Базы данных. Экзамен.

2019

Содержание

1 Инструкции языка описания данных, инструкции языка обработки данных, инструкции безопасности, инструкции управления транзакциями. . . . 3
2 Реляционная модель данных. Структурная, целостная, манипуляционная части. Реляционная алгебра. Исчисление кортежей . . . 5
3 Семантическое моделирование данных . . . 7
4 Теория проектирования реляционных баз данных. Функциональные зависимости . . . 8
5 Теория проектирования реляционных баз данных. Нормальные формы . . . 10
6 Инструкции управления потоком . . . 12
7 Системы типов данных языка SQL . . . 14
8 Скалярные выражения . . . 15
9 Приведение и преобразование типов данных . . . 16
10 Условие поиска . . . 17
11 Сценарии и пакеты . . . 18
12 Массовый импорт и экспорт данных . . . 20
13 Представления, функции, хранимые триггеры (DDL, DML), курсоры . . . 21
14 Метаданные и доступ к ним . . . 22
15 Методы физического хранения данных на диске . . . 23
16 Индексы . . . 24
17 Управление транзакциями . . . 26
18 Неблагоприятные эффекты, вызванные параллельным выполнением транзакций, и их устранение 27
19 Уровни изоляции транзакций и блокировки . . . 28
20 Гранулярность блокировок, иерархии блокировок и эскалация блокировок . . . 29
21 Взаимоблокировки, их обнаружение и устранение . . . 30
22 Режимы блокировок и совместимость блокировок . . . 31
23 Обработка ошибок в транзакциях . . . 32
24 Логическая и физическая архитектуры журнала транзакций . . . 33
25 Контрольные точки, активная часть журнала и усечение журнала транзакций . . . 34
26 Управление участниками системы безопасности . . . 35
27 Управление разрешениями . . . 36

1 Инструкции языка описания данных, инструкции языка обработки данных, инструкции безопасности, инструкции управления транзакциями.

1.1 Инструкции языка описания данных (Data Definition Language, DDL)

- `CREATE` создает объект БД (саму базу, таблицу, представление, пользователя и т. д.)
- `ALTER` изменяет объект
- `DROP` удаляет объект
- `ENABLE TRIGGER` включает триггер DML, DDL или logon
- `DISABLE TRIGGER` отключает триггер
- `TRUNCATE TABLE` удаляет все строки в таблице, не записывая в журнал удаление отдельных строк
- `UPDATE STATISTICS` обновляет статистику оптимизации запросов для таблицы или индексированного представления

1.2 Инструкции языка обработки данных (Data Manipulation Language, DML)

- `SELECT` считывает данные, удовлетворяющие заданным условиям
- `INSERT` добавляет новые данные
- `UPDATE` изменяет существующие данные
- `DELETE` удаляет данные
- `MERGE` выполняет операции вставки, обновления или удаления для целевой таблицы на основе результатов соединения с исходной таблицей
- `BULK INSERT` выполняет импорт файла данных в таблицу или представление базы данных в формате, указанном пользователем
- `READTEXT` считывает значения text, ntext или image из столбцов типа text, ntext или image начиная с указанной позиции
- `WRITETEXT` обновляет и заменяет все поле text, ntext или image
- `UPDATETEXT` обновляет часть поля text, ntext или image

1.3 Инструкции безопасности (ранее инструкции языка доступа к данным - Data Control Language, DCL)

- `GRANT` предоставляет пользователю разрешения на определенные операции с объектом
- `REVOKE` отзывает ранее выданные разрешения
- `DENY` задает запрет, имеющий приоритет над разрешением
- `ADD SIGNATURE` добавляет цифровую подпись для хранимой процедуры, функции, сборки или триггера
- `OPEN MASTER KEY` открывает главный ключ в текущей базе данных
- `CLOSE MASTER KEY` закрывает главный ключ в текущей базе данных
- `OPEN SYMMETRIC KEY` расшифровывает симметричный ключ и делает его доступным для использования
- `CLOSE SYMMETRIC KEY` закрывает симметричный ключ или все симметричные ключи, открытые в текущем сеансе
- `EXECUTE AS` контекст выполнения сеанса переключается на заданное имя входа и имя пользователя
- `REVERT` переключает контекст выполнения в контекст участника, вызывавшего последнюю инструкцию EXECUTE AS
- `SETUSER` позволяет члену предопределенной роли сервера sysadmin или члену предопределенной роли базы данных db_owner олицетворять другого пользователя

1.4 Инструкции управления транзакциями (Transaction Control Language, TCL)

- `BEGIN DISTRIBUTED TRANSACTION` запускает распределенную транзакцию, управляемую координатором распределенных транзакций
- `BEGIN TRANSACTION` отмечает начальную точку явной локальной транзакции
- `COMMIT TRANSACTION` отмечает успешное завершение явной или неявной транзакции
- `COMMIT WORK` действует так же, как и инструкция COMMIT TRANSACTION
- `ROLLBACK TRANSACTION` откатывает явные или неявные транзакции до начала или до точки сохранения транзакции
- `ROLLBACK WORK` действует так же, как и инструкция ROLLBACK TRANSACTION
- `SAVE TRANSACTION` устанавливает точку сохранения внутри транзакции

2 Реляционная модель данных. Структурная, целостная, манипуляционная части. Реляционная алгебра. Исчисление кортежей

2.1 Реляционная модель

Реляционная модель данных включает следующие компоненты:

- Структурный (данные в базе данных представляют собой набор отношений),
- Целостностный (отношения (таблицы) отвечают определенным условиям целостности),
- Манипуляционный (манипулирования отношениями осуществляется средствами реляционной алгебры и/или реляционного исчисления).

Кроме того, в состав реляционной модели данных включают теорию нормализации.

Структурная часть реляционной модели

Структурная часть реляционной модели описывает, из каких объектов состоит реляционная модель. Постулируется, что основной структурой данных, используемой в реляционной модели, являются нормализованные «n-арные» отношения. Основными понятиями структурной части реляционной модели являются

- тип данных,
- домен(подмножество значений некоторого типа данных),
- атрибут (пара вида <имя_атрибута, имя_домена >),
- схема отношения (именованное множество упорядоченных пар <имя_атрибута, имя_домена>),
- схема базы данных(множество именованных схем отношений),
- кортеж (множество упорядоченных пар <имя_атрибута, значение_атрибута>, которое содержит одно вхождение каждого имени атрибут),
- отношение (содержит две части: заголовок (схему отношения) и тело (множество из m кортежей)),
- потенциальный, первичный и альтернативные ключи,
- реляционная база данных (набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных).

Целостностная часть реляционной модели

В целостностной части реляционной модели фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и ссылочная целостность (или целостность внешних ключей).

Поддержание целостности сущностей обеспечивается средствами СУБД. Это осуществляется с помощью двух ограничений:

1. при добавлении записей в таблицу проверяется уникальность их первичных ключей,
2. не позволяется изменение значений атрибутов, входящих в первичный ключ.

Внешний ключ в отношении R2 – это непустое подмножество множества атрибутов FK этого отношения, такое, что:

- существует отношение R1 (причем отношения R1 и R2 необязательно различны) с потенциальным ключом CK;
- каждое значение внешнего ключа FK в текущем значении отношения R2 обязательно совпадает со значением ключа CK некоторого кортежа в текущем значении отношения R1.

Требование ссылочной целостности состоит в следующем: для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

Манипуляционная часть реляционной модели

Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными – реляционную алгебру и реляционное исчисление.

Реляционная алгебра

Реляционная алгебра является основным компонентом реляционной модели и состоит из восьми операторов, составляющих две группы по четыре оператора:

1. Традиционные операции над множествами: объединение (UNION), пересечение (INTERSECT), разность (MINUS) и декартово произведение (TIMES).

2. Специальные реляционные операции: ограничение (WHERE) , проекция (PROJECT), соединение (JOIN) и деление (DIVIDE BY).

А также:

- переименование (RENAME) ,
- SEMIJOIN (полусоединение),
- SEMIMINUS (полувычитание),
- EXTEND (расширение), SUMMARIZE (обобщение)
- TCLOSE (транзитивное замыкание),
- GROUP (группировка),
- COUNT | SUM | AVG | MAX | MIN | ALL | ANY | COUNTD | SUMD | AVGD

Результат выполнения любой операции реляционной алгебры над отношениями также является отношением.

Исчисление кортежей

Реляционное исчисление является альтернативой реляционной алгебре. Внешне два подхода очень отличаются – исчисление описательное, а алгебра предписывающая, но на более низком уровне они представляют собой одно и то же, поскольку любые выражения исчисления могут быть преобразованы в семантически эквивалентные выражения в алгебре и наоборот.

Упрощенный синтаксис выражений исчисления кортежей в форме БНФ имеет вид.

```
объявление-кортежной-переменной ::=
    RANGE OF переменная IS список-областей
область ::= отношение |
    реляционное-выражение
реляционное-выражение ::=
    (список-целевых-элементов)[WHERE wff]
целевой-элемент ::=
    переменная |
    переменная.атрибут [AS атрибут]
wff ::=
    условие |
    NOT wff |
    условие AND wff |
    условие OR wff |
    IF условие THEN wff |
    EXISTS переменная (wff) |
    FORALL переменная (wff) |
    (wff)
условие ::=
    (wff) |
    компаранд операция-отношения компаранд
```

По приведенной грамматике можно сделать следующие замечания.

1. Квадратные скобки здесь указывают на компоненты, которые по умолчанию могут быть опущены.
2. Категории отношение, атрибут и переменная – это идентификаторы (т. е. имена).
3. Реляционное выражение содержит заключенный в скобки список целевых элементов и выражение WHERE, содержащее формулу wff («правильно построенную формулу»). Такая формула wff составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, констант, операторов сравнения, логических (булевых) операторов и скобок. Каждая свободная переменная, которая встречается в формуле wff, должна быть также перечислена в списке целевых элементов.
4. Категория условие представляет или формулу wff, заключенную в скобки, или простое скалярное сравнение, где каждый компаранд оператора сравнения – это либо скалярная константа, либо значение атрибута в форме переменная.атрибут.

3 Семантическое моделирование данных

Любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части.

Наиболее известным представителем класса семантических моделей предметной области является модель **«сущность-связь» или ER-модель**. Модель сущность-связь – модель данных, позволяющая описывать концептуальные схемы предметной области. Предметная область – часть реального мира, рассматриваемая в пределах данного контекста.

С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями. Основными понятиями ER-модели являются *сущность, связь и атрибут (свойство).*

Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах). Сущности подразделяются на сильные и слабые. Сильные сущности существуют сами по себе, а существование слабых сущностей зависит от существования сильных.

Связь – это ассоциация, устанавливаемая между сущностями. Эта ассоциация может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Связи в ER-модели могут иметь тип «один к одному», «один ко многим», «многие ко многим». Именно тип связи «многие ко многим» является единственным типом, представляющим истинную связь, поскольку это единственным тип связи, который требует для своего представления отдельного отношения. Связи типа «один к одному» и «один ко многим» всегда могут быть представлены с помощью механизма внешнего ключа, помещаемого в одно из отношений.

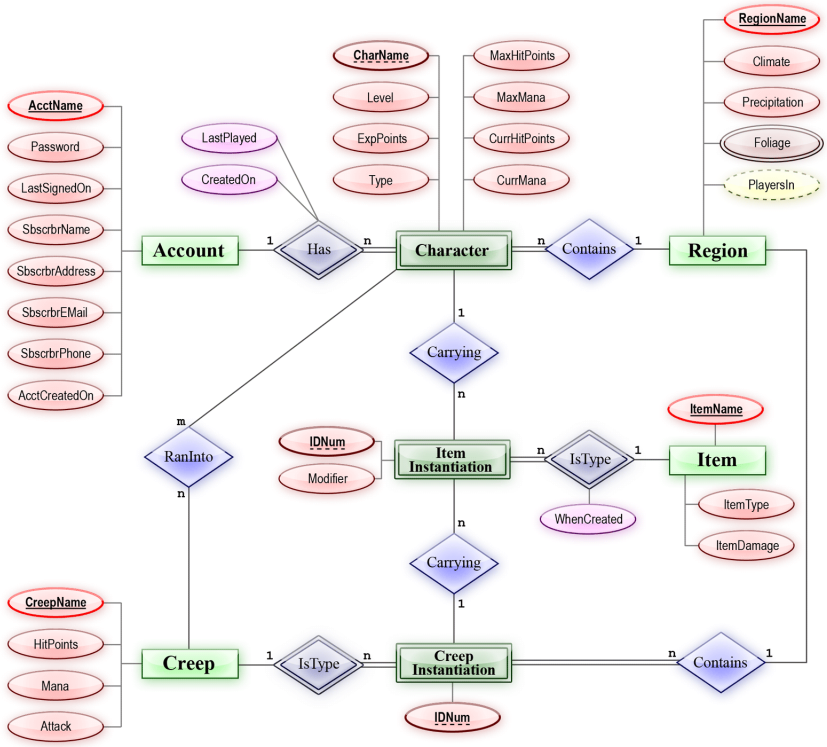
Свойством сущности (и связи) является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности (или связи). Значения свойств каждого типа извлекаются из соответствующего множества значений, которое в реляционной терминологии называется доменом. Свойства могут быть простыми или составными, ключевыми, однозначными или многозначными, опущенными (т. е. «неизвестными» или «непредставленными»), базовыми или производными.

Изображение ER-модели

На ER-диаграммах множества сущностей изображаются в виде прямоугольников, множества отношений изображаются в виде ромбов. Слабый тип сущности изображают в виде прямоугольника с двойным контуром.

Если сущность участвует в отношении, они связаны линией. Вид типа связи обозначается над линиями в виде соответствующих надписей возле типов сущностей. Например, если это вид бинарной связи «один ко многим», то делают надписи 1, n (или m), соответственно, возле соответствующих типов сущностей.

Атрибуты изображаются в виде овалов и связываются линией с одним отношением или с одной сущностью. Именование сущности обычно выражается уникальным существительным, именование связи обычно выражается глаголом, именование атрибута обычно выражается существительным. Неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности, являются ключом сущности.



4 Теория проектирования реляционных баз данных. Функциональные зависимости

4.1 Теория проектирования реляционных баз данных.

Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1НФ или 1NF);
- вторая нормальная форма (2НФ или 2NF);
- третья нормальная форма (3НФ или 3NF);
- нормальная форма Бойса-Кодда (НФБК или BCNF);
- четвертая нормальная форма (4НФ или 4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

Основные свойства нормальных форм такие:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели:

- избежать избыточности хранения данных;
- устранить аномалии обновления отношений.

В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы. Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

4.2 Функциональные зависимости.

Пусть R - это отношение, а X и Y - произвольные подмножества множества атрибутов отношения R. Тогда Y функционально зависимо от X, что в символическом виде записывается как X -> Y $\Longleftrightarrow$ $\forall$значение множества X связано в точности с одним значением множества Y.

Левая и правая стороны ФЗ будут называются **детерминантом и зависимой частью** соответственно.

Очевидным способом сокращения размера множества ФЗ было бы исключение тривиальных зависимостей, т. е. таких, которые не могут не выполняться.

Множество всех ФЗ, которые задаются данным множеством ФЗ S, называется **замыканием** S и обозначается символом S+.

Пусть в перечисленных ниже правилах A, B и C – произвольные подмножества множества атрибутов заданной переменной-отношения R, а символическая запись AB означает { A, B }. Тогда правила вывода определяются следующим образом.

1. Правило рефлексивности: $(B \subseteq A) \Rightarrow (A \rightarrow B)$
2. Правило дополнения: $(A \rightarrow B) \Rightarrow (AC \rightarrow BC)$
3. Правило транзитивности: $(A \rightarrow B) AND (B \rightarrow C) \Rightarrow (A \rightarrow C)$

   Каждое и этих правил может быть непосредственно доказано на основе определения ФЗ. Более того, эти правила являются полными в том смысле, что для заданного множества ФЗ S минимальный набор ФЗ, которые подразумевают все зависимости из множества S, может быть выведен из S на основе этих правил. Они также являются исчерпывающими.

   Из трех описанных выше правил для упрощения задачи практического вычисления замыкания S+ можно вывести несколько дополнительных правил. (Примем, что D - это другое произвольное подмножество множества атрибутов R.)

4. Правило самоопределения: $A \to A$
5. Правило декомпозиции: $(A \to BC) \Rightarrow (A \to B) AND (A \to C)$
6. Правило объединения: $(A \to B) AND (A \to C) \Rightarrow (A \to BC)$
7. Правило композиции: $(A \to B) AND (C \to D) \Rightarrow (AC \to BD)$
8. Общая теорема объединения: $(A \to B) AND (C \to D) \Rightarrow (A(C - B) \to BD)$

Два множества ФЗ S1 и S2 **эквивалентны** тогда и только тогда, когда они являются покрытиями друг для друга, т. е. S1+ = S2+.

Множество ФЗ является **неприводимым** тогда и только тогда, когда оно обладает всеми перечисленными ниже свойствами.

- Каждая ФЗ этого множества имеет одноэлементную правую часть.
- Ни одна ФЗ множества не может быть устранена без изменения замыкания этого множества.
- Ни один атрибут не может быть устранен из левой части любой ФЗ данного множества без изменения замыкания множества.

Алгоритм поиска минимального покрытия F для множества функциональных зависимостей E.
Вход: Множество функциональных зависимостей E.

1. Пусть F: = E.
2. Заменить каждую функциональную зависимость X -> {A1, A2, ..., An} из F на n функциональных зависимостей X -> A1, X -> A2, ..., X -> An.
3. Для каждой функциональной зависимости X -> A из F Для каждого атрибута B, который является элементом X если { {F – {X -> A} } ∪ { (X – {B}) -> A} } эквивалентно F заменить X -> A на (X – {B}) -> A в F.
4. Для каждой оставшейся функциональной зависимости X -> A из F если {F - {X -> A}} эквивалентно F, удалить X -> A из F.

Алгоритм нахождения ключа K схемы отношения R для заданного множество функциональных зависимостей F
Вход: Схема отношения R и множество функциональных зависимостей F на атрибутах R.

1. Пусть K := R.
2. Для каждого атрибута из K { вычислить Closure((K - A), F); если замыкание содержит все атрибуты из R, то установите K: = K - {A} };

5 Теория проектирования реляционных баз данных. Нормальные формы

5.1 Теория проектирования реляционных баз данных.

Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1НФ или 1NF);
- вторая нормальная форма (2НФ или 2NF);
- третья нормальная форма (3НФ или 3NF);
- нормальная форма Бойса-Кодда (НФБК или BCNF);
- четвертая нормальная форма (4НФ или 4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

Основные свойства нормальных форм такие:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели:

- избежать избыточности хранения данных;
- устранить аномалии обновления отношений.

В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы. Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

5.2 Нормальные формы.

1. Первая нормальная форма (1НФ или 1NF).

   Переменная отношения находится **в первой нормальной форме (1НФ)** тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов. **Первая нормальная форма** - это условие, согласно которому каждый компонент каждого кортежа является атомарным значением. В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение.

   Функциональная зависимость R.X -> R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

   Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z -> R.X.

   Неключевым атрибутом называется любой атрибут отношения, не входящий в состав потенциального ключа (в частности, первичного).

   Два или более атрибута **взаимно независимы**, если ни один из этих атрибутов не является функционально зависимым от других.

2. Вторая нормальная форма (2НФ или 2NF).

   Отношение R находится **во второй нормальной форме (2NF)** в том и только в том случае, когда оно находится в 1НФ, и каждый неключевой атрибут полностью зависит от каждого ключа R.

3. Третья нормальная форма (3НФ или 3NF).

   Отношение R находится **в третьей нормальной форме (3НФ)** в том и только в том случае, если оно находится в 2НФ, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

4. Нормальная форма Бойса-Кодда (НФБК или BCNF).

   Детерминант - любой атрибут (или группа атрибутов), от которого полностью функционально зависит некоторый другой атрибут.

   Отношение R находится **в нормальной форме Бойса-Кодда** (НФБК) в том и только в том случае, если каждый детерминант является потенциальным ключом.

5. Четвертая нормальная форма (4НФ или 4NF).

   Это определение можно также сформулировать в следующей эквивалентной форме: переменная-отношение R находится **в 4НФ**, если она находится НФБК и все МЗ в переменной-отношении R фактически представляют собой ФЗ от ее ключей.

6. Пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

   Переменная-отношение R находится **в пятой нормальной форме (5НФ)**, которую иногда иначе называют проекционно-соединительной нормальной формой (ПСНФ), тогда и только тогда, когда каждая нетривиальная ЗС в переменной-отношении R подразумевается ее потенциальными ключами.

   ЗС *{ A, B, ..., Z } называется тривиальной тогда и только тогда, когда одна из проекций A, B, ..., Z является проекцией, идентичной R (т.е. проекцией по всем атрибутам переменной-отношения R).

Общая схема процедуры нормализации

1. Переменную-отношение в 1НФ следует разбить на такие проекции, которые позволят исключить все функциональные зависимости, не являющиеся неприводимыми. В результате будет получен набор переменных-отношений в 2НФ.

2. Полученные переменные-отношения в 2НФ следует разбить на такие проекции, которые позволят исключить все существующие транзитивные функциональные зависимости. В результате будет получен набор переменных-отношений в 3НФ.

3. Полученные переменные-отношения в 3НФ следует разбить на проекции, позволяющие исключить любые оставшиеся функциональные зависимости, в которых детерминанты не являются потенциальными ключами. В результате такого приведения будет получен набор переменных-отношений в НФБК. Замечание. Правила 1-3 могут быть объединены в одно: "Исходную переменную-отношение следует разбить на проекции, позволяющие исключить все функциональные зависимости, в которых детерминанты не являются потенциальными ключами".

4. Полученные переменные-отношения в НФБК следует разбить на проекции, позволяющие исключить любые многозначные зависимости, которые не являются функциональными. В результате будет получен набор переменных- отношений в 4НФ.

5. Полученные переменные-отношения в 4НФ следует разбить на проекции, позволяющие исключить любые зависимости соединения, которые не подразумеваются потенциальными ключами. В результате будет получен набор переменных-отношений в 5НФ.

6 Инструкции управления потоком

Ключевыми словами языка управления потоком Transact-SQL являются следующие:

- BEGIN...END
- BREAK
- CONTINUE
- GOTO
- IF... ELSE
- RETURN
- TRY... CATCH

```
BEGIN TRY
    { sql_инструкция | блок_инструкций }
END TRY
BEGIN CATCH
    { sql_инструкция | блок_инструкций }
END CATCH [ ; ]
```

- WAITFOR

  Блокирует выполнение пакета, хранимой процедуры или транзакции до наступления указанного времени или интервала времени, либо заданная инструкция изменяет или возвращает, по крайней мере, одну строку.

```
WAITFOR { DELAY 'период_времени_ожидания' | TIME 'время_завершения_инструкции
    ' }
```

- WHILE

  Ставит условие повторного выполнения SQL-инструкции или блока инструкций. Эти инструкции вызываются в цикле, пока указанное условие истинно. Вызовами инструкций в цикле WHILE можно контролировать из цикла с помощью ключевых слов BREAK и CONTINUE.

```
WHILE булевское_выражение
    { sql_инструкция | блок_инструкций }
    [ BREAK ]
    { sql_инструкция | блок_инструкций }
    [ CONTINUE ]
    { sql_инструкция | блок_инструкций }
```

В сочетании с инструкциями языка управления потоком могут использоваться и другие конструкции Transact-SQL:

- CASE
- Комментарии (в стиле Си /* комментарий */ и в стиле Ада – комментарий)
- DECLARE

  Переменные объявляются в теле пакета или процедуры при помощи инструкции DECLARE, а значения им присваиваются при помощи инструкций SET или SELECT. Т. о., областью локальной переменной является пакет или процедура, в которых она объявлена. После объявления все переменные инициализируются значением NULL. Переменная не может принадлежать к типу данных text, ntext или image.

```
DECLARE { @локальная_переменная [AS] тип_данных [ = константное_выражение ] }
    [ ,...n]
```

- SET

  Устанавливает локальную переменную, предварительно созданную при помощи инструкции DECLARE локальная_переменная, в указанное значение.

```
SET @локальная_переменная = выражение
```

- EXECUTE

  Выполняет командную строку — строку символов, в которой содержится пакет Transact-SQL или один из следующих модулей: системная хранимая процедура, пользовательская хранимая процедура, скалярная пользовательская функция или расширенная хранимая процедура.

```
[ { EXEC | EXECUTE } ]
    {
        [ @код_возврата = ]
        { имя_модуля | @переменная_именя_модуля }
        [ [ @параметр = ] { значение | @параметр [ OUTPUT ] | [ DEFAULT ] }
        ]
        [ ,...n ]
    }
[;]
```

- PRINT

  Возвращает клиенту пользовательское сообщение.

```
PRINT строковая_константа | @локальная_переменная | строковое_выражение
```

- RAISERROR

  Создает сообщение об ошибке и запускает обработку ошибок для сеанса. Инструкция RAISERROR может либо ссылаться на определенное пользователем сообщение, находящееся в представлении каталога sys.messages, либо динамически создавать сообщение. Это сообщение возвращается как сообщение об ошибке сервера вызывающему приложению или соответствующему блоку CATCH конструкции TRY...CATCH.

```
RAISERROR ( { номер_ошибки | строковая_константа | @локальная_переменная }
    { , серьезность_ошибки , состояние_ошибки
    } [ , аргумент [ ,...n ] ] )
```

7 Системы типов данных языка SQL

7.1 Типы данных SQL

Все допустимые в SQL типы данных, которые можно использовать при определении столбцов, разбиваются на следующие категории:

- точные числовые типы (exact numerics);
- приближенные числовые типы (approximate numerics);
- типы символьных строк (character strings);
- типы битовых строк (bit strings);
- типы даты и времени (datetimes);
- типы временных интервалов (intervals);
- булевский тип (Booleans);

  При определении столбца булевского типа указывается просто спецификация BOOLEAN. Булевский тип состоит из трех значений: true, false и unknown (соответствующие литералы обозначаются TRUE, FALSE и UNKNOWN). Поддерживается возможность построения булевских выражений, которые вычисляются в трехзначной логике.
- типы коллекций (collection types);

  Под термином коллекция обычно понимается одно из следующих образований: массив, список, множество и мультимножество.
- анонимные строчные типы (anonymous row types);

  Анонимный строчный тип – это конструктор типов ROW, позволяющий производить безымянные типы строк (кортежей).
- типы, определяемые пользователем (user-defined types);

  Структурные типы (Structured Types), Индивидуальные типы.
- ссылочные типы (reference types).

  Фактически значениями ссылочного типа являются строки соответствующей типизированной таблицы.

7.2 Типы данных SQL Server

- Точные числа: bigint int smallint tinyint bit decimal numeric money smallmoney
- Приблизительные числа: float real
- Дата и время: date, datetime2, datetime, datetimeoffset, smalldatetime, time
- Символьные строки: char varchar text
- Символьные строки в Юникоде: nchar nvarchar ntext
- Двоичные данные: binary varbinary image
- Прочие типы данных: cursor, hierarchyid, sql_variant, table, timestamp, uniqueidentifier, xml, пространственные типы (geography и geometry)

Константы T-SQL – Константа, также называемая литералом или скалярным значением, зависит от типа данных.

8 Скалярные выражения

Сочетание операндов и операторов, используемое компонентом SQL Server для получения одиночного значения данных.

В простейшем случае выражение может быть:

- литералом (константой);
- функцией;
- именем столбца;
- переменной;
- вложенным запросом;
- функцией CASE.

В выражениях символы и значения типа datetime необходимо заключать в одинарные кавычки. Символьные константы, используемые в качестве шаблона для предложения LIKE, должны быть заключены в одинарные кавычки.

Таблица 1: Категории операторов T-SQL

Категория	**Операторы**
Арифметические операторы	+,-,*,/,%
Логические операторы	ALL, AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR, SOME
Оператор присваивания	=
Оператор разрешения области	:: (обеспечивает доступ к статическим элементам составного типа данных)
Битовые операторы	&,|, ∧
Операторы наборов	EXCEPT, INTERSECT, UNION
Операторы сравнения	=, >, <, >=, <=, <>, !=, !<, !>
Оператор объединения строк	+ (сцепление строк)
Составные операторы	+=, -=, *=, /=, %=, &=, ∧=, |=
Унарные операторы	+,-,

Если два оператора в выражении имеют один и тот же уровень старшинства, они выполняются в порядке слева направо по мере их появления в выражении.

Чтобы изменить приоритет операторов в выражении, следует использовать скобки.

9 Приведение и преобразование типов данных

Следующие функции поддерживают приведение и преобразование типов данных:

- CAST и CONVERT
- PARSE
- TRY_CAST
- TRY_CONVERT
- TRY_PARSE

Использование функций CAST и CONVERT для преобразования выражений одного типа в другой
CAST (выражение AS целевой_тип_данных [(длина)])
CONVERT (целевой_тип_данных [(длина)] , выражение [, стиль])
TRY_CAST, TRY_CONVET, TRY_PARSE возвращают NULL в случае если данные невозможно конвертировать, в отличие от их аналогов завершающихся с ошибкой.

10 Условие поиска

Сочетание одного или нескольких предикатов, в котором используются логические операторы AND, OR и NOT. Параметр `search_condition` в инструкции DELETE, MERGE, SELECT или UPDATE указывает, сколько строк возвращается или обрабатывается инструкцией.

```
<search_condition> ::=
    { [ NOT ] <predicate> | ( <search_condition> ) } [ { AND | OR } [ NOT ] { <
       predicate> | ( <search_condition> ) } ]
    [ ,...n ]

<predicate> ::=
{   expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } expression
    | match_expression [ NOT ] LIKE pattern [ ESCAPE 'escape_character' ] |
       expression [ NOT ] BETWEEN expression AND expression
    | expression IS [ NOT ] NULL
    | CONTAINS  ( { column | * } , '<contains_search_condition>' )
    | FREETEXT ( { column | * } , 'freetext_string' )
    | test_expression [ NOT ] IN ( subquery | expression [ ,...n ] )
    | expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }  { ALL | SOME
       | ANY} ( subquery )
    | EXISTS ( subquery )
}
```

`expression` – скалярное выражение.

`match_expression` – любое допустимое выражение символьного типа данных.

`pattern` – конкретная строка символов для поиска в `match_expression`, которая может содержать следующие допустимые символы-шаблоны: %, _, [], [∧]. Длина значения pattern не может превышать 8000 байт.

`escape_character` – символ, помещаемый перед символом-шаблоном, чтобы символ-шаблон рассматривался как обычный символ, а не как шаблон.

`CONTAINS` – осуществляет поиск столбцов, содержащих символьные данные с заданной точностью (fuzzy), соответствующие заданным отдельным словам и фразам на основе похожести словам и точному расстоянию между словами, взвешенному совпадению. Этот параметр может быть использован только в инструкции SELECT.

`FREETEXT` – предоставляет простую форму естественного языка ввода запросов на осуществление поиска столбцов, содержащих символьные данные, совпадающие с содержанием предиката не точно, а по смыслу. Этот параметр может быть использован только в инструкции SELECT Список значений необходимо заключать в скобки.

`subquery` – может рассматриваться как ограниченная инструкция SELECT и являющаяся подобной на `query_expresssion` в инструкции SELECT. Использование предложений ORDER BY, COMPUTE и ключевого слова INTO не допускается.

11 Сценарии и пакеты

11.1 Общие сведения о сценариях

В общем случае сценарии (script) пишутся для решения от начала и до конца какой-то определенной задачи. С этой целью в один сценарий объединяются несколько команд. Примерами могут служить сценарии, используемые для построения баз данных (они часто применяются при установке систем). Сценарий не является сценарием до тех пор, пока он не сохранен в файле, из которого он может быть извлечен и использован в дальнейшем. SQL-сценарии хранятся в текстовых файлах. Любой сценарий всегда рассматривается как единое целое. Либо выполняется весь сценарий целиком, либо совсем ничего не выполняется. В сценариях могут использоваться системные функции и локальные переменные.

Для объявления переменных используется оператор **DECLARE**, который имеет следующий синтаксис:

DECLARE список-объявлений-переменных

где объявление переменной имеет вид

@имя-переменной тип-переменной

Существует два способа присваивания значений переменным. Можно использовать операторы **SELECT или SET.**

Функционально они практически эквивалентны, за исключением того, что прямо внутри оператора SELECT можно использовать в качестве присваиваемых значений содержимое полей.

Оператор SET традиционно используется для присвоения значений переменным подобно тому, как это делается во многих процедурных языках. Типичным примером может служить следующий фрагмент кода:

SET @Tota1Cost = 10

SET @Tota1Cost = @UnitCost * 1.1

SELECT обычно используется в том случае, когда переменной непосредственно нужно присвоить значение, полученное в результате запроса.

11.2 Общие сведения о пакетах

Пакет (batch) - это несколько объединенных в одну логическую группу операторов Transact-SQL. Все операторы в рамках пакета комбинируются в единый план исполнения (execution plan) таким образом, что пока все операторы не будут успешно проанализированы синтаксическим анализатором, ни один из операторов пакета не будет исполняться. Если на этапе синтаксического анализа какой-либо оператор из пакета оказывается ошибочным, то ни один оператор пакета не выполняется. Если какой-либо оператор пакета вызывает ошибку на этапе выполнения программы, то выполняются все операторы пакета вплоть до ошибочного.

Для того чтобы разделить сценарий на несколько пакетов, необходимо использовать оператор GO.

Оператор GO:

- должен писаться в отдельной строке (ничего, кроме комментариев, не должно следовать за ним в этой же строке); есть исключения, которые будут рассматриваться ниже;
- все операторы от начала сценария (или ближайшего предыдущего оператора GO) и до данного оператора GO компилируются в один план исполнения и пересылаются на сервер отдельно от других пакетов, т. е. ошибка в одном пакете не может помешать выполнению другого пакета;
- это не команда Transact-SQL, а директива, которую распознают различные утилиты с командным интерфейсом SQL Server (OSQL, ISQL и Анализатор Запросов).

Цели применения пакетов различны, но обычно они используются в ситуациях, когда в сценарии необходимо что-нибудь выполнить либо перед, либо отдельно от всего остального. Чаще всего пакеты используются для явного задания предшествования, – когда необходимо полностью завершить решение одной задачи перед тем, как начнет решаться другая.

Правила использования пакетов

1. Инструкции CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER и CREATE VIEW не могут быть объединены с другими инструкциями в пакете. Инструкция CREATE должна быть первой инструкцией в пакете. Все последующие инструкции в этом пакете рассматриваются как часть определения первой инструкции CREATE.
2. В одном и том же пакете нельзя сначала изменить таблицу и затем обратиться к новым столбцам.
3. Если инструкция EXECUTE — первая инструкция в пакете, ключевое слово EXECUTE не требуется. Ключевое слово EXECUTE требуется, если инструкция EXECUTE не является первой инструкцией в пакете.
4. Если необходимо удалить объект, то следует поместить оператор DROP в отдельный пакет или, по крайней мере, в пакет с другими операторами DROP. Объяснение этому следующее. Если необходимо создать объект с тем именем, которое перед этим удалили, то CREATE вызовет ошибку во время синтаксического

анализа, несмотря на то, что оператор DROP уже был выполнен. Это означает, что следует запускать DROP в отдельном пакете, который предшествует CREATE, чтобы удаление старого объекта завершилось к тому моменту, когда начнет выполняться создание нового объекта с таким же именем.

12 Массовый импорт и экспорт данных

Массовый экспорт означает копирование данных из таблицы SQL Server в файл данных. Массовый импорт означает загрузку данных из файла данных в таблицу SQL Server. Например, можно экспортировать данные из приложения Microsoft Excel в файл данных, а затем выполнить массовый импорт данных в таблицу SQL Server. SQL Server поддерживает массовый экспорт данных (массовых данных) из таблиц SQL Server и импорт массовых данных в таблицу SQL Server. Массовый импорт и массовый экспорт имеют большое значение для эффективной передачи данных между SQL Server и разнородными источниками данных.

12.1 Массовый импорт и экспорт данных с помощью программы bcp

Программа bcp используется для массового копирования данных между экземпляром Microsoft SQL Server и файлом данных в пользовательском формате. С помощью программы bcp можно выполнять импорт большого количества новых строк в таблицы SQL Server или экспорт данных из таблиц в файлы данных. Сведения о синтаксисе команды bcp можно получить, выполнив команду без аргументов. В простейших случаях используются следующие аргументы.

```
{[[database_name.][schema].]{table_name | view_name}
```

Имя базы данных database_name и имя схемы schema являются необязательными параметрами.

- `in data_file`

  Выполняется копирование из файла data_file в таблицу базы данных или представление.

- `out data_file`

  Выполняется копирование из таблицы базы данных или представления в файл data_file. Если указать существующий файл, то файл перезаписывается.

- `queryout` Выполняется копирование из запроса.

12.2 Массовый импорт и экспорт данных с помощью мастера служб Integration Services

В состав служб MSIS входит несколько мастеров, одним из которых является мастер импорта и экспорта SQL Server. Этот мастер может копировать данные из любого источника в таблицы и представления, если для этого источника существует поставщик данных .NET Framework или собственный поставщик данных OLE DB. Для источников данных SQL Server, Плоские файлы, Microsoft Access, Microsoft Excel есть соответствующие поставщики. Чтобы выполнить импорт, нужно запустить созданный мастером пакет. При желании пакет можно сохранить для повторного использования.

Запуск мастера импорта и экспорта SQL Server в среде SSMS выполняется так:

1. Подключитесь к серверу типа Database Engine,
2. Разверните базы данных,
3. Правой кнопкой мыши щелкните базу данных, выберите пункт Задачи, затем выберите пункт Импорт данных.

12.3 Массовый импорт и экспорт данных с помощью программы bcp

Программа bcp используется для массового копирования данных между экземпляром Microsoft SQL Server и файлом данных в пользовательском формате. С помощью программы bcp можно выполнять импорт большого количества новых строк в таблицы SQL Server или экспорт данных из таблиц в файлы данных.

12.4 Массовый импорт данных при помощи инструкции BULK INSERT

Инструкция `BULK INSERT` загружает данные из файла данных в таблицу. Принцип работы здесь тот же, что и при выполнении команды bcp с параметром in, однако файл данных считывается процессом SQL Server.

```
BULK INSERT dbSPJ.dbo.S FROM 'C:\Documents and Settings\Admin\S.txt'
WITH (DATAFILETYPE = char', FIELDTERMINATOR='\t', ROWTERMINATOR='\n', CODEPAGE='
   ACP')
```

13 Представления, функции, хранимые триггеры (DDL, DML), курсоры

13.1 Представление

Представление (View) – это виртуальная таблица, содержимое которой (столбцы и строки) определяется запросом.

Представление можно использовать в следующих целях:

- Для направления, упрощения и настройки восприятия информации в базе данных каждым пользователем.
- В качестве механизма безопасности, позволяющего пользователям обращаться к данным через представления, но недающего им разрешений на непосредственный доступ к базовым таблицам.

Создание представления

Для создания представления данных, содержащихся в одной или более таблицах базы данных, необходимо использовать инструкцию CREATE VIEW, которая должна быть первой в пакетном запросе.

Базовый синтаксис:

CREATE VIEW [имя-схемы .] имя-представления [(столбец [,...n])]

AS инструкция-select

столбец – имя, которое будет иметь столбец в представлении. Имя столбца требуется только в тех случаях, когда столбец формируется на основе арифметического выражения, функции или константы, если два или более столбцов могут по иной причине получить одинаковые имена (как правило, в результате соединения) или если столбцу представления назначается имя, отличное от имени столбца, от которого он произведен. Назначать столбцам имена можно также в инструкции SELECT.

Если аргумент столбец не указан, столбцам представления назначаются такие же имена, которые имеют столбцы в инструкции SELECT.

Обновляемые представления

Можно изменять данные базовой таблицы через представление до тех пор, пока выполняются следующие условия:

- Любые изменения, в том числе инструкции UPDATE, INSERT и DELETE, должны ссылаться на столбцы только одной базовой таблицы.
- Изменяемые в представлении столбцы должны непосредственно ссылаться на данные столбцов базовой таблицы.

  Столбцы нельзя сформировать каким-либо другим образом, в том числе:

  – при помощи агрегатной функции: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR и VARP;

  – на основе вычисления. Столбец нельзя вычислить по выражению, включающему другие столбцы. Столбцы, сформированные при помощи операторов UNION, UNION ALL, CROSSJOIN, EXCEPT и INTERSECT, считаются вычисляемыми и также не являются обновляемыми.

  – Предложения GROUP BY, HAVING и DISTINCT не влияют на изменяемые столбцы.

  – Предложение TOP не используется нигде в инструкции select представления вместе с предложением WITH CHECK OPTION.

13.2 Функции

Различают детерминированные и недетерминированные функции.

Функция является **детерминированной**, если при одном и том же заданном входном значении она всегда возвращает один и тот же результат. Например, встроенная функция DATEADD является детерминированной; она возвращает новое значение даты, добавляя интервал к указанной части заданной даты.

Функция является **недетерминированной**, если она может возвращать различные значения при одном и том же заданном входном значении. Например, встроенная функция GETDATE является недетерминированной; при каждом вызове она возвращает различные значения даты и времени компьютера, на котором запущен экземпляр SQL Server.

Пользовательские функции в зависимости от типа данных возвращаемых ими значений могут быть скалярными и табличными. Табличные пользовательские функции бывают двух типов: подставляемые и многооператорные.

Создание и вызов скалярной функции

Для создания скалярной функции используется инструкция CREATE FUNCTION, имеющая следующий синтаксис:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS скалярный-тип-данных
[ WITH список-опций-функций ]
[ AS ]
BEGIN
    тело-функции
    RETURN скалярное-выражение
END [ ; ]
```

Создание и вызов подставляемой табличной функции

Синтаксис создания подставляемой табличной функции выглядит так:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS TABLE
[ WITH список-опций-функций ]
[ AS ]
    RETURN [ ( ] выражение-выборки [ ) ]
END [ ; ]
```

Тело подставляемой табличной функции фактически состоит из единственной инструкции SELECT.

Создание и вызов многооператорной табличной функции

Синтаксис создания многооператорной табличной функции выглядит так:

```
CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )
RETURNS @имя-возвращаемой-переменной TABLE определение-таблицы
[ WITH список-опций-функций ]
[ AS ]
BEGIN
    тело функции
    RETURN
END [ ; ]
```

13.3 Триггеры

Триггеры DDL

Триггер DDL - это хранимая процедура особого типа, которая выполняет одну или несколько инструкций T-SQL в ответ на событие из области действия сервера или базы данных. Например, триггер DDL может активироваться, если выполняется такая инструкция, как ALTER SERVER CONFIGURATION, или если происходит удаление таблицы с использованием команды DROP TABLE.

Триггеры DML

Для поддержания согласованности и точности данных используются декларативные и процедурные методы.

Триггеры представляют собой особый вид хранимых процедур, привязанных к таблицам и представлениям. Они позволяют реализовать в базе данных сложные процедурные методы поддержания целостности данных. События при модификации данных вызывают автоматическое срабатывание триггеров.

Триггеры применяются в следующих случаях:

- если использование методов декларативной целостности данных не отвечает функциональным потребностям приложения. Например, для изменения числового значения в таблице при удалении записи из этой же таблицы следует создать триггер;
- если необходимо каскадное изменение через связанные таблицы в базе данных. Чтобы обновить или удалить данные в столбцах с ограничением foreign key, вместо пользовательского триггера следует применять ограничения каскадной ссылочной целостности;
- если база данных денормализована и требуется способ автоматизированного обновления избыточных данных в нескольких таблицах;
- если необходимо сверить значение в одной таблице с неидентичным значением в другой таблице;
- если требуется вывод пользовательских сообщений и сложная обработка ошибок.

События, вызывающие срабатывание триггеров (типы триггеров)

Автоматическое срабатывание триггера вызывают три события: INSERT, UPDATE и DELETE, которые происходят в таблице или представлении. Триггеры нельзя запустить вручную. В синтаксисе триггеров перед фрагментом программы, уникально определяющим выполняемую триггером задачу, всегда определено одно или несколько таких событий. Один триггер разрешается запрограммировать для реакции на несколько событий, поэтому несложно создать процедуру, которая одновременно является триггером на обновление и добавление. Порядок этих событий в определении триггера является несущественным.

Классы триггеров
В SQL Server существуют два класса триггеров:

- INSTEAD OF. Триггеры этого класса выполняются в обход действий, вызывавших их срабатывание, заменяя эти действия. Например, обновление таблицы, в которой есть триггер INSTEAD OF, вызовет срабатывание этого триггера. В результате вместо оператора обновления выполняется код триггера. Это позволяет размешать в триггере сложные операторы обработки, которые дополняют действия оператора, модифицирующего таблицу.
- AFTER. Триггеры этого класса исполняются после действия, вызвавшего срабатывание триггера. Они считаются классом триггеров по умолчанию.

13.4 Курсоры

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE. Такой полный набор строк, возвращаемых инструкцией, называется результирующим набором. Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз. Курсоры предоставляют такой механизм.
Методика использования курсора языка T-SQL такова:

1. С помощью инструкции DECLARE необходимо объявить переменные T-SQL, которые будут содержать данные, возвращенные курсором. Для каждого столбца результирующего набора надо объявить по одной переменной. Переменные должны быть достаточно большого размера для хранения значений, возвращаемых в столбце и имеющих тип данных, к которому могут быть неявно преобразованы данные столбца.
2. С помощью инструкцией DECLARE CURSOR необходимо связать курсор T-SQL с инструкцией SELECT. Инструкция DECLARE CURSOR определяет также характеристики курсора, например имя курсора и тип курсора (read-only или forward-only).
3. С помощью инструкции OPEN выполнить инструкцию SELECT и заполнить курсор.
4. С помощью инструкции FETCH INTO организовать перемещение по курсору для выборки отдельных строк; значение каждого столбца отдельной строки заносится в указанную переменную. После этого другие инструкции T- SQL могут ссылаться на эти переменные для доступа к выбранным значениям данных. Курсоры T-SQL не поддерживают выборку группы строк.
5. После завершения работы с курсором необходимо применить инструкцию CLOSE. Закрытие курсора освобождает некоторые ресурсы, например результирующий набор курсора и его блокировки на текущей строке, однако структура курсора будет доступна для обработки, если снова выполнить инструкцию OPEN. Поскольку курсор все еще существует на этом этапе, повторно использовать его имя нельзя.
6. Наконец инструкция DEALLOCATE полностью освобождает все ресурсы, выделенные курсору, в том числе имя курсора. После освобождения курсора его необходимо строить заново с помощью инструкции DECLARE CURSOR.

В SQL Server поддерживаются четыре типа серверных курсоров:

1. Статические курсоры (STATIC). Создается временная копия данных для использования курсором. Все запросы к курсору обращаются к указанной временной таблице в базе данных tempdb, поэтому изменения базовых таблиц не влияют на данные, возвращаемые выборками для данного курсора, а сам курсор не позволяет производить изменения.
2. Динамические курсоры (DYNAMIC). Отображают все изменения данных, сделанные в строках результирующего набора при просмотре этого курсора. Значения данных, порядок, а также членство строк в каждой выборке могут меняться. Параметр выборки ABSOLUTE динамическими курсорами не поддерживается.
3. Курсоры, управляемые набором ключей (KEYSET). Членство или порядок строк в курсоре не изменяются после его открытия. Набор ключей, однозначно определяющих строки, встроен в таблицу в базе данных tempdb с именем keyset.
4. Быстрые последовательные курсоры (FAST_FORWARD). Параметр FAST_FORWARD указывает курсор FORWARD_ONLY, READ_ONLY, для которого включена оптимизация производительности. Параметр FAST_FORWARD не может указываться вместе с параметрами SCROLL или FOR_UPDATE.

По области видимости имени курсора различают:

1. Локальные курсоры (LOCAL). Область курсора локальна по отношению к пакету, хранимой процедуре или триггеру, в которых этот курсор был создан. Курсор неявно освобождается после завершения выполнения пакета, хранимой процедуры или триггера, за исключением случая, когда курсор был передан параметру OUTPUT. В этом случае курсор освобождается при освобождении всех ссылающихся на него переменных или при выходе из области видимости.

2. Глобальные курсоры (GLOBAL). Область курсора является глобальной по отношению к соединению. Имя курсора может использоваться любой хранимой процедурой или пакетом, которые выполняются соединением. Курсор неявно освобождается только в случае разрыва соединения. Глобальность курсора позволяет создавать его в рамках одной хранимой процедуры, а вызывать его из совершенно другой процедуры, причем передавать его в эту процедуру необязательно.

По способу перемещения по курсору различают:

3. Последовательные курсоры (FORWARD_ONLY). Курсор может просматриваться только от первой строки к последней. Поддерживается только параметр выборки FETCH NEXT. Если параметр FORWARD_ONLY указан без ключевых слов STATIC, KEYSET или DYNAMIC, то курсор работает как DYNAMIC. Если не указан ни один из параметров FORWARD_ONLY или SCROLL, а также не указано ни одно из ключевых слов STATIC, KEYSET или DYNAMIC, то по умолчанию задается параметр FORWARD_ONLY. Курсоры STATIC, KEYSET и DYNAMIC имеют значение по умолчанию SCROLL.

4. Курсоры прокрутки (SCROLL). Перемещение осуществляется по группе записей как вперед, так и назад. В этом случае доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Параметр SCROLL не может указываться вместе с параметром для FAST_FORWARD.

По способу распараллеливания курсоров различают:

1. READ_ONLY. Содержимое курсора можно только считывать.

2. SCROLL_LOCKS. При редактировании данной записи вами никто другой вносить в нее изменения не может. Такую блокировку прокрутки иногда еще называют «пессимистической» блокировкой. При блокировке прокрутки важным фактором является продолжительность действия блокировки. Если курсор не входит в состав некоторой транзакции, то блокировка распространяется только на текущую запись в курсоре. В противном случае все зависит от того, какой используется уровень изоляции транзакций.

3. OPTIMISTIC. Означает отсутствие каких бы то ни было блокировок. «Оптимистическая» блокировка предполагает, что даже во время выполнения вами редактирования данных, другие пользователи смогут к ним обращаться. Если кто-то все-таки попытается выполнить модификацию данных одновременно с вами, генерируется ошибка 16394. В этом случае вам придется повторно выполнить доставку данных из курсора или же осуществить полный откат транзакции, а затем попытаться выполнить ее еще раз.

На содержательном уровне синтаксис инструкции DECLARE CURSOR будет иметь вид:
DECLARE имя-курсора CURSOR
[область-видимости-имени-курсора]
[возможность-перемещения-по-курсору]
[типы-курсоров]
[опции-распараллеливания-курсоров]
[выявление-ситуаций-с-преобразованием-типа-курсора]
FOR инструкция_select
[опция-FOR-UPDATE]

Перемещение внутри курсора (прокрутка курсора) Основой всех операций прокрутки курсора является ключевое слово FETCH. Оно может использоваться для перемещения по курсору в обоих направлениях, в том числе и для перехода к заданной позиции.

В качестве аргументов оператора FETCH могут выступать:

- NEXT – возвращает строку результата сразу же за текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.

- PRIOR – возвращает строку результата, находящуюся непосредственно перед текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.

- FIRST – возвращает первую строку в курсоре и делает ее текущей.

- LAST – возвращает последнюю строку в курсоре, и делает ее текущей.
- ABSOLUTE n | @nvar. Если n или @nvar имеют положительное значение, возвращает строку, стоящую дальше на n строк от передней границы курсора, и делает возвращенную строку новой текущей строкой. Если n или @nvar имеют отрицательное значение, возвращает строку, отстоящую на n строк от задней границы курсора, и делает возвращенную строку новой текущей строкой. Если n или @nvar равны 0, не возвращается никакая строка. n должно быть целым числом, а @nvar должна иметь тип данных smallint, tinyint или int.
- RELATIVE n | @nvar. Если n или @nvar имеют положительное значение, возвращает строку, отстоящую на n строк вперед от текущей строки, и делает возвращенную строку новой текущей строкой. Если n или @nvar имеют отрицательное значение, возвращает строку, отстоящую на n строк назад от текущей строки, и делает возвращенную строку новой текущей строкой. Если n или @nvar равны 0, возвращает текущую строку. Если при первой выборке из курсора инструкция FETCH RELATIVE указывается с отрицательными или равными нулю параметрами n или @nvar, то никакая строка не возвращается. n должно быть целым числом, а @nvar должна иметь тип данных smallint, tinyint или int.

14 Метаданные и доступ к ним

Метаданные, в общем случае, это данные о данных, информация об информации, описание контента. Каждая СУБД сохраняет метаданные обо всех сущностях базы данных. Так в SQL Server с помощью инструкции CREATE можно создать 52 сущности.

В разных СУБД применяются разные названия для метаданных - системный каталог, словарь данных и др. Однако общим свойством всех современных реляционных СУБД является то, что каталог/словарь сам состоит из таблиц, а точнее - системных таблиц. В результате пользователь может обращаться к метаданным так же, как и к прикладным данным, используя инструкцию SELECT. Изменения же в каталоге/словаре производятся автоматически при выполнении пользователем инструкций, изменяющих состояние объектов базы данных. Системные таблицы не должны изменяться непосредственно ни одним пользователем. Например, не стоит изменять системные таблицы с помощью инструкций DELETE, UPDATE или INSERT либо с помощью пользовательских триггеров. Обращение к документированным столбцам системных таблиц разрешено. Однако многие столбцы системных таблиц не документированы. В приложениях непосредственные запросы к недокументированным столбцам применять не следует. Чтобы исключить прямой доступ к системным таблицам, пользователь «видит» не сами таблицы, а созданные на их базе представления, которые он, конечно же, не может изменять. Состав и структура каталога/словаря очень различны для различных СУБД.

Microsoft SQL Server предоставляет следующие коллекции системных представлений, содержащие метаданные:

- Представления информационной схемы
- Представления каталога
- Представления совместимости
- Представления репликации
- Динамические административные представления и функции
- Представления приложения уровня данных (DAC)

Представления информационной схемы определяются в особой схеме с именем INFORMATION_SCHEMA. Эта схема содержится в любой базе данных и состоит из 20 представлений.

15 Методы физического хранения данных на диске

Хранение данных и в прошлых, и в новых версиях организовано в виде иерархических структур:

- База данных. Рассматривается как наивысший уровень абстракции для хранилища данных (конкретного сервера). Является высшим уровнем, на который может быть наложена блокировка (lock), хотя явно задать блокировку на уровне базы данных нельзя.
- Файл. Рассматривать как ближайшее подобие устройства (в более ранних версиях). Для хранения единственной базы данных могут быть выделены несколько файлов. Но в один файл нельзя поместить более одной базы данных. По умолчанию база данных использует два файла.
  1. В первом физическом файле базы данных хранится реальная информация. Файл должен иметь расширение .mdf (это рекомендация, а не требование).
  2. Второй файл является вспомогательным файлом базы данных – ее журналом. Журнал постоянно хранится в файле с расширением .ldf, и без него база данных работать не будет.
- Экстент. Являет собой основную единицу пространства, выделяемую под таблицу или индекс. Экстент состоит из восьми смежных страниц данных. В SQL Server при создании таблицы изначально не выделяется ни одной страницы. Они добавляются только при вставке в таблицу новых строк. По мере добавления в таблицу записей, для хранения информации выделяются все новые страницы до тех пор, пока их количество не достигнет восьми. С этого момента при необходимости дополнительного пространства для хранения данных SQL Server будет выделяться как минимум экстент.

  Экстенты бывают двух типов:

  1. Разделяемые экстенты. Разделяемые экстенты могут совместно использоваться восемью различными объектами (т. е. каждая страница экстента может принадлежать другому объекту). Все создаваемые таблицы и индексы помещаются в разделяемые экстенты. Как только количество страниц, выделенных под объект, достигает восьми, для объекта будет выделен новый однородный экстент.
  2. Однородные экстенты. Структура однородных экстентов понятна из названия. Каждая страница такого экстента принадлежит одному и тому же объекту.

  Об экстентах должно быть известно следующее:

  1. После заполнения экстента при вставке новой записи для ее хранения выделяется целый новый экстент, а не только пространство, соответствующее размеру добавляемой записи.
  2. Благодаря предварительному выделению пространства, SQL Server экономит время, которое потребовалось бы в случае выделения пространства во время добавления каждой новой строки.
  3. Экстенты могут блокироваться. Однако блокировка может накладываться на экстент только во время выделения нового либо высвобождения старого экстента.
- Страница. Является элементарной единицей пространства выделяемого внутри отдельного экстента. В SQL Server размер страницы составляет 8 КБ. Страница может рассматриваться как контейнер для хранения и строк таблиц и индексов. Одна строка не может быть разделена между двумя страницами. В состав страницы входят:
  1. 96-байтный заголовок страницы (page header); заголовок включает: номер страницы, тип страницы, количество свободного пространства на странице, идентификатор единицы распределения объекта, которому принадлежит страница.
  2. сами данных и
  3. указатели смещения строк (row offset).

16 Индексы

Индекс — это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных. Индекс может создаваться на одном или нескольких столбцах. Это означает, что индексы бывают простыми и составными. Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице. При наличии индекса время поиска нужных строк можно существенно уменьшить.

К недостаткам индексов следует отнести:

- дополнительное место на диске и в оперативной памяти,
- замедляются операции вставки, обновления и удаления записей.

SQL Server индексы хранятся в виде сбалансированных деревьев. Представление индекса в виде сбалансированного дерева означает, что стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка. Сбалансированное дерево состоит из:

- корневого узла (root node), содержащего одну страницу,
- нескольких промежуточных уровней (intermediate levels), содержащих дополнительные страницы, и
- листового уровня (leaf level).

На страницах листового уровня находятся отсортированные элементы, соответствующие индексируемым данным. По мере добавления данных в таблицу индекс будет разрастаться, но по-прежнему оставаться в форме сбалансированного дерева.

16.1 Кластерные индексы

В кластерном индексе таблица представляет собой часть индекса, или индекс представляет собой часть таблицы в зависимости от вашей точки зрения. Листовой узел кластерного индекса – это страница таблицы с данными. Поскольку сами данные таблицы являются частью индекса, то очевидно, что для таблицы может быть создан только один кластерный индекс.

SQL Server кластерный индекс является уникальным индексом по определению. Это означает, что все ключи записей должны быть уникальные. Если существуют записи с одинаковыми значениями, SQL Server делает их уникальными, добавляя номера из внутреннего (невидимого снаружи) 4-х байтного счетчика. Кластерный индекс использовать необязательно. Тем не менее, кластерные индексы часто используют в качестве первичного ключа. На уровне листьев кластерного индекса информация упорядочивается и физически сохраняется на диске в соответствии с заданными критериями сортировки. На каждом уровне индекса страницы организованы в виде двунаправленного связного списка. Вход в корень кластерного индекса дает поле root_page системного представления sys.system_internals_allocation_units.

При добавлении каждой новой записи хранимые данные пересортировываются, чтобы сохранить физическую упорядоченность данных. Если требуется создать новую запись в середине индексной структуры, то происходит обычное разбиение страницы. Половина записей из старой страницы переносится в новую, а новая запись добавляется либо в новую страницу, либо в старую страницу (для сохранения равновесия). Если же новая запись логически попадает в конец индексной структуры, тогда в новую создаваемую страницу помещается только новая строка (информация не делится поровну между двумя рассматриваемыми страницами).

16.2 Некластерные индексы на основе кучи

В листьях некластерного индекса на основе кучи хранятся указатели на строки данных. Указатель строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID). С точки зрения физического размещения данных – они хранятся в полном беспорядке. С точки зрения технической реализации нужные записи приходится искать по всему файлу. Вполне может статься, что в этом случае SQL Server придется несколько раз возвращаться к одной и той же странице для чтения различных строк, поскольку определить заранее, откуда придется физически считывать следующую строку нет никакой возможности.

16.3 Некластерные индексы, основанные на кластерных таблицах

В листьях некластерного индекса, основанного на кластерных таблицах, хранятся указатели на корневые узлы кластерных индексов. Поиск в таком индексе состоит из двух этапов:

- поиск в некластерном индексе и
- поиск в кластерном индексе.

16.4 Создание индекса с помощью инструкции CREATE INDEX

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX имя-индекса
ON имя-таблицы-или-представления ( список-столбцов ) [ INCLUDE (список-столбцов)
   ]
[WITH список-опций]
[ ON файловая-группа ]
```

17 Управление транзакциями

Транзакция — это последовательность операций, выполняемая как единое целое. В составе транзакций можно исполнять почти все операторы языка Transact-SQL. Если при выполнении транзакции не возникает никаких ошибок, то все модификации базы данных, сделанные во время выполнения транзакции, становятся постоянными. Транзакция выполняется по принципу «все или ничего». Транзакция не оставляет данные в промежуточном состоянии, в котором база данных не согласована. Транзакция переводит базу данных из одного целостного состояния в другое.

Для поддержания целостности транзакция должна обладать четырьмя свойствами АСИД: атомарность, согласованность, изоляция и долговечность.

По признаку определения границ различают автоматические, неявные и явные транзакции.

Автоматические транзакции.

Режим автоматической фиксации транзакций является режимом управления транзакциями SQL Server по умолчанию. В этом режиме каждая инструкция T-SQL выполняется как отдельная транзакция. Если выполнение инструкции завершается успешно, происходит фиксация; в противном случае происходит откат. Если возникает ошибка компиляции, то план выполнения пакета не строится и пакет не выполняется.

Неявные транзакции.

Если соединение работает в режиме неявных транзакций, то после фиксации или отката текущей транзакции SQL Server автоматически начинает новую транзакцию. В этом режиме явно указывается только граница окончания транзакции с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION. Для ввода в действие поддержки неявных транзакций применяется инструкция SET IMPLICIT_TRANSACTION ON. В конце каждого пакета необходимо отключать этот режим. По умолчанию режим неявных транзакций в SQL Server отключен.

Явные транзакции.

Для определения явных транзакций используются следующие инструкции:

- BEGIN TRANSACTION – задает начальную точку явной транзакции для соединения;
- COMMIT TRANSACTION или COMMIT WORK – используется для успешного завершения транзакции, если не возникла ошибка;
- ROLLBACK TRANSACTION или ROLLBACK WORK – используется для отмены транзакции, во время которой возникла ошибка.
- SAVE TRANSACTION – используется для установки точки сохранения или маркера внутри транзакции. Точка сохранения определяет место, к которому может возвратиться транзакция, если часть транзакции условно отменена. Если транзакция откатывается к точке сохранения, то ее выполнение должно быть продолжено до завершения с обработкой дополнительных инструкций языка T-SQL, если необходимо, и инструкции COMMIT TRANSACTION, либо транзакция должна быть полностью отменена откатом к началу. Для отмены всей транзакции следует использовать инструкцию ROLLBACK TRANSACTION; в этом случае отменяются все инструкции транзакции.

Функции для обработки транзакций

- @@TRANCOUNT возвращает число активных транзакций для текущего соединения. Инструкция BEGIN TRANSACTION увеличивает значение @@TRANCOUNT на 1, а инструкция ROLLBACK TRANSACTION уменьшает его до 0 (исключение — инструкция ROLLBACK TRANSACTION имя_точки_сохранения, которая не влияет на значение @@TRANCOUNT). Инструкции COMMIT TRANSACTION уменьшают значение @@TRANCOUNT на 1.
- XACT_STATE () сообщает о состоянии пользовательской транзакции текущего выполняемого запроса в соответствии с данными, представленными в таблице.

18 Неблагоприятные эффекты, вызванные параллельным выполнением транзакций, и их устранение

Проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;

Проблема «грязного» чтения (Dirty Read) возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;

Проблема неповторяемого чтения (Non-repeatable or Fuzzy Read) является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;

Проблема чтения фантомов (Phantom) появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

18.1 Управлением параллельным выполнением транзакций

Когда множество пользователей одновременно пытаются модифицировать данные в базе данных, необходимо создать систему управления, которая защитила бы модификации, выполненные одним пользователем, от негативного воздействия модификаций, сделанных другими. Выделяют два типа управления параллельным выполнением:

- Пессимистическое управление параллельным выполнением.
- Оптимистическое управление параллельным выполнением.

19 Уровни изоляции транзакций и блокировки

Стандарт ISO определяет следующие уровни изоляции:

- read uncommitted (самый низкий уровень, при котором транзакции изолируются до такой степени, чтобы только уберечь от считывания физически поврежденных данных);
- read committed (уровень по умолчанию);
- изоляция повторяющегося чтения repeatable read;
- изоляция упорядочиваемых транзакций serializable (самый высокий уровень, при котором транзакции полностью изолированы друг от друга).

SQL Server также поддерживает еще два уровня изоляции транзакций, использующих управление версиями строк.

- read committed с использованием управления версиями строк;
- уровень изоляции моментальных снимков snapshot.

Следующая таблица показывает побочные эффекты параллелизма, допускаемые различными уровнями изоляции.

Уровень изоляции	«Грязное» чтение	Неповторяющееся чтение	Фантомное чтение
read uncommitted	Да	Да	Да
read committed	Нет	Да	Да
repeatable read	Нет	Нет	Да
snapshot	Нет	Нет	Нет
serializable	Нет	Нет	Нет

Замечания.

1. Одновременно может быть установлен только один параметр уровня изоляции, который продолжает действовать для текущего соединения до тех пор, пока не будет явно изменен.
2. Уровни изоляции транзакции определяют тип блокировки, применяемый к операциям считывания.
3. В любой момент транзакции можно переключиться с одного уровня изоляции на другой, однако есть одно исключение. Это смена уровня изоляции на уровень изоляции SNAPSHOT. Такая смена приводит к ошибке и откату транзакции. Однако для транзакции, которая была начата с уровнем изоляции SNAPSHOT, можно установить любой другой уровень изоляции.
4. Когда для транзакции изменяется уровень изоляции, ресурсы, которые считываются после изменения, защищаются в соответствии с правилами нового уровня. Ресурсы, которые считываются до изменения, остаются защищенными в соответствии с правилами предыдущего уровня. Например, если для транзакции уровень изоляции изменяется с READ COMMITTED на SERIALIZABLE, то совмещаемые блокировки, полученные после изменения, будут удерживаться до завершения транзакции.
5. Если инструкция SET TRANSACTION ISOLATION LEVEL использовалась в хранимой процедуре или триггере, то при возврате управления из них уровень изоляции будет изменен на тот, который действовал на момент их вызова. Например, если уровень изоляции REPEATABLE READ устанавливается в пакете, а пакет затем вызывает хранимую процедуру, которая меняет уровень изоляции на SERIALIZABLE, при возвращении хранимой процедурой управления пакету, настройки уровня изоляции меняются назад на REPEATABLE READ.
6. Определяемые пользователем функции и типы данных среды CLR не могут выполнять инструкцию SET TRANSACTION ISOLATION LEVEL. Однако уровень изоляции можно переопределить с помощью табличной подсказки.

20 Гранулярность блокировок, иерархии блокировок и эскалация блокировок

SQL Server поддерживает мультигранулярную блокировку, позволяющую транзакции блокировать различные типы ресурсов. Чтобы уменьшить издержки применения блокировок, компонент Database Engine автоматически блокирует ресурсы на соответствующем задаче уровне. Блокировка при меньшей гранулярности, например на уровне строк, увеличивает параллелизм, но в то же время увеличивает и накладные расходы на обработку, поскольку при большом количестве блокируемых строк требуется больше блокировок. Блокировки на большем уровне гранулярности, например на уровне таблиц, обходится дорого в отношении параллелизма, поскольку блокировка целой таблицы ограничивает доступ ко всем частям таблицы других транзакций. Однако накладные расходы в этом случае ниже, поскольку меньше количество поддерживаемых блокировок.

Компонент Database Engine часто получает блокировки на нескольких уровнях гранулярности одновременно, чтобы полностью защитить ресурс. Такая группа блокировок на нескольких уровнях гранулярности называется иерархией блокировки. Например, чтобы полностью защитить операцию чтения индекса, экземпляру компоненту Database Engine может потребоваться получить разделяемые блокировки на строки и намеренные разделяемые блокировки на страницы и таблицу.

Следующая таблица содержит перечень ресурсов, которые могут блокироваться компонентом Database Engine.

Ресурс	**Описание**
`RID`	Идентификатор строки, используемый для блокировки одной строки в куче
`KEY`	Блокировка строки в индексе, используемая для защиты диапазонов значений ключа в сериализуемых транзакциях.
`PAGE`	8-килобайтовая (КБ) страница в базе данных, например страница данных или индекса.
`EXTENT`	Упорядоченная группа из восьми страниц, например страниц данных или индекса.
`HOBT`	Куча или сбалансированное дерево. Блокировка, защищающая индекс или кучу страниц данных в таблице, не имеющей кластеризованного индекса.
`TABLE`	Таблица полностью, включая все данные и индексы.
`FILE`	Файл базы данных.
`APPLICATION`	Определяемый приложением ресурс.
`METADATA`	Блокировки метаданных.
`ALLOCATION_UNIT`	Единица размещения.
`DATABASE`	База данных, полностью.

21 Взаимоблокировки, их обнаружение и устранение

Взаимоблокировки или тупиковые ситуации (deadlocks) возникают тогда, когда одна из транзакций не может завершить свои действия, поскольку вторая транзакция заблокировала нужные ей ресурсы, а вторая в то же время ожидает освобождения ресурсов первой транзакцией.

Как SQL Server обнаруживает взаимоблокировки? Обнаружение взаимоблокировки выполняется потоком диспетчера блокировок, который периодически производит поиск по всем задачам в экземпляре компонента Database Engine.

Следующие пункты описывают процесс поиска:

- Значение интервала поиска по умолчанию составляет 5 секунд.
- Если диспетчер блокировок находит взаимоблокировки, интервал обнаружения взаимоблокировок снижается с 5 секунд до 100 миллисекунд в зависимости от частоты взаимоблокировок.
- Если поток диспетчера блокировки прекращает поиск взаимоблокировок, компонент Database Engine увеличивает интервал до 5 секунд.
- Если взаимоблокировка была только что найдена, предполагается, что следующие потоки, которые должны ожидать блокировки, входят в цикл взаимоблокировки. Первая пара элементов, ожидающих блокировки, после того как взаимоблокировка была обнаружена, запускает поиск взаимоблокировок вместо того, чтобы ожидать следующий интервал обнаружения взаимоблокировки. Например, если текущее значение интервала равно 5 секунд и была обнаружена взаимоблокировка, следующий ожидающий блокировки элемент немедленно приводит в действие детектор взаимоблокировок. Если этот ожидающий блокировки элемент является частью взаимоблокировки, она будет обнаружена немедленно, а не во время следующего поиска взаимоблокировок.
- Компонент Database Engine обычно выполняет только периодическое обнаружение взаимоблокировок. Так как число взаимоблокировок, произошедших в системе, обычно мало, периодическое обнаружение взаимоблокировок помогает сократить издержки от взаимоблокировок в системе.
- Если монитор блокировок запускает поиск взаимоблокировок для определенного потока, он идентифицирует ресурс, ожидаемый потоком. После этого монитор блокировок находит владельцев определенного ресурса и рекурсивно продолжает поиск взаимоблокировок для этих потоков до тех пор, пока не найдет цикл. Цикл, определенный таким способом, формирует взаимоблокировку.
- После обнаружения взаимоблокировки компонент Database Engine завершает взаимоблокировку, выбрав один из потоков в качестве жертвы взаимоблокировки. Компонент Database Engine прерывает выполняемый в данный момент пакет потока, производит откат транзакции жертвы взаимоблокировки и возвращает приложению ошибку 1205. Откат транзакции жертвы взаимоблокировки снимает все блокировки, удерживаемые транзакцией. Это позволяет транзакциям потоков разблокироваться, и продолжить выполнение. Ошибка 1205 жертвы взаимоблокировки записывает в журнал ошибок сведения обо всех потоках и ресурсах, затронутых взаимоблокировкой.

Как выбирается жертва взаимоблокировки?

Если у двух сеансов имеются различные приоритеты, то в качестве жертвы взаимоблокировки будет выбран сеанс с более низким приоритетом. Если у обоих сеансов установлен одинаковый приоритет, то в качестве жертвы взаимоблокировки будет выбран сеанс, откат которого потребует наименьших затрат. Если сеансы, вовлеченные в цикл взаимоблокировки, имеют один и тот же приоритет и одинаковую стоимость, то жертва взаимоблокировки выбирается случайным образом.

Рекомендации по предотвращению взаимоблокировок

Невозможно полностью исключить возникновение взаимоблокировок в сложных программных системах, но с практической точки зрения можно сделать их вероятность настолько малой, что данный вопрос перестанет быть актуальным. Для того чтобы избежать появления взаимоблокировок или, по крайней мере, свести их количество к минимуму, надо следовать простым правилам:

- обращайтесь к объектам в одном и том же порядке;
- делайте транзакции как можно более короткими и размещайте их в одном пакете;
- используйте наименьший возможный уровень изоляции транзакций;
- не допускайте разрывов внутри транзакции (со стороны пользователя либо в результате разделения пакета);
- в контролируемой среде используйте связанные подключения.

22 Режимы блокировок и совместимость блокировок

Блокировка — это механизм, с помощью которого компонент Database Engine синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных.

22.1 Режимы блокировок

Режим блокировки	Описание
Совмещаемая блокировка (S)	Используется для операций считывания, которые не меняют и не обновляют данные, такие как инструкция SELECT.
Блокировка обновления (U)	Применяется к тем ресурсам, которые могут быть обновлены. Предотвращает возникновение распространенной формы взаимоблокировки, возникающей тогда, когда несколько сеансов считывают, блокируют и затем, возможно, обновляют ресурс.
Монопольная блокировка (X)	Используется для операций модификации данных, таких как инструкции INSERT, UPDATE или DELETE. Гарантирует, что несколько обновлений не будет выполнено одновременно для одного ресурса.
Блокировка с намерением	Используется для создания иерархии блокировок. Типы намеренной блокировки: с намерением совмещаемого доступа (IS), с намерением монопольного доступа (IX), а также совмещаемая с намерением монопольного доступа (SIX).
Блокировка схемы	Используется во время выполнения операции, зависящей от схемы таблицы. Типы блокировки схем: блокировка изменения схемы (Sch-S) и блокировка стабильности схемы (Sch-M).
Блокировка массового обновления (BU)	Используется, если выполняется массовое копирование данных в таблицу и указана подсказка TABLOCK.
Диапазон ключей	Защищает диапазон строк, считываемый запросом при использовании уровня изоляции сериализуемой транзакции. Запрещает другим транзакциям вставлять строки, что помогает запросам сериализуемой транзакции уточнять, были ли запросы запущены повторно.

22.2 Совместимость блокировок

Совместимость блокировок определяет, могут ли несколько транзакций одновременно получить блокировку одного и того же ресурса. Если ресурс уже блокирован другой транзакцией, новая блокировка может быть предоставлена только в том случае, если режим запрошенной блокировки совместим с режимом существующей. В противном случае транзакция, запросившая новую блокировку, ожидает освобождения ресурса, пока не истечет время ожидания существующей блокировки.

Например, с монопольными блокировками не совместим ни один из режимов блокировки. Пока удерживается монопольная (X) блокировка, больше ни одна из транзакций не может получить блокировку ни одного из типов (разделяемую, обновления или монопольную) на этот ресурс, пока не будет освобождена монопольная (X) блокировка. И наоборот, если к ресурсу применяется разделяемая (S) блокировка, другие транзакции могут получать разделяемую блокировку или блокировку обновления (U) на этот элемент, даже если не завершилась первая транзакция. Тем не менее, другие транзакции не могут получить монопольную блокировку до освобождения разделяемой. Полная матрица совместимости блокировок приводится в справочниках.

23 Обработка ошибок в транзакциях

Если ошибка делает невозможным успешное выполнение транзакции, SQL Server автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Если сетевое соединение клиента с SQL Server разорвано, то после того, как SQL Server получит уведомление от сети о разрыве соединения, выполняется откат всех необработанных транзакций для этого соединения. В случае сбоя клиентского приложения, выключения либо перезапуска клиентского компьютера соединение также будет разорвано, а SQL Server выполнит откат всех необработанных транзакций после получения уведомления о разрыве от сети. Если клиент выйдет из приложения, выполняется откат всех необработанных транзакций.

В случае ошибки (нарушения ограничения целостности) во время выполнения инструкции в пакете по умолчанию SQL Server выполнит откат только той инструкции, которая привела к ошибке. Это поведение можно изменить с помощью инструкции SET XACT_ABORT. После выполнения инструкции SET XACT_ABORT ON любая ошибка во время выполнения инструкции приведет к автоматическому откату текущей транзакции.

На случай возникновения ошибок код приложения должен содержать исправляющее действие: COMMIT или ROLLBACK. Эффективным средством для обработки ошибок, включая ошибки транзакций, является конструкция языка Transact-SQL TRY...CATCH.

Для получения сведений об ошибках в области блока CATCH конструкции TRY...CATCH можно использовать следующие системные функции:

- ERROR_LINE() - возвращает номер строки, в которой произошла ошибка.
- ERROR_MESSAGE() - возвращает текст сообщения, которое будет возвращено приложению. Текст содержит значения таких подставляемых параметров, как длина, имена объектов или время.
- ERROR_NUMBER() - возвращает номер ошибки.
- ERROR_PROCEDURE() - возвращает имя хранимой процедуры или триггера, в котором произошла ошибка. Эта функция возвращает значение NULL, если данная ошибка не была совершена внутри хранимой процедуры или триггера.
- ERROR_SEVERITY() - возвращает уровень серьезности ошибки.
- ERROR_STATE() - возвращает состояние.

Инструкция RAISERROR позволяет вернуть приложению сообщение в формате системных ошибок или предупреждений.

24 Логическая и физическая архитектуры журнала транзакций

24.1 Логическая архитектура журнала транзакций

На логическом уровне журнал транзакций состоит из последовательности записей.
Каждая запись содержит:

- Log Sequence Number: регистрационный номер транзакции (LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи.
- Prev LSN: обратный указатель, который предназначен для ускорения отката транзакции.
- Transaction ID number: идентификатор транзакции.
- Type: тип записи Log-файла.
- Other information: Прочая информация.

Двумя основными типами записи Log-файла являются:

- Transaction Log Operation Code: код выполненной логической операции, либо
- Update Log Record: исходный и результирующий образ измененных данных. Исходный образ записи – это копия данных до выполнения операции, а результирующий образ – копия данных после ее выполнения.

Каждая транзакция резервирует в журнале транзакций место, чтобы при выполнении инструкции отката или возникновения ошибки в журнале было достаточно места для регистрации отката. Объем резервируемого пространства зависит от выполняемых в транзакции операций, но обычно он равен объему, необходимому для регистрации каждой из операций. Все это пространство после завершения транзакции освобождается.

24.2 Физическая архитектура журнала транзакций

На физическом уровне журнал транзакций состоит из одного или нескольких физических файлов. Каждый физический файл журнала разбивается на несколько виртуальных файлов журнала (VLF). VLF не имеет фиксированного размера. Не существует также и определенного числа VLF, приходящихся на один физический файл журнала. Компонент Database Engine динамически определяет размер VLF при создании или расширении файлов журнала. Компонент Database Engine стремится обслуживать небольшое число VLF. Администраторы не могут настраивать или устанавливать размеры и число VLF.

25 Контрольные точки, активная часть журнала и усечение журнала транзакций

Так как все изменения страниц данных происходят в страничных буферах, то изменения данных в памяти не обязательно отражаются в этих страницах на диске. Процесс кэширования происходит по алгоритму последней использованной страницы, поэтому страница, подверженная постоянным изменениям, помечается как последняя использованная, и она не записывается его на диск. Чтобы эти страницы были записаны на диск применяется контрольная точка. Все грязные страницы должны быть сохранены на диске в обязательном порядке.

Контрольная точка выполняет в базе данных следующее:

- Записывает в файл журнала запись, отмечающую начало контрольной точки.
- Сохраняет данные, записанные для контрольной точки в цепи записей журнала контрольной точки. Одним из элементов данных, регистрируемых в записях контрольной точки, является номер LSN первой записи журнала, при отсутствии которой успешный откат в масштабе всей базы данных невозможен. Такой номер LSN называется минимальным номером LSN восстановления (MinLSN). Номер MinLSN является наименьшим значением из:
  - номера LSN начала контрольной точки;
  - номера LSN начала старейшей активной транзакции;
  - номера LSN начала старейшей транзакции репликации, которая еще не была доставлена базе данных распространителя.
  - Записи контрольной точки содержат также список активных транзакций, изменивших базу данных.
- Если база данных использует простую модель восстановления, помечает для повторного использования пространство, предшествующее номеру MinLSN.
- Записывает все измененные страницы журналов и данных на диск.
- Записывает в файл журнала запись, отмечающую конец контрольной точки.
- Записывает в страницу загрузки базы данных номер LSN начала соответствующей цепи.

Активный журнал

Часть журнала, начинающаяся с номера MinLSN и заканчивающаяся последней записью, называется активной частью журнала, или активным журналом. Этот раздел журнала необходим для выполнения полного восстановления базы данных. Ни одна часть активного журнала не может быть усечена. Все записи журнала до номера MinLSN должны быть удалены из частей журнала.

Как работает усечение журнала

Кроме прочих данных, в контрольной точке записывается номер LSN первой записи журнала, которую необходимо сохранить для успешного отката на уровне базы данных. Этот номер LSN называется минимальным номером LSN восстановления (MinLSN). Начало активной части журнала занято VLF, содержащим MinLSN. При усечении журнала транзакций освобождаются только те записи, которые находятся перед этим VLF.

26 Управление участниками системы безопасности

Участники системы безопасности или принципалы – это сущности, которые могут запрашивать ресурсы SQL Server. Принципалы могут быть иерархически упорядочены. Область влияния принципала зависит

- от области его определения: Windows, SQL Server, база данных,
- от того, коллективный это участник или индивидуальный. Имя входа Windows является примером индивидуального(неделимого) участника, а группа Windows — коллективного.

При создании учетной записи SQL Server ей назначается идентификатор и идентификатор безопасности. В представлении каталога sys.server_principals они отображаются в столбцах principal_id и SID.

Участники уровня Windows – это a) Имя входа домена Windows, b) Локальное имя входа Windows.

Участники уровня SQL Server – это a) Имя входа SQL Server, b) Роль сервера.

Участники уровня базы данных – это a) Пользователь базы данных, b) Роль базы данных, c) Роль приложения.

Замечания.

1. Имя входа «sa» SQL Server. Имя входа sa SQL Server является участником уровня сервера. Оно создается по умолчанию при установке экземпляра. В SQL Server для имени входа sa базой данных по умолчанию будет master.
2. Роль базы данных public. Каждый пользователь базы данных является членом роли базы данных public. Если пользователю не были предоставлены или запрещены особые разрешения на защищаемый объект, то он наследует на него разрешения роли public.
3. Пользователи INFORMATION_SCHEMA и sys. Каждая база данных включает в себя две сущности, которые отображены в представлениях каталога в виде пользователей: INFORMATION_SCHEMA и sys. Они необходимы для работы SQL Server; эти пользователи не являются участниками и не могут быть изменены или удалены.

27 Управление разрешениями

У субъекта системы есть только один путь получения доступа к объектам - иметь назначенные непосредственно или опосредовано разрешения. При непосредственном управлении разрешениями они назначаются субъекту явно, а при опосредованном разрешения назначаются через членство в группах, ролях или наследуются от объектов, лежащих выше по цепочке иерархии. Управление разрешениями производится путем выполнения инструкций языка DCL (Data Control Language): GRANT (разрешить), DENY (запретить) и REVOKE (отменить).

Предоставление разрешений на объекты (инструкция GRANT)

Инструкция GRANT предоставляет разрешения на таблицу, представление, функцию, хранимую процедуру, очередь обслуживания, синоним.

Синтаксис инструкции GRANT:
GRANT ALL [PRIVILEGES] | список_разрешений
ON список_объектов
TO список_принципалов
[WITH GRANT OPTION]
[AS принципал]

Отмена разрешений на объекты (инструкция REVOKE)

Инструкция REVOKE отменяет разрешения, ранее предоставленные инструкцией GRANT.
Синтаксис инструкции REVOKE:
REVOKE [GRANT OPTION FOR] список_разрешений
ON список_объектов
FROM | TO список_принципалов
[CASCADE]
[AS принципал]

Запрет разрешений на объекты (инструкция DENY)

Инструкция DENY запрещает разрешения на члены класса OBJECT защищаемых объектов.
Синтаксис инструкции DENY:
DENY список_разрешений
ON список_объектов
TO список_принципалов
[CASCADE]
[AS принципал]