# Базы данных

## Модуль 2. Управление данными в базах данных

Лекция 9 -10. Управление транзакциями. Предоставление привилегий пользователям.

Преподаватель:
Семенов Геннадий Николаевич, к.т.н., доцент

2024

# Основные определения

Транзакцией называется последовательность операций, производимых над базой данных и переводящих базу данных из одного непротиворечивого (согласованного) состояния в другое непротиворечивое (согласованное) состояние.

Типы транзакций:

- плоские или классические транзакции,
- цепочечные транзакции,
- вложенные транзакции.

Плоские транзакции характеризуются 4 классическими свойствами:

- атомарности,
- согласованности,
- изолированности,
- долговечности (прочности)

***ACID (Atomicity, Consistency, Isolation,Durability).***

Иногда традиционные транзакции называют ACID-транзакциями.

# Свойства ACID

- ***Свойство атомарности*** (***Atomicity***) выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе. «Все или ничего». Транзакция - динамически составная операция над базой данных.
- ***Свойство согласованности*** (***Consistency***) гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое - транзакция не нарушает *целостность* базы данных(удовлетворяет набору ограничений *целостности).* Точки согласованности в стандарте SQL-1999 можно устанавливать.
- ***Свойство изолированности*** (***Isolation***) означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно. Результаты ***T1*** не должны быть видны никакой ***T2*** до тех пор, пока ***T1*** не завершится успешно.
- ***Свойство долговечности*** (***Durability***) трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае сбоев аппаратуры и программного обеспечения).

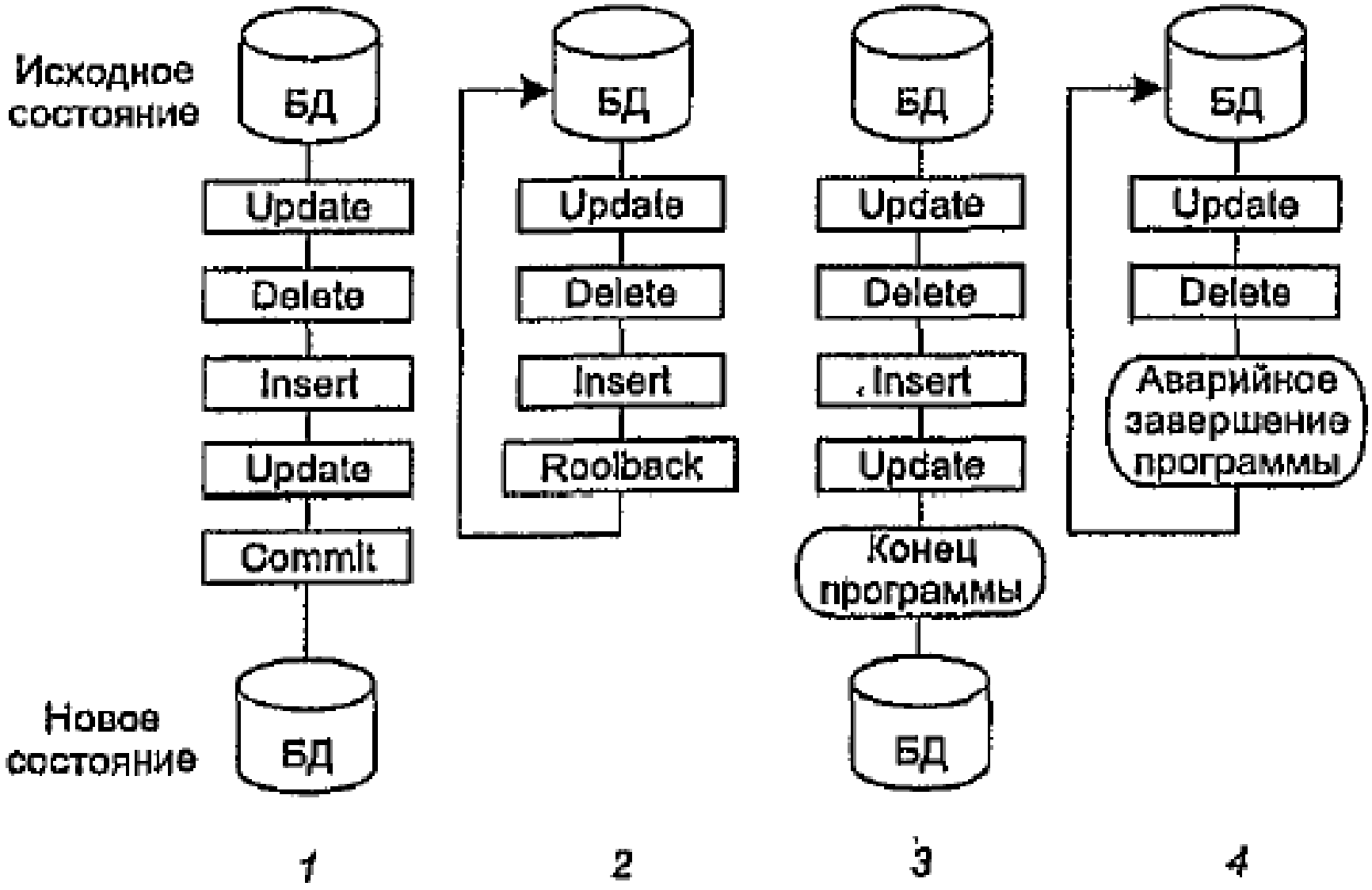
# Смысл концепции сериализации транзакций

- Для двух транзакций, скажем, A и B, возможны только два варианта упорядочения при их последовательном выполнении: сначала A, затем B или сначала B, затем A. Причем результаты реализации двух вариантов могут в общем случае не совпадать.
- Например, при выполнении двух банковских операций — внесения некоторой суммы денег на какой-то счет и начисления процентов по этому счету — важен порядок выполнения операций.
- Если первой операцией будет увеличение суммы на счете, а второй — начисление процентов, тогда итоговая сумма будет больше, чем при противоположном порядке выполнения этих операций.
- Если описанные операции выполняются в рамках двух различных транзакций, то оказываются возможными различные итоговые результаты, зависящие от порядка их выполнения.

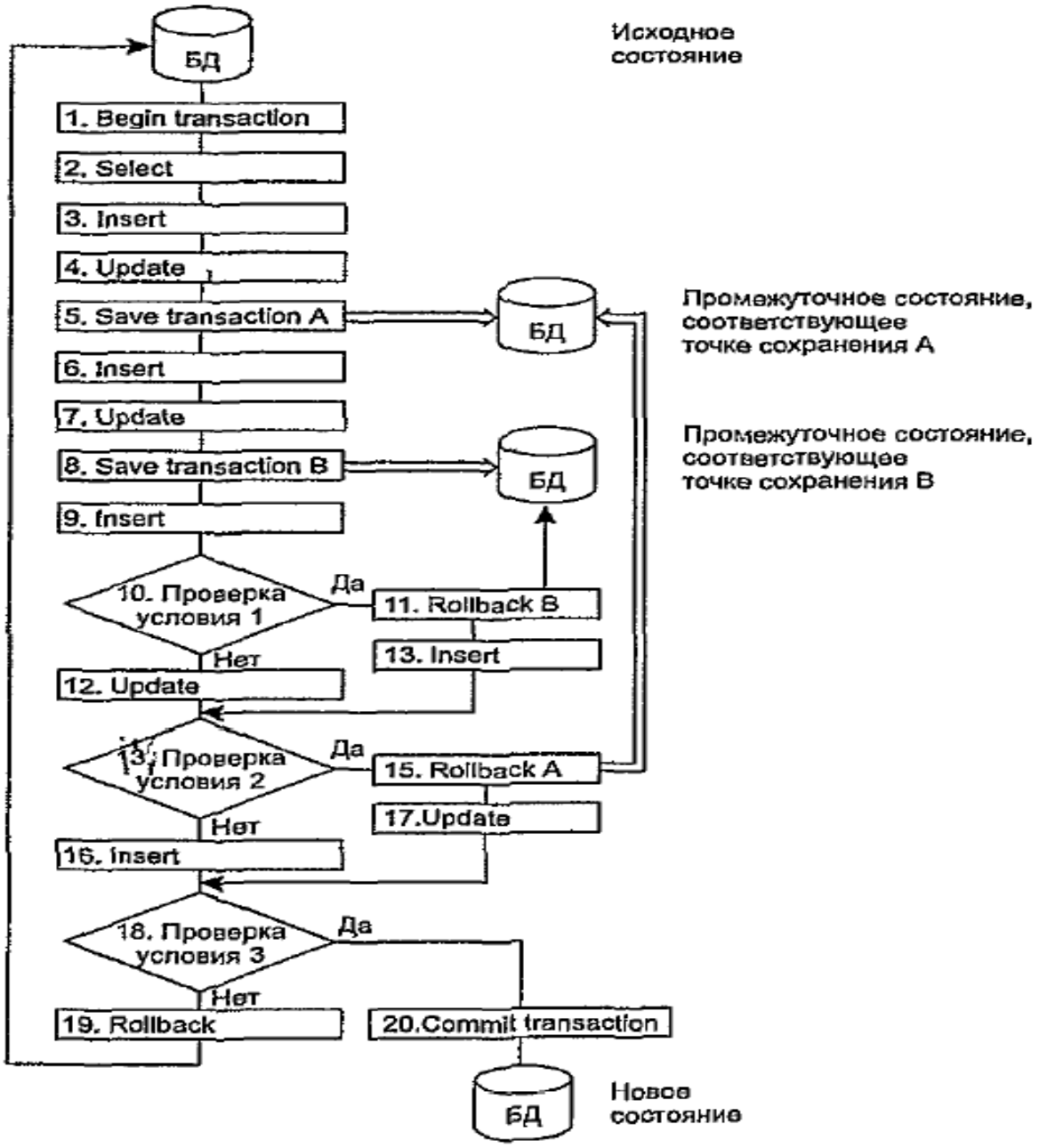
**Сериализация** двух транзакций при их параллельном выполнении означает, что полученный результат будет соответствовать *одному из двух возможных* вариантов упорядочения транзакций при их последовательном выполнении.

- При этом нельзя сказать точно, какой из вариантов будет реализован.
- Если параллельно выполняется более двух транзакций, тогда результат их параллельного выполнения также должен быть таким, каким он был бы в случае выбора *некоторого варианта упорядочения транзакций*, если бы они выполнялись последовательно, одна за другой. Чем больше транзакций, тем больше вариантов их упорядочения. Концепция сериализации не предписывает выбора какого-то определенного варианта. Речь идет лишь об *одном из них*.

# Возможные пути завершения транзакции.



# Расширенная модель транзакций.



# Расширенная модель транзакций.

- Оператор **BEGIN TRANSACTION** сообщает о начале транзакции
- Оператор **COMMIT TRANSACTION** сообщает об успешном завершении транзакции. Он эквивалентен оператору **COMMIT** в модели стандарта ANSI/ISO.
- Оператор **SAVE TRANSACTION** создает внутри транзакции точку сохранения, которая соответствует промежуточному состоянию БД, сохраненному на момент выполнения этого оператора.
- В операторе **SAVE TRANSACTION** может стоять имя точки сохранения. Поэтому в ходе выполнения транзакции может быть запомнено несколько точек сохранения, соответствующих нескольким промежуточным состояниям.
- Оператор **ROLLBACK** имеет две модификации.
- Если этот оператор используется без дополнительного параметра, то он интерпретируется как оператор отката всей транзакции, то есть в этом случае он эквивалентен оператору отката **ROLLBACK** в модели ANSI/ISO.
- Если же оператор отката имеет параметр и записан в виде **ROLLBACK B**, то он интерпретируется как оператор частичного отката транзакции в точку сохранения **B**.

# Общие принципы восстановления:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

**Ситуации, для восстановления состояния базы данных:**

- Индивидуальный откат транзакции.
- Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой).
- Восстановление после поломки основного внешнего носителя базы данных(жесткий сбой)

Для восстановления согласованного состояния базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

При **мягком сбое** для восстановления непротиворечивого состояния БД необходимо восстановить содержимое БД по содержимому журналов транзакций, хранящихся па дисках.

При **жестком сбое** для восстановления согласованного состояния БД надо восстановить содержимое БД по архивным копиям и журналам транзакции, которые хранятся на неповрежденных внешних носителях.

# Журнал транзакций

Реализация в СУБД принципа сохранения промежуточных состоянии, подтверждения пли отката транзакции обеспечивается специальным механизмом, для поддержки которого создается некоторая системная структура, называемая *Журналом транзакций.*

Два основных варианта ведения журнальной информации:

1. для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакцией. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.
2. поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов.

Протокол с отложенными обновлениями

Протокол с немедленными обновлениями.

# Конфликтные ситуации

При параллельном выполнении транзакций возможны следующие фено'мены:

1. **Потерянное обновление (lost update).** Когда разные транзакции одновременно изменяют одни и те же данные, то после фиксации изменений может оказаться, что одна транзакция перезаписала данные, обновленные и зафиксированные другой транзакцией.
2. **«Грязное» чтение (dirty read).** Транзакция читает данные, измененные параллельной транзакцией, которая еще не завершилась. Если эта параллельная транзакция в итоге будет отменена, тогда окажется, что первая транзакция прочитала данные, которых нет в системе.
3. **Неповторяющееся чтение (non-repeatable read).** При повторном чтении тех же самых данных в рамках одной транзакции оказывается, что другая транзакция успела изменить и зафиксировать эти данные. В результате тот же самый запрос выдает другой результат.
4. **Фантомное чтение (phantom read).** Транзакция повторно выбирает множество строк в соответствии с одним и тем же критерием. В интервале времени между выполнением этих выборок другая транзакция добавляет новые строки и успешно фиксирует изменения. В результате при выполнении повторной выборки в первой транзакции может быть получено другое множество строк.
5. **Аномалия сериализации (serialization anomaly).** Результат успешной фиксации группы транзакций, выполняющихся параллельно, не совпадает с результатом ни одного из возможных вариантов упорядочения этих транзакций, если бы они выполнялись последовательно.

# Конфликтные ситуации и уровни изоляции

1. неповторяющееся чтение (non-repeatable read);
2. "грязное" чтение (dirty read) - чтение данных, которые были записаны откатанной транзакцией;
3. потерянное обновление (lost update);
4. фантомная вставка (phantom insert).
5. аномалия сериализации (serialization anomaly).

**Уровни изоляции:**

- Стандарт SQL-92 определяет **уровни изоляции**, установка которых предотвращает определенные конфликтные ситуации. Введены следующие четыре уровня изоляции:
- **SERIALIZABLE - последовательное выполнение** (используется по умолчанию). Этот уровень гарантирует предотвращение всех описанных выше конфликтных ситуаций, но, соответственно, при нем наблюдается самая низкая степень параллелизма;
- **REPEATABLE READ - повторяющееся чтение**. На этом уровне разрешено выполнение операторов **INSERT**, приводящих к конфликтной ситуации "фантомная вставка". Этот уровень целесообразно использовать, если на выполняющиеся SQL-операторы не влияет добавление новых строк;
- **READ COMMITED - фиксированное чтение**. Этот уровень позволяет получать разные результаты для одинаковых запросов, но только после фиксации транзакции, повлекшей изменение данных;
- **READ UNCOMMITED - нефиксированное чтение**. Здесь возможно получение разных результатов для одинаковых запросов без учета фиксации транзакции.

Определение параметров транзакции выполняется оператором **SET TRANSACTION,**

# Блокировки

| Уровень изоляции | Предотвращение конфликтной ситуации | | | |
|---|---|---|---|---|
| | неповторяющееся чтение (non-repeatable read) | "грязное" чтение (dirty read) | потерянное обновление (lost update) | фантомная вставка (phantom insert) |
| SERIALIZABLE | + | + | + | + |
| REPEATABLE READ | + | + | + | - |
| READ COMMITED | - | + | + | - |
| READ UNCOMMITED | - | - | + | - |

Блокировки используются для приостановки выполнения одних SQL-операторов, пока выполняются другие.

•**оптимистические блокировки** (optimistic locks) - предотвращают возникновение конфликтных ситуаций, выполняя при необходимости откат

Если при пессимистической блокировке выполнен SQL-оператор, который может вызвать конфликтную ситуацию для другого SQL-оператора, то выполнение второго SQL-оператора будет приостановлено.

При оптимистической блокировке могут выполняться любые SQL-операторы, но в случае возникновения конфликтной ситуации все изменения будут потеряны.

# Блокировки

Кроме поддержки уровней изоляции транзакций, многие СУБД позволяют также создавать блокировки данных как на уровне отдельных строк, так и на уровне целых таблиц.

**Пример**:

1. Команда **SELECT** имеет предложение **FOR UPDATE**, которое позволяет заблокировать отдельные строки таблицы с целью их последующего обновления.
2. Если одна транзакция заблокировала строки с помощью этой команды, тогда параллельные транзакции не смогут заблокировать эти же строки до тех пор, пока первая транзакция не завершится, и тем самым блокировка не будет снята.
3. Таким образом если выполнять данную команду:

**SELECT** * **FROM** table_tame **WHERE** column_name ~ 'some text' **FOR UPDATE**;

на двух терминалах — сначала на одном — а затем на втором (с учетом начала транзакции **BEGIN**):

1. То можно заметить, что выполнение на втором терминале приостановится до тех пор пока не завершится транзакция первого терминала
2. При этом если на первом терминале выполнить какую-нибудь другую команду: **UPDATE** table_name **SET** column_name = 'kek' **WHERE** column_value = 404; то, перейдя на второй терминал станет видно, что там была, наконец, выполнена выборка, которая покажет уже измененные данные с учетом данной UPDATE-команды

# Реализация транзакций **в PostgreSQL**

Реализация транзакций в СУБД PostgreSQL основана на многоверсионной модели (Multiversion Concurrency Control, MVCC).

Эта модель предполагает, что каждый SQL-оператор видит так называемый **снимок данных (snapshot)**, т. е. то согласованное состояние (версию) базы данных, которое она имела на определенный момент времени.

**Снимок** – это не физическая копия всей базы данных, это несколько чисел, которые идентифицируют текущую транзакцию и те транзакции, которые уже выполнялись в момент начала текущей.

При этом параллельно исполняемые транзакции, даже вносящие изменения в базу данных, не нарушают согласованности данных этого снимка.

Такой результат в PostgreSQL достигается за счет того, что когда параллельные транзакции изменяют одни и те же строки таблиц, тогда создаются *отдельные версии* этих строк, доступные соответствующим транзакциям.

Это позволяет ускорить работу с базой данных, однако требует больше дискового пространства и оперативной памяти.

*Важное следствие применения MVCC — операции чтения никогда не блокируются операциями записи, а операции записи никогда не блокируются операциями чтения.*

# Уровни изоляции транзакций в PostgreSQL

1. **Read Uncommitted.** Это самый низкий уровень изоляции. Согласно стандарту SQL на этом уровне допускается чтение «грязных» (незафиксированных) данных. В **PostgreSQL** требования, предъявляемые к этому уровню, более строгие, чем в стандарте: чтение «грязных» данных на этом уровне не допускается.
2. **Read Committed**. Не допускается чтение «грязных» (незафиксированных) данных. Таким образом, **в PostgreSQL** уровень Read Uncommitted совпадает с уровнем Read Committed. Транзакция может видеть только те  незафиксированные изменения данных, которые произведены в ходе выполнения ее самой.
3. **Repeatable Read**. Не допускается чтение «грязных» (незафиксированных) данных и неповторяющееся чтение. В **PostgreSQL** на этом уровне не допускается также фантомное чтение. Таким образом, реализация этого уровня является более строгой, чем того требует стандарт SQL. Это не противоречит стандарту.
4. **Serializable**. Не допускается ни один из феноменов, перечисленных выше, в том числе и аномалии сериализации.

# Уровни изоляции транзакций в PostgreSQL

По умолчанию PostgreSQL использует уровень изоляции Read Committed.

```
SHOW default_transaction_isolation;
 default_transaction_isolation
-------------------------------
 read committed
```

# Защита информации в БД

# Подходы защиты информации в БД:

- избирательный подход
- обязательный подход.

В обоих подходах «объектом данных», для которых должна быть создана система безопасности, может быть как вся база данных целиком, так и любой объект внутри базы данных.

В случае **избирательного** подхода пользователь обладает различными **привилегиями** (полномочиями) при работе с данными объектами. Разные пользователи могут обладать разными правами доступа к одному и тому же объекту. (*гибкость*!)

В случае **обязательного** подхода, наоборот, каждому объекту данных присваивается некоторый классификационный уровень. Каждый пользователь обладает некоторым уровнем допуска к определенному объекту данных в соответствии со своим уровнем допуска

# Реализация избирательного подхода

- В базу данных вводится новый тип объектов БД — это пользователи.
- Каждому пользователю в БД присваивается уникальный идентификатор.
- Для дополнительной защиты каждый пользователь кроме уникального идентификатора снабжается уникальным паролем(известны только самим пользователям)
- Пользователи могут быть объединены в специальные группы пользователей
- Один пользователь может входить в несколько групп.
- В стандарте введено понятие группы **PUBLIC**, для которой должен быть определен минимальный стандартный набор прав.
- **Привилегии** пользователей или групп — это набор действий (операций), которые они могут выполнять над объектами БД.
- Вводится понятие **роль** — это поименованный набор привилегий. Имеется возможность создавать новые роли, группируя в них произвольные привилегии.
- Пользователю может быть назначена одна или несколько ролей.
- Существует ряд стандартных ролей, которые определены в момент установки сервера баз данных.

# Привилегии

**Привилегии** пользователей или групп — это набор действий (операций), которые они могут выполнять над объектами БД.

SQL поддерживает следующие привилегии:

- ALTER
- SELECT
- INSERT
- UPDATE
- DELETE
- REFERENCES
- INDEX
- DROP

В различных СУБД список расширен

# Привилегии роли

Привилегии пользователей или групп — это набор действий (операций), которые они могут выполнять над объектами БД.

**Роль** — это поименованный набор привилегий.

- Существует ряд стандартных ролей, которые определены в момент установки сервера баз данных.
- Имеется возможность создавать новые роли, группируя в них произвольные привилегии.
- Введение ролей не связано с конкретными пользователями, поэтому роли могут б
до того, как определ
- Пользователю може
ролей.
- Введение ролей поз
привилегиями польз
процесс.

Создание роли:

**CREATE ROLE** <роль>

Добавление привилегий в роль:

**GRANT** <привилегия> **TO** <роль>

Выдача роли пользователю:

**GRANT** <роль> **TO** <пользователь>

# Основные объекты БД, которые подлежат защите

Объектами БД, которые подлежат защите, являются все объекты, хранимые в БД:

- таблицы,
- представления,
- хранимые процедуры,
- триггеры

Для каждого типа объектов есть свои действия, поэтому для каждого типа объектов могут быть определены разные права доступа.

На элементарном уровне безопасности БД необходимо поддерживать два фундаментальных принципа:

1. проверку привилегий
2. проверку подлинности (аутентификацию).

# Система назначения привилегий

**Иерархическая модель**:

системный администратор(адм-р сервера БД) может создавать пользователей и наделять их определенными **правами** (**привилегиями)**.

СУБД в своих системных каталогах хранит как описание самих пользователей, так и описание их привилегий по отношению ко всем объектам.

- Каждый объект в БД имеет владельца - пользователя, который создал данный объект.
- Владелец объекта обладает всеми правами-привилегиями на данный объект,

в том числе он имеет право предоставлять другим пользователям привилегия по работе с данным объектом и забирать у пользователей ранее предоставленные привилегии.

# Оператор предоставления привилегий

**В стандарте SQL определены операторы:**

**GRANT - предоставления привилегий**

```
GRANT {<список действий> | ALL PRIVILEGES }
    ON <имя_объекта>
    TO {<имя_пользователя> | PUBLIC }
        [WITH GRANT OPTION ]
```

- **ALL PRIVILEGES** указывает, что разрешены все действия из допустимых для объектов данного типа.
- <**имя_объекта**> — задает имя конкретного объекта: таблицы, представления, хранимой процедуры, триггера.
- <**имя_пользователя**> или **PUBLIC** определяет, кому предоставляются данные привилегии.
- **WITH GRANT OPTION** (необязательным) и определяет режим, при котором передаются не только права на указанные действия, но и право передавать эти права другим пользователям.

# Оператор предоставления привилегий

**В стандарте SQL определены оператор:**

**REVOKE - отмены привилегий**

```
REVOKE {<список действий> | ALL PRIVILEGES }
    ON <имя_объекта>
    FROM{<список_пользователей> | PUBLIC }
    {CASCADE | RESTRICT}
```

Параметры **CASCADE** или **RESTRICT** определяют, каким образом должна производиться отмена привилегий.

- Параметр **CASCADE** отменяет привилегии не только пользователя, который непосредственно упоминался в операторе **GRANT** при предоставлении ему привилегий, но и всем пользователям, которым этот пользователь присвоил привилегии, воспользовавшись параметром **WITH GRANT OPTION**
- Параметр **RESTRICT** ограничивает отмену привилегий только пользователю, непосредственно упомянутому в операторе **REVOKE**.

# Пользователи СУБД :

- **Администратор сервера баз данных**. Он ведает установкой, конфигурированием сервера, регистрацией пользователей, групп, ролей и т.п. Прямо или косвенно он обладает всеми привилегиями, которые имеют или могут иметь другие пользователи.
- **Администраторы базы данных.** К этой категории относится любой пользователь, создавший базу данных, и, следовательно, являющийся ее владельцем. Он может предоставлять другим пользователям доступ к базе и к содержащимся в ней объектам.
- Администратор базы отвечает за ее сохранение и восстановление. В принципе в организации может быть много администраторов баз данных.
- Чтобы пользователь мог создать базу и стать ее администратором, он должен получить (вероятно, от администратора сервера) привилегию createdb.
- прочие (конечные) пользователи. Они оперируют данными, хранящимися в базах, в рамках выделенных им привилегий.

# Создание пользователя

**CREATE USER** <пользователь> **IDENTIFIED BY** <пароль> | EXTERNALLY

**DEFAULT TABLESPACE** <tablespace>

**TEMPORARY TABLESPACE** <tablespace>

**QUOTA** { <целое> K | M ON tablespace }

| **UNLIMITED**

# Раздача прав

GRANT INSERT

ON Tab

TO user2

GRANT SELECT

ON Tab

TO user3

Например, менеджер может изменить цену товара в нашей таблице

```
GRANT SELECT, UPDATE (COST)
ON Tab
TO user3
```

# Отмена привелегий

- Для отмены ранее назначенных привилегий в стандарте SQL определен оператор REVOKE.

REVOKE {<список операций> | ALL PRIVILEGES}
ON <имя_объекта>

FROM {<список пользователей> | PUBLIC }
{CASCADE | RESTRICT }

# Пример

Отнимаем права на вставку данных

REVOKE INSERT

ON Tab

TO user2, user4 CASCADE

# Роли

поименованный набор привилегий

Создание роли:

**CREATE ROLE** <роль>

Добавление привилегий в роль:

**GRANT** <привилегия> **TO** <роль>

Выдача роли пользователю:

**GRANT** <роль> **TO** <пользователь>

# Роли

Удаление роли:

**DROP ROLE** <роль>

Удаление  привилегий из роли:

**REVOKE** <привилегия> **FROM** <роль>

Удаление роли у пользователя:

**REVOKE** <роль> **FROM** <пользователь>

# Пример

- **User1** создал объект **Tab**, он является владельцем этого объекта и может передать права на работу с эти объектом другим пользователям.
- Пользователь **user2** является оператором, который должен вводить данные в **Tab**(например, таблицу новых заказов)
- Пользователь **user3** например, менеджером отдела, который должен регулярно просматривать введенные данные.

- Для объекта типа таблица полным допустимым перечнем действий является набор из четырех операций: **SELECT**, **INSERT**. **DELETE**, **UPDATE**. При этом операция обновление может быть ограничена несколькими столбцами.

- **GRANT** { [**SELECT**] [,**INSERT**][,**DELETE**] [,**UPDATE** (<список столбцов>] } **ON** <имя_таблицы> **TO** {<имя_пользователя> | PUBLIC } [WITH GRANT OPTION ]

- **GRANT INSERT** ON **Tab1** TO **user2**
- **GRANT SELECT** ON **Tab1** TO **user3**

# Привилегии в MySQL

Синтаксис команды **GRANT**:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    TO user_name [IDENTIFIED BY [PASSWORD] 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [REQUIRE
     NONE |
        [{SSL| X509}]
        [CIPHER cipher [AND]]
        [ISSUER issuer [AND]]
        [SUBJECT subject]]
    [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
                          MAX_UPDATES_PER_HOUR # |
                          MAX_CONNECTIONS_PER_HOUR #]]
```

# в PostgreSQL

Команда **GRANT** имеет две основные разновидности: первая назначает права для доступа к объектам баз данных
(таблицам, столбцам,
представлениям,
сторонним таблицам,
последовательностям,
базам данных,
обёрткам сторонних данных,
сторонним серверам,
функциям,
процедурам,
процедурным языкам,
схемам или табличным пространствам),
а вторая назначает одни роли членами других. Эти разновидности во многом похожи, но имеют достаточно отличий, чтобы рассматривать их отдельно.

**Примеры**:

Следующая команда разрешает всем добавлять записи в таблицу **films**:

```
GRANT INSERT ON films TO PUBLIC;
```

Эта команда даёт пользователю **manuel** все права для представления **kinds**:

```
GRANT ALL PRIVILEGES ON kinds TO manuel;
```

# Роль в PostgreSQL

**CREATE ROLE имя [ [ WITH ] параметр [ ... ] ]**

Здесь параметр:

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| BYPASSRLS | NOBYPASSRLS
| CONNECTION LIMIT предел_подключений
| [ ENCRYPTED ] PASSWORD 'пароль'
| PASSWORD NULL
| VALID UNTIL 'дата_время'
| IN ROLE имя_роли [, ...]
| IN GROUP имя_роли [, ...]
| ROLE имя_роли [, ...]
| ADMIN имя_роли [, ...]
| USER имя_роли [, ...]
| SYSID uid

- **Параметры**
- ***Имя*** Имя создаваемой роли.
- **SUPERUSER, NOSUPERUSER**

Эти предложения определяют, будет ли эта роль «**суперпользователем**», который может переопределить все ограничения доступа в базе данных. Статус **суперпользователя** несёт опасность и назначать его следует только в случае необходимости. Создать нового **суперпользователя** может только **суперпользователь**. В отсутствие этих предложений по умолчанию подразумевается **NOSUPERUSER**.

- **CREATEDB**
  **NOCREATEDB**

Эти предложения определяют, сможет ли роль создавать базы данных. Указание **CREATEDB** даёт новой роли это право, а **NOCREATEDB** запрещает роли создавать базы данных. По умолчанию подразумевается **NOCREATEDB**.

- **CREATEROLE**, **NOCREATEROLE**

Эти предложения определяют, сможет ли роль создавать новые роли (т. е. выполнять CREATE ROLE). Роль с правом **CREATEROLE** может также изменять и удалять другие роли. По умолчанию подразумевается **NOCREATEROLE**.

- **INHERIT**, **NOINHERIT**

Эти предложения определяют, будет ли роль «наследовать» права ролей, членом которых она является. Роль с атрибутом **INHERIT** может автоматически использовать в базе данных любые права, назначенные всем ролям, в которые она включена, непосредственно или опосредованно. По умолчанию подразумевается **INHERIT**.

- **LOGIN**, **NOLOGIN** Эти предложения определяют, разрешается ли новой роли вход на сервер; то есть, может ли эта роль стать начальным авторизованным именем при подключении клиента. Можно считать, что роль с атрибутом LOGIN соответствует пользователю. Роли без этого атрибута бывают полезны для управления доступом в базе данных, но это не пользователи в обычном понимании. По умолчанию подразумевается вариант NOLOGIN
- **REPLICATION**, **NOREPLICATION** Эти предложения определяют, будет ли роль ролью репликации. Чтобы роль могла подключаться к серверу в режиме репликации (в режиме физической или логической репликации) и создавать/удалять слоты репликации, у неё должен быть этот атрибут (либо это должна быть роль суперпользователя). Роль, имеющая атрибут REPLICATION, обладает очень большими привилегиями и поэтому этот атрибут должны иметь только роли, фактически используемые для репликации. По умолчанию подразумевается вариант NOREPLICATION. Создавать роли с атрибутом REPLICATION разрешено только суперпользователям.
- **BYPASSRLS**, **NOBYPASSRLS** Эти предложения определяют, будут ли для роли игнорироваться все политики защиты на уровне строк (RLS). По умолчанию подразумевается вариант **NOBYPASSRLS**. Создавать роли с атрибутом **NOBYPASSRLS** разрешено только **суперпользователям**.
- Заметьте, что **pg_dump** по умолчанию отключает **row_security** (устанавливает значение OFF), чтобы гарантированно было выгружено всё содержимое таблицы. Если пользователь, запускающий pg_dump, не будет иметь необходимых прав, он получит ошибку. Однако суперпользователи и владелец выгружаемой таблицы всегда обходят защиту RLS.

**CONNECTION LIMIT *предел_подключений***Если роли разрешён вход, этот параметр определяет, сколько параллельных подключений может установить роль. Значение -1 (по умолчанию) снимает ограничение. Заметьте, что под это ограничение подпадают только обычные подключения. Ни подготовленные транзакции, ни соединения фоновых рабочих процессов в расчёт не берутся.

[ **ENCRYPTED** ] **PASSWORD** '***пароль***'
PASSWORD NULL Задаёт пароль роли. (Пароль полезен только для ролей с атрибутом LOGIN, но задать его можно и для ролей без такого атрибута.) Если проверка подлинности по паролю не будет использоваться, этот параметр можно опустить. При указании пустого значения будет задан пароль NULL, что не позволит данному пользователю пройти проверку подлинности по паролю. При желании пароль NULL можно установить явно, указав PASSWORD NULL.

- **VALID UNTIL** '***дата_время***'Предложение VALID UNTIL устанавливает дату и время, после которого пароль роли перестаёт действовать. Если это предложение отсутствует, срок действия пароля будет неограниченным.
- **IN ROLE *имя_роли***В предложении IN ROLE перечисляются одна или несколько существующих ролей, в которые будет немедленно включена новая роль. (Заметьте, что добавить новую роль с правами администратора таким образом нельзя; для этого надо отдельно выполнить команду GRANT.)
- **IN GROUP *имя_роли***IN GROUP — устаревшее написание предложения IN ROLE.
- **ROLE *имя_роли***В предложении **ROLE** перечисляются одна или несколько существующих ролей, которые автоматически становятся членами создаваемой роли. (По сути таким образом новая роль становится «группой».)

- **ADMIN *имя_роли*** Предложение ADMIN подобно ROLE, но перечисленные в нём роли включаются в новую роль с атрибутом WITH ADMIN OPTION, что даёт им право включать в эту роль другие роли.
- **USER *имя_роли*** Предложение USER является устаревшим написанием предложения **ROLE**.
- **SYSID *uid*** Предложение SYSID игнорируется, но принимается для обратной совместимости.

Для изменения атрибутов роли применяется **ALTER ROLE**, а для удаления роли — **DROP ROLE**. Все атрибуты, заданные в **CREATE ROLE**, могут быть изменены позднее командами **ALTER ROLE**.

Для добавления и удаления членов ролей, используемых в качестве групп, рекомендуется использовать **GRANT** и **REVOKE**.

# Вопросы?