

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	5
1. МНОГОПОЛЬЗОВАТЕЛЬСКИЕ БАЗЫ ДАННЫХ. ПЕРЕПРОЕКТИРОВАНИЕ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ.....	6
1.1. Причины перепроектирования многопользовательских баз данных .	6
1.2. Инструментарий исследования физической модели данных.....	8
1.3. Инструкции перепроектирования баз данных	10
1.4. Тестовые задания для самопроверки	13
2. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ И ТРАНЗАКЦИИ	17
2.1. Определение параллельной обработки данных	17
2.2. Транзакции и аномалии транзакций.....	19
2.3. Уровни изоляции транзакций и блокировки	22
2.4. Тестовые задания для самопроверки.	25
3. РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ БАЗ ДАННЫХ .	28
3.1. Компоненты резервного копирования базы данных	28
3.2. Структура журнала транзакции и восстановление баз данных. ...	30
3.3. Документация и скрипты резервного копирования.	32
3.4. Вопросы для самостоятельного изучения по итогам лекции.....	36
3.5. Тестовые задания для самопроверки	36
4. РЕПЛИКАЦИЯ В МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗАХ ДАННЫХ	39
4.1. Определение репликации базы данных	39
4.2. Распределенная транзакция и репликация	40
4.3. Типы репликации MS SQL Server 2018	43
4.4. Настройка репликации и шардинга в MongoDB.....	45
4.5. Вопросы для самостоятельного изучения по итогам лекции.....	47
4.6. Тестовые задания для самопроверки	47
5. АВТОМАТИЗАЦИЯ ЗАДАЧ АДМИНИСТРИРОВАНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ	50
Вопросы для самостоятельного изучения по итогам лекции.....	56
6. МОДЕЛИ БЕЗОПАСНОСТИ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ.....	57
6.1. Принципы обеспечения безопасности баз данных. Определения элементов безопасности	57
6.2. Документация и скрипты модели безопасности баз данных	59

6.3. Элементы безопасности и скрипты MongoDB.....	63
6.4. Вопросы для самостоятельного изучения по итогам лекции	64
6.5. Тестовые задания для самопроверки	65
7. ВРЕДОНОСНЫЕ АТАКИ НА МНОГОПОЛЬЗОВАТЕЛЬСКИЕ БАЗЫ ДАННЫХ.....	67
7.1. Вредоносные атаки на серверы баз данных	67
7.2. Управление сетевой безопасностью серверов баз данных	71
7.3. Вопросы для самостоятельного изучения по итогам лекции	72
Ответы на тестовые задания.	73
Список литературы	74
Сведения об авторе	75

ВВЕДЕНИЕ

Современная организация среднего и крупного бизнеса, абсолютно невозможна без интеграции в бизнес-процессы организаций многопользовательских баз данных. Такого рода базы данных обеспечивают непрерывность бизнеса, круглосуточный доступ информации, как сотрудников компаний, так и многочисленных их клиентов. В настоящее время, многопользовательские физические модели данных реализуются как с помощью реляционных СУБД, так и с использованием noSQL СУБД.

Вне зависимости от выбранной технологии реализации, многопользовательские базы данных являются важной ценностью для организаций, которые их создают и эксплуатируют. Вместе с этим, такого рода базы данных создают для этих же организаций серьезные проблемы, связанные с их проектированием, разработкой и эксплуатацией.

Огромное количество пересекающихся друг с другом пользовательских точек зрения (user view) на структуру многопользовательской базы данных создает очевидные проблемы при создании оптимальных схем баз данных.

Изменяющиеся со временем требования к структуре базы данных создают очевидные проблемы планирования и имплементации изменений в уже имеющуюся структуру многопользовательских баз данных.

Само по себе большое количество пользователей таких баз данных создает проблемы, связанные с оптимизацией нагрузки на серверы баз данных. А как быть с безопасностью? Чем больше пользователей имеют доступ к чувствительной информации, тем сложнее мониторить безопасное функционирование серверов многопользовательских баз данных.

Чем больше данных храниться в базе данных, чем больше сервисы компаний “завязаны” на серверах баз данных, тем более критичным может стать выход серверов баз данных или самих файлов базы данных из строя.

Решение перечисленных проблем – это задача администраторов данных или администраторов баз данных. В ходе нашего курса, мы рассмотрим теоретически и практически основные функции, выполняемые специалистами по администрированию баз данных, а также актуальные способы решения проблем функционирования многопользовательских баз данных.

1. МНОГОПОЛЬЗОВАТЕЛЬСКИЕ БАЗЫ ДАННЫХ. ПЕРЕПРОЕКТИРОВАНИЕ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ

1.1. Причины перепроектирования многопользовательских баз данных

Специалисты выделяют две основные причины перепроектирования многопользовательских баз данных.

Первая причина – это проблема комплексности (сложности) проекта многопользовательской базы данных. Весьма непросто выполнить проектирование базы данных без ошибок с первого раза. Это касается даже компактных моделей баз данных для небольших приложений. Что уж говорить о сложных моделях для интернет-бизнеса или для принципиально новых информационных систем. Даже при наличии подробного технического задания, правильной и понятной логической модели, при реализации физической модели в выбранной СУБД с большой вероятностью могут возникнуть разного рода проблемы. Если проект базы данных очень большой, то такой проект может потребовать даже несколько стадий (слоев) проектирования и перепроектирование уже готовых элементов физической модели, в ходе реализации последующих этапов проекта – это стандартная практика. Аналогично проектированию, процесс исправления ошибок, допущенных при моделировании баз данных – это обычная и правильная практика проектирования и эксплуатации многопользовательских баз данных.

Вторая причина – это взаимное влияние информационных систем и организаций друг на друга. Манифесты DevOps, ITIL, CobIT уже давно декларируют взаимное эволюционное изменение IT-инфраструктуры организаций и их организационной архитектуры. Когда внедряется новая информационная система – пользователи подстраиваются под нее. Со временем, сами пользователи (организация) начинают подстраивать информационную систему под свои нужды. Этот процесс эволюционных взаимных изменений становится перманентным. И такими же перманентными становятся изменения в структуре физической модели многопользовательской базы данных.

Приведем *стандартные причины перепроектирования* многопользовательских баз данных:

1. Увеличение компании за счет новых дивизионов.
2. Выход компании на новые рынки.
3. Добавление новых продуктов или услуг.
4. Переход компании на новую форму функционирования.

5. Внедрение новых политик безопасности информации.
6. Внедрение новых политик по принципам хранения информации
7. etc.

Развитие, расширение сферы деятельности компании всегда ведет за собой существенные изменения в ее IT-инфраструктуре и информационных системах. На рис. 1 показаны изменения, произошедшие в последние несколько лет в компании Amazon.

Начав свою операционную деятельность, как компания, предоставляющая услуги при продаже книг (посредник для вендоров, печатающих и продающих книжную продукцию), компания в сравнительно короткий срок диверсифицировала свою деятельность и в настоящий момент времени не только продает и отправляет всевозможную продукцию от тех же книг до автомобилей Феррари, но и предоставляет огромное количество сервисов и услуг, связанных с распространением и продажами медиаконтента.

Сервис AWS (Amazon Web Services) в настоящее время – одна из самых популярных платформ облачных вычислений для коммерческих и государственных сервисов.



Рисунок 1. Диверсификация цифровых сервисов Amazon

Продавая только книги, компании Amazon, в рамках структуры их базы данных достаточно было в качестве первичных ключей использовать названия компаний-вендоров. Очевидно, что после диверсификации продаж, использовать названия компаний стало проблематичным, из-за появления названий-дубликатов. В сложившейся ситуации, чтобы не допустить конфликта экземпляров сущностей в базе данных, возникает необходимость перепроектирования баз данных в сторону изменения первичных ключей

сущностей (вероятно, переход на собственные кодированные идентификаторы вендоров Amazon).

Так что же произойдет, если нам необходимо поменять существующий набор ключей в таблице? Во-первых, добавится список новых ключевых значений, запланированных при перепроектировании. Но это далеко не все. Помимо измененных первичных ключей, должны быть изменены и все внешние ключи, связанные с первичными. Для этого необходимо знать, в каких связях участвовал старый первичный ключ. То же самое касается и созданных в базе данных представлений. Не забываем и про триггеры и хранимые процедуры, которые также могли быть «завязаны» на старом первичном ключе. Все приложения баз данных, которые использовались компанией, тоже могут быть чувствительны к старым значениям первичного ключа и также должны быть пересмотрены при перепроектировании.

И при всем вышеперечисленном, все действия, описанные выше должны быть произведены без ошибок с обеспечением непрерывности бизнеса компании. Можете вы представить на странице веб-приложения Amazon сообщение «Простите, у нас сломалась база данных, - заходите завтра (но это неточно)»?

Это все накладывает серьезные требования и ограничения на каждую процедуру перепроектирования баз данных. Квалификация администратора должна позволять безошибочно и в срок выполнять эту функцию.

С точки зрения администрирования баз данных, процедура перепроектирования относится к *SDLC (systems development life cycle)* и регламентируется тремя основными принципами:

- «семь раз отмерь, один раз отрежь». Тщательная неоднократная проверка решений по перепроектированию, перед их имплементацией;
- предварительное тестирование всех изменений на тестовой базе данных с проходкой основных тестовых дата-кейсов;
- полное резервное копирование изменяемой базы данных (об этом будет подробнее рассказано в материале лекции 3 этого пособия).

1.2. Инструментарий исследования физической модели данных

Подготовка к процедуре перепроектирования базы данных начинается с изучения физической модели базы данных. Для получения компактной и информативной модели используются практики *обратного инжиниринга (reverse engineered, RE)* и составления *графов зависимостей (dependency graph)*.

Обратный инжиниринг – это процесс чтения схемы физической модели базы данных и формирование модели данных в качестве результата. Полученная

модель не является логической, поскольку может содержать элементы, для этой модели неприсущие (физические типы данных с пользовательскими ограничениями; пересечения таблиц, не имеющие ключевых значений и т.д.). Данные модели формируются в специализированном программном обеспечении в автоматическом режиме, после процедуры настройки мастера (ErWin Data Modeler, MySQL Workbench). Образец RE-модели, созданной в программном средстве ErWin Data Modeler в результате чтения реляционной базы данных приложения msuniversity.ru показан на рис. 2.

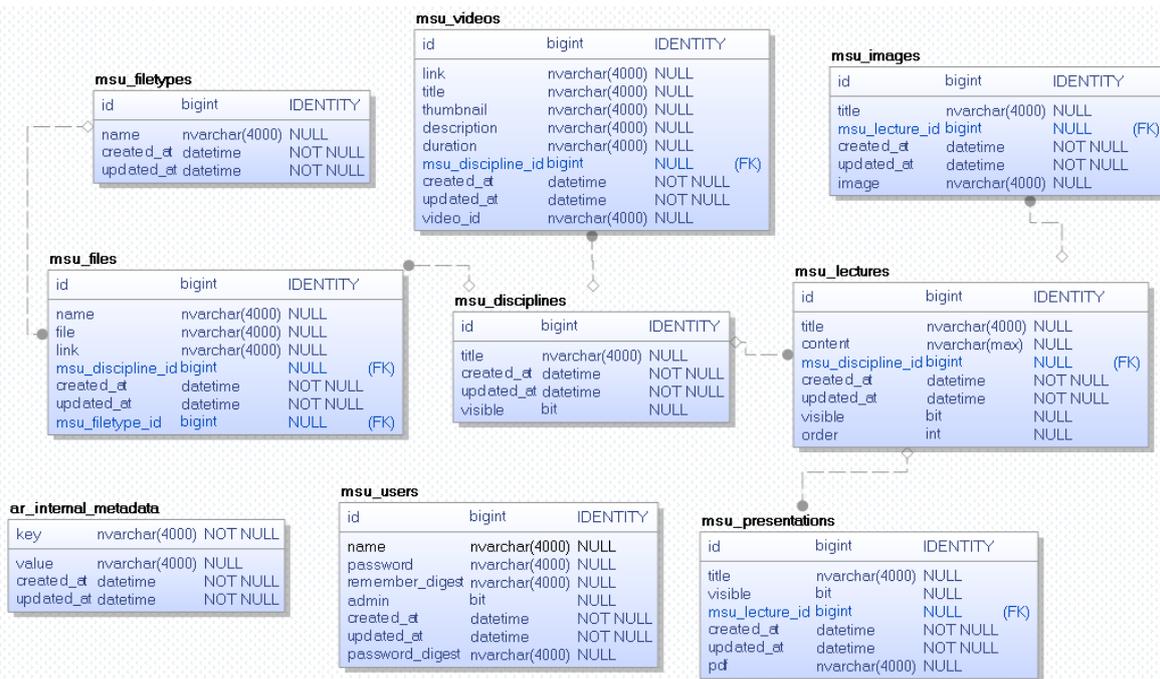


Рисунок 2. RE-модель веб-приложения msuniversity

На RE-модели можно увидеть структуру таблиц (не все из которых соединены связями), описание типов связей и подробное описание столбцов приведенных таблиц.

Однако, стоит заметить, что у инструмента RE-моделирования есть один существенный недостаток, не позволяющий при перепроектировании баз данных руководствоваться только им. В модели не показываются представления, хранимые процедуры, триггеры и приложения баз данных, на которые могут повлиять изменения в ходе перепроектирования базы данных. С целью избавиться от этого недостатка, для проекта перепроектирования также создается граф зависимостей.

Граф зависимостей состоит из узлов, являющихся артефактами баз данных (таблицы, представления, триггеры, хранимые процедуры и т.д.). Узлы связаны друг с другом многочисленными связями. Одна сторона связи показывает ее источник, а другая – адресат. Основной задачей графа зависимости является в компактном виде показать все связи между многочисленными элементами базы

данных. Пример графа зависимостей для фрагмента базы данных аукциона произведений искусства, который используется нами в ходе выполнения практической работы, приведен на рис. 3.

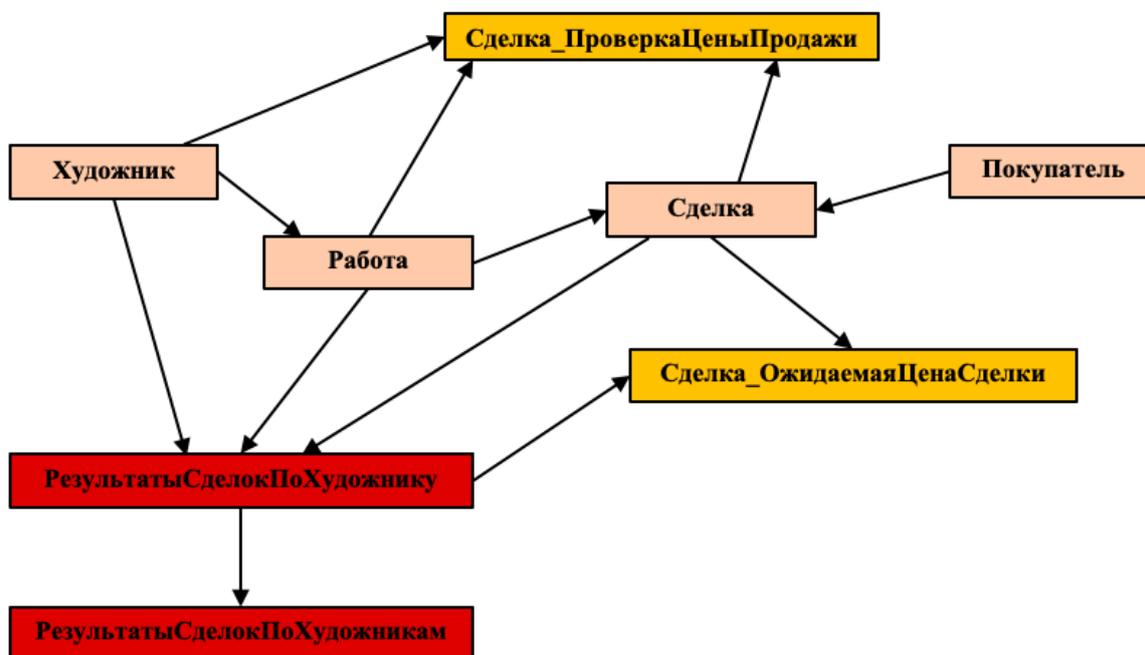


Рисунок 3. График зависимости для базы данных аукциона произведений искусств

На графике «телесным» цветом показаны таблицы базы данных, оранжевым цветом – триггеры и хранимые процедуры, а красным цветом – представления. Очевидно, что при изменении структуры таблицы Работа (изменение свойств первичного ключа, переименование таблицы и т. Д.), это повлияет на связь Работа-Сделка (внешний ключ таблицы Сделка ссылается на первичный ключ таблицы Работа), на работоспособность триггера Сделка_ПроверкаЦеныПродажи и на оба представления РезультатыСделокПоХудожнику и РезультатыСделокПоХудожникам. Исследование графика зависимости дало нам понимание – какие элементы базы данных будут затронуты в ходе перепроектирования, что позволит нам заранее создать скрипт, позволяющий без рисков провести запланированное перепроектирование.

1.3. Инструкции перепроектирования баз данных

Далее рассмотрим основные элементарные процедуры и инструкции языка SQL, которые реализуются в ходе перепроектирования баз данных.

Изменение названия таблиц

Изучение БД через граф зависимостей часто показывает, что у одной

таблицы может быть несколько связанных с ней элементов. Это провоцирует проблему переименования таблицы. Данную проблему принято решать через создание новой таблицы, переноса всех данных старой таблицы и уничтожение старой таблицы, потерявшей актуальность.

Пример данной процедуры, показан в описании рис. 3 выше.

Давайте предположим, что в процессе перепроектирования структуры данных, представленной на рис. 3 таблицу Работа (Work) необходимо переименовать в таблицу Работа_Версия2 (Work_Version2). Это может быть обусловлено, как было описано выше диверсификацией деятельности компании или же изменениями в политике информационного менеджмента или информационной безопасности компании. Стандартный набор действий для этой процедуры, будет следующий:

1. Создаем переменные с внешними и внутренними ключами для новой таблицы Work_Version2.
2. С помощью вложенного запроса или иным доступным способом копируем данные из старой таблицы Work в новую Work_Version2, если необходимо, предварительно их заменив или скорректировав.
3. Заменяем ограничения внешних ключей в других связанных таблицах (в нашем случае, это таблица Сделка (Deal)).
4. Изменяем скрипты запросов для связанных с таблицей представлений.
5. Переписываем заново связанные с таблицей триггеры и хранимые процедуры.
6. Если необходимо, переписываем настройки соединения приложения баз данных и встроенный SQL-код в приложении.

Более подробно принцип изменение названия таблицы в рамках перепроектирования баз данных будет рассмотрен в одной из практических работ по дисциплине в текущем семестре.

Добавление и изменение столбцов существующих таблиц

Добавление новых столбцов в уже существующую таблицу сопряжено с решением одной проблемы. Дело в том, что каждый столбец будет добавлен уже к имеющимся экземплярам данных существующей таблицы, поэтому проектировщик баз данных должен задать себе вопрос – каким образом (какими данными) будет при добавлении заполняться новый столбец. Существует два варианта решения этой проблемы.

Вариант 1. Создание столбца со свойством NULL. В этом случае, все значения уже существующих экземпляров данных таблицы в добавляемом столбце примут неопределенное значение NULL. Типовой скрипт этой

процедуры выглядит следующим образом:

```
ALTER TABLE /Название таблицы/ ADD /Название столбца/ /Тип данных столбца/ NULL;
```

Важно отметить, что указание типа данных при добавлении столбца обязательно.

Вариант 2. Создание столбца со значением по умолчанию (DEFAULT). В этом случае, все значения уже существующих экземпляров данных таблицы в добавляемом столбце примут значение, которое будет указано проектировщиком в скрипте. Типовой скрипт этой процедуры выглядит следующим образом:

```
ALTER TABLE /Название таблицы/ ADD /Название столбца/ /Тип данных столбца/ NULL DEFAULT  
/значение по умолчанию/;
```

Значение по умолчанию должно удовлетворять требованиям указанного для столбца типа данных и пользовательских ограничений.

Удаление столбца выполняется следующей типовой инструкцией:

```
ALTER TABLE /Название таблицы/ DROP COLUMN /Название столбца/;
```

Отдельно стоит рассмотреть случай удаления столбца первичного ключа таблицы. Для осуществления этого необходимо совершить следующую последовательность действий:

1. Удалить все ограничения внешних ключей, ссылающихся на удаляемый первичный ключ из связанных таблиц (если таковые есть).
2. Удалить ограничение первичного ключа.
3. Создать новое ограничение первичного ключа.
4. Создать новые ограничения внешних ключей, связав их с множеством нового первичного ключа.
5. Удалить столбец, бывший первичным ключом ранее.

Изменение минимальных кардинальностей связей

Минимальные кардинальности в бинарной связи – это значения zero и one на сторонах связи между родителем и потомком. При перепроектировании соответственно необходимо изменять zero на one или наоборот на стороне родителя или на стороне потомка. В зависимости от стороны практикуется два принципиально разных подхода к перепроектированию.

Изменение минимальной кардинальности *на стороне родителя* происходит за счет изменения свойства NULL для внешнего ключа в выбранной связи. Свойство внешнего ключа NULL говорит о минимальной кардинальности zero, а свойство NOT NULL – о минимальной кардинальности one.

Для примера предположим, что имеется бинарная связь между таблицами EMPLOYEE (сотрудники) и DEPARTMENT (подразделение). Для этой бинарной

связи мы хотим изменить минимальную кардинальность с zero на one. Порядок инструкций в скрипте, будет следующий:

```
ALTER TABLE EMPLOYEE DROP CONSTRAINT DepartmentFK;
```

```
ALTER TABLE EMPLOYEE ALTER COLUMN DepartmentNumber Int NOT NULL;
```

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DepartmentFK FOREIGN KEY REFERENCES DEPARTMENT (DepartmentNumber);
```

Отметим, что при изменении свойства NULL внешнего ключа, как и в случае переименования ключа, необходимо сперва удалить текущее ограничение внешнего ключа, затем изменить свойство столбца на NOT NULL и после этого пересоздать ограничение первичного ключа.

Изменение минимальной кардинальности на стороне потомка (речь идет только об изменении с zero на one), необходимо написать служебные триггеры, которые будут срабатывать на вставку (INSERT) в таблицу-родитель и на изменения (UPDATE) и удаления (DELETE) в таблице-потомке. Обратное изменение минимальной кардинальности потребует удаления созданных триггеров.

1.4. Тестовые задания для самопроверки

1. Набор специальным способом оформленных инструкций, который выполняется каждый раз при возникновении какого-то определённого действия в таблице называется:

- А) хранимой процедурой
- Б) триггером
- В) системной функцией
- Г) событием

2. Является ли понятие «хранилище данных» и «база данных» синонимами?

- А) нет
- Б) да

3. Искусственно сгенерированное значение поля первичного ключа называется:

- А) суррогатным ключом
- Б) вторичным ключом
- В) внешним ключом

Г) индексом

4. Столбец (группа столбцов) таблицы потомка, массив значений которых совпадает с массивом значений первичного ключа родителя называется:

А) суррогатным ключом

Б) индексом

В) внешним ключом

Г) ссылкой

5. Одним из инструментов изучения физической модели данных при перепроектировании является:

А) граф зависимостей

Б) сетевой граф

В) граф узлов и элементов БД

6. В чем заключается смысл практики «прямого инжиниринга базы данных»?

А) переход из логической модели данных в физическую

Б) переход из концептуальной модели данных в логическую

В) переход из объектно-ориентированной модели хранения данных в реляционную

Г) ни один из перечисленных вариантов

7. В чем заключается смысл практики «обратного инжиниринга базы данных»?

А) переход из логической модели данных в физическую

Б) выгрузка физической модели данных в виде схемы

В) переход из объектно-ориентированной модели хранения данных в реляционную

Г) ни один из перечисленных вариантов

8. Какой из перечисленных артефактов не является видом узла графа зависимостей?

А) таблица

Б) представление

В) триггер

Г) хранимая процедура

Д) все перечисленные могут быть узлами

9. Ограничение первичного ключа создается инструкцией...

А) CREATE/ALTER TABLE... CONSTRAINT...REFERENCES...

Б) CREATE/ALTER TABLE... CONSTRAINT...

В) CREATE PRIMARY KEY

Г) CREATE FOREIGN KEY

10. Ограничение внешнего ключа создается инструкцией...

А) CREATE/ALTER TABLE... CONSTRAINT...REFERENCES...

Б) CREATE/ALTER TABLE... CONSTRAINT...

В) CREATE PRIMARY KEY

Г) CREATE FOREIGN KEY

11. Какой элемент данных в ходе процедуры переименования таблицы должен быть переписан?

А) представление

Б) политика

В) триггер

Г) агент

12. Какая инструкция изменяет свойства столбца таблицы?

А) ALTER COLUMN

Б) ALTER TABLE.... COLUMN

В) ALTER TABLE... ALTER COLUMN

13. Какая инструкция добавляет столбец в таблицу?

А) ADD COLUMN

Б) ALTER TABLE.... ADD COLUMN

В) ALTER TABLE... ADD

14. Какая инструкция удаляет ограничение (IC)?

А) DELETE CONSTRAINT

Б) ALTER TABLE... DELETE CONSTRAINT

В) ALTER TABLE... DROP CONSTRAINT

Г) DROP CONSTRAINT

15. Какое из действий применяется в случае необходимости изменения минимальной кардинальности связей на стороне «таблицы-родителя»?

- А) изменение свойств первичного ключа
- Б) изменение типа данных внешнего ключа
- В) изменение свойства NULL/NOT NULL внешнего ключа
- Г) создание служебного триггера

16. Какое из действий применяется в случае необходимости изменения минимальной кардинальности связей на стороне «таблицы-потомка»?

- А) изменение свойств первичного ключа
- Б) изменение типа данных внешнего ключа
- В) изменение свойства NULL/NOT NULL внешнего ключа
- Г) создание служебного триггера

2. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ И ТРАНЗАКЦИИ

2.1. Определение параллельной обработки данных

Одной из ключевых особенностей многопользовательской базы данных является обеспечение возможности совместной работы нескольких пользователей (зачастую их количество может исчисляться тысячами или даже сотнями тысяч пользователей) с одной и той же (зачастую весьма локализованной и небольшой) совокупностью данных.

Инструментами, обеспечивающими возможности одновременной работы с данными для большого количества пользователей, обладают практически все современные реляционные СУБД, рис. 4.

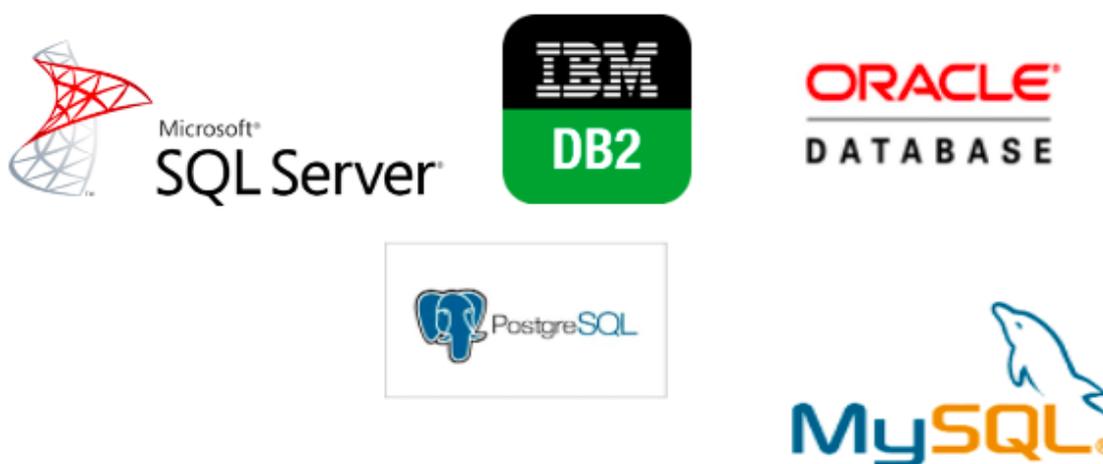


Рисунок 4. Актуальные реляционные СУБД с поддержкой большого количества пользователей

В данной лекции будет рассмотрена реализация параллельной обработки данных и инструмент под названием транзакция.

Управление параллельной обработкой данных – меры и мероприятия, предпринимаемые с целью ограничить влияние действий, осуществляемых пользователями во время сеанса работы с БД друг на друга. Как становится понятно из контекста – пользователи, в ходе работы с данными, если этот процесс не контролируется СУБД могут создавать друг другу существенные помехи. Первая проблема, которая может возникнуть в структуре базы данных в любой момент – это потеря ценных данных из-за возникновения в ходе обработки данных нештатной ситуации. Наглядно такого рода проблема показана на рис. 5.

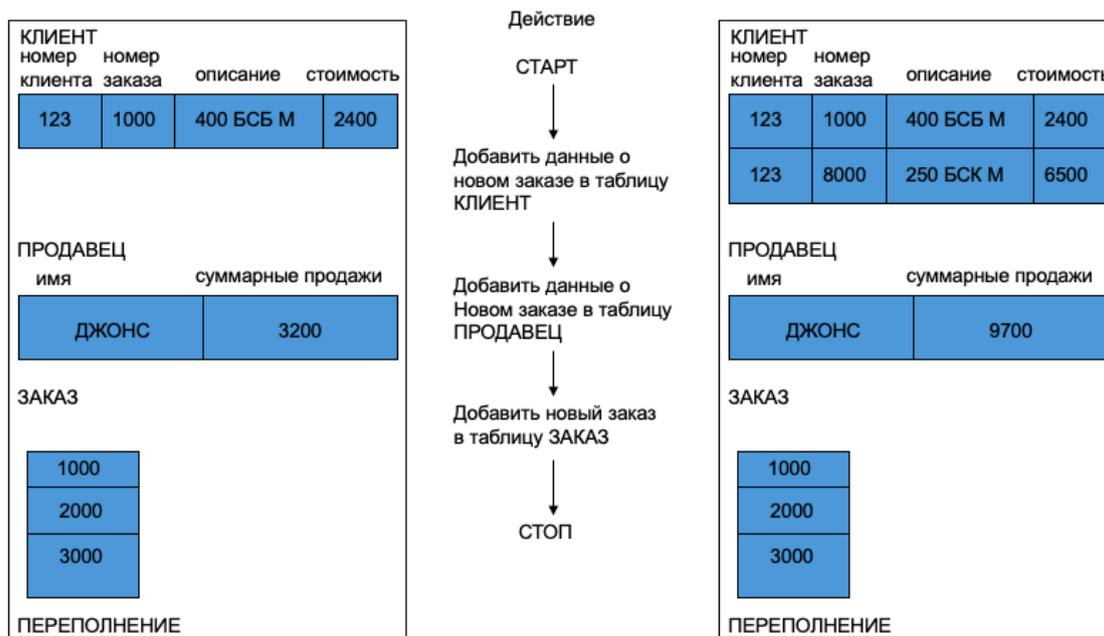


Рисунок 5. Визуализация проблемы обработки данных

В некоторой базе данных магазина товаров для спорта есть три таблицы – КЛИЕНТ, ПРОДАВЕЦ и ЗАКАЗ. При формировании заказа, продавец через приложение последовательно вносит следующие изменения в базу данных: номер клиента, номер заказа, описание заказа и суммарная стоимость всех элементов заказа вносится в таблицу КЛИЕНТ. Стоимость последнего заказа в виде накапливаемого итога вносится в суммарные продажи в таблицу ПРОДАВЕЦ. Наконец номер заказа записывается в таблицу ЗАКАЗ и товары отправляются (передаются) клиенту. Предположим, следующую внештатную ситуацию – данные о заказе 8000 для клиента 123 были внесены в таблицы КЛИЕНТ и ПРОДАВЕЦ, а при внесении данных в таблицу ЗАКАЗ произошло переполнение физического пространства, выделенного под базу данных. Таким образом возникает ситуация, когда данные в таблицах КЛИЕНТ и ПРОДАВЕЦ останутся в измененном виде, тогда как информация для таблицы ЗАКАЗ была потеряна. Это зачастую может привести к неприятному итогу – товары со склада списаны, продавец получает премию, а заказ клиенту не отправляется.

Суть данной проблемы в том, что в приведенном примере одна функция магазина реализована последовательной вставкой нескольких экземпляров данных в разные таблицы. Очевидно, инструкции INSERT (а их будет реализовано 3) выполняются независимо друг от друга, что и провоцирует риск возникновения сложностей в процессе их обработки. Логика решения данной проблемы в следующем – если все участвующие инструкции объединить в один атомарный конструкт с правилами исполнения, то потери данных в любом случае можно избежать. Такие атомарные конструкты называются транзакции.

2.2. Транзакции и аномалии транзакций

AT (атомарной транзакцией) или *LUW* (logical units of work) считаются серии действий, предпринимаемых с базой данных, которые выполняются успешно все целиком, или не выполняются совсем (в случае неуспеха).

В случае применения атомарной транзакции, логика реализации примера на рис. 5 будет выглядеть следующим образом (рис. 6).



Рисунок 6. Логика применения транзакции при обработке данных

В языке SQL транзакция в базовом виде вводится следующим образом:

```
BEGIN /Название транзакции/  
  
/Последовательность инструкций, входящих в транзакцию, например/  
  
Изменить данные Клиент  
Изменить данные Продавец  
Вставить данные в Заказ  
  
IF /Все и успешно/ THEN  
  
COMMIT /Название транзакции/  
  
ELSE  
  
ROLLBACK /Название транзакции/  
  
END IF
```

Инструкция COMMIT сохраняет все изменения, произошедшие в базе данных в ходе реализации транзакции, инструкция ROLLBACK наоборот,

отказывает базу данных в состояние “до транзакции”. Дополнительные свойства транзакций будут описаны позже.

Применение транзакций при выполнении комплексных инструкций в СУБД действительно решает базовые проблемы многопользовательских баз данных. Вместе с этим, в некоторых случаях *параллельное применение транзакций* может стать причиной возникновения одной из пяти аномалий.

1. *Потерянное обновление (lost update)*. Аномалия, проявляющаяся, когда несколько разных транзакций изменяют один и тот же массив данных. При фиксации результата получается, что одна транзакция изменила и переписала данные, внесенные другой транзакцией. Рассмотрим причину и логику появления этой аномалии более подробно.

Для начала посмотрим, как СУБД обрабатывает параллельные транзакции в обычном режиме. Предположим, что два пользователя А и Б параллельно запустили две транзакции. Цель транзакции А – изменить и записать элемент 100 (значение в одной из строк таблицы базы данных), цель транзакции Б – изменить и записать элемент 200 (значение в другой строке, отличной от элемента 100). В это случае транзакции пройдут в штатном режиме, не создавая проблем с феноменами. Последовательность обработки транзакций на стороне СУБД показана на рис. 7.

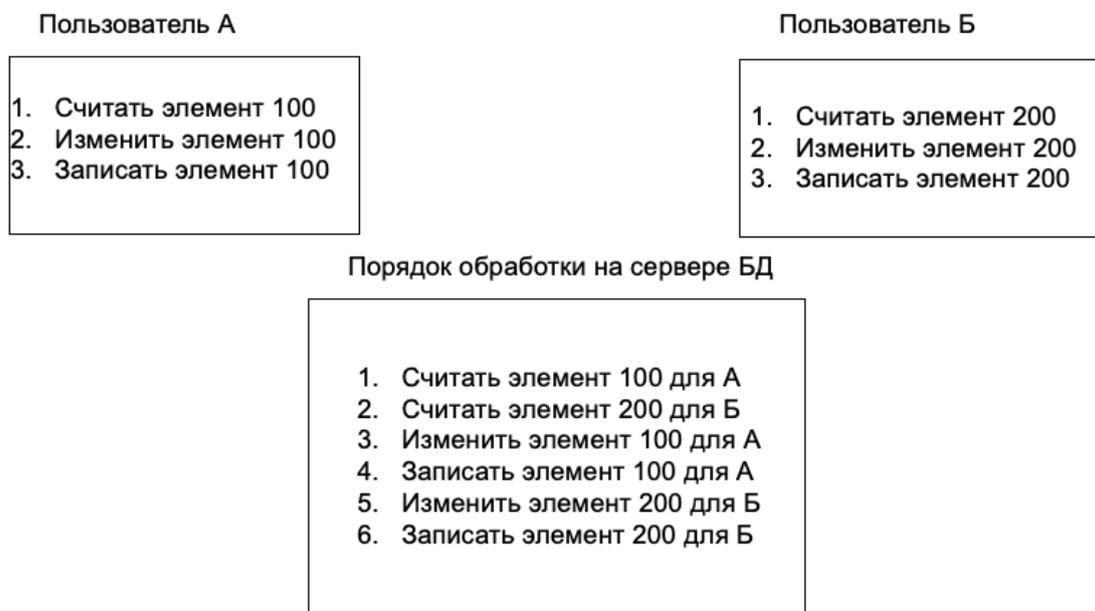


Рисунок 7. Порядок обработки параллельных транзакций СУБД

Пользователи А и Б получают в результате запроса текущее значение элемента 100 и 200 соответственно. Получивший приоритет пользователь А (например, обратился к СУБД чуть раньше) получает возможность с помощью транзакции изменить и записать изменения в элементе 100. После этого возможность изменить и записать изменения в элементе 200 получает

пользователь Б. После этих инструкций БД остается в согласованном состоянии, поскольку транзакции в этом случае друг другу «не мешали».

Аномалия потерянного обновления возникнет, когда параллельные транзакции попытаются изменить и записать изменения одного и того же элемента базы данных. Пример аномалии показан на рис. 8.

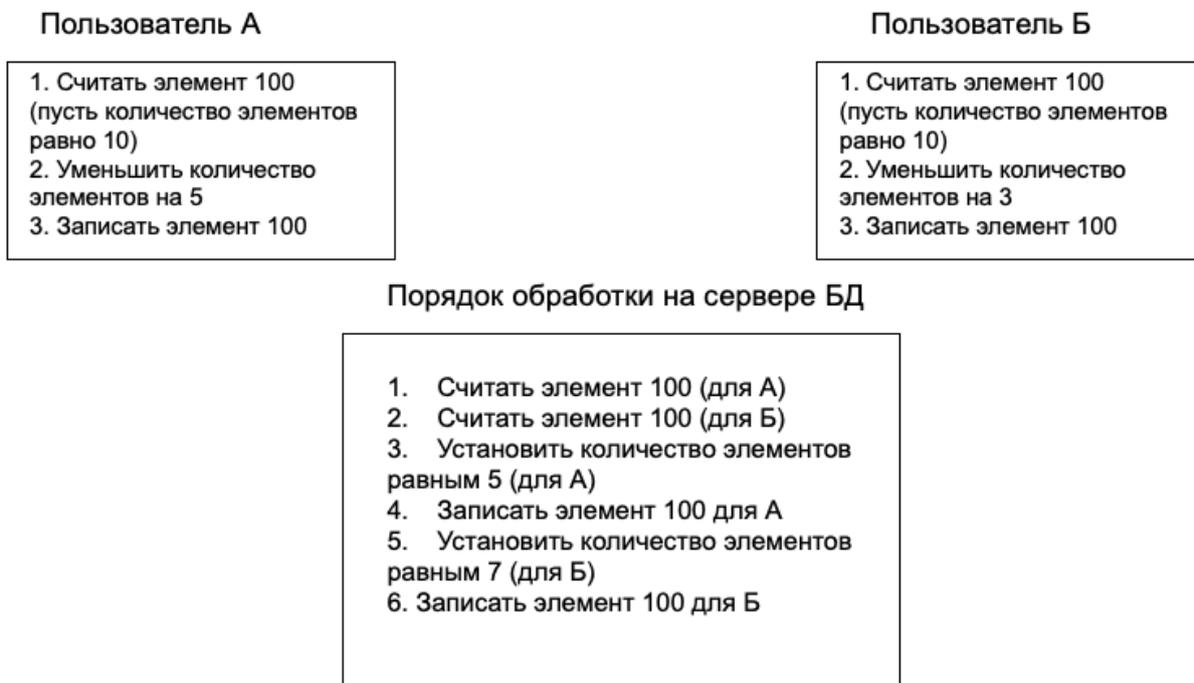


Рисунок 8. Аномалия потерянного обновления при параллельных транзакциях.

Отметим, что последовательность действий идентична тем, которые показаны на рис. 7 и для описанного случая считаются эталонными. Но, в данном случае, будет потеряна транзакция, уменьшающая количество элементов на 5 ($10 - 5 = 5$), поскольку сразу после нее будет выполнена транзакция, уменьшающая начальное количество элементов на 3 ($10 - 3 = 7$). Результат второй транзакции и станет последним записанным значением.

2. *Грязное чтение (dirty read)*. Аномалия возникает тогда, когда одна параллельная транзакция читает данные, измененные транзакцией, которая все еще обрабатывается в СУБД (этот процесс может теоретически занимать от нескольких секунд, до нескольких часов). Если транзакция, которая выполняется в СУБД сейчас будет завершения с итогом ROLLBACK (откат на стартовое состояние из-за ошибки в инструкциях), то данных, полученных параллельной транзакцией уже в системе физически не будет.

3. *Неповторяющееся чтение (non-repeatable read)*. Аномалия проявляется тогда, когда при повторном чтении одного и того же массива данных (выполнение инструкции SELECT), транзакция получает нетождественные (разные) массивы данных. Происходит это из-за того, что в промежутке между

первичным и повторным чтением данных другая транзакция уже успела их изменить. В итоге один и тот же запрос выдает различные результаты.

4. *Фантомное чтение (phantom read)*. Похоже на случай неповторяющегося чтения, описанный выше, но между первичным и повторным чтением другая транзакция добавляет новые строки в массив данных и фиксирует изменения. В итоге, результатом выполнения запроса является другое множество строк.

5. *Сериализация (serialization anomaly)*. Результат успешно выполненных параллельных транзакций не совпадает с результатом последовательного выполнения этих же транзакций.

2.3. Уровни изоляции транзакций и блокировки

Решением всех перечисленных аномалий является выбор настройки параллельной транзакции (*изоляция*). Общий стандарт SQL подразумевает четыре возможных уровня изоляции: Read Uncommitted (незавершенное чтение), Read Committed (завершенное чтение), Repeatable Read (повторное чтение) и Serializable (сериализация). Реакция уровня изоляции на возможные аномалии показана в табл. 1.

Таблица 1. Уровни изоляции транзакций

	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Потерянное обновление	Невозможно	Невозможно	Невозможно	Невозможно
Грязное чтение	Возможно	Невозможно	Невозможно	Невозможно
Невоспроизвод. чтение	Возможно	Возможно	Невозможно	Невозможно
Фантомное чтение	Возможно	Возможно	Возможно	Невозможно
Аномалия сериализации	Возможно	Возможно	Возможно	Невозможно

В коде инструкции SQL уровень изоляции транзакции устанавливается командой `SET TRANSACTION ISOLATION LEVEL [выбор из... READ UNCOMMITTED/READ COMMITTED/REPEATABLE READ/SERIALIZABLE]`. После выбора уровня изоляции транзакции, дальнейшие действия по устранению аномалии СУБД выполнит в автоматическом режиме.

Подробнее примеры аномалий и изоляции транзакций будут разобраны на практических занятиях курса.

На уровне изоляции READ COMMITTED возможна более гибкая настройка изоляции элементов базы данных в ходе выполнения параллельных транзакции.

Сделать это проектировщик запросов может с помощью инструмента *Блокировка*. С помощью блокировок проектировщик может заблокировать на время транзакции как таблицы базы данных, так и отдельные строки таблиц. Базовый принцип работы инструмента блокировка показан на рис. 9. Для примера выбрана ситуация, аналогичная рис. 8.

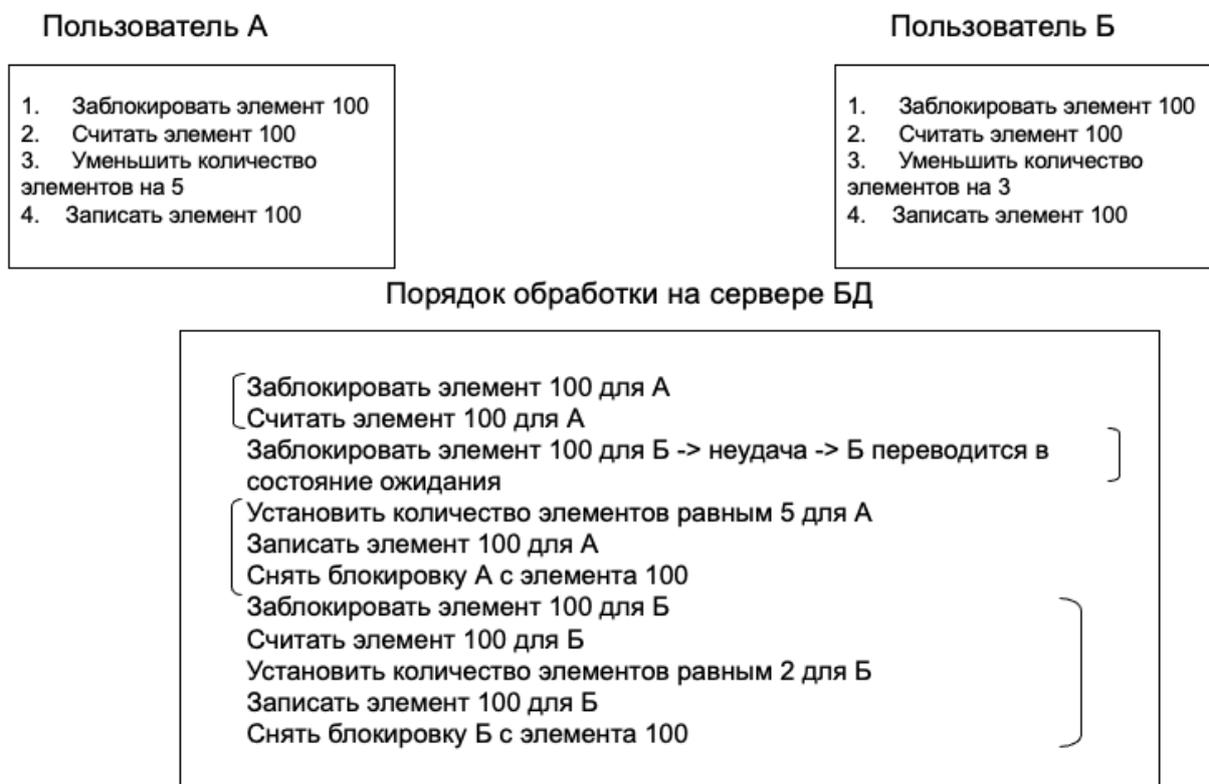


Рисунок 9. Блокировка в процессе параллельных транзакций.

В порядке обработки на сервере «рамками» показаны изменения в последовательности действий СУБД, вызванные введенной проектировщиком блокировкой. В данном случае блокировке подлежит лишь тот элемент, что будет изменен в ходе исполнения параллельных транзакций.

В зависимости от используемой СУБД существует значительное количество разных способов наложения блокировки. Рассмотрим наиболее простой способ, с использованием инструкции LOCK. Блокировки делятся на оптимистические и пессимистические.

Принцип *оптимистической блокировки*: все пройдет хорошо, процесс будет успешный, конфликта транзакции не случится. Если случится конфликт – переделаем транзакцию столько раз, сколько нужно для отсутствия конфликта. Блокировка будет установлена только на время исполнения инструкций, изменяющих данные. Приведем пример скрипта, реализующего оптимистическую блокировку.

```

@newquantity int
SELECT product.name, product.quantity
FROM product
WHERE product.name = 'Mountain Bike';
SET newquantity = product.quantity - 20;
/*полагаем, что все пройдет хорошо*/
LOCK product;
UPDATE product
SET product.quantity = newquantity
WHERE product.name = 'Mountain Bike';
UNLOCK product;

```

Инструкции блокировки LOCK и UNLOCK накладываются на таблицу PRODUCT только на время выполнения инструкции UPDATE.

Принцип *пессимистической блокировки*: все пройдет плохо, обязательно может случиться конфликт. Элемент, подлежащий изменению, блокируется на все время прохождения транзакции до ее успешного завершения. Блокируются все инструкции, входящие в транзакцию. Далее приведем пример скрипта, реализующего пессимистическую блокировку.

```

@newquantity int
/*полагаем, что со скриптом будут проблемы*/
LOCK product;
SELECT product.name, product.quantity
FROM product
WHERE product.name = 'Mountain Bike';
SET newquantity = product.quantity - 20;
UPDATE product
SET product.quantity = newquantity
WHERE product.name = 'Mountain Bike';
UNLOCK product;

```

Инструкции блокировки LOCK и UNLOCK накладываются на таблицу PRODUCT на все время исполнения транзакции, начиная с инструкции на выборку SELECT.

Блокировки – мощный инструмент оптимизации выполнения параллельных транзакций в руках проектировщика запросов. Но при его использовании нельзя

забывать об осторожности. В некоторых случаях неправильно настроенная блокировка параллельных транзакций может привести к возникновению аномалии под названием *смертельный замок (death lock)*. Логику аномалии можно увидеть на рис. 10.

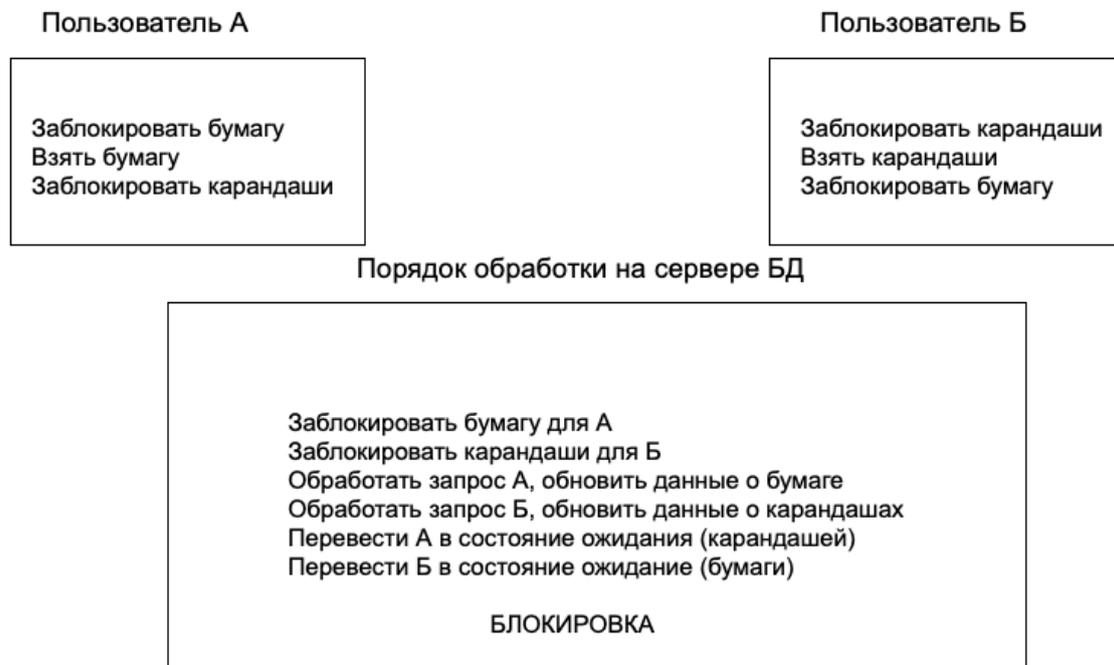


Рисунок 10. Аномалия некорректной блокировки параллельных транзакций.

Пользователь А и Пользователь Б изменяют состояние двух объектов, параллельно, в ходе транзакции, блокируя их друг от друга. Это бесконечное состояние ожидания двух параллельных транзакций, которое может быть разрешено средствами СУБД. Также, при составлении скриптов запросов, современные средства СУБД могут предупредить от опасности возникновения этой аномалии блокировки.

2.4. Тестовые задания для самопроверки.

1. Управление параллельной обработкой данных выполняется на стороне:

- А) администратора данных
- Б) администратора СУБД
- В) СУБД
- Г) сторонней утилитой

2. LUW (Logical Unit of Work), это:

- А) SQL инструкция
- Б) хранимая процедура
- В) транзакция
- Г) триггер

Д) ни одно из перечисленного

3. Инструкция COMMIT транзакции...

- А) сохраняет результат транзакции
- Б) запускает транзакцию на исполнение
- В) задает условия выполнения транзакции
- Г) откатывает транзакцию в начало

4. Инструкция ROLLBACK транзакции...

- А) сохраняет результат транзакции
- Б) запускает транзакцию на исполнение
- В) задает условия выполнения транзакции
- Г) откатывает транзакцию в начало

5. На какой элемент базы данных влияет «изоляция» транзакции?

- А) на таблицу
- Б) на строки
- В) на экземпляр базы данных
- Г) на табличное пространство базы данных

6. «Смертельный замок» разрешается на уровне...

- А) администратора базы данных
- Б) СУБД
- В) пользователя
- Г) может быть разрешен на любом из перечисленных уровней

7. Какая инструкция не относится к инструкциям управления транзакциями?

- А) BEGIN TRANSACTION
- Б) REVOKE
- В) SAVEPOINT
- Г) ROLLBACK

8. Какой из перечисленных видов блокировок устанавливается СУБД по умолчанию?

- А) мягкая блокировка
- Б) исключительная блокировка
- В) неявная блокировка

9. Максимально строгую блокировку элемента БД при выполнении транзакции обеспечивает:

- А) сериализуемость
- Б) завершенное чтение
- В) воспроизводимое чтение
- Г) незавершенное чтение

10. Серии действий, выполняющихся на экземпляре БД полностью или не выполняющихся совсем называются:

- А) LUW
- Б) DML
- В) SQL
- Г) скрипт

3. РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ БАЗ ДАННЫХ

3.1. Компоненты резервного копирования базы данных

Следующей ключевой функцией администрирования многопользовательских баз данных является проведение мероприятий по *резервному копированию (backup)* и *восстановлению (restore)* баз данных.

Данные функции неразрывно связаны с обеспечением безопасности баз данных и сопутствуют жизненному циклу любой базы данных. В контексте этой лекции процедура резервного копирования и восстановления баз данных будет рассмотрена теоретически, а также в контексте инструкций языков tSQL, PostgreSQL, а также для СУБД mongoDB.

Резервное копирование базы данных – это процедура сохранения копии данных на носителе, не являющимся основным местом их хранения. Цель данной процедуры – получить возможность восстановления данных в случае их потери на основном сервере. Технически, для правильного проведения процедур резервного копирования, помимо скриптов, речь о которых пойдет ниже, необходимо иметь фундаментальное представление о структуре файлов баз данных, видах резервных копий и сопровождающей процесс документации. Структура необходимых знаний схематично показана на рис. 11.

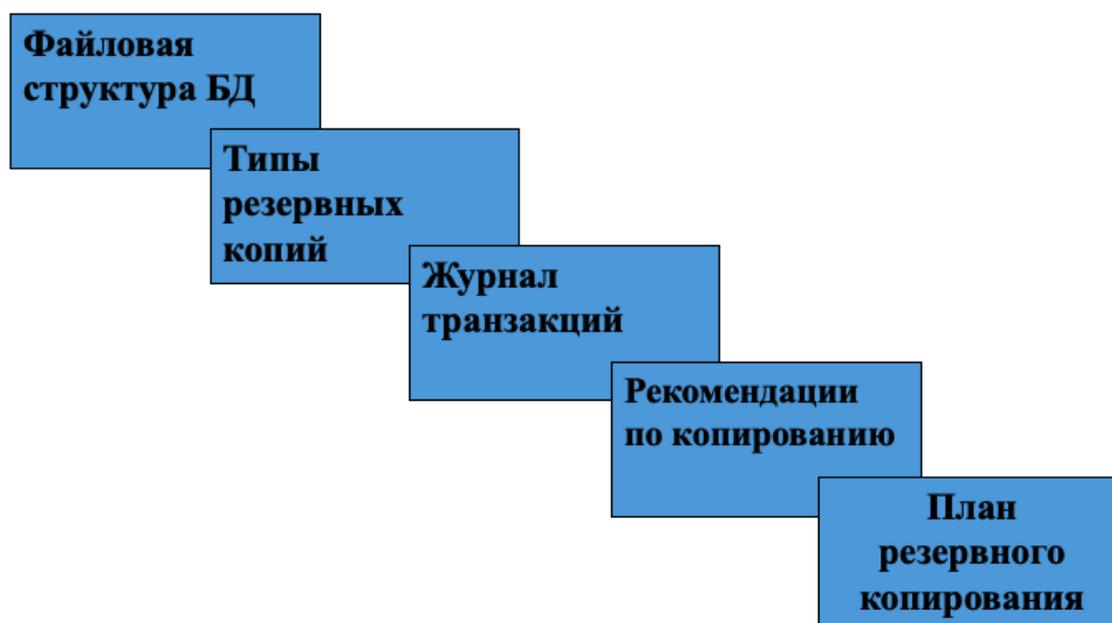


Рисунок 11. Компоненты знаний процедуры резервного копирования

Сразу отметим, что файловая структура баз данных и возможные типы резервных копий могут существенно различаться, в зависимости от выбранной СУБД. Принцип функционирования файловой структуры MS SQL Server 2018

будет подробно разобран на практических занятиях, сопровождающих эту лекцию. Ниже будут приведены типы резервного копирования для СУБД MS SQL Server 2018. Типы резервных копий СУБД PostgreSQL и MongoDB будут кратко рассмотрены в части лекции, посвященной скриптам резервного копирования.

Для СУБД MS SQL Server 2018 выделяют 3 основных типа резервного копирования баз данных: полная резервная копия базы данных, разностная резервная копия базы данных и резервная копия журналов транзакций. Также, отдельно отметим возможность частичного резервного копирования файлов баз данных. На рис. 12 показано соотношение типов копий по объему данных.

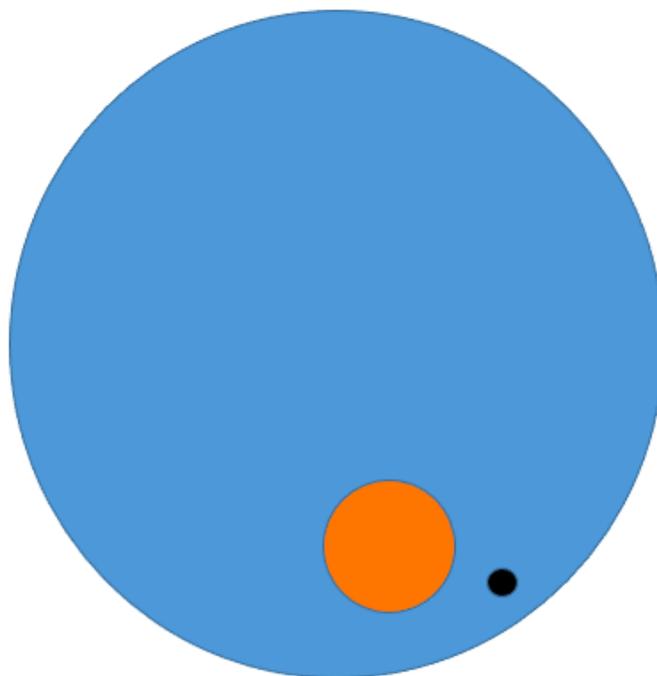


Рисунок 12. Соотношение типов резервных копий по объему данных

На рисунке самый большой круг (синий), это *полная резервная копия базы данных*. Она отображает состояние всей базы данных на момент копирования. Как правило, такая копия весьма внушительна по размеру и процесс ее создания занимает значительное время, создавая сильную нагрузку на сервер баз данных. Копия содержит все данные заданной базы данных и журналы транзакций, которые будут сопровождать процесс восстановления данных в случае необходимости.

Второй по размеру круг на рис. 12 (коричневый) – это *разностная (дифференцированная) резервная копия базы данных*. Копия этого типа зависит от предварительно созданной полной резервной копии базы данных. Разностная копия содержит в себе только те данные, которые были изменены в базе с момента последнего создания полной резервной копии базы данных. Дело в том, что, как было сказано выше, процесс создания полной резервной копии зачастую

бывает весьма объемным, длительным по времени и может сильно нагрузить сервер. Отсюда невозможно регулярно (например, каждый день) создавать полные резервные копии и создаются они, как правило, с промежутком в неделю. Чтобы не потерять массивы данных, которые добавляются или изменяются в базе данных в промежутке между снятием полных копий, применяется разностное копирование.

Наконец последний, самый маленький круг на рис. 12 (черный), это резервная копия журналов транзакций. В эту самую маленькую по объему резервную копию базы данных входят все записи журнала транзакций, которые не вошли во все предыдущие резервные копии журналов.

3.2. Структура журнала транзакции и восстановление баз данных.

Журнал транзакций – один из ключевых элементов СУБД. В этот файл записываются все транзакции, которые были осуществлены в соответствующей СУБД. Именно благодаря этому файлу, который ведется СУБД в автоматическом режиме, сама СУБД сможет определить, когда, где и какие изменения были осуществлены пользователями в базе данных. Это является критически важным при многих системных процедурах СУБД, не исключая процедуры восстановления баз данных. Упрощенная типовая структура журнала транзакций показана на рис. 13.

Идентификатор транзакции	Указатель назад	Указатель вперед	Время	Тип операции	Объект	Исходный образ	Конечный образ
--------------------------	-----------------	------------------	-------	--------------	--------	----------------	----------------

Рисунок 13. Структура журнала транзакций

Идентификатор транзакции – уникальный номер, выдаваемый каждой транзакции СУБД, позволяющий однозначно отличить одну транзакцию от всех остальных.

Указатель назад – номер этапа транзакции, на котором осуществляется указанное в данной записи действие. Указатель назад в положении 0 говорит о начале транзакции на этой этапе.

Указатель вперед – номер следующего этапа транзакции. Указатель вперед в положении 0 говорит об окончании транзакции на этом этапе.

Время – системное время в тот момент, когда было осуществлено действие в рамках транзакции.

Тип операции – одна из инструкций, входящих в состав транзакции (BEGIN, INSERT, UPDATE и так далее).

Объект – объект базы данных, в котором были внесены изменения транзакцией.

Исходный образ – значение до изменения транзакцией.

Конечный образ – значение после изменения транзакцией.

В табл. 2 показан образец журнала для группы произвольных транзакций.

Таблица 2. Фрагмент журнала транзакций

OT1	0	2	12:00	BEGIN			
OT1	1	4	12:01	UPDATE	product	/C3/	/H3/
OT2	0	7	12:05	BEGIN			
OT1	2	5	12:10	UPDATE	salesman	/C3/	/H3/
OT1	4	6	12:10	INSERT	order		/3/
OT1	5	0	12:12	COMMIT			
OT2	3	0	12:13	COMMIT			

Логика процесса восстановления данных для случая, описанного на рис. 5

Лекции 2 показана на рис. 14.

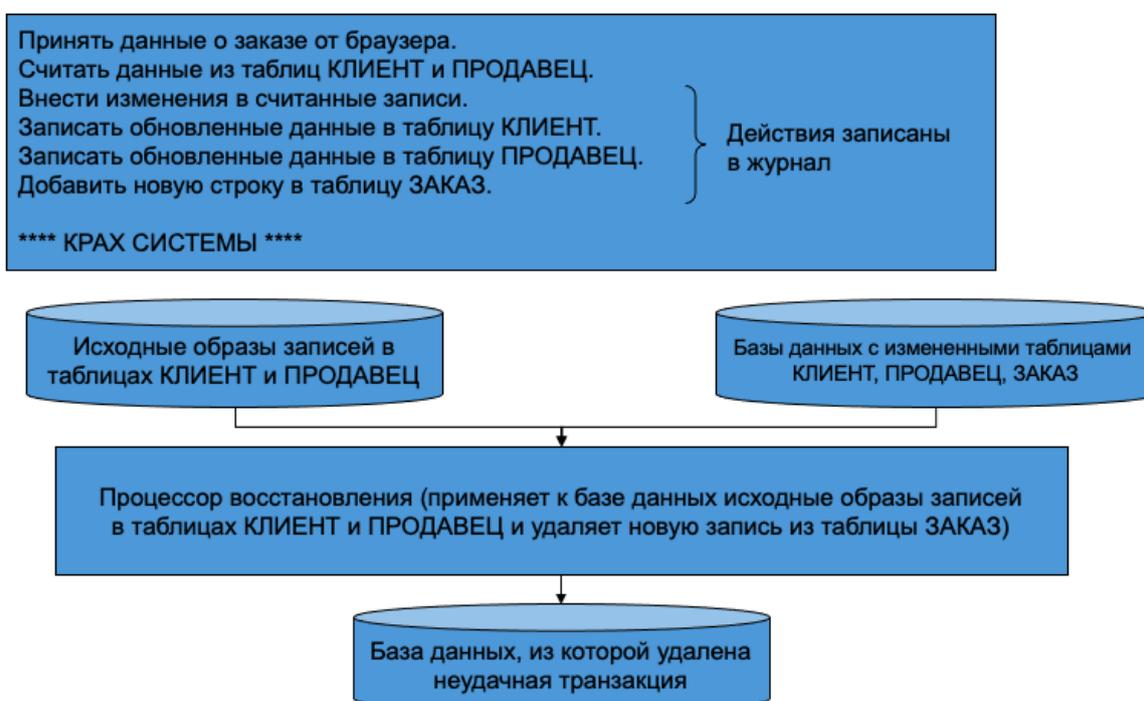


Рисунок 14. Восстановление базы данных с помощью журнала транзакций

Продавец магазина последовательно, с помощью транзакции заносит данные в таблицы КЛИЕНТ и ПРОДАВЕЦ. При попытке добавить новую строку в таблицу ЗАКАЗ происходит крах системы (перебой питания, переполнение физической или оперативной памяти и так далее). После восстановления системы, процессор восстановления с помощью записей журнала транзакций откатывает изменения, внесенные некорректно законченной транзакцией, возвращая таблицы КЛИЕНТ и ПРОДАВЕЦ в состояние «до транзакции». Транзакцию можно повторить снова, уже с положительным результатом.

Ниже приведены основные правила взаимодействия администратора баз данных с журналами транзакций.

1. Правильным решением всегда будет размещать копию журнала транзакций на твердотельный, ленточный или иной быстрый и надежный отдельный накопитель.
2. Ввиду их небольшого размера, резервные копии журналов следует делать по возможности чаще, и точно не реже чем через час.
3. После создания очередной полной резервной копии базы данных, все ранее сохраненные копии журнала транзакций (как и разностные копии) можно и следует удалить.
4. Постоянный мониторинг состояния и наличия свободного места на диске, на котором происходит журналирование и создание резервных копий журнала транзакций. В случае возникновения проблем с журналом транзакций, СУБД потеряет возможность осуществлять дальнейшие транзакции.
5. Добавление в скрипты резервного копирования журналов транзакций функции усечения (SHRINK) для того, чтобы избежать их переполнения (действие опционально).

3.3. Документация и скрипты резервного копирования.

Сам процесс резервного копирования в организациях регламентируется соответствующей нормативной документацией. Как правило, комплекты документов создаются в рамках текущей политики безопасности самой организации, но в целом, ключевые документы имеют одинаковые заложенные параметры резервного копирования. В контексте нашей лекции мы рассмотрим типовую документацию, связанную с анализом параметров сервера баз данных и его окружения, а также план резервного копирования.

Исследование параметров функционирования базы данных позволяет определить эффективное время и допустимое время процедуры резервного копирования баз данных, а также усредненную длительность этого процесса и нагрузку на сетевое оборудование без сбора большого количества бизнес-требований и длительного исследования регламента работы организации. Так, для определения оптимального времени проведения процедур резервного копирования, как правило, достаточно знать время основной нагрузки на сервер баз данных в целом, и на отдельные базы данных, подвергаемые копированию в частности. Также требуется получить базовые количественные показатели скорости записи и загрузки на стороне серверного оборудования. Пример документа «Планируемые характеристики объекта резервного копирования» показан в табл. 3.

Таблица 3. Образец документа «Планируемые характеристики объекта резервного копирования»

Характеристики	Ожидаемое значение
Размер базы данных	2 ГБ
Совокупная скорость резервного копирования базы данных	35 МБ/с
Совокупная скорость резервного восстановления базы данных	75 МБ/с
Средний уровень изменения базы данных за рабочее время (под нагрузкой)	2 МБ/д
Средняя доля изменения базы данных за рабочее время, по отношению к общему размеру (под нагрузкой)	1%
Рабочее время (пн-пт)	9:00-18:00

Перед составлением *плана резервного копирования*, в результате коллективной работы пользователей и администраторов баз данных составляется набор бизнес-требований к процедуре резервного копирования баз данных. В этих требованиях, в произвольной форме отмечаются базы данных, которые подлежат резервному копированию, а также периодичность и время снятия резервных копий. Приведем пример фрагмента таких бизнес-требований.

1. Полная копия базы данных Adventure Works должна создаваться один раз в неделю.
2. Разностная копия базы данных Adventure Works делается каждый день.
3. Копии журнала транзакций базы данных Adventure Works делаются каждый час.
4. Копия системной базы данных master (ключевая для нормальной работы сервера системная база данных) делается раз в неделю.
5. Копия системной базы данных msdb (ключевая для нормальной работы сервера системная база данных) делается раз в неделю.
6. Копии с высокой и длительной нагрузкой на сервер баз данных (в первую очередь полная копия базы данных Adventure Works) должны выполняться в нерабочее время офиса организации.

Сам документ План резервного копирования регламентирует день, время и частоту регулярных процедур резервного копирования и является базой для автоматизации этого рутинного процесса. Пример документа показан в табл. 4.

Таблица 4. Документ “План резервного копирования”

День недели	Время	Действия	Частота	Описание
Понедельник-пятница	8-00 до 21-00	Журнал транзакций	Каждый час	Сжатие
Суббота, Воскресенье	8-00 до 18-00	Журнал транзакций	Каждый час	Сжатие
Понедельник-Воскресенье	22-00	Дифферен. копия	Раз в день	Старые копии журналов удалить
Суббота	12-00	Проверка БД	Раз в день	Целостность
Суббота	18-00	Полная копия	Раз в день	Удаление всех неактуальных копий
Воскресенье	23-30	Копия master	Раз в день	
Воскресенье	12-30	Копия msdb	Раз в день	

В поле описание добавлены дополнительные заметки, которые необходимо учитывать при осуществлении настройки процедуры копирования. В приведенном примере указаны следующие дополнительные указания – необходимость применения функции SHRINK для всех журналов транзакций, необходимость удаления старых резервных копий, регулярная проверка целостности базы данных и согласованности данных.

На основании плана резервного копирования настраиваются агенты автоматизации СУБД, после чего обслуживание резервных копий проходит в полуавтоматическом режиме. О принципах автоматизации функций администратора баз данных будет рассказано в одной из последующих лекций курса.

Типовые скрипты резервного копирования и восстановления данных для языка tSQL приведены в Рабочей тетради студента по дисциплине «Проектирование и администрирование хранилищ и баз данных, Часть 2» и будут разобраны в ходе курса в рамках соответствующей практической работы. Ниже будут разобраны базовые скрипты резервного копирования и восстановления для СУБД PostgreSQL и MongoDB.

Базовым методом резервного копирования PostgreSQL является применение процедуры *pg_dump*. В ходе своего выполнения, данная процедура соединяется с целевой базой данных и открывает большую транзакцию с настройкой Repeatable Read (см. глава 3), которая считывает все данные с целевой базы

данных. Благодаря функции Repeatable Read, на действия транзакции, осуществляющей копирование не сможет повлиять ни одна другая транзакция, параллельно происходящая в базе данных, и итоговый массив данных гарантированно будет согласованным. Результатом работы процедуры `pg_dump` по умолчанию является простой текстовый документ со всеми данными целевой базы данных. Приведем базовую команду процедуры `pg_dump`:

```
[linuxpc ~] $ pg_dump -U /пользователь БД/ /копируемая БД/ > /путь к файлу и название файла.sql/
```

Флаг `-U` указывает на пользователя, от имени которого будет осуществляться резервное копирование. Если СУБД не требует такого рода проксирования, этот флаг можно опустить. Перечислим иные важные флаги для процедуры `pg_dump`.

`-d, --dbname = /название базы данных/,` — явное указание целевой базы данных.

`-h, --host = /название хоста/,` — имя хоста на котором находится целевой сервер баз данных.

`-p, --port = /номер порта/,` — указание на порт доступа к БД на стороне хоста.

`-W, --password,` — указание на то, что при использовании пользователя необходимо указать его пароль.

Также, с помощью флагов можно добиться более гибкой настройки массива копируемых данных.

`-a:` копируются только данные, структура данных не копируется;

`-s:` копируется только структура данных, без данных;

`-n:` копируется только указанная схема данных;

`-N:` копируется все, кроме указанных схем данных;

`-t:` копируются только указанные таблицы;

`-T:` копируется все, кроме указанных таблиц.

В случае возникновения необходимости скопировать не какую-то отдельную базу данных, а все содержимое сервера целиком используется процедура `pg_dumpall`.

```
[linuxpc ~] $ pg_dumpall > /путь к файлу и название файла.sql/
```

Восстановление базы данных из резервной копии происходит с использованием процедуры `pg_restore`. При восстановлении следует обратить внимание на то, что процедура может добавлять данные даже в существующие базы данных (возможна ошибка, но технически данные будут добавлены).

```
[linuxpc ~] $ pg_restore -d /целевая база данных/ /путь к файлу и название файла/
```

Процедура `pg_dump` хорошо показывает себя на маленьких и средних массивах данных, хранимых в базах. В случае больших массивов данных будет

целесообразным рассмотреть возможность настройки репликации базы данных. Об этом будет рассказано в последующих лекциях.

Базовым методом резервного копирования документальной СУБД MongoDB является применение процедуры *mongo_dump*. Приведем базовую команду процедуры *mongo_dump*:

```
mongodump --host /имя хоста целевой базы данных/ --port /номер порта доступа к серверу/ --username /имя пользователя/ --password /пароль пользователя/ --out /путь к файлу резервной копии/
```

Для масштабирования резервной копии можно применять флаги *--db /имя базы данных для резервной копии/* и *--collection /имя коллекции для резервной копии/*. Восстановление базы данных из резервной копии происходит с использованием процедуры *mongorestore*.

```
mongorestore --host /имя хоста/ --port /номер порта доступа к серверу/ --username /имя пользователя/ --password /пароль пользователя/ --out /путь к файлу резервной копии/
```

3.4. Вопросы для самостоятельного изучения по итогам лекции

1. С помощью какой инструкции SQL вызвать просмотр журнала транзакций MS SQL Server?
2. Как настроить инструкцию SQL, чтобы увидеть журнал транзакций, максимально приближенный к образцу на рисунке 17?
3. Составьте таблицу и скрипты резервного копирования для вашего проекта из Курсовой работы по дисциплине Проектирование и администрирование хранилищ и баз данных.

3.5. Тестовые задания для самопроверки

1. Для получения списка файлов данных и журналов транзакций, входящих в набор резервных копий используется следующий оператор Transact-SQL:
 - A) RESTORE labelOnly FROM
 - Б) RESTORE headerOnly FROM
 - В) RESTORE data FROM
 - Г) RESTORE fileListOnly FROM
2. При применении модели восстановления SIMPLE для восстановления БД минимум необходимо:
 - A) только разностные копии
 - Б) соответствующая полная копия БД и при необходимости разностные копии
 - В) соответствующая полная копия и вся цепочка копий журналов

транзакций

Г) полный набор копий (полная, разностная, журнал транзакций)

3. Какая из предложенных инструкций установит полную модель восстановления для конкретной БД.

А) ALTER DATABASE dbname SET RECOVERY FULL

Б) RECOVERY MODEL FULL TO DATABASE dbname

В) ALTER DATABASE dbname RECOVERY AS FULL

Г) ALTER TABLE dbname SET RECOVERY FULL

4. Какие модели восстановления поддерживает компонент Database Engine?

А) простая, полная, с неполным протоколированием

Б) простая, сложная, комбинированная

В) простая, полная, комбинированная

Г) простая, полная, простая с протоколированием

5. Что из перечисленного нельзя реализовать с помощью журнала транзакций?

А) восстановление отдельных транзакций

Б) восстановление всех незавершенных транзакций

В) накат файла, файловой группы

Г) все перечисленное реализуемо

6. Усечение журнала транзакций происходит с использованием команды:

А) DBCC LOW

Б) DBCC SMALL

В) DBCC SHRINK

Г) DBCC CLEAR

7. Резервная копия, содержащая данные, накопленные «между» полным копированием, называется

А) дифференцированная

Б) диверсифицированная

В) дискриминированная

8. Резервная копия, каждый раз подлежащая ручной настройке со стороны администратора, называется

- А) файловая копия
- Б) полная копия
- В) копия журнала транзакций

9. Последовательность восстановления БД MS SQL Server «по умолчанию»:

- А) полная копия – файловая копия
- Б) полная копия – дифференцированная копия
- В) файловая копия – копия журнала транзакций

10 Обязательно исполнение условия копирования журнала транзакций:

- А) копирование с периодичностью в 10 минут
- Б) проверка файла после создания резервной копии
- В) копирование на отдельный носитель

4. РЕПЛИКАЦИЯ В МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗАХ ДАННЫХ

4.1. Определение репликации базы данных

Понятие репликация, в вычислительной технике неразрывно связано с процедурой копирования. По своей сути *репликация* – это механизм синхронизации содержимого нескольких копий одного объекта. В разговоре про многопользовательские базы данных под этим объектом понимается база данных. Суть репликации данных – тиражирование изменений данных, произошедших на одном сервере, на все связанные репликацией сервера.

Исторически, репликация связана с определением распределенной базы данных. *Распределённая база данных (DDB)* - база данных, составные части которой размещаются в различных узлах компьютерной сети в соответствии с каким-либо критерием.

Данные представляют собой DDB, только если они связаны в соответствии с некоторым структурным формализмом (правилами проектирования), реляционной моделью, а доступ к ним обеспечивается единым высокоуровневым интерфейсом.

Центральная идея распределенной базы данных – доступность данных в любое время, в любом месте.

Очевидно, что возникновению феномена распределенных баз данных способствовал рост крупных компаний. Они открывали филиалы в новых городах, странах и даже на новых для себя континентах. IT-инфраструктура этих компаний усложнялась, доступ к данным, обращающимся внутри системы существенно замедлялся и усложнялся. В этих обстоятельствах и зародилась идея создания распределенных баз данных, которые создавали существенные преимущества для крупных и разросшихся до невероятных размеров компаний. Перечислим ключевые из них:

- Данные доступны людям, которым они нужны, и когда они нужны.
- Система позволяет локальному пользователю автономно оперировать данными (нет лишних проблем с параллельной обработкой данных, см. лекция 2).
- Система, с точки зрения общей IT-архитектуры организации, сокращает сетевой трафик.
- Процесс обеспечения непрерывности бизнеса с точки зрения БД дешевле (значительно большее количество точек хранения данных ускоряет процесс восстановления после критических сбоев).

Далее будут рассмотрены три основных инструмента обеспечения

функционирования распределенных баз данных – распределенная транзакция, репликация и шардинг. Особо внимание, ввиду наибольшей актуальности будет уделено репликации.

4.2. Распределенная транзакция и репликация

Самой первой технологией управления распределенными базами данных стали так называемые распределенные транзакции. *Распределенная транзакция* – транзакция, по определенным правилам собирающая в себе все актуальные изменения, которые происходят в распределенных базах данных, а затем управляющая процессом синхронизации массивов данных на всех узлах распределенных баз данных.

Ключевым инструментом при осуществлении распределенных транзакций является *Менеджер глобального восстановления (или проще - координатор)*. В ходе двухфазового подтверждения (все ли узлы готовы принять новые данные; везде ли распределенная транзакция завершилась успешно), этим менеджером осуществляется реализация актуальной распределенной транзакции на всех узлах распределенной сети. Схематично, распределенная транзакция показана на рис. 15.



Рисунок 15. Этапы выполнения распределенной транзакции

Обратите внимание на то, что процедура распределенной транзакции не закончится, пока координатор-менеджер не убедится в том, что транзакция была успешно выполнена на всех узлах распределенной базы данных. А представим, что этих узлов не 4-5, а 50-60. Какова вероятность быстро и успешно закончить распределенную транзакцию? Именно по причине наличия такого ограничения, в случае необходимости работы с распределенными базами данных администраторы предпочитают настраивать один из вариантов репликации.

В настоящее время, помимо указанного выше требования «данные должны обрабатываться там, где они находятся», выделяют 4 основных причины применения инструмента репликации при обеспечении работы базы данных организации.

1. *Производительность и масштабируемость.* Снятие с основного сервера базы данных части нагрузки по обработке данных. В основном это касается делегирования вспомогательным серверам инструкций по чтению и обработке данных. Чем больше операций чтения приходится на одну операцию записи, тем больше выгоды от репликации.

2. *Отказоустойчивость.* Если вдруг по какой-то причине, один или несколько вспомогательных серверов баз данных в репликации перестанут функционировать, главный сервер тут же может «перехватить» на себя операции, которые ранее выполняли вспомогательные серверы. Такие действия возможны и в обратном направлении, когда вспомогательный сервер на время выполняет функции записи и хранения данных, характерные для основного сервера.
3. *Резервное копирование данных.* Существенно упрощается работа администратора, связанная с необходимостью создания резервных копий (Лекция 3). Зачастую можно даже не руководствоваться бизнес-требованиями и планами резервного копирования. Дело в том, что вспомогательный сервер в репликации можно остановить и снять резервную копию в любое время.
4. *Отложенные вычисления.* Огромные по объему SQL запросы, которые регулярно и очень сильно нагружают вычислительные мощности основного сервера выгодно выполнять на отдельном вспомогательном сервере.

Приведем основную терминологию, применяемую в репликации.

Издатель — основной сервер баз данных в репликации. В большинстве случаев ориентирован на процедуры, связанные с записью и хранением данных.

Подписчик — вспомогательные серверы баз данных в репликации. В большинстве случаев ориентированы на процедуры, связанные с чтением и обработкой данных.

Распространитель — скрипты, осуществляющие репликацию данных между издателем и подписчиками.

Публикация — информация, которая подвергается репликации. Одна публикация – это один согласованный набор данных.

В состав публикации входят *статьи*, которые могут быть:

- целой таблицей или ее частью;
- хранимой процедурой или представлением;
- пользовательской функцией.

Отметим, что для публикаций и статей (как для единиц данных для репликации) существует *ряд важных ограничений*.

- Статья содержит данные из таблицы и одной или нескольких хранимых процедур.
- Таблица может быть как целой, так и подмножеством.
- В публикации можно собрать несколько статей.

- Каждая публикация должна содержать данные только из одной базы данных.
- Подписаться на статью нельзя.

На рис. 16 показаны фильтры, которые можно применить при формировании массива данных для статьи.



Рисунок 16. Фильтры для массива данных статьи

Так, в состав статьи может входить таблица данных целиком, выборка из таблицы с условием по столбцам, выборка из таблицы с условием по строкам, выборка из таблицы с комбинацией условий по столбцам и строкам.

Инициализация подписки происходит push (принудительная) или pull (запрос) методом. Описание типов подписок показано на рис. 17.

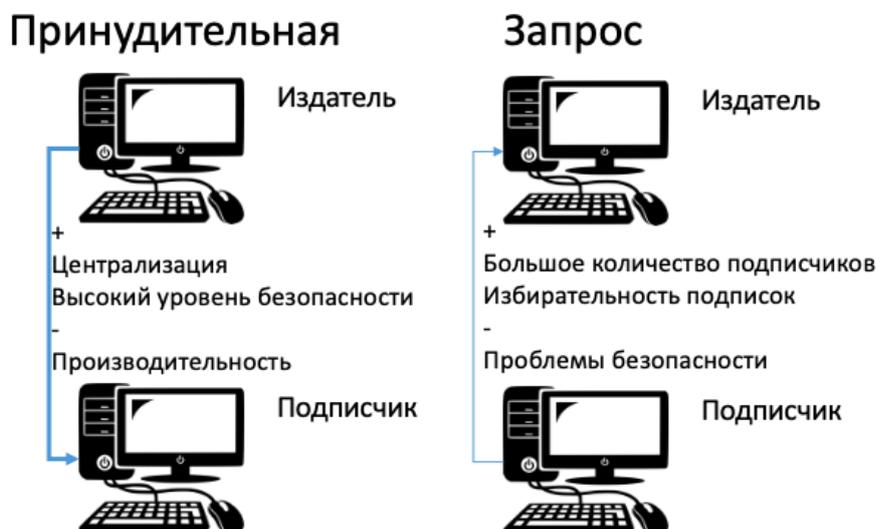


Рисунок 17. Схема типов подписок репликации

Ключевым различием в типах подписок является направление подписки. В случае push (принудительной) подписки, издатель обязует подписчика принять направленные ему публикации. Ключевой проблемой при этом является производительность распределенной базы данных, поскольку управление всеми подписками происходит централизованно, на стороне издателя.

В случае pull (запроса) подписки, подписчик запрашивает необходимые ему публикации, и издатель в ответ их предоставляет. Ключевой проблемой является проблема безопасности, поскольку у большого количества подписчиков появится возможность формировать запросы к издателю.

4.3. Типы репликации MS SQL Server 2018

1. *Репликация транзакций.* Все транзакции, которые отмечены для репликации «вылавливаются» из журналов транзакций агентом чтения журналов (Log reader agent), после чего копируются в специальную системную базу данных distribution. Далее, с помощью распространителя (Distribution agent), транзакции распределяются по подписчикам, где и исполняются. Существует два ограничения, которые должны быть соблюдены при реализации такого варианта репликации:

- перед настройкой репликации транзакций каждый Подписчик должен получить и развернуть полную резервную копию Издателя у себя;
- у каждой реплицируемой таблицы должен быть первичный ключ.

Схематично репликация транзакций показана на рис. 18.



Рисунок 18. Схема репликации транзакций

2. *Репликация мгновенного снимка (snapshot).* Репликация осуществляется не по необходимости, а периодически, в рамках заранее настроенного интервала времени. Компонент СУБД Snapshot Agent генерирует схему и пакет данных для таблиц входящих в Публикацию, и агрегирует их в один файл, после чего, этот файл передается Подписчиком. Ряд условий, сопровождающих данный вид репликации:

- база данных Distribution, в отличие от случая репликации транзакций, напрямую не используется;
- при репликации мгновенного снимка возможен только один тип подписки - от Издателя к Подписчику;

- наличие первичного ключа в реплицируемых таблицах не обязательно.

Схематично репликация мгновенного снимка показана на рис. 19.

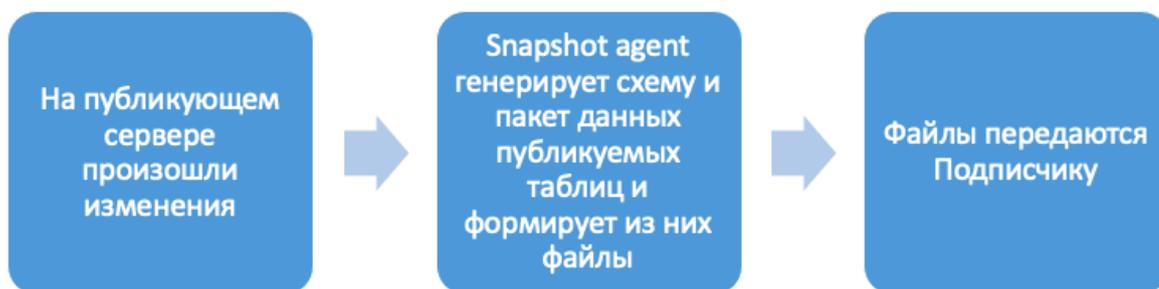


Рисунок 19. Схема репликации мгновенного снимка

3. *Репликация слияния (merge)*. Данный вид репликации во многом дублирует репликацию мгновенного снимка, за исключением этапа, когда специальный Merge Agent изменяет файлы, содержащие схему и пакет данных для Публикации перед тем, как передать их Подписчикам. К файлу Публикации добавляются системные таблицы, первичные ключи (где это необходимо) и системные триггеры. Происходит этого для того, чтобы обеспечить оба типа подписки – как push, так и pull. Из-за существенных изменений в файле Публикации, при обновлении серверов могут возникать конфликты, которые будут решаться проверкой приоритетов (по умолчанию – «первый побеждает») или действиями пользователя.

Схематично репликация слияния показана на рис. 20.

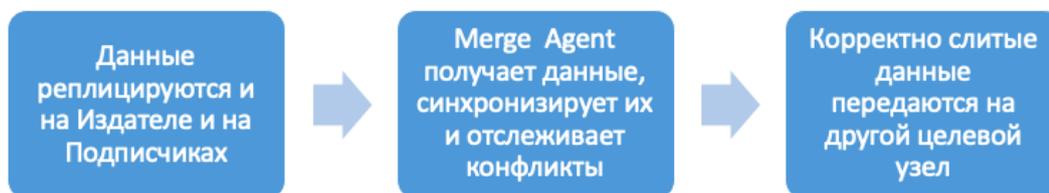


Рисунок 20. Схема репликации слияния

4. *Одноранговая репликация (peer-to-peer)*. Вид репликации, при котором все серверы баз данных, участвующие в репликации не имеют четкого иерархического разделения. Перечислим особенности данного типа репликации:

- все серверы баз данных, участвующие в репликации настроены с возможностью функционирования как издатель-распространитель-подписчик и по необходимости могут переключаться в любой режим работы;
- все серверы обладают одним и тем же набором и схемами данных, постоянно синхронизируя их друг с другом;
- каждый сервер манипулирует своим подмножеством данных, передавая его в режиме издателя остальным подписчикам;
- любой конфликт обновления при передаче Публикации будет считаться ошибкой, требующей исправления.

Схематично репликация слияния показана на рис. 21.



Рисунок 21. Схема одноранговой репликации

Сравнительное описание типов репликации показано в табл. 5.

Таблица 5. Типы репликации баз данных

Аспект	Репликация транзакций	Snapshot	Репликация слияния	Одноранговая репликация
Транзакционность	Да	Нет	Нет	Да
Сложность	Средняя	Низкая	Средняя	Высокая
Подписчики обновляются?	Ограничения	Нет	Да	Разделенными данными
Обнаружение конфликта	Нет	Нет	Да	Да
Решение конфликта	Нет	Нет	Да	Нет
Производительность	Высокая	Средняя	Низкая	Высокая

Практическая реализация механизма репликации сервера баз данных MS SQL Server 2018, в соответствии с технической документацией, рекомендуется с использованием Management Studio и встроенного мастера настройки агентов репликации. Практическая работа с мастером будет разобрана в ходе лабораторных работ курса. Более подробно про объекты автоматизации будет рассказано в материале Лекции 5.

4.4. Настройка репликации и шардинга в MongoDB

Отдельно следует рассмотреть базовые принципы репликации для noSQL структуры хранения данных. Как правило, noSQL модель используется в случае обработки параллельных массивов больших данных, что, в большинстве случаев создает предпосылки к репликации хранимых данных. В документной СУБД MongoDB выделяют два способа управления распределёнными данными – репликация и шардирование (шардинг).

Оптимальный подход к репликации в MongoDB – *асинхронная репликация с тремя узлами*. Свойство асинхронности говорит о том, что синхронизация данных Издателя с Подписчиками происходит не в режиме реального времени, а с задержкой. Очевидно, что это позволит не увеличивать и без того огромную

нагрузку на сервер данных и на сетевое оборудование. Три узла репликации – это соответственно Издатель, Подписчик и Арбитр.

Издатель – первичный узел хранения данных.

Подписчик – вторичный узел хранения данных.

Арбитр – узел, не содержащий массива данных, но обеспечивающий принятие решения об изменении свойств первичного и вторичного узла (например, в случае временной недоступности первичного узла хранения).

У данной структуры есть одна важная особенность – очень нежелательно узел Арбитр размещать на одном из узлов хранения. Это связано с автоматической системой опроса Арбитром узлов хранения, которая называется Сердцебиение (heartbeats), которая, в случае отсутствия отклика со стороны узла хранения позволяет Арбитру считать его вышедшим из строя и включает функцию изменения статуса других узлов хранения, находящихся в строю. Соответственно, если Арбитр будет находиться на одной машине с вышедшим из строя узлом хранения, он свои автоматизированные функции выполнять не сможет.

Рассмотрим процедуры MongoDB, использующиеся при создании и обеспечении функционирования репликации. Для создания схемы репликации, описанной выше, необходимо выполнить следующие шаги.

1. Создание узлов, участвующих в репликации. Как минимум необходимо создать три узла командой:

```
mongod --replSet /название репликации, одно для всех узлов/ --host /адрес сервера базы данных/ --port /номер порта базы данных/ --dbpath /путь к категории базы данных, созданной заранее/.
```

2. Инициация первичного узла хранения. Запустив терминал MongoDB на узле-сервере, который будет служить как первичный, необходимо выполнить команду `rs.initiate()`, которая устанавливает выбранный узел в режим Primary (первичный).

3. Добавление вторичного узла хранения. Происходит также в терминале первичного узла-сервера выполнением команды `rs.add(«адрес вторичного сервера:порт вторичного сервера»)`.

4. Добавление узла Арбитр. Происходит также в терминале первичного узла-сервера выполнением команды `rs.add(«адрес сервера арбитра:порт сервера арбитра», {arbiterOnly: true})`.

В отличие от репликации, *шардирование* MongoDB – это разделение большого массива данных между несколькими узлами. Эта функция будет полезна при обработке и хранении действительно больших объемов данных (например петабайт), с чем вряд ли справится один отдельно взятый сервер. При этом, интерфейс MongoDB позволяет работать со всей совокупностью *шардов* (кусочков

данных, разнесенных по нескольким серверам), как с базой данных, расположенной на одном сервере.

Для настройки шардинга проделывают следующие процедуры.

1. Запуск конфигурационного сервера, маршрутизирующего потоки данных между шардами командой `mongod --configsvr --dbpath /путь к категории базы данных, созданной заранее/ --port /номер порта базы данных/`

2. Настройка маршрутизации с конфигурационным сервером командой `mongos --configdb --host /адрес сервера базы данных/ --port /номер порта базы данных/`

3. Создание шардов командой `mongos> sh.addShard(«адрес сервера базы данных:номер порта базы данных»)`, повторяем выполнение команд столько раз, сколько необходимо шардов.

4. Запуск шардинга командой `mongos> sh.enableSharding(«название коллекции подлежащей шардингу»)`.

Попробовать создать структуру репликации и шардинга `mongoDB` студенты смогут, выполнив самостоятельную домашнюю работу по материалам данной лекции.

4.5. Вопросы для самостоятельного изучения по итогам лекции

1. Как называется сервер, который содержит полную актуальную копию мастер-сервера БД? В каких случаях используется?

2. Компания имеет филиалы в разных городах. В каждом филиале есть свой сервер БД, который имеет такой же набор данных, что и остальные. Какую репликацию данных стоит реализовать в данном случае. Почему?

3. В чем отличие репликации от шардинга в `MongoDB`?

4.6. Тестовые задания для самопроверки

1. Какой из приведенных терминов не относится к технологии репликации данных?

- А) тираж
- Б) издатель
- В) статья
- Г) подписчик

2. Какой из перечисленных фильтров не может быть применен для формирования статей при репликации?

- А) горизонтальный

- Б) вертикальный
- В) таблица
- Г) все перечисленные могут быть использованы

3. В каком из перечисленных типов репликации допустимо формирование публикаций на стороне подписчика?

- А) репликация слияния
- Б) репликация мгновенного снимка
- В) репликация транзакций

4. В каком из перечисленных типов репликации издатель является одновременно и подписчиком?

- А) одноранговая репликация
- Б) репликация слияния
- В) репликация мгновенного снимка
- Г) репликация транзакций

5. При какой репликации необходима передача полной резервной копии издателя подписчику?

- А) одноранговая репликация
- Б) репликация слияния
- В) репликация мгновенного снимка
- Г) репликация транзакций

6. Какой из перечисленных критериев не характерен для термина распределенные данные?

- А) есть возможность автономного оперирования данными
- Б) доступ к данным в точке их получения-хранения
- В) система увеличивает сетевой трафик
- Г) все критерии характерны

7. Распределенной транзакцией управляет...

- А) координатор
- Б) мастер
- В) репликатор
- Г) распространитель

8 Какое из перечисленных ограничений статей неверное?

А) статья содержит данные из таблицы

Б) несколько статей, это публикация

В) подписаться на статью нельзя

Г) все ограничения верны

5. АВТОМАТИЗАЦИЯ ЗАДАЧ АДМИНИСТРИРОВАНИЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ

Рассмотренные в предыдущих главах функции администрирования баз данных, такие как резервное копирование и репликация, как правило, являются рутинными процедурами, выполняемыми с заранее заданной цикличностью. В таких условиях целесообразным является максимальная автоматизация запуска, мониторинга и подведения итогов этих процедур. Требуемая автоматизация достигается или созданием программных оболочек администрирования баз данных, или имеющихся в составе СУБД средств автоматизации. Одним из наиболее простых и наглядных средств автоматизации рутинных операций администрирования баз данных является SQL Server Agent для СУБД Microsoft SQL Server.

В рамках современных СУБД, функцию автоматизации с целью обеспечения присутствия правильного управляющего или корректирующего воздействия берут на себя специальные программные компоненты. Так, например, в MS SQL Server такой программный компонент называется агентом. Агент SQL Server является одним из наиболее простых и наглядных средств автоматизации рутинных операций администрирования баз данных.

Перечислим типовые задачи, доступные для автоматизации при помощи агента SQL Server:

- запуск, мониторинг и журналирование процедур полного, дифференцированного резервного копирования;
- запуск, мониторинг и журналирование процедур резервного копирования журнала транзакций;
- запуск исполняемой программы или утилиты;
- выполнение задач Distribution Agent при репликации;
- выполнение задач Merge Agent при репликации;
- выполнение задач Snapshot Agent при репликации.

Схема компонентов СУБД MS SQL Server, настраиваемых при автоматизации администрирования баз данных показана на рис. 21.

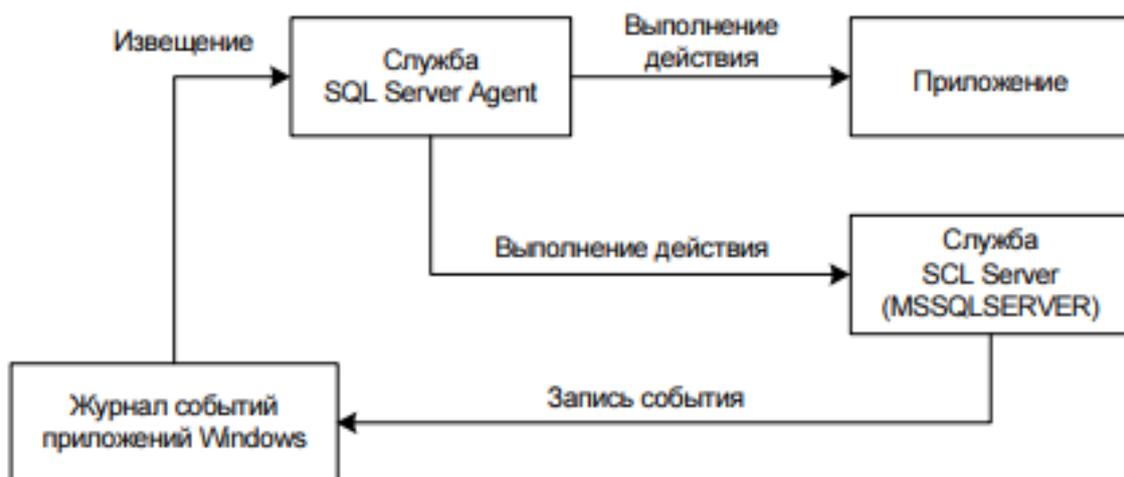


Рисунок 22. Структурные компоненты СУБД, обслуживающие процедуру автоматизации администрирования

Ключевым компонентом является *Служба SQL Server Agent*, которая хранит скрипты действий для агентов, настройки агентов и другую метаинформацию, которая требуется для их функционирования.

Приложение – это исполняемое приложение или утилита, к которой в рамках автоматизации администрирования может обратиться агент. Как правило, это утилиты, контролирующие нормальную работу сервера баз данных, а также его запуск-останов.

Служба MSSQLSERVER – служба, обеспечивающая функционирование ядра сервера. Действия агента, направленные на эту службу, автоматизируют перечисленные выше основные функции администрирования.

Все события, полученные в ходе мониторинга исполняемой процедуры администрирования и по итогам этого процесса журналируются в *Журнале событий приложений Windows*. Этот журнал обладает удобной фильтрацией, что позволяет достаточно быстро получить информацию о ходе и результатах любой процедуры агента.

Перечислим компоненты Агента MS SQL Server.

Задание – определенная администратором последовательность действий, реализующая процедуру автоматизации администрирования баз данных. Задание может быть выполнено один или несколько раз на локальном или удаленных серверах. Выполнение задания может осуществляться несколькими способами:

- с помощью настроенного в мастере одного или нескольких расписаний;
- с помощью настроенного в СУБД *алерта* (alert, предупреждения);
- через выполнение хранимой процедуры `sp_start_job`.

Задание может состоять из нескольких последовательных шагов (*связанные задачи*). Базовые настройки задания показаны на рис. 23.

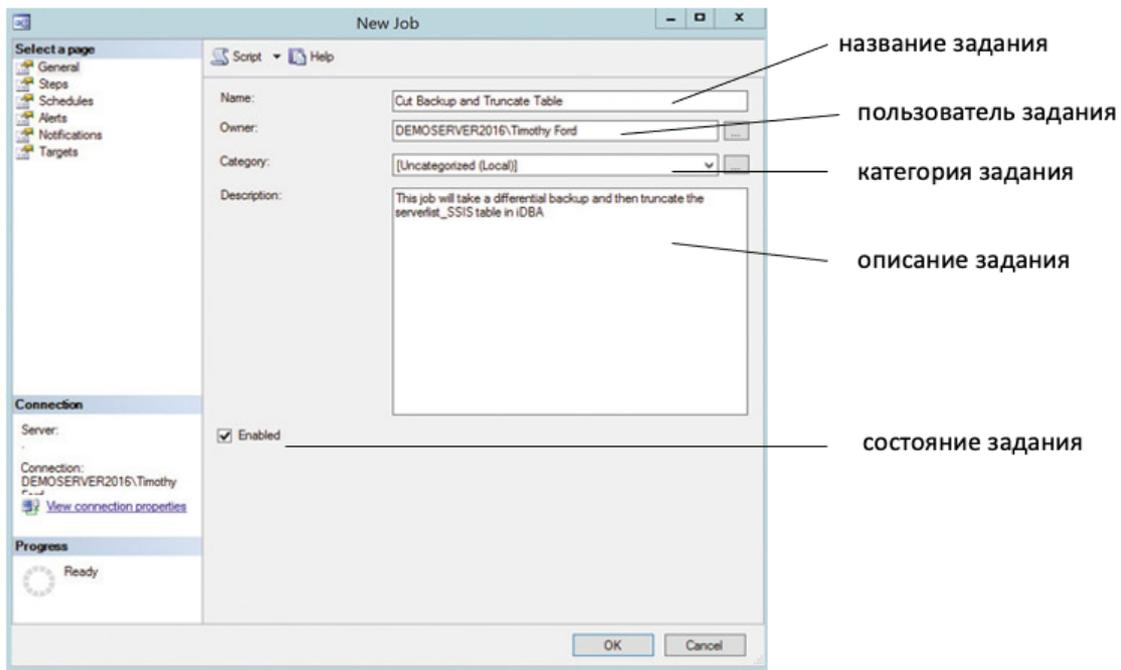


Рисунок 23. Базовые (General) настройки задания (Job) агента

В базовых настройках задания указывается имя задания (желательно, чтобы имя, при своей лаконичности, указывало на процедуру, которая реализуется в задании агентом), пользователь задания (прокси-пользователь, от имени которого будет выполняться задание), категория задания и краткое описание процедуры задания.

В случае необходимости разбиения задания на шаги используются продвинутые (Advanced) настройки агента автоматизации, рис. 24.

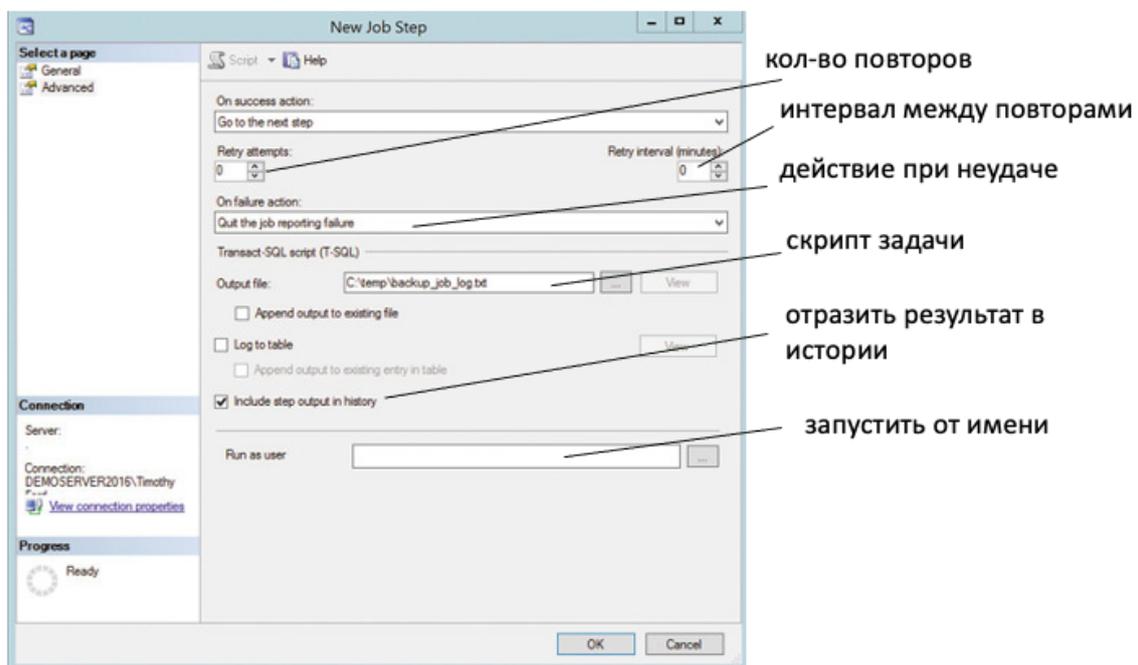


Рисунок 24. Продвинутые настройки шагов задания

Ключевыми элементами настройки шагов задания является указание действия при успехе (on success) и неудаче (on failure) при выполнении задания.

Расписание – определенное администратором время выполнения задания. Для одного задания может быть создано несколько расписаний, также как и несколько заданий могут управляться одним расписанием. Условием времени выполнения задания по расписанию может являться:

- запуск агента MS SQL Server;
- загрузка центрального процессора сервера, определенная как: «простой»;
- однажды, в указанные дату и время или с установленной периодичностью.

Настройки расписания агента (Schedule) показаны на рис. 25.

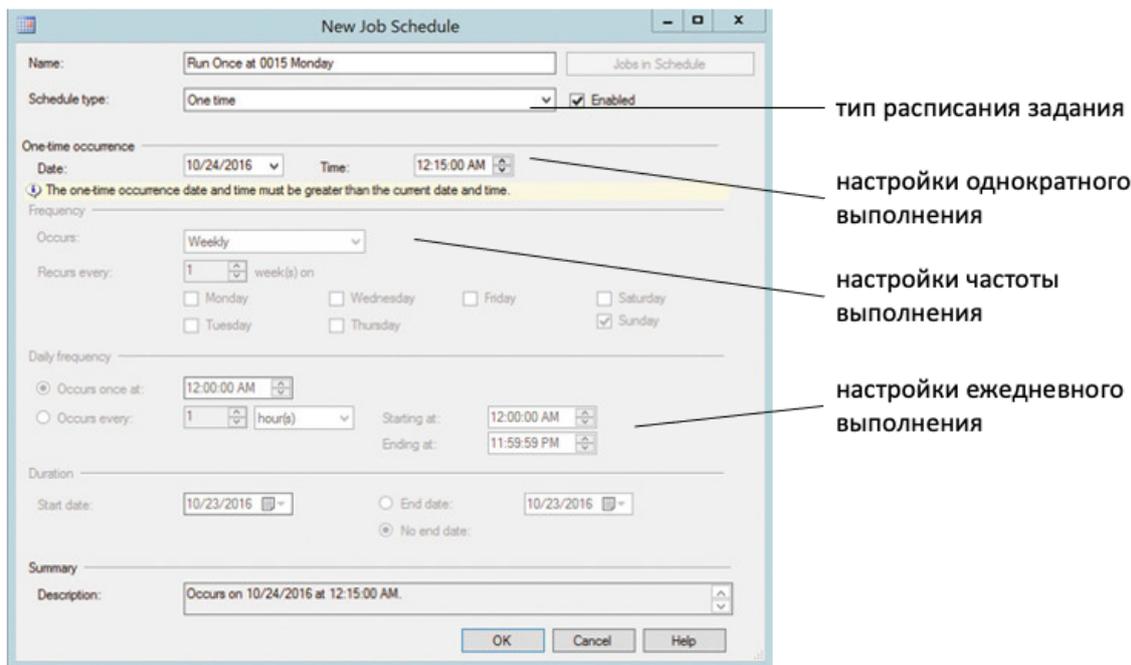


Рисунок 25. Составление расписание агента MS SQL Server

В случае выбора однократного выполнения процедуры, достаточно указать дату и время ее выполнения. Есть варианты ежедневного, еженедельного выполнения процедуры. Также можно указать дни недели и время выполнения процедуры. Дополнительно можно указать длительность периода регулярных выполнений задания, если это необходимо.

Предупреждение (Notification) – автоматический ответ на наступление запрограммированного события. Этим событием может быть начало или конец выполнения задания, или же системным ресурсом, которое достигло установленного порогового значения. Результатом предупреждения может стать уведомление администратора о возникшем событии или автоматическое выполнение запрограммированного задания.

Настройки предупреждения (Notifications) показаны на рис. 26.

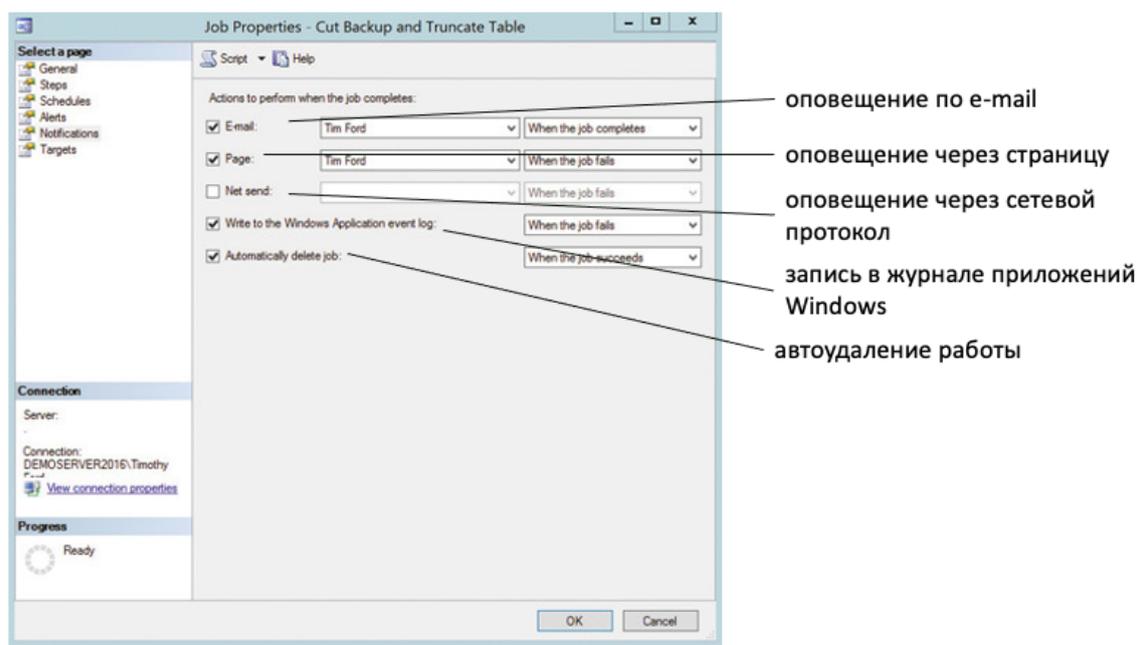


Рисунок 26. Настройки предупреждения для задания агента MS SQL Server

Адресатом получения Предупреждения является Оператор. Это совокупность контактных сведений лица, ответственного за администрирование сервера баз данных. Как правило, предупреждения настраиваются на доставку электронной почтой.

Ключевым элементом обработки результатов выполнения заданий является обработка *сообщений об ошибках (alert)*. Это сообщения, содержащие информацию о нештатной ситуации выполнения задания. Структура сообщения об ошибке:

- однозначный номер сообщения об ошибке;
- число в диапазоне от 0 до 25, представляющее уровень серьезности ошибки;
- номер строки, в которой произошла ошибка;
- текст описания ошибки.

Пример сообщения об ошибке: (Сообщение 208, Уровень 15, Строка 2 Недействительное имя объекта 'authors').

Уровень серьезности ошибки является самым важным показателем любого alert и указывает на критичность и последствия возникшей в ходе выполнения процедуры ошибки.

Уровни от 0 до 10 обозначают просто информационные сообщения, где ничего не требуется исправлять.

Все уровни ошибок от 11 до 16 указывают программные ошибки, которые могут быть разрешены пользователем.

Значения уровней 17 и 18 обозначают программные и аппаратные ошибки,

которые обычно не завершают выполнение процесса.

Все ошибки уровня 19 и выше являются неисправимыми системными ошибками. Соединение программы, вызвавшей такую ошибку, закрывается, после чего ее процесс удаляется.

Базовые настройки сообщения об ошибке (General) показаны на рис. 27.

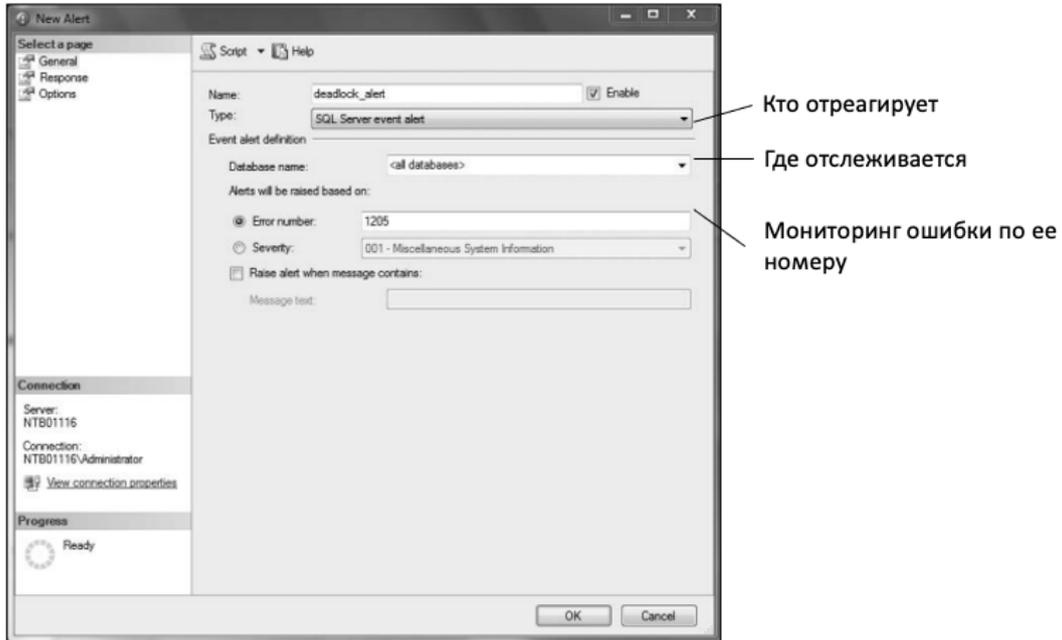


Рисунок 27. Настройки сообщения об ошибке

В базовых настройках необходимо указать имя сообщения, имя базы данных в которой будет отслеживаться возникновение ошибки (есть настройка all databases), а также номер ошибки или уровень серьезности ошибки. Следующим этапом настройки сообщения об ошибке является настройка ответа СУБД на ее возникновение (рис. 28, Response).

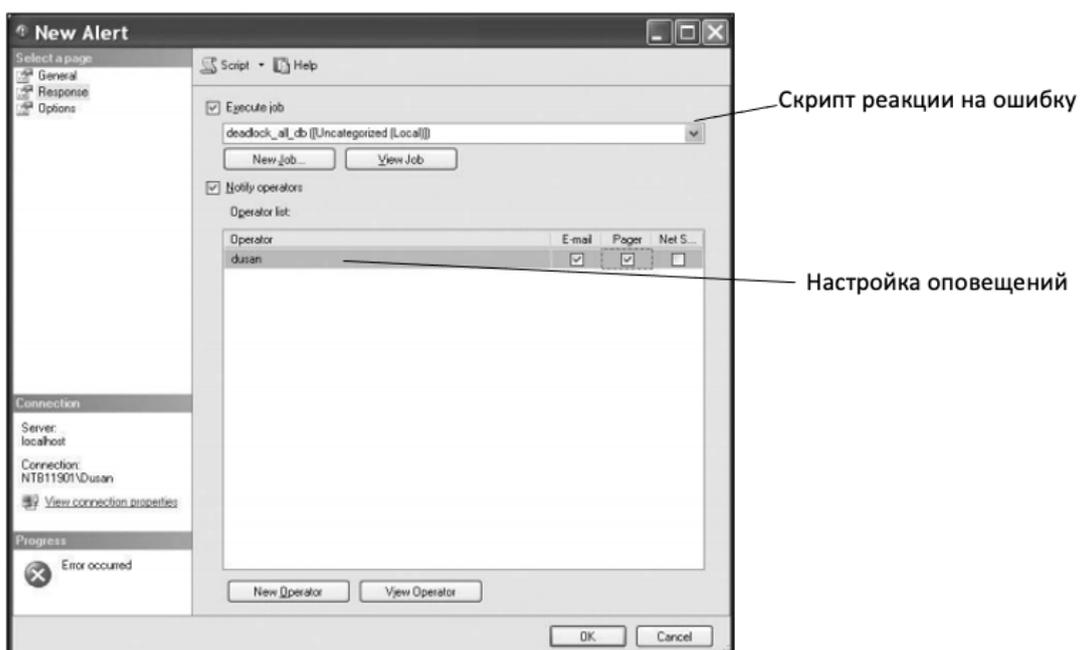


Рисунок 28. Настройка ответа на возникновение ошибки

Вопросы для самостоятельного изучения по итогам лекции.

1. Перечислите все задачи администрирования многопользовательских баз данных, которые на ваш взгляд можно было бы автоматизировать средствами агента?
2. Что такое журнал событий приложений Windows? Каким образом он связан с автоматизацией задач администрирования?
3. С помощью каких средств возможна автоматизация задач администрирования в СУБД Oracle и в СУБД PostgreSQL?

6. МОДЕЛИ БЕЗОПАСНОСТИ МНОГОПОЛЬЗОВАТЕЛЬСКИХ БАЗ ДАННЫХ

6.1. Принципы обеспечения безопасности баз данных. Определения элементов безопасности

Безопасность многопользовательских баз данных, это не только обеспечение непрерывности работы серверов баз данных и доступа к ним, это еще и процедуры, связанные с обеспечением защиты данных и баз данных от несанкционированного доступа.

Приведем *общие принципы обеспечения безопасности СУБД.*

1. Настройка брандмауэра (сетового экрана). При настройке такого программного обеспечения, стоит особое внимание уделить его выбору, а также документации по настройке как самого брандмауэра, так и по настройке СУБД для наиболее корректной и эффективной работы связки. При этом заранее следует планировать меры безопасности в предположении, что брандмауэр был обойден злоумышленником.

2. Самая важная мера безопасности по мнению большинства специалистов по информационной безопасности, это своевременные обновления и установка пакетов исправлений операционной системы, в среде которой работает СУБД и ее компоненты, и пакеты обновлений самой СУБД. Выход нового пакета обновлений – это сигнал для злоумышленника, что компания-производитель ОС или СУБД выявила и закрыла одну или несколько критических уязвимостей, которые злоумышленник будет пытаться искать в не обновлённых версиях программного обеспечения.

3. Использовать только необходимый набор функций операционной системы и СУБД. Неиспользованные функции при возможности удалить или выключить:

- свести к минимуму число поддерживаемых сетевых протоколов;
- удалить системные хранимые процедуры, которые не нужны или не используются;
- по возможности запретить вход в систему по умолчанию и с гостевыми правами;
- не позволять пользователям работать с СУБД в интерактивном режиме (если в этом нет насущной необходимости).

К *мероприятиям*, которые реализуются в ходе укрепления защиты сервера баз данных относятся:

- меры по разработке политики безопасности организации и донесения

ключевых позиций политики до всех сотрудников (в контексте дисциплины не рассматривается);

- меры по обеспечению безопасности сетевого окружения организации, включающие установку, настройку и мониторинг программного и аппаратного обеспечения организации (в контексте дисциплины не рассматривается);

- меры по защите сервера базы данных (определены в соответствии со стандартом безопасности серии NIST):

- не позволять никому из пользователей работать за компьютером, на котором работает СУБД;

- компьютер, на котором располагается СУБД, должен находиться в помещении, запираемом на замок;

- все визиты в помещение, где находится компьютер с работающей СУБД, должны записываться в журнал;

- меры, связанные с управлением учетными записями и ролями пользователей баз данных (будут рассмотрены подробно).

Пользователем базы данных считается человек или программное средство, успешно прошедшие проверку по логину, сертификату, асимметричному ключу или любому иному способу аутентификации, наделяемые соответствующими правами работы с данными (процедура авторизации). Политика управления пользователями баз данных также регламентируется рядом требований:

- использовать для СУБД учетную запись операционной системы с наименьшими возможными привилегиями (учетные записи операционных систем, как правило, защищены значительно сильнее, чем учетные записи программного обеспечения, в том числе и СУБД);

- защищать учетные записи базы данных сильными паролями (сильными паролями считается произвольный набор прописных и заглавных букв, цифр и разрешенных символов, длиной не менее 12-ти знаков, изменяемые и обновляемые с периодичностью 2-3 недели);

- отслеживать неудачные попытки входа в систему;

- регулярно проверять членство в группах и роли (информация о ролях и группах сохраняется в системных представлениях СУБД);

- проверять учетные записи без паролей (если это возможно – вообще избавиться от них);

- назначать учетным записям наименьшие возможные привилегии (доступ только к тому, что реально требуется этой учетной записи для работы);

- ограничивать привилегии учетной записи администратора базы данных (в случае наличия в организации нескольких администраторов баз данных – четкая

дифференциация их функций: управление файлами данных, управление пользователями, управление структурой хранения данных и т. д.).

6.2. Документация и скрипты модели безопасности баз данных

Далее, рассмотрим базовые процессы создания модели безопасности в организации. Ключевыми являются два документа – непосредственно Модель безопасности и Таблица требований к ролям и пользователям.

Модель безопасности базы данных – документ, классифицирующий и связывающий актуальных пользователей и роли, в которых они участвуют, с объектами базы данных, к которым они имеют доступ в рамках требуемых им для работы полномочий. Записи в модели приводятся в виде бизнес-требований к модели, слабо формализованным языком. Образец модели безопасности показан на рис. 29.

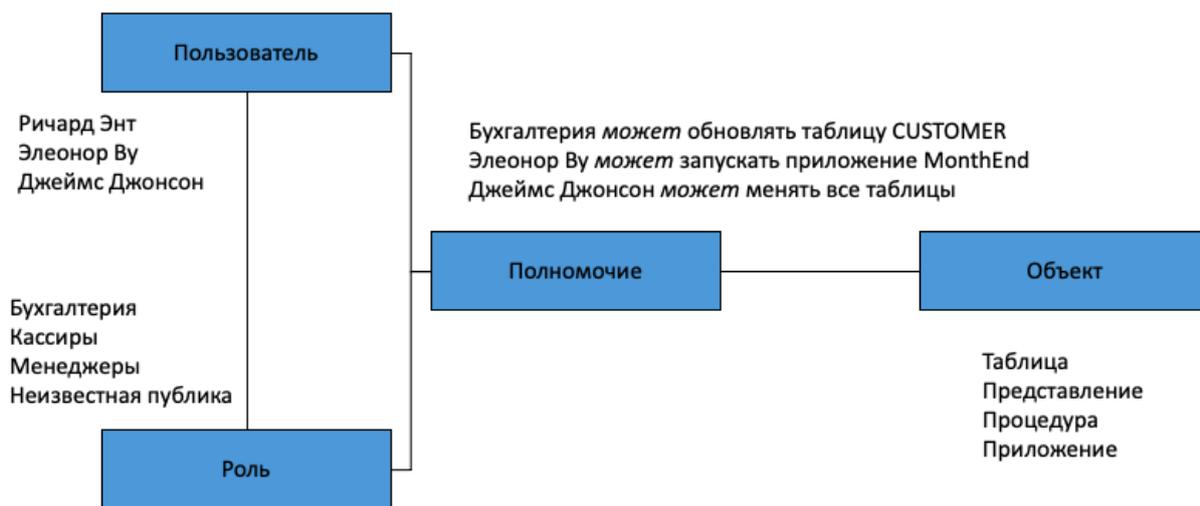


Рисунок 29. Фрагмент схемы “Модель безопасности базы данных”

Таблица требований к ролям и пользователям – двухмерная таблица, написанная формальным языком. По одной оси откладываются роли и пользователи базы данных, по другой оси – элементы базы данных. На пересечениях фиксируется наличие или отсутствие прав доступа пользователя к объекту. При наличии этих прав, они конкретизируются, рис. 30.

	CUSTOMER	TRANSACTION	WORK	ARTIST
Продавцы	Вставка, изменение, запрос	Изменение, запрос	Запрос	Запрос
Менеджеры	Вставка, изменение, запрос	Вставка, изменение, запрос	Вставка, изменение, запрос	Вставка, изменение, запрос
Системный администратор	Вставка, изменение, запрос, удаление, предоставление прав, модификация структуры	Вставка, изменение, запрос, удаление, предоставление прав, модификация структуры	Вставка, изменение, запрос, удаление, предоставление прав, модификация структуры	Вставка, изменение, запрос, удаление, предоставление прав, модификация структуры

Рисунок 30. Фрагмент документа “Требования к ролям и пользователям базы данных”

После составления Модели и Таблицы, администратор создает и настраивает учетные записи пользователей базы данных (в примерах ниже рассмотрена структура учетных записей пользователей СУБД MS SQL Server 2018).

Для Учетной записи пользователя, администратору необходимо создать два элемента базы данных – логин (LOGIN, для процедуры аутентификации) и пользователя (USER, для процедуры авторизации).

Логин пользователя (LOGIN) – элемент базы данных, который используется СУБД при процедуре определения “является ли пользователь действительно тем, кем он представляется”. Эта процедура называется аутентификацией, и, в случае успешного ее прохождения, пользователь получает возможность соединиться с сервером баз данных. Логин может быть создан как для пользователя операционной системы MS Windows, так и для пользователя самой СУБД. Скрипты для двух вариантов:

```
CREATE LOGIN [база данных\имя пользователя в ОС] FROM WINDOWS;
```

```
CREATE LOGIN /Имя пользователя/ WITH PASSWORD = 'пароль пользователя';
```

Пользователь базы данных (USER) – элемент базы данных, добавляется к существующему LOGIN. Наделяет пользователя, успешно прошедшего аутентификацию соответствующими ему правами доступа к другим элементам базы данных. Эта процедура называется авторизацией.

```
CREATE USER /Имя пользователя/ [FOR {LOGIN...}, {CERTIFICATE...}, {ASYMMETRIC_KEY}] [WITH  
DEFAULT_SCHEMA = schema_name]
```

В контексте пользователя упоминается еще один важный элемент модели безопасности СУБД. *Схема* – группа логически объединенных элементов базы данных, принадлежащих одному из пользователей или ролей базы данных. Схемы делятся на системные (схемы по умолчанию) и пользовательские. Перечислим *системные схемы* MS SQL Server:

- guest, минимально возможные права доступа к элементам базы данных, для обеспечения работы в базе данных администратор должен выдать дополнительные разрешения;
- dbo, владелец базы данных, полный доступ ко всем элементам ядра базы данных. Пользователь, осуществивший релиз СУБД получает эти права по умолчанию;
- INFORMATION_SCHEMA, доступ к метаданным всех объектов баз данных;
- Sys, доступ к системным представлениям (например, просмотр системных каталогов).

Пользовательские схемы базы данных создаются администратором в случае необходимости. Скрипт создания схемы – это атомарная процедура, состоящая из инструкций создания таблиц, представлений и определения прав доступа к ним.

```
CREATE SCHEMA /Имя схемы/ AUTHORIZATION [/Имя пользователя/]
```

Сгруппированные вследствие одинаковых прав доступа к одинаковым элементам или схемам базы данных пользователи формируют Роли.

Роль - фиксированная системная или пользовательская группа пользователей или приложений, имеющих одинаковые права на действия в рамках схем баз данных. Выделяют серверные роли (права на уровне сервера) и роли баз данных (права на уровне баз данных). Описание серверных ролей приведено в табл. 6, описание ролей баз данных приведено в табл. 7.

Таблица 6. Серверные роли.

Sysadmin	Выполняет любые действия в СУБД (учетная запись sa)
Serveradmin	Конфигурирует экземпляры серверов
Setupadmin	Управление репликациями и процедурами
Securityadmin	Управляет учетными записями, мониторит отчетность сервера
Processadmin	Управление системными процессами
Dbcreator	Создает и изменяет базы данных
Discadmin	Управляет файлами

Таблица 7. Роли баз данных

Db_owner	Доступ к функциям БД
Db_accessadmin	Возможность добавлять и удалять пользователей
Db_datareader	Возможность просматривать данные в БД
Db_datawriter	Возможность добавлять, модифицировать, удалять данные в БД
Db_denydatareader	Невозможность просматривать данные в БД
Db_denydatawriter	Невозможность добавлять, модифицировать, удалять данные в БД
Public	Полномочия по умолчанию

Создание пользовательской серверной роли в случае необходимости осуществляется выполнением инструкции:

```
CREATE SERVER ROLE /Имя роли/;
```

Для добавления и удаления пользователя в роли используется инструкция:

```
ALTER SERVER ROLE /Имя роли/ ADD (DROP) MEMBER /Имя пользователя/;
```

После успешного прохождения авторизации, пользователь получает права доступа к объектам базы данных, в соответствии с определением требований (рис. 30). Для предоставления полномочий доступа пользователям используется инструкция GRANT:

GRANT [ALL или /перечисление полномочий через запятую/] ON /Имя объекта базы данных, к которому дается доступ/ TO /имя или имена пользователей, наделяемых полномочиями, через запятую/.

Ниже разберем наиболее часто определяемые полномочия.

- SELECT, возможность считывать строки, можно ограничить это полномочие задав список выводимых столбцов; применяется к таблицам, представлениям, функциям;
- INSERT, возможность добавлять строки; применяется к таблицам и представлениям;
- UPDATE, возможность изменять значения столбцов, можно указать список столбцов через запятую; применяется к таблицам и представлениям;
- DELETE, возможность удалять строки; применяется к таблицам и представлениям;
- EXECUTE, возможность запустить на исполнение пользователем хранимую процедуру или функцию; применяется к хранимым процедурам и функциям;

- ALTER, возможность изменять свойства объекта базы данных; применяется к таблицам, представлениям, хранимым процедурам, функциям.

Для блокировки доступа пользователя к объекту (включая ранее выданные полномочия) используется инструкция DENY.

DENY [ALL или /перечисление полномочий через запятую/] ON /Имя объекта базы данных, к которому блокируется доступ/ TO /имя или имена пользователей, для которых блокируются полномочия, через запятую/.

Для удаления ранее выданных полномочий пользователя к объекту используется инструкция REVOKE. REVOKE снимает как «положительные» полномочия GRANT, так и «отрицательные» полномочия DENY.

REVOKE [ALL или /перечисление полномочий через запятую/] ON /Имя объекта базы данных, для которого удаляются полномочия/ TO /имя или имена пользователей, для которых удаляются полномочия, через запятую/.

Более подробно инструкции управления полномочиями и документация, сопровождающая процесс будут рассмотрены в практических работах в рамках данного курса.

6.3. Элементы безопасности и скрипты MongoDB

Рассмотрим основы процедуры создания пользователей в noSQL СУБД на примере MongoDB. В отличие от реляционных СУБД, создание LOGIN, как инструмента аутентификации не требуется, так как аутентификации и авторизация будет происходить на уровне Пользователя (USER). В общем случае, пользователи создаются для конкретной базы данных, этот факт необходимо учитывать в контексте того, что в системе могут находиться разные пользователи с одинаковыми именами (созданные для разных баз данных). Новый пользователь создается следующим типовым скриптом:

```
{user: "<имя пользователя>",  
  pwd: "<пароль пользователя>",  
  customData: {любая дополнительная информация о пользователе},  
  roles: {role: "<имя роли>", db: "<название базы данных>"}  
}
```

Как и все инструкции MongoDB, скрипт написан на языке JSON. Произвольная информация из customData необязательна, но очень желательна. Учитывая специфику применения noSQL модели хранения данных, пользователей у базы может быть очень много, и дополнительная поясняющая

информация может существенно облегчить труд администратора. Далее, приведем встроенные (built-in) роли, которые можно использовать в скрипте создания пользователя.

`read` – дает возможность считывать данные из всех несистемных коллекций и коллекции `system.js` (содержит специальный Java Script код, используемый в приложениях баз данных);

`readWrite` – все полномочия роли `read`, а также возможность изменения данных во всех несистемных коллекциях и коллекциях `system.js`;

`dbAdmin` – дает возможность выполнения задач администрирования, таких как: перепроектирование схем базы данных, индексирование, мониторинг статистики. У этой роли нет полномочия управлять пользователями;

`userAdmin` – дает возможность создавать и изменять роли и пользователей для указанной базы данных;

`dbOwner` – комбинация полномочий `readWrite`, `dbAdmin`, `userAdmin`;

Для указания распространения полномочий на все базы данных сервера используются роли: `readAnyDatabase`, `readWriteAnyDatabase`, `userAdminAnyDatabase`, `dbAdminAnyDatabase`.

`backup` – привилегии, необходимые для осуществления резервного копирования баз данных;

`restore` – привилегии, необходимые для осуществления восстановления баз данных из резервной копии;

`root` – привилегии суперпользователя, доступ ко всем процедурам со всеми объектами сервера.

На практике, с процедурой создания, изменения пользователей поSQL СУБД MongoDB можно ознакомиться в ходе выполнения самостоятельной работы по практическим материалам данного курса.

6.4. Вопросы для самостоятельного изучения по итогам лекции

1. Перечислите все возможные полномочия пользователя MS SQL Server (таблица требований к ролям).

2. Что такое CERTIFICATE и ASSYMETRIC_KEY в системе безопасности MS SQL SERVER?

3. Чем USER отличается от LOGIN в MS SQL Server?

4. Чем оператор REVOKE отличается от оператора DENY?

6.5. Тестовые задания для самопроверки

1. Для задания предоставления разрешений пользователю в языке Transact-SQL применяются следующий оператор:

- А) ALLOW
- Б) GRANT
- В) REVOKE
- Г) DENY

2. В типовые задачи автоматизации функций администрирования не входит:

- А) управление репликацией данных
- Б) управление резервным копированием
- В) управление оптимизацией выполнения инструкций
- Г) входит все перечисленное

3. Критическими для процесса функционирования СУБД являются ошибки:

- А) группы 0-10
- Б) группы 11-16
- В) группы 17-18
- Г) группы 19+

4. Что не входит в сообщение об ошибке MS SQL SERVER?

- А) информация о способе решения проблемы
- Б) текст описания ошибки
- В) номер сообщения об ошибке
- Г) все перечисленное является частью сообщения

5. За фиксирование критических ошибок в работе СУБД MS SQL SERVER отвечает...

- А) журнал событий приложений Windows
- Б) приложение баз данных
- В) служба MSSQLSERVER

6. Какой из перечисленных методов не используется при аутентификации MS SQL Server

- А) сертификат
- Б) асимметричный ключ
- В) цифровая подпись

- Г) логин
- Д) используются все перечисленные

7. Наивысшим из перечисленных инструкций приоритетом обладает инструкция...

- А) GRANT
- Б) DENY
- В) REVOKE

8. Что из перечисленного не относится к фиксированной роли БД?

- А) db_user
- Б) db_owner
- В) public
- Г) db_datereader
- Д) относятся все перечисленные

9. Что из перечисленного не относится к серверным ролям БД?

- А) sysadmin
- Б) setupadmin
- В) dbcreator
- Г) user
- Д) относятся все перечисленные

7. ВРЕДОНОСНЫЕ АТАКИ НА МНОГОПОЛЬЗОВАТЕЛЬСКИЕ БАЗЫ ДАННЫХ

7.1. Вредоносные атаки на серверы баз данных

Поскольку именно на серверах баз данных организации содержится самая чувствительная для организации информация, именно они часто подвергаются вредоносным атакам со стороны злоумышленников. Также, этим частым атакам способствует то, что настроенная по умолчанию или неправильно настроенная СУБД очень уязвима даже к самым простым в реализации типам атак. Результатом этих атак, как правило, является прямое изменение и порча данных, перехват управления сервером баз данных с целью требования выкупа, или просто уничтожение всех данных и нарушение работы сервера.

Можно выделить три основных вида атак на серверы баз данных: изучение cookies в пользовательских приложениях баз данных, атаки blind-перебором и поиск уязвимостей СУБД с помощью SQL инъекций.

Атака с модификацией cookies нацелена на перехват сеанса взаимодействия пользователя с сервером базы данных, с целью получить конфиденциальную информацию. *Cookie* - небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя приложения баз данных. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

Очевидно, что доступная информация о сеансах пользователя или об аутентификации может быть очень интересна для злоумышленника, задумавшего получить доступ к информации, хранящейся в базе данных.

Для понимания сути атаки необходимо разобраться с принципом работы cookie-файлов. Ниже приведен небольшой пример обмена данными сервера с пользовательским приложением. Подробная информация о принципах работы HTTP приводится в курсе дисциплины «Управление данными».

1. Получив команду пользователя, браузер отправляет веб-серверу короткий текст с HTTP-запросом. Например, для доступа к странице <http://www.example.org/index.html>, браузер отправляет на сервер

www.example.org следующий запрос: GET /index.html HTTP/1.1 HOST: www.example.org

2. Предположим, что это первое посещение пользователем указанного URL. Сервер отвечает, отправляя запрашиваемую страницу, с дополнительным указанием браузеру сохранить файл cookie со следующим содержимым: HTTP:/1.1 200 ok Content-type: text/html SET-cookie: name=value (содержимое страницы). В последствие, это существенно облегчит браузеру загрузку этой страницы, при любом повторном обращении.

3. При следующем обращении к серверу, браузер прикрепит этот, сгенерированный на сервере файл cookie к сформированному запросу, вот так: GET /index.html HTTP/1.1 HOST: www.example.org cookie: name=value Accept: /

Такое изначально направленное на извлечение максимального юзабилити при работе пользователя с веб-приложениями и сервером баз данных взаимодействие может стать причиной утечки приватной и чувствительной информации.

Приведем пример использования подмены файла cookies при вредоносных атаках на сервер. Для этого продолжим взаимодействие, которое было показано выше.

4. Сервер отвечает, отправляя запрашиваемую страницу и, возможно, добавив новые значения в файл cookie, после переслав его на сторону пользователя.

5. Предположим, что набор значений, хранимых в файле cookie изменяется путем добавления новых строк SET- cookie: name=newvalue.

6. С момента добавления в файл cookie значения, которое может быть изменено, этот файл становится уязвимым. Дело в том, что значения файлов cookie могут устанавливаться скриптами, написанными на языке JavaScript, встроенными в текст страниц, или аналогичными скриптами, работающими в браузере. В JavaScript для этого используется объект document.cookie. Например, действие скрипта с аргументом document.cookie = "temperature=20" создаст запись cookie под именем «temperature» и присвоит ей значение 20.

7. Перехватив в ходе сеанса обмена данными между приложением и сервером cookie, злоумышленник в списке параметров cookie потенциально может найти возможность для осуществления атаки. Значение изменяется, перехваченный и измененный файл cookie отправляется злоумышленником дальше на сервер. Например, такое изменение неудачного параметра cookie файла позволит злоумышленнику завладеть правами администратора сервера: isAdmin [0] -> [1].

Смысл атаки *blind-перебором* – с помощью упорядоченного набора SQL запросов к серверу, основываясь на реакции СУБД на эти запросы, восстановить структуру таблиц с чувствительной информацией (см. alerts в Лекции 5). Оставленные без настройки реакции СУБД на запросы дают очень много информации не только рядовым пользователям, но и злоумышленникам. В общем виде, атака blind-перебором выглядит следующим образом.

1. Выяснение названия и версии СУБД. Это уже может подсказать злоумышленнику приблизительную структуру системных и ключевых таблиц базы данных.

2. Определение одной или нескольких интересных для проникновения таблиц в базе данных. Как правило – это списки пользователей, их паролей. Например, таблицы admins, users, policy, rights и т.д. могут представлять особенный интерес.

3. После выявления интересных таблиц происходит последовательное восстановление метаинформации о структуре таблицы: определение количества столбцов, определение типа данных столбцов и их названия, последовательности их в структуре таблицы.

4. После выявления структуры таблицы, получив права на вставку в нее новых значений, злоумышленник может внести строку со значениями своего пользователя с максимальным доступом к функциям базы данных.

Атаки с использованием *SQL-инъекций* осуществляются с помощью динамических (встроенных) SQL инструкций, которые могут быть введены на стороне пользователя в незащищенную веб-форму, принимающую SQL параметры, или даже в строку URL, соединяющую пользователя с целевым сервером. По причине своей простоты и эффективности, случаи с SQL инъекциями являются наиболее распространенными среди всех вредоносных атак (по данным сайта OWASP, это около 80% случаев). Приведем пример случая с такой атакой. Предположим, что пользовательское приложение включает в себя функцию аутентификации через html-форму, скрипт которой выглядит следующим образом:

```
<form action='index.php' method='post'>
<input type='email' name='user_email' />
<input type='password' name='usr_pass' />
<input type='checkbox' name='always logged' value='always logged' />
<input type='submit' value='submit' />
</form>
```

После ввода данных со стороны пользователя, в СУБД отправляется запрос типа:

```
SELECT * FROM users WHERE email = $_POST['email'] AND password = md5($_POST['password']);
```

Если результатом этого запроса является существующая строка с Email и password (хешированным md5), то пользователь успешно проходит аутентификацию на сервере.

Зная структуру этого запроса к СУБД злоумышленнику достаточно с помощью динамического SQL заменить неизвестные ему параметры почты и пароля пользователя на условие, которое в случае выполнения им на сервере выведет все значения из целевой таблицы с адресами почты и паролями пользователей. Как правило, это условие инструкции SQL, исполняемое в любом случае, например, для нашего случая `xxx@xxx.xxx' OR 1 = 1 LIMIT 1 -- ']`.

Учитывая значительное количество атак инъекциями, уже давно были сформированы базовые правила защиты веб приложений и серверов от этого вида воздействия:

- настройка параметризованного запроса с *плейсхолдерами*, то есть запроса с переменными, которые жестко прописаны в скрипте с кодом и не меняются в зависимости от данных;
- применение *escaping-функций*, заложенных в языки программирования, которая заменит потенциально опасные SQL символы на безопасные escape-последовательности. Например, в языке веб-приложений PHP для СУБД MySQL это функция `mysql_real_escape_string`;
- использование хранимых процедур для инкапсуляции всех SQL-запросов. Пользователь лишь передает входные данные, обрабатываются и передаются на сервер они уже по итогам работы процедуры. По своей логике этот способ похож на параметризованный запрос с плейсхолдерами;
- использование в коде форм приложения *регулярных выражений* с целью обнаружения потенциально вредоносного кода и его удаления или изменения перед передачей этого кода в СУБД;
- создание и постоянный мониторинг прав доступа на подключение к базе данных. Учетным записям, которые используются для подключения к базе данных, должны предоставляться только необходимые права доступа. Подробно этот элемент безопасности рассмотрен в Лекции 6.
- модификация сообщений об ошибках на стороне СУБД. Они не должны раскрывать конфиденциальную информацию. Разумно использовать простые пользовательские сообщения об ошибках, такие как «*Извините,*

возникла техническая ошибка. Служба поддержки уже уведомлена о ней. Повторите попытку позже».

7.2. Управление сетевой безопасностью серверов баз данных

Действия, направленные на мониторинг безопасного состояния серверов баз данных, можно представить в виде чек-листа администратора по безопасности. Ниже перечислены элементы контроля, которые должны регулярно подвергаться мониторингу администратором.

- определение и постоянный мониторинг настроек удаленного доступа к серверу баз данных;
- настройка и мониторинг контроля доступа на стороне хоста (сервера баз данных);
- управление полномочиями на уровне экземпляра сервера базы данных (пользователи, роли, создание баз данных, логин и репликация);
- управление полномочиями на уровне баз данных (соединение, создание схем и т. д.);
- управление полномочиями на уровне схем баз данных (использование схем, создание объектов внутри схем);
- управление полномочиями на уровне таблиц базы данных (выборки, добавление значений, изменение значений и т. д.);
- управление полномочиями на уровне столбцов (предоставление и изъятие доступа к столбцам);
- RLS (ограниченный доступ к строкам).

Большинство перечисленных в чек-листе функций были описаны в материале Лекции 6 и сопутствующих ей практических занятий, поэтому обратим внимание лишь на первые два пункта. Рассмотрим настройки удаленного доступа к серверу на примере СУБД PostgreSQL.

Настройки доступа PostgreSQL изменяются при помощи командной консоли, путем изменения параметров соответствующих конфигурационных файлов. Для выполнения поставленной задачи понадобятся конфигурационные файлы `postgresql.conf` и `pg_hba.conf`.

Конфигурационный файл `postgresql.conf` используется для базовой настройки удаленного доступа к серверу. Важно отметить, что изменения этого файла возможны в ходе работы сервера, но изменения будут приняты только после его перезагрузки. Ключевые параметры файла:

`#listen_addresses` – адрес или адреса серверов баз данных;

`#port` – номер порта или номера группы портов, открытые для

взаимодействия с сервером базы данных;

`#max_connections` – максимально допустимое количество одновременных сеансов работы с сервером;

`#superuser_reserved_connections` – количество сеансов, зарезервированных для суперпользователей (пользователи, имеющие полный доступ ко всем объектам и функциям базы данных);

`#ssl` – параметр, включающий/выключающий использование сертификатов SSL для безопасного зашифрованного обмена между сервером и клиентом. При включении этой функции, для обеспечения работы шифрования, необходимо создать файл с сертификатом сервера (server certificate) и приватным ключом сервера (server private key) и разместить их в папке сервера `$PGDATA`. После указания пути к этим файлам в конфигурационном файле и перезагрузки, заработает SSL шифрование.

Конфигурационный файл `pg_hba.conf` устанавливает условия доступа к серверу баз данных. В этом файле содержатся строки с описанием возможных вариантов установления соединения с сервером. Строка выглядит следующим образом: `# /правило/ /имя база или базы данных/ /имя или имена пользователей/ /адрес и порт, доступные для соединения/ /метод соединения/`. Рассмотрим варианты параметров строки.

Правила соединения могут быть следующие: `local`, `host`, `hostssl` (доступен обмен данными только с настроенным ssl шифрованием).

Методы соединения могут быть следующие: `trust` (без пароля), `reject` (соединение в любом случае будет отклонено), `md5 password` (пароль для соединения хеширован md5), `ident` (идентификация на уровне операционной системы). Ниже приведем пример нескольких строк конфигурационного файла с указанными параметрами:

```
#IPv4 local connections:
```

```
host all all 127.0.0.1/32 trust
```

Доверять всем пользователям доступ ко всем базам данных по адресу 127.0.0.1 и номеру порта 32.

7.3. Вопросы для самостоятельного изучения по итогам лекции

1. Что такое GitHub? Для чего используется этот ресурс? Почему он является уязвимостью для СУБД?
2. Что такое HTTP-форма и как в общем виде формируется браузером HTTP-запрос?
3. Перечислите, какие бывают типы файлов cookie.

Ответы на тестовые задания.

Лекция 1.

1.Б. 2.А 3.А 4.В 5.А 6.А 7.Б 8.Д 9.Б 10.А 11.В 12.В 13.Б 14.Г 15.В 16.Г.

Лекция 2.

1.В 2.В 3.А 4.Г 5.А 6.А 7.Б 8.В 9.А 10.А

Лекция 3.

1.Г 2.Б 3.А 4.А 5.Г 6.В 7.А 8.А 9.Б 10.В

Лекция 4.

1.А 2.Г 3.А 4.А 5.Г 6.В 7.А 8.Г

Лекция 6.

1.Б 2.Г 3.Г 4.А 5.А 6.В 7.Б 8.А 9.Г

Список литературы

1. Бегг К., Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Пер. с англ. - М.: Вильямс, 2017. - 1440 с.: ил..
2. Kroenke D.M., Auer D.J. Database Processing: Fundamentals, Design and Implementation. PEARSON, 2019. - 691 с.: ил..
3. Кренке Д. Теория и практика построения баз данных - Спб.: Питер, 2005. - 859 с.: ил..
4. Кузнецов С. Базы данных: Модели и языки. - Мск.: Бином, 2008. - 720 с.: ил..
5. Kroenke D.M., Auer D.J. Database Concepts. англ. - PEARSON, 2018. - 579 с.: ил..
6. Моргунов Е.П. PostgreSQL. Основы языка SQL. - Спб.: БХВ-Петербург, 2021. - 335 с.:ил..
7. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих. - Спб.: БХВ-Петербург, 2013. - 817 с.: ил..
8. Мартишин С.А., Симонов В.Л., Храпченко В.Л. Базы данных. Практическое применение СУБД SQL и noSQL типа для проектирования информационных систем. - Мск.: ИНФРА-М, 2016. - 367 с.: ил..
9. Schonig H-J. Mastering PostgreSQL 13. - Packt, 2020. - 458 с.: ил..



Сведения об авторе

Смирнов Михаил Вячеславович, кандидат экономических наук, доцент кафедры “Предметно-ориентированные информационные системы” Института комплексной безопасности и специального приборостроения РТУ-МИРЭА.