Часть 1- Введение в БД

1. Приложения БД. Требования к приложениям БД. Процесс разработки БД.

Основная цель создания приложений БД – хранение, накопление, обработка и представление информации различного рода. Области и масштабы их применения обширны.

Примеры приложений БД: адресная/телефонная книга, каталог продукции, э-каталог библиотеки.

Требования к приложениям БД могут зависеть от предметной области и от масштабов системы: производительность; характер накапливаемой информации; возможность сетевого доступа к данным; масштабируемость; безопасность; переносимость; надежность и отказоустойчивость; …

Процесс разработки БД:

Общие стратегии:

– *разработка сверху вниз* (получение абстрактной модели данных исходя из анализа стратегических целей организации, способов их достижения, инфо и способов ее представления, которые необходимы для достижения целей);

– *разработка снизу вверх* (для разработки выбирается конкретная система, которая выполняет только часть функций предприятия; исходя из сформированных требований, выбранная подсистема создается быстрее и с меньшими рисками).

Одна из основ эффективной реализации приложения БД – ясное представление модели предметной области.

2. Моделирование данных. Модель «сущность-связь» и модель семантических объектов: основные понятия.

Моделирование данных – процесс создания логического представления БД.

Модель данных содержит языковые и изобразительные стандарты для своего представления.

Модели данных (ANSI): внешняя (набор представлений пользователей о той части предметной области, с которой он сталкивается); концептуальная (набор ключевых объектов предметной области, их взаимосвязей и ограничений, накладываемых на объекты); логическая (представление концептуальной модели в соответствии с логической моделью, используемой для хранения данных); физическая (описание физического представления, хранения и обработки данных).

Модель «сущность-связь» и ее основные элементы:

Сущность (entity) – некоторый объект, который способен существовать независимо и может быть идентифицирован:

- класс сущностей – набор сущностей определенного типа;
- экземпляр сущности – сущность некоторого класса с заданными значениями атрибутов;
- сущность может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут (attribute) – свойство, которым обладает сущность:

- все сущности определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов;
- в общем случае, атрибуты могут быть: составными (composite); многозначными (multi-value);
- идентификатор сущности (entity identifier) – набор атрибутов, которые определяют, именуют сущность;
- идентификаторы могут быть составными, т.е. состоять более, чем из одного атрибута.

Связь (relationship) определяет, как сущности соотносятся друг с другом:

- подобно классам и экземплярам сущностей, можно выделить классы связей и экземпляры связей;
- степень (degree) связи показывает, сколько сущностей определяют эту связь (участвуют в связи): бинарные связи (binary); тернарные связи (ternary).

Семантическая объектная модель – альтернативный способ представления модели данных.

Семантический объект (semantic object) – некоторый объект, который способен существовать независимо и может быть идентифицирован (именованный набор атрибутов, исчерпывающе описывающий отдельную сущность). Аналогично сущностям ER-модели можно определить:

- класс объектов (object class) – набор объектов определенного типа;
- экземпляр объекта (object instance) – объект некоторого класса с заданными значениями атрибутов;
- атрибут (attribute) – свойство, которым обладает объект;
- объект может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут (attribute) – свойство, которым обладает объект:

- все объекты определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов (исчерпывающее описание, sufficient description);
- в общем случае, атрибуты могут быть: простыми (simple): атрибут со значением элементарного типа; групповыми, или составными (group): атрибут состоит из набора простых атрибутов; объектными (object): атрибут является семантическим объектом (очевидно, атрибуты данного типа являются <зеркальными> для двух соответствующих объектов).

3. История развития БД. Системы обработки файлов и базы данных. Системы управления БД (СУБД).

История развития БД:

до 1970-х : переход от записей вручную к системам обработки файлов
1970 – 1980 : появление первых БД (dBase, System2000)
1978 – 1985 : появление реляционных БД (Oracle)
1982 – 1992 : развитие БД для настольных компьютеров (dBase-II, Access)
1985 – 2000 : появление и развитие ОО БД
1995 – н.в. : интеграция технологий БД и интернет-технологий.

Системы обработки файлов: разделенные и изолированные файлы – каждому приложению может соответствовать свой набор файлов; зависимость прикладных программ от форматов файлов и несовместимость файлов; дублирование данных – возможное нарушение целостности данных; трудность представления данных в файлах в различных видах / совместного использования данных; сложность разработки и поддержки.

База данных – самодокументированный (т.е. БД содержит не только данные, но и их описание - метаданные) набор интегрированных записей (байты -> поля -> записи -> файлы, метаданные, индексы, метаданные приложения).

СУБД – посредник между приложениями и данными:

- изоляция от физической организации хранения данных;
- данные интегрированы;
- уменьшение дублирования данных;
- независимость программ от форматов файлов;
- возможность гибкого выбора формы представления данных.

4. Понятие реляционной модели и нормализации. Реляционные БД. Метаданные в реляционных БД.

Реляционная модель определяет способ хранения данных в виде таблиц со строками и столбцами, при этом минимизируется дублирование и исключаются определенные типы ошибок обработки. Она дает стандартный способ структурирования и обработки данных.

Нормализация – последовательность преобразований, обеспечивающих определенный набор требований.

Объекты реляционной БД:

основной объект – отношение, представляющее собой двумерную таблицу (столбцы – поля, атрибуты, определяющие набор данных, которыми обладает описываемый объект; строки – записи, содержащие набор значений атрибутов для конкретного объекта);

домен – множество допустимых значений атрибута (физическое описание – тип данных и дополнительные ограничения; семантическое описание – описание назначения данного атрибута);

ключ – набор одного или нескольких атрибутов, однозначно идентифицирующий конкретную запись.

Хранение метаданных в реляционной БД осуществляется в системных таблицах, которые могут располагаться в специализированной системной БД.

5. Системы управления БД (СУБД). Ядро СУБД. Язык SQL. Дополнительные компоненты современных СУБД.

СУБД – посредник между приложениями и данными:

- изоляция от физической организации хранения данных;
- данные интегрированы;
- уменьшение дублирования данных;
- независимость программ от форматов файлов;
- возможность гибкого выбора формы представления данных.

Ядро СУБД обеспечивает хранение, обработку и защиту данных (преобразование запросов в действия над данными; безопасность; управление транзакциями и блокировками; резервное копирование и восстановление).

Компоненты современных СУБД: ядро; средства интерактивной аналитической обработки данных (OLAP); средства интеллектуального анализа данных; средства интеграции данных; средства создания и публикации форм и отчетов; средства репликации (копирование и распространение данных с синхронизацией); средства выполнения полнотекстовых запросов к неструктурированным символьным данным в таблицах.

Язык SQL – язык манипулирования реляционными данными.

Язык представляет собой совокупность операторов (операторы определения данных (CREATE), манипуляции данными (SELECT), определения доступа к данным (GRANT) и управления транзакциями (COMMIT)), инструкций и вычисляемых функций.

6. Системы управления БД (СУБД). Создание БД и основные объекты ядра СУБД.

СУБД – посредник между приложениями и данными:

- изоляция от физической организации хранения данных;
- данные интегрированы;
- уменьшение дублирования данных;
- независимость программ от форматов файлов;
- возможность гибкого выбора формы представления данных.

Создание БД заключается в определении схемы БД, т.е. набора объектов, которые отражают разработанную модель предметной области: таблиц, связей, доменов и объектов, реализующих бизнес-логику (набор ограничений на возможные действия пользователя с данными в БД).

Ядро СУБД обеспечивает хранение, обработку и защиту данных (преобразование запросов в действия над данными; безопасность; управление транзакциями и блокировками; резервное копирование и восстановление).

7. Модели БД. Иерархическая и сетевая модели. Реляционная модель.

Иерархическая модель – модель данных, в которой данные представляют собой древовидную структуру.

Сетевая модель – модель данных с сетевой структурой, возможно наличие циклов (как на уровне модели данных, так и на уровне самих данных).

Реляционная модель определяет способ хранения данных в виде таблиц со строками и столбцами, при этом минимизируется дублирование и исключаются определенные типы ошибок обработки. Она дает стандартный способ структурирования и обработки данных.

8. Модели БД. Реляционная модель. Обзор постреляционных моделей (объектно-ориентированные, документо-ориентированные, графовые и столбцовые базы данных).

Реляционная модель определяет способ хранения данных в виде таблиц со строками и столбцами, при этом минимизируется дублирование и исключаются определенные типы ошибок обработки. Она дает стандартный способ структурирования и обработки данных.

Пост-реляционные модели:

ОО СУБД предназначены для прозрачной и унифицированной обработки структур данных ООП, т.к. реляционная модель для этого не подходит (сложность переноса накопленных данных, не обладают достаточной универсальностью);

хранилища ключей и значений – хранилища сопоставляют значения ключам, дополнительные возможности: поддержка различных типов значений, поддержка веб-технологий, механизм запросов, коммуникационные средства, также могут рассматриваться более сложные способы организации данных на уровне хранения;

документо-ориентированные БД предназначены для хранения и обработки документо-ориентированной информации, основана на понятии документа, как сущности, включающей в себя некоторый набор данных, закодированных в стандартных форматах (XML, JSON, PDF);

графовые БД предназначены для хранения структуры графа: узлов и связей между ними (как с узлами, так и со связями могут быть ассоциированы свойства (атрибуты), предоставляющие собой пару ключ – значение и позволяющие хранить дополнительные данные);

столбцовые БД ориентированы на хранение данных по столбцам, в отличие от реляционных (по строкам) – несущественные накладные расходы на добавление нового столбца к уже существующим данным, гибкость схемы данных (набор столбцов у различных строк может быть разным), возможна высокая степень сжатия данных для столбцов с повторяющимися значениями.

9. Схемы доступа к данным. Терминальный доступ. Доступ в режиме разделения файлов. Архитектура «клиент-сервер».

Терминальный доступ: единственный мейнфрейм и множество терминалов, предназначенных исключительно для ввода-вывода; недостаток – высокая нагрузка на мейнфрейм.

Доступ в режиме разделения файлов: файл-сервер – компьютер, преимущественно использующийся для разделяемого хранения файлов; все или часть компонентов СУБД выполняются на компьютере пользователя, для этого может понадобиться передача файлов; недостатки – высокая нагрузка на сеть; сложные схемы обеспечения целостности данных, совместного доступа к данным и восстановления, высокая стоимость владения.

Архитекрура «клиент-сервер»: на клиенте выполняется приложение пользователя, которое обращается к СУБД на сервере. Функции клиента – представления данных, выполнение некоторой бизнес-логики, отправка запросов и прием результатов их выполнения. Функции сервера – выполнение бизнес-логики, обеспечение совместного доступа к данным.

«+» : возможность увеличения производительности (параллельное выполнение запросов и настройка сервера под конкретную СУБД), возможность обеспечения целостности данных и безопасности, снижение сетевого трафика и стоимости владения.

«-» : ограниченная возможность масштабирования, большие издержки на поддержание клиентских приложений и требований к производительности.

10. Схемы доступа к данным. Архитектура «клиент-сервер». Распределенные БД. Параллельные БД.

Архитекрура «клиент-сервер»: на клиенте выполняется приложение пользователя, которое обращается к СУБД на сервере. Функции клиента – представления данных, выполнение некоторой бизнес-логики, отправка запросов и прием результатов их выполнения. Функции сервера – выполнение бизнес-логики, обеспечение совместного доступа к данным.

«+» : возможность увеличения производительности (параллельное выполнение запросов и настройка сервера под конкретную СУБД), возможность обеспечения целостности данных и безопасности, снижение сетевого трафика и стоимости владения.

«-» : ограниченная возможность масштабирования, большие издержки на поддержание клиентских приложений и требований к производительности.

Распределенная БД: логическая совокупность данных и метаданных, распределенных внутри сети.

«+» : возможность интеграции и прозрачного доступа к данным других подсистем; высокая степень доступности данных; более дешевые и масштабируемые, чем кластерные системы.

Параллельные БД: позволяют повысить производительность за счет параллельного выполнения транзакций – разделяемая память (эффективный обмен между процессорами, невозможность масштабирования); разделяемое дисковое пространство (определенная степень отказоустойчивости при отказе памяти/процессора + отказоустойчивость дисковой подсистемы за счет организации RAID-массивов, узкое место – обмен с дисковой подсистемой); нет разделяемых ресурсов (наиболее масштабируемая, неравномерная скорость доступа к различным дискам).

11. Схемы доступа к данным. Архитектура «точка-точка». Облачные вычисления. БД для мобильных устройств.

Архитектура «точка-точка»: отсутствует четко выраженное разделение клиентов и серверов – связи могут формироваться динамически для обмена данными и услугами. В связи с отсутствием выделенного центрального узла и глобальной схемы данных могут потребоваться дополнительные средства для скрытия разнородности данных/систем и обеспечения унифицированного представления услуг и данных.

Облачные вычисления: частично основаны на концепциях архитектуры, ориентированной на службы – функциональность разбита на набор взаимодействующих служб (службы не являются тесно связанными, обеспечивается интероперабельность служб, реализованных на разных платформах, службы могут инкапсулировать функции других служб произвольным образом).

Модели служб: инфраструктура как услуга – визуализация физических ресурсов; платформа как услуга – ОС, СУБД, веб-сервер; ПО как услуга – приложения и БД.

БД для мобильных устройств.

Требования к таким БД: компактность ПО для обеспечения возможности его функционирования на мобильных устройствах в условиях ограничения доступных ресурсов; учет контекста и местоположения пользователей при формировании запросов; различные способы подключения к центральному серверу БД; необходимость синхронизации данных на центральном сервере и на мобильном устройстве; необходимость учитывать возможные сбои в сети; учет требований безопасности при репликации данных на мобильное устройство; возможность обмена информацией между мобильными устройствами при появлении такой возможности (в режиме «точка-точка»).

12. Управление параллельной обработкой в БД. Приложения БД для оперативной обработки транзакций (OLTP) и поддержки принятия решений (OLAP).

Управление параллельной обработкой в БД направлено на то, чтобы исключить непредусмотренное влияние действий одного пользователя на действия другого. Как правило, предполагает компромисс между уровнем изоляции различных операций друг от друга и производительностью системы.

Основные категории приложений БД:

оперативная обработка транзакций (OLTP) – приложениями пользуются многие пользователи, которые одновременно совершают транзакции, изменяя таким образом текущие данные

поддержка принятия решений (OLAP) – предназначены для организации хранения большого количества неизменных данных с целью упрощения их анализа и получения.

Часть 2 – Моделирование данных

1. Процесс разработки БД и уровни моделирования информационных систем. Моделирование данных. Модель «сущность-связь» и ее основные элементы (сущность, связь, атрибуты).

Процесс разработки БД:
Общие стратегии:

- разработка сверху вниз - получение абстрактной модели данных исходя из анализа стратегических целей организации, способов их достижения, а также информации и способов ее представления, которые необходимы для достижения этих целей; лучшее взаимодействие подсистем, глобальный охват;
- разработка снизу вверх - для разработки выбирается конкретная система, которая выполняет только часть функций предприятия; исходя из сформированных требований, выбранная подсистема создается быстрее и с меньшими рисками.

!!!Одна из основ эффективной реализации приложения базы данных – ясное представление модели предметной области.

Уровни моделирования информационных систем:

- внешние модели (external schema): представление пользователей о системе (user view);
- концептуальная модель (conceptual schema): абстрактное представление системы (данных);
- внутренние модели (internal schema).

Моделирование данных – процесс создания логического представления БД. (пользовательская
модель данных, модель требований к данным, концептуальная модель).

Модели данных:
внешняя модель – набор представлений пользователей о той части предметной области, с которой он сталкивается;
концептуальная модель – набор ключевых объектов предметной области, их взаимосвязей
и ограничений, накладываемых на объекты;
логическая модель – представление концептуальной модели в соответствии с логической
моделью, используемой для хранения данных;
физическая модель – описание физического представления, хранения и обработки данных.

БД является «моделью модели», то есть через объекты базы данных описывает представление пользователей о конкретной предметной области. При любом процессе разработки необходимо сформулировать требования и построить на их основе модель данных. Модель данных содержит языковые и изобразительные стандарты для своего представления. Существует модель «сущность – связь», семантическая объектная модель.

Модель «сущность-связь» и ее основные элементы:
Сущность – некоторый объект, который способен существовать независимо и может быть идентифицирован:

- класс сущностей – набор сущностей определенного типа;
- экземпляр сущности – сущность некоторого класса с заданными значениями атрибутов;
- сущность может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут – свойство, которым обладает сущность:

- все сущности определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов;
- в общем случае, атрибуты могут быть: составными; многозначными;
- идентификатор сущности – набор атрибутов, которые определяют, именуют сущность;
- идентификаторы могут быть составными, т.е. состоять более чем из одного атрибута.

Связь определяет, как сущности соотносятся друг с другом:

- подобно классам и экземплярам сущностей, можно выделить классы связей и экземпляры связей;
- степень связи показывает, сколько сущностей определяют эту связь (участвуют в связи): бинарные связи; тернарные связи.

2. Модель «сущность-связь» и ее основные элементы (сущность, связь, атрибуты). Кардинальные числа связей.

Модель «сущность-связь» и ее основные элементы:
Сущность – некоторый объект, который способен существовать независимо и может быть идентифицирован:

- класс сущностей – набор сущностей определенного типа;
- экземпляр сущности – сущность некоторого класса с заданными значениями атрибутов;
- сущность может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут – свойство, которым обладает сущность:

- все сущности определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов;
- в общем случае, атрибуты могут быть: составными; многозначными;
- идентификатор сущности – набор атрибутов, которые определяют, именуют сущность;
- идентификаторы могут быть составными, т.е. состоять более чем из одного атрибута.

Связь определяет, как сущности соотносятся друг с другом:

- подобно классам и экземплярам сущностей, можно выделить классы связей и экземпляры связей;
- степень связи показывает, сколько сущностей определяют эту связь (участвуют в связи): бинарные связи; тернарные связи.

кардинальное число указывает число экземпляров сущностей, которые участвуют в связи;
максимальное кардинальное число - максимальное количество экземпляров сущностей, которые могут участвовать в связи.
Существует три типа бинарных связей, в зависимости от макс кард числа:
– «один-к-одному»: [1:1].
– «один-ко-многим»: [1:N]. В таких связях сущность на стороне 1 называют родительской, а на стороне N – дочерней.
– «многие-ко-многим»: [N:M];
минимальное кардинальное число - минимальное количество экземпляров сущностей, которые должны участвовать в связи.
Мин кард число, как правило, указывается как ноль или единица:
–равно 0, то участие некоторого экземпляра сущности в связи не является обязательным (optional);
–равно 1, то участие некоторого экземпляра сущности в связи является обязательным(mandatory);
Нотация: || one-mandatory; |< many-mandatory; O| one-optional; O< many-optional.

3. Модель «сущность-связь». Идентификационно-зависимые, слабые и сильные сущности. Сущности категория-подтип.

Идентификационно-зависимой называется такая дочерняя сущность, идентификатор которой содержит идентификатор родительской сущности (дочерняя сущность является логическим расширением или составной частью родительской). Мин кард число родительской сущности для идентификационно-зависимой сущности всегда равно 1. Все идентификационно-зависимые сущности - слабые. Пример: квартира(id_1) || – O < комната(id_2, id_1)

Слабой сущностью называется сущность, существование которой зависит от наличия родительской сущности (сильной сущности). Пример: книга(id_1) || – O < издание(id_2, id_1)

Возможно наличие в модели слабых сущностей, которые при этом не являются идентификационно-зависимыми, в этом случае условия зависимости реализуются дополнительными средствами бизнес-логики.

Сильной сущностью, соответственно, называется сущность, существование которой НЕ зависит от других сущностей. Пример: доктор |O – O< пациент.

Сущности категория-подтип
-родительская сущность (супертип) содержит атрибуты, общие для нескольких подтипов;
-подтип расширяет набор атрибутов родительской сущности;
-в ряде случаев один из атрибутов супертипа (дискриминатор) указывает на то, каким из подтипов является конкретный экземпляр сущности;
-подтипы могут быть взаимоисключающими (exclusive) и невзаимоисключающими (inclusive).
примеры:
а) Student(супертип); Дискриминатор:isGraduateStudent (кружок с крестом X с линией снизу); Взаимоисключающие подтипы:Undergraduate, Graduate
б) Student(супертип); Дискриминатора нет (кружок с линией снизу);
Невзаимоисключающие подтипы:Tennis_club, Chemistry_club

4. Модель «сущность-связь»: шаблоны моделирования типа «сопряжение» и «прототип-экземпляр», представление многозначных атрибутов и рекурсивных связей.

Шаблон «сопряжение»

Когда у сильных сущностей есть общая идентификационно зависимая слабая сущность, то есть существует “связь” через эту слабую сущность.

Фактически: многозначные атрибуты, связанные с сильной сущностью.

Пример: recipe[id_1] ‖ – |< ingredient_use(id_1,id_2) >O – O| ingredient[id_2]

Шаблон «прототип-экземпляр» необходим в случае, когда идентификационно-зависимая дочерняя сущность - физическая реализация некоторой абстрактной или логической родительской сущности.

Пример: class[id_1, numberHours, description] ‖ – O< section(id_2,id_1,classDays,time,teacher)

Также в шаблоне **«прототип-экземпляр»** могут быть использованы слабые, но не id-зависимые сущности.

Многозначные атрибут – атрибут, который содержит несколько значений для каждого экземпляра сущности определенного типа.

Пример: У человека может быть больше 1 мобильного телефона. Phone – многозначный атрибут.

модель: Man(man_id)[city,country] ‖ – O< Phone(man_id, phoneNumber)

Рекурсивная связь возникает, когда некоторый класс сущностей имеет связь с самим собой. Пример: Каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;

каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН.

МУЖЧИНА >| –стрелочка в себя– ‖

5. Процесс разработки БД и уровни моделирования информационных систем. Моделирование данных. Модель семантических объектов и ее основные элементы (объекты, атрибуты).

Процесс разработки БД:

Общие стратегии:

- разработка сверху вниз - получение абстрактной модели данных исходя из анализа стратегических целей организации, способов их достижения, а также информации и способов ее представления, которые необходимы для достижения этих целей; лучшее взаимодействие подсистем, глобальный охват;
- разработка снизу вверх - для разработки выбирается конкретная система, которая выполняет только часть функций предприятия; исходя из сформированных требований, выбранная подсистема создается быстрее и с меньшими рисками.

!!!Одна из основ эффективной реализации приложения базы данных – ясное представление модели предметной области.

Уровни моделирования информационных систем:

- внешние модели (external schema): представление пользователей о системе (user view);
- концептуальная модель (conceptual schema): абстрактное представление системы (данных);
- внутренние модели (internal schema).

Моделирование данных – процесс создания логического представления БД. (пользовательская

модель данных, модель требований к данным, концептуальная модель).

Модели данных:

внешняя модель – набор представлений пользователей о той части предметной области, с которой он сталкивается;

концептуальная модель – набор ключевых объектов предметной области, их взаимосвязей

и ограничений, накладываемых на объекты;

логическая модель – представление концептуальной модели в соответствии с логической

моделью, используемой для хранения данных;

физическая модель – описание физического представления, хранения и обработки данных.

БД является «моделью модели», то есть через объекты базы данных описывает представление пользователей о конкретной предметной области. При любом процессе разработки необходимо сформулировать требования и построить на их основе модель данных. Модель данных содержит языковые и изобразительные стандарты для своего представления. Существует модель «сущность – связь», семантическая объектная модель.

Модель семантических объектов и ее основные элементы (объекты, атрибуты):

Семантическая объектная модель – альтернативный способ представления модели данных.

Семантика - значение единиц языка.

Семантический объект – некоторый объект, который способен существовать независимо и может быть идентифицирован

- класс объектов – набор объектов определенного типа;
- экземпляр объекта – объект некоторого класса с заданными значениями атрибутов
- именованный набор атрибутов, исчерпывающе описывающий отдельную сущность).
- объект может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут – свойство, которым обладает объект;

- все объекты определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов (исчерпывающее описание);
- в общем случае, атрибуты могут быть: простыми (атрибут со значением элементарного типа), составными (набор простых атрибутов); объектными (атрибут является семантическим объектом)

6. Модель семантических объектов и ее основные элементы (объекты, атрибуты). Кардинальные числа атрибутов. Идентификаторы.

Модель семантических объектов и ее основные элементы (объекты, атрибуты):

Семантическая объектная модель – альтернативный способ представления модели данных.

Семантика - значение единиц языка.

Семантический объект – некоторый объект, который способен существовать независимо и может быть идентифицирован

- класс объектов – набор объектов определенного типа;
- экземпляр объекта – объект некоторого класса с заданными значениями атрибутов
- именованный набор атрибутов, исчерпывающе описывающий отдельную сущность).
- объект может являться абстракцией: реального физического объекта; события, действия или процесса (заказ, услуга, транзакция).

Атрибут – свойство, которым обладает объект;

- все объекты определенного класса обладают одинаковым набором атрибутов, но различаются значениями данных атрибутов (исчерпывающее описание);
- в общем случае, атрибуты могут быть: простыми (атрибут со значением элементарного типа), составными (набор простых атрибутов); объектными (атрибут является семантическим объектом)

Кардинальное число указывает количество значений, которые может принимать атрибут в корректном экземпляре объекта:

- максимальное кардинальное число - максимальное количество значений, которые может принимать атрибут – как правило, 1 или N;
- минимальное кардинальное число - минимальное количество значений, которые должен принимать атрибут – как правило, 0 или 1

Дополнительные термины для описания атрибутов:

- однозначный атрибут – кардинальность ..1;
- многозначный атрибут – кардинальность ..N;
- необъектный атрибут – простой или групповой.

Идентификатор объекта – набор атрибутов, которые определяют, именуют сущность:

- идентификаторы могут быть составными, т.е. состоять более чем из одного атрибута;
- обозначаются «ID» или «ID» в случае уникальности.

7. Типы семантических объектов. Простой объект. Составной объект. Составной объект с независимыми и вложенными группами.

Простой объект – объект, содержащий только однозначные простые или групповые атрибуты.

Составной объект – объект, содержащий один или более многозначный простой или групповой атрибуты. Пример: store-bill(ID invoiceNumber, LineItem[Date 1.1, Description1.1, Price1.1]0.N, TotalPrice1.1) LineItem-групповой.

-с **независимыми группами** – если групповые атрибуты не вложены.

Пример: object1(ID O1, …Group1[ID G1,…] 0.N, Group2[ID G2,…] 0.N)

-с **вложенными группами** – если групповые атрибуты вложены.

Пример: object1(ID O1,…Group1[ID G1,… , Group2[ID G2,…] 0.N] 1.N)

8. Типы семантических объектов. Сложный объект. Гибридный объект. Ассоциативный объект.

Сложный объект – объект, содержащий один или более объектный атрибут.

Атрибут называется объектными, если атрибут является семантическим объектом.

Пример:

Customer(ID CustomerID, Phone 1.1, Name 1.1,… Order 0.N),

Order(ID OrderID, Descriprion 1.1,… Customer 1.1)

Сложные объекты могут быть 1.1, 1.N, M.N

Гибридный объект – объект, содержащий один или более групповой атрибут, содержащий в себе объектный атрибут.

Атрибут называется групповым (составным), если атрибут состоит из набора простых атрибутов;

Пример гибридного объекта-общага:

Dormitory (ID DormName, ResidentAssistant 1.1, Phone 1.1, StudentRent[*Student* 1.1, Rent 0.1]1.N), *Student*(ID StudentName,… *Dormitory* 0.1)

Ассоциативный объект – объект, сопоставляющий два или более объектов, и содержащий дополнительную информацию о таком сопоставлении

Пример: ассоциативный объект – рейс, сопоставляет объекты самолет и пилот.

Flight(ID FlightID[FlightNumber, Date]1.1, FromCity 1.1, ToCity 1.1, …, *Airplane 1.1*, *Pilot* 1.1)

Airplane(ID AirplaneNumber,…Type 1.1,Capacity 1.1,… *Flight* 0.N)

Pilot(ID SpecialPilotID, ID Name, …Phone 1.1,Hours 1.1,… *Flight* 0.N)

9. Типы семантических объектов. Схема «родитель-подтип». Взаимоисключающие и вложенные подтипы. Схема «прототип-экземпляр».

Схема «родитель-подтип»

Родительский объект(объект-супертип) – содержит атрибуты, общие для нескольких объектов-подтипов (кард число определяет обязательность объекта-подтипа)

Объект-подтип расширяет набор атрибутов родительского объекта.

Обозначения атрибутов: если подтип – ST, если супертип - P

Пример: объект супертип-сотрудник employee, подтипы- менеджер, программист

Employee(ID EmplNum, ID EmplName, HireDate 1.1, Salary 1.1, *Manager* 0.ST, *Programmer* 0.ST)

Manager(*Employee* P, ManagerLevel 1.1,…)

Programmer(*Employee* P, Language 0.N, OperSystem 0.N)

Взаимоисключающие подтипы:

Кардинальность взаимоисключающих типов обозначается как X.Y.Z

-X (0/1) – минимальное кардинальное число;

-Y – минимальное количество атрибутов со значениями;

-Z – максимальное количество атрибутов со значениями.

Пример: Банк обслуживает клиентов – физических лиц (людей) и юридических лиц (компании)

Client(ID ClientNum, ID ClientName, Phone 1.1, [*Individual* 0.ST, *Company* 0.ST] 0.1.1)

Individual(Client P,…)

Company(Client P, …)

Вложенные подтипы:

Значит, что подтип супертипа1 также является супертипом.

Пример: Есть организации/корпорации, облагаемые и необлагаемые налогом. Организации, необлагаемые налогом – взаимоисключающие подтипы: школы, благотворительные организации.

*Corporation(*ContactName 1.1, … [*TaxableCorp* 0.ST, *NonTaxableCorp* 0.ST]1.1.1)

TaxableCorp(*Corporation* P, …)

NonTaxableCorp(*Corporation* P, …, ExemptionID[*School* 0.ST, *Charity* 0.ST] 1.1.1)

School(*NonTaxableCorp* P, SchoolDistrictName 1.1)

Charity(*NonTaxableCorp* P, CharityName 1.1)

Схема «прототип-экземпляр».

Объект-прототип является некоторым абстрактным объектом, обладающим набором версий.

Объект-версия описывает различные реализации объекта-прототипа. ID версии – составной из ID прототипа и какого-то поля версии.

Пример: книга и ее издание

TextBook(ID ISBN, Title 1.1, Author 1.1,…)

Edition(ID EditID[*TextBook* 1.1, EditNumber 1.1] 1.1, PubHouse 1.1, …)

Часть 3 – Реляционная модель и преобразование моделей данных

1. Понятие реляционной модели. Отношения. Ключи, их свойства и типы. Функциональная зависимость. Многозначная зависимость.

Реляционная модель:

- основана на применении концепции реляционной алгебры к проблеме хранения больших объемов данных.
- определяет способ хранения данных в виде таблиц со строками столбцами, при этом минимизируется дублирование и исключаются определенные типы ошибок обработки.
- дает стандартный способ структурирования и обработки данных
- фактически является промышленным стандартом хранения и обработки информации в базах данных.

Отношение – двумерная таблица, обладающая следующими свойствами:

- Каждая строка содержит данные, описывающие некоторую сущность или ее часть;
- Каждый столбец представляет один из атрибутов сущности;
- Каждая ячейка содержит одиночное значение;
- Все значения в одном столбце имеют один и тот же тип;
- Каждый столбец имеет уникальное имя;
- Не важен порядок строк;
- Не важен порядок столбцов;
- Не может существовать двух идентичных строк.

Ключ - один или несколько атрибутов, идентифицирующих строку отношения. уникальность ключа:

- Уникальный ключ – ключ, каждое из значений которого может присутствовать в отношении не более одного раза;
- Неуникальный ключ– ключ, каждое из значений которого может присутствовать в отношении более одного раза;

Составной (композитный) ключ – ключ, состоящий из более чем одного атрибута

Первичный ключ – один из уникальных ключей отношения, выбранный в качестве основного для идентификации записей отношения.

Ключ-кандидат – любой из уникальных ключей отношения, не являющийся первичным.

Суррогатный ключ – искусственно созданный уникальный ключ, используемый в качестве первичного ключа отношения; как правило, создаются средствами СУБД; значения суррогатных ключей бессмысленны для пользователей, соответственно, они не должны отображаться в формах и отчетах.

Внешний ключ – первичный ключ одного отношения, помещенный в качестве атрибута в другое отношение для формирования связи между отношениями. Очевидно, что внешний ключ, как и первичный, может быть составным. Внешние ключи также позволяют формировать ограничения ссылочной целостности

Функциональная зависимость имеет место, когда один из атрибутов (или несколько атрибутов) определяет значения другого атрибута (нескольких атрибутов).

Другими словами: по значению одного столбца можно однозначно определить соответствующее значение другого столбца (То же самое действительно и для групп столбцов.)

$A \to B \quad A \to (B,C) => A \to B, A \to C \quad (A,B) \to C \neq> A \to C, B \to C$

Левая часть функциональной зависимости называется детерминантом

определение функциональных зависимостей (факта уникальности детерминанта) производится на основе требований, а НЕ фактических значений атрибутов

Пример: Отношение – (OrderNumber, Item, Quantity, Price, ExtendedPrice)

(OrderNumber, Item) → (Quantity, Price, ExtendedPrice)

(Quantity, Price) → (ExtendedPrice)

Многозначная зависимость имеет место, когда детерминанту соответствует некоторый набор значений.

Другими словами: многозначная зависимость задается для многозначных атрибутов. Это означает, что, используя известное значение одного атрибута (столбца), можно однозначно определить набор значений другого многозначного атрибута.

Отношение - $R(A,B,C)$, $A \to\to B \quad A \to\to C \quad B,C$- независимы

возникает аномалия модификации – необходимо внести все комбинации для многозначно-зависимых атрибутов!

детерминант многозначной зависимости никогда не может быть первичным ключом.

Пример: ISBN →→ Authors. Значение ISBN книги всегда определяет всех ее авторов.

2. Реляционная модель: аномалии модификации. Функциональная зависимость. Нормализация. Первая, вторая, третья нормальные формы. Нормальная форма Бойса-Кодда.

Аномалии модификации – нежелательные побочные эффекты, возникающие при модификации записей отношения:

- аномалия удаления: удаление единственной строки отношения приводит к удалению информации о двух сущностях;
- аномалия вставки: невозможность вставки информации об одной сущности до тех пор, пока не будет внесен некоторый факт о другой сущности;
- аномалия обновления: некорректные результаты операции обновления.

Функциональная зависимость имеет место, когда один из атрибутов (или несколько атрибутов) определяет значения другого атрибута (нескольких атрибутов).

Другими словами: по значению одного столбца можно однозначно определить соответствующее значение другого столбца (То же самое действительно и для групп столбцов.)

$A \to B \quad A \to (B, C) => A \to B, A \to C \quad (A, B) \to C \neq> A \to C, B \to C$

Левая часть функциональной зависимости называется детерминантом

определение функциональных зависимостей (факта уникальности детерминанта) производится на основе требований, а НЕ фактических значений атрибутов

Пример: Отношение– (OrderNumber, Item, Quantity, Price, ExtendedPrice)

(OrderNumber, Item) → (Quantity, Price, ExtendedPrice)

(Quantity, Price) → (ExtendedPrice)

Нормализация – процесс классификации и преобразования отношений с целью исключения аномалий модификации.

Источник аномалии	Норм. Формы	Принципы проектирования
Функциональные зависимости	1НФ, 2НФ, 3НФ, БКНФ	БКНФ: каждый детерминант – ключ-кандидат
Многозначные зависимости	4НФ	4НФ: каждая мн.зав. должна формировать отдельное отношение
Ограничения	5НФ, ДКНФ	ДКНФ: каждое ограничение должно являться следствием ключа-кандидата или домена. Домен– именованное множество допустимых значений атрибута

1НФ: любая таблица, удовлетворяющая определению отношения, находится в первой нормальной форме

2НФ: отношение находится во второй нормальной форме, если каждый из неключевых атрибутов зависит от всего первичного ключа. Это означает, что если (A, B) - первичный ключ, тогда таблица не содержит столбцов, зависящих только от A или только от B.

Необходимо:

- выделить атрибуты функциональной зависимости, нарушающей требования 2НФ, в отдельное отношение (с детерминантом в качестве первичного ключа);
- оставить детерминант в качестве внешнего ключа в исходном отношении.

3НФ: отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и не имеет транзитивных зависимостей (т.е. где $A \to B \to C \to \cdots$). Т.е. отсутствуют функциональные зависимости между неключевыми атрибутами.

Необходимо: выделить функциональную зависимость в отдельное отношение (аналогично 2НФ).

БКНФ: отношение находится в нормальной форме Бойса-Кодда, если каждый детерминант является ключом-кандидатом.

Необходимо:

1. выделить все функциональные зависимости;
2. определить все ключи-кандидаты;
3. если существует функциональная зависимость, детерминант которой не является ключом-кандидатом необходимо:

– выделить атрибуты, участвующие в этой функциональной зависимости в новое отношение;

– в качестве первичного ключа нового отношения будет детерминант функциональной зависимости;

– детерминант функциональной зависимости также должен остаться в исходном отношении в качестве внешнего ключа (с заданием ограничения ссылочной целостности);

4. шаги могут повторяться многократно до тех пор, пока для каждого из отношений не выполняются требования БКНФ.

3. Реляционная модель: аномалии модификации. Многозначная зависимость. Нормализация. Четвертая нормальная форма. Пятая нормальная форма. Нормализация и денормализация.

Аномалии модификации – нежелательные побочные эффекты, возникающие при модификации записей отношения:

- аномалия удаления: удаление единственной строки отношения приводит к удалению информации о двух сущностях;
- аномалия вставки: невозможность вставки информации об одной сущности до тех пор, пока не будет внесен некоторый факт о другой сущности;
- аномалия обновления: некорректные результаты операции обновления.

Многозначная зависимость имеет место, когда детерминанту соответствует некоторый набор значений.

Другими словами: многозначная зависимость задается для многозначных атрибутов. Это означает, что, используя известное значение одного атрибута (столбца), можно однозначно определить набор значений другого многозначного атрибута.

Отношение - $R(A, B, C)$, $A \to\to B \quad A \to\to C \quad B, C$- независимы

возникает аномалия модификации – необходимо внести все комбинации для многозначно-зависимых атрибутов!

детерминант многозначной зависимости никогда не может быть первичным ключом.

Пример: ISBN →→ Authors. Значение ISBN книги всегда определяет всех ее авторов.

Нормализация – процесс классификации и преобразования отношений с целью исключения аномалий модификации.

Источник аномалии	Норм. Формы	Принципы проектирования
Функциональные зависимости	1НФ, 2НФ, 3НФ, БКНФ	БКНФ: каждый детерминант – ключ-кандидат
Многозначные зависимости	4НФ	4НФ: каждая мн.зав. должна формировать отдельное отношение
Ограничения	5НФ, ДКНФ	ДКНФ: каждое ограничение должно являться следствием ключа-кандидата или домена. Домен– именованное множество допустимых значений атрибута

4НФ: Отношение находится в четвертой нормальной форме, если оно находится в БКНФ и не имеет нетривиальных многозначных зависимостей.

проблема возникает, если:

• существуют атрибуты, не участвующие в многозначной зависимости;

• существуют несколько независимых многозначных зависимостей.

5НФ:

–рассматривает случаи, когда информация может быть восстановлена из отдельных составляющих, обладающих меньшей избыточностью;

–в случае отсутствия дополнительных семантических правил 4НФ и 5НФ эквивалентны;

–при условии отсутствия составного первичного ключа и выполнении условий БКНФ, отношение автоматически находится в 5НФ.

Нормализация и денормализация:

преимущества нормализации:

– отсутствие аномалий модификации;

– минимизация избыточности данных: • более простое поддержание целостности данных; • меньшие накладные расходы на хранение;

недостатки нормализации:

– более сложные запросы для извлечения данных;

– следовательно, большие накладные расходы на их выполнение, что снижает производительность;

Денормализация — намеренное приведение структуры базы данных в состояние, не удовлетворяющее требованиям нормализации. Денормализация обычно проводится путем добавления тех данных, которые по требованиям той или иной нормальной формы должны выноситься в отдельную таблицу.

Простыми словами: нормализация - подразумевает хранение информации максимально просто и не избыточно. Максимально просто: не хранить в одном столбце ФИО, а сделать 3 отдельных. Не избыточно: фамилия клиента должна храниться только в справочнике клиентов, ее не нужно добавлять в сделанные клиентом заказы. Денормализация - процесс противоположный нормализации.

В связи с этим выделяют типы базы данных:

– **обновляемая** («updateable», operational): **OLTP** (OnLine Transaction Processing). Их основная цель - ввод/редактирование/удаление данных в режиме онлайн.

Примеры использования: мессенджеры, социальные сети, 1С: Бухгалтерия и т.д. везде повсеместно.

– преимущественно не обновляемая («read mostly», non-operational): **OLAP** (OnLine Analytic Processing) - это базы данных, которые служат непосредственно для проведения быстрого анализа больших объемов данных.

Обычно такие БД используются на больших предприятиях для построения аналитической отчетности за большой промежуток времени (месяц, квартал, год).

OLTP придерживаются принципов нормализации, OLAP - денормализации.

4. Преобразование модели «сущность-связь»: основные этапы. Создание таблиц для каждой из сущностей. Определение свойств столбцов таблицы. Нормализация.

Основные этапы:

1. создание таблицы для каждой сущности:
   - определение первичного ключа (возможно, суррогатного);
   - определение ключей-кандидатов;
   - определение свойств каждого столбца: типа данных (data type), возможности неопределенного значения (null value), значения по умолчанию (default value), ограничений на значения (data constraints);
   - проверка нормализации (как правило, нормализация производится до БКНФ/4НФ);
2. создание связей с помощью внешних ключей:
   - между сильными сущностями (1:1, 1:N, N:M);
   - для идентификационно-зависимых сущностей;
   - для слабых сущностей;
   - для сущностей тип-подтип;
   - рекурсивных;
3. обеспечение условий минимальной кардинальности

Создание таблиц (отношений) для каждой из сущностей

Определение первичного ключа:
- обозначается символом ключика
- идеальным является короткий, неизменяемый ключ числового типа.
- **суррогатный ключ** – искусственно созданный уникальный ключ, используемый в качестве первичного ключа

Определение ключей-кандидатов:
- **ключ-кандидат** (candidate/alternate key) – любой из уникальных ключей отношения, не являющийся первичным
- для обозначения можно использовать AKn.m (n – номер ключа-кандидата, m – номер атрибута в соответствующем ключе-кандидате)

Пример: * - ключ
Employee(*EmplNumber, EmplName,Phone, Email AK1.1,…)
Customer(*CustomerNumber, Name AK1.1, City AK1.2, Phone, Email AK2.1)

Определение свойств столбцов таблицы:

типы данных: binary, char, varchar, nvarchar, datetime, image, integer, money, numeric, smalldatetime, smallint, smallmoney, text, tinyint, …

неопределенные значения (missing values, null values): неприменимое значение (inappropriate value), неизвестное значение (unknown value), не введенное значение.

значения по умолчанию - автоматически создаваемое СУБД значение при вставке новой записи, если оно не задано явно

Категории (ограничения) целостности данных:
- сущностная: определяет строку как уникальную сущность в конкретной таблице, поддерживается через введение ключа;
- доменная: определяет достоверность записей в конкретном столбце;
- ссылочная: сохраняет определенные связи между таблицами при вводе или удалении записей;
- пользовательская целостность (ограничения на значения столбцов, опред_мые бизнес-логикой);

ограничения на значения:
- домена: значения атрибута должны быть из определенного множества;
- интервальные: значения атрибута должны быть в определенном интервале;
- внутренние: значения атрибута должны удовлетворять определенным условиям сравнения с другими атрибутами того же отношения;
- внешние: значения атрибута должны удовлетворять определенным условиям сравнения с атрибутами другого отношения ограничения ссылочной целостности посредством внешних ключей).

Нормализация – процесс классификации и преобразования отношений с целью исключения аномалий модификации.

Аномалии модификации – нежелательные побочные эффекты, возникающие при модификации записей отношения (аномалии удаления, вставки, обновления)

Как происходит нормализация отношений:
- Минимизация избыточности данных (если одни и те же данные хранятся в нескольких местах, то это избыточная информация).

Пример: вводятся суррогатные ключи в качестве первичных
Customer(*Email, *Phone,…) → Customer(*CustomerID, … Email AK1.1, Phone AK1.2)
- Мин. Исп. Неопределенных значений. (Из-за неопределенности интерпретации null-значений их использование лучше свести к минимуму)
- Предотвращение потери информации.

Пример: Таблица student где есть поле course. Если студент1 решит не изучать немецкий язык, то придется удалить запись из таблицы, тогда потеряна информация о курсе. Решить можно, например так – добавить атрибут для истории прошлых курсов студента.

5. Преобразование модели «сущность-связь»: основные этапы. Создание связей: сильные сущности, идентификационно-зависимые сущности, шаблоны «сопряжение» и «прототип-экземпляр».

Основные этапы:

1. создание таблицы для каждой сущности:
   - определение первичного ключа (возможно, суррогатного);
   - определение ключей-кандидатов;
   - определение свойств каждого столбца: типа данных (data type), возможности неопределенного значения (null value), значения по умолчанию (default value), ограничений на значения (data constraints);
   - проверка нормализации (как правило, нормализация производится до БКНФ/4НФ);
2. создание связей с помощью внешних ключей:
   - между сильными сущностями (1:1, 1:N, N:M);
   - для идентификационно-зависимых сущностей;
   - для слабых сущностей;
   - для сущностей тип-подтип;
   - рекурсивных;
3. обеспечение условий минимальной кардинальности

Создание связей:

сильные сущности

1:1 Первичный ключ отношения, соответствующего одной из сущностей, помещается в качестве внешнего ключа (FK) в отношение, представляющее вторую сущность.

Пример: Club_member(*memNum, memName,…) |O – O| Locker(*LockerNum,…memNum FK AK1.1)

1:N Первичный ключ отношения, соответствующего родительской сущности, помещается в качестве внешнего ключа в отношение, представляющее дочернюю сущность.

Пример: Flat(*FlatNum,..) ‖ – O < Room(*RoomNum, …, FlatNum FK)

N:M Создается промежуточное отношение:
- содержит в качестве внешних первичные ключи обоих отношений;
- оба внешних ключа формируют составной первичный ключ нового отношения;
- другие поля отсутствуют.

Пример: Company(*CompName,City,…) ‖ – O < Comp_Part_Int(*CompName FK, *PartNum FK) >| – ‖ Part(*PartNum, PartName,…)

идентификационно-зависимые сущности

идентификационно-зависимой называется дочерняя сущность, идентификатор которой содержит идентификатор родительской сущности (дочерняя сущность является логическим расширением или составной частью родительской);

Пример: Company(*CompName, City, County) ‖ – O < Phone(*CompName FK, *PhoneNum)

Phone – многозначный атрибут

Варианты использования id-зависимых сущностей:
– шаблон «сопряжение»;
– многозначные атрибуты;
– шаблон «прототип-экземпляр»;

шаблон «сопряжение»

Создается промежуточное отношение:
– содержит в качестве внешних первичные ключи обоих отношений;
– оба внешних ключа формируют составной первичный ключ нового отношения;
– присутствуют дополнительные атрибуты

Пример: Company(*CompName,City,…) ‖ – O < Comp_Part_Int(*CompName FK, *PartNum FK, Price) >| – ‖ Part(*PartNum, PartName,…)

шаблон «прототип-экземпляр»

Случай, когда id-зависимая дочерняя сущность - физическая реализация некоторой абстрактной или логической родительской сущности.

Пример: class(*ClassName)[numberHours, description] ‖ – O< section(*ClassName FK, *refNum)[classDays,time,teacher]

Также в шаблоне **«прототип-экземпляр»** могут быть использованы слабые, но не id-зависимые сущности.

Тогда: class(*ClassName)[numberHours, description] ‖ – O< section(*refNum)[classDays,time,teacher, ClassName FK]

слабая сущность - сущность, существование которой зависит от наличия родительской сущности. Все id-зависимые – слабые.

6. Преобразование модели «сущность-связь»: основные этапы. Создание связей: шаблон «категория-подтип», представление многозначных атрибутов. Рекурсивные связи.

Основные этапы:

1. создание таблицы для каждой сущности:
   - определение первичного ключа (возможно, суррогатного);
   - определение ключей-кандидатов;
   - определение свойств каждого столбца: типа данных (data type), возможности неопределенного значения (null value), значения по умолчанию (default value), ограничений на значения (data constraints);
   - проверка нормализации (как правило, нормализация производится до БКНФ/4НФ);
2. создание связей с помощью внешних ключей:
   - между сильными сущностями (1:1, 1:N, N:M);
   - для идентификационно-зависимых сущностей;
   - для слабых сущностей;
   - для сущностей тип-подтип;
   - рекурсивных;
3. обеспечение условий минимальной кардинальности

Создание связей:

шаблон «категория-подтип»

-родительская таблица (супертип) содержит атрибуты, общие для нескольких подтипов;
-подтип расширяет набор атрибутов родительской таблицы;
-в ряде случаев один из атрибутов супертипа (дискриминатор) указывает на то, каким из подтипов является конкретный экземпляр отношения;
-подтипы могут быть взаимоисключающими (exclusive) и невзаимоисключающими (inclusive).

примеры:

а) Student(супертип); Дискриминатор:isGraduateStudent (кружок с крестом X с линией снизу); Взаимоисключающие подтипы:Undergraduate, Graduate
Student(*StudentID, Name, IsGradStudent)
Undegraduate(*StudentID FK,GPA, ExamScore)
Graduate(*StudentID FK, GovExamScore)

б) Student(супертип); Дискриминатора нет (кружок с линией снизу); Невзаимоисключающие подтипы:Tennis_club, Chemistry_club

представление многозначных атрибутов

Многозначные атрибут – атрибут, который содержит несколько значений для каждого экземпляра отношения определенного типа.
Пример: У человека может быть больше 1 мобильного телефона. Phone – многозначный атрибут.
Man (*ManName, City, County) ‖ – O < Phone(*ManName FK, *PhoneNum)

Рекурсивные связи

Возникают, когда отношение имеет связь с самим собой.
1:1 в отношение добавляется внешний ключ
Пример: Каждый вагон поезда имеет связь с вагоном впереди
Boxcar(*BoxCarNum, capacity, BoxCarNumAhead FK AK1.1) |O –стрелочка в себя– O|
1:N в отношение добавляется внешний ключ
Пример: Каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;
каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН.
МУЖЧИНА(*Паспорт, Имя, Отец FK) ‖ –стрелочка в себя– O|
M:N добавляется id-зависимая сущность без дополнительных полей.
Пример: Часть компании может иметь свои части
Part(*PartName, Description) ‖–contains part– O| Part_Part(*PartName FK, *ContainedPartName FK) >|–is contained in part стрелка в Part– ‖

7. Преобразование модели «сущность-связь»: основные этапы. Создание связей: ограничения минимальной кардинальности связей.

Основные этапы:

1. создание таблицы для каждой сущности:
   - определение первичного ключа (возможно, суррогатного);
   - определение ключей-кандидатов;
   - определение свойств каждого столбца: типа данных (data type), возможности неопределенного значения (null value), значения по умолчанию (default value), ограничений на значения (data constraints);
   - проверка нормализации (как правило, нормализация производится до БКНФ/4НФ);
2. создание связей с помощью внешних ключей:
   - между сильными сущностями (1:1, 1:N, N:M);
   - для идентификационно-зависимых сущностей;
   - для слабых сущностей;
   - для сущностей тип-подтип;
   - рекурсивных;
3. обеспечение условий минимальной кардинальности

Создание связей: ограничения минимальной кардинальности связей:

Связи могут иметь следующие варианты минимальной кардинальности:

- O-O (optional-optional): родительская и дочерняя сущности являются необязательными;
- M-O (mandatory-optional): родительская сущность является обязательной, дочерняя сущность является необязательной;
- O-M (optional-mandatory): родительская сущность является необязательной, дочерняя сущность является обязательной;
- M-M (mandatory-mandatory): родительская и дочерняя сущности являются обязательными.

Часто для обозначения процедуры обеспечения ограничений минимальной кардинальности используют термин action (действие);
Рассматриваемые операции над записями: вставка (insert); изменение (update) первичного или внешнего ключа; удаление (delete).

Каскадные ограничения ссылочной целостности

-каскадное обновление (cascade update) имеет место, когда необходимо изменить значение(я) внешнего ключа (в дочерней таблице) при изменении первичного ключа родительской. (суррогатные ключи не изменяются, следовательно, для них не нужно каскадное обновление)
- каскадное удаление (cascade delete) имеет место, когда необходимо удалить строки в дочерней таблице при удалении записи в родительской таблице. Используется для слабых сущностей, для сильных нет.

Действия, необходимые для обеспечения ограничений мин кард связей:

O-O не требует никаких действий.
M-M = наборы действий M-x и x-M
Обеспечивается средствами СУБД (путем создания триггеров) и внешними к СУБД средствами

M-x (**M-O** в том числе)

операция	Над родительской таблицей	над дочерней таблицей
вставка	-	подбор новой род. записи / запрет
изменение	каскад. обновление /запрет	Допустимо, если новое значение FK соотв. PK в род. таблице / запрет
удаление	каскад. удаление /запрет	-

Обеспечивается средствами СУБД (огр ссылочной целостности, задание условия not null для FK)
Пример: Department ‖ – O< Employee

x-M (**O-M** в том числе)

операция	Над родительской таблицей	над дочерней таблицей
вставка	подбор новое дочерней записи / запрет	-
изменение	обновление FK хотя бы 1 дочерней записи / запрет	Если есть другие дочерние записи у соотв род. записи – ничего. Иначе – запрет/ поиск замены
удаление	-	Если есть другие дочерние записи у соотв род. записи – ничего. Иначе – запрет/ поиск замены

Обеспечивается средствами СУБД (путем создания триггеров) и внешними к СУБД средствами
Пример: Department |O – |< Employee

8. Преобразование модели семантических объектов. Типы объектов: простой, составной, сложный.

Простой объект – объект, содержащий только однозначные простые или групповые атрибуты.

Преобразование: ID становится PK

Пример: Customer(ID CustomerID, Phone 1.1,…) → Customer(*CustomerID, …)

Составной объект – объект, содержащий один или более многозначный простой или групповой атрибуты.

Преобразование: Каждый групповой атрибут – новая таблица с PK = (FK на изначальный объект и поле ID группового объекта)

- с **независимыми группами** – если групповые атрибуты не вложены.

Пример: store-bill(ID invoiceNumber, DailyItem[ID Date 1.1, Description1.1, Price1.1]0.N, TotalPrice1.1) → store-bill(*invoiceNumber, TotalPrice) ‖ – O< DailyItem(*invoiceNumber FK, *Date,…)

- с **вложенными группами** – если групповые атрибуты вложены.

Пример: object1(ID O1,…Group1[ID G1,… , Group2[ID G2,…] 0.N] 1.N) → object1(*O1,…) ‖ – |< object1_1(*O1 FK, *G1,…) ‖ – O< object1_2(*O1 FK, *G1 FK, *G2…)

Сложный объект – объект, содержащий один или более объектный атрибут.

Атрибут называется объектными, если атрибут является семантическим объектом.

Преобразование:

1:1 – в одно из отношений добавляется FK = ID второго отношения

Пример: obj1(ID O1,… obj2 1.1), obj2(ID O2,…,obj1 1.1) → obj1(*O1,…) ‖ – ‖ obj2(*O2,…, O1 FK)

или obj1(*O1,…, O2 FK) ‖ – ‖ obj2(*O2,…)

1:N – в отношение, с которым связь 1:N добавляется FK = ID второго отношения

Пример: obj1(ID O1,… obj2 1.N), obj2(ID O2,…,obj1 0.1) → obj1(*O1,…) |O – |< obj2(*O2,…, O1 FK)

M:N – создается новое отношение без других атрибутов, с PK= (FK=ID 1 отношения, FK= ID 2 отношения)

Пример: obj1(ID O1,… obj2 1.N), obj2(ID O2,…,obj1 0.N) →
obj1(*O1,…) ‖ – |< obj_1_2(*O1 FK, *O2 FK) >O – ‖ obj2(*O2, …)

9. Преобразование модели семантических объектов. Типы объектов: гибридный, ассоциативный, «родитель-подтип», «прототип-экземпляр».

Гибридный объект – объект, содержащий один или более групповой атрибут, содержащий в себе объектный атрибут.

Атрибут называется групповым (составным), если атрибут состоит из набора простых атрибутов;

Преобразование: для группового объекта создается отдельная таблица. Её PK состоит из FK = ID изначального объекта, FK = ID группового объекта (это или просто поле группового объекта, либо внутренний объект). Если ID группового объекта - просто поле, то в таблицу добавляется FK= ID внутреннего объекта.

Пример:

- ID группового объекта - внутренний объект

obj1(ID O1,… Group1[ID obj2 1.1,…] 1.N), obj2(ID O2,…,obj1 0.N) →
obj1(*O1,…) ‖ – |< obj_group1(*O1 FK, *O2 FK) >O – ‖ obj2(*O2, …)

- ID группового объекта - другое поле

obj1(ID O1,… Group2[ID G2, obj3 1.1,…] 0.N), obj3(ID O3,…,obj1 0.N) →
obj1(*O1,…) ‖ – O< obj_group2(*O1 FK, *G2, O3 FK) >O – ‖ obj3(*O3, …)

Ассоциативный объект – объект, сопоставляющий два или более объектов, и содержащий дополнительную информацию о таком сопоставлении

Преобразование: в таблицу ассоциативного объекта добавляются внешние ключи FK= ID всех сопоставляемых объектов. Если ID ассоциативного объекта составной, то в таблицу добавляются поля составного ID в качестве PK.

Пример: ассоциативный объект – рейс, сопоставляет объекты самолет и пилот.

Flight(ID FlightID[FlightNumber, Date]1.1, FromCity 1.1, ToCity 1.1, …, *Airplane 1.1*, *Pilot* 1.1)
Airplane(ID AirplaneNumber, Type 1.1, Capacity 1.1,… *Flight* 0.N)
Pilot(ID SpecialPilotID, ID Name, …Phone 1.1, Hours 1.1,… *Flight* 0.N)
→
Airplane(*AirplaneNumber, Type, Capacity,…) ‖ – O<
Flight(*FlightNumber, *Date, FromCity, ToCity, …, AirplaneNumber FK, SpecialPilotID FK) >O – ‖
Pilot(*SpecialPilotID, Name, …Phone, Hours,…)

«родитель-подтип»

Родительский объект(объект-супертип) – содержит атрибуты, общие для нескольких объектов-подтипов (кард число определяет обязательность объекта-подтипа)

Объект-подтип расширяет набор атрибутов родительского объекта.

Обозначения атрибутов: если подтип – ST, если супертип - P

Преобразование: в таблицах подтипов PK = ID супертипа.

обозначение в схеме для взаимоисключающих подтипов: связи от супертипа к подтипам перечеркнуты —— X.Y.Z

Если подтипы не взаимоисключающие, то особых обозначений в схеме нет.

Кардинальность взаимоисключающих типов обозначается как X.Y.Z

-X (0/1) – минимальное кардинальное число;

-Y – минимальное количество атрибутов со значениями;

-Z – максимальное количество атрибутов со значениями

Пример: объект супертип-сотрудник employee, подтипы- менеджер, программист

Employee(ID EmplNum, ID EmplName, HireDate 1.1, Salary 1.1, [*Manager* 0.ST, *Programmer* 0.ST]0.1.1)
Manager(*Employee* P, ManagerLevel 1.1,…)
Programmer(*Employee* P, Language 0.N, OperSystem 0.N)
→
Employee(* EmplNum, EmplName, HireDate, Salary) ——— 0.1.1
–‖*Manager*(* EmplNum,…)
–‖*Programmer*(* EmplNum, Language, OperSystem)

«прототип-экземпляр»

Объект-прототип является некоторым абстрактным объектом, обладающим набором версий.

Объект-версия описывает различные реализации объекта-прототипа.

Преобразование: в таблицах экземпляров PK = ID прототипа = (FK=ID экземпляра и какое-то поле прототипа).

Пример: книга и ее издание

TextBook(ID ISBN, Title 1.1, Author 1.1,…)
Edition(ID EditID[*TextBook* 1.1, EditNumber 1.1] 1.1, PubHouse 1.1, …)
→
TextBook(*ISBN, Author 1.1,…)
Edition(**TextBook* FK, *EditNumber, PubHouse, …)

Часть 4 – Язык SQL

1. Ядро СУБД. Язык SQL: стандарты, категории операторов.

Ядро СУБД(database engine) основная служба, обеспеч. хранение, обработку и защиту данных:
1) преобразование запросов в действиях над данными
2) управление транзакциями и блокировками
3) резервное копирование и восстановление
Язык SQL - (“язык структурированных запросов”) - язык манипулирования реляционными данными. Представляет собой совокупность операторов, инструкций и вычисляемых функций.
Стандарты:
-Язык был разработан в конце 1970-х в IBM Corporation
-Был принят в качестве национального стандарта в 1992 году. Известен как SQL-92
-Новые версии стандарта включает в себя элементы XML и объектно-ориентированных концепций:
SQL:1999, SQL:2003, SQL:2006, SQL:2008
Категории операторов:
-операторы определения данных - CREATE, ALTER, DROP
-операторы манипуляции данными - SELECT, INSERT, UPDATE, DELETE
-операторы определения доступа к данным - GRANT, REVOKE, DENY
-операторы управления транзакциями - COMMIT, ROLLBACK

2. Язык SQL. Выборка данных: структура оператора SELECT. Указание списка выбора и источника. Использование псевдонимов.

Выборка данных: SELECT:

- FROM – источник выборки;
- WHERE – условие выборки;
- HAVING – условие группировки;
- GROUP BY – список группировки;
- ORDER BY – сортировка результатов;
- UNION, EXCEPT, INTERCEPT – комбинирование результатов нескольких. выборок.

SELECT: список выбора и указание источника:

- Список выбора:
  - список всех полей (SELECT *);
  - отдельные поля;
  - определение порядка отдельных полей;
  - псевдонимы для полей;
  - использование выражений и выражений с псевдонимами (!);
- Источник (FROM…):
  - единственная таблица;
  - несколько таблиц;
  - псевдонимы для таблиц;
  - Фильтрация уникальных строк: DISTINCT (!!).

(!): Немного про использование выражений в списке выбора. Рассмотрим следующий запрос:
SELECT ProductName + ' (' + Manufacturer + ')', Price, Price * ProductCount
FROM Products
Тогда при выборке будут создаваться три столбца. Первый столбец представляет результат объединения двух столбцов ProductName и Manufacturer. Второй столбец - стандартный столбец Price. А третий столбец представляет значение столбца Price, умноженное на значение столбца ProductCount.
(!!): Оператор DISTINCT позволяет выбрать уникальные строки. Например, в таблице может быть по несколько товаров от одних и тех же производителей. Выберем всех производителей:
SELECT DISTINCT Manufacturer
FROM Products
В данном случае критерием разграничения строк является столбец Manufacturer. Поэтому в результирующей выборке будут только уникальные значения Manufacturer. И если, к примеру, в базе данных есть два товара с производителем Apple, то это название будет встречаться в результирующей выборке только один раз.
SELECT: псевдонимы полей. SELECT [PartID] **as [Identity]**, [PartNumber], [Description] FROM [Parts] (если таблица состоит из 3 колонок: Identity, PartNumber, Descpription)
Синтаксис: SELECT *column_name* AS *alias_name*
FROM *table_name;*
SELECT: псевдонимы таблиц. SELECT a.[PartNumber], a.[Description], c.[SupplierCode], b.[Price], b.[DeliveryTime] FROM [Parts] as a, [Supply] as b, [Suppliers] as c WHERE a.[PartID] and b.[Supplier] = c.[SupplierID]
(таблица из5 колонок: PartNumber, Description, SupplierCode, Price, DeliveryTime).
Синтаксис:
SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

3. Язык SQL. Выборка данных: условие отбора записей и сортировка. Значение NULL и трехзначная логика.

Условие выбора записей(WHERE…)–задается с помощью логич.выражения, формир.с помощью:

- операторов сравнения (>, <, =, <>, <=, >=);
- логических операторов (AND, OR, NOT);
- задания диапазона (value BETWEEN min AND max);
- условия принадлежности множеству значений (value IN (list), список может формироваться с помощью вложенных запросов);
- условия непустого множества строк, возвращаемого в результате выполнения подзапроса (EXISTS (subquery));
- задания шаблона для строковых значений (value LIKE ‘…’) – символ % является шаблоном подстроки; символ _ является шаблоном единственного символа; [],[^] позволяют указать/ограничить допустимый набор/диапазон символов;
- определения наличия неопределенного значения (IS | IS NOT NULL).

Сортировка (ORDER BY):
ASC – по возрастанию (по умолчанию); DESC – по убыванию.(цифры в начале)
Сортировка строк в выборке: SELECT a.[PartNumber], a.[Description] FROM [Parts] as a ORDER BY a.[PartNumber]
Порядок сортировки: SELECT a.[PartNumber], a.[Description] FROM [Parts] as a ORDER BY a.[PartNumber] asc/desc
Трехзначная логика:
В SQL используется трехзначная логика, т.е. результат логического выражения может принимать три значения – истина (True), ложь (False) и неизвестно (NULL). Если значение в ячейке отсутствует (ячейка пуста, не заполнена), то говорят, что значение в этой ячейке есть NULL.
Следует четко понимать, что NULL – это не ноль, это вообще не значение, это отсутствие всякого значения.
NULL в Системах управления базами данных (СУБД) — специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных (БД). NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения». Введено для того, чтобы различать в полях БД пустые (визуально не отображаемые) значения (например, строку нулевой длины) и отсутствующие
значения (когда в поле не записано вообще никакого значения, даже пустого).
При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равно NULL.
Пример: SELECT * FROM suppliers WHERE Name IS (NOT) NULL
ISNULL
Выражение: В этом параметре мы указываем выражение, в котором нам нужно проверить значения NULL.
Замена: мы хотим заменить NULL определенным значением. Нам нужно указать значение замены здесь
Функция SQL Server ISNULL возвращает значение замены, если первое выражение параметра оценивается как NULL. SQL Server преобразует тип данных замены в тип данных выражения.
SELECT ISNULL(NULL, 100) result;

4. Язык SQL. Выборка данных: группировка, условия группировки и функции агрегирования.

FROM–источник выборки;
•WHERE–условие выборки;
•HAVING–условие группировки;
•GROUP BY–список группировки;
•ORDER BY–сортировка результатов;
•UNION, EXCEPT, INTERCEPT –комбинирование результатов нескольких выборок.
список полей, по которым осуществляется группировка:
GROUP BY …
•HAVING позволяет задавать условия с функциями агрегирования -данное условие выполняется после отбора записей по условию WHERE;
•функции агрегирования: COUNT, MIN, MAX, SUM, AVG
–все поля из списка выборки, не задействованные в функциях агрегирования, должны присутствовать в списке GROUP BY;
–с помощью COUNT можно подсчитывать количество строк в таблице (COUNT(*))или значений в столбце

5. Язык SQL. Выборка данных: комбинация результатов нескольких выборок и их сортировка.

• UNION – объединение результатов с исключением повторяющихся записей; Оператор языка SQL UNION предназначен для объединения результирующих таблиц базы данных, полученных с применением слова SELECT. Условие объединения результирующих таблиц: совпадение числа, порядка следования и типа данных столбцов.
При использовании оператора UNION без слова ALL результат не содержит дубликатов.
• UNION ALL - объединение результатов без исключения повторяющихся записей;
Все то же самое, что для UNIOIN, НО результат содержит дубликаты
• EXCEPT – исключение результатов второй выборки из результатов первой; оператор SQL EXCEPT возвращает те строки, которые возвращает первый запрос, и которых нет среди строк, возвращаемых вторым запросом.
• INTERSECT – получение пересечения результатов выборок. оператор SQL INTERSECT возвращает те и только те строки, которые возвращает и первый, и второй запросы
В этих конструкциях запросы могут иметь условия в секции WHERE, а могут не иметь их.
синтаксис:

```
SELECT ИМЕНА_СТОЛБЦОВ (1..N)
  FROM ИМЯ_ТАБЛИЦЫ
  UNION
  SELECT ИМЕНА_СТОЛБЦОВ (1..N)
  FROM ИМЯ_ТАБЛИЦЫ
```

аналогичный синтаксис для остальных команд, но UNION меняется на UNION ALL, EXCEPT, INSERTSECT
Для сортировки полученной выборки применяют ORDER BY. Оператор следует размещать только в конце составного запроса.

6. Язык SQL. Выборка данных: соединения и их типы.

внутреннее соединение: INNER JOIN(JOIN)
внешнее соед:LEFT OUTER JOIN(LEFT JOIN),RIGHT OUTER JOIN(RIGHT JOIN),FULL OUTER JOIN(FUL JOIN)
Запрос с оператором INNER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON.
Выведет только те строки, если условие соединения выполняется (является истинным, т.е. TRUE). В запросах необязательно прописывать INNER – если написать только JOIN, то СУБД по умолчанию выполнить именно внутреннее соединение
Пример: таблица1 колонки: 1, 2,3. Таблица2 колонки: 4,5,6

```
SELECT имя_табл2.5, имя_табл1.1 AS 6, имя_табл1.3
        FROM имя_табл2 INNER JOIN имя_табл1
                ON имя_табл2.6 = имя_табл1.1
```

(соединение таблиц так, чтобы были поля 5,6,3 и данные совпадали по условию совпадения значений категории 1 в имя_табл1 и ссылки на категорию в имя_табл2.)
Запрос с оператором LEFT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой по порядку (левой) таблицы, даже если они не соответствуют условию. У записей левой таблицы, которые не соответствуют условию, значение столбца из правой таблицы будет NULL (неопределённым).
Запрос с оператором RIGHT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из второй по порядку (правой) таблицы, даже если они не соответствуют условию. У записей правой таблицы, которые не соответствуют условию, значение столбца из левой таблицы будет NULL (неопределённым).
Главное отличие внешнего(левое и правое) соединения от внутреннего соединения в том, что строка из левой (для LEFT JOIN) или из правой таблицы (для RIGHT JOIN) попадет в результаты в любом случае.
Запрос с оператором FULL OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой (левой) и второй (правой) таблиц, даже если они не соответствуют условию. У записей, которые не соответствуют условию, значение столбцов из другой таблицы будет NULL (неопределённым).

7. Язык SQL. Вставка данных: оператор INSERT.

Для добавлен. данных прим.команда INSERT, которая имеет следующий формальный синтаксис:
INSERT [INTO] имя_таблицы [(список_столбцов)] VALUES (значение1, значение2, ... значениеN):

- Вставка единственной строки:
  - с указанием всех значений; INSERT INTO имя_табл VALUES(…); GO
  - с указанием некоторых значений (!); INSERT INTO имя_таблицы (значения) VALUES(…);GO
  - с указанием значений в порядке, отличающемся от порядка полей таблицы (!);
- ставка нескольких строк:
  - с явным указанием значений; INSERT INTO имя_табл VALUES(…),(…); GO
  - с использованием результатов выборки из другой таблицы.

(!): В этих случаях нужно непосредственно указать столбцы, в которые добавляются значения (значения потом должны добавляться в порядке следования столбцов).Для неуказанных столбцов будет добавляться значение по умолчанию, если задан атрибут DEFAULT, или значение NULL. При этом неуказанные столбцы должны допускать значение NULL или иметь атрибут DEFAULT. Для вставки всех значений по умолчанию можно использовать INSERT INTO имя_таблицы DEFAULT VALUES
(!!): В случае вставки в таблицу с автоинкрементным столбцом IDENTITY, то явно вставить в этот столбец свое значение можно только, если назначить свойству IDENTITY_INSERT значение ON (SET IDENTITY_INSERT [[database_name .] schema_name .] table_name { ON | OFF }).

8. Язык SQL. Обновление данных: оператор UPDATE. Условие отбора записей. Значение NULL и трехзначная логика.

Для изменения уже имеющихся строк в таблице применяется команда UPDATE. Она имеет следующий формальный синтаксис:

```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN
[FROM выборка AS псевдоним_выборки]
[WHERE условие_обновления]
```

Трехзначная логика:
В SQL используется трехзначная логика, т.е. результат логического выражения может принимать три значения – истина (True), ложь (False) и неизвестно (NULL).
Если значение в ячейке отсутствует (ячейка пуста, не заполнена), то говорят, что значение в этой ячейке есть NULL.
Следует четко понимать, что NULL – это не ноль, это вообще не значение, это отсутствие всякого значения.
NULL в Системах управления базами данных (СУБД) — специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных (БД). NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения».
Введено для того, чтобы различать в полях БД пустые (визуально не отображаемые) значения (например, строку нулевой длины) и отсутствующие значения (когда в поле не записано вообще никакого значения, даже пустого). При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равно NULL.
Если вы обновляете столбец, который был объявлен NOT NULL путем установки значения NULL, возникает ошибка, если включен строгий режим SQL; в противном случае для столбца устанавливается неявное значение по умолчанию для типа данных столбца, а количество предупреждений увеличивается. Неявное значение по умолчанию 0 для числовых типов, пустая строка ('') для строковых типов и « нулевое » значение для типов даты и времени
ISNULL
Выражение: В этом параметре мы указываем выражение, в котором нам нужно проверить значения NULL.
Замена: мы хотим заменить NULL определенным значением. Нам нужно указать значение замены здесь
Функция SQL Server ISNULL возвращает значение замены, если первое выражение параметра оценивается как NULL. SQL Server преобразует тип данных замены в тип данных выражения.

9. Язык SQL. Удаление данных: оператор DELETE. Условие отбора записей. Значение NULL и трехзначная логика.

Примеры: DELETE FROM ИМЯ_ТАБЛИЦЫ
  WHERE УСЛОВИЕ

```
GO------
----- DELETE FROM Sales.SalesPersonQuotaHistory WHERE BusinessEntityID IN
        (SELECT BusinessEntityID FROM Sales.SalesPerson WHERE SalesYTD > 2500000.00);
GO---------
----- DELETE FROM Sales.SalesPersonQuotaHistory
FROM Sales.SalesPersonQuotaHistory AS spqh
        INNER JOIN Sales.SalesPerson AS sp ON spqh.BusinessEntityID = sp.BusinessEntityID
        WHERE sp.SalesYTD > 2500000.00;
GO
```

Трехзначная логика:
В SQL используется трехзначная логика, т.е. результат логического выражения может принимать три значения – истина (True), ложь (False) и неизвестно (NULL).
Если значение в ячейке отсутствует (ячейка пуста, не заполнена), то говорят, что значение в этой ячейке есть NULL.
Следует четко понимать, что NULL – это не ноль, это вообще не значение, это отсутствие всякого значения.
NULL в Системах управления базами данных (СУБД) — специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных (БД). NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения».
Введено для того, чтобы различать в полях БД пустые (визуально не отображаемые) значения (например, строку нулевой длины) и отсутствующие значения (когда в поле не записано вообще никакого значения, даже пустого). При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равно NULL.
NULL-значение используется для идентификации подлежащих удалению строк.

Часть 5 – Создание баз данных в SQL Server 2012+

1. Системные базы данных: состав и назначение. Именование объектов базы данных. Схемы.

Системные БД:

Состав	Назначение
master	главная база данных, содержит всю системную информацию СУБД SQL Server (общие для всего экземпляра метаданные, такие как сведения об учетных записях, связанных серверах, и параметры конфигурации системы), а также информацию обо всех остальных базах данных;
Resource	доступна только для чтения, содержит все входящие в SQL Server системные объекты (такие как sys.objects), которые физически расположены в базе данных Resource, а логически отображаются для каждой базы данных в схеме sys;
msdb	используется агентом SQL Server (SQL Server Agent) для создания расписания предупреждений и заданий;
model	используется в качестве шаблона для всех баз данных, созданных для экземпляра SQL Server;
tempdb	хранит все рабочие и временные таблицы и временные хранимые процедуры, создаваемые как пользователями, так и службами SQL Server; повторно создается при каждом запуске SQL Server, временные таблицы и хранимые процедуры удаляются автоматически при отключении.

Имена объектов: [[server.][database].][schema_name].object_name

Примеры**:** server.database.schema_name.object_name
server.database..object_name
database_name.schema_name.object_name.column_name
database_name..object_name.column_name

Схемы:

Схема – это коллекция сущностей базы данных, формирующая единое пространство имен (набор, в котором у каждого элемента есть свое уникальное имя).

В SQL Server 2012 схемы существуют независимо от создавшего их пользователя базы данных. Независимость пользователей баз данных от схем предоставляет администраторам и разработчикам следующие **преимущества**:

- права владения схемами могут быть переданы без изменения имен схем;
- одной схемой могут владеть несколько пользователей – членов ролей или групп Windows; это расширяет функциональность, позволяя ролям и группам владеть объектами. Несколько пользователей могут делить одну схему по умолчанию для единого разрешения имен;
- удаление пользователей баз данных более не требует переименования объектов, содержащихся в схеме этого пользователя;
- разрешениями на схемы и содержащиеся в них объекты можно управлять с большей степенью гранулярности, чем в более ранних выпусках;
- полные имена объектов состоят их четырех компонентов: server.database.schema.object.

Создание схемы:

```
CREATE SCHEMA Schema1
    CREATE TABLE Tab1();
```

Изменение схемы:

```
ALTER SCHEMA Schema1 TRANSFER …;
```

Удаление схемы:

```
DROP TABLE Schema1.Tab1;
DROP SCHEMA Schema1;
```

2. Физическая структура базы данных: файлы и файловые группы.

Файлы данных

первичный (*.mdf)	содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных; данные и объекты пользователя могут храниться в данном файле или во вторичном файле данных; в каждой базе данных имеется один первичный файл данных;
вторичный (.ndf)	не являются обязательными; могут быть использованы для распределения данных на несколько дисков, в этом случае каждый файл записывается на отдельный диск;
журнал транзакций (.ldf)	содержат сведения, используемые для восстановления базы данных; для каждой базы данных должен существовать хотя бы один файл журнала.

Файловые группы. Файлы данных могут быть объединены в файловые группы для удобства распределения и администрирования.

первичная	первичная файловая группа, содержащая первичный файл все системные таблицы размещены в первичной файловой группе;
пользовательская	любая файловая группа, созданная пользователем при создании или изменении базы данных;
файловые группы по умолчанию	если в базе данных создаются объекты без указания файловой группы, к которой они относятся, они назначаются файловой группе по умолчанию; только одна файловая группа создается как файловая группа по умолчанию; первичная файловая группа является файловой группой по умолчанию.

3. Основные элементы синтаксиса Transact-SQL: именование объектов, идентификаторы, константы, встроенные функции, выражения.

Имена объектов имеют вид [[[server.][database.]][schema_name].]object_name
После имени объекта (object_name) может быть записано имя конкретного столбца объекта (column_name):
database_name.schema_name.object_name.column_name
Сервер, база данных и имя схемы могут быть не указаны, тогда будут использованы значения по умолчанию. (сервер - локальный сервер, база - текущая база, схема - имя пользователя в указанной базе данных, связанное с идентификатором имени входа текущего соединения)
Идентификатор - имя объекта базы данных (сервера, базы данных и ее объектов - таблиц, представлений, столбцов, индексов, триггеров, процедур, ограничений и правил, и т.п.), создаваемое при определении объекта и используемое для обращения к нему;
классы идентификаторов: обычные, расширенные (идентификаторы с разделителем)
-**Обычные** не совпадают с зарезервированными словами; не содержат пробелы и спецсимволы; 1ый символ – буква/символы _ , @ или # ; последующие символы (до 128) – буквы, цифры или символы _ , @ , # или $;
-**Расширенные** заключаются в квадратные скобки или двойные кавычки: [..], "..."; могут содержать любые символы;
Используются команды SET QUOTED_IDENTIFIER ON/OFF
Виды идентификаторов:

- имена объектов (см именования объектов)
- локальные переменные и параметры: @.....
- локальные временные таблицы и процедуры: #.....
- глобальные временные объекты (таблицы и процедуры): ##.....

константы

- числовые (bit, integer, decimal, float, real, money, uniqueidentifier, константы двоичных строк)
- строковые (заключены в одинарные кавычки; строки Unicode используют префикс N; одинарные кавычки, содержащиеся в строке, дублируются; если SET QUOTED_IDENTIFIER OFF, можно использовать двойные кавычки)
- константы даты (формат ISO 8601 - 'yyyy-mm-ddThh:mm:ss[.mmm]' ; алфавитный формат; числовой формат; строковый формат без разделителей (6- и 8-разрядные строки))

встроенные функции можно использовать и включать в следующие конструкции:
– список выбора инструкции SELECT;
– предложение WHERE инструкций SELECT, INSERT, DELETE или UPDATE для ограничения количества строк, удовлетворяющих условиям запроса;
– условие поиска в предложении WHERE представления, которое должно динамически соответствовать пользователю или среде во время выполнения;
– любое выражение;
– ограничение CHECK или триггер для поиска указанных значений при вставке данных;
– ограничение DEFAULT или триггер для вставки значений по умолчанию;
Примеры категорий функций:
– статические функции (AVG, MIN, SUM, COUNT, MAX)
– Функции для работы с курсорами (@@CURSOR_ROWS, CURSOR_STATUS, @@FETCH_STATUS)
– Математические функции (ABS, RAND, COS, EXP, ROUND)
– Функции метаданных (DB_NAME, OBJECT_NAME,)
– Строковые функции (STR, REPLACE, SUBSTRING, LEN, LOWER,)
– Системные функции (ISDATE, ISNULL, NEWID, CAST, CONVERT,…)
выражение - это сочетание идентификаторов, значений и операторов, которое SQL Server 2012 может вычислить для получения результата;
Выражение может быть: константой; функцией; именем столбца; переменной; вложенным запросом; функцией CASE, NULLIF или COALESCE; выражение также может быть построено из комбинаций этих сущностей, соединенных операторами
Выражения могут использоваться в различных контекстах во время доступа к данным или их изменения, например: как часть данных, получаемых в запросе; В качестве условий поиска данных, отвечающих определенному набору критериев;

4. Основные элементы синтаксиса Transact-SQL: управление выполнением, обработка ошибок.

Управление выполнением. язык Transact-SQL содержит специальные ключевые слова, известные как язык управления выполнением, которые управляют порядком выполнения инструкций на языке Transact-SQL, блоками инструкций и хранимыми процедурами;
Инструкции управления выполнением не могут быть распределены по разным пакетам или хранимым процедурам;
Существуют следующие **ключевые слова**, управляющие выполнением:
• **BEGIN...END, WHILE** (используются для группирования нескольких инструкций в одном логическом блоке: в цикле WHILE, в элементе функции CASE или в предложение IF или ELSE)
• **WHILE** (цикл)
• **CASE** - специальное выражение Transact-SQL, которое позволяет формировать альтернативные значения в зависимости от условия; Функция CASE состоит из:
1) ключевого слова CASE; 2) имени преобразуемого столбца; 3) предложений WHEN, которые задают выражения для поиска, и предложений THEN, которые задают выражения для замены; 4) ключевого слова END; 5) необязательного предложения AS, задающего псевдоним функции CASE
• **BREAK** (завершение цикла)
• **GOTO** (безусловный переход к метке: GOTO label ... label_name: …)
• **CONTINUE** (переход к очередной итерации)
• **IF...ELSE** (ветвление, проверка выполнения условия)
• **RETURN** (безусловное завершение запроса, хранимой процедуры или пакета: RETURN [@ReturnCode])
• **WAITFOR** (инструкция WAITFOR приостанавливает выполнение пакета, хранимой процедуры или транзакции до тех пор, пока:
1) не пройдет заданный интервал времени: DELAY(time_to_pass)
2) не наступит заданное время: TIME(time_to_execute)
3) инструкция RECEIVE не изменит или не возвратит, по крайней мере одну строку в очередь компонента Service Broker: RECEIVE / TIMEOUT)
Обработка ошибок. Ошибки, вызываемые компонентом Database Engine, можно обрабатывать на двух уровнях:
1) в компоненте Database Engine, вводя код обработки ошибок в пакеты Transact-SQL, хранимые процедуры, триггеры или в пользовательские функции.
Механизмы обработки ошибок Transact-SQL включают:
• конструкцию TRY...CATCH…END CATCH (получение информации об ошибке с помощью функций ERROR_LINE, ERROR_MESSAGE, ERROR_NUMBER, ERROR_PROCEDURE, ERROR_SEVERITY и ERROR_STATE)
• инструкции RAISERROR / THROW / PRINT (возврат приложению сообщения в формате системных ошибок или предупреждений / в формате символьной строки)
• функцию @@ERROR (получение номера ошибки, созданной предыдущей инструкцией языка Transact-SQL)
2) на уровне вызывающего приложения. Каждый из API, используемых приложением для доступа к компоненту Database Engine, наделен механизмами для возвращения сведений об ошибках соответствующему приложению.

5. Категории целостности данных: ссылочная целостность. Каскадные ограничения ссылочной целостности.

Ссылочная целостность основана на связи первичных (Primary) и внешних (Foreign key) ключей. Во всех таблицах значения этих ключей должны быть согласованы. Ссылочная целостность также обеспечивается ограничением check. Ссылочная целостность требует отсутствие ссылок на несуществующие значения, а также обеспечивает согласованное изменение ссылок по всей бд при изменении значения ключа.
Не допускаются следующие действия:

1. Добавление или изменение записей в связанной таблице, если в первичной таблице нет соответствующей записи
2. Изменение значений в первичной таблице, которое приводит к появлению потерянных записей в связанной таблице
3. Удаление записей в первичной таблице, если имеются совпадающие ключи в других таблицах

Каскадные ограничения ссылочной целостности:
С помощью каскадных ограничений ссылочной целостности можно определять действия, которые SQL Server 2012 будет предпринимать, когда пользователь попытается удалить или обновить ключ, на который указывают существующие внешние ключи.
На ключи можно добавить следующие ограничения **on delete** и **on update** при **create table** и **alter table**:

1. No action: по умолчанию. Сообщается об ошибке и производится откат операции;
2. CASCADE: каскадное изменение ссылающихся таблиц;
3. SET NULL: установка NULL для ссылающихся ключей;
4. SET DEFAULT: установка значений по умолчанию для ссылающихся ключей;

Каскадные ссылочные действия запускают триггеры AFTER UPDATE или AFTER DELETE.
Ограничения:

1. Последовательности каскадных ссылочных действий, запускаемые отдельными инструкциями DELETE или UPDATE, должны образовывать дерево без циклических ссылок;
2. никакая таблица не должна появляться больше одного раза в списке всех каскадных ссылочных действий, вызванных инструкциями DELETE или UPDATE;
3. В дереве каскадных ссылочных действий к любой из задействованных таблиц должен быть только один путь;
4. Любая ветвь в дереве прерывается, как только встречается таблица, для которой указано действие NO ACTION или вообще не указано действие

6. Категории целостности данных: сущностная, доменная и пользовательская целостность.

Сущностная целостность - определяет строку как уникальную сущность в конкретной таблице; обеспечивает целостность столбцов идентификаторов или первичного ключа таблицы с помощью: индексов; ограничений UNIQUE; ограничений PRIMARY KEY; свойств IDENTITY;

Доменная целостность - определяет достоверность записей в конкретном столбце; включает: ограничения типа данных; ограничения формата при помощи ограничений CHECK; ограничения диапазона возможных значений при помощи ограничений FOREIGN KEY, CHECK, DEFAULT, определений NOT NULL;

Пользовательская целостность - позволяет определять бизнес-правила, не входящие ни в одну из категорий целостности. Поддержку пользовательской целостности обеспечивают все остальные категории целостности: любые типы ограничений уровня столбца и уровня таблицы в: инструкции CREATE TABLE; хранимых процедурах; триггерах

7. Типы данных: атрибуты, категории, применение. Уникальные идентификаторы и особенности их использования.

С объектом, содержащим данные, связан тип, определяющий виды данных, которые могут храниться в объекте. При назначении типа данных объекту определяются четыре атрибута объекта: вид данных, содержащихся в объекте; размер или длина хранимого объектом значения; точность числа (только в случае численных типов); масштаб числа (только в случае численных типов)

Категории типов данных в SQL SERVER 2012:

битовый: bit**; целые**: smallint, int, bigint; **вещественные**: float, real; **финансовые**: money; **дата и время**: datetime, date, time; **строковые**: Unicode: nchar, nvarchar; non-Unicode: char, varchar; **двоичные**: binary; **пространственные**:geometry; **специальные**: cursor, table;

Применение: Если, оператор соединяет два выражения разных типов, то производится приведение значений левой и правой частей к типу с высшим приоритетом согласно правилам предшествования типов – по мере убывания приоритета. Если неявное приведение типов невозможно, выдается сообщение об ошибке;

Идентификаторы:

IDENTITY:

1)свойства IDENTITY позволяет разработчику указать как начальное значение идентификатора для первой строки, вставляемой в таблицу, так и шаг его увеличения;

3)уникальность свойства IDENTITY гарантирована только в рамках таблицы, в которой оно использовано;

4) таблица может содержать только один столбец со свойством IDENTITY;

5)столбец должен иметь тип данных decimal, int, numeric, smallint, bigint или tinyint;

6) столбец идентификатора не должен допускать значений и содержать определений или объектов по умолчанию;

Глобальные уникальные идентификаторы:

Если в приложении нужно сформировать столбец идентификатора, уникальный для всей базы данных или всех баз данных во всех компьютерных сетях, необходимо использовать свойство ROWGUIDCOL, тип данных uniqueidentifier и функцию NEWID;

– в таблице может содержаться только один столбец ROWGUIDCOL, который должен иметь тип данных uniqueidentifier;

–Database Engine не формирует значения для этого столбца автоматически;

– не обеспечивается уникальность значений, хранимых в столбце;

– для вставки глобального уникального значения, необходимо:

- использовать для столбца определение DEFAULT, которое использует функцию NEWID для формирования глобального уникального значения;
- использовать функцию NEWID в инструкции INSERT;

8. Представления: назначение, типы (стандартные, индексированные, секционированные).

Представление - это виртуальная таблица, состоящая из совокупности именованных столбцов и строк данных, содержимое которой определяется запросом.

• определяющий представление запрос может быть инициирован в одной или нескольких таблицах или в других представлениях текущей или других баз данных;

• пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений: строки и столбцы динамически формируются при обращении к представлению на основе данных из таблиц (и других представлений), указанных в определяющем представление запросе, и динамически создаваемых при обращениях к представлению;

Функции представлений:

– упрощение и настройка восприятия информации базы данных;

– реализация механизмов безопасности, обеспечивающих возможность обращения пользователей к данным без предоставления им разрешений на непосредственный доступ к базовым таблицам;

– обеспечение интерфейса обратной совместимости, моделирующего таблицу, которая существует, но схема которой изменилась;

– определение представлений с данными из нескольких гетерогенных источников;

Типы представлений:

• *стандартные*: сочетание данных из одной или нескольких таблиц;

• *индексированные*: вычисленное и сохраненное в базе данных представление, изменения данных в базовых таблицах, отражаются в данных, хранимых в индексированном представлении;

– индексировать представление можно, создав для него уникальный кластеризованный индекс:

• индексированные представления значительно повышают производительность некоторых типов запросов;

• индексированные представления эффективнее всего использовать в запросах, группирующих множество строк;

• индексированные представления не очень хорошо подходят для часто обновляющихся базовых наборов данных;

• *секционированные*: объединение горизонтально секционированных данных набора базовых таблиц, находящихся на одном или нескольких серверах.

Создание представлений:

CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;

9. Индексы: назначение, типы (кластеризованные и некластеризованные).

Индекс является структурой на диске, которая связана с таблицей или представлением и улучшает производительность запроса за счет сокращения количества дисковых операций ввода-вывода и уменьшения потребления системных ресурсов.

• индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении (ключи хранятся в виде структуры сбалансированного дерева);

• индексы могут быть уникальными, т.е. никакие две строки не имеют одинаковое значение для ключа индекса;

• индексы создаются автоматически при определении ограничений PRIMARY KEY или UNIQUE на основе столбцов таблицы;

Типы индексов:

• *Кластеризованный*

– кластеризованные индексы сортируют и хранят строки данных в таблицах или представлениях на основе их ключевых значений (т.е. значений столбцов, включенных в определение индекса);

– существует только один кластеризованный индекс для каждой таблицы;

– строки данных в таблице хранятся в порядке сортировки только в том случае, если таблица содержит кластеризованный индекс:

• если у таблицы есть кластеризованный индекс, то таблица называется кластеризованной;

• если у таблицы нет кластеризованного индекса, то строки данных хранятся в неупорядоченной структуре, которая называется кучей;

• *Некластеризованный*

– некластеризованные индексы имеют структуру, отдельную от строк данных;

– в некластеризованном индексе содержатся значения его ключа, и каждая запись значения ключа содержит указатель на соответствующую строку данных;

– указатель из строки индекса в некластеризованном индексе, который указывает на строку данных, называется указателем строки, структура которого зависит от того, хранятся ли страницы данных в куче или в кластеризованной таблице:

• для кучи указатель строки является указателем на строку;

• для кластеризованной таблицы указатель строки данных является ключом кластеризованного индекса;

в SQL Server 2012 можно добавить неключевые столбцы на конечный уровень некластеризованного и выполнять полностью индексированные запросы индекса и выполнять полностью индексированные запросы.

Создание индекса:

CREATE INDEX index_name ON table_name (column1, column2, ...);

CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);

10. Хранимые процедуры: назначение, типы.

Хранимые процедуры в Microsoft SQL Server аналогичны процедурам в других языках программирования: они обрабатывают входные аргументы и возвращают вызывающей процедуре или пакету значения в виде выходных аргументов, содержат программные инструкции, которые выполняют операции в базе данных, в том числе вызывающие другие процедуры и возвращают код состояния, сообщающий о успешном завершении (как main в C)

В отличие от программ T-sql, которые хранятся локально, хранимые процедуры регистрируются на сервере, могут иметь атрибуты безопасности, позволяют уменьшить сетевой трафик.

Типы хранимых процедур:

1. Пользовательские хранимые процедуры.
2. Расширенные хранимые процедуры (устарели) - представлены в виде динамических библиотек (DLL), которые могут загружаться и выполняться экземпляром Microsoft SQL Server.
3. Системные хранимые процедуры - предназначены для выполнения различных административных действий. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами. Системные хранимые процедуры имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

Создание хранимых процедур:

```
CREATE PROCEDURE procedure_name
[ { @ parameter data_type }
  [ VARYING ] [ = default ] [ OUTPUT ] ]
AS
  Sql_statement
```

До as идет перечисление входных параметров. Output - значение параметра возвращается из процедуры. = default - указать значение по умолчанию для параметра. Varying - ‘тот параметр динамически конструируется процедурой, и его содержимое может меняться; нужно, чтобы возвращать курсоры.Процедуры вызываются с помощью команды execute.

11. Курсоры: назначение, типы, обработка.

Результирующий набор - набор строк, возвращаемых инструкцией Transact-SQL (например, select). **Курсор** - это связанный с ним указатель текущей записи.

Курсоры **обеспечивают возможность:**

– позиционирования на отдельные строки результирующего набора;

– получения одной или нескольких строк от текущей позиции в результирующем наборе;

– изменения данных в строках в текущей позиции результирующего набора.

Типы курсоров:

• STATIC: результирующий набор фиксируется при открытии курсора (не отражает изменения, влияющие на вхождение в результирующий набор); в Microsoft SQL Server 2012 доступны только для чтения;

• KEYSET: курсоры управляются с помощью набора ключей, которые создаются из набора столбцов, уникально идентифицирующего строки результирующего набора;

членство и порядок строк фиксируются при открытии курсора; изменения в значениях неключевых столбцов видны при прокрутке курсора;

• DYNAMIC: отражаются все изменения строк в результирующем наборе при прокрутке курсора.

Обработка курсоров:

1. связать курсор с результирующим набором и задать его характеристики (например, возможность обновления строк);
2. выполнить инструкцию Transact-SQL для заполнения курсора;
3. получить в курсор необходимые строки;
4. при необходимости выполнить операции изменения (обновления или удаления) строки в текущей позиции курсора;
5. закрыть курсор.

12. Определяемые пользователем функции: Скалярные функции. Свойства функций и их влияние на эффективность использования функций.

Как и функции на языках программирования, SQL Server **определяемые пользователем функции** являются подпрограммами, которые принимают параметры, выполняют действие, такие как сложное вычисление, и возвращают результат этого действия в качестве значения. Возвращаемое значение может быть либо единичным скалярным значением (скалярная ф_ия), либо результирующим набором (табличная ф_ия). Скалярные функции могут использоваться везде, где допустимы выражения возвращаемого типа. Пользовательские скалярные функции возвращают одно значение типа данных, заданного в предложении RETURNS.

Преимущества: Модульное программирование, Быстрое выполнение, Уменьшение сетевого трафика.

Любая пользовательская функция состоит из двух частей:

– заголовка (содержащего имя функции с необязательным именем схемы или владельца; имя и тип данных входного параметра; тип данных возвращаемого значения и необязательное имя);

– тела (состоящего из одной или нескольких инструкций TransactSQL, реализующих логику функции; ссылки на сборку .NET)

свойства пользовательских функций определяют способность ядра СУБД индексировать результаты функции как с помощью индексов вычисляемых столбцов, вызывающих функцию, так и с помощью индексированных представлений, которые на нее ссылаются:

– детерминизм: способность возвращать один и тот же результат, если предоставлять им один и тот же набор входных значений и использовать одно и то же состояние базы данных;

– точность: отсутствие операций над числами с плавающей запятой;

– доступ к данным: осуществление доступа к локальному серверу баз данных с помощью внутрипроцессного управляемого поставщика SQL Server;

– доступ к системным данным: осуществление доступа к системным метаданным на локальном сервере баз данных с помощью внутрипроцессного управляемого поставщика SQL Server

– свойство IsSystemVerified:определяет, может ли ядро СУБД проверить детерминизм и точность функции.

Скалярные функции (могут использоваться везде, где допустимы выражения возвращаемого типа;)

13. Определяемые пользователем функции: Табличные функции. Сравнение функций и хранимых процедур.

Как и функции на языках программирования, SQL Server определяемые пользователем функции являются подпрограммами, которые принимают параметры, выполняют действие, такие как сложное вычисление, и возвращают результат этого действия в качестве значения. Возвращаемое значение может быть либо единичным скалярным значением (скалярная ф-ия), либо результирующим набором **(табличная ф-ия).**

Табличные функции (могут использоваться везде, где допустимы табличные выражения (в том числе, как альтернатива представлений и хранимых процедур, возвращающих один результирующий набор).) Определяемые пользователем табличные функции (TVFs) возвращают табличный тип данных. **Отличия функций от хранимых процедур:**

Функция	Хранимая процедура
Должна возвращать значение.	Может как возвращать, так и не возвращать значение.
Не могут возвращать несколько результирующих наборов.	Может сформировать и вернуть несколько результирующих наборов данных.
Можно использовать в операторе SELECT.	Нельзя использовать в операторе SELECT и во всех его секциях (WHERE, JOIN, HAVING и т.д.), так как процедуры вызываются с помощью команды EXECUTE или EXEC.
В функциях можно использовать только оператор SELECT на выборку данных. Операторы DML (INSERT, UPDATE, DELETE) для модификации данных использовать нельзя.	В хранимых процедурах можно использовать оператор SELECT, а также операторы DML (INSERT, UPDATE, DELETE) для модификации данных.
Из функции нельзя вызвать хранимые процедуры.	В хранимых процедурах можно вызывать и функции, и другие хранимые процедуры.
Конструкцию для обработки ошибок TRY CATCH нельзя использовать в функциях. Так же как нельзя использовать инструкцию RAISERROR.	В хранимых процедурах можно использовать и конструкцию TRY CATCH, и инструкцию RAISERROR.
В функциях запрещено использование транзакций.	В хранимых процедурах транзакции разрешены.
В функциях можно использовать только табличные переменные, временные таблицы использовать не получится.	В хранимых процедурах можно использовать как табличные переменные, так и временные таблицы.
В функциях нельзя использовать динамический SQL.	В процедурах можно использовать динамический SQL.
В функциях можно использовать только входные параметры.	В хранимых процедурах можно использовать как входные, так и выходные параметры.

14. Триггеры: назначение, типы. Триггеры DDL.

триггером называют хранимую процедуру особого типа, которая автоматически выполняется при возникновении языкового события; Триггеры являются одним из двух механизмов реализации бизнес-правил и целостности данных (наряду с ограничениями), поддерживаемых SQL Server 2012;

типы триггеров:

– DML-триггер: действие, которое выполняется при наступлении события языка DML

– DDL-триггер: действие, которое выполняется при возникновении событий языка определения данных (DDL) на сервере баз данных

Триггеры DDL активируются в ответ на различные события языка DDL. Эти события в основном соответствуют инструкциям Transact-SQL, которые начинаются с ключевых слов CREATE, ALTER, DROP, GRANT, DENY, REVOKE или UPDATE STATISTICS. Системные хранимые процедуры, выполняющие операции, подобные операциям DDL, также могут запускать триггеры DDL.

Нужны для того, чтобы: Предотвращать внесение определенных изменений в схему базы данных. Настроить выполнение в базе данных некоторых действий в ответ на изменения в схеме базы данных. Записывать изменения или события схемы базы данных.

- триггеры DDL срабатывают только после выполнения соответствующих инструкций DDL;
- триггер DDL и инструкция, приводящая к его срабатыванию, выполняются в одной транзакции;
- для одной инструкции Transact-SQL можно создать несколько триггеров DDL.

15. Триггеры DML: триггеры типа INSTEAD OF.

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 можно использовать для принудительного применения бизнес-правил и целостности данных, запроса других таблиц 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.

Instead of: Используются в таблицах и представлениях (с одной или более базовыми таблицами). Выполняются вместо обычных действий, запускающих триггеры. Выполняются до проверки ограничений. Единственный триггер на каждое из запускающих триггеры действие (UPDATE, DELETE или INSERT). Не допускаются в таблицах, в которых используется каскадное обновление (для UPDATE и DELETE). Могут работать со столбцами типа text, ntext и image.

16. Триггеры DML: триггеры типа AFTER.

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 можно использовать для принудительного применения бизнес-правил и целостности данных, запроса других таблиц 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.

17. Триггеры DML: сравнение триггеров типа INSTEAD OF и AFTER. Дополнительные средства управления данными, доступные в триггерах.

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 можно использовать для принудительного применения бизнес-правил и целостности данных, запроса других таблиц 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.

Функция	**Триггер AFTER**	**Триггер INSTEAD OF**
Сущности, к которым применяется триггер	Таблицы	Таблицы и представления
Количество триггеров на таблицу или представление	Несколько триггеров на одно запускающее триггеры действие (UPDATE, DELETE или INSERT).	Один триггер на одно запускающее триггеры действие (UPDATE, DELETE или INSERT).
Каскадные ссылки	Нет ограничений.	Триггеры INSTEAD OF UPDATE и DELETE нельзя определять для таблиц, на которые распространяются каскадные ограничения ссылочной целостности.
Выполнение	После следующих операций: -Обработка ограничений. -Декларативные ссылочные действия. -Создание таблиц inserted и deleted . -Действие, запускающее триггер.	До: обработка ограничений Вместо: действие активации После: вставка и удаление таблиц
Порядок выполнения	Можно задать выполнение в первую и в последнюю очередь.	Неприменимо
Ссылки на столбцы varchar(max), nvarchar(max)и varbinary(max) в таблицах inserted и deleted	Разрешено	Разрешено
Ссылки на столбцы text, ntext и image в таблицах inserted и deleted	Нельзя использовать	Разрешено

Доп средства управления данными

• таблицы inserted и deleted (соответственно, вставленных и удаленных значений), формат таблиц соответствует формату таблицы или представления, для которых задан триггер;

• функция COLUMNS_UPDATED() возвращает битовую маску обновленных столбцов;

• функция UPDATE(column_name) возвращает TRUE, если столбец был обновлен.

Часть 6 - Управление параллельной обработкой в базах данных

1. Управление параллельной обработкой в БД. Транзакции. Блокировка и взаимная блокировка.

Управление параллельной обработкой в БД:

- управление параллельной обработкой (concurrency control) направлено на то, чтобы исключить непредусмотренное влияние действий одного пользователя на действия другого;
- управление параллельной обработкой, как правило, предполагает компромисс между уровнем изоляции различных операций друг от друга и производительностью системы;
- транзакция (transaction) является последовательностью операций, выполненных как одна логическая единица работы (logical unit of work), т.е. либо все действия внутри транзакции выполняются успешно, либо не выполняется ни одно из них:
  - BEGIN TRANSACTION;
  - COMMIT | ROLLBACK TRANSACTION.

Параллельная обработка транзакций:

- параллельные транзакции (parallel transactions) – две или более транзакций, обрабатываемые одновременно;
- проблемы параллельной обработки:
  - грязное / несогласованное чтение (dirty/inconsistent reads);
  - невоспроизводимое / фантомное чтение (unrepeatable / phantom reads);
  - потерянное обновление (lost / concurrent update);
- необходимо упорядочить операции внутри транзакций таким образом, чтобы предотвратить нежелательное влияние одной транзакции на другую

Блокировка:

Инициатор блокировки:

- СУБД: неявная блокировка (implicit lock);
- программа / запрос: явная блокировка (explicit lock).

Степень (глубина) детализации блокировки (lock granularity):

- на уровне базы данных;
- на уровне таблицы;
- на уровне страницы;
- на уровне строки.

Режим (тип) блокировки:

- монопольный (exclusive lock) : блокируются все виды доступа к элементу;
- разделяемый (shared lock) : блокируется только запись.

Двухфазная блокировка:

один из способов достижения сериализуемости – использование двухфазной блокировки (two-phase-locking): 1) фаза нарастания (growing phase): транзакция может налагать блокировки по мере необходимости, до тех пор, пока не снимается хотя бы одна блокировка; 2) фаза сжатия (shrinking phase): после снятия одной из блокировок никакие другие накладываться уже не могут;

Модификация двухфазной блокировки: снятие блокировок непосредственно в момент фиксации / отката транзакции (предотвращает каскадные откаты).

Взаимная блокировка (deadlock, deadly embrace):

- предотвращение взаимной блокировки:
  - блокирование всех ресурсов одновременно в начале операции;
  - требование блокировки в одинаковом порядке;
- обнаружение и устранение взаимной блокировки: откат одной из транзакций, вызвавших блокировку (deadlock breaking).

2. Транзакции: свойства, уровни изоляции. Восстановление баз данных.

Свойства транзакций

–атомарность(atomicity): транзакция должна быть атомарной единицей работы; должны быть выполнены либо все входящие в нее модификации данных, либо ни одно из них не должно быть выполнено;

–согласованность(consistency): по завершении транзакция должна оставить все данные в согласованном состоянии (включая все внутренние структуры данных), для этого модификации должны применяться к данным в том состоянии, в котором они были на момент начала операции: на уровне оператора, на уровне транзакции

–изоляция(isolation): выполняемые транзакцией модификации должны быть(могут быть)изолированы от любых модификаций, проводимых другими транзакциями;

–устойчивость(durability): после завершения транзакции произведенные ею действия фиксируются в системе (даже в случае системного сбоя).

проблемы изоляции транзакций:

–«грязное» чтение(dirty read)–чтение транзакцией записи, измененной другой транзакцией, при этом эти изменения еще не зафиксированы;

–невоспроизводимое чтение(non-repeatable read)–при повторном чтении транзакция обнаруживает измененные или удаленные данные, зафиксированные другой транзакцией;

–фантомное чтение(phantom read)–при повторном чтении транзакция обнаруживает новые строки, вставленные другой завершенной транзакцией;

Уровни изоляций

–незавершенное чтение (read uncommited);

–завершенное чтение (read commited);

–воспроизводимое чтение (repeatable read);

–сериализуемость (serializable);

Восстановление баз данных – функция СУБД, которая в случае логических и физических сбоев приводит базу данных в актуальное и консистентное состояние

•**повторная отработка** (reprocessing):

–периодические снимки базы данных и записи о всех операциях со времени последнего сохранения;

–нереализуемый подход в силу отсутствия времени и асинхронного характера первоначальных изменений;

•**откат-накат**(rollback-rollforward):

–периодические снимки базы данных и журнал транзакций со времени последнего сохранения, содержащий все изменения в базе данных в хронологическом порядке;

–rollforward: восстановление базы данных и выполнение всех корректных транзакций;

–rollback: отмена изменений, вызванных некорректными или незавершенными транзакциями, и выполнение всех корректных транзакций, выполнявшихся в момент сбоя;

–для отмены транзакции журнал должен содержать копию записей базы данных до их изменения (исходные образы, before-images);

–для повторного выполнения транзакции журнал должен содержать копию записей базы данных после их изменения (конечные образы, after-images).

3. Управление параллельной обработкой в MS SQL Server.

осуществляется с помощью:

• определение уровня изоляции транзакции:

SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE

• определение характеристик курсора:

DECLARE cursor_name CURSOR
FORWARD_ONLY | SCROLL – возможность прокрутки курсора;
READ_ONLY | SCROLL_LOCKS | OPTIMISTIC (WITH VALUES / WITH ROW VERSIONING) – управление изменениями и блокировками записей;
STATIC | KEYSET | DYNAMIC | FAST_FORWARD – тип курсора;

• блокировочные подсказки инструкций (SELECT, INSERT, UPDATE, DELETE):

SELECT … FROM … WITH()
NOLOCK, READUNCOMMITTED, READCOMMITTED, READCOMMITTEDLOCK, REPEATABLEREAD,HOLDLOCK, PAGLOCK | ROWLOCK | TABLOCK

4. Транзакции в SQL Server. Режимы транзакций. Распределенные транзакции.

Режимы транзакций

1**. явные** транзакции:
запускаются посредством инструкции BEGIN TRANSACTION; завершение транзакции осуществляется с использованием инструкций COMMIT TRANSACTION / WORK (фиксация) или ROLLBACK TRANSACTION / WORK (откат); управление транзакциями также может осуществляться через функции API – ADO, ODBC, OLEDB (ITransactionLocal::StartTransaction(), ITransaction::Commit() и ITransaction::Abort());

2. **автоматически фиксируемые транзакции** (режим по умолчанию): каждая отдельная инструкция фиксируется после завершения;

3**. неявные транзакции**: режим запускается через инструкцию SET IMPLICIT_TRANSACTIONS ON или через соответствующие функции API (ODBC, OLEDB); очередная инструкция DDL, DML, DCL автоматически запускает новую транзакцию; необходимо только фиксировать или выполнять откат каждой транзакции (COMMIT / ROLLBACK), после завершения транзакции следующая инструкция (из перечисленного списка) запускает новую транзакцию.

Распределенные транзакции

Участники распределенной транзакции:
- два или более сервера, которые называются диспетчерами ресурсов;
- диспетчер транзакций, который обеспечивает управление транзакцией путем координации между диспетчерами ресурсов.

Двухфазовая фиксация транзакций:
- фаза подготовки: каждый из диспетчеров ресурсов обеспечивает устойчивость транзакции, в частности, записывая журнал транзакции на диск;
- фаза фиксации: начинается в случае успешного завершения подготовки всеми диспетчерами ресурсов и завершается после получения диспетчером транзакций подтверждений об успешной фиксации от каждого из диспетчеров ресурсов.

Управление распределенными транзакциями может осуществляться через вызов инструкций Transact-SQL, а также через вызов функций API OLE DB и ODBC.

5. Восстановление баз данных. Восстановление в MS SQL Server.

Восстановление базы данных — функция СУБД, которая в случае логических и физических сбоев приводит базу данных в актуальное и консистентное состояние.

Повторная отработка (reprocessing):
- периодические снимки базы данных и записи о всех операциях со времени последнего сохранения;
- нереализуемый подход в силу отсутствия времени и асинхронного характера первоначальных изменений.

Откат-накат (rollback-rollforward):
- периодические снимки базы данных и журнал транзакций со времени последнего сохранения, содержащий все изменения в базе данных в хронологическом порядке;
- rollforward: восстановление базы данных и выполнение всех корректных транзакций;
- rollback: отмена изменений, вызванных некорректными или незавершенными транзакциями, и выполнение всех корректных транзакций, выполнявшихся в момент сбоя;
- для отмены транзакции журнал должен содержать копию записей базы данных до их изменения (исходные образы, before-images);
- для повторного выполнения транзакции журнал должен содержать копию записей базы данных после их изменения (конечные образы, after-images).

В MS SQL Server:

Создание резервной копии:

1. Full backup - делает копию всей базы данных, включая все объекты и данные системных таблиц.
2. Differential (частичный) backup - это копирование только тех данных, которые появились с момента последней полной резервной копии.

Модели восстановления:

1. Простая (simple recovery model) - автоматически урезает журналы транзакций, освобождая место на диске. Вручную журналы транзакций обслуживать не нужно. В случае аварии, данные могут быть восстановлены только на момент снятия резервной копии.
2. Полная (full recovery model) - Полная модель восстановления хранит все транзакции в журнале транзакций до усечения журнала (посредством снятия резервной копии журнала). Это самая “надежная” модель восстановления, при аварийном сбое можно вы сможете восстановить все транзакции, кроме тех, которые не успели завершиться при аварии.
3. Частичная (bulk-logged recovery model)

Часть 7 – ADO.NET

1. ADO.NET: режимы работы, объектная модель, базовые интерфейсы и базовые классы. Поставщики данных.

Режимы работы: режим с поддержанием соединения(связный уровень):
- используются объекты Connection, Command, DataReader, через которые устанавливается подключение к источнику данных и выполняется с ним взаимодействие.

Режим без поддержания подключения(несвязный уровень):
- операции производятся над локальной копией данных источника (объект - DataSet) и связанными объектами (DataTable etc.)
- объект DataAdapter обеспечивает взаимодействие приложения и источника данных, при этом операции открытия и закрытия соединения выполняются автоматически по мере необходимости (операции с объектами несвязного уровня можно выполнять вообще без подключения к источнику данных).

Объектная модель:

Connection -> Transaction, Command
DataAdapter -> Command
Command -> Parameter, DataReader
DataSet -> DataTable, DataView, DataRelation
DataTable -> DataRow, DataColumn, Constraint
DataView -> DataRowView

Базовые интерфейсы и классы поставщиков данных:

Connection (класс - DbConnection; интерфейс - IDbConnection) - обеспечивает подключение к источнику данных
Command (DbCommand; IDbCommand) - представляет запрос к источнику данных
DataReader (DbDataReader; IDataReader, IDataRecord) - считывает полученные в результате запроса данные.
DataAdapter (DbDataAdapter; IDataAdapter, IDbDataAdapter) - обеспечивает обмен данными между источниками данных объектами набора данных(DataSet)
Parameter (DbDataParameter ; IDataParameter , IDbDataParameter) - представляет именованных параметр в параметризованном запросе
Transaction (DbTransaction; IDbTransaction) - инкапсулирует транзакцию

Поставщики данных(data providers) - это набор классов ADO.NET, которые позволяют получать доступ к определенной БД, выполнять команды SQL и извлекать данные.

Конкретные имена основных классов различаются у различных поставщиков (например, SqlConnection, OracleConnection, OdbcConnection и MySqlConnection), но все эти объекты порождены от одного и того же базового класса (в случае объектов подключения это DbConnection), который реализует идентичные интерфейсы (вроде IDbConnection).

2. Связный уровень ADO.NET. Объекты связного уровня и общий сценарий работы.

Используются объекты Connection, Command, DataReader, через которые устанавливается подключение к источнику данных и выполняется с ним взаимодействие.

Connection (класс - DbConnection; интерфейс - IDbConnection) - обеспечивает подключение к источнику данных
Command (DbCommand; IDbCommand) - представляет запрос к источнику данных
DataReader (DbDataReader; IDataReader, IDataRecord) - считывает полученные в результате запроса данные.

Общий сценарий работы:

1. создание, настройка и открытие объекта соединения
2. создание и настройка объекта запроса, в том числе:
    a. определение объекта соединения для объекта запроса
    b. определение строки запроса
3. вызов метода ExecuteReader (ExecuteNonQuery, ExecuteScalar)
4. прокрутка результатов выполнения запроса

3. Связный уровень ADO.NET. Общий сценарий работы. Пул подключений.

Используются объекты Connection, Command, DataReader, через которые устанавливается подключение к источнику данных и выполняется с ним взаимодействие.

Общий сценарий работы:

5. создание, настройка и открытие объекта соединения
6. создание и настройка объекта запроса, в том числе:
    a. определение объекта соединения для объекта запроса
    b. определение строки запроса
7. вызов метода ExecuteReader (ExecuteNonQuery, ExecuteScalar)
8. прокрутка результатов выполнения запроса

Пул соединений:
- позволяет улучшить производительность за счет повторного использования установленного физического соединения с источником
- поддерживается поставщиками подключений ADO.NET и по умолчанию используется (Pooling = true)
- соединение в пуле, которое не использовалось, уничтожается по истечении таймаута;
- возможна принудительная очистка пулов (вызов функций ClearPool() и ClearAllPools())
- мин/макс размер пула определяется параметром MinPoolSize/MaxPoolSize
- поставщик поддерживает минимальное число соединений в пуле независимо от того, являются ли они активными;
- если достигнут максимальный размер пула и предпринимается попытка открытия нового соединения: производится ожидание освобождения соединения пула в течении таймаута; по истечении таймаута возникает исключение InvalidOperationException;
- при закрытии соединения и уничтожении объекта соединения внутреннее соединение с СУБД не разрывается, а помещается в пул для дальнейшего использования

4. Связный уровень ADO.NET. Поставщики данных. Создание программного кода, не зависящего от поставщика.

Используются объекты Connection, Command, DataReader, через которые устанавливается подключение к источнику данных и выполняется с ним взаимодействие.

Поставщики данных(data providers) - это набор классов ADO.NET, которые позволяют получать доступ к определенной БД, выполнять команды SQL и извлекать данные.

Конкретные имена основных классов различаются у различных поставщиков (например, SqlConnection, OracleConnection, OdbcConnection и MySqlConnection), но все эти объекты порождены от одного и того же базового класса (в случае объектов подключения это DbConnection), который реализует идентичные интерфейсы (вроде IDbConnection).

Для повышения гибкости приложений ADO.NET, которые **не зависят от поставщика**, можно использовать файл конфигурации app.config, элементы которого будут содержать произвольные пары ключ-значение. Чтобы программно получить соответствующий поставщик данных, создается фабрика объектов подключений, позволяющая изменить поставщик без необходимости перекомпиляции кодовой базы. Нужно лишь изменить файл *.config.

5. Связный уровень ADO.NET. Параметризованные запросы.

Используются объекты Connection, Command, DataReader, через которые устанавливается подключение к источнику данных и выполняется с ним взаимодействие.

Для определения параметров используется объект SqlParameter и его конструкторы. Для конфигурации параметров можно использовать их свойства. Например: SqlDbType задает тип параметра, Direction хранит направление параметра (это входной параметр или выходной, или оба варианта, или он возвращает данные из хранимой процедуры), IsNullable указывает допускается ли значение null. После определения, параметры добавляются в коллекцию Parameters объекта SqlCommand методом Add.

Пример

```
using(SqlCommand cmd = new SqlCommand())
{
        SqlParameter param = new SqlParameter();
        param.ParameterName = "@CarID";
        param.Value = id;
        param.SqlDbType = SqlDbType.Int;
        cmd.Parameters.Add(param);
        cmd.ExecuteNonQuery();
}
```

6. Несвязный уровень ADO.NET. Объекты несвязного уровня и общий сценарий работы.

Режим без поддержания подключения(несвязный уровень):

- операции производятся над локальной копией данных источника (объект DataSet) и связанными объектами (DataTable etc.)
- объект DataAdapter обеспечивает взаимодействие приложения и источника данных, при этом операции открытия и закрытия соединения выполняются автоматически по мере необходимости (операции с объектами несвязного уровня можно выполнять вообще без подключения к источнику данных).

DataAdapter:

- автономно управляет подключением к источнику данных;
- получает результаты запроса и записывает их в DataSet;
- переносит в БД изменения, внесенные в DataSet;
- устанавливает соответствие между результатами запроса и таблицами /столбцами в DataSet.

DataSet:

- содержит данные в виде набора таблиц с возможными связями между ними;
- является источником данных для элементов управления интерфейса пользователя;
- хранит локальную копию исходных данных и внесенных изменений;
- позволяет преобразовывать данные в XML документ и обратно, а также в двоичный формат;

DataTable

- содержит данные в виде набора записей, состоящих из одинаковых наборов полей;
- каждая запись хранится как в исходном варианте, так и в текущем – с учетом внесенных изменений.

DataColumn

- определяет столбец в таблице и его свойства: тип, значение по умолчанию, может ли содержать NULL, может ли значение быть изменено;
-задает выражение для вычисляемых столбцов.

DataRow

- соответствует одной записи в таблице;
- может быть в одном из состояний: Unchanged; Detached; Added; Modified;Deleted;
- содержит следующие версии значений полей: Original; Current; Proposed; Default.

DataColumn

- определяет столбец в таблице, его свойства: тип, значение по умолчанию, isNullable, может ли быть изменен
- задает выражение для вычисляемых столбцов

DataRelation

- устанавливает связь 1:1, 1:N, M:N между 2 таблицами
- по умолчанию для родительских столбцов добавляет ограничение Unique Constraint, для дочерних – ForeignKey Constraint

Позволяет определять дочерние и родительские записи для других записей.

DataView

- отбирает записи удовлетворящие заданному условию
- соритрует записи по значению указанных столбцов
- может основываться на 1 таблице

Constraint

- PK, Unique, FK

Сценарий работы: DataAdapter, Command, DataReader, Fill, Update

7. Несвязный уровень ADO.NET. Настройка запросов для синхронизации с источником данных.

Режим без поддержания подключения(несвязный уровень):

- операции производятся над локальной копией данных источника (объект - DataSet) и связанными объектами (DataTable etc.)
- объект DataAdapter обеспечивает взаимодействие приложения и источника данных, при этом операции открытия и закрытия соединения выполняются автоматически по мере необходимости (операции с объектами несвязного уровня можно выполнять вообще без подключения к источнику данных).

Настройка для синхронизации:

-sqlConnection
- sqldataAdapter
- SqlCommand
- sqlCommandBuilder – объект для автоматического создания инструкция sql для обновления таблиц из DataAdapter.
- Fill – заполнение
- Update - обновление объектов

Часть 8 - Создание распределенных баз данных

1. Методы распределения данных: горизонтальная фрагментация.

Горизонтальная фрагментация выполняется операцией селекции, распределяющей кортежи в соответствии с предикатом селекции (CHECK в столбцах ID); таблица разбивается на несколько частей так, чтобы каждая часть содержала записи со значениями только из некоторого диапазона;

Пример: на каждом сервере:

```
CREATE TABLE Table (
        ID INTEGER PRIMARY KEY CHECK (ID < 1000),
        ColumnA INTEGER,
        ColumnB INTEGER)
```

на сервере, содержащем представление:

```
USE master
EXEC sp_serveroption OtherServer', 'lazy schema validation', 'true'
CREATE VIEW View AS
SELECT * FROM MyDB.dbo.Table
UNION ALL
SELECT * FROM OtherServer.MyDB.dbo.Table
```

2. Методы распределения данных: вертикальная фрагментация.

Вертикальная фрагментация реализуется с применением операции проекции, необходимы дополнительные соединения для достижения значений нерезидентных атрибутов; таблица разбивается на части так, чтобы каждая часть содержала только некоторые из столбцов исходной таблицы (разбиение строк)

Пример: Создание вертикальной фрагментации:

• на первом сервере:

```
CREATE TABLE Table (
ID INTEGER PRIMARY KEY, ColumnA INTEGER)
```

• на втором сервере:

```
CREATE TABLE Table (
ID INTEGER PRIMARY KEY, ColumnB INTEGER)
```

• на сервере, содержащем представление:

```
CREATE VIEW View AS
SELECT one.ID, one.ColumnA, two.ColumnB
FROM MyDB.dbo.Table one, OtherServer.MyDB.dbo.Table two
WHERE one.ID = two.ID
```

3. Методы распределения данных: тиражирование. Связанные таблицы. Обновление данных.

Тиражирование данных (Data Replication - DR) - это асинхронный перенос изменений объектов исходной базы данных (source database) в БД, принадлежащим различным узлам распределенной системы. Функции DR выполняет специальный модуль СУБД - сервер тиражирования данных, называемый репликатором (replicator). Его задача - поддержка идентичности данных в принимающих базах данных (target database) данным в исходной БД. Сигналом для запуска репликатора служит срабатывание некоторого правила. Принципиальная характеристика тиражирования данных заключается в отказе от физического распределения данных (любая база данных для СУБД и для работающих с ней пользователей всегда является локальной)

- на каждом сервере хранится копия исходной таблицы
- тиражирование целесообразно, если менее 5% инструкций, ссылающихся на таблицу, инструкции INSERT, UPDATE или DELETE.

В зависимости от требований к целостности можно использовать два подхода к обновлению данных:

1) Если требуется высокая целостность транзакций, можно создать триггеры, выполняющие распределенное обновление всех копий контексте распределенной транзакции;
2) Если высокая целостность не требуется, можно для переноса изменений от одной копии таблицы на все остальные воспользоваться механизмами репликации SQL Server;

Механизмы репликации:

1) Репликация моментальных снимков (snapshot replication) - распространяет данные целиком и точно в том виде, в котором они были представлены в момент синхронизации, и не контролирует обновления этих данных;
2) Транзакционная репликация (transactional replication) - начинается создания исходного моментального снимка объектов и данных базы данных публикации, последующие изменения данных и схемы на издателе обычно доставляются без задержек (практически в реальном времени), а изменения данных применяются на подписчике в том же порядке и в тех же рамках транзакций, в которых они выполнялись у издателя (в пределах публикации гарантируется согласованность транзакций).

4. Федеративные серверы баз данных. Симметричное и асимметричное секционирование.

Федеративные серверы баз данных - механизм доступа и управления разнородными данными на серверах, управляющихся независимо, но взаимодействующих во время обработки запросов к базе данных

- цель: обеспечение высокого уровня доступности путем создания федерации серверов баз данных (горизонтального секционирования данных базы), что позволит распределить нагрузку на группу серверов, управляющихся независимо, но взаимодействующих во время обработки запросов к базе данных;
- процесс построения федерации серверов баз данных включает разработку набора распределенных секционированных представлений, используемых для распределения данных по серверам;
- секционирование дает хорошие результаты, если строение таблиц в базе данных позволяет поделить их на одинаковые секции так, чтобы большинство строк, к которым обращается любая из инструкций SQL, были бы размещены на одном федеративном сервере.

Симметричные секции

- наиболее эффективно секционирование работает в случае, если все таблицы в базе данных могут быть секционированы симметрично (то есть равномерно по федеративным серверам), при этом:
  - взаимосвязанные данные размещаются на одном федеративном сервере так, чтобы большинство инструкций SQL минимально нуждалось в данных на других серверах;
  - большинство инструкций SQL должно обращаться к серверу с 80% требуемых данных, а на распределенные запросы должно приходиться менее 20% данных;
- пример: секционирование данных по регионам
  - большинство запросов нуждается в данных с сервера одного региона;
  - приложения направляют инструкции SQL на федеративный сервер с данными того региона, который был определен из контекста входных данных.

Асимметричные секции

- исп-ся в приложениях со сложными схемами доступа к данным, не позволяющими осуществить симметричное секционирование;
- некоторым федеративным серверам приходится назначать более масштабные роли, чем другим. В базе данных могут секционироваться некоторые таблицы (остальные таблицы приходится оставлять на первоначальном сервере), что улучшает производительность
- можно добиться производительности, как и при симметричном секционировании. При этом имеется ряд преимуществ:
  - можно значительно улучшить производительность баз данных, которые не могут быть секционированы симметрично;
  - большую существующую систему можно успешно секционировать посредством серии последовательных, асимметричных процедур секционирования;

5. Проектирование распределенных секционированных представлений.

Федеративные серверы баз данных - механизм доступа и управления разнородными данными на серверах, управляющихся независимо, но взаимодействующих во время обработки запросов к базе данных.
Создаются с целью распределить нагрузку на группу серверов.
Секционирование дает хорошие результаты, если строение таблиц в базе данных позволяет поделить их на одинаковые секции так, чтобы большинство строк, к которым обращается любая из инструкций SQL, были бы размещены на одном федеративном сервере

При **проектировании** набора распределенных секционированных представлений для реализации федерации серверов баз данных, необходимо выполнить следующие процедуры:

- определить шаблон инструкций SQL, выполняемых приложением;
- определить, каким образом таблицы связаны друг с другом;
- сопоставить частоту инструкций SQL с секциями, определенными при анализе внешних ключей;
- определить правила маршрутизации инструкций SQL.

6. Создание распределенных секционированных представлений в СУБД SQL Server.

Основные требования к представлениям для распределенного горизонтального секционирования:

- запросы должны быть соединены с помощью UNION ALL;
- базовые таблицы не должны входить в запрос более одного раза;
- столбцы должны быть одного типа, идти в одном и том же порядке и встречаться только один раз;
- столбец, определяющий сегментирование:
  - не должен быть вычисляемым;
  - должен входить в первичный ключ;
  - должен иметь единственное ограничение CHECK с операторами [BETWEEN, AND, OR, <, <=, >, >=, =] , определяющее набор непересекающихся диапазонов значений.

пример:

-- On server1: CREATE TABLE USER(ID INT PRIMARY KEY CHECK (ID BETWEEN 1 AND 1000), …)	-- On server2: CREATE TABLE USER(ID INT PRIMARY KEY CHECK (ID BETWEEN 1001 AND 2000), …)	-- On server3: CREATE TABLE USER(ID INT PRIMARY KEY CHECK (ID > 2000), …)

-- On each server:

```
CREATE VIEW userView AS
SELECT * FROM server1.db.dbo.USER
UNION ALL
SELECT * FROM server2.db.dbo.USER
UNION ALL
SELECT * FROM server3.db.dbo.USER
```

7. Связанные серверы. Нерегламентированные имена (OPENDATASOURCE, OPENROWSET). Сквозные запросы (OPENQUERY).

Связанным называется сервер, зарегистрированный на SQL Server 2012 со всеми сведениями, необходимыми для доступа к источнику данных OLE DB, включая данные о:

- поставщике OLE DB - динамической библиотеке, осуществляющей управление и взаимодействие с определенными источниками данных;
- источнике данных OLE DB - конкретной базе данных, доступ к которой выполняется через интерфейс OLE DB;

Доступ к связанному серверу может осуществляться следующими способами:

- через распределенные запросы, обращающиеся к таблицам на связанном сервере при помощи инструкций SELECT, INSERT, UPDATE и DELETE с именем, относящимся к такому серверу;
- через удаленные хранимые процедуры, выполняющие обращения к связанному серверу по имени, состоящему из четырех частей;
- через инструкцию EXECUTE, реализуемую произвольной параметризованной передаваемой командой при помощи расширения AT linked_server_name.

в SQL Server макрокоманды OPENROWSET и OPENDATASOURCE предоставляют сведения для подключения, выполняемого с целью доступа к данным из источников данных OLE DB:

функция OPENROWSET:

- может использоваться с любым поставщиком OLE DB, возвращающим набор строк;
- может применяться везде, где в инструкции языка Transact-SQL используется ссылка на таблицу или представление;
- требует задания следующих данных:
  - всех сведений, необходимых для подключения к источнику данных OLE DB;
  - имени объекта или запроса, которые должны формировать набор строк.

функция OPENDATASOURCE:

- применяется только в тех случаях, если поставщик предоставляет наборы строк и использует нотацию каталог.схема.объект;
- может использоваться там, где синтаксис Transact-SQL позволяет указать имя связанного сервера;
- может использоваться в нотации каталог.схема.объект в качестве ссылки на таблицу или представление;
- требует задания следующих данных:
  - зарегистрированного как PROGID имени поставщика OLE DB, используемого для доступа к источнику данных;
  - строки соединения, задающей различные свойства соединения, передаваемые поставщику OLE DB (последовательность пар "ключевое слово-значение").

Сквозные запросы: функции OPENQUERY позволяют выполнить передаваемый запрос к
указанному связанному серверу:

- на функцию OPENQUERY как на имя таблицы можно ссылаться из предложения FROM, в инструкциях INSERT, UPDATE или DELETE;
- в качестве аргументов нельзя использовать переменные;
- функция OPENQUERY не может быть использована для выполнения расширенных хранимых процедур на связанном сервере.

8. Распределенные запросы. Доступ к удаленным данным: поставщики OLE DB. Ограничения распределенных запросов.

Распределенные запросы используются для доступа к данным из нескольких источников данных, которые могут размещаться на одном или различных компьютерах.

Microsoft SQL Server поддерживает распределенные запросы к следующим типам источников данных:

1) Распределенные данные, хранящиеся в нескольких экземплярах SQL Server
2) Удаленные данные, доступ к которым предоставляется с использованием поставщика OLE DB

OLE DB - это низкоуровневый интерфейс для доступа к данным. Он определяет набор COM-интерфейсов для предоставления приложениям единообразного доступа к данным, хранящимся в различных источниках информации (данные предоставляются в табличных объектах, именуемых «наборами строк»).

(Ограничения?)При выработке стратегий распределенного выполнения запросов следует:

– Адекватно оценивать интенсивность обмена данными и время доступа к удаленным узлам системы;

– Снизить время вычислений и операций ввода/вывода в запросах;

– По возможности осуществлять параллельную обработку данных на узлах.

Иначе распределенный запрос может выполняться очень медленно.

(Или)Распределенные ограничения целостности данных:

- БД разделена на фрагменты таким образом, что родительская таблица находится на одном узле, а дочерняя, связанная с ней по внешнему ключу, – на другом; при добавлении записи в дочернюю таблицу система обратится к узлу, на котором расположена родительская таблица, для проверки наличия соответствующего значения ключа;
- Разбиение одной таблицы на фрагменты и размещение этих фрагментов по разным узлам сети; здесь необходима проверка соблюдения ограничений первичного ключа и уникальных ключей.

9. Распределенные запросы. Оптимизация и выполнение распределенных запросов. Уровни диалекта SQL. Распределённые транзакции.

Распределенные запросы используются для доступа к данным из нескольких источников данных, которые могут размещаться на одном или различных компьютерах.

Microsoft SQL Server поддерживает распределенные запросы к следующим типам источников данных:

3) Распределенные данные, хранящиеся в нескольких экземплярах SQL Server
4) Удаленные данные, доступ к которым предоставляется с использованием поставщика OLE DB

OLE DB - это низкоуровневый интерфейс для доступа к данным. Он определяет набор COM-интерфейсов для предоставления приложениям единообразного доступа к данным, хранящимся в различных источниках информации (данные предоставляются в табличных объектах, именуемых «наборами строк»).

Оптимизация распределенных запросов:

- индексированный доступ при помощи поставщиков индексов OLE DB: SQL Server может пользоваться стратегиями выполнения, которые включают применение индексов от поставщика индексов для вычисления предикатов и выполнения операций сортировки для удаленных таблиц;
- удаленное выполнение запроса при помощи поставщиков OLE DB SQL-команд: SQL Server пытается Делегировать выполнение как можно большей части распределенного запроса поставщику SQL-команд.

Удаленное выполнение распределенных запросов

Запрос, который производит доступ только к удаленным таблицам, находящимся в источнике данных поставщика, извлекается из исходного распределенного запроса и выполняется поставщиком

На степень делегирования выполнения распределенного запроса влияют следующие факторы:

- уровень диалекта, поддерживаемый поставщиком SQL-команд: SQL Server делегирует только те операции, которые поддерживаются имеющимся уровнем диалекта;
- совместимость параметров сортировки: для распределенного запроса семантика сравнения для всех символьных данных определяется кодировкой и порядком сортировки локального экземпляра SQL Server. SQL Server может делегировать операции сравнения и ORDER BY для символьных столбцов поставщику только в том случае, если тот сможет определить, что
    - столбец в источнике данных имеет ту же кодировку и параметры сортировки;
    - семантика сравнения символов соответствует стандартам SQL-92 и SQL Server.

Уровни диалекта SQL

существуют следующие уровни диалекта, в порядке убывания от старшего к младшему (каждый из перечисленных уровней диалекта является надмножеством более низких уровней):

- SQL Server - внешние соединения (OUTER JOIN), CUBE, ROLLUP, оператор получения остатка от деления (%), битовые операторы (&, I, ^, ~), строковые и системные арифметические функции;
- SQL-92 начального уровня (SQL-92 Entry Level) - UNION, UNION ALL;
- базовый ODBC(ODBC Core) - агрегационные функции с ключевым словом DISTINCT и символьные константы;
- Jet (SQL Minimum) - статистические функции без ключевого слова DISTINCT, сортировка (ORDER BY), внутренние соединения (INNER JOIN). предикаты, операторы вложенных запросов (EXISTS, ALL, SÔME, IN), DISTINCT, арифметические операции и константы, не упомянутые на более высоких уровнях, и логические операторы;

дополнительные свойства SQL, поддерживаемые провайдерами:

- поддержка GROUP BY и HAVING;
- поддержка подзапросов (вложенных запросов);
- поддержка констант даты/времени;
- поддержка LIKE;
- поддержка внутренних соединений;
- поддержка параметрических запросов (маркеров параметров).

Распределенные транзакции

Участники распределенной транзакции:

- два или более сервера, которые называются диспетчерами ресурсов (экземпляры SQL Server Database Engine);
- диспетчер транзакций, который обеспечивает управление транзакцией путем координации между диспетчерами ресурсов

Двухфазовая фиксация транзакций (two-phase commit protocol - 2PC):

- фаза подготовки: каждый из диспетчеров ресурсов обеспечивает устойчивость транзакции, в частности, записывая журнал транзакции на диск;
- фаза фиксации: начинается в случае успешного завершения подготовки всеми диспетчерами ресурсов и завершается после получения диспетчером транзакций подтверждений об успешной фиксации от каждого из диспетчеров ресурсов;

Управление распределенными транзакциями может осуществляться через вызов инструкций Transact-SQL (как непосредственно, так и через ADO, OLE DB и ODBC), а также через вызов функций API OLE DB и ODBC.