологий: в 3 ч. Введён в действие Приказом Гостехкомиссии России от 19.06.02 г. № 187.
2. *Девянин П. Н.* Модели безопасности компьютерных систем. Управление доступом и информационными потоками: учеб. пособие для вузов. М.: Горячая линия-Телеком, 2011. 320 с.
3. *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.
4. *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.