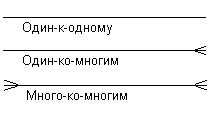
**Семантическое моделирование данных: ER-диаграммы**

В реальном проектировании структуры базы данных применяется метод семантического моделирования. Семантическое моделирование представляет собой моделирование структуры данных, опираясь на смысл этих данных. В качестве инструмента семантического моделирования используются различные варианты диаграмм сущность-связь (ER - Entity-Relationship). | 1) Сущность - это класс однотипных объектов, информация о которых должна быть учтена в модели. Каждая сущность должна иметь наименование, выраженное существительным в единственном числе. Примерами сущностей могут быть такие классы объектов как "Поставщик", "Сотрудник", "Накладная". Каждая сущность в модели изображается в виде прямоугольника с наименованием; 2) Экземпляр сущности- это конкретный представитель данной сущности. Например, представителем сущности "Сотрудник" может быть "Сотрудник Иванов". Экземпляры сущностей должны быть различимы, т.е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.; 3) Атрибут сущности - это именованная характеристика, являющаяся некоторым свойством сущности. Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными). Примерами атрибутов сущности "Сотрудник" могут быть такие атрибуты как "Табельный номер", "Фамилия", "Имя", "Отчество", "Должность", "Зарплата" и т.п. Атрибуты изображаются в пределах прямоугольника, определяющего сущность; 4) Ключ сущности- это неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточность заключается в том, что удаление любого атрибута из ключа нарушается его уникальность. Сущность может иметь несколько различных ключей. Ключевые атрибуты изображаются на диаграмме подчеркиванием; 5) Связь - это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою. Связи позволяют по одной сущности находить другие сущности, связанные с нею. Например, связи между сущностями могут выражаться следующими фразами - "СОТРУДНИК может иметь несколько ДЕТЕЙ", "каждый СОТРУДНИК обязан числиться ровно в одном ОТДЕЛЕ". Графически связь изображается линией, соединяющей две сущности. Каждая связь имеет два конца и одно или два наименования. Наименование обычно выражается в неопределенной глагольной форме: "иметь", "принадлежать" и т.п. Каждое из наименований относится к своему концу связи. Иногда наименования не пишутся ввиду их очевидности.   
  
 Каждая связь может иметь один из следующих типов связи:  


Связь типа один к одному означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один к одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две. Связь типа один ко многим означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи. Левая сущность (со стороны "один") называется родительской, правая (со стороны "много") - дочерней. Связь типа многие ко многим означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.

**Реляционная модель данных: домены и отношения**

**Реляционная модель**: Наиболее распространенная трактовка реляционной модели данных принадлежит К.Дейту. Согласно Дейту, реляционная модель состоит из трех частей: Структурной части, Целостной части, Манипуляционной части. Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения. Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей. Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными - реляционную алгебру и реляционное исчисление. |  
**Домен:** Домен - это семантическое понятие. Домен можно рассматривать как подмножество значений некоторого типа данных имеющих определенный смысл. Домен характеризуется следующими свойствами: Домен имеет уникальное имя (в пределах базы данных); Домен определен на некотором простом типе данных или на другом домене; Домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена; Домен несет определенную смысловую нагрузку. |

**Отношение:** Пусть дана совокупность типов данных T1, T2, ..., Tn, называемых также доменами, не обязательно различных. Тогда n-арным отношением R, или отношением R степени n называют подмножество декартовa произведения множеств T1, T2, ..., Tn. Отношение R состоит из заголовка (схемы) и тела. Заголовок представляет собой множество атрибутов (именованных вхождений домена в заголовок отношения), а тело — множество кортежей, соответствующих заголовку. Количество кортежей называют кардинальным числом отношения (кардинальностью), или мощностью отношения. Количество атрибутов называют степенью, или «арностью» отношения. Основные свойства отношения:В отношении нет двух одинаковых элементов (кортежей); Порядок кортежей в отношении не определён; Порядок атрибутов в заголовке отношения не определён; Все значения атрибутов атомарны. Подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости), называется потенциальным ключом. Поскольку все кортежи в отношении по определению уникальны, в любом отношении должен существовать по крайней мере один потенциальный ключ. Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

**Реляционная модель данных: целостность**

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей. Кодд предложил два декларативных механизма поддержки целостности реляционных баз данных, которые затверждены в реляционной модели данных и должны поддерживаться в любой реализующей ее СУБД: ограничение целостности сущности, или ограничение первичного ключа и ограничение ссылочной целостности, или ограничение внешнего ключа. Ограничение целостности сущности: не должно быть двух одинаковых кортежей. ограничение ссылочной целостности: если атрибуты внутри отношения как-то связаны, то эта связь определяется функциональными связями. Если же связаны отношения, то выделяют понятие родительского отношения с потенциальным ключом (который может быть первичным) и дочернего отношения с копией потенциального ключа родительского. Таким образом, если есть связанные отношения и внешний ключ, то он должен ссылаться на существующий кортеж. |

**Первичный ключ**: Первичный ключ — в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию). Если в отношении имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют «альтернативными». |

**Потенциальный ключ:** Потенциальный ключ — в реляционной модели данных — подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).Уникальность означает, что нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны). Свойство уникальности определяется не для конкретного значения переменной отношения в тот или иной момент времени, а по всем возможным значениям, то есть следует из внешнего знания о природе и закономерностях данных, которые могут находиться в переменной отношения. Минимальность (несократимость) означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности. Поскольку все кортежи в отношении по определению уникальны, в нём всегда существует хотя бы один потенциальный ключ (например, включающий все атрибуты отношения). В отношении может быть одновременно несколько потенциальных ключей. Один из них может быть выбран в качестве первичного ключа отношения, тогда другие потенциальные ключи называют альтернативными ключами. |

**Внешний ключ:** Пусть R1 и R2 — две переменные отношения, не обязательно различные. Внешним ключом FK в R2 является подмножество атрибутов переменной R2 такое, что выполняются следующие требования: В переменной отношения R1 имеется потенциальный ключ CK такой, что FK и CK совпадают с точностью до переименования атрибутов (то есть переименованием некоторого подмножества атрибутов FK можно получить такое подмножество атрибутов FK’, что FK’ и CK совпадают как по именами, так и по типам атрибутов). В любой момент времени каждое значение FK в текущем значении R2 идентично значению CK в некотором кортеже в текущем значении R1. Иными словами, в каждый момент времени множество всех значений FK в R2 является (нестрогим) подмножеством значений CK в R1. При этом для данного конкретного внешнего ключа FK → CK отношение R1, содержащее потенциальный ключ, называют главным, целевым, или родительским отношением, а отношение R2, содержащее внешний ключ, называют подчинённым, или дочерним отношением.

**Реляционная модель данных: реляционная алгебра**

Реляционная алгебра — замкнутая система операций над отношениями в реляционной модели данных. Операции реляционной алгебры также называют реляционными операциями.| **Переименование R RENAME Atr1, Atr2, … AS NewAtr1, NewAtr2, …** В результате применения операции переименования получаем новое отношение, с измененными именами атрибутов|

**Объединение A UNION B** Отношение с тем же заголовком, что и у совместимых по типу отношений A и B, и телом, состоящим из кортежей, принадлежащих или A, или B, или обоим отношениям. |

**Пересечение A INTERSECT B** Отношение с тем же заголовком, что и у отношений A и B, и телом, состоящим из кортежей, принадлежащих одновременно обоим отношениям A и B. |

**Вычитание A MINUS B** Отношение с тем же заголовком, что и у совместимых по типу отношений A и B, и телом, состоящим из кортежей, принадлежащих отношению A и не принадлежащих отношению B. |

**Декартово произведение A TIMES B** Отношение (A1, A2, …, Am, B1, B2, …, Bm), заголовок которого является сцеплением заголовков отношений A(A1, A2, …, Am) и B(B1, B2, …, Bm), а тело состоит из кортежей, являющихся сцеплением кортежей отношений A и B: (a1, a2, …, am, b1, b2, …, bm) таких, что (a1, a2, …, am)∈ A, (b1, b2, …, bm)∈ B. |

**Выборка (ограничение) A WHERE c** Отношение с тем же заголовком, что и у отношения A, и телом, состоящим из кортежей, значения атрибутов которых при подстановке в условие c дают значение ИСТИНА. cпредставляет собой логическое выражение, в которое могут входить атрибуты отношения A и/или скалярные выражения. |

**Проекция PROJECT A {x, y, …, z}** Отношение с заголовком (X, Y, …, Z) и телом, содержащим множество кортежей вида (x, y, …, z), таких, для которых в отношении A найдутся кортежи со значением атрибута X равным x, значением атрибута Y равным y, …, значением атрибута Z равным z. При выполнении проекции выделяется «вертикальная» вырезка отношения-операнда с естественным уничтожением потенциально возникающих кортежей-дубликатов. |

**Соединение A JOIN c** Операция соединения есть результат последовательного применения операций декартового произведения и выборки. Если в отношениях имеются атрибуты с одинаковыми наименованиями, то перед выполнением соединения такие атрибуты необходимо переименовать. |

**Деление A DIVIDEBY B** Отношение с заголовком (X1, X2, …, Xn) и телом, содержащим множество кортежей (x1, x2, …, xn), таких, что для всех кортежей (y1, y2, …, ym) ∈ B в отношении A(X1, X2, …, Xn, Y1, Y2, …, Ym) найдется кортеж (x1, x2, …, xn, y1, y2, …, ym).

**Реляционная модель данных: реляционное исчисление кортежей**

Задача – нахождение кортежей, для которых предикат является истинным. В построении формул используются переменные кортежа, областью определения которых является отношение, что записывается в виде RANGE OF S IS Divers (областью определения переменной S является отношение Divers). Выражение "множество всех кортежей, для которых предикат P(S) является истинным" записывается как {S|P(S)}. При построении формул для указания количества экземпляров, к которым должен быть применен предикат, используются кванторы общности (ꓯ) и существования (ꓱ). | Правила построения формул: • Если P является n-арной формулой, а t1, t2, …, tn – константы или переменные, то P(t1, t2, …, tn) – правильная формула; • Если t1 и t2 являются константами или переменными из одного домена, а (\*) - один из операторов сравнения (<, <=, >, >=, =, !=), то t1(\*)t2 – правильная формула; • Конъюнкция, дизъюнкция и отрицание формул являются формулами; • Если F1 – выражение со свободной переменной X, то выражения ꓱF(X) и ꓯF(X) – также формулы.   


**Проектирование реляционных баз данных на основе принципов нормализации: функциональные зависимости**

Функциональная зависимость – А функционально определяет В (В функционально зависит от В) тогда и только тогда, когда для любых двух кортежей T1,T2 отношение r над схемой R из равенства t1[A]=t2[A] следует t1[B]=t2[B]. Функциональная зависимость - это связь типа многие к одному между двумя множествами атрибутов заданной переменной-отнощения. Для заданной переменной-отнощения R зависимость А -> В (где А и В являются подмножествами множества атрибутов переменной-отнощения R) выполняется для переменной-отнощения R тогда и только тогда, когда любые два кортежа переменной-отнощения R с одинаковыми значениями атрибутов множества А имеют одинаковые значения атрибутов множества В.|

Аксиомы Армстронга можно использовать при первой попытке получения множества функциональных зависимостей относительно заданного множества:

1) правило рефлексивности. Если B ⊆ A => A → B (→ функциональная зависимость);

2) дополнение. A → B => AC → BC, где С – новый атрибут в пределах данной схемы. АС, BC – объединение множеств;

3) транзитивность. A → B, B → C=> A → C. |

Существует доказательство этих правил из определения функциональных зависимостей. Набор правил Армстронга является полным – для заданного множества ФЗ минимальный набор может быть выведен только с помощью этих трех правил; является исчерпывающим – в результате применений этих ФЗ никакие дополнительные ФЗ не могут быть получены. Тем не менее, набор был дополнен:

4) самоопределение. A → A;

5) декомпозиция. A → BC => A → B, A → C;

6) объединение. A → B, A → C => A → BC;

7) композиция. A → B, C → D => AB → CD;

8) правило унификации (общая теорема определения). A → B, C → D => A(C-B) → CD |

Применяя избыточные правила, можно получить замыкание заданного множества функциональных зависимостей. Неприводимое (минимальное в нормальной форме) покрытие : правая часть всегда одноэлементна; левая часть неприводима; нет лишних зависимостей (минимальное – объединяем зависимости с одинаковой левой частью).

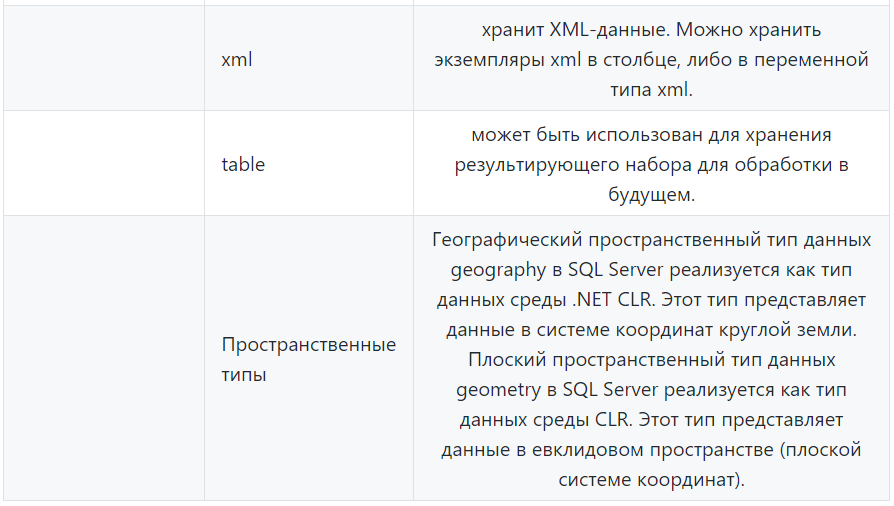
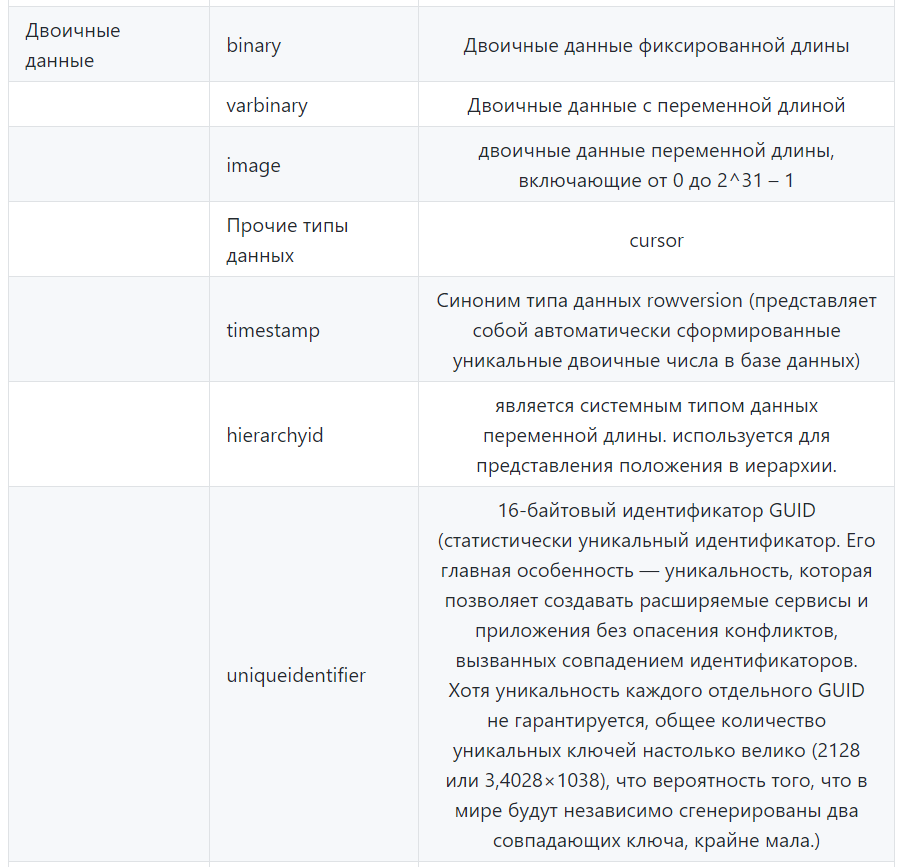
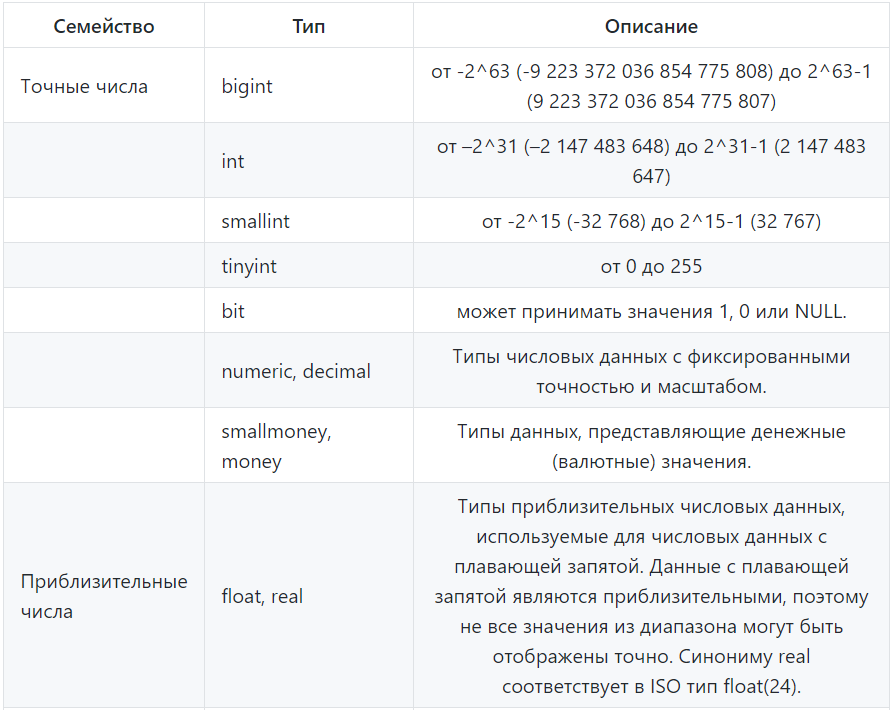
**Проектирование реляционных баз данных на основе принципов нормализации: 1NF, 2NF, 3NF, BCNF**

**Первая нормальная форма (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. В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ. |

**Вторая нормальная форма (2NF):** Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме, и каждый неключевой атрибут неприводимо (функционально полно) зависит от ее потенциального ключа. |

**Третья нормальная форма (3NF):** Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых. |

**Нормальная форма Бойса — Кодда (BCNF):** Переменная отношения находится в нормальной форме Бойса — Кодда (иначе — в усиленной третьей нормальной форме) тогда и только тогда, когда каждая ее нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ. | В 1НФ каждый атрибут атомарен . Пусть в отношении имеется набор атрибутов, которые являются потенциальным ключом, и этот набор единственный – одновременно первичный ключ. Тогда каждый неключевой атрибут должен зависеть от первичного ключа – но зависеть он может только от части ключа. Тогда это отношение находится в 1НФ, но не в 2НФ – найдутся неключевые атрибуты, зависящие от части ключа. Пусть отношение находится в 2НФ – каждый неключевой атрибут ПОЛНОСТЬЮ зависит от ключевого. Если в отношении наблюдаются транзитивные функциональные зависимости, то оно не находится в 3НФ.

**Системы типов данных языка SQL и T SQL**   


**Приведение и преобразования типа Transact SQL**

Преобразование типов данных бывает явным и неявным. Неявное преобразование скрыто от пользователя. SQL Server автоматически преобразует данные из одного типа в другой. Например, если тип данных smallint сравнивается с типом int, то перед сравнением тип smallint неявно преобразуется в тип int. GETDATE() выполняет неявное преобразование в стиль даты 0. SYSDATETIME() выполняет неявное преобразование в стиль даты 21. Явное преобразование выполняется с помощью функций CAST и CONVERT. Функции CAST и CONVERT преобразуют значение (локальную переменную, столбец или выражение) из одного типа данных в другой. Например, приведенная ниже функция CAST преобразует числовое значение $157.27 в строку символов '157.27'. Некоторые виды явного и неявного преобразования типов данных не поддерживаются при преобразовании типа данных одного объекта SQL Server в тип данных другого объекта. Например, значение типа nchar нельзя преобразовать в значение типа image. Тип данных nchar можно преобразовать только в тип данных binary, причем только явно. Неявное преобразование в binary не поддерживается. Однако тип данных nchar можно преобразовать в тип nvarchar как явно, так и неявно.

**Представления T SQL**

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 не дающего им разрешений на непосредственный доступ к базовым таблицам. |

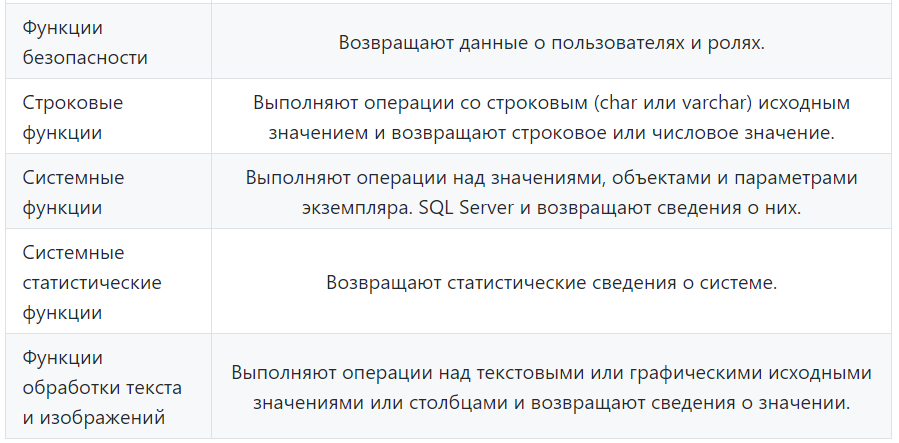
Создание представления: Для создания представления данных, содержащихся в одной или более таблицах базы данных, необходимо использовать инструкцию CREATE VIEW, которая должна быть первой в пакетном запросе. |

Базовый синтаксис: *CREATE VIEW [ имя-схемы . ] имя [ (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.) |

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. )

**Скалярные выражения Transact SQL**

Скалярное выражение – это выражение, вырабатывающее результат некоторого типа, специфицированного в стандарте. Скалярные выражения являются основой языка SQL, поскольку, хотя это реляционный язык, все условия, элементы списков выборки и т. д. базируются именно на скалярных выражениях. В SQL:1999 имеется несколько разновидностей скалярных выражений. К числу наиболее важных разновидностей относятся численные выражения; выражения со значениями-строками символов; выражения со значениями даты-времени; выражения со значениями-временными интервалами; булевские выражения. |

Пользовательские скалярные функции возвращают одно значение типа данных, заданного в предложении RETURNS. Встроенная скалярная функция не имеет тела, скалярное значение является результатом одной инструкции. Скалярная функция из нескольких инструкций имеет текст, ограниченное блоком BEGIN...END, и содержит последовательность инструкций.  
  


**Логические выражения Transact SQL**

Логические операторы проверяют истину некоторого условия. Логические операторы, например оператор сравнения, возвращают значение типа Boolean: TRUE, FALSE или UNKNOWN

**Табличные выражения Transact SQL**

Табличное выражение предоставляет некую таблицу. Табличное выражение содержит предложение FROM, за которым необязательно следуют предложения WHERE, GROUP BY и HAVING.|

GROUP BY используется для группировки вместе тех строк таблицы, которые имеют те же самые значения во всех перечисленных колонках. Порядок, в котором перечисляются колоки значения не имеет. Эффект группировки состоит в комбинировании каждого списка строк, имеющих общие значения колонок в одну сгруппированную строку, которая представляет все строки в данной группе. Это осуществляется исключением избыточности в выводе и/или подсчётом агрегатов, которые применяются к группам. Группировка без агрегирующих выражений эффективно подсчитывает список отдельных значений в колонке. Это же может быть достигнуто, используя предложение DISTINCT. Если таблица была сгруппирована с помощью предложения GROUP BY, но интерес представляют только определённые группы, можно использовать предложение HAVING, которое очень похоже на предложение WHERE, но только для удаления групп из результата. |

Предложение FROM: FROM создаёт таблицу из одной или более других таблиц, которые задаются списком разделённых запятыми ссылок на таблицы: *FROM ссылка\_на\_таблицу [, ссылка\_на\_таблицу [, ...]]* |

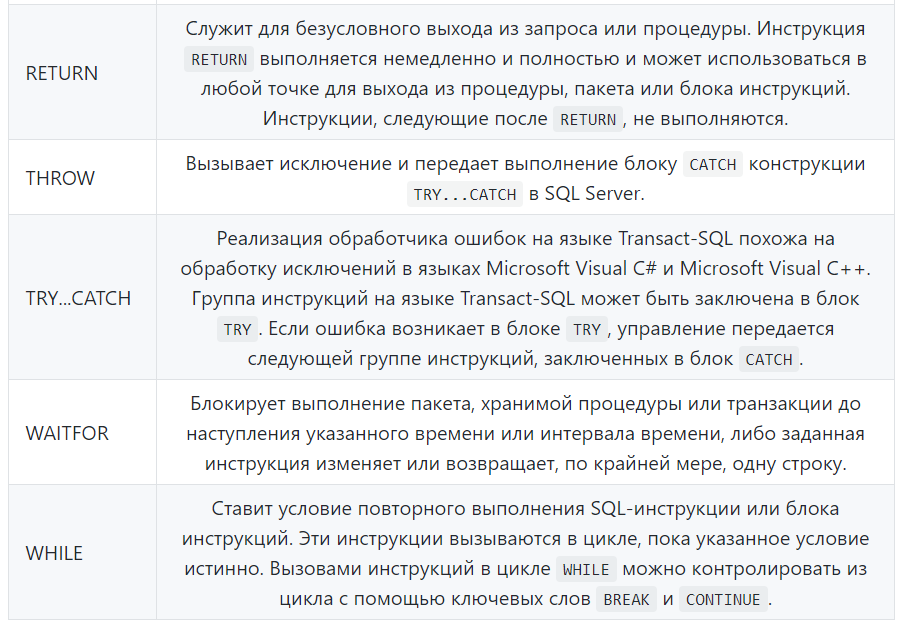
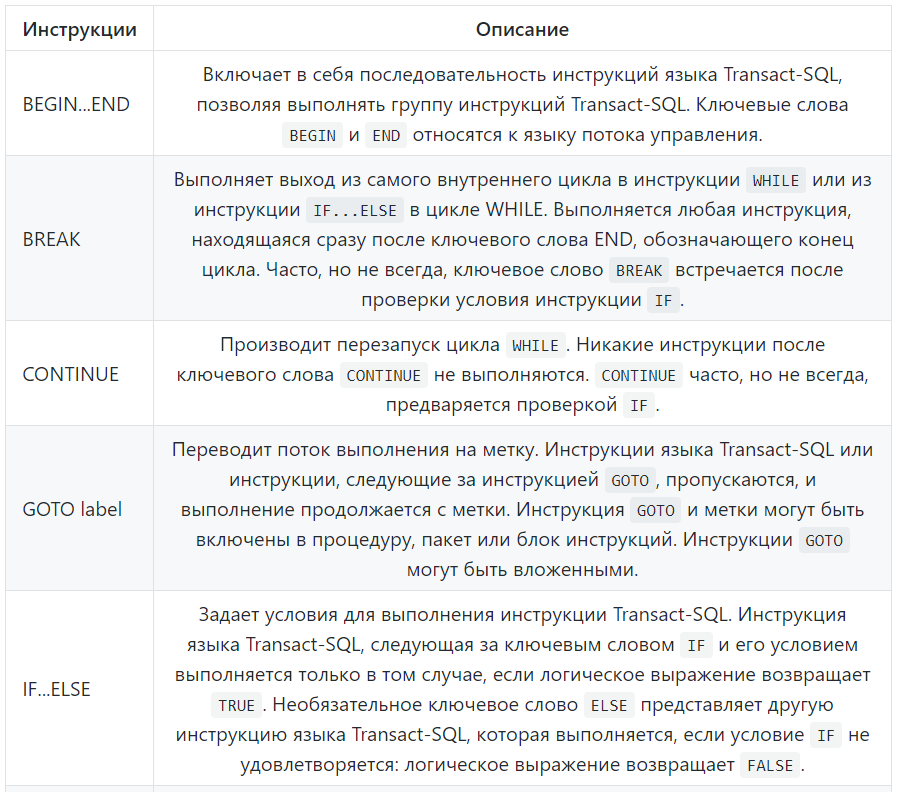
Соединённые таблицы: Соединённая таблица — это таблица, производная от двух других (реальных или в свою очередь производных) таблиц, полученая в соответствии с правилами определённого типа соединения. Доступные типы соединения: inner (внутреннее), outer (внешнее) и cross (перекрёсное). Слова INNER и OUTER являются необязательными во всех формах. INNER применяется по умолчанию; LEFT, RIGHT и FULL неявно указывают на внешнее (outer) соединение. Условие соединения задаётся в предложениях ON или USING, или неявно, с помощью слова NATURAL. Предложение ON является наиболее обычным способом задать условие соединения: в нём указывается логическое выражение того же типа, что используется в предложении WHERE. Пара строк из T1 и T2 считается совпавшей, если для неё, выражение, заданное в ON принимает значение истина. USING имеет более краткую форму: в ней указывается разделённый запятыми список имён колонок, которые должны быть общими в соединяемых таблицах и условия соединения, задавая равенство каждой из этих пар колонок. NATURAL — это краткая форма USING: она формирует список USING, содержащий все имена колонок, которые есть в обеих входных таблицах. Как и с USING, эти колонки появляются в выходной таблице только один раз. Если общих колонок нет, NATURAL ведёт себя как CROSS JOIN. |

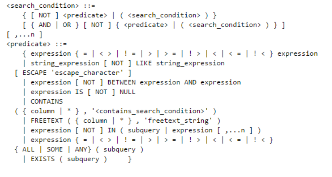
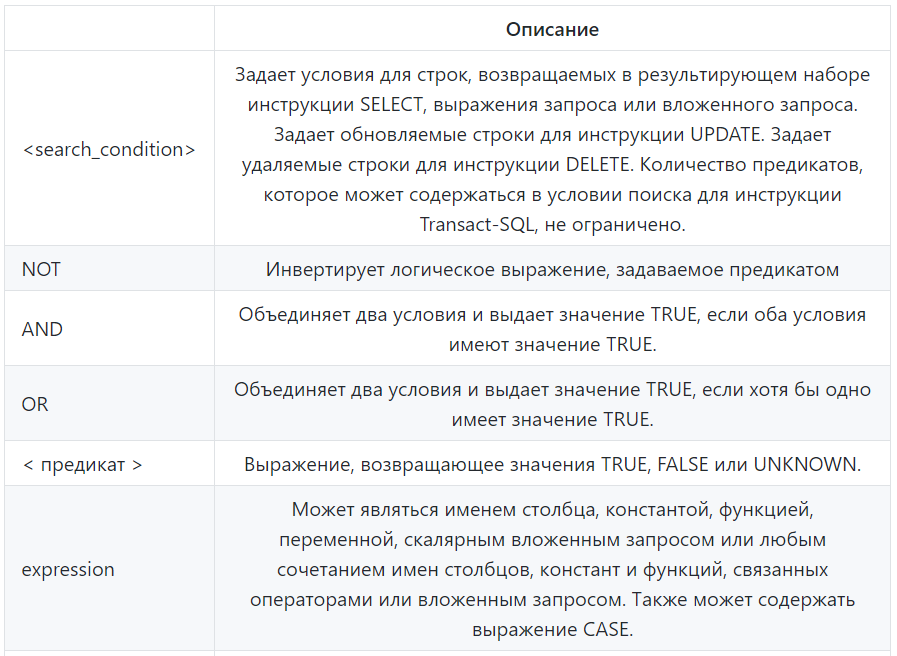
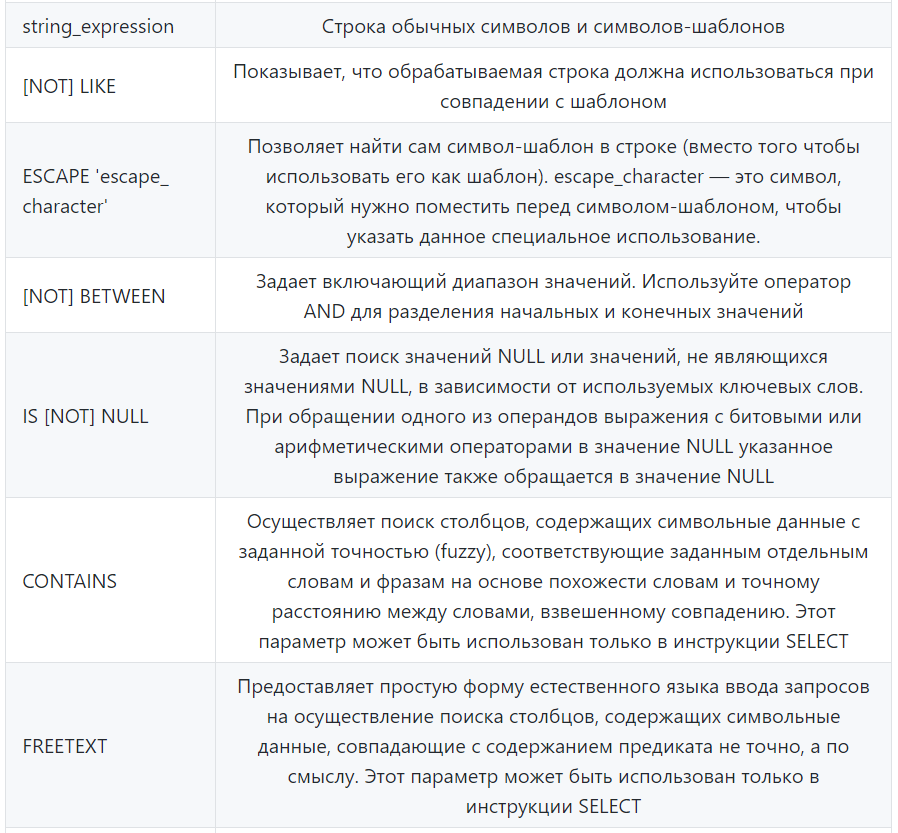
Псевдонимы таблиц и колонок: В сложных ссылках на таблицы, для дальнейших ссылок в остальной части запроса, могут быть использованы временные имена. Они называются псевдонимами таблиц.  
*FROM ссылка\_на\_таблицу AS псевдоним*Ключевое слово AS является необязательным. Псевдоним может быть любым идентификатором. Псевдоним даёт новое имя ссылающееся на ту же таблицу в текущем запросе — поэтому где-либо ещё в текущем запросе не разрешается ссылаться на таблицу по первоначальному имени. Псевдноним также требуется, если ссылка на таблицу является подзапросом.

**Обобщенные табличные выражения Transact SQL**

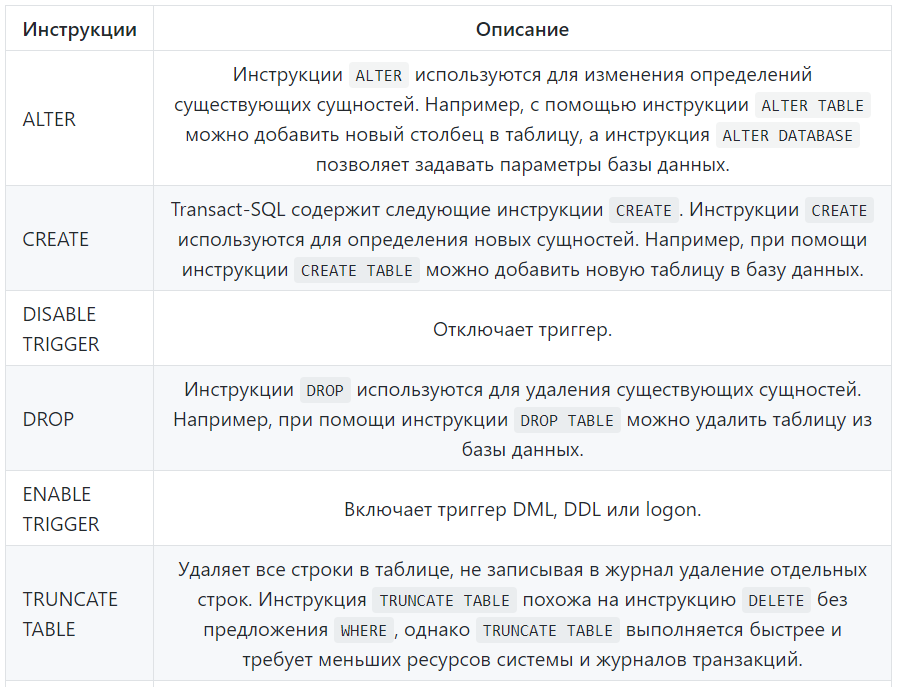
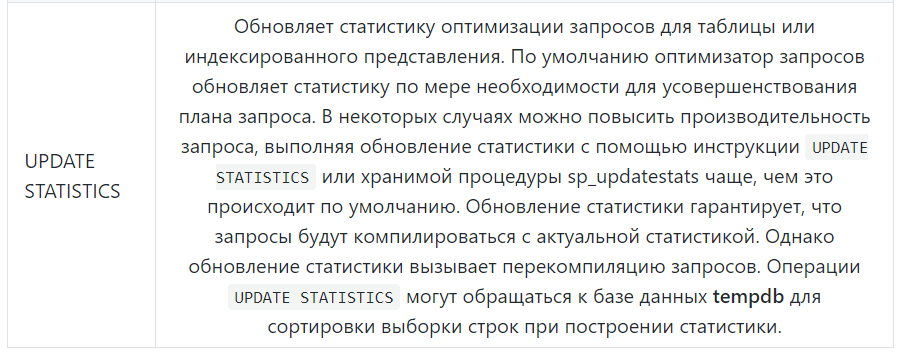
Обобщенные табличные выражения (ОТВ) можно представить себе как временные результирующие наборы, определенные в области выполнения единичных инструкций SELECT, INSERT, UPDATE, DELETE или CREATE VIEW. ОТВ, как и производные таблицы, не сохраняются в базе данных в виде объектов, время их жизни ограничено продолжительностью запроса. Но, в отличие от производных таблиц, ОТВ могут ссылаться сами на себя, а на них один и тот же запрос может ссылаться несколько раз. В основном ОТВ предназначены для: Создания рекурсивных запросов; Замены представлений в тех случаях, когда нет необходимости сохранять в метаданных базы его определение; Многократных ссылок на результирующую таблицу из одной и той же инструкции. ОТВ могут быть определены в пользовательских подпрограммах (функциях, хранимых процедурах, триггерах, представлениях). Базовый синтаксиса ОТВ: *WITH имя\_ОТВ [ ( список\_имен\_столбцов ) ] AS ( определение\_ОТВ )* Список имен столбцов необязателен только в том случае, если всем результирующим столбцам в определении запроса присвоены уникальные имена.|

Структура рекурсивного ОТВ: Рекурсивное ОТВ должно содержать минимум один закрепленный элемент и один рекурсивный элемент. Следующий псевдокод отображает компоненты простого рекурсивного ОТВ, которое содержит один закрепленный элемент и один рекурсивный элемент.  
*WITH имя\_ОТВ [(список\_имен\_столбцов )]  
AS  
(  
-- Определение закрепленного элемента  
UNION ALL  
-- Определение рекурсивного элемента  
)-- Инструкция, использующая ОТВ  
SELECT {\*|список\_имен\_столбцов} FROM имя\_ОТВ*

**Инструкции языка управления потоком Transact SQL**

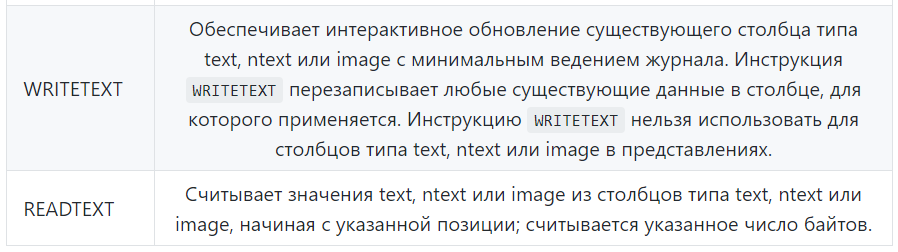
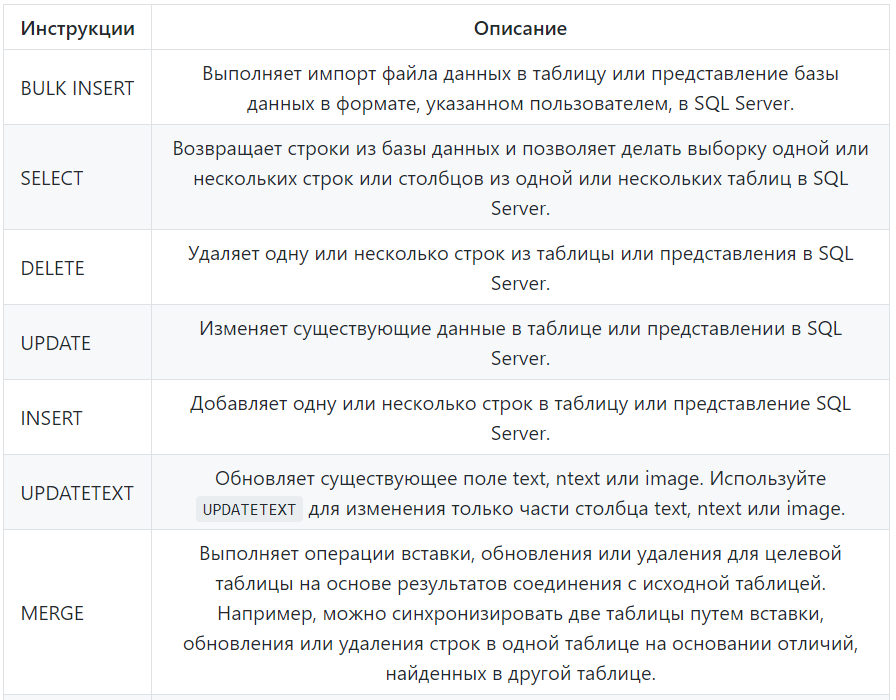
**Условие поиска T SQL**WHERE определяет условия поиска строк, возвращаемых запросом. Приоритеты выполнения логических операторов распределяются следующим образом: NOT (наивысший приоритет), AND, OR (низший приоритет). Для перераспределения указанных приоритетов в условии поиска используются скобки. Порядок выполнения логических операторов может меняться в зависимости от настроек оптимизатора запросов.  
  
  
  
  
**Сценарии и пакеты Transact SQL**

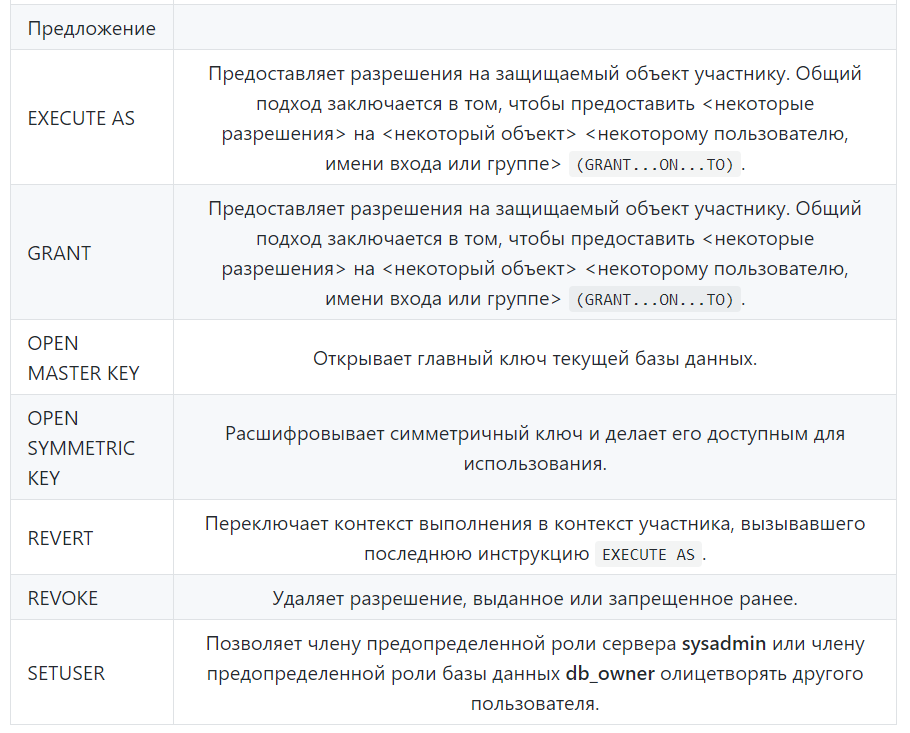
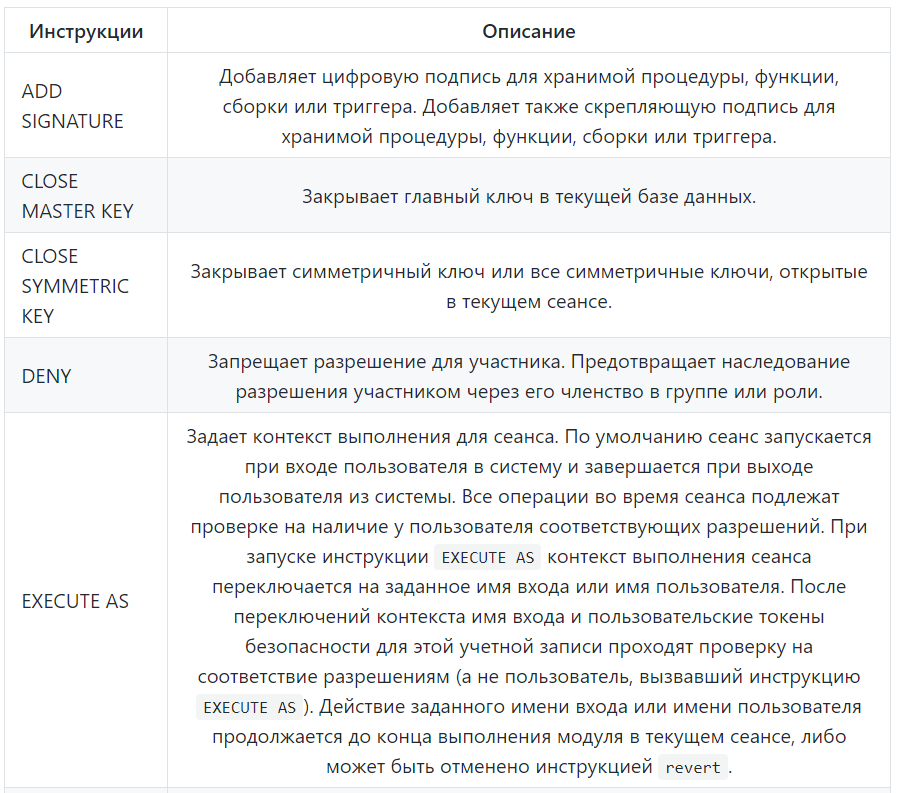
Сценарий — это серия инструкций языка Transact-SQL, которая хранится в файле. Данные из этого файла можно использовать в качестве исходных для редактора кода среды Среда SQL Server Management Studio или программ sqlcmd и osql. Эти программы будут выполнять инструкции SQL из файла. Сценарии языка Transact-SQL содержат один или несколько пакетов. Команда GO означает конец пакета. Если сценарий языка Transact-SQL не содержит команд GO, то он выполняется как единый пакет. Сценарии языка Transact-SQL можно использовать следующим образом: 1) Сохраните как механизм резервной копии постоянную копию шагов, которые выполнялись при создании и заполнении базы данных на сервере; 2)Перенесите инструкции с одного компьютера на другой, когда появится соответствующее приглашение; 3)Быстро обучите новых работников разбираться в коде, изменять код и находить в нем ошибки.

**Инструкции языка описания данных Transact SQL**   
  


**Ограничения целостности Transact SQL**

DBCC CHECKCONSTRAINTS: Проверяет целостность указанного ограничения или всех ограничений заданной таблицы в текущей базе данных.  
*DBCC CHECKCONSTRAINTS  
[   
 (   
 table\_name | table\_id | constraint\_name | constraint\_id  
 )  
]  
 [ WITH   
 [ { ALL\_CONSTRAINTS | ALL\_ERRORMSGS } ]  
 [ , ] [ NO\_INFOMSGS ]   
 ]**table\_name | table\_id | constraint\_name | constraint\_id* (Проверяемая таблица или ограничение. Если указан аргумент table\_name или table\_id, проверяются все включенные ограничения на данной таблице. Если указан аргумент constraint\_name или constraint\_id, проверяется только указанное ограничение. Если не указаны ни идентификатор таблицы, ни идентификатор ограничения, проверяются все включенные ограничения всех таблиц в текущей базе данных. Имя ограничения однозначно определяет таблицу, к которой оно принадлежит. Дополнительные сведения см. в разделе Идентификаторы баз данных.)   
*WITH* (Позволяет задавать параметры.)   
*ALL\_CONSTRAINTS* (Проверяет все включенные и отключенные ограничения таблицы, если указано имя таблицы или отмечены все таблицы. В противном случае проверяет только включенные ограничения. ALL\_CONSTRAINTS не работает, если указано имя ограничения.)  
*ALL\_ERRORMSGS* (Возвращает все строки, которые не соответствуют ограничениям в проверяемой таблице. По умолчанию, это первые 200 строк.)  
*NO\_INFOMSGS* (Подавляет вывод всех информационных сообщений.)

**Инструкции языка обработки данных Transact SQL**   


**Инструкции языка контроля доступа к данным Transact-SQL. Инструкции безопасности T-SQL**

**Инструкции управления транзакциями T SQL**

Транзакция является единственной единицей работы. Если транзакция выполнена успешно, все модификации данных, сделанные в течение транзакции, принимаются и становятся постоянной частью базы данных. Если в результате выполнения транзакции происходят ошибки и должна быть произведена отмена или выполнен откат, все модификации данных будут отменены. SQL Server работает в следующих режимах транзакций:

1)Автоматическое принятие транзакций. Каждая отдельная инструкция является транзакцией;

2)Явные транзакции. Каждая транзакция явно начинается с инструкции BEGIN TRANSACTION и явно заканчивается инструкцией COMMIT или ROLLBACK;

3)Неявные транзакции. Новая транзакция неявно начинается, когда предыдущая транзакция завершена, но каждая транзакция явно завершается инструкцией COMMIT или ROLLBACK;

4)Транзакции контекста пакета. Будучи применимой только к множественным активным результирующим наборам (MARS), явная или неявная транзакция Transact-SQL, которая запускается в сеансе MARS, становится транзакцией контекста пакета. SQL Server автоматически выполняет откат транзакции контекста пакета, если эта транзакция не зафиксирована или выполнен ее откат при завершении пакета.  


**Инструкции SET T SQL***SET @local\_variable*SET используется для устанавки указанной локальной переменной, предварительно созданную с помощью инструкции DECLARE @local\_variable, в указанное значение.  
Язык Transact-SQL предоставляет несколько инструкций SET, которые изменяют текущий сеанс, управляя специфическими данными. Инструкции SET группируются в категории, показанные в следующей таблице.  
Следующий пример создает переменную @myvar, записывает строковое значение в переменную и печатает значение переменной @myvar.  
*DECLARE @myvar char(20);   
SET @myvar = 'This is a test';   
SELECT @myvar;   
GO*

**Хранимые процедуры Transact SQL**

Хранимая процедура является группой инструкций языка Transact-SQL, которая была один раз откомпилирована, и может выполняться много раз. Это увеличивает производительность при выполнении процедуры, так как инструкции языка Transact-SQL не нужно повторно компилировать. Инструкции CREATE PROCEDURE (или CREATE TRIGGER) не могут выходить за пределы одного пакета. Это означает, что хранимая процедура (или триггер) всегда создаются в едином пакете и компилируются в план выполнения. План выполнения создается при первом выполнении хранимой процедуры (или триггера). Хранимые процедуры в аналогичны процедурам в других языках программирования:

1) они обрабатывают входные параметры и возвращают вызывающей процедуре или пакету значения в виде выходных параметров;

2) они содержат программные инструкции, которые выполняют операции в базе данных, в том числе вызывающие другие процедуры;

3) они возвращают значение состояния вызывающей процедуре или пакету, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего). |

Для создания хранимой процедуры используется инструкция CREATE PROCEDURE, имеющая следующий синтаксис:  
*CREATE PROCEDURE [ имя-схемы. ] имя-процедуры [ список-объявлений- параметров ]  
[ WITH список-опций-процедуры ]  
[ FOR REPLICATION ]  
AS  
тело-процедуры [; ]*Замечания:

1) FOR REPLICATION указывает, что процедура создается для репликации;

2)Параметрами процедуры могут быть любые типы данных, за исключением table;

3)Тип данных cursor может быть использован только в качестве выходного параметра;

4) Одна процедура может вызывать другую. Уровень вложенности увеличивается на 1, когда начинается выполнение вызванной процедуры, и уменьшается на 1, когда вызванная процедура завершается. Уровень вложенности процедур может достигать 32. Текущий уровень вложенности процедур можно получить при помощи функции @@NESTLEVEL. |

Чтобы выполнить процедуру, надо использовать инструкцию EXECUTE или EXEC. Также можно выполнить процедуру без использования ключевого слова EXECUTE или EXEC, если процедура является первой инструкцией в пакете.

**Функции Transact SQL**

SQL Server содержит набор встроенных функций и предоставляет возможность создавать пользовательские функции (User Defined Function – UDF). 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. Функция является недетерминированной, если она может возвращать различные значения при одном и том же заданном входном значении. Пользовательские функции в зависимости от типа данных возвращаемых ими значений могут быть скалярными и табличными. Табличные пользовательские функции бывают двух типов: подставляемые и многооператорные. Скалярные пользовательские функции обычно используются в списке столбцов инструкции SELECT и в предложении WHERE. Табличные пользовательские функции обычно используются в предложении FROM, и их можно соединять с другими таблицами и представлениями. |

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 [ ; ]*

Синтаксис вызова скалярных функций:   
*имя-схемы.имя-функции ( [ список-параметров ] )  
Для скалярной функции можно использовать инструкцию EXECUTE или EXEC:  
EXECUTE @возвращаемое-значение = имя-функции ( [ список-параметров ] )*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 [ ; ]*

Синтаксис создания многооператорной табличной функции:  
*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 [ ; ]*

**Хранимые триггеры T-SQL**

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.  
Формальное определение триггера:  
*CREATE TRIGGER имя\_триггера  
ON {имя\_таблицы | имя\_представления}  
{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]  
AS выражения\_sql*Для создания триггера применяется выражение CREATE TRIGGER, после которого идет имя триггера. Как правило, имя триггера отражает тип операций и имя таблицы, над которой производится операция. Каждый триггер ассоциируется с определенной таблицей или представлением, имя которых указывается после слова ON. Затем устанавливается тип триггера. Мы можем использовать один из двух типов:

1)AFTER: выполняется после выполнения действия. Определяется только для таблиц;

2)INSTEAD OF: выполняется вместо действия (то есть по сути действие - добавление, изменение или удаление - вообще не выполняется). Определяется для таблиц и представлений. После типа триггера идет указание операции, для которой определяется триггер: INSERT, UPDATE или DELETE. Для триггера AFTER можно применять сразу для нескольких действий, например, UPDATE и INSERT. В этом случае операции указываются через запятую. Для триггера INSTEAD OF можно определить только одно действие. И затем после слова AS идет набор выражений SQL, которые собственно и составляют тело триггера.

**Триггеры DDL T SQL**

Триггеры DDL — это особый вид триггеров, которые срабатывают при выполнении инструкций языка описания данных DDL. Они могут применяться при выполнении административных задач (например, для аудита и регулирования операций в базе данных). Триггеры DDL, как и обычные триггеры, вызывают срабатывание хранимых процедур в ответ на событие. Однако в отличие от триггеров DML они не срабатывают в ответ на инструкции UPDATE, INSERT или DELETE для таблицы или представления. Вместо этого они срабатывают в ответ на разнообразные события языка определения данных (DDL). Эти события в основном соответствуют инструкциям языка Transact-SQL, начинающимся ключевыми словами CREATE, ALTER или DROP. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL. Триггеры DDL срабатывают только после выполнения соответствующих инструкций DDL.  
Пример:  
*CREATE TRIGGER safety  
ON DATABASE  
FOR DROP\_TABLE, ALTER\_TABLE  
AS  
 PRINT 'You must disable Trigger "safety" to drop or alter tables!'  
ROLLBACK ;*При разработке триггеров DDL учитывайте их следующие отличия от триггеров DDL. Триггеры DDL выполняются только после завершения инструкции Transact-SQL. Триггеры DDL нельзя использовать в качестве триггеров INSTEAD OF. Триггеры DDL не создают таблицы inserted и deleted. Сведения о событии, приведшем к срабатыванию триггера DDL, и последующих изменениях, выполненных триггером, можно получить при помощи функции EVENTDATA.

**Триггеры DML T SQL**

Триггеры DML — это хранимые процедуры особого типа, автоматически вступающие в силу, если происходит событие языка обработки данных DML, которое затрагивает таблицу или представление, определенное в триггере. События DML включают инструкции INSERT, UPDATE или DELETE. Триггеры DML могут использоваться для предписания бизнес- правил и правил целостности данных, выполнения запросов к другим таблицам и включения сложных инструкций Transact-SQL. Триггер и инструкция, при выполнении которой он срабатывает, считаются одной транзакцией, которую можно откатить назад внутри триггера. При обнаружении серьезной ошибки (например, нехватки места на диске) вся транзакция автоматически откатывается назад. |

Триггер AFTER: Триггеры AFTER выполняются после выполнения действий инструкции INSERT, UPDATE, MERGE или DELETE. Триггеры AFTER никогда не выполняются, если происходит нарушение ограничения, поэтому эти триггеры нельзя использовать для какой-либо обработки, которая могла бы предотвратить нарушение ограничения. Для каждой из операций INSERT, UPDATE или DELETE в указанной инструкции MERGE соответствующий триггер вызывается для каждой операции DML. |

Триггер INSTEAD OF: Триггеры INSTEAD OF переопределяют стандартные действия инструкции, вызывающей триггер. Поэтому они могут использоваться для проверки на наличие ошибок или проверки значений на одном или нескольких столбцах и выполнения дополнительных действий перед вставкой, обновлением или удалением одной или нескольких строк. |

Триггеры CLR:  
Триггер CLR может быть либо триггером AFTER, либо триггером INSTEAD OF. Вместо вызова хранимой процедуры на языке Transact-SQL триггер CLR вызывает один или несколько методов управляемого кода, являющихся членами сборки, созданной с помощью среды .NET Framework и загружены в SQL Server.

**Курсоры Transact SQL**

Операции в реляционной базе данных выполняются над множеством строк. Например, набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE инструкции. 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 являются расширением результирующих наборов, которые предоставляют такой механизм. Инструкции Microsoft SQL Server создают полный результирующий набор, но бывают случаи, когда результаты удобнее обрабатывать построчно. Открытие курсора на результирующем наборе делает возможной его построчную обработку. Можно назначить курсору переменную или параметр с типом данных cursor. Курсоры позволяют усовершенствовать обработку результатов:   
1) позиционируясь на отдельные строки результирующего набора;

2) получая одну или несколько строк от текущей позиции в результирующем наборе;

3) поддерживая изменение данных в строках в текущей позиции результирующего набора;

4)поддерживая разные уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе;

5)предоставляя инструкциям Transact-SQL в скриптах, хранимых процедурах и триггерах доступ к данным результирующего набора.

Операции с курсором поддерживаются в таких инструкциях: close, CREATE PROCEDURE, DEALLOCATE, DECLARE CURSOR DECLARE @локальная\_переменная, delete, FETCH, OPEN, UPDATE, SET  
Следующие системные функции и системные хранимые процедуры также поддерживают курсоры: @@CURSOR\_ROWS, CURSOR\_STATUS, @@FETCH\_STATUS, sp\_cursor\_list sp\_describe\_cursor, sp\_describe\_cursor\_columns, sp\_describe\_cursor\_tables

**Индексы Transact SQL**

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. |

Создание индекса: *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 ]*|

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, для таблицы может быть создан только один кластерный индекс. В SQL Server кластерный индекс является уникальным индексом по определению. |

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). |

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:   
1) поиск в некластерном индексе; 2)поиск в кластерном индексе. |

Индексы, создаваемые вместе с ограничениям: Такой тип индексов часто называют «связанными индексами». Связанные индексы создаются при добавлении одного из следующих двух типов ограничений:

1) ограничения первичного ключа (PRIMARY KEY);

2) ограничения уникальности (UNIQUE).

**Управление транзакциями в SQL Server**

Управление транзакциями в приложениях реализуется, главным образом, путем указания того, когда транзакция начинается и заканчивается. Это можно указать либо с помощью инструкций языка Transact-SQL, либо используя функции интерфейса прикладного программирования (API) для баз данных. В системе также должна быть возможность правильной обработки ошибок, прерывающих транзакцию до ее окончания. |

Явные транзакции: Явный запуск транзакции через функцию API или посредством инструкции языка Transact-SQL BEGIN TRANSACTION. Автоматическая фиксация транзакций- Режим по умолчанию для компонента Database Engine. Каждая отдельная инструкция языка Transact-SQL фиксируется после завершения. Нет необходимости указывать какие-либо инструкции для управления транзакциями. |

Неявные транзакции: Установка неявного режима транзакции либо через функцию API, либо через инструкцию языка Transact-SQL SET IMPLICIT\_TRANSACTIONS ON. Следующая инструкция автоматически запускает новую транзакцию. После завершения этой транзакции следующая инструкция языка Transact-SQL запускает новую транзакцию. Транзакции можно завершить инструкцией COMMIT или ROLLBACK |

COMMIT: Если транзакция выполнена успешно, ее следует зафиксировать. Инструкция COMMIT гарантирует, что все изменения в пределах данной транзакции стали постоянной частью базы данных. |

ROLLBACK: Если в транзакции произойдет ошибка или пользователь решит отменить транзакцию, следует выполнить ее откат. Инструкция ROLLBACK отменяет все изменения, сделанные в пределах транзакции, возвращая данные в то состояние, в котором они находились на начало транзакции. Инструкция ROLLBACK также освобождает удерживаемые транзакцией ресурсы.  
*BEGIN TRANSACTION;  
GO  
 DELETE FROM Example.sample  
WHERE ID = 17;  
GO  
COMMIT TRANSACTION;  
GO  
BEGIN TRAN @TransactionName  
 INSERT INTO ValueTable VALUES(1)  
 INSERT INTO ValueTable VALUES(2)  
ROLLBACK TRAN @TransactionName  
INSERT INTO ValueTable VALUES(3)  
INSERT INTO ValueTable VALUES(4)  
SELECT \* FROM ValueTable*

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

Изменение данных пользователями может оказывать влияние на других пользователей, считывающих или изменяющих эти же данные в этот же момент времени. В этом случае говорят, что пользователи получают параллельный доступ к этим данным. Если в системе хранения данных отсутствует управление параллелизмом, то могут наблюдаться следующие побочные эффекты. |

Потерянные обновления: Потерянные обновления возникают, когда две или более транзакций выбирают одну и ту же строку и изменяют ее на основании ее исходного значения. Каждая транзакция не знает о других транзакциях. Последнее обновление изменяет данные других транзакций, что приводит к потере данных. Этой проблемы можно избежать, если у одного редактора не будет доступа к этому файлу до завершения и фиксации транзакции другого редактора. |

Незафиксированная зависимость («грязное» чтение): Незафиксированная зависимость возникает, когда вторая транзакция выбирает строку, изменяемую в данный момент другой транзакцией. Вторая транзакция будет считывать данные, которые еще не зафиксированы и могут быть изменены первой транзакцией. Этой проблемы можно избежать, если никто не сможет считать измененный документ, пока первый редактор не сохранит окончательную версию изменений и не зафиксирует эту транзакцию. |

Анализ несогласованности (неповторяющееся чтение): Анализ несогласованности возникает, когда вторая транзакция осуществляет доступ к одной строке несколько раз, и каждый раз считывает разные данные. Например, редактор считывает один и тот же документ дважды, но между этими операциями модуль записи перезаписывает этот документ. Когда редактор считывает документ во второй раз, он уже изменен. Первую операцию чтения повторить нельзя. Этой проблемы можно избежать, если модуль записи не сможет изменять документ, пока редактор не считает его в последний раз. |

Фантомные операции чтения: Чтение фантомных данных — это ситуация, которая возникает при выполнении двух идентичных запросов, если набор строк, возвращаемых вторым запросом, отличается от первого. Предположим, что две транзакции выполняются одновременно. Две инструкции SELECT в первой транзакции могут возвращать разные результаты, поскольку инструкция INSERT во второй транзакции изменяет данные, используемые обеими этими инструкциями.  
*// Transaction 1  
SELECT \* FROM dbo.employee ...  
// Transaction 2  
SELECT \* FROM dbo.employee COMMIT  
// Transaction 2  
INSERT INTO dbo.employee SET name = 'BBB' WHERE id = 5*|

Отсутствующие или дублированные операции чтения, вызванные обновлениями строк: Если во время сканирования индекса другой пользователь изменит ключевой столбец индекса для строки, считывание которой происходит в данный момент, причем строка была перемещена в позицию, до которой операция сканирования еще не дошла, эта строка может появиться повторно. Аналогично, если изменение ключа переместило строку в позицию, считывание которой уже прошло, то она может не отобразиться. Чтобы избежать этого, воспользуйтесь указаниями SERIALIZABLE или HOLDLOCK либо управлением версиями строк. |

Отсутствие одной или нескольких строк, которые не подвергались обновлению: Пропажа строк может возникнуть в случае, если при использовании уровня READ UNCOMMITTED запрос читает строки в порядке их расположения (с использованием IAM-страниц), а другая транзакция вызывает разбиение страницы. Этого не может произойти при использовании уровня изоляции READ COMMITTED, поскольку во время разбиения страницы включается блокировка таблицы.Также этого не может произойти, если таблица не имеет кластеризованного индекса, поскольку в таком случае обновления не вызывают разбиения страниц.

**Уровни изоляции транзакций и блокировки**

Уровни изоляции в компоненте Database Engine: Транзакции указывают уровень изоляции, который определяет степень, до которой одна транзакция должна быть изолирована от изменений ресурса или данных, произведенных другими транзакциями. Уровни изоляции описаны с точки зрения того, какие из побочных эффектов параллелизма разрешены (например, «грязные» чтения или фантомные чтения). Уровни изоляции транзакций контролируют следующие параметры: Применение и типы блокировки при чтении данных; Время удержания блокировок чтения; Использование операции чтения ссылок на строки, измененные другой транзакцией (Блокировка до тех пор, пока не будет снята монопольная блокировка строки; Извлечение зафиксированной версии строки, которая существовала в то время, когда началось выполнение инструкции или транзакции; Считывание незафиксированного изменения данных). |

Более низкий уровень изоляции увеличивает возможность получения доступа к данным несколькими пользователями одновременно, но увеличивает число эффектов параллелизма (таких как «грязное» чтение или потерянные обновления), с которыми может столкнуться пользователь. Наоборот, более высокий уровень изоляции уменьшает число эффектов параллелизма, с которыми может столкнуться пользователь, но требует больше системных ресурсов и увеличивает шанс того, что одна транзакция блокирует другую. Выбор соответствующего уровня изоляции зависит от баланса между требованиями к целостности данных приложения и издержек каждого уровня изоляции. |

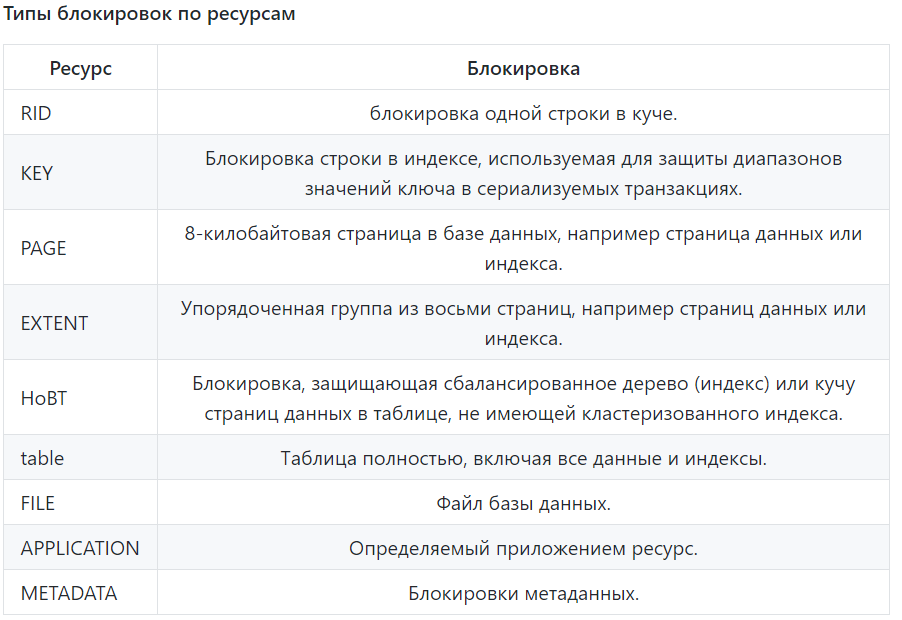
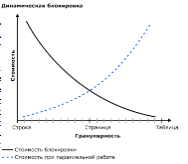
read uncommitted: Самый низкий уровень изоляции, при котором транзакции изолируются до такой степени, чтобы только уберечь от считывания физически поврежденных данных. На этом уровне разрешено «грязное» чтение, поэтому одна транзакция может видеть еще не зафиксированные изменения, совершенные другими транзакциями. |

read committed: Позволяет транзакции считывать данные, считанные до этого, но не измененные другой транзакцией, не ожидая завершения выполнения этой другой транзакции. Компонент Компонент Database Engine сохраняет блокировки записи (сформированные для выделенных данных) до конца транзакции, а блокировки чтения снимаются сразу же после выполнения инструкции SELECT. Это уровень Компонент Database Engine, заданный по умолчанию. |

repeatable read: Компонент Database Engine сохраняет блокировки чтения и записи, сформированные для выделенных данных, до конца транзакции. Однако из-за того, что блокировки диапазона не являются управляемыми, может возникнуть фантомное чтение |

  
Блокировка — это механизм, с помощью которого компонент SQL Server Database Engine синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных. Прежде чем транзакция сможет распоряжаться текущим состоянием фрагмента данных, например для чтения или изменения данных, она должна защититься от изменений этих данных другой транзакцией. Для этого транзакция запрашивает блокировку фрагмента данных. Режим блокировки определяет уровень подчинения данных транзакции. Ни одна транзакция не может получить блокировку, которая противоречит другой блокировке этих данных, предоставленной другой транзакции.

**Гранулярность блокировок, иерархии блокировок и эскалация блокировок**

Компонент SQL Server Database Engine поддерживает многогранулярную блокировку, позволяющую транзакции блокировать различные типы ресурсов. Чтобы уменьшить издержки применения блокировок, компонент 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.  
  
Динамические блокировки (Эскалация): Использование блокировок низкого уровня, например блокировок строк, увеличивает параллелизм, поскольку снижается вероятность одновременной блокировки области данных двумя различными транзакциями. Использование блокировок низкого уровня также увеличивает их количество, что приводит к большей загрузке ресурсов. При использовании блокировок страниц и таблиц высокого уровня загруженность снижается, однако при этом уменьшается степень параллелизма.  
 

**Взаимоблокировки, их обнаружение и устранение**

Взаимоблокировка возникает, когда две или более задачи постоянно блокируют друг друга из-за того, что задача каждой из сторон блокирует ресурс, необходимый другой стороне. Например:

1) Транзакция А создает общую блокировку строки 1;

2) Транзакция Б создает общую блокировку строки 2;

3)Транзакция А теперь запрашивает монопольную блокировку строки 2 и блокируется до того, как транзакция Б закончится и освободит общую блокировку строки 2. Транзакция Б теперь запрашивает монопольную блокировку строки 1 и блокируется до того, как транзакция A закончится и освободит общую блокировку строки 1.;

4) Транзакция А не может завершиться до того, как завершится транзакция Б, а транзакция Б заблокирована транзакцией А. Данное условие также называется цикличной зависимостью. Транзакция А зависит от транзакции Б, а транзакция Б зависит от транзакции А и этим замыкает цикл. |

Монитор взаимоблокировок компонента SQL Server Database Engine периодически проверяет задачи на состояние взаимоблокировки. Если монитор обнаруживает цикличную зависимость, то выбирается одна задача, для которой транзакция будет завершена с ошибкой. Это позволяет другой задаче завершить свою транзакцию. Позднее приложение может повторно выполнить транзакцию, которая завершилась с ошибкой. В качестве альтернативы пользователь может указать приоритет сеансов в ситуации взаимоблокировки, используя инструкцию SET DEADLOCK\_PRIORITY. Если у двух сеансов имеются различные приоритеты в случае взаимоблокировки, то в качестве жертвы взаимоблокировки будет выбран сеанс с более низким приоритетом. Если у обоих сеансов установлен одинаковый приоритет в случае взаимоблокировки, то в качестве объекта взаимоблокировки будет выбран сеанс, откат которого потребует наименьших затрат. Если сеансы, вовлеченные в цикл взаимоблокировки, имеют один и тот же приоритет в случае взаимоблокировки и одинаковую стоимость, то жертва взаимоблокировки выбирается случайным образом. Ресурсы, которые могут принимать участие во взаимоблокировке: Блокировки, Рабочие потоки, Память, Ресурсы, связанные с параллельным выполнением запросов, Ресурсы режима MARS. |

Для минимизации взаимоблокировок:

1)Осуществляйте доступ к объектам в одинаковом порядке;

2)Избегайте взаимодействия с пользователем в транзакциях;

3)Уменьшайте размер транзакций, желательно помещая их в один пакет;

4)Используйте низкий уровень изоляции;

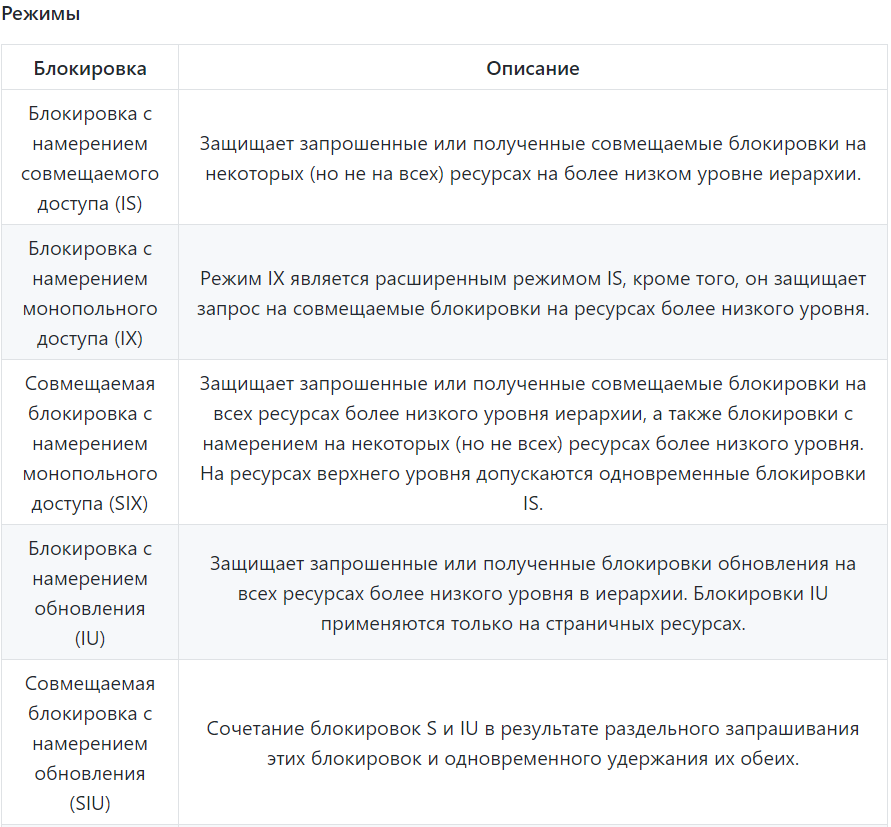
5)Используйте связанные соединения;

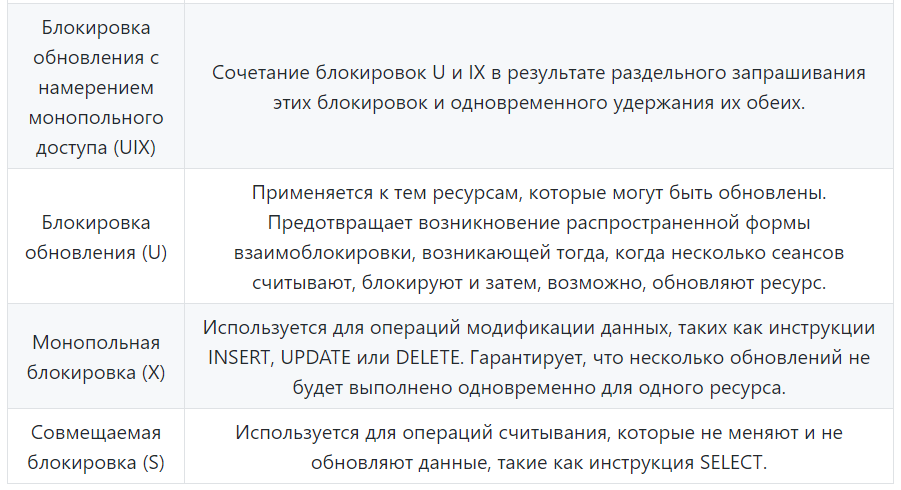
6)Используйте уровень изоляции строк, основанный на управлении версиями строк;

7) Установите параметр базы данных READ\_COMMITTED\_SNAPSHOT в ON для разрешения использования управления версиями строк транзакциями с уровнем изоляции READ COMMITTED;

8) Используйте изоляцию моментальных снимков.

**Режимы блокировок и совместимость блокировок**



Совместимость: 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.  


**Обработка ошибок в транзакциях**

@@ERROR - Возвращает номер ошибки для последней выполненной инструкции Transact-SQL. Возвращает 0, если в предыдущей инструкции Transact-SQL не возникли ошибки. Просмотреть текст, связанный с номером ошибки @@ERROR можно в sys.messages. |

@@ROWCOUNT - Возвращает число строк, затронутых при выполнении последней инструкции Transact-SQL. |

@@TRANCOUNT - Возвращает число инструкций BEGIN TRANSACTION, выполненных в текущем соединении.

*BEGIN TRY  
{ sql\_statement | statement\_block }  
END TRY BEGIN CATCH  
[ { sql\_statement | statement\_block } ] END CATCH  
[;]*Реализация обработчика ошибок на языке Transact-SQL. В области блока CATCH для получения сведений об ошибке, приведшей к выполнению данного блока CATCH, можно использовать следующие системные функции:

1) ERROR\_NUMBER() возвращает номер ошибки;

2) ERROR\_SEVERITY() возвращает степень серьезности ошибки;

3) ERROR\_STATE() возвращает код состояния ошибки.;

4) ERROR\_PROCEDURE() возвращает имя хранимой процедуры или триггера, в котором произошла ошибка.;

5) ERROR\_LINE() возвращает номер строки, которая вызвала ошибку, внутри подпрограммы.;

6) ERROR\_MESSAGE() возвращает полный текст сообщения об ошибке. Текст содержит значения подставляемых параметров, таких как длина, имена объектов или время.;

7) XACT\_STATE() - Возвращает состояние транзакции текущего выполняемого запроса. (“1” - Текущий запрос содержит активную пользовательскую транзакцию. Запрос может выполнять любые действия, включая запись данных и фиксирование транзакции; “0” - У текущего запроса нет активной пользовательской транзакции.; “-1” - В текущем запросе есть активная транзакция, однако произошла ошибка, из-за которой транзакция классифицируется как нефиксируемая. Запросу не удается зафиксировать транзакцию или выполнить откат до точки сохранения; можно только запросить полный откат транзакции. Запрос не может выполнить никакие операции записи, пока не будет проведен откат транзакции. До отката транзакции запрос может выполнять только операции считывания. После отката транзакции запросу будут доступны как операции считывания, так и операции записи, а также запуск новых транзакций. После завершения работы пакета компонент Database Engine автоматически выполнит откат любых активных нефиксируемых транзакций. Если при переходе транзакции в нефиксируемое состояние не было отправлено сообщение об ошибке, после завершения выполнения пакета сообщение об ошибке будет отправлено клиентскому приложению. Сообщение показывает, что обнаружены нефиксируемые транзакции и выполнен откат.);

8) SET XACT\_ABORT { ON | OFF } - Указывает, выполняет ли SQL Server автоматический откат текущей транзакции, если инструкция языка Transact-SQL вызывает ошибку выполнения. Если выполнена инструкция SET XACT\_ABORT ON и инструкция языка Transact-SQL вызывает ошибку, вся транзакция завершается и выполняется ее откат. Если выполнена инструкция SET XACT\_ABORT OFF, в некоторых случаях выполняется откат только вызвавшей ошибку инструкции языка Transact-SQL, а обработка транзакции продолжается.

**Журналирование в SQL Server**

Каждая база данных SQL Server имеет журнал транзакций, в котором фиксируются все изменения данных, произведенные в каждой из транзакций. Журнал транзакций необходимо регулярно усекать, чтобы избежать его переполнения. Но при этом по ряду причин его усечение может быть отложено, поэтому очень важно следить за размером журнала. Некоторые операции можно выполнять с минимальным протоколированием, чтобы сократить их вклад в размер журнала транзакций. Журнал транзакций является критическим компонентом базы данных и в случае системного сбоя может потребоваться для приведения базы данных в согласованное состояние. Журнал транзакций нельзя ни удалять, ни изменять, если только не известны возможные последствия. |

Журнал транзакций поддерживает следующие операции:

1) восстановление отдельных транзакций;

2) восстановление всех незавершенных транзакций при запуске SQL Server;

3) накат восстановленной базы данных, файла, файловой группы или страницы до момента сбоя;

4) поддержка репликации транзакций;

5) Поддержка решений высокого уровня доступности и аварийного восстановления: Группы доступности AlwaysOn, зеркальное отображение базы данных и доставка журналов. |

Просмотр журнала ошибок SQL Server: Журнал ошибок сервера SQL Server позволяет убедиться, что процессы были завершены успешно (например, операции резервного копирования и восстановления, пакеты команд или другие сценарии и процессы). Это полезно при определении любых текущих или потенциальных проблем, включая сообщения автоматического восстановления (особенно если экземпляр SQL Server был остановлен и перезапущен), сообщения ядра и другие сообщения об ошибках на уровне сервера. По умолчанию журнал ошибок содержится в файлах Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\LOG\ERRORLOG и ERRORLOG.n. Новый журнал ошибок создается при каждом запуске экземпляра сервера SQL Server, хотя циклическую смену журнала ошибок можно организовать при помощи системной хранимой процедуры sp\_cycle\_errorlog без перезапуска экземпляра сервера SQL Server. Обычно сервер SQL Server хранит резервные копии шести предыдущих журналов и присваивает наиболее свежей копии расширение «.1», следующей расширение «.2» и т. д. Файл текущего журнала ошибок расширения не имеет.

**Логическая архитектура журнала транзакций в SQL Server**

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

1) Log Sequence Number: регистрационный номер транзакции (LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи.;

2)Prev LSN: обратный указатель, который предназначен для ускорения отката транзакции.;

3)Transaction ID number: идентификатор транзакции.;

4) Type: тип записи Log-файла.; 5)Other information: Прочая информация. |

Двумя основными типами записи Log-файла являются:

1) Transaction Log Operation Code: код выполненной логической операции;

2) 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. |

Действия, которые необходимо выполнить для восстановления операции, зависят от типа Log-записи:

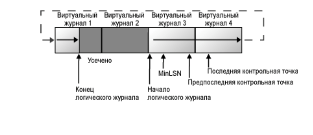
1) Зарегистрирована логическая операция. (Для наката логической операции выполняется эта операция; Для отката логической операции выполняется логическая операция, обратная зарегистрированной.);

2) Зарегистрированы исходный и результирующий образы записи (Для наката операции применяется результирующий образ; Для отката операции применяется исходный образ.) |

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. |

Раздел журнального файла, который начинается от первой записи, необходимой для успешного отката на уровне базы данных, до последней зарегистрированной записи называется активной частью журнала, или активным журналом. Именно этот раздел необходим для выполнения полного восстановления базы данных. Активный журнал не может быть усечен.

**Физическая архитектура журнала транзакций в SQL Server**

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. Журнал транзакций является оборачиваемым файлом. Рассмотрим пример. Пусть база данных имеет один физический файл журнала, разделенный на четыре виртуальных файла журнала. При создании базы данных логический файл журнала начинается в начале физического файла журнала. Новые записи журнала добавляются в конце логического журнала и приближаются к концу физического файла журнала. Усечение журнала освобождает любые виртуальные журналы, все записи которых находятся перед минимальным регистрационным номером восстановления в журнале транзакций (MinLSN). MinLSN является регистрационным номером самой старой записи, которая необходима для успешного отката на уровне всей базы данных. Журнал транзакций рассматриваемой в данном примере базы данных будет выглядеть примерно так же, как на следующей иллюстрации. Когда конец логического журнала достигнет конца физического файла журнала, новые записи журнала будут размещаться в начале физического файла журнала.  
  
Этот цикл повторяется бесконечно, пока конец логического журнала не совмещается с началом этого логического журнала. Если старые записи журнала усекаются достаточно часто, так что при этом всегда остается место для новых записей журнала, созданных с новой контрольной точки, журнал постоянно остается незаполненным. Однако, если конец логического журнала совмещается с началом этого логического журнала, происходит одно из двух событий, перечисленных ниже:

1) Если для данного журнала применена установка FILEGROWTH и на диске имеется свободное место, файл расширяется на величину, указанную в growth\_increment, и новые записи журнала добавляются к этому расширению.;

2) Если установка FILEGROWTH не применяется или диск, на котором размещается файл журнала, имеет меньше свободного места, чем это указано в growth\_increment, формируется ошибка 9002.;

3) Если в журнале содержится несколько физических файлов журнала, логический журнал будет продвигаться по всем физическим файлам журнала до тех пор, пока он не вернется на начало первого физического файла журнала.

**Контрольные точки, активная часть журнала и усечение журнала транзакций в SQL Server**

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:

1) Записывает в файл журнала запись, отмечающую начало контрольной точки.;

2) 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). |

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. |

Действия, приводящие к срабатыванию контрольных точек:

1) При явном выполнении инструкции CHECKPOINT. Контрольная точка срабатывает в текущей базе данных соединения.;

2) При выполнении в базе данных операции с минимальной регистрацией, например при выполнении операции массового копирования для базы данных, на которую распространяется модель восстановления с неполным протоколированием.;

3) При добавлении или удалении файлов баз данных с использованием инструкции ALTER DATABASE.;

4) При остановке экземпляра SQL Server с помощью инструкции SHUTDOWN или при остановке службы SQL Server (MSSQLSERVER). И в том, и в другом случае будет создана контрольная точка каждой базы данных в экземпляре SQL Server. Если экземпляр SQL Server периодически создает в каждой базе данных автоматические контрольные точки для сокращения времени восстановления базы данных.;

5) При создании резервной копии базы данных.;

6) При выполнении действия, требующего отключения базы данных. Примерами могут служить присвоение параметру AUTO\_CLOSE значения ON и закрытие последнего соединения пользователя с базой данных или изменение параметра базы данных, требующее перезапуска базы данных. |

Автоматические контрольные точки: Компонент Database Engine создает контрольные точки автоматически. Интервал между автоматическими контрольными точками определяется на основе использованного места в журнале и времени, прошедшего с момента создания последней контрольной точки. Интервал между автоматическими контрольными точками колеблется в широких пределах и может быть довольно длительным, если база данных изменяется редко. При крупномасштабных изменениях данных частота автоматических контрольных точек может быть гораздо выше. |

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. На следующем рисунке изображена упрощенная схема журнала завершения транзакций, содержащего две активные транзакции. Записи контрольных точек были сжаты в одну запись. |

Управление журналом транзакций: Чтобы логический журнал не увеличивался до размера физических файлов журнала, следует периодически выполнять его усечение. Процесс усечения журнала уменьшает размер файла логического журнала, помечая виртуальные файлы журнала, которые не содержат частей логического журнала, как неактивные. В некоторых случаях может оказаться полезным физическое сжатие или расширение размера реального файла 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.

**Управление участниками системы безопасности в SQL Server** Участники — сущности, которые могут запрашивать ресурсы SQL Server. Поддерживаются два вида учетных записей подключения к серверу:

1)учетные записи сервера, создаваемые на основании учетных записей операционной системы;

2)учетные записи сервера, создаваемые для прямого подключения к серверу. |

Проверка подлинности Windows означает, что для подключения к SQL Server проверка подлинности полностью выполняется операционной системой Microsoft Windows. В этом случае клиент идентифицируется на основании учетной записи Windows. Проверка подлинности SQL Server означает, что для подключения к SQL Server проверка подлинности выполняется путем сравнения имени пользователя и пароля с хранящимся на сервере SQL Server списком действительных имен пользователей и паролей). Учетные данные сервера создаются при помощи инструкции CREATE CREDENTIAL. После создания учетных данных можно сопоставить их имени входа SQL Server, используя инструкцию CREATE LOGIN или ALTER LOGIN. Как и другие компоненты модели авторизации SQL Server, участников можно иерархически упорядочить. Область влияния участника зависит от области его определения: Windows, сервер, база данных, — а также от того, коллективный это участник или индивидуальный. Имя входа Windows является примером индивидуального (неделимого) участника, а группа Windows — коллективного. Каждый из участников имеет идентификатор безопасности (SID). |

Имя входа «sa» SQL Server: Имя входа SQL Server sa является участником уровня сервера. По умолчанию оно создается при установке экземпляра. В SQL Server 2005 базой данных для имени входа sa по умолчанию является master. Это поведение было изменено по сравнению с предыдущими версиями SQL Server. |

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, он наследует на него разрешения роли public. |

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. |

Пользователь-гость: Каждая база данных включает в себя пользователей гость. Разрешения, предоставленные пользователю гость, наследуются пользователями, которые имеют доступ к базе данных, но не обладают учетной записью пользователя в базе данных. Пользователя гость нельзя удалить, но его можно отключить, если отменить его разрешение CONNECT. Разрешение CONNECT можно отменить, выполнив инструкцию REVOKE CONNECT FROM GUEST в любой базе данных, кроме master или tempdb. |

Клиент и сервер базы данных: По определению клиент и сервер базы данных являются защищаемыми субъектами безопасности. Данные сущности могут пройти взаимную проверку подлинности перед установкой безопасного сетевого соединения. SQL Server поддерживает протокол проверки подлинности Kerberos, который определяет, как клиенты взаимодействуют со службой проверки подлинности в сети. |

Защищаемые объекты: Защищаемые объекты – это ресурсы, доступ к которым регулируется системой авторизации. Некоторые защищаемые объекты могут храниться внутри других, создавая иерархии «областей», которые сами могут защищаться. К областям защищаемых объектов относятся (a) сервер, (b) база данных и (c) схема.

**Управление безопасностью в SQL Server**

**Управление разрешениями в SQL Server**

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.   
*GRANT { ALL [ PRIVILEGES ] | список\_разрешений }  
ON список\_объектов  
TO список\_принципалов  
[ WITH GRANT OPTION ] [ AS принципал ]*Ключевое слово ALL с необязательным словом RIVILEGES не включает все возможные разрешения, оно эквивалентно предоставлению всех разрешений ANSI-92, применимых к указанному объекту. Значение ALL различается для разных типов объектов. Полный список разрешений содержит 195 пунктов. Если разрешение предоставляется на таблицу, представление или функцию, возвращающую табличное значение, то справа от разрешения в круглых скобоках могут указываться имена столбцов. На столбец могут быть предоставлены только разрешения SELECT, REFERENCES и UPDATE. Объект, на который предоставляется разрешение, имеет следующее описание: [ OBJECT :: ] [ имя\_схемы ].имя\_объекта. Фраза OBJECT необязательна, если указан аргумент имя\_схемы. Если же она указана, указание квалификатора области (::) обязательно. Если не указан аргумент имя\_схемы, подразумевается схема по умолчанию. Если указан аргумент \_имя\_схемы), обязательно указание квалификатора области схемы (.). Принципалом может быть:

1)пользователь базы данных;

2)роль базы данных; 3)роль приложения. |

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 ]*Необязательная фраза GRANT OPTION FOR показывает, что право на предоставление заданного разрешения другим принципалам будет отменено. Само разрешение отменено не будет. Необязательное ключевое слово CASCADE показывает, что отменяемое разрешение также отменяется для других принципалов, для которых оно было предоставлено или запрещено этим принципалом. Каскадная отмена разрешения, предоставленного с помощью параметра WITH GRANT OPTION, приведет к отмене прав GRANT и DENY для этого разрешения. Необязательная фраза AS принципал указывает принципала, от которого принципал, выполняющий данный запрос, получает право на отмену разрешения. Пример. Отмена разрешения EXECUTE для хранимой процедуры |

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 ]*

**Метаданные в SQL Server 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. (Например: CREATE AGGREGATE, CREATE FUNCTION, CREATE ASSEMBLY, CREATE INDEX, CREATE TABLE, CREATE DATABASE ) |

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

1) Представления информационной схемы;

2) Представления каталога;

3) Представления совместимости;

4) Представления репликации;

5) Динамические административные представления и функции;

6)Представления приложения уровня данных (DAC) |

Представления каталогов баз данных и файлов содержит 6 представлений: Например:

1)sys.database\_mirroring;

2)sys.database\_recovery\_status;

3)sys.databases. |

Так же содержатся:

1) Системные хранимые процедуры;

2)Функции метаданных;

3) Функции конфигурации;

4) Функции безопасности;

5) Системные функции;

6)DBCC команда

**Распределенные запросы и распределенные транзакции в SQL Server**

Распределенные запросы используются для доступа к данным из нескольких разнородных источников данных. Эти источники данных могут храниться на одном или различных компьютерах. Пользователи SQL Server могут применять распределенные запросы для доступа к следующим данным:

1) Распределенные данные, хранящиеся в нескольких экземплярах SQL Server;

2)Разнородные данные, хранящиеся в различных реляционных и нереляционных источниках данных, доступ к которым осуществляется с использованием поставщика OLE DB. |

Инструкции Transact-SQL, контролирующие распределенные транзакции, немногочисленны, поскольку большая часть работы производится внутри экземпляра SQL Server Database Engine и MS DTC. Ниже представлены инструкции Transact-SQL, которые необходимо выполнить в сценарии или приложении Transact-SQL:

1) Запуск распределенной транзакции;

2)Выполнение распределенных запросов к связанным серверам или выполнение удаленного вызова процедур на удаленных серверах;

3)Вызов стандартных инструкций Transact-SQL COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION или ROLLBACK WORK для завершения транзакции. |

Запуск распределенных транзакций: Распределенную транзакцию в Transact-SQL можно запустить следующими способами:

1)Явно начать распределенную транзакцию, используя инструкцию BEGIN DISTRIBUTED TRANSACTION. Можно также выполнить распределенный запрос к связанному серверу. Экземпляр компонента Database Engine вызовет MS DTC для обслуживания распределенной транзакции на связанном сервере. В рамках распределенной транзакции можно также вызвать удаленные хранимые процедуры на удаленном экземпляре компонента Database Engine.;

2)Находясь в локальной транзакции, выполнить распределенный запрос. Если источник данных OLE DB поддерживает интерфейс ITransactionJoin, транзакция превращается в распределенную, даже если этот запрос является запросом только для чтения. Если источник данных не поддерживает интерфейс ITransactionJoin, допустимы лишь инструкции только для чтения.;

3)Если была выполнена инструкция SET REMOTE\_PROC\_TRANSACTIONS ON и локальная транзакция вызывает удаленную хранимую процедуру на другом экземпляре компонента Database Engine, локальная транзакция становится распределенной. Компонент Database Engine использует MS DTC, чтобы координировать транзакцию с удаленным сервером. Обращения к удаленным хранимым процедурам выполняются вне области локальной транзакции, если REMOTE\_PROC\_TRANSACTIONS установлен в OFF. При откате локальной транзакции изменения, сделанные удаленной процедурой, не возвращаются. Фиксация изменений, произведенных удаленной хранимой процедурой, производится во время ее завершения, а не при фиксации локальной транзакции.