**Базы и банки данных.**

Захарченко Константин Васильевич.

Intuit.ru

[\\Global\BD\Books](file:///%5C%5CGlobal%5CBD%5CBooks)

**Введение. Понятие информационной системы.**

**Понятие информационной системы. Классификация информационных систем.**

*Информационная система* это автоматизированная система, предназначенная для организации хранения, пополнения, поддержки, представления пользователем информации в соответствии с их запросами.

По типу хранимых данных информационные системы:

- *фактографические* – предназначены для хранения и обработки структурированных данных в виде чисел и текста;

- *документальные –* информация представлена в виде документов состоящих из наименований, описаний, рефератов и текстов.

Поиск по неструктурированным данным осуществляется с использованием семантических признаков. Найденные документы предоставляются пользователям, обработка данных в таких системах практически не производится.

Основываясь на степени автоматизации информационных процессов в системе информационные системы делятся на:

- *ручные* – характеризуются отсутствием технических средств обработки информации и выполнения всех операций человеком;

- *автоматические* - все операции обработки информации выполняются без участия человека;

- *автоматизированные* – предполагают участие в процессе обработки информации и человека и технических средств, причем главную роль в выполнении рутинных операций обработки информации находится компьютеру. Именно этот класс информационных систем в настоящее время обозначается термином *информационная система*.

В зависимости от характера обработки данных информационные системы:

- *информационно-поисковые* – производят ввод, систематизацию, хранение и выдачу информации по запросу пользователя без сложных преобразований данных (информационные системы библиотек);

- *информационно-решающие* – кроме того осуществляют операции переработки информации по определенному алгоритму.

По характеру использования выходной информации такие системы принято делить на:

- *управляющие* – результирующая информация трансформируется в принимаемое человеком решение, для этих систем характерны задачи расчетного характера и обработка больших объемов данных (информационные системы планирования производства, бухучета).

- *советующие* – вырабатывают информацию которая принимается человеком к сведению и учитывается при формировании управленческих решений, а не инициирует конкретные действия. Такие системы имитируют интеллектуальные процессы обработки знаний а не данных (например, экспертные системы).

В зависимости от сферы применения различают следующие классы информационных систем:

1. *Информационные системы организационного управления* предназначены для автоматизации функций управленческого состава как промышленных предприятий так и непромышленных объектов. Основные функции: оперативный контроль и регулирование, оперативный учет и анализ, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом и снабжением, и другие экономические и организационные задачи.

*2. Информационные системы управления технологическими процессами* служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно предусматривается наличие развитых средств, изменение параметров технологических процессов (температуры, давления, хим. состава).

*3. Информационные системы автоматизированного проектирования (ИСАП)* – предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями подобных систем являются инженерные расчеты, создание проектной документации, моделирование проектируемых объектов.

*4. Корпоративные информационные системы* – используются для автоматизации всех функций предприятия и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности.

**Жизненный цикл программного обеспечения.**

Жизненный цикл это модель создания и использования программной системы. Жизненный цикл отражает различные состояния программной системы, начиная с момента возникновения необходимости в этой программной системе и принятия решения о ее создании и заканчивая полным изъятием программной системы из эксплуатации. Жизненный цикл программного обеспечения носит, как правило, итерационный характер, т.е. реализуются этапы, начиная с самых ранних которые циклически повторяются в соответствии с изменением требований, внешних условий, введением ограничений.

**Модели жизненного цикла программного обеспечения.**

Существует несколько моделей жизненного цикла программного обеспечения, которые определяют порядок исполнения этапов в разработке и критерии перехода от этапа к этапу. К настоящему времени наиболее распространенными являются две модели жизненного цикла:

- *каскадные*;

- *спиральные*.

В существовавших ранее однородных информационных системах каждое приложение представляло собой единое целое. Для разработки таких приложений применялась *каскадная* модель жизненного цикла, которую также называют *классической* или *водопадной*. При использовании каскадной модели жизненного цикла разработка рассматривается как последовательность этапов, причем переход на следующий, более низкий, этап происходит только после того, как завершены все работы на предыдущем этапе. Подразумевается что при использовании каскадной модели разработка начинается на системном уровне и проходит через анализ проектирования, кодирования, тестирования и сопровождения.

Системный анализ определяет роль каждого элемента в комплексной системе и взаимодействие каждого элемента друг с другом.

СА

Анализ

Проектирование

Разработка

Тестирование

Сопровождение

Поскольку программное обеспечение рассматривается как часть большой системы, то анализ начинается с определения требований ко всем систем элементам. На этом этапе начинается решение задач, планирование проектов. В ходе планирования проекта определяется объем проектных работ, необходимые трудозатраты, формируются рабочие задачи, план и график работ.

Анализ требований. На этапе анализа уточняются и детализируются функции каждого элемента, его характеристики и интерфейс. На этом же этапе завершается решение задач планирования проекта.

Проектирование. Состоит в создании:

1. Архитектуры программного обеспечения.

2. Модульной структуры программного обеспечения.

3. Алгоритмической структуры программного обеспечения.

4. Структуры данных.

5. Входного и выходного интерфейса.

При решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

Разработка (кодирование). Заключается в переводе результатов проектирования в программы.

Тестирование. Выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

Сопровождение. Внесение изменений в эксплуатируемое программное обеспечение с целью исправления ошибок, адаптации к изменениям внешней для программного обеспечения среды и усовершенствование программного обеспечения в соответствии с требованиями заказчика.

*Достоинства применения каскадной модели:*

- дает план и временной график по всем этапам проекта, упорядочивая, таким образом, ход разработки;

- на каждом этапе формируется законченный набор проектной документации проверенной на полноту и согласованность;

- выполняемые в логической последовательности этапы работы позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадная модель хорошо себя зарекомендовала при построении информационных систем для которых в самом начале разработки можно достаточно точно сформулировать все требования к системе.

*Недостатки каскадной модели:*

- реальные проекты часто требуют отклонения от стандартной последовательности шагов;

- каскадная модель основана на точной формулировке исходных требований к программному обеспечению, однако реально в ряде случаев требования заказчика определены лишь частично;

- результат реализации проекта доступен заказчику только после завершения всех работ.

Из за необходимости в процессе создания программного обеспечения постоянного возврата к предыдущим этапам и уточнения или пересмотра ранее принятых решений реальный процесс разработки программного обеспечения на основе каскадной модели имеет следующий вид:

СА

Анализ

Проектирование

Разработка

Тестирование

Сопровождение

**Спиральная модель** является классическим примером эволюционной стратегии конструирования программного обеспечения, базируется на лучших свойствах каскадной модели с добавлением анализа риска. Спиральная модель включает четыре основных этапа.

*Планирование.* Определение целей, вариантов и ограничений.

*Анализ риска.* Включает анализ вариантов и распознавание риска.

*Конструирование.* Предполагает разработку программного продукта следующего уровня.

*Оценивание заказчиком текущих результатов конструирования.*

При движении по спирали строятся все более полные версии программного обеспечения по мере продвижения от центра к периферии. Проект может быть остановлен в результате анализа риска, если риск оказывается слишком велик. Движение по спирали продвигает заказчика и разработчика к более общей модели системы. Количество действий по разработке возрастает по мере продвижения от центра спирали.

*Достоинства спиральной модели:*

- наиболее реально отражает процесс разработки программного обеспечения;

- позволяет явно учитывать риск на каждом витке эволюции разработки;

- использует моделирование для уменьшения риска и совершенствования данного продукта.

*Недостатки спиральной модели*:

- трудности контроля и управления временем разработки;

**Структурный анализ и проектирование.**

**Понятие структурного анализа.**

На этапе анализа требований к системе формализуются, документируются и уточняются требования заказчика из-за неполноты или неточностей которых разработка проекта может закончиться неудачей. *Список требований* в разрабатываемой системе должен включать:

- совокупность условий при которых будет эксплуатироваться программная система, в том числе аппаратных и программных ресурсов и требований квалификаций сотрудников, который будут работать с системой;

- написание выполняемых системой функций;

- ограничение на процессы разработки, к которым можно отнести сроки завершения работ и мероприятия по защите информации.

Особенностью разработки программного обеспечения является то, что наиболее сложные работы выполняются на начальных этапах жизненного цикла, т.е. на этапах анализа и проектирования. Последующие этапы имеют значительно меньшую сложность и трудоемкость. Ошибки, допущенные на этапах анализа и проектирования порождают на следующих этапах трудные не разрешимые проблемы. Анализ системы оказывает более существенное влияние на все последующие этапы создания программного продукта при этом является наименее изученным процессом. Если требования не были зафиксированы документально, то для участников проекта они остаются как бы несуществующими. Язык, на котором формулируются требования к системе должен быть достаточно простым и понятным как заказчику, так и представителю разработчика – обычному системному аналитику.

*Системный аналитик должен уметь решать следующие задачи:*

- получение исчерпывающей информации для оценки требований к системе;

- уметь выбирать только существенную информацию на предметной области поскольку заказчик не может судить о том какие из его требований являются выполнимыми, а какие – нет, в результате чего системный аналитик получает от заказчика чрезмерное количество информации;

- спецификация системы, которую составляет аналитик из-за технических терминов и значительного объема часто непонятны заказчику. Если спецификации системы не были поняты заказчиком, то они будут непонятны и некорректны, в результате чего проектировщики и программисты получат недостоверную информацию.

Решение этой проблемы состоит в использовании методов структурного анализа. Для метода структурного анализа характерно разбиение описания системы на уровне абстрактного представления. Метод структурного анализа состоит в том, что исследования системы начинается с общего обзора, а затем выполняется более детальное исследование результаты которого приобретают иерархическую структуру. На каждом уровне абстрактного представления учитываются только существенные для данного уровня детали, определяются правила и формальное описание компонентов.

Методология структурного анализа базируется на ряде общих принципов. Эти принципы регламентируют организацию работ на начальных этапах жизненного цикла. А также используется для выработки рекомендаций по организации работ на последующих этапах.

*Основные принципы структурного анализа:*

- решение трудных задач выполняется путем из разбиения на множество меньших относительно независимых задач;

- принцип иерархического упорядочивания, этот принцип означает что система может быть разбита на несколько уровней, каждый уровень должен добавлять в систему новые существенные детали;

- принцип абстрагирования заключается в выделении наиболее существенных аспектов системы для представления проблемы в простом общем виде;

- принцип формализаций, состоит в необходимости применения строгого методического подхода для решения всех задач;

- принцип упрятывания, заключается в том, что несущественная на конкретном этапе информация скрывается, т.е. каждый компонент системы получает только ту информацию, которая ему необходима;

- принцип концептуальной общности означает, что на всех этапах жизненного цикла должна использоваться единая методология;

- принцип полноты, заключается в выполнении контроля присутствия в функциях системе лишних элементов;

- принцип непротиворечивости, состоит в проверке обоснованности использования и согласованности всех элементов системы;

- принцип логической независимости, состоит в том, что проектирование выполняющееся на логическом уровне не должно определяться последующим физическим проектированием;

- принцип независимости данных, состоит в том, что модель данных должна быть спроектирована независимо от процесса и их логической обработки;

- принцип структурирования данных означает, что данные в программной системе должны быть представлены в виде набора структур;

- принцип доступа конечного пользователя, означает что пользователь должен иметь возможность без программирования изменять значения данных в базе данных.

**Средства структурного анализа.**

Существует три группы средств структурного анализа которые иллюстрируют:

- функции, которые система должна выполнять;

- отношение между данными;

- поведение системы зависящее от времени.

Методологией структурного анализа наиболее популярными являются следующие средства:

- диаграммы потока данных (Data Flow Diagram) – эти диаграммы используются для иллюстрации функций, которые система должна выполнять, и информационных потоков связывающих процессы в системе;

- диаграммы сущности связи (Entity Relationship Diagram);

- диаграммы переходов состояний.

**Моделирование потоков данных.**

В основе методологии моделирования потоков данных лежит построение модели, анализируемой информационной системой. Модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхнего уровня иерархии определяют основные процессы внешними входами и выходами. Они детализируются при помощи диаграмм более низкого уровня. Детализация продолжается до тех пор пока не будет достигнут такой уровень декомпозиции, на котором процесс становится элементарным. Результатом детализации является многоуровневая иерархия диаграмм. Источники информации которыми как правило являются внешние сущности порождают информационные потоки, которые переносят информацию к процессам либо к системам. Процессы в свою очередь обрабатывают информацию и порождают новые потоки, которые переносят информацию к другим процессам, накопителям данных или к внешним сущностям, являющимся потребителями информации. Основными элементами диаграмм потоков данных являются внешние сущности, процессы или подсистемы, хранилища или накопители информации, потоки данных. Изображение диаграмм потоков данных пользуется две нотации:

- нотация Йодана;

- нотация Гейна-Сарсона.

… таблица …

1. *Поток данных*. Используется для моделирования передачи информации от источника к получателю, ориентация стрелки указывает направление движения информации. В некоторых случаях информация передается в одном направлении, обрабатывается и возвращается к источнику. Такая ситуация может моделироваться двумя встречными потоками или одной двунаправленной стрелкой.

2. *Процесс*. Выполняет преобразование входных потоков данных в выходные в соответствии с действием, которое определяется именем процесса. Имя процесса должно содержать глагол в неопределенной форме или отглагольное существительное и, возможно, дополнение. (Например: «Выдать информацию о расходах», «Проверить платежеспособность»). Использование глаголов подобных «обработать», «отредактировать», «организовать» и т.д. означает что процесс требует дальнейшего анализа и детализации. Фактически процесс может быть реализован программой, аппаратными средствами, некоторыми подразделениями в организации, некоторым должностным лицом и т.д.. Каждый процесс имеет внутри диаграммы уникальный номер для ссылки на него внутри диаграммы. Номер процесса совместно с номером диаграммы образует уникальный индекс процесса во всей модели.

3. *Хранилище (накопитель данных).* Обеспечивает хранение данных, которые сохраняются между процессами. Накопитель данных представляет собой абстрактное устройство для хранения информации. Информацию можно в любой момент времени поместить в накопитель и через некоторое время извлечь, причем в любом порядке. Физически накопитель может быть реализован в виде картотеки, массивов оперативной памяти, файлов на диске, базы данных и т.д.. В общем случае накопитель является прообразом базы данных информационной системы и служит для описания данных и их связи с информационной моделью. Имя хранилища идентифицирует его содержимое и должно быть существительным. Если поток данных входит и выходит из хранилища, его структура соответствует структуре хранилища, то поток данных должен иметь то же имя, что и хранилище.

4. *Внешняя сущность* - объект предметной области не входящий в контекст информационной системы и являющийся источником или получателем данных. Определение объектов предметной области в качестве внешней сущности указывает на то, что этот объект находится за пределами границ информационной системе и в обработке данных не участвует. Имя внешней сущности должно быть существительным, в процессе анализа предметной области некоторые внешние сущности могут быть перенесены внутрь информационной системы или наоборот часть процессов информационной системы может быть вынесено за ее пределы и представлено внешними сущностями.

**Контекстная диаграмма.**

При построении модели сложной системы информационная система может быть представлена в самом общем виде на контекстной диаграмме. Контекстная диаграмма представляет модель системы наиболее общим образом. Контекстная диаграмма имеет звездообразную структуру в центре находится главный процесс, соединенный с источниками или получателями информации в котором участвуют внешние сущности. В общем случае каждый проект информационной системы должен иметь только одну контекстную диаграмму и ее единственный процесс не нумеруется. Однако для сложной информационной системы ограничится единственной контекстной диаграммой трудно, т.к. она будет содержать слишком большое количество внешних сущностей, которые будет сложно расположить на одной диаграмме. Признаками сложности контекстной диаграммы являются:

- наличие … сущностей;

- многофункциональность информационной системы с выявленной группировкой функций, в отдельные подсистемы.

Для сложных информационных систем строятся иерархии контекстных диаграмм в которых контекстная диаграмма верхнего уровня содержит не единственный процесс, а набор подсистем соединенных потоками данных. Иерархия контекстных диаграмм в этом случае определяет взаимодействие основных функциональных подсистем проектируемой информационной системы, как между собой, так и с внешними входными и выходными потоками данных и с внешними сущностями. Для каждой подсистемы выполняется декомпозиция контекстной диаграммы при помощи диаграмм потоков данных более низких уровней.

**Построение иерархии диаграмм потоков данных.**

Главная цель в построении иерархии диаграмм потоков данных состоит в том, чтобы сделать ясными и понятными требования к проектируемой системе на каждом уровне ее детализации. В процессе построения иерархии диаграмм потоков данных следует придерживаться следующих правил:

1. *Правило балансировки* – означает, что при детализации процесса детализирующая диаграмма будет содержать только те компоненты информационных потоков которые определены на детализируемой диаграмме.

2. На каждой диаграмме может быть размещено от 2 до 9 процессов. Однако рекомендуется располагать от 3 до 6 процессов.

3. Несущественные на данном уровне детали использоваться не должны.

4. Декомпозиция потоков данных производится одновременно с декомпозицией процессов.

5. Имена процессов и потоков данных должны отражать их суть.

6. Функционально идентичные процессы следует определять один раз на самом верхнем уровне где процесс необходим, а затем на более низких уровнях на этот процесс ссылаться.

7. Следует разделять управляющие и входные потоки.

8. Правило нумерации состоит в том, что при детализации процессов должна поддерживаться их иерархическая нумерация.

**Методология функционального моделирования SADT (Structured analysis and design technique).**

Основоположником методологии SADT является Д. Росс. Методология SADT является основой методологии IDEF0 (ICAM (Interaction computer and manufacture) Definition). Методология SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели какой-либо предметной области. Функциональная модель SADT отражает структуру объекта, т.е. производимые им действия и связи между этими действиями. Методология SADT основана на следующих *концепциях:*

1. Графическое представление блочного моделирования. Каждая функция изображается в виде блока, интерфейс входа и выхода представляются входными и выходными дугами. Взаимодействие … друг с другом определяется посредством интерфейсных дуг, описывающих когда и каким образом выполняются процессы и осуществляется управление ими.

2. Строгость и точность. Методология SADT требует точности исполнения процессов, но не накладывает чрезмерных ограничений на действия аналитиков.

*Правила методологии SADT:*

1. Ограниченное количество блоков на каждом уровне декомпозиции. Обычно от 3 до 6. Допускается от 2 до 9.

2. Связность диаграмм посредством нумерации блоков.

3. Уникальность меток и наименований.

4. Синтаксические правила для блоков и дуг.

5. Разделение входных и управляющих дуг.

**Состав функциональной модели SADT.**

Результатом применения методологии SADT является модель которая состоит из диаграмм, … и словарей данных. Диаграммы являются главными компонентами функциональной модели SADT. Все функции на диаграмме представляются в виде блоков. Соединение дуги с блоком определяет тип интерфейса между блоками. Диаграммы объединяются в иерархические структуры. Блоки изображаются прямоугольниками и сопровождаются текстами описывающими действия. В отличие от диаграмм потоков данных в методологии SADT каждая сторона блока имеет определенное особое значение. Левая сторона блока предназначена для входных дуг, описывающих информацию, которая подвергается обработке. Верхняя сторона блока предназначена для управляющей информации, описывающей условия выполнения процесса, правая сторона для результата выходной информации. Нижняя сторона блока предназначена для описания механизмов или исполнителей за счет которых выполняются преобразования.

**Иерархия диаграмм в методологии SADT.**

Построение модели в методологии SADT начинается с представления всей системы в виде одного блока и дуг, изображающих интерфейсы с функциями системы. Этот единственный блок представляет всю систему как единое целое и имя этого блока является общим, интерфейсные дуги представляют полный набор внешних интерфейсов системы в целом. Далее блок, который представляет систему в виде единого модуля детализируется на несколько блоков, соединенных интерфейсными дугами. При этом каждая подфункция может содержать только те потоки которые входят в исходную функцию. Дуги входящие в блок и выходящие из него на диаграмме верхнего уровня те же самые, что и на диаграмме нижнего уровня, т.к. и блок и диаграмма представляют одну и ту же часть системы. Блоки на диаграмме размещаются по ступенчатой схеме в соответствии с их доминированием. Под доминированием понимается влияние, оказываемое одним блоком на другие. Блоки должны быть пронумерованы в соответствии с их доминированием, номера блоков служат номерами идентификаторов для процессов и автоматически организуют эти процессы в иерархию моделей. В методологии SADT используется 5 типов связей между блоками для описания типов их отношений:

- управление;

- вход;

- управленческая обратная связь;

- входная обратная связь;

- связь типовых выход-исполнителей.

Отношения управления входа являются простейшими т.к. они отражают очевидные виды воздействия. Отношения управления возникают тогда, когда выход одного блока непосредственно влияет на блок с меньшим доминированием. Обратные связи это более сложные отношения, т.к. они отражают итерации. В частности управленческая обратная связь возникает когда выход некоторого блока влияет на блок с большим доминированием. Обходная обратная связь имеет место, когда выход одного блока становится входом другого блока с большим доминированием. Отношение выход-исполнитель встречаются редко и отражают ситуацию …

…

Обратная связь по входу.

…

Обратная связь по управлению.

…

Связь выход-исполнитель

…

Дуги в методологии SADT изображают наборы предметов. Поэтому они могут разветвляться и соединяться вместе. Разветвление дуги означает что часть ее содержимого или весь набор предметов может появиться в каждом ответвлении дуги. Дуга всегда помечается до разветвления. Каждая ветвь дуги может быть помечена по следующим правилам:

1. Считается что непомеченная ветвь содержит все предметы указанные в метке перед разветвлением.

2. Каждая ветка ветви уточняет что именно содержит эта ветвь.

Слияние дуг указывает что содержимое каждой ветви участвует в формировании объединенной дуги. После слияния дуга всегда помечается для указания нового набора. Перед слиянием каждая ветвь дуги может помечаться в соответствии со следующими правилами:

1. Непомеченные ветви содержат все предметы указанные в общей метке после слияния.

2. Каждая метке ветви до слияния указывает что именно содержит эта ветвь.

**Построение информационной модели системы. Проектирование баз данных.**

**Диаграммы сущности связи.**

Диаграммы сущности связи (ERD) предназначены для разработки информационной модели системы (модели данных). Диаграммы ERD обеспечивают стандартный способ определения данных и отношений между ними. С помощью диаграмм ERD иногда осуществляется детализация хранилищ данных функциональной модели системы. Также с помощью диаграмм ERD документируются объекты предметной области, которые называются сущностями. Важные для данной системы их свойства, которые называют атрибуты и их отношения с другими объектами, которые называют связями. Такой подход был предложен Ченом и получил дальнейшее развитие в работах Паркера. Нотация Чена для построения диаграмм ERD представляет набор средств моделирования данных.

**Сущности отношения и связи в нотации Чена.**

Сущность в нотации Чена представляет собой множество экземпляров реальных или абстрактных объектов, обладающих общими атрибутами, характеристиками. Любой объект системы может быть представлен только одной сущностью. Причем имя сущности должно соответствовать типу объекта а не его конкретному экземляру.

**Основные символы диаграммы ERD в нотации Чена.**

**…**

Независим представляет собой независимые данные которые всегда присутствуют в системе. При этом отношения с другими сущностями могут как существовать так и отсутствовать.

Зависимая сущность представляет данные зависящие от других сущностей в системе, поэтому, она всегда должна иметь отношение с другими сущностями.

Ассоциативная (ассоциированная) сущность представляет данные которые ассоциируются соотношением между двумя другими сущностями.

Неограниченное или обязательное отношение представляет собой безусловное отношение, т.е. отношение которое всегда существует пока существуют относящиеся к нему сущности.

Ограниченное – условное отношение между сущностями, существенно ограниченное отношение используется в сущности выступает отношение в соответствии с требованиями которые описываются с помощью связей. Значение связи характеризует ее тип. Тип может быть одним из следующего множества: 0 или 1 , 0 или более, 1, 1 или более, диапазон [m, n]. Пара значений связи принадлежащих одному и тому же отношению определяет тип этого отношения который может быть одним из следующих.

1. один к одному

2. один ко многим

3. многие ко многим

Отношение многие ко многим обычно используется на ранних этапах проектирования базы данных. На последующих этапах каждое из отношений типа многие ко многим должно быть преобразовано к сочетанию отношений типа один ко многим. При этом как правило появляются ассоциативные сущности и вводятся новые отношения.

**Типы связей в нотации Чена.**

Существуют обязательные и необязательные связи. Необязательные связи в нотации Чена считается связь в которой могут участвовать не все экземпляры сущности. В обязательной связи участвуют все экземпляры хотя бы одной сущности. Обязательные связи в нотации Чена бывают трех подтипов

- слабая связь

- связь супертип-подтип

- ассоциативная связь.

Обязательная связь описывает связь между независимой и зависимыми сущностями. Все экземпляры зависимой сущности могут существовать только при наличии экземпляров независимой сущности. Слабые связи существования одной сущности зависит от существования второй сущности. Ключ одной сущности является частью составного ключа другой сущности. Связи супертип-подтип общие атрибуты всех сущностей определяются сущности супертипа. Сущности подтипа наследуют все атрибуты супертипа. Экземпляры ассоциативной связи может существовать только при условии существования определенных экземпляров в каждой из взаимосвязанных сущностей. Ассоциативный объект в нотации Чена – объект который одновременно является и сущностью и связью. Связь между независимыми сущностями имеет атрибуты которые определяются в ассоциативной сущности. Диаграмма сущность-связь в нотации Чена строится по следующим правилам:

1. Каждая сущность, каждый атрибут и связь должны иметь имя. Но при этом связь супертипа и подтипа и ассоциативная связь могут не иметь имени.

2. Имя сущности должно быть уникально в рамках всей системы.

3. Имя атрибута должно быть уникально в рамках сущности.

4. Каждый атрибут должен иметь определенный тип данных.

5. Сущность связанная только необязательными связями с другими сущностями должна иметь ключевой атрибут. Ключевой атрибут должна иметь также сущность супертип связи супертип-подтип, а также необязательная сущность в обязательной связи.

6. Сущность подтип связи супертип-подтип не может иметь ключевых атрибутов.

**Диаграммы атрибутов классической модели Чена.**

Каждая сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности. Любой из этих атрибутов может быть определен как ключевой. Детализация сущности классической модели Чена осуществляется с помощью диаграмм атрибутов. Которые определяют наборы атрибутов входящие в состав каждой сущности. На диаграмме атрибутов каждый атрибут представляется в виде связи между сущностью и соответствующим доменом. Домен представляет собой множество допустимых значений атрибута. В результате диаграммы атрибутов состоят из сущности соответствующих ей атрибутов и соответствующих атрибутам доменов. Для идентификации ключевого атрибута на диаграмме атрибутов используется подчеркивание имени атрибута.

**Диаграммы категоризации.**

Сущность может быть разделена и представлена в виде двух или более сущностей и категорий или подтипов. Каждая сущность-категория имеет общий атрибут и отношение которое определяется на верхнем уровне один раз, а затем наследуется на нижнем. Сущности категорий могут иметь свои собственные атрибуты и отношения. Сущности-категории также могут быть закомпозированы на другие сущности-категории на следующем уровне. Расчипленная на категории сущность в терминах модели Чена соответствует понятие подтип. Одна и та же сущность может быть как общей сущностью так и сущностью категорий. Для демонстрации композиции сущностей используются диаграммы категоризации. Диаграммы категоризации модели Чена содержат общую сущность, две или более сущности-категории и специальный узел который называется дискриминатором. Дискриминаторы бывают четырех типов:

1. Определяет полное и обязательное вхождение. Т.е. сущность должна принадлежать к одной и только к одной из перечисленных категорий.

2. Определяет полное, но не обязательное вхождение. Т.е. сущность может быть одной и только одной из категорий.

3. Определяет неполное и обязательное вхождение. Т.е. сущность должна относится по крайней мере к одной из категорий.

4. Определяет неполное и необязательное вхождение. Т.е. сущность может быть по крайней мере одной из категорий.

**Нотация Баркера. Модель сущность-связь.**

Баркер предложил оригинальную нотацию которая позволяет интегрировать предложенные Ченом средства описания информационной модели. В нотации Баркера используется только один тип диаграмм – диаграммы сущности связи. На диаграмме сущность связь сущность представляется прямоугольником, содержащим имя сущности и список имен атрибутов, а также указатели ключевых атрибутов.

…

Все связи в нотации Баркера являются бинарными и представляют собой линии, соединяющие сущности. Для каждого конца связи должно быть определено имя и степень множественности. Т.е. один или много объектов участвуют в связи. По степени обязательности связи делятся на обязательные и необязательные. Для обозначения обязательной связи используется непрерывная линия, для необязательной – пунктирная.

CASE пакете ERWin – категоризация выполняется с помощью специальных символов-дескриминаторов. Семантика и смысл которых отличаются от нотации IE и IDEF1X.

…

Процесс разработки информационной модели системы включает следующие основные шаги:

1. *Идентификация сущностей –* каждая сущность должна обладать уникальным идентификатором. Уникальный идентификатор сущности – атрибут или совокупность атрибутов предназначенная для уникальной идентификации каждого экземпляра данной сущности. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от других экземпляров данной сущности по ключевым признакам. В случае полной идентификации каждый экземпляр сущности идентифицируется собственными ключевыми атрибутами. В случае неполной идентификации в идентификации экземпляров сущности участвуют экземпляры другой сущности с которой данная сущность связана идентифицирующей связью. Каждая сущность может обладать любым количеством связей с другими сущностями.

2. *Идентификация связей и указание типов отношений.* Каждой связи может присваиваться имя выражаемое глаголом или грамматическим оборотом.

3. *Идентификация атрибутов.* Каждый атрибут должен иметь имя, определяемое существительным. Имя описывающие представлено атрибутом, характеристикой. Атрибуты изображаются в виде списка внутри символа сущности. Каждый атрибут занимает отдельную строку в сущности. Атрибуты определяющие первичный ключ размещаются в верхней части списка. Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ это один или несколько атрибутов значение которых однозначно идентифицирует каждый экземпляр сущности. При существовании нескольких возможных ключей один из них выбирается в качестве первичного ключа, а остальные называются альтернативными ключами. В нотации Баркера модель данных может содержать сущности супертипы и подтипы, а также рекурсивные связи связывающие сущность саму с собой. Атрибуты бывают обязательные и необязательные. Обязательные атрибуты не могут принимать неопределенных значений. Обязательными являются все атрибуты первичного ключа а также могут являться некоторые не ключевые атрибуты.

**Метод IDEF1X.**

Метод IDEF1 был предложен Римеем. Данный метод основан на использовании подхода предложенного Ченом эквивалентного построению реляционной модели данных в 3й нормальной форме. В настоящее время наиболее широко используется новая версия методологии IDEF1, которая называется IDEF1X, обеспечивающая простоту изучения и наглядное представление проекта базы данных, а также возможность автоматизации проектирования.

*Сущность является независимой* если каждый экземпляр сущности может быть однозначно идентифицирован без его отношений с другими сущностями. Сущность называется зависимой если однозначна идентификация экземпляра сущности зависит от ее отношений с другими сущностями с которыми данная сущность связана идентифицирующей связью.

В IDEF1X допускаются следующие типы (степени) связей:

1. Каждый экземпляр сущности может иметь 0, 1, или более связанных с ним экземпляров другой сущности.

2. Каждый экземпляр сущности может иметь не мене 1 связанного с ним экземпляра другой сущности.

3. Каждый экземпляр сущности должен иметь не более одного связанного с ним экземпляра другой сущности.

4. Каждый экземпляр сущности связан с некоторым фиксированным числом экземпляра другой сущности.

Если экземпляр сущности однозначно определяется своей связью с другой сущностью связь называет идентифицирующей, в противном случае – не идентифицирующей. Подчиненная сущность в идентифицирующей связи является зависимой сущностью. Подчиненная сущность не идентифицирующей связи является независимой сущностью при условии что она не связана идентифицирующей связью с какой-либо другой сущностью. Сущности могут иметь также внешние ключи которые могут использоваться в качестве части или целого первичного ключа или в качестве не ключевого атрибута. Внешний ключ обозначается с помощью имени атрибута после которого в скобках пишется (FK) – Foreign Key.

**Методика построения информационной модели системы или модели данных или модели сущность связь.**

Разработка информационной модели системы включает следующие основные этапы:

1. Идентификация сущностей, их атрибутов, их первичных и альтернативных ключей.

2. Идентификация отношений между сущностями и указание типов этих отношений.

3. Разрешение неспецифических отношений. Неспецифическими для реализованной модели данных являются отношения типа многие ко многим. … является определяющим при построении модели данных. Исходной информацией для данного этапа является содержимое хранилищ функциональной модели системы. На первом этапе осуществляется упрощение схемы отношений за счет ее нормализации путем сокращения повторяющихся строк в таблице. Нормализация всегда выполняется путем расчипления сущности на две или более сущности.

Методы нормализации реляционной базы данных были предложены Коддом в работах посвященных реляционной модели данных. Он определил для схемы отношений существование трех нормальных форм (1-й, 2-й и 3-й). В дальнейшем было установлено также нормальной формы Бойса-Кодда, а также 4-й и 5-й нормальной формы. Сущность находится в первой нормальной форме если в таблице, созданной на основе этой сущности отсутствуют повторяющиеся записи. Сущность находится во второй нормальной форме, если все не ключевые атрибуты функционально полно зависят от первичного ключа и не зависят функционально полно ни от какой части первичного ключа. Сущность находится в третьей нормальной форме, если она находится во второй нормальной форме и никакой из не ключевых атрибутов не зависит ни от какого другого не ключевого атрибута.

**Алгоритм приведения ненормализованной схемы к 3й нормальной форме.**

1. Создать схему состоящую из сущностей таблицы которой не будут содержать повторяющихся строк.

2. Объявить один или более атрибутов с первичными ключами таким образом, чтобы количество атрибутов входящих в состав первичного ключа было минимальным.

3. Для схем сущностей где ключи состоят более чем из одного атрибута надо проверить чтобы каждый не ключевой атрибут функционально полно зависел от первичного ключа. Если это не так, схему сущности необходимо расщепить.

4. Проверить являются ли все не ключевые атрибуты взаимно независимыми, если это не так необходимо выполнить декомпозицию соответствующей сущности.

Определение отношений состоит в выявлении связей между сущностями последующей их проверке в обоих направлениях.

Каждое не специфическое отношение преобразуется в 2 специфических с введением новых ассоциативных сущностей. Каждый экземпляр ассоциативной сущности связан с одним экземпляром каждой из сущностей в отношении многие ко многим. Таким образом ассоциативная сущность по своей природе является представлением связи реальных объектов и при проектировании появляется только на последнем этапе.

**Реляционная модель данных.**

**Классификация баз данных и их архитектур.**

В истории развития баз данных было разработано множество способов хранения, структуризации и обработки информации. По технологии обработки информации базы данных подразделяют на распределенные и централизованные. Для распределенных баз данных предполагается использование нескольких серверов на которых может храниться пересекающаяся или даже дублирующаяся информация. Для работы с распределенными базами данных используется системы управления распределенными базами данных. Централизованная база данных располагается на одном компьютере. Если для этого компьютера включена поддержка сети то множество пользователей с других компьютеров могут одновременно обращаться к информации хранящейся в центральной базе данных.

Система централизованных баз данных с сетевым доступом может иметь различные архитектуры:

1. Файл-сервер – эта архитектура предполагает использование выделенного компьютера в качестве сервера файлов. На этом сервере хранятся файлы базы данных которые по запросу пользователей передаются на их локальные компьютеры где и производятся основная обработка данных. После того, как пользователи выполняют необходимые изменения данных они копируют файл обратно на сервер где другие пользователи смогут снова использовать этот файл. Кроме того каждый пользователь может создавать на своем компьютере свои собственные базы данных которые используются монопольно. Использование архитектуры файл-сервер производительность системы резко падает по увеличению количества пользователей.

2. Клиент-сервер. При использовании этой архитектуры выделенный компьютер используется не только в качестве хранилища файлов но и выполняет основной объем действий по обработке информации. Пользователь с рабочей станцией отправляет на сервер список операций которые необходимо выполнить в виде запроса на языке SQL. Сервер выполняет необходимое вычисление, выборку данных или другие операции по обработке информации и отправляет готовый результат клиенту.

**Модели данных.**

Помимо подразделения баз данных по методам обработки информации их можно классифицировать по используемым моделям или структурам данных. Модель данных включает в себя структуры данных, обеспечивающие хранение информации, операции обработки данных и ограничение целостности. С помощью модели данных можно наглядно представить структуру объектов базы данных и установленные между ними связи. К настоящему времени разработано множество различных моделей данных но на практике используются три основных:

1. *Иерархическая модель данных* имеет иерархическую структуру. Т.е. каждый из ее элементов связан только с одним вышестоящим элементом и в то же время на него могут ссылаться один или несколько ниже стоящих элементов. Иерархическая модель схематично изображается в виде дерева и представляет собой совокупность элементов расположенных в порядке из подчинения от общего к частному. Достоинствами иерархической модели являются быстродействие и простота. В СУБД (системах управления базы данных) реализовано на основе иерархической модели отношение предок-потомок реализован в виде физических указателей из одной записи на другую. В следствие чего перемещение в базе данных происходит очень быстро. Иерархические модели идеально подходят для систем с большим количеством неделимых операций. Например, систем управления банкоматами.

2. *Сетевая модель данных* – использует ту же терминологию что и иерархическая с тем отличием, что в сетевой модели данных каждый элемент данных может быть связан с любым другим элементом. Если структура данных оказывается сложнее чем сеть или иерархия простота иерархической или сетевой модели данных становится ее существенным недостатком.

3. *Реляционная* *модель данных.* Основная идея реляционной модели данных заключается в том, чтобы представить любой набор данных в виде двумерных таблиц. В простейшем случае реляционная модель описывает единственную двумерную таблицу, но как правило эта модель описывает структуру и взаимодействие нескольких двумерных таблиц. Развитие реляционных баз данных началось в 60х годах когда появились первые работы в которых обсуждалась возможность использования при проектировании баз данных привычных и естественных способов представления данных так называемых табличных дата-логических моделей. Теория реляционных баз данных разработана в 60х годах в США доктором Коддом. Теория имеет мощную математическую основу описывающую правила эффективной организации данных. Разработанная Коддом теоретическая база стала основой теории проектирования баз данных. Кодд предложил использовать для обработки данных классический аппарат теории множеств и доказал что любой набор данных можно представить в виде двумерных таблиц особого вида известных в математике как отношения. Термин отношения в реляционной модели данных обозначает таблицу. Наименьшая единица данных с которой оперирует реляционная модель данных это отдельная атомарная для данной предметной области значение данных которая не может быть разложена на более простые составляющие. Множество атомарных значений одного и того же типа образует домен. В самом общем виде домен определяется заданием некоторого базового типа данных к которому относятся элементы домена и произвольного логического условия применяемого к этим элементам данных. В простейшем случае домен определяется как допустимое потенциальное множество значений одного типа. В один домен могут входить значения из нескольких колонок объединенных помимо одинакового типа данных еще и логически. Если два значения берутся из одного и того же домена можно выполнить сравнение этих двух значений. Каждый элемент данных в отношении может быть определен с указанием его адреса в формате aij, где a – элемент данных, i – строка отношения, j – номер атрибута отношения. Количество атрибутов в отношении определяет его порядок. Множество значений aij-x при постоянном i и всех возможных j образует кортеж отношения или просто строку таблицы. Количество всех кортежей в отношении определяет его мощность или кардинальное число. Мощность отношения в отличие от порядка отношения может со временем меняться. Совокупность всех кортежей образует тело отношения. Поскольку некоторые отношения являются математическими множествами, которые по определению не могут содержать совпадающих элементов никакие два кортежа отношения не могут быть дубликатами друг друга в любой момент времени. Некоторые множества атрибутов образуют ключ для данного отношения если задание значений этих атрибутов определяет значение всех остальных атрибутов таблицы. Множество атрибутов отношения является возможным ключом этого отношения тогда и только тогда, когда выполняются два независимых условия.

1. *Уникальность*. В каждый момент времени никакие два различный кортежа отношения не имеют одинакового значения для сочетания входящих в ключ атрибутов. Т.е. в таблице не может быть двух строк имеющих одинаковый ключ.

2. *Минимальность.* Ни один из входящих и исходящих в ключ атрибутов не может быть исключен из ключа без нарушения условия уникальность.

**Реляционная база данных**

Реляционная база данных это совокупность отношений содержащих всю информацию которая должна храниться в базе данных. Т.е. реляционная база данных представляет собой набор таблиц необходимых для хранения всех данных. Таблицы реляционной базы данных, как правило, логически связаны между собой. Требования к проектированию реляционных баз данных в общем случае можно свести к нескольким правилам:

1. Каждая таблица в базе данных имеет уникальное имя и состоит из однотипных строк.

2. Каждая таблица состоит из фиксированного числа клеток и значений, причем в одной колонке строки не может быть сохранено больше одного значения.

3. Ни в какой момент времени в таблице не должно быть двух строк дублирующих друг друга. Строки должны отличаться хотя бы одним значением чтобы была возможность …

4. Каждой колонке присваивается уникальное в пределах таблицы имя. Для каждой колонки устанавливается конкретный тип данных чтобы в этой колонке размещались однотипные значения. Умные информационные содержания базы данных представляется в виде явных значений самих данных и такой метод представления является единственным.

5. При выполнении обработки данных можно свободно обращаться к любой строке или к колонке таблицы. Значение хранимое в таблице не накладывает никаких ограничений на порядок обращения к данным.

**Функции систем управления базами данных (СУБД).**

Традиционных возможностей файловых систем оказывается недостаточным для построения даже простых информационных систем. Считается, что если прикладная информационная система опирается на некоторую систему управления данными, то эта система управления данными является СУБД при условии что она выполняет следующие функции:

1. *Непосредственное управление данными во внешней памяти*. Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, так и для служебных целей. Например, ускорения доступа к данным. В некоторых реализациях СУБД используются возможности существующих файловых систем. В других – эта функция выполняется в СУБД самостоятельно вплоть до уровня устройств внешней памяти. В развитых СУБД пользователи не обязаны знать использует ли СУБД файловую систему и если использует, то как организованы файлы.

2. *Управление буферами оперативной памяти.* СУБД обычно работает с базой данных значительных размеров которая обычно существенно больше доступного объема оперативной памяти. Если при обращении к любому элементу данных будет производиться обмен с внешней памятью то вся система будет работать со скоростью устройств внешней памяти. Фактически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти, при этом даже если операционная система выполняет общесистемную буферизацию, для нужд СУБД этого не достаточно поскольку СУБД располагает гораздо большей информацией о необходимости буферизации той или иной части базы данных. Поэтому в различных СУБД как правило поддерживается собственный набор буферов, собственной дисциплиной их замены.

3. *Управление транзакциями.* Транзакция – последовательность операций с базой данных рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменение базы данных, производя транзакцию во внешнюю память, либо ни одно из изменений никак не отражается на состоянии базы данных. Понятие транзакции необходимо для поддержания логической целостности базы данных. Поддержание минимума логической целостности является обязательным требованием даже для однопользовательских СУБД. Транзакция фактически является единицей активности пользователя по отношению к базе данных. Причем каждая транзакция начинается при целостном состоянии базы данных и оставляет это состояние целостным после своего завершения. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый пользователь может ощущать себя единственным пользователем базы данных. Управление транзакциями в многопользовательских СУБД связано с понятием сериализации транзакции и сериального плана транзакции. Под сериализацией параллельного выполнения транзакций понимают такой порядок планирования их работы при котором суммарный эффект вместе с транзакцией эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения транзакцией это план который приводит к реализации транзакции. Если удается добиться сериального выполнения смеси транзакций то для каждого пользователя по инициативе которого образуется транзакция присутствие транзакции других пользователей происходит практически незаметно если не считать некоторого закрепления работы по сравнению с однопользовательским режимом. При использовании любого алгоритма с реализацией транзакции возможны ситуации конфликта между двумя или более транзакциями при доступе к объектам базы данных. В случае конфликта для поддержания реализации транзакции необходимо выполнить откат одной или более транзакции чтобы ликвидировать все изменения произведенных в базе данных. Это один из случаев когда пользователь многопользовательской СУБД может реально ощутить присутствие в системе других пользователей.

4. *Журнализация.* Одно из основных требований СУБД является требование надежного хранения данных во внешней памяти. Под надежным хранением понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние базы данных после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратного сбоя: мягкие и жесткие. Мягкие сбои можно трактовать как внезапную остановку работы компьютера, например из-за выключения питания. Жесткие сбои характеризуются потерей информации на носителях внешней памяти. Кроме того, существуют программные сбои примерами которых могут быть аварийное завершение работы СУБД по причине ошибки в программе или аварийное завершение пользовательской программы в результате чего некоторые транзакции останутся незавершенными. В любом случае для восстановления базы данных необходимо располагать некоторой дополнительной информацией, т.е. поддержания надежного хранения данных в базе данных требует хранение избыточной информации. При этом та часть базы данных которая используется для восстановления информации должна храниться с особой надежностью. Наиболее распространенным способом поддержания избыточного хранения информации является ведение журнала изменений базы данных. Журнал это особая часть базы данных непосредственно недоступная пользователям СУБД в которую поступают записи обо всех изменениях основной части базы данных. В разных СУБД изменения базы данных журнализируется на разных уровнях. Иногда запись в журнале соответствует некоторой логической операции изменения базы данных. Например операция удаления строки из таблицы. В других случаях запись в журнале соответствует некоторой логической операции модификации страницы внешней памяти изменения базы данных. В некоторых СУБД используются одновременно оба подхода. Во всех случаях используется стратегия упреждающей записи в журнал (Write Ahead Log). Эта стратегия заключается в том, что запись об изменении любого объекта базы данных должна попасть в журнал прежде чем изменения отразятся во внешней памяти основной части базы данных. Если СУБД корректно соблюдает протокол упреждающей записи в журнал, то с помощью журнала можно решить все проблемы восстановления базы данных после любого программного или мягкого аппаратного сбоя.

Самая простая ситуация восстановления базы данных индивидуальный откат транзакций. Для этого не требуется общесистемный журнал базы данных. Достаточно поддерживать локальный журнал изменений базы данных. При мягком сбое во внешней памяти основной части базы данных могут находиться объекты модифицированные транзакциями не закончившиеся к моменту сбоя. И могут отсутствовать объекты модифицированные транзакциями которые к моменту сбоя успешно завершились. Это обусловлено использованием буферов оперативной памяти содержимое которых пропадает при мягком сбое. При соблюдении протокола упреждающей записи в журнал во внешней памяти журнала должны гарантировано находиться записи относящиеся к модификации обоих видов объектов. Целью процесса восстановления после мягкого или программного сбоя является состояние внешней памяти основной части базы данных которая возникла бы при фиксации во внешней памяти изменений всех завершившихся транзакций и не содержало бы никаких следов незаконченных транзакций. Для того чтобы этого добиться сначала производят откат незавершенных транзакций, а потом повторно воспроизводят те операции завершившихся транзакций которые не отображены во внешней памяти.

Для восстановления базы данных после жесткого сбоя используется журнал и архивная копия базы данных. Архивная копия базы данных это полная копия к моменту начала заполнения журнала. Восстановление базы данных после жесткого сбоя состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций которые закончились к моменту сбоя. Можно также воспроизвести работу незавершенных транзакций и продолжить их выполнение после завершения восстановления, однако в реальных системах этого обычно не делается.

5. *Поддержка языков баз данных.* Стандартным языком наиболее распространенных СУБД является SQL.

**Типовая организация СУБД.**

В современных СУБД логически можно выделить следующие *основные компоненты*:

1. *Ядро СУБД*.

2. *Компилятор языка баз данных.*

3. *Подсистема поддержки времени выполнения.*

4. *Набор утилит.*

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно можно выделить такие компоненты ядра как менеджер данных, менеджер буфера, менеджер транзакций, менеджер журнала. Функции всех компонентов ядра взаимосвязаны и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать под тщательно продуманным и проверенным протоколом. Ядро СУБД обладает собственным интерфейсом, недоступным пользователям напрямую и используемым в программах созданных средствами SQL а также в утилитах баз данных. При использовании архитектуры клиент-сервер ядро является основным компонентом серверной части системы.

Основной функцией компилятора языка баз данных является компиляция операторов языка базы данных в некоторую управляемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем являются не процедурными поэтому компилятор должен решить каким образом выполнить операторы языка прежде чем воспроизвести программу. Результатом компиляции является выполняемая программа представленная в некоторых системах в машинных кодах, но, как правило, выполняемым внутренним машинно-независимом коде. В последнем случае реальное выполнение операторов производится с привлечением подсистемы поддержки времени выполнения, которая представляет собой интерпретатор внутреннего языка СУБД. В отдельные утилиты базы данных обычно выделяют такие процедуры которые слишком накладно выполнять с использованием языка баз данных, например, глобальная проверка целостности базы данных, сбор статистики и т.д.. Утилиты программируются с использованием интерфейса ядра СУБД а в некоторых случаях проникновением внутрь ядра.

**Целостность сущностей и ссылок.**

Согласно концепции предложенной Дейтом реляционная модель состоит из трех частей: структурной, манипуляционной и целостной. В целостной части реляционной модели данных фиксируется два базовых принципа которые должны поддерживаться в любой реляционной СУБД:

1. *Требование целостностей сущностей* – объекту или сущности реального мира в реляционной базе данных ставится в соответствие кортеж некоторого отношения. Требование целостностей сущностей состоит в том, что любой кортеж любого отношения должен быть отличен от любого другого кортежа этого же отношения или любое отношение должно обладать первичным ключом.

2. *Требование целостности по ссылкам* – атрибут называется внешним ключом если его значение однозначно характеризует сущности представленными кортежами некоторого другого отношения, т.е. задает значение их первичного ключа. Требование целостности по ссылкам состоит в следующем – для каждого значения внешнего ключа появляющемся в ссылающемся отношении в отношении на которое ведет ссылка должен найтись кортеж с таким же значением первичного ключа либо значение внешнего ключа должно быть неопределенным.

Ограничения целостности сущности и целостности по ссылкам должны поддерживаться на уровне СУБД. Для соблюдения требования целостностей сущностей достаточно гарантировать отсутствие в отношении кортежей с одним и тем же значением первичного ключа. При обновлении ссылающегося отношения или модификации значений внешнего ключа в существующих кортежах необходимо следить за тем, чтобы не появились некорректные значения внешнего ключа. При удалении кортежа из отношения существует три подхода поддержания целостности по ссылкам:

1. Запрещается производить удаление ключа на который существуют ссылки. Сначала нужно либо удалить ссылающиеся кортежи либо соответствующим образом изменить значение их внешнего ключа.

2. При удалении кортежа на который существуют ссылки во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным.

3. Каскадное удаление. При удалении кортежа из отношения в отношении на которое ведет ссылка автоматически удаляются все ссылающиеся кортежи.

**Базисные средства манипулирования реляционными данными.**

Выделяют два базовых механизма манипулирования реляционными данными: реляционная алгебра и реляционное исчисление. Эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает что выражение реляционной алгебры и формулы реляционного исчисления определяется над отношениями и результатом вычислений также является отношение. Язык манипулирования реляционными данными называется реляционно-полным если любой запрос выражаемый с помощью одного выражения реляционной алгебры или одной формулой реляционного исчисления может быть выражен с помощью одного оператора этого языка. Механизмы реляционной алгебры и реляционного исчисления эквивалентны, т.е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную, т.е. производящую такой же результат формулу реляционного исчисления и наоборот. Выражение реляционной алгебры строится на основе алгебраических операций и подобно тому, как интерпретируются арифметические и логические выражения, выражения реляционной алгебры имеют процедурную интерпретацию. Другими словами запрос представленный на языке реляционной алгебры может быть вычислен на основе элементарных алгебраических операций с учетом их старшинства и наличия скобок. Для формул реляционного исчисления однозначная интерпретация отсутствует. Формула только ставит условия которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются непроцедурными или декларативными.

**Реляционная алгебра.**

Основная идея реляционной алгебры состоит в том, что т.к. отношения являются множествами, то средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях дополненных некоторыми специальными операциями специфичными для баз данных. В варианте реляционной алгебры предложенной Фордом набор основных алгебраических операций состоит из восьми. Операции реляционной алгебры делятся на два класса: теоретико-множественные и специальные реляционные. В состав теоретико-множественных операций входят: объединение отношений, произведение отношений, взятие разностей отношений и прямого произведения. Специальные реляционные операции включают: ограничение отношений, проекцию отношений, соединение отношений по заданному условию и деление отношений. В состав операций реляционной алгебры входят также операции присваивания и переименования. Результатом ограничения отношений по некоторому условию является отношение включающее кортежи отношения операнда удовлетворяющие этому условию. При выполнении проекции отношения на заданный набор его атрибутов производится отношение кортежи которого получаются путем взятия соответствующих значений из кортежей отношения операнда. При соединении двух отношений по некоторому условию образуется отношение кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.

Операция реляционного деления. В операции реляционного деления

… два операнда бинарного и унарного отношения. Результатом операции является унарное отношение состоящее из кортежей включающих значение первого атрибута, кортежей первого операнда таких что множество значений второго атрибута при фиксированном значении первого атрибута совпадает с множеством значений второго операнда.

Операция присваивания предназначена для сохранения результатов вычислений в существующем отношении.

**Особенности теоретико-множественных операций реляционной алгебры.**

Смысл операций объединения, пересечения и взятия разности в реляционной алгебре остается теоретико-множественным. Но если в теории множеств операция объединения, пересечения, взятия разности имеет смысл для двух любых операндов, то в случае реляционной алгебры результатом операции объединения, пересечения и взятия разности должны являться отношения. Если допустить в реляционной алгебре возможность объединения двух отношений с разными схемами то результатом операции будет множество разнотипных кортежей но не обязательно отношения. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения то такая операция объединения, пересечения, взятия разности является бессмысленной. Подобные соображения привели к появлению понятия совместимости отношений по объединению. Два отношения совместимы по объединению в том и только в том случае когда они обладают одинаковыми заголовками. Это означает что в заголовках обоих отношений содержится один и тот же набор атрибутов и одноименные атрибуты определены на одном и том же домене. Если два отношения совместимы по объединению то при выполнении над ними операций объединения, пересечения или взятия разности результатом операции является отношение с корректно определенным заголовком который совпадает с заголовком каждого из отношения операндов. Если два отношения совместимы по объединению на всем кроме имен атрибутов то их можно сделать полностью совместимыми по объединению путем применения операций переименования. Другие проблемы связаны с операцией взятия прямого произведения двух отношений. В реляционной алгебре используется специализированная формула операции взятия прямого произведения так называемая расширенное прямое произведение отношений. Проблемой получения расширенного прямого произведения может быть именование атрибутов результирующего отношения если отношение операнды обладают одноименными атрибутами. Два отношения совместимы по взятию прямого произведения в том и только в том случае если множества имен атрибутов этих отношений не пересекаются. Любые два отношения могут быть сделаны совместимыми по взятию прямого произведения путем применения операции переименования к атрибутам одного из этих отношений.

**Реляционное исчисление.**

Базисными понятиями реляционного исчисления являются понятия переменной с опеределенной для нее областью допустимых значений и понятие правильно построенной формулы опирающееся на переменные, предикаты и кванторы. В зависимости от того что является областью определения переменных различают исчисление кортежей и исчисление доменов. При исчислении кортежей областями определения переменных являются отношения базы данных. Допустимые значения каждой переменной являются кортежем некоторого отношения. При исчислении доменов областью определения переменных являются домены для которых определены атрибуты отношений базы данных. Т.е. допустимым значением каждой переменной является значение некоторого домена. Правильно построенная формула служит для выражения условий накладываемых на кортежные переменные. Основой правильно построенных формул являются простые сравнения, которые представляют собой операции сравнения скалярных значений, т.е. значений атрибутов, переменных или констант. Более сложные варианты правильно построенных формул строятся с помощью логических операций ИЛИ, НЕ и предиката ЕСЛИ-ТО. Допускается также построение правильно построенных формул с помощью кванторов EXISTS и FORALL

Переменные входящие в правильно построенные формулы могут быть свободными или связанными. Все переменные входящие в правильно построенные формулы при построении которых не используются кванторы являются свободными. Если переменная используется в правильно построенной формуле сразу после квантора, то эта переменная называется связанной и это означает что такая переменная не видна за пределами правильно построенной формулы связавшей эту переменную.

**СУБД в архитектуре клиент-сервер.**

Применительно к системам баз данных архитектура клиент-сервер актуальна главным образом потому, что обеспечивает простое и относительно дешевое решение проблемы коллективного доступа к базам данных в локальной вычислительной сети. В каком-то смысле система баз данных основанная на архитектуре клиент-сервер является приближением к системе распределенных баз данных не требующим решения основного набора проблем действительно распределенных баз данных.

**Открытые системы.**

Реальное распространение архитектуры клиент-сервер стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем. Основным смыслом концепции открытых систем является упрощение комплектования вычислительных систем за счет международной и национальной стандартизации которая касается как аппаратных так и программных интерфейсов. Главной побудительной причиной развития концепции открытых систем явился повсеместный переход к использованию локальных вычислительных сетей и те проблемы комплектования аппаратных программных средств которые этот переход вызвал. В связи с бурным развитием технологии глобальных коммуникаций открытые системы получили еще большее развитие и приобрели еще больший масштаб. Основой открытых систем дающих существенное преимущество пользователю является независимость от конкретного поставщика. Ориентируясь на продукцию компании поддерживающей стандарты открытых систем потребитель который приобретает продукты компании не попадает в зависимость от ее продуктов и может продолжить наращивание мощности своей системы за счет приобретения продуктов любой другой компании придерживающихся стандартов открытых систем. Причем это касается как аппаратных, так и программных средств. Технология стандарта открытых систем обеспечивает реальную и проверенную практикой возможность производства системных и прикладных программных средств со свойствами мобильности и интероперабельности. Свойство мобильности означает сравнительную простоту переноса программной системы в широком спектре аппаратно-программных средств соответствующих стандартам открытых систем. Интероперабельность означает упрощение комплектования новых программных систем на основе готовых компонентов со стандартными интерфейсами. Использование концепции открытых систем выгодно и производителям программного обеспечения и пользователям. Открытые системы обеспечивают естественное решение проблемы смены поколений аппаратных программных средств. Производителям аппаратных программных средств нет необходимости заново решать все проблемы создания нового продукта. Они могут по крайней мере некоторое время продолжать комплектовать системы используя существующие компоненты. При этом вытекает новый уровень конкуренции, поскольку все производители аппаратного и программного обеспечения должны создавать некоторую стандартную среду и при этом добиваться как можно лучшей ее реализации. Преимуществом для пользователей является то, что они могут постепенно заменять компоненты системы на более совершенные и при этом система не утрачивает своей функциональности.

….

Сервера баз данных интерфейс которых основан на языке SQL обладают своими преимуществами и недостатками. Главным преимуществом является стандартный открытый интерфейс, т.е. клиентская часть любой SQL среды может работать с любым SQL сервером независимо от того какая компания его разработала. Недостатком является то, что при высоком уровне интерфейса между клиентской и серверной частями системы на стороне клиента работает слишком мало программ СУБД. Это нормально если на стороне клиента используется маломощная рабочая станция, но если клиентский компонент обладает достаточной мощностью то часто возникает необходимость возложить на него больше функций по управлению базой данных, разгрузив сервер который в этом случае является узким местом всей системы.

**Типичное распределение функций между клиентами и серверами.**

Типичным на сегодняшний день является такое распределение функций между клиентами и сервером при котором на стороне клиента работает только такое программное обеспечение, которое не имеет непосредственного доступа к базе данных, обращается для этого к серверу с использованием языка SQL. В некоторых случаях необходимо включить в состав клиентской части системы функции для работы с локальной памятью базы данных, в частности с той ее частью, которая наиболее интенсивно используется клиентской прикладной программой. Используя современные технологии это можно сделать путем создания на стороне клиента локальной копии части базы данных с сервера и рассмотрение всей системы как набора взаимодействующих серверов. Требования к аппаратному и программному обеспечению компьютера клиентов и серверов различают в зависимости от вида используемой системы. Если разделение функций между клиентской и серверной частями достаточно жесткое как в большинстве современных СУБД, то пользователям на рабочей станции все равно какая аппаратура и операционная система работает на стороне сервера при условии, что он справляется с возникающим потоком запросов. Если могут возникнуть потребности перераспределения функций между клиентами и серверами, то программное обеспечение сервера должно это позволять.

**Распределенные базы данных.**

Основной задачей системы управления распределенной базы данных является интеграция локальных баз данных располагающихся в разных узлах сети для того, чтобы пользователи работающие в любом узле сети имели доступ ко всем базам данных, как к единственное базе данных. Существуют однородные и неоднородные базы данных. В однородной базе данных каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относится даже к разным моделям данных. Наиболее успешно в настоящее время решается задача интеграции неоднородных SQL ориентированных систем. Этому способствует стандартизация языка SQL и общее следование концепции открытых систем. Основная цель проекта – создание распределенной системы управления базами данных может быть сформулирована следующим образом: необходимо обеспечить средство интеграции локальных баз данных располагающихся в узлах вычислительной сети так чтобы пользователь работающий в любом узле сети имел доступ ко всем этим базам данных так, как если бы они были централизованными и при этом должны обеспечиваться:

1. Легкость использования системы.

2. Возможность автономного функционирования при нарушении связности сети.

3. Высокая степень эффективности работы системы.

Для решения этих задач был принят ряд проектных решений касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечение синхронизации, обнаружение и разрешение тупиков, восстановление согласованного состояния баз данных после разного рода сбоев в узлах сети. Легкость использования системы достигается за счет того, что пользователи остаются в среде языка SQL. Возможность использования SQL обеспечивает прозрачность месторасположения данных, т.е. система автоматически обнаруживает местоположение указанных в пользовательском запросе объектов данных. При этом в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе. Автономность узлов сети может быть обеспечена следующим образом: каждая локальная база данных администрируется независимо от других, возможно автономное подключение новых пользователей, смена версий автономной части системы и т.д. Система проектируется таким образом что ей не требуется централизованные службы именования объектов.

В индивидуальных узлах не требуется наличия глобального знания об операциях выполняющихся в других узлах сети. Работа с доступными базами данных может продолжаться при выходе из строя отдельных узлов сети и линий связи. Для достижения высокой степени эффективности работы системы используется два основных приема. Во-первых выполнению запроса предшествует его компиляция. На стадии компиляции производится множество действий предшествующих реальному выполнению запроса. Скомпилированная программа включающая приложение на SQL может выполняться много раз без дополнительных расходов ресурсов на компиляцию. Во-вторых средством повышения эффективности работы системы является возможность перемещения удаленных отношений в локальные базы данных.

**Распределенная компиляция запросов.**

Компиляция запросов может производиться на стадии предкомпиляции прикладной программы написанной на обычном языке программирования с включением приложения на SQL или в процессе выполнения транзакций с использованием SQL. С точки зрения пользователя процесс компиляции приводит к следующим результатам:

Для каждого приложения на SQL образуется программа в машинных кодах вызовы которой помещаются в текст исходной прикладной программы. А т.к. в действительности процесс компиляции запросов более сложен из-за наличия сетевых взаимодействий которые требуется при реальном выполнении программы. Будем называть главным узлом сети тот узел в котором инициирован процесс компиляции приложений SQL и дополнительными узлами те узлы которые вовлекаются в этот процесс в ходе его выполнения. При достаточно грубом приближении процесс распределенной компиляции запросов можно разбить на следующие стадии:

1. В главном узле сети производится грамматический разбор с построением внутреннего представления запроса в виде дерева. На основе информации из локального каталога главного узла и удаленных каталогов дополнительных узлов производится замена имен объектов фигурирующих в запросе на их системные идентификаторы.

2. В главном узле генерируется глобальный план выполнения запроса в котором учитывается только порядок взаимодействия узлов при реальном выполнении запроса. Глобальный план отображается в преобразованном соответствующим образом дереве запросов.

3. Если в глобальном плане выполнения запроса участвуют дополнительные узлы производится его декомпозиция на части каждую из которых можно выполнить в одном узле.

4. В каждом узле участвующем в глобальном плане выполнения запроса главном или дополнительном выполняется завершающая стадия компиляции. Эта стадия включает по существу две последние фазы процесса распределенной компиляции запросов. Т.е. оптимизацию и генерацию машинных кодов. На этой стадии производится проверка прав пользователя от имени которого выполняется компиляция на выполнение соответствующих действий. Происходит обработка представлений базы данных. Осуществляется локальная оптимизация обрабатываемой части запроса в соответствии с имеющимися индексами и производится генерация кода.

**Нормализация и ее необходимость.**

При проектировании структуры базы данных заказчик часто предоставляет разработчику описание используемых форм и бланков существующих в бумажном виде. При простом переносе полей бумажных форм таблицы базы данных неизбежно возникает ряд проблем связанных с добавлением, изменением, удалением и выборкой данных. Нормализация таблиц – формальный аппарат ограничений на формирование таблиц описывающий разбиение таблиц на две или более частей и обеспечивающий создание лучших методов добавления, удаления и изменения данных. Нормализация позволяет устранить дублирование, обеспечивает непротиворечивость хранимых данных и уменьшает трудозатраты на ввод и изменение данных. Окончательной целью нормализации является получение такого проекта базы данных в котором любая часть информации хранится только в одном месте, т.е. исключается избыточность информации. Это делается не столько с целью экономии места, ненормализованные таблицы иногда занимают больше места чем нормализованные, сколько для исключения противоречий в хранимых данных. Таблица считается нормализованной на определенном уровне когда она удовлетворяет условиям, накладываемыми соответствующей нормальной формой. Процесс нормализации представляет собой последовательные изменения структуры таблиц до тех пор пока они не будут удовлетворять требованиям соответствующей нормальной формы. При приведении к последней, 5-й нормальной форме таблица считается полностью нормализованной использование последних трех нормальных форм оправдано только в том случае, если таблица имеет несколько десятков колонок поскольку полная нормализация таблиц достаточно трудоемкий процесс. Про описании нормальных форм используются следующие *понятия*:

1. *Функциональной зависимостью* между полями a и b называется зависимость при которой каждому значению поля a в любой момент времени соответствует единственное значение поля b из всех возможных.

2. *Полной функциональной зависимостью* между составным полем a и полем b называется зависимость при которой поле b зависит функционально от поля a и не зависит функционально от любого подмножества поля a.

3. *Транзитивная функциональная зависимость* между полями a и c имеет место в том случае, если поле b функционально зависит от поля a, а поле c функционально зависит от поля b. При этом функциональной зависимости поля a от поля c существовать не должно.

4. Несколько полей *взаимно независимы* если ни одно из них не является функционально зависимым от другого.

Таблица находится в первой нормальной форме тогда и только тогда когда ни одно поле строки не содержит более одного значения и любое ключевое поле не пусто. Можно сказать что таблица находится в первой нормальной форме если ни один из ее элементов сам не является таблицей. Т.е. нет полей хранящих более одного значения. Таблица в первой нормальной форме называется универсальной таблицей в проектируемой базе данных. В нее можно включить все необходимые колонки и использовать для хранения всю информацию которую предполагается внести в базу данных. Таблица находящаяся в первой нормальной форме обладает следующими недостатками:

1. Добавление новых данных требует ввода значений для всех колонок даже если в таблице уже существуют необходимые данные. Рано или поздно при вводе дублирующихся данных может возникнуть ошибка, что приведет к появлению двух различных значений вместо одного. Кроме того нельзя добавить информацию раньше чем она понадобиться.

2. При изменении необходимо просматривать все строки в которых имеется данное значение и изменять его во всех строках где оно встречается. При удалении одной строки может быть потеряна информация которая в дальнейшем может понадобиться при вводе новых значений в таблицу.

Таблица находится во *второй нормальной форме*, если она удовлетворяет требованиям первой нормальной формы, и все ее поля не входящие в первичный ключ связаны полной функциональной зависимостью с первичным ключем. Если таблица имеет простой первичный ключ, состоящий только из одной колонки, то она автоматически находится во второй нормальной форме. Если первичный ключ составной то таблица необязательно находится во второй нормальной форме и тогда необходимо разделить ее на две или более таблиц таким образом, чтобы первичный ключ однозначно идентифицировал значения в любой колонке. Если в таблице имеется хотя бы одно поле независящее от первичного ключа то в первичный ключ необходимо включить дополнительные колонки. А если таких колонок нет, необходимо добавить новую колонку. Одну и ту же таблицу можно привести ко второй нормальной форме различными путями. Простое разбиение таблицы на несколько более мелких таблиц может привести к потере логической целостности данных. Необходимо предусмотреть создание дополнительных колонок которые могут быть использованы в качестве первичного ключа в каждой из таблиц появившихся в результате нормализации.

Таблица находится в третьей нормальной форме если она удовлетворяет определению второй нормальной формы и ни одно из ее не ключевых полей не зависит функционально от любого другого не ключевого поля. Таблица находится в *третьей нормальной форме* если она находится во второй нормальной форме и каждое не ключевое поле не транзитивно зависит от первичного ключа. Требование третьей нормальной формы сводится к тому, чтобы все не ключевые поля зависели только от первичного ключа и не зависели друг от друга.

Таблица находится в нормальной форме Б. Кодда только в том случае если любая функциональная зависимость между ее полями сводится к полной функциональной зависимости от возможного ключа.

*Полной декомпозицией таблицы* называют такую совокупность произвольного числа ее проекций соединения которых полностью совпадает с содержимым таблицы. Проекция таблицы получается при использовании вертикального фильтра.

Проекция – это копия таблицы в которую не включены одна или несколько колонок исходной таблицы.

Четвертая нормальная форма является частным случаем пятой нормальной формы, когда полная декомпозиция должна быть соединением ровно двух проекций. Таблица находится в пятой нормальной форме тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ.

**Язык реляционных баз данных SQL (Structured Query Language).**

**История развития SQL.**

Язык SQL был разработан в 70-х годах прошлого века корпорацией IBM как язык управления реляционными базами данных. До него и после него были попытки создания альтернативных языков, но стандартом стал именно SQL. Практически все производители промышленных СУБД используют и развивают в своих программных продуктах ту или иную модификацию SQL. С 70-х годов было разработано много версий языка часто несовместимых друг с другом. В результате в 1992 г. Американским институтом стандарта был разработан стандарт SQL92 описывающий поведение SQL ориентированных серверов баз данных и регламентирующий основные правила работы с ними. Целью разработки этого стандарта было уменьшение несовместимости различных версий языка SQL. Однако несмотря на все усилия и попытки добиться единого стандарта каждый из производителей использует и развивает свою модификацию SQL.

…

**Типы данных в … SQL.**

Тип данных определяет диапазон значений которые можно сохранить в переменной или колонке таблицы.

В Transact-SQL выделяют следующие группы типов данных:

1. Числовые целые типы данных (Int, SmallInt, TinyInt, Long). Целочисленные типы данных часто используются при описании полей по которым создаются индексы и первичные ключи.

2. Нецелочисленные типы данных (float, double, decimal). Данные типа decimal хранятся в SQL-Server не в виде чисел а в идее последовательности цифр, для представления каждой десятичной цифры используется 4 бита.

3. Денежные типы данных обеспечивают хранение до 4-х знаков после запятой, к ним относятся Money, SmallMoney.

4. Типы данных для хранения информации о времени (DateTime – 8 байт , первые 4 – дата, хранимое значение – смещение относительно базовой даты в качестве которой выбрано ***1 января 1753 года***, вторые 4 – информация о количестве миллисекунд, прошедших после полудня данного дня. Диапазон данных которые хранит тип DateTime позволяет хранить значения по дате до 31 декабря 9999; SmallDateTime).

Формат даты в SQL-Server год-месяц-день.

5. Бинарные типы данных:

binary(n) – позволяет хранить до 8000 байт, значение n – определяет количество байт которое будет храниться и +4 байта для хранения длины поля.

varbinary(n) – идентичен типу binary с тем отличием, что в таблице для хранения данных этого типа будет выделяться ровно столько байт, сколько необходимо для хранения значения которое ввел пользователь.

image – позволяет преодолевать ограничение длины 8000 байт. При использовании типа image возможно хранение данный длиной до 231-1. SQL-Server первоначально выделяет одну страницу памяти, а затем по мере необходимости выделяются дополнительные страницы, которые связываются в цепочку.

6. Символьные типы данных:

char(n) – этот тип данных разрешает хранение набора символов длиной до 8000 знаков. Аргумент n определяет максимальное количество символов которое может быть введено. В таблице для хранения данных типа char выделяется заданное количество байт независимо от того сколько символов реально введено. Если пользователь сохраняет строку менее объявленной n, то SQL-Server добавляет в конец строки пробелы. Если строка превышает объявленную длину, то она усекается и часть конечных символов будет потеряна.

varchar(n) – этот тип подобен типу char с тем отличием что количество выделяемого в таблице пространства зависит от реального числа байт которое занимает значение.

Тип Nchar(n) подобен типу char с тем отличием, что тип Nchar используется для хранения символов в формате Unicode. Каждый символ в формате Unicode занимает 2 байта, поэтому максимальное количество символов которое может хранить поле этого типа равно 4000.

Nvarchar(n) – аналогичен типу varchar, только символы в формате Unicode.

7. Текстовые типы данных – позволяют хранить до 2 гб символов. К ним относятся:

text – обеспечивает хранение текста длиной до 231-1 символов, при работе с данными данного типа сервер первоначально выделяет одну страницу памяти, а затем, при необходимости выделяются дополнительные страницы которые связываются в цепочку.

ntext – используется при работе с большими объемами текста в формате unicode.

Работа с текстовыми типами данных отличается от работы с символьными типами данных использованием набора специализированных функций для обработки текста.

8. Специальные типы данных.

Bit - Тип данных принимающий значения 0 или 1. Используется для хранения данных которые могут принимать только два значения.

TimeStamp – используется в SQL-Server в качестве индикатора версий изменения строки в пределах базы данных. С помощью значений полей этого типа данных можно отслеживать последовательность изменений строк как в пределах таблицы, так и в пределах базы данных. Если в таблице имеется колонка типа TimeStamp, то сервер автоматически будет выполнять обновление значений в этой колонке при изменении или добавлении строки в таблицу. Значение типа TimeStamp уникально в пределах базы данных и может быть успешно использовано для идентификации строки. Для каждой базы данных SQL-Server ведет отдельный счетчик значений типа TimeStamp. При внесении любых изменений в таблицу, в которой имеется колонка типа TimeStamp значение счетчика увеличивается на 1 после чего новое значение записывается в таблицу. Тип TimeStamp может иметь только одна колонка в таблице.

UniqueIdentifier – этот тип данных используется для хранения глобальных уникальных идентификационных номеров. Значение занимает 16 байт. В SQL-Server имеет функция NewID() с помощью которой получается уникальное значение UniqueIdentifier, полученное значение может быть использовано для уникальной идентификации строки. В отличие от колонки типа TimeStamp значение в колонке типа UniqueIdentifier автоматически не изменяется.

SysName – этот тип данных используется для описания имен объектов базы данных таких как таблицы, колонки, хранимые процедуры и т.д..

**Управляющие конструкции Transact-SQL.**

1. Begin … End

С помощью этой конструкции выполняется группировка двух и более команд в единый блок. Сгруппированные команды воспринимаются интерпретатором Transact-SQL как один оператор. Подобная группировка требуется для условных и циклических конструкций. Блоки Begin…End могут быть вложенными. SQL-Server фактически не ограничивает глубину вложенности блоков Begin…End. Однако на практике уровень вложенности редко превышает 5 блоков. Для обеспечения хорошего восприятия структуры программы вложенные блоки принято выделять добавляя в начале строки символы табуляции и пробела. Некоторые команды Transact-SQL не должны выполняться вместе с другими командами, поэтому их включение в конструкции Begin…End совместно с другими командами не допускается. К таким командам относятся команды резервного копирования, создания, изменения, удаления объектов базы данных.

2. If … Else

If < boolean\_expression >

{sql\_statement | statement block}

[Else]

{sql\_statement | statement block}

Аргумент boolean\_expression определяет логическое условие которое возвращает значение true если условие выполняется, false если условие не выполняется.

3. Case … End

CASE input\_expression

WHEN {when\_expression | Boolean\_expression} THEN result\_expression

[..n]

[ ELSE else\_result\_expression]

END

Конструкция case…end возвращает результат который можно использовать в качестве переменной в других выражениях, при этом он рассматривается как функция. Аргумент Input\_expression определяет входное значение. В качестве Input\_expression можно указать имя переменной или использовать функцию. Аргумент when\_expression определяет один из вариантов значения аргумента input\_expression. Если значение input\_expression и when\_expression совпадают то конструкция case…end возвращает значение result\_expression. Если значение входного параметра не найдено ни в одной строке when … then будет возвращено значение указанное после else.

4. While…break&continue

WHILE Boolean\_expression

{sql\_statement | statement\_block}

[BREAK]

{sql\_statement | statement\_block}

[CONTINUE]

Аргумент Boolean\_expression задает логическое условие при истинности которого будет выполняться следующая за ним команда или блок команд {sql\_statement | statement\_block}

 заключенное в конструкцию begin…end. После выполнения команд опять осуществляется проверка boolean\_expression. Цикл повторяется до тех пор пока логическое условие не возвращает значение false. Цикл можно принудительно остановить, если в его теле выполнить команду BREAK. Если нужно начать цикл заново, не дожидаясь выполнения всех команд в теле цикла необходимо выполнить команду CONTINUE.

**Создание, модификация и удаление таблиц.**

Процесс создания таблицы начинается с проектирования ее будущей структуры. В процессе проектирования необходимо решить следующие вопросы:

1. Для хранения каких данных предназначена создаваемая таблица.

2. Какие колонки могут содержать пустые значения.

3. На какие колонки будут наложены проверочные ограничения и для каких колонок будут определены значения по умолчанию.

4. Какие колонки будут использоваться в качестве первичного и внешних ключей.

Проектируя таблицу следует решить каким образом будет обеспечиваться целостность данных в ней. Для этого следует определить ограничение для значения колонок в таблице в целом.

SQL-Server позволяет управлять значениями колонок при помощи следующих механизмов:

1. Определение первичного ключа (Primary key).

2. Определение внешнего ключа (Foreign key).

3. Определение уникальных колонок (Unique).

4. Наложение проверочных ограничений на значения колонок (Check).

5. Определение значения по умолчанию (Default)

6. Определение возможности принимать неопределенные значения (NULL).

**Определение идентификационной колонки.**

При проектировании таблицы часто возникает необходимость в колонке, которая будет содержать последовательность числовых значений. Как правило эта колонка используется для хранения идентификационных номеров объектов. Эти номера получаются путем увеличения значения полученного для предыдущей строки на заданное приращение. В подобной ситуации можно поручить заполнение колонки непосредственно SQL-Server. Для этого при создании таблицы необходимо определить свойство identity и задать начальное значение и приращение. По умолчанию начальное значение и приращение равны 1. В таблице можно определить только одну идентификационную колонку, которую обычно используют для идентификации строк.

**Создание таблиц средствами Transact-SQL.**

Чтобы создать таблицу необходимо прежде всего определить ее имя. Имя таблицы можно указать по одному из трех сценариев. В самом простом случае, когда таблица создается в текущей базе данных достаточно просто указать имя таблицы, при условии что таблица создается адиминистратором или владельцем базы данных. Если таблица создается другим пользователем, необходимо указывать перед именем таблицы имя пользователя. Если таблица создается не в текущей базе данных необходимо указывать имя базы данных.

Синтаксис создания таблицы:

CREATE TABLE [ data base [owner] | owner] name

({<column\_definition> | <column\_name AS computerd\_col\_exspr | <table\_contraint>})

[ON {filegroup | DEFAULT}]

Ключевое слово ON позволяет указать файловую группу в которой будет располагаться таблица. Можно явно указать имя файловой группы либо указать ключевое слово ON которое … расположив таблицу в файловой группе по умолчанию.

Определение каждой колонки таблицы в синтаксисе команды CREATE TABLE обозначенным как column\_definition имеет следующий формат:

column\_definition

{column\_name data\_type}

[DEFAULT constant | [IDENTITY [(seed, increment) [NOT FOR REPLICATION]]]

[ROWGUIDCOL] [<column\_contraint>]

Подобным образом необходимо описать каждую колонку таблицы. Прежде всего следует определить имя колонки column\_name а также тип хранимых в ней данных. При описании колонки могут быть использованы следующие ключевые слова:

1. DEFAULT определяет значение по умолчанию constant которое будет использовано если при вводе строки явно не указано другое значение.

2. IDENTITY предписывает системе осуществлять заполнение колонки автоматически. При использовании этого ключевого слова необходимо также указать начальное значение seed, и приращение increment.

3. В случае когда указано NOT FOR REPLICATION эта колонка не будет автоматически заполняться для строк, вставляемых в таблицу в процессе репликации, поэтому эти строки сохранят свои значения.

Группа ROWGUIDCOL. Использование данного ключевого слова указывает что колонка будет использоваться для хранения глобального уникального идентификационного номера. Кроме того, для колонки можно определить ограничение назначения column\_contraint.

column\_contrait

[NULL | NOT NULL] | [{PRIMARY KEY | UNIQUE } [CLUSTERED | NONCLUSTERED]

[WITH FILLFACTOR = fill factor]

[ON {file | DEFAULT} | [[FOREIGN] REFEREBCES ref\_table]

Для каждой колонки рекомендуется указывать может ли она содержать пустые значения. Primary key определяет колонку как первичный ключ таблицы. В качестве альтернативы можно определить колонку как уникальную воспользовавшись ключевым словом unique. При необходимости можно также указать будет ли индекс создаваемый для данного ограничения кластерным, если указано ключевое слово CLUSTER, или не кластерным, если указано NONCLUSTER. При создании индекса необходимо указать степень заполнения его страниц с помощью FILL FACTOR. FOREIGN KEY определяет колонку как внешний ключ таблицы. Одновременно используя ключевое слово REFERENCES необходимо указать имя таблицы с которой будет связана создаваемая таблица и имя колонки, которая будет связана с данной колонкой. Ограничение на значение колонок можно определять на уровне всей таблицы.

[CONTRAINT contrait\_name] {[{PRIMARY KEY | UNIQUE} [CLUSTERED | NON CLUSTERED]

{{ COLUMN […n]}} [ON FILEGROUP [DEFAULT}]] | FOREIGN KEY [{column […n]}]

REFERENCES ref\_table [{ref\_column […n]}] [NOT FOR REPLICATION] | CHECK {search\_conditions}}

Ограничение на уровне таблицы начинается с ключевого слова CONTRAINT после которого необходимо указать имя ограничения. CHECK накладывает проверочное ограничение которое будет использоваться системой для осуществления контроля вводимыми данными.

**Изменение структуры таблицы средствами Transact-SQL.**

Для изменения структуры таблицы средствами Transact-SQL используется команда ALTER TABLE.

ALTER TABLE table

{ALTER TABLE COLUMN COLUMN\_NAME {NEW\_DATA\_TYPE [{precision [, scale]}]

[NULL | NOT NULL] | {ADD | DROP} ROWGUIDCOL }]

добавление/удаление столбца

[ADD {[<column\_definition>] column\_name AS compuled\_column\_expression}[..n]{WITH CHECK | WITH NOCHECK] ADD {<table\_contrait>}[..n]

|DROP [CONSTRAINT] contrait\_name | COLUMN column} [..n] { CHECK | NOCHECK} CONSTRAIT {ALL | constraint\_name [..n]} {ENABLE | DISABLE} TRIGGER {ALL | trigger\_name[…n]]}}

 Изменение определения колонки осуществляется используя ключевое слово ALTER COLUMN после чего определяется имя изменяемой колонки. Можно изменять тип данных в колонке, размерность и точность, указывать разрешено ли колонке содержать неопределенные значения NULL. В случае … обязательно нужно указать тип данных для колонки, даже если он не изменяется. При определении для колонки свойства NOT NULL необходимо предварительно позаботиться о том, чтобы на момент изменения колонка не содержала ни одного значения NULL. Для определения новой колонки команде ALTER TABLE необходимо указать ключевое слово ADD. За ним следует описание колонки которая имеет такой же формат как и при создании колонки с помощью CREATE TABLE. При удалении колонки из таблицы с помощью команды ALTER TABLE используется ключевое слово DROP. При этом можно удалить как саму колонку, так и ограничения на значения в колонке. При этом нельзя обнулять следующие колонки:

1. Колонки задействованные в индексе.

2. Колонки полученные в результате репликации.

3. Колонки для которых определены ограничения на значения.

4. Колонки для которых определены значения по умолчанию.

Ограничение конкретного … DROP CONTRAINT означает что при вводе новых строк данные не будут проверяться на соответствие этому ограничению. При помощи команды ALTER TABLE можно также управлять состоянием триггеров определенных для данной таблицы. При этом в процессе изменения данных в таблице те действия, которые определены в триггерах как реакция на вставку, изменение или удаление данных не выполняются, хотя сами триггеры продолжают существовать. Чтобы активировать триггер необходимо использовать команду с ключевым словом ENABLE TRIGGER.

Для удаления таблицы используется команда DROP TABLE. Перед удалением необходимо убедиться что удаляемая таблица не связана с другими таблицами.

**Управление данными.**

**Добавление данных.**

Данные в таблице SQL-Server могут быть внесены следующими способами:

1. С помощью команды INSERT.

2. С помощью программного интерфейса DatabaseAPI который предполагает использование технологий ADO, OLE DB, ADO .NET, ODBC и др.

3. С помощью команды SELECT INTO, когда результат выполнения запроса копируется в таблицу.

**Использование команды INSERT.**

Синтаксис команды INSERT

INSERT [INTO] {имя\_таблицы | имя\_представления} { [ {список\_столбцов}] { VALUES {[DEFAULT] NULL | выражение }[…n]} | временная\_таблица ] инструкция\_выполнения }} | DEFAULT VALUES

Ключевое слово INTO может быть использовано в команде INSERT для обозначения того, что следующим параметром является имя таблицы в которую будут вставлены данные.

имя\_таблицы – имя таблицы в которую необходимо вставить данные

VALUES – ключевое слово которое определяет набор данных которые будут вставлены в таблицу.

Количество аргументов VALUES определяется количеством столбцов в таблице или списком столбцов в таблице. Для каждой колонки таблицы можно указать один из следующих вариантов: DEFAULT – будет вставлено значение по умолчанию, определенное для колонки.

NULL – будет вставлено значение NULL.

выражение – в явном виде задает значение которое вставлено будет в колонку таблицы.

DEFAULT VALUES – при указании этого параметра строка будет содержать только значения по умолчанию. Если для колонки не определено значение по умолчанию, но разрешено хранение значений NULL, то будут вставления значения NULL. Если колонка определена с использование ключевого слова IDENTITY, то в нее также будут вставлены соответствующие значения. Если для колонки не разрешено хранение значений NULL, не определено значение по умолчанию, колонка не определена как IDENTITY и не указано значение для вставки Server выдаст сообщение об ошибке и прервет выполнение команды INSERT. То же самое произойдет в случае неправильного указания типа данных.

**Изменение данных.**

Для изменения данных в таблице можно использовать различные методы включая удаление и повторную вставку строк. Чаще всего для изменения данных в таблицах средствами Transact-SQL используется команда UPDATE позволяющая выполнить как простое обновление данных в колонке, так и в строке. Если при обновлении строк необходимо реализовать сложный алгоритм, то можно создать хранимые процедуры, которые будут содержать всю логику обновления. В этом случае пользователю не придется всякий раз когда необходимо обновление восстанавливать заново сложные фрагменты кода.

Синтаксис команды UPDATE

UPDATE ( имя\_таблицы | имя\_представления) SET (имя\_столбца = {выражение | DEFAULT | NULL | @переменная-столбец = выражение}) ( [FROM (исходная\_таблица) [..n]] [WHERE условие\_поиска])

Начиная с ключевого слова SET начинается блок в котором определяется список столбцов присваемых им значений. Команда UPDATE позволяет изменять не только значения в колонках таблицы но и значения переменных. Для каждой колонки нужно определить значение которое она примет после выполнения изменений. С помощью ключевого слова DEFAULT можно присвоить колонке значения определенные для нее по умолчанию можно также установить неопределенное значение колонки NULL. Если необходимо установить в колонке определенное значение следует указать константу, переменную или выражение. Изменению подвергнуться все строки удовлетворяющие критерию ограничения области действия команды UPDATE, которые задаются с помощью WHERE. При составлении выражений можно ссылаться на любые колонки таблицы, включая изменяемые.

**Удаление записей из таблиц.**

Удаление данных из таблиц выполняется построчно. За одну операцию можно удалить как одну строку так и несколько тысяч строк. Если необходимо удалить из таблицы все данные В некоторых случаях можно удалить и саму таблицу. Для удаления данных средствами Transact-SQL используется команда DELETE

DELETE [FROM] (имя\_таблицы | имя\_представления) [FROM исходная таблица] [ WHERE условие поиска]

При работе с командой DELETE необходимо определить диапазон строк которые будут удалены. Для определения этого диапазона используется WHERE.

**Извлечение данных из таблиц.**

В большинстве случаев для выборки данных из таблиц используется упрощенный синтаксис команды SELECT.

SELECT select\_list [INTO new\_table] FROM table\_sourse [WHERE search\_conditions] [GROUP BY group\_by\_expression] [HAVING search\_condition] [ ORDER BY order\_expression [ASC | DESC]]

Раздел SELECT

SELECT [ ALL | DISTINCT ] TOP n[PERCENT] <select\_list>

ALL – при указании этого ключевого слова результат разрешается включением дублирующихся строк

DISTINCT – запрещает появление в результате дублирующихся строк

TOP n – предписывает серверу выбирать не все строки, а только n первых.

PERCENT – позволяет выбирать нефиксированное количество строк, а процент от всех строк.

Раздел INTO предназначен для сохранения результата выполнения запроса в заданной таблице. Аргумент new\_table определяет имя таблицы в которой будет сохраненный результат.

Раздел FROM

Аргумент table\_sourse имеет следующую структуру

<table\_sourse> :: = table\_name [[AS] table\_alias] | view\_name {{ AS] table\_alias] | <join\_table>

Аргумент table\_name должен содержать имя таблицы в которой осуществляется выборка данных.

Аргумент view\_name указывает имя представления из которого необходимо выбрать данные.

Синтаксис команды SELECT позволяет изменять имена таблиц, представлений и колонок, которые получены в результате выборки. Для этого используются псевдонимы. Аргумент table\_alias указывает псевдоним таблицы из которой необходимо выбрать колонки.

Конструкция join\_table имеет следующий синтаксис:

<join table> :: = <table\_source> <join\_type> <table\_source> ON <search\_condition>

Эта конструкция используется для связывания при выборке из нескольких таблиц. Конструкции <table\_source> описывают связываемые таблицы. Конструкция <join\_type> описывает типа связывания двух таблиц. Исходная таблица указывается слева от конструкции <join\_type> и называет левой таблицей, справа указывается зависимая таблица, которая называется правой таблицей.

Конструкция <join\_type> имеет следующий синтаксис:

<join\_type> :: = [ INNER | (( LEFT | RIGHT | FULL ) [OUTER] ) ]

INNER - При использовании этого типа связи выбираются пары строк для которых имеются строки удовлетворяющие критерию связывания в обеих таблицах. Строки из левой и правой таблиц для которых имеются пары связанной таблицы в результате исключаются.

LEFT – в результат будут включены все строки левой таблицы, независимо от того, есть ли для них соответствующая строка в правой таблице или нет. Для соответствующих колонок правой таблицы включенных в результат устанавливается значение NULL.

RIGHT – при использовании этого ключевого слова в результат будут включены все строки правой таблицы независимо от того, есть ли для них соответствующая строка в левой таблице.

FULL – в результат будут включены все строки как правой, так и левой таблицы. Применение ключевого слова FULL можно рассматривать как одновременное использование ключевых слов LEFT и RIGHT.

ON <search\_condition> - логическое условие, определяющее условие связывания двух таблиц. В этом условии используются операторы сравнения >,<,=.

С помощью конструкции WHERE можно сузить количество обрабатываемых строк данных, определив одно или несколько логических условий. В результат будут включены только те строки которые соответствуют заданным условиям. Условие может включать константы, переменные и любые выражения возвращающие логические значения TRUE или FALSE. Можно также указать несколько условий, объединив их с помощью логический операторов OR, AND, NOT. Логическое условие может быть произвольным в том числе и не связанным с данными.

Раздел GROUP BY позволяет выполнить группировку строк таблицы по определенным критериям. Для каждой группы можно выполнить специальные функции агрегирования которые будут применены.

GROUP BY [ALL | group\_by\_expression […n]]

HAVING – этот раздел в основном используется для указания условий поиска при выполнении группировки данных с помощью GROUP BY. Конструкция search\_condiotion в разделе HEAVING содержит логическое условие определяющее диапазон строк обрабатываемых запросом.

Раздел ORDER BY используется когда необходимо отсортировать данные в результирующем наборе. Аргумент order\_expression содержит имя колонок включенных в запрос. Можно использовать для сортировки колонку не включенную в результат выполнения оператора SELECT. При указании ключевого слова ASC данные будут отсортированы по возрастанию. Если необходимо отсортировать данные по убыванию нужно использовать ключевое слово DESC. По умолчанию используется сортировка по возрастанию. Данные можно отсортировать по нескольким колонкам. Для этого необходимо ввести имена колонок через запятую с указанием порядка сортировки.

Непосредственному написанию кода хранимой процедуры предшествуют следующие этапы:

1. Определение типа создаваемой хранимой процедуры.

Помимо тех хранимых процедур, которые пользователь создает в собственной базе данных существует возможность создавать системные и временные хранимые процедуры. С точки зрения написания кода эти хранимые процедуры мало отличаются друг от друга. Различие заключается в способе хранения этих процедур. Чтобы создать системную хранимую процедуру необходимо дать ей имя начинающееся с префикса sp\_ и сохранить эту хранимую процедуру в системной базе данных master. Временные хранимые процедуры доступны пользователям в течении времени пока активно соединение в контексте которого эта хранимая процедура создана. SQL-Server позволяет создавать как локальные, так и глобальные временные хранимые процедуры, которые хранятся в базе данных tempdb. Локальные временные хранимые процедуры доступны только пользователю, работающему в контексте соединения создавшего данную хранимую процедуру. При определении хранимой процедуры такого типа необходимо дать ей имя начинающееся с символа #. Глобальные временные хранимые процедуры доступны пользователям из любых соединений данного сервера. Имя глобальной временной хранимой процедуры начинается с ##.

2. Определение входных и выходных параметров хранимой процедуры.

Подобно процедурам в большинстве языков программирования в SQL-Server хранимые процедуры могут иметь входные и выходные параметры. Полученные процедурой параметры могут использоваться как обычные переменные Transact-SQL участвуя в любых операциях наравне с другими переменными. По окончании выполнения кода хранимой процедуры можно передать в программу или в другие хранимые процедуры некоторые результаты используя выходные параметры. Может сложиться ситуация что выполнение хранимой процедуры будет невозможно по каким-либо причинам. В связи с этим при определении хранимой процедуры необходимо продумать каким образом процедура будет информировать вызвавшее ее приложение об успешности исполнения ее кода.

3. Разработка кода хранимой процедуры.

Код хранимой процедуры может содержать последовательность любых команд Transact-SQL, включая вызов других хранимых процедур. Код должен реализовывать логически законченное действие. Плохо написанная хранимая процедура может существенно снизить эффективность работы всей системы, особенно если ее будут использовать многие пользователи.

4. Включение созданной хранимой процедуры в группу.

Для удобства управления хранимыми процедурами можно группировать логически однотипные процедуры. Для этого всем процедурам группы при создании дают одинаковые имена но присваивают различные идентификационные номера. Ссылка на группу осуществляется по линии процедур. Достоинства такого подхода заключается в том, что можно одновременно управлять всей группой хранимых процедур. Например, можно удалить все хранимые процедуры входящие в группы.

**Создание хранимых процедур средствами Transact-SQL.**

Определение хранимой процедуры заключается в выполнении команды CREATE PROC имеющей следующий синтаксис.

CREATE PROC[EDURE] procedure\_name [number] [ (@parameter data\_type) {VARYING] [= default] [ OUTPUT]] […n] [WITH (RECOMPILE | ENCRYPTION | RECOMPILE ENCRYPTION) ] [ FOR REPLICATION] AS sql\_statement […n]

….

CREATE VIEW view\_name[(column [..n])] [WITH ENCRYPTION] AS select\_statement [WITH CHECK OPTION]

….

Фактор заполнения определяет плотность записи данных на странице. Его значение определяет какой процент доступного пространства индексных страниц будет заполнен данными. Оставшееся пространство будет постепенно заполняться по мере записи данных в таблицу. Использование фактора заполнения позволяет более гибко управлять поведением сервера при индексировании таблицы. Чем более высокий фактор заполнения задан при построении индекса тем меньше свободного места останется на странице и тем более компактно будет размещена информация об индексе. При выборе фактора заполнения необходимо оценивать насколько интенсивно будет выполняться добавление и изменение данных в таблице. Если таблица в основном используется только для чтения то лучше установить фактор заполнения близкий к 100%. Это позволит использовать пространство базы данных более экономно. Если же данные в таблице часто изменяются следует установить небольшой фактор заполнения как для самих данных так и для индексов. Пространство базы данных в этом случае будет расходоваться неэкономно, зато производительность выполнения операций изменения и добавления данных будет максимальной. Если интенсивно изменяемые таблицы установить высокий фактор заполнения то сервер будет вынужден часто выполнять расчипление страниц при вставке новых данных.