КАРАГАНДИНСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ИВС

Дисциплина «Проектирование и разработка приложений БД»

МОДЕЛИ УПРАВЛЕНИЯ ДОСТУПОМ

DAC, MAC, RBAC, ABAC

ОП «Информационные системы, IT-медицина»

Авторы:
ст.преп. каф. ИВС Саданова Б.М.
ст.преп. каф. ИВС Олейникова А.В.

DAC и MAC: дискреционное и мандатное управление

Две фундаментальные модели: DAC — гибкость и простота, MAC — максимальная безопасность через централизованный контроль

DAC (Discretionary Access Control)

Дискреционное управление: владелец ресурса сам решает, кому дать доступ.

- Владелец (owner) объекта назначает права доступа
- Используются Access Control Lists (ACL) — списки доступа
- Права: read, write, execute, delete, administer
- Пользователь может делегировать права другим

Преимущества:

- Простота реализации и понимания
- Гибкость — быстрое изменение прав
- Хорошо для совместной работы

MAC (Mandatory Access Control)

Мандатное управление: система централизованно назначает метки безопасности.

- Каждый субъект (пользователь) и объект (данные) имеют метки безопасности
- Метка = уровень секретности + категория (compartment)
- Пользователь НЕ может изменить свою метку или метку объекта
- Решения принимает система, а не пользователь

Модель Bell-LaPadula:

- No Read Up (простая свойство): субъект не может читать объект с более высоким уровнем
- No Write Down (*-свойство): субъект не может писать в объект с более низким уровнем

RBAC: ролевое управление доступом

Role-Based Access Control — доступ назначается через роли, а не напрямую пользователям

Принцип RBAC: пользователь получает права через назначенные ему роли.

Основные концепции:

- Пользователь (User) — субъект, запрашивающий доступ
- Роль (Role) — набор разрешений, соответствующий должностной функции
- Разрешение (Permission) — право выполнить операцию над объектом
- Сессия (Session) — активация подмножества ролей пользователя

RBAC Flow
User → assigned to Role(s) → inherits Permissions → accesses Object

Иерархия RBAC (NIST стандарт):

- RBAC0 — базовая модель: пользователи, роли, разрешения, назначения
- RBAC1 — иерархия ролей (inheritance): старшая роль наследует права младшей
- RBAC2 — ограничения (constraints): mutex roles, cardinality, prerequisite roles
- RBAC3 — комбинация RBAC1 + RBAC2

ABAC и сравнительный анализ моделей

Attribute-Based Access Control — самая гибкая модель, использующая атрибуты для принятия решений о доступе

ABAC (Attribute-Based Access Control)

Атрибутивное управление: доступ определяется политиками на основе атрибутов.

Три типа атрибутов:

- Subject attributes — атрибуты субъекта:
  — роль, отдел, уровень допуска, должность
- Resource attributes — атрибуты объекта:
  — тип, классификация, владелец, department
- Environment attributes — атрибуты среды:
  — время, местоположение, IP-адрес, устройство

Сравнение моделей управления доступом

	Гибкость	Безопасность	Сложность
DAC	Высокая	Низкая	Низкая
MAC	Низкая	Очень высокая	Высокая
RBAC	Средняя	Средняя	Средняя
ABAC	Очень высокая	Высокая	Очень высокая

Примеры применения:

- DAC — файловые системы, Google Drive, малые команды
- MAC — военные, правительство, ядерная энергетика
- RBAC — корпоративные системы, ERP, CRM
- ABAC — облака, healthcare, zero-trust, IoT

Выбор модели управления доступом: выводы

Рекомендации по выбору

• Малые организации / совместная работа:
→ DAC — простота и гибкость

• Высокая секретность / регуляторные требования:
→ MAC — максимальная защита
→ Bell-LaPadula модель

• Корпоративные системы / четкая иерархия:
→ RBAC — масштабируемость
→ RBAC1 (иерархия) + RBAC2 (constraints)

• Облака / динамичные среды / zero-trust:
→ ABAC — контекстно-зависимый доступ
→ RBAC как подмножество ABAC

• Комбинированный подход:
→ RBAC + ABAC (роли + атрибуты)
→ RBAC + RLS (роли + политики строк)
→ Наиболее распространен в enterprise

Ключевые выводы

• DAC — владелец решает, кому дать доступ (ACL)
• MAC — система решает по меткам (Bell-LaPadula)
• RBAC — доступ через роли, соответствующие должностям
• ABAC — доступ по атрибутам субъекта, объекта и среды
• Эволюция моделей:
DAC → MAC → RBAC → ABAC
Простота → Безопасность → Масштабируемость → Гибкость
• В современных СУБД чаще всего:
— RBAC для структуры прав
— RLS / VPD для детализации (аналог ABAC)
— Шифрование и аудит для дополнительной защиты
• Нет «лучшей» модели — выбор зависит от:
— Размера организации
— Чувствительности данных
— Регуляторных требований
— Динамичности среды

GRANT — предоставление привилегий

Команда GRANT назначает права пользователю или роли.

Объектные привилегии:

- SELECT — чтение данных
- INSERT — добавление строк
- UPDATE — изменение данных
- DELETE — удаление строк
- REFERENCES — создание внешних ключей
- EXECUTE — выполнение процедур/функций
- USAGE — использование последовательностей
- ALL PRIVILEGES — все права на объект

Системные привилегии:

- CREATE — создание объектов (TABLE, VIEW, SCHEMA)
- DROP — удаление объектов
- CONNECT — подключение к базе данных
- SUPERUSER — административные права

```
-- Предоставить SELECT на таблицу
GRANT SELECT ON employees TO analyst;

-- Несколько привилегий
GRANT SELECT, INSERT, UPDATE ON orders
TO manager;

-- Все права на объект
GRANT ALL PRIVILEGES ON employees TO
admin;

-- На все столбцы кроме одного
GRANT SELECT(id, name, dept) ON
employees TO public;
```

Только владелец объекта или суперпользователь может выдать привилегии.

REVOKE — отзывание привилегий

Команда REVOKE удаляет ранее выданные права.

Синтаксис:

```
REVOKE [привилегии] ON [объект] FROM [пользователь];
```

Варианты отзывания:

- CASCADE — отозвать у всех, кому передали
- RESTRICT — отказать, если есть зависимые

Поведение по умолчанию:

- PostgreSQL: CASCADE
- Oracle: CASCADE CONSTRAINTS
- SQL Server: CASCADE

Особенности:

- Отзыв у PUBLIC не затрагивает явные GRANT
- REVOKE ALL отзывает все привилегии
- Нельзя отозвать часть ALL PRIVILEGES

```
-- Отозвать SELECT
REVOKE SELECT ON employees FROM analyst;
-- Отозвать несколько прав
REVOKE INSERT, UPDATE ON orders FROM manager;
-- Отозвать все права (CASCADE)
REVOKE ALL PRIVILEGES ON employees FROM admin CASCADE;
-- RESTRICT: отказать, если есть зависимости
REVOKE SELECT ON employees FROM manager RESTRICT;
-- Ошибка: manager передал права другому!
-- Отозвать системную привилегию
REVOKE CREATE TABLE FROM developer;

-- Отозвать у роли
REVOKE ALL ON SCHEMA hr FROM hr_manager;
```

WITH GRANT OPTION — цепочки передачи прав

WITH GRANT OPTION позволяет получателю передавать привилегии дальше.

Цепочка передачи:

- Админ → Менеджер (WITH GRANT OPTION)
- Менеджер → Аналитик (WITH GRANT OPTION)
- Аналитик → Стажер

Проблема:

- Если отозвать у Менеджера — что делать с правами Аналитика и Стажера?

Решения:

- CASCADE — отозвать у всех в цепочке
- RESTRICT — запретить отзыв, пока есть зависимые привилегии

```
-- Создаем цепочку
GRANT SELECT ON employees TO manager
  WITH GRANT OPTION;
GRANT SELECT ON employees TO analyst
  WITH GRANT OPTION;  -- manager передает
GRANT SELECT ON employees TO intern;
-- analyst передает
-- Отзыв с CASCADE: все теряют права
REVOKE SELECT ON employees FROM manager CASCADE;
-- intern и analyst тоже лишаются SELECT
-- Отзыв с RESTRICT: ошибка, если есть цепочка
REVOKE SELECT ON employees FROM manager RESTRICT;
-- ERROR: dependent privileges exist
```

Реализация в PostgreSQL и Oracle

PostgreSQL

```
-- Создание роли
CREATE ROLE analyst NOLOGIN;
CREATE USER alice PASSWORD 'secret';
GRANT analyst TO alice;
-- Привилегии на таблицу
GRANT SELECT, INSERT
  ON employees
  TO analyst
  WITH GRANT OPTION;
-- Row-Level Security
CREATE POLICY dept_isolation
  ON employees
  FOR SELECT
  USING (dept = current_user());
-- Аудит
\set ECHO all
\du  -- список ролей
\dp  -- список привилегий
```

Oracle

```
-- Системные привилегии
GRANT CREATE SESSION,
      CREATE TABLE,
      CREATE VIEW
  TO developer;
-- Объектные привилегии
GRANT SELECT, INSERT
  ON hr.employees
  TO analyst
  WITH GRANT OPTION;
-- Роль с привилегиями
CREATE ROLE app_user;
GRANT SELECT ON app.orders TO app_user;
GRANT app_user TO alice;
-- VPD (Virtual Private Database)
DBMS_RLS.ADD_POLICY(...);
-- Аудит
AUDIT SELECT ON hr.employees;
SELECT * FROM DBA_TAB_PRIVS;
```

ТЕХНОЛОГИЯ TDE: ПРИНЦИП РАБОТЫ

Transparent Data Encryption (TDE) решает задачу защиты "данных в покое" (Data-at-Rest) без модификации клиентских приложений.

- **Шифрование «на лету» (on-the-fly):** Данные шифруются при записи на диск и дешифруются при чтении в кэш оперативной памяти (Buffer Pool).
- **Полная прозрачность:** С точки зрения разработчика и пользователя база данных выглядит и ведет себя стандартным образом.
- **Защита метаданных:** Автоматически шифруются файлы данных (.mdf), файлы логов транзакций (.ldf) и файлы резервных копий (бэкапы).

Поддержка в ведущих СУБД:

• **MS SQL Server:** Доступно начиная с версии 2008 (Enterprise), а с 2019 года — также в Standard Edition.

• **Oracle Database:** Компонент опции Advanced Security (TDE Tablespace & Columns).

• **MySQL & PostgreSQL:** Доступно в коммерческих enterprise-версиях или через сторонние расширения (например, pg_tde).

ИЕРАРХИЯ КЛЮЧЕЙ В АРХИТЕКТУРЕ TDE

Service Master Key (SMK)

Ключ уровня сервера. Создается автоматически при установке СУБД. Шифруется средствами операционной системы.

↓ ШИФРУЕТ

Database Master Key (DMK) / Сертификат

Хранится в системной базе данных master. Используется для защиты ключей баз данных.

↓ ШИФРУЕТ

Database Encryption Key (DEK)

Симметричный ключ (AES-256), который непосредственно шифрует файлы данных текущей базы данных.

Как это работает на практике:

СУБД при запуске считывает SMK, дешифрует DMK и сертификат, с помощью которого затем дешифрует DEK в оперативной памяти.

Преимущество двухуровневой схемы: Для смены ключа шифрования базы данных достаточно перегенерировать только DEK, не затрагивая SMK и всю инфраструктуру операционной системы. Это гарантирует быстродействие.

ШИФРОВАНИЕ НА УРОВНЕ СТОЛБЦОВ (CLE)

Column-Level Encryption — технология выборочного шифрования конкретных полей таблицы. Это идеальное решение для сверхчувствительных данных (персональные данные, номера кредитных карт, пароли).

Методы и реализации в СУБД:

В отличие от TDE, здесь шифрование выполняется на уровне СУБД с помощью встроенных криптографических функций.

Пример запроса (MS SQL Server):
INSERT INTO Users (UserName, Password)
VALUES ('Ivan', ENCRYPTBYKEY(KEY_GUID('UserKeys'), 'MyPassword'));

СУБД	Механизм / Функция шифрования
MS SQL Server	ENCRYPTBYKEY(), Always Encrypted
Oracle	Пакет DBMS_CRYPTO, TSDP
PostgreSQL	Расширение pgcrypto (AES, PGP)
MySQL	Функции AES_ENCRYPT() / AES_DECRYPT()

КЛИЕНТСКОЕ ШИФРОВАНИЕ: ALWAYS ENCRYPTED

Технология **Always Encrypted** (представленная в SQL Server) переносит процесс шифрования полностью на сторону клиента (драйвера приложения ADO.NET/JDBC).

- **СУБД "вслепую":** База данных никогда не видит ключи и расшифрованные значения данных.
- **Безопасность анклавов (Secure Enclaves):** Позволяет выполнять вычисления над зашифрованными данными непосредственно в СУБД внутри защищенной области памяти.

Режимы шифрования столбцов:

Детерминированное

Одному и тому же тексту всегда соответствует один и тот же шифртекст. Допускает операции поиска, равенства (WHERE, JOIN, GROUP BY).

Рандомизированное

Использует случайную соль. Одинаковые данные шифруются по-разному. Максимально безопасно, но исключает поиск и индексацию.

Что такое SQL-инъекция?

SQL Injection (SQLi) — это тип атаки, при котором злоумышленник внедряет вредоносный SQL-код в запрос к базе данных через входные данные приложения.

> *"Если программа доверяет пользователю и конкатенирует (склеивает) строки для создания SQL-запроса — она уязвима."*

⚠️ **OWASP Top 10:** Инъекции годами занимают верхние строчки рейтинга угроз.

🗄️ **Последствия:** Утечка данных, удаление таблиц, обход авторизации.

Механизм атаки

Атака происходит на стыке веб-интерфейса и логики сервера приложений. Злоумышленник обманывает парсер SQL, заставляя его воспринимать **данные** как часть **команд**.

Этапы:

1. Поиск поля ввода (логин, поиск, URL-параметр).
2. Внедрение управляющих символов (кавычки, комментарии).
3. Выполнение измененного запроса с правами СУБД.

Классический пример: Обход логина

Уязвимый PHP-код:

```
$query = "SELECT * FROM users WHERE user = '" . $_POST['username'] . "' AND pass = '" . $_POST['password'] . "'";
```

Если злоумышленник введет в поле username: admin' --

Итоговый запрос в СУБД:

```
SELECT * FROM users WHERE user = 'admin' --' AND pass = '...'
```

Результат: Всё, что после --, считается комментарием. Хакер залогинен как админ без пароля.

ТРИГГЕРЫ АУДИТА (AUDIT TRIGGERS)

- **Управление доступом:** Любые попытки изменения прав пользователей, создание учетных записей или смена паролей администраторов.
- **Доступ к объектам:** Обращение к чувствительным файлам, базам данных или ключам шифрования, особенно если доступ был отклонен (Permission Denied).
- **Аномальные входы:** Множественные неудачные попытки аутентификации (брутфорс) или вход в систему в нерабочее время из необычных локаций.
- **Системные изменения:** Установка нового ПО, изменение конфигурационных файлов сервисов или внесение правок в системный реестр и политики безопасности.

СИСТЕМНЫЕ ЖУРНАЛЫ

Источники телеметрии

Современная инфраструктура генерирует потоки данных из разных слоев. Журналы ОС фиксируют работу ядра и сервисов. Сетевые журналы (брандмауэры, маршрутизаторы) показывают трафик и попытки вторжений.
Журналы приложений и БД предоставляют контекст бизнес-логики, позволяя отследить конкретные транзакции и действия внутри софта, которые могут быть пропущены на уровне операционной системы.

СТАНДАРТЫ ЖУРНАЛОВ ОС

Тип лога	Windows (Event Viewer)	Linux (Syslog/Systemd)
Безопасность	Security.evtx (Аудит входов, доступ)	/var/log/auth.log или secure
Система	System.evtx (Драйверы, ошибки ОС)	/var/log/syslog или journalctl
Ядро	Setup.evtx (Установка обновлений)	/var/log/dmesg (Загрузка железа)
Приложения	Application.evtx (Ошибки софта)	Логи конкретных сервисов в /var/log/

Классификация уровней защиты данных

RLS (Row-Level)

Горизонтальная фильтрация: ограничивает доступ к конкретным строкам в таблице на основе контекста пользователя (например, регион, ID отдела или владелец записи).
Идеально для мультиарендных (SaaS) архитектур и разделения доступа по филиалам.

CLS (Column-Level)

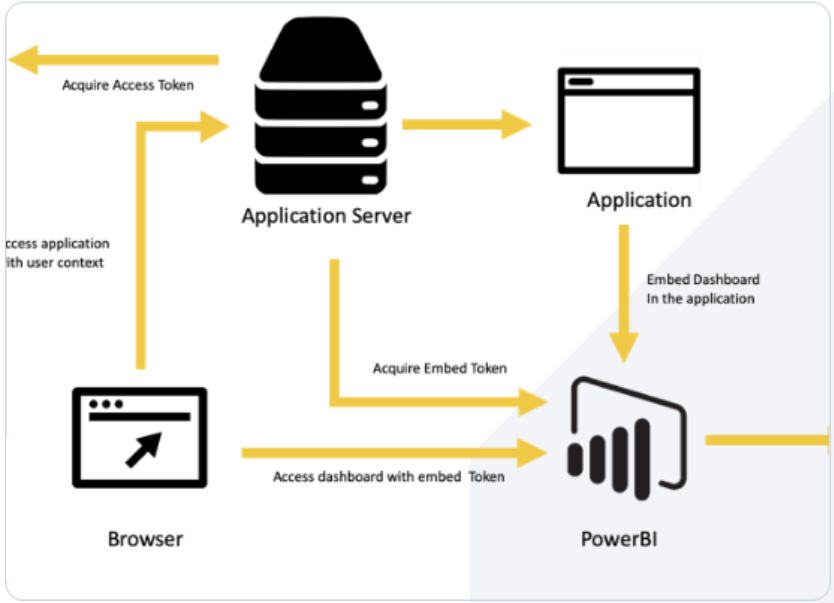
Вертикальная фильтрация: скрывает или маскирует конкретные столбцы с чувствительными данными (PII, зарплаты, медицинские данные).
Позволяет пользователю видеть таблицу целиком, но ограничивает видимость атрибутов.

RLS: Контекстная фильтрация записей

Row-Level Security обеспечивает автоматическое применение фильтров ко всем SQL-запросам без изменения кода приложения.

Основные преимущества:

- Централизация логики безопасности в БД.
- Снижение риска ошибок на стороне Front-end/API.
- Масштабируемость политик доступа.



Техническая реализация RLS

- **Создание предикатов:** Разработка функций (Security Predicates), которые возвращают логическое значение для каждой строки.
- **Политики безопасности:** Определение Security Policies, которые связывают предикаты с конкретными таблицами и операциями (SELECT, UPDATE).
- **Контекст сессии:** Использование системных переменных (например, SESSION_CONTEXT или USER_NAME()) для идентификации пользователя.
- **Производительность:** Оптимизация индексов для эффективной работы фильтрующих предикатов при больших объемах данных.