# Содержание

Введение. Понятие информации и информационной системы. Требования к организации данных

Глава 1. Базы данных

1.1 Модели баз данных

1.1.1 Реляционная модель

1.1.2 Иерархическая модель

1.1.3 Сетевая модель

1.1.4 Объектно-ориентированная модель данных

1.2 Теория нормальных форм

1.3 Достоверность и безопасность информации

Глава 2. Основы разработки базы данных

2.1 Методы проектирования БД

2.1.1 Метод декомпозиции

2.1.2 Метод синтеза

2.1.3 Метод объектной связи

2.2 Организация СУБД

2.2.1 Требования к современной СУБД

2.2.2 Архитектура СУБД

2.2.3 Работа СУБД

2.3 Организация данных

2.3.1 Физическая организация данных

2.3.2 Организация индексных таблиц

2.4 Обновление и восстановление данных

2.4.1 Типы ключевых полей

2.4.2 Создание и изменение ключевых полей

2.5 БД в сетях

2.6 Доступ к данным в Windows

Глава 3. Работа с таблицами базы данных на примере СУБД Microsoft Access

3.1 Структура таблицы, ее создание

3.1.1 Создание новой пустой таблицы

3.1.2 Создание таблицы в режиме конструктора

3.2 Ключи и индексы

3.2.1 Типы ключевых полей

3.2.2 Индексы

3.2.3 Создание и изменение ключевых полей

3.3 Общая картина ограничений и поддержания целостности данных

3.3.1 Ограничения в базе данных

3.3.2 Типы ограничений в базе данных

3.3.3 Поддержание целостности данных

Заключение

# Введение. Понятие информации и информационной системы. Требования к организации данных

Веками человечество накапливало знания, навыки работы, сведения об окружающем мире, другими словами – собирало информацию. Вначале информация передавалась из поколения в поколение в виде преданий и устных рассказов. Возникновение и развитие книжного дела позволило передавать и хранить информацию в более надежном письменном виде. Открытия в области электричества привели к появлению телеграфа, телефона, радио, телевидения – средств, позволяющих оперативно передавать и накапливать информацию. Развитие прогресса обусловило резкий рост информации, в связи, с чем вопрос о ее сохранении и переработке становился год от года острее. С появлением вычислительной техники значительно упростились способы хранения, а главное, обработки информации. Развитие вычислительной техники на базе микропроцессоров приводит к совершенствованию компьютеров и программного обеспечения. Появляются программы, способные обработать большие потоки информации. С помощью таких программ создаются информационные системы. Целью любой информационной системы является обработка данных об объектах и явлениях реального мира и предоставление человеку нужной информации о них. Если мы рассмотрим совокупность некоторых объектов, то сможем выделить объекты, обладающие одинаковыми свойствами. Такие объекты выделяют в отдельные классы. Внутри выделенного класса объекты можно упорядочивать как по общим правилам классифицирования, например по алфавиту, так и по некоторым конкретным общим признакам, например по цвету или материалу. Группировка объектов по определенным признакам значительно облегчает поиск и отбор информации.

Информационные системы (ИС) можно условно разделить на фактографические и документальные.

В фактографических ИС регистрируются факты – конкретные значения данных (атрибутов) об объектах реального мира. Основная идея таких систем заключается в том, что все сведения об объектах (фамилии людей и названия предметов, числа, даты) сообщаются компьютеру в каком-то заранее обусловленном формате (например дата – в виде комбинации ДД.ММ.ГГГГ). Информация, с которой работает фактографическая ИС, имеет четкую структуру, позволяющую машине отличать одно данное от другого, например фамилию от должности человека, дату рождения от роста и т.п. Поэтому фактографическая система способна давать однозначные ответы на поставленные вопросы.

Документальные ИС обслуживают принципиально иной класс задач, которые не предполагают однозначного ответа на поставленный вопрос. Базу данных таких систем образует совокупность неструктурированных текстовых документов (статьи, книги, рефераты и т.д.) и графических объектов, снабженная тем или иным формализованным аппаратом поиска. Цель системы, как правило, - выдать в ответ на запрос пользователя список документов или объектов, в какой-то мере удовлетворяющих сформулированным в запросе условиям.

Указанная классификация ИС в известной мере устарела, так как современные фактографические системы часто работают с неструктурированными блоками информации (текстами, графикой, звуком, видео), снабженными структурированными описателями. Чтобы пояснить, как фактографическая система может превратиться в документальную (и наоборот), рассмотрим условный пример.

Пусть объектом обработки фактографической ИС является некий список ученых-экономистов, причем для каждого ученого имеются следующие данные:

* Имя;
* Дата рождения в формате ДД.ММ.ГГГГ;
* Национальность (русский или иностранец);
* Биография (произвольный текст);
* Названия трудов ученого.

Требования к организации данных информационных систем:

1. Интеграция данных — когда все данные хранятся централизованно, создавая динамически обновляемую модель.
2. Максимальная независимость прикладных программ от данных или обеспечение физической и логической независимости данных.

Выполнение этих требований привело к созданию единого для всех задач блока данных — базы данных и разработки одной управляющей программы для манипулирования данными на физическом уровне — СУБД.\

### БД

#### СУБД

Прикладные программы

Прикладные программы

Физический уровень

Именно СУБД обеспечивает независимость данных, изменение физической организации воспринимается СУБД и не влияет на прикладную программу. С другой стороны, изменение логики программы не требует реорганизации и изменения механизма доступа к физическим данным. Введение СУБД разделяет логическую структуру данных от физической структуры данных. Отличительной чертой современных БД следует считать совместное хранение данных с их описанием. Современный подход требует, чтобы в программе были заданы лишь имена и форматы обрабатываемых данных. Поставляя данные в программу, СУБД их предварительно обрабатывает, в связи с чем изменение организации данных не отражается на прикладных программах, в этом случае меняются только процедуры СУБД. Описание БД называют метаданными.

Располагая структурированными описателями (имя, дата, пол), система может выдать строгие ответы на вопросы: а) о любом ученом персонально; б) о распределении ученых по дате рождения и полу (в любых сочетаниях). Заметим, что те же данные в той или иной форме дублируются в биографии, например: "Уильям Стаффорд родился в 1554 году в семье…", "Иван Тихонович Посошков жил с 1652 по 1726 год…" и т.д. Однако, если удалить из списка структурированные описатели, система превратится в документальную и, если не принять мер, утратит способность находить и классифицировать ученых. В отличие от нас, компьютер не знает, что Стаффорд – иностранец, а Посошков – русский, что "родиться" и "жить с… по…" - синонимы и т.д.

В данной работе рассматриваются фактографические ИС, которые используются буквально во всех сферах человеческой деятельности, а практика работы с ними будет рассмотрена на примере современной системы управления базами данных (СУБД) Microsoft Access.

# Глава 1. Базы данных

Что такое база данных (БД)? В широком смысле слова можно сказать, что БД – это совокупность сведений о конкретных объектах реального мира в какой-либо предметной области. Синоним термина "база данных" – "банк данных".

Чтобы обеспечить быстроту и качество поиска данных в базе, этот процесс должен быть автоматизирован. Компьютерную базу данных можно создать несколькими способами:

* С помощью алгоритмических языков программирования, таких как Basic, Pascal, C++ и т.д. Данный способ применяется для создания уникальных баз данных.
* С помощью прикладной среды, например Visual Basic. С его помощью можно создавать базы данных, требующие каких-то индивидуальных особенностей построения.
* С помощью специальных программных сред, которые называются системами управления базами данных.

В настоящее время существует несколько видов СУБД. Наиболее известными и популярными СУБД являются Access, FoxPro и Paradox.

Каждый объект, сущность обладает набором свойств или атрибутов. Мыслить в терминах конкретных объектов трудно, поэтому прибегают к разбиению всего множества объектов на группы объектов однородных по структуре и поведению, и называемых типами объектов, типами записей (в ООП — класс, объект). При этом предполагается, что все экземпляры объектов одного типа обладают одинаковым наборам атрибутов. Свойства по своей структуре могут быть любой степени сложности. Большинство современных СУБД плохо развито с точки зрения описания сложных свойств: графика, текст.

Свойства могут быть:

во времени:

1. статические;
2. динамические;

по структуре:

1. неделимые (атомарные);
2. составные.

Байт — наименьшая единица адресуемых битов.

Элемент данных — наименьшая единица поименованных данных, называют полем.

Агрегат данных — поименованная совокупность данных внутри записи, рассматриваемая как единое целое. Позволяет в приложениях за одно обращение получить некоторую логически связанную совокупность данных. Может содержать в себе и другие агрегаты данных. Если агрегат состоит из одномерной упорядоченной совокупности элементов данных одного типа, то говорят, что в этом типе записи определен вектор. Вектор может быть фиксированной и переменной длины. Если агрегат встречается несколько раз в экземпляре записи, то агрегат называется повторяющейся группой. В повторяющуюся группу могут входить отдельные элементы данных, вектор, другие агрегаты или повторяющиеся группы. Повторяющиеся группы могут быть фиксированной и переменной длины.

Пример:

Тип объекта (записи) "СОТРУДНИК".

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ФИО | АДРЕС(ул., дом, кв.) | ТЕЛЕФОН | ЗАРПЛАТА(за 3 месяца) | ДИПЛОМ | ДЕТИ(имя, возраст) |
| Жуков И.П. | Победы, 5, 36 | 42-37-05 | 10 540,66 | АО-3628 | Иван, 10 |
| Семенов А.В.  | Водная, 15, 105 | 29-45-99 | 15 530,07 | ВН-5491, КР-1367 | Мария, 5Алексей, 8 |
| Элемент | Агрегат | Элемент | Вектор, фиксированный | Вектор, переменный | Повторяющаяся группа, переменной длины |

Запись — агрегат, который не входит в состав никакого другого агрегата. Основная единица обработки данных.

Каждый экземпляр типа записи должен быть отличимым от других объектов данного типа. С этой целью каждому объекту данного типа назначается идентификатор, позволяющий на них однозначно ссылаться, он уникален. Он называется первичным ключом. В качестве первичного ключа может использоваться атрибут, комбинация атрибутов и даже части атрибутов. На практике используют не уникальные идентификаторы называемые вторичным ключом. К каждому такому множеству относятся объекты, которым соответствует одинаковое значение вторичного ключа.

* 1. Модели баз данных

БД может быть основана на одной модели или на совокупности нескольких моделей. Любую модель данных можно рассматривать как объект, который характеризуется своими свойствами (параметрами), и над ней, как над объектом, можно производить какие-либо действия.

Так как пользователя БД не интересует подробности физического хранения данных, в представлении данных можно выделить два уровня абстракции: логическая и физическая. На логическом уровне описываются данные информационной моделью, выделяют внешнюю и внутреннюю информационные модели, а на физическом – физической моделью. Информационная модель должна отражать предметную область в терминах понятных и привычных пользователю. Обычная информационная модель описывает объекты любой природы в терминах: сущность, атрибуты и связи. Абстрактная модель должна быть описана для представления в ЭВМ. Это описание делается средствами модели данных, которую поддерживает СУБД и называется внутренней или концептуальной информационной схемой. Итак, СУБД поддерживает некоторую модель данных и отображает её в соответствующие структуры физической БД.

Модель Данных (МД) — средство логического представления физических данных. Формализованное описание данных, отражающее их состав, типы данных, а также взаимосвязь между ними.

МД, которую поддерживает СУБД на логическом уровне определяется:

1. допустимой структурой данных, разнообразием и количеством типов данных, которые можно описать с помощью МД;
2. множеством допустимых операций над данными;
3. ограничениями контроля целостности.

В зависимости от поддерживаемой структуры данных, МД подразделяются на:

1. сетевые;
2. иерархические;
3. реляционные.

Большинство БД реляционные.

* + 1. Реляционная модель

Термин "реляционный" (от латинского relation – отношение) указывает, прежде всего, на то, что такая модель хранения данных построена на взаимоотношении составляющих ее частей. В простейшем случае она представляет собой двухмерный массив или двухмерную таблицу, а при создании сложных информационных моделей составит совокупность взаимосвязанных таблиц. Каждая строка такой таблицы называется записью, а столбец – полем.

Реляционная модель данных имеет следующие свойства:

* Каждый элемент таблицы – один элемент данных.
* Все поля в таблице являются однородными, т.е. имеют один тип.
* Каждое поле имеет уникальное имя.
* Одинаковые записи в таблице отсутствуют.
* Порядок записей в таблице может быть произвольным и может характеризоваться количеством полей, типом данных.

Реляционная таблица — множество отношений содержащих всю необходимую информацию о предметной области.

|  |  |  |  |
| --- | --- | --- | --- |
| Неспециалист | Математик | Программист ОД | Проектировщик БД |
| ТаблицаСтрокаСтолбец | ОтношениеКортежАтрибут | Файл данныхЗаписьПоле | ОбъектЭкземпляр данныхАтрибут |

Отношение реляционной БД в зависимости от содержания подразделяется на два класса:

1. объектные;
2. связанные.

Таблица объектов хранит данные об экземплярах объекта, каждая строка содержит ключ и признаки объекта, которые функционально зависят от этого ключа.

Таблица связи хранит ключи двух или более таблиц объектов и таким образом устанавливаются связи между таблицами объектов, каждая строка связи содержит внешние ключи и характеристики связей.

Связь между объектами обычно выражается глаголами: "Служащие работают в отделах". Поэтому название таблице связей часто дают по имени связи или в названии указывают имена объектов участвующих в связи.

Отличительный признак объекта:

—объект может существовать сам по себе, ни с кем не связываясь, связь автономно существовать не может, требуя наличия хотя бы двух объектов.

Пример:

СТУДЕНТ (СТ#, ФИО, Группа, Адрес);

ПРЕДМЕТ (П#, Название, Вид контроля);

ИЗУЧАЕТ (СТ#, П#, Оценка).

Допустимые и недопустимые связи.

Пример:

Институт делится на подразделения, которые могут работать над несколькими проектами, над каждым проектом могут работать сотрудники из нескольких отделов.

Три таблицы:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сотрудники  |  | Отделы  |  | Проекты  |
| С# | ФИО | О# | Название | П# | Название  |
|  |  |  |  |  |  |

Связи:

Сотрудники - Проекты. Сотрудники - Отдел.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сотрудник | Проект |  | Сотрудник | Отдел | Должность |
|  |  |  |  |  |

Для логической завершенности не хватает третьей связи "отдел – проекты", которая отражает какое подразделение над чем работает. Связь содержится в таблице, но косвенно через сотрудников, выделим её явно. С появлением 6-ой таблицы появляется противоречие. Причиной служит то, что один и тот же факт хранится в двух разных местах, косвенно — через исполнителя и явно (С - П) и явно в дополнительной шестой таблице. Вывод: связи между объектами не могут замыкаться. Недоразумение в этом примере вызвано нечеткой формулировкой задачи: отделы как таковые не работают над проектом, они работают над проектом тогда и только тогда, когда над проектом работает хотя бы один сотрудник. Две связи допустимы, а третья запрещена.

СОТРУДНИКИ

ОТДЕЛЫ

ПРОЕКТЫ

связь

Если убрать связь СОТРУДНИКИ – Проекты, то изменится постановка задачи, тогда проекты

СОТРУДНИКИ

ОТДЕЛЫ

ПРОЕКТЫ

закрепляются за отделами, и все сотрудники должны будут работать над проектом.

СОТРУДНИКИ

ОТДЕЛЫ

ПРОЕКТЫ

Сотрудники не закреплены за отделами и свободно переходят в отдел занятый их проектом.

Реляционные модели связей не могут замыкаться, какие связи запрещены, а какие разрешены, зависит от постановки задачи. Для двух объектов возможна одна связь, для трех – две, а для n – (n-1). Для модели из n объектов необходимо (2n-1) объектов.

Работа с БД подразумевает задание и выполнение запросов.

Пример:

Найти всех студентов сдавших на отлично сессию. При этом используются специальные языковые средства, основанные на:

1. реляционной алгебре;
2. реляционном исчислении.

Алгебраический подход требует наличия операндов и совокупности операций над ними. В реляционной алгебре используются в качестве операндов — отношения (таблица).

Операции РА были предложены Коддом. Язык РА — аппликативный язык. Кодд доказал, что запросы сформированные с помощью реляционного исчисления могут сформировать язык РА и наоборот. Результатом любой реляционной операции является некоторое отношение.

Реляционная модель кроме РА может также включать операции реляционного присваивания Target := Source, где левые и правые части — реляционные выражения, представляющие совместимые по типу отношения. Target — базовое отношение.

Операция присваивания дает возможность запоминать значения некоторых алгебраических выражений в БД и изменять состояние БД.

Замечание: язык РА не применяется. Используется язык реляционного исчисления.

Исчисление — это знаковая система, в которой имеется некоторое количество исходных объектов и некоторое количество правил построения объектов из исходных и уже построенных. Дедуктивная система.

Были созданы различные разновидности языков исчисления предикатов, оно называлось реляционным исчислением.

Рассмотрим два языка, широко используемых в СУБД: SQL и QBE.

Язык Структурных Запросов (SQL)

SQL разработан фирмой IBM. Одобрен в качестве стандарта для больших и малых ЭВМ. Если D-Base ориентирована на операции с данными в виде записи, то SQL – на операции с данными, в виде таблиц. Кроме обычных таблиц SQL позволяет создавать особый тип таблиц – выборку (подмножество строк и столбцов из одной или нескольких таблиц). Часто выборку называют виртуальной таблицей.

Язык SQL оперирует понятием БД, которая содержит всю информацию, необходимую для обработки данных: таблицы, выборки, синонимы (альтернативные имена таблицы) и индексы (файлы для быстрого поиска данных, присоединяемые к таблице). В SQL можно выделить следующие средства:

1. средства запросов;
2. средства манипулирования данными;
3. средства определения данных;
4. средства контроля данных;
5. средства встраивания в основной язык.

Средства запросов

Большинство запросов на извлечение данных из БД строится на основе команды SELECT.

SELECT <список атрибутов и/или функций от атрибутов>

FROM <список отношений>

WHERE <условие>

GROUP BY <список колонок>

HAVING <условие>

ORDER BY <список сортировки>

TO FILE | TO PRINTER | TO SCREEN | INTO

Принципиальная схема выполнения запроса:

1. образуется декартово произведение таблиц, перечисленных в FROM;
2. для каждой строки декартового произведения вычисляется значение логического выражения, заданного в WHERE, строки с ложным значением – удаляются;
3. если заданно GROUP BY, то оставшиеся строки делятся на группы соответственно значениям указанных в ней колонок;
4. для каждой группы или строки вычисляется выражение, заданное во фразе SELECT;
5. для каждой группы производится проверка условий заданного фразой HAVING, варианты с false удаляются;
6. результат сортируется по колонкам из ORDER BY, в соответствии с заданным порядком сортировки.

В строке SELECT указываются через запятую имена столбцов в выходной таблице. Символ "\*" означает, что выбираются все столбцы, указанные в предложении FROM. Если в нескольких таблицах имеются колонки с одинаковым именем, то перед этими именами указываются имена таблицы, разделённые ".". Чтобы вывести все столбцы таблицы, можно указать <имя таблицы>.\*. Для того, чтобы вывести только уникальные столбцы таблицы, используют слово DISTINCT <столбцы>. Предложением SELECT можно задавать вывод символьного выражения и так же исчисляемую колонку (виртуальную).

Пример:

SELECT Tab\_No, Fam, Oklad + Prem AS "оклад + премия".

В команде SELECT можно использовать специальные агрегатные функции:

COUNT () – количество отображаемых строк.

SUM () – суммирует значения числовых столбцов

MIN () – находит минимум числового столбца

MAX () – находит максимум числового столбца

AVG() – находит среднее значение числовой колонки, в скобках имя столбца.

Пример:

SELECT SUM(Oklad);

SELECT Name, SUM(Cena \* Kol-vo) AS "Сумма";

SELECT COUNT(\*);

SELECT MAX(Oklad), MIN(Oklad), AVG(Oklad).

Для отбора строк по заданному критерию используют предложение WHERE:

Пример:

SELECT имя покупателя

FROM заказ

WHERE Изд# = 139

Предикат IN в неявном виде заменяет квантор существования.

Пример:

WHERE X IN P(X) (эквивалентно ∃xP(x))

WHERE X NOT IN P(X) (эквивалентно ∀x¬P(x))

Множество, задающееся в предложении IN(), можно определить не только перечислением его элементов, но и косвенно, используя вложенный подзапрос:

Пример:

WHERE X IN (SELECT ... FROM ...)

Предикат LIKE осуществляет выбор на включаемые надстройки, задаваемой переменной или константой. Подстрока определяется заданными символами, замещёнными "-" (замещает один символ) и "%" (любое число символов):

Пример:

WHERE ФИО LIKE "ПЕТ%"

В предложении WHERE предикаты BETWEEN, IN, LIKE могут объединятся связями AND, OR, NOT.

Два дополнительных предложения GROUP BY и HAVING позволяют располагать строки по группам. Затем можно выполнять операции с этими группами. Например, использовать операцию агрегирования.

Предложение GROUP BY позволяет строки результирующей таблицы, определяемой предложением SELECT, сгруппировать в группы с одинаковым значением заданных колонок. После агрегирования каждая такая колонка сводится к одной строке результирующей таблицы. В качестве имени колонки используется имя агрегирующей функции с номером этой функции.

Предложение HAVING позволяет отображать составные группы строк, удовлетворяющих заданным условиям. Действие HAVING аналогичное WHERE, но определяет условие, которому должна удовлетворять каждая группа для вывода результирующей таблицы. Например, для вывода упорядоченного по алфавиту перечня деталей стоимостью более 300р., имеющихся на складе в количестве более 10 шт., можно использовать команду SELECT.

SELECT <номер>, SUM(<количество>), <стоимость>, <название>

FROM <склад>

WHERE <стоимость> > 300

GROUP BY <название>, <номер>, <стоимость>

HAVING SUM(<количество>) > 10

ORDER BY <название>

Если HAVING используется без GROUP BY, то его действие распространяется на всю таблицу и эквивалентно WHERE.

ORDER BY ‘<колонка | целое число [ASC, DESC]>’

Вместо имени колонки допустимо использование целого числа, определяющего её позицию в таблице SELECT.

Особенности языка SQL

1. фактический – стандарт обращения к современным БД;
2. ориентирован на операции с данными, представленными в виде совокупности таблиц (DBASE работает с записями);
3. ориентирован на конечный результат обработки. Система сама определяет оптимальный путь организации запроса в отличие от процедурных языков, требующих задания последовательности выполнения операций обработки. Поэтому говорят, что SQL более декларативен, чем язык РА, который ближе к процедурным языкам.

Объекты современных реляционных БД

Данные реляционных БД хранятся в виде таблиц. Для поддержки уникальности данных в колонках таблиц, а также для ускорения доступа используются индексы. Кроме таблиц, многие СУБД поддерживают представление и рассматривают их как отдельные объекты БД.

Представление

Представление — виртуальная таблица, то есть таблица, которая сама по себе не существует, но для пользователя выглядит так, словно она существует. Тогда как базовая таблица — реальная таблица, для любой строки которой существует двойник в физической памяти. То есть представления не поддерживаются их собственными физически хранимыми данными. Вместо этого в каталоге хранятся их определения в терминах других таблиц. Представление может быть создано с помощью команды SQL

CREATE VIEW <имя представления>

[(<имя поля>[,<имя поля>…])]

AS SELECT

FROM

WHERE

Когда исполняется это предложение, подзапрос, следующий за AS, не исполняется. Вместо этого, он просто сохраняется в каталоге. Но для пользователя это выглядит так, будто в БД действительно существует такая таблица. Эта таблица представляет собой фактически окно в реальную таблицу. Это окно является динамическим. Изменение в реальной таблице будут автоматически видны через это окно. Изменения в виртуальной таблице также будут автоматически внесены в реальную таблицу. Пользователь может производить операции над представлением, как если бы это была реальная таблица.

Синонимы

Синонимы представляют собой альтернативные имена таблиц, либо базовых, либо виртуальных. Чаще всего, синоним создается для таблицы. Которая была создана каким-нибудь другим пользователем и для которой вы должны были бы в противном случае использовать полностью уточненное имя. Например, пользователь Ivan создает таблицу

CREATE TABLE Primer,

тогда Петр может использовать таблицу Ivan.Primer. чтобы избежать длинных обращений, Петр может создать синоним

CREATE SYNONYM P1 FOR Ivan.Primer

и обращаться к таблице через P1.

Имя Р1 является частным для пользователя Петра. Другой пользователь может создать собственный синоним. Описание всех таблиц, представлений, синонимов хранится в специальном каталоге.

Каталог

Системная БД, содержащая дескриптор (информацию) относительно разных объектов, а именно: таблиц, представлений, синонимов, индексов, БД, прав доступа и т.д. Сам каталог состоит из множества таблиц, точно таких же, как обычные пользовательские таблицы.

SYS TABLES — таблицы и представления. Обычно содержат поля:

Name – имя таблицы;

Creator – имя пользователя;

ColCount – количество столбцов.

SYS COLUMNS — колонки БД. В каждой строке этой таблицы содержится информация о столбце какой-либо таблицы. Для этого служат поля

Name – имя столбца;

TBName – имя таблицы;

ColType – тип данных для столбца.

SYS INDEX — любому индексу в системе отводится одна строка в этой таблице. Для каждого индекса в ней указано его имя, имя индексной таблицы TBName, имя пользователя и т.д.

Так как каталог состоит из таблиц точно таких же, как обычные пользовательские таблицы, из него можно запрашивать данные с помощью предложения SELECT.

Пример:

SELECT FROM SYSIBM.SYS COLUMNS WHERE Creator=’’

Хранимые программы и процедуры

Обычно в среде клиент-сервер конечные приложения располагаются на клиентской машине и там же выполняются. Любой доступ к БД из приложения использует контакт с сервером по сети. Когда же приложение рассматривается внутри БД, оно называется хранимой командой или хранимой процедурой. Она выполняется непосредственно на компьютере сервера БД.

Хранимая команда — SQL-запрос, который хранится на сервере в скомпилированном виде. Выполнение хранимой программы производится гораздо быстрее, чем обычные SQL-запросы, однако хранение команды имеет ограниченную функциональность в рамках синтаксиса SQL.

Хранимая процедура — предложения, объединяющие SQL-запросы и процедурную логику. Хранимые процедуры позволяют хранить на сервере сложные приложения, выполняющие большой объем работы без передачи данных по сети и взаимодействия с клиентом. Хранимые процедуры обычно пишутся на автономном языке разработки конечных приложений.

Использование хранимых команд и процедур преследует следующие цели:

1. Повышение производительности;
2. Простота использования. Обычно технология, которая подразумевает большое число клиентов, подключаемых к серверу. Тогда использование хранимых команд и процедур позволяет хранить приложения на одном компьютере, а не на каждом компьютере в отдельности.
3. Усиление защищенности данных — набор операций, которые могут быть осуществлены пользователем, легко контролируются с помощью управления доступом к небольшому количеству хранимых процедур. Нормальное завершение процедуры возвращает нулевое значение в вызвавшее ее приложение. Возвращенное значение может быть найдено в специальном файле, чтобы выдать пользователю сообщение об ошибке.

Триггеры

Триггеры — определяемое пользователем действие, которое выполняется, когда над таблицей, к которой подключен триггер, выполняется операция INSERT, UPDATE или DELETE.

Триггеры используются для решения трех основных задач:

1. Усиление ссылочной целостности. Например, триггер можно использовать для реализации ограничений ссылочной целостности, выходящих за пределы стандартных ограничений СУБД. Пользователь может захотеть реализовать правило каскадного изменения данных. для этого необходимо создать триггер, который будет обновлять дочерние таблицы при каждом изменении колонки родительской таблицы.
2. Проверка данных => значения колонок могут пересекаться в триггере, представляемом в качестве параметров, где над ними могут производиться самые разнообразные операции. Например, пользователь может создать триггер, который будет получать из приложения данные для ввода в строку таблицы, создавать уникальный ключ для этой строки внутри своей процедуры и перерабатывать в БД полную строку с ключом.
3. Регистрация изменений данных. Создатель таблицы, или администратор БД может пожелать иметь информацию о времени каждого изменения данных в таблице, а также какой пользователь… Для этого он создает триггер, который поступает в системное время операции UPDATE имя пользователя. Процедура триггера затем вводит эту информацию в специальную таблицу регистрации изменений. Триггер может быть определен для выполнения либо перед операцией (BEFORE), либо после (AFTER) INSERT, UPDATE, DELETE.

Формат команды

CREATE TRIGGER ON <имя таблицы>

FOR DELETE | UPDATE | INSERT

AS <логическое выражение>

<логическое выражение> — задает выражение, определяемое для выражения триггера. Параметром логического выражения может быть функция либо сохраненная процедура, возвращающая логическое выражение.

* + 1. Иерархическая модель

Иерархическая модель БД представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих перевернутое дерево (граф). Данная модель характеризуется такими параметрами, как уровни, узлы, связи. Принцип работы модели таков, что несколько узлов более низкого уровня

где A, B, C, D, E – типы записей (сегменты)

A1

B1 B2 C­1 C2 C3

D1D2D3 E­1E2E3E4 E5

A

B

C

D

E

соединяются при помощи связи с одним узлом более высокого уровня.

Узел – информационная модель элемента, находящегося на данном уровне иерархии.

Совокупность корневой записи и множества подчинённых ей записей называется деревом. Число деревьев определяется числом корневых записей. Групповые отношения этой модели не именуются, т.к. определяются парой типов записей. К любой записи существует единственный путь от корня записи. Этот путь называется иерархическим. Каждая запись идентифицируется полным сцепленным ключом, под которым понимают совокупность ключей всех записей от корня вдоль иерархического пути.

Свойства иерархической модели данных:

* Несколько узлов низшего уровня связано только с одним узлом высшего уровня.
* Иерархическое дерево имеет только одну вершину (корень), не подчиненную никакой другой вершине.
* Каждый узел имеет свое имя (идентификатор).
* Существует только один путь от корневой записи к более частной записи данных.

Особенности иерархической модели данных

* Данные организованны в иерархической структуре;
* При отображении сетевых структур необходимо дублирование данных и меры по поддержанию семантической целостности;
* Основная единица обработки – запись;
* Обработка начинается с корневой записи, доступ к некорневым записям осуществляется по иерархическому пути.

Иерархическая структура реализуется отношением "один ко многим" между исходной и подчиненной записями, однако для предоставления отношения многих ко многим необходимо дублирование деревьев. При этом сохранение целостности БД ложится на программиста.

Операции над данными:

1. Извлечь по значению ключа;
2. Запомнить;
3. Обновить;
4. Удалить.
	* 1. Сетевая модель

Сетевая модель БД похожа на иерархическую. Она имеет те же основные составляющие (узел, уровень, связь), однако характер их отношений принципиально иной. В сетевой модели принята свободная связь между элементами разных уровней.

Поликлиника

Организация

Сбербанк

Житель

Сберкнижка

Поликлиника

Поликлиника

Поликлиника

Поликлиника

Поликлиника

Поликлиника

Поликлиника

Каждую сетевую структуру можно представить в виде иерархической модели, но при этом сеть нуждается в преобразовании:

С точки зрения теории графов сетевой модели соответствует произвольный граф, возможно — с циклами и петлями, узлы которого – типы записей, а ребра – связи между ними.

Особенности сетевой модели

* Позволяет устанавливать несколько признаков одинаково направленных групповых отношений между двумя типами записи;
* Допускает циклические структуры.

Возможные операции над данными

1. Запомнить — занести новую запись в БД и автоматически включить её в групповое отношение (ГО), где она объявлена подчиненной соответствующим режимам включения;
2. Включить — позволяет подчинённую запись связать с записью-владельцем;
3. Переключить — переключить подчинённую запись на другого владельца в том же ГО;
4. Исключить — разрывает связь между владельцем и подчинённой записью, сохранив обе записи в БД.

Каждый тип ГО характеризуется:

1. способом упорядочивания подчиненной записи (ПЗ)
2. a) произвольный;

б) хронологический;

в) обратнохронологический;

г) сортированный;

1. режимом включения ПЗ

а) автоматический – подчинённая запись включается в отношение одновременно с запоминанием в БД, т.е. происходит автоматическое закрепление ПЗ за ее владельцем;

б) ручной – позволяет запомнить ПЗ в БД, а не включать сразу в ГО;

1. режимом исключения ПЗ вводится понятие класса принадлежности

Для сетевой модели:

a) фиксированный — ПЗ жёстко закрепляется за владельцем и не может существовать без него (поэт и произведение). При удалении записи-владельца (ЗВ) система автоматически удаляет ПЗ;

б) обязательный — каждая ПЗ всегда будет связана с некоторой ЗВ, но может быть переназначена на другую ЗВ. Для успешного удаления ЗВ необходимо, чтобы не было ПЗ с обязательным членством;

в) необязательный — позволяет исключить ПЗ из экземпляра ГО, но сохранить её в БД, не прикрепляя к другому владельцу.

Основные особенности обработки данных в сетевых моделях

1. Обработка может быть начата с записи любого типа, независимо от её расположения в структуре БД;
2. От извлечённой записи возможны переходы, как к её подчинённым записям, так и к тем, которым она подчинена;
3. Основная структурная единица – набор, единица обработки – запись.
	* 1. Объектно-ориентированная модель данных

В настоящий момент времени до конца не разработана.

* 1. Теория нормальных форм

Можно доказать, что любую структуру данных можно преобразовать в простую двухмерную таблицу. Такое представление является наиболее удобным и для пользователя, и для машины, - подавляющее большинство современных информационных систем работает именно с такими таблицами, т.е. с реляционными базами данных.

Основная идея реляционного подхода состоит в том, чтобы представить произвольную структуру данных в виде двухмерной таблицы, т.е. нормализовать структуру.

Каждая запись в таблице должна иметь первичный ключ, т.е. идентификатор (или адрес), значение которого однозначно определяет эту и только эту запись. Первичный ключ должен обладать двумя свойствами.

1. Однозначная идентификация записи: запись должна однозначно определяться значением ключа.
2. Отсутствие избыточности: никакое поле нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации.

Каждое значение первичного ключа в пределах таблицы должно быть уникальным. В противном случае невозможно отличить одну запись от другой. Указание ключа – это единственный способ отличить одну запись от другой. Обычно используют придуманные разработчиком уникальные цифровые значения – код, табельные номера и т.д.

Кроме первичного, могут использоваться так называемые простые (или вторичные) ключи таблицы. Простых ключей может быть множество. Они используются при упорядочивании (индексировании) таблиц.

Процесс превращения иерархической или сетевой структуры данных в реляционную называется нормализацией. Внешне эта операция очень проста, но содержит некоторые нюансы, игнорирование которых может привести к неприятностям. Нюансы эти заключаются в том, что даже для простых двухмерных структур приходится подправлять состав полей.

Например, мы включим в таблицу поле, значение которого не зависит от первичного ключа. В таком случае появляется возможность утери информации. Однако важнее другое: повторяя многократно одни и те же данные, мы не только переделаем массу лишней работы, но и неминуемо ошибемся. Поэтому следует стремиться к исключению из таблицы полей, которые не связаны непосредственно с первичным ключом таблицы. Для этого, помимо оперативной, можно создать несколько справочных таблиц. Оперативная таблица меняется часто, а справочники – редко, их легко выправить раз и навсегда, внося в дальнейшем лишь небольшие изменения.

При проектировании таблиц рекомендуются следующие "золотые правила":

1. Надо уяснить себе, что есть первичный ключ таблицы (т.е. убедиться, что двух записей с одинаковым значением ключа в таблице быть не может)
2. Если первичный ключ не просматривается, подумать, правильно ли подобран состав полей
3. Если первичный ключ безупречен, к нему можно дописывать любые атрибуты, зависящие только от ключа.

Если при просмотре подготовленной БД в паре таблиц обнаружится одноименное поле, которое не входит в первичный ключ ни одной из этих таблиц, - это ошибка нормализации. Система не сможет контролировать согласованность значений таких полей.

Состав атрибутов отношений БД должен удовлетворять двум основным требованиям:

1. Между атрибутами не должно быть нежелательных функциональных зависимостей (ФЗ);
2. Группировка атрибутов должна обеспечивать минимальное дублирование данных.

Удовлетворение этих требований достигается нормализацией отношений БД – пошаговый обратимый процесс декомпозиции (разложения) исходных отношений на более мелкие и простые отношения. При этом устраняются все нежелательные функциональные зависимости. Аппарат нормализации был разработан Е. Коддом.

В чём определяются различные нормальные формы (НФ)? Кодд выделил три НФ. На сегодня определены ещё четвёртая и пятая. Каждая НФ ограничена определённым типом ФЗ и устраняет соответствующие аномалии при вычислении операций над отношениями. При декомпозиции встает проблема обратимости. Декомпозиция гарантирует такое восстановление без потерь. В результате применения последовательности операций естественных соединений над проекциями исходных отношений должно получиться отношение, эквивалентное исходному, причём в результирующем отношении не должны появляться раньше отсутствующие кортежи – следствия ошибочного соединения.

Если есть атрибуты А и В, говорят что В функционально зависит от А, если для каждого значения А в любой момент времени существует ровно одно связанное с ним значение В, причём А и В могут быть составными – представлять собой группы, состоящие из 2-х и более атрибутов.

I нормальная форма

Простой атрибут – атрибут, значения которого неделимы, атомарны.

Сложный атрибут – атрибут, значение которого представляет собой конкатенацию значений одного/нескольких доменов (аналоги – агрегат, вектор, повторяющаяся группа)

Отношение находится в первой нормальной форме (НФ), если все атрибуты простые (атомарные) и нет повторяющихся групп. Отношение в I НФ должно быть прежде постановки вопроса о разбиении на два или более отношений, т.е. к I НФ необходимо привести универсальное отношение.

Универсальное отношение – отношение, в состав которого входят все атрибуты проектируемой БД.

Пример:

Сотрудники

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ФИО | Таб. № | Отдел | Тел. | Дети |
| Имя | Возраст |
| БорисовБорисов | 211211 | СССС | 3-573-57 | ИванМиша | 1015 |
| АндреевАндреев | 364364 | УПУП | 2-152-15 | МашаЕгор | 64 |

II нормальная форма

Отношение находится во II НФ, если оно находиться в I НФ, и каждый не ключевой атрибут функционально полно зависит от составного ключа.

Чтобы привести отношение ко 2 НФ необходимо:

1. Построить его проекцию, исключив атрибуты, которые не находятся в полной функциональной зависимости от составного ключа;
2. построить дополнительно одну или несколько проекций на часть составного ключа и атрибуты, функционально зависящие от этой части ключа. Атрибуты, функционально зависящие от одной и той же части ключа, объединяются в одно отношение.

Пример:

Универсальное отношение "Сотрудник" разбивается на два отношения:

Сотрудники Дети

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таб. № | ФИО | Отдел | Тел. | Таб. № | Имя | Возраст |
| 211 | Борисов | СС | 3-57 | 211 | Иван | 10 |
| 211 | Миша | 15 |
| 364 | Андреев | УП | 2-15 | 364 | Маша | 6 |
| 364 | Егор | 4 |

Наличие транзитивной зависимости порождает неудобства и аномалии следующего характера (например, атрибут Тел.):

1. Имеет место дублирование информации для сотрудников одного отдела;
2. Существует проблема контроля избыточности, поскольку изменение номера телефона отдела ведёт к необходимости поиска и замены номеров сотрудников отдела;
3. Аномалия добавления и удаления: нельзя включить данные о новом отделе, если на данный момент отсутствуют его сотрудники, и наоборот: при увольнении всех сотрудников отдела данные о нём нельзя сохранить.

Таким образом, отношение во II НФ также может потребовать дальнейших преобразований.

III нормальная форма

Отношение находиться в III НФ, если оно находиться во II НФ и в нём отсутствует транзитивная зависимость неключевых атрибутов от ключа.

Для преобразования к III НФ необходимо построить несколько проекций.

Пример:

(в нашем случае — отношение "Сотрудник" разбить на два: Отдел – Тел.,

Таб. № - ФИО – Отдел).

|  |  |  |
| --- | --- | --- |
| Сотрудники | Отдел | Дети |
| Таб. № | ФИО | Отдел | Отдел | Тел. | Дети- // - |
| 211 | Борисов | СС | СС | 3-56 |
| 364 | Андреев | УП | УП | 2-15 |

В итоге получили три таблицы: "Сотрудники", "Отдел", "Дети"

III НФ освобождает от избыточности и аномалий редактирования, если отношения имеют один ключ и другие зависимости (в т.ч. многозначные) в нём отсутствуют.

Но если имеются многозначные зависимости от ключа, то III НФ не обеспечивает отсутствия аномалий обновления. В этом случае применяют усиленную III НФ.

Усиленная III нормальная форма (Бойса-Кодда) НФБК

Отношение находятся в НФБК, если оно находится в III НФ, и в нём отсутствуют зависимости ключей от не ключевых атрибутов.

Пусть имеется отношение R(А, В, С) с ключом К = {А, В}. Между атрибутами этого отношения существуют ФЗ А, В → С и С → В:

А

С

С

А, В → С – зависит от ключа

С → В – зависимость ключа от не ключевых атрибутов

Пример:

А – адрес, В – город, С – индекс;

Проект (Деталь#, Проект#, Поставщик#);

Дет#, Пр# → Пост#, Пост# → Пр#

Каждый проект может обслуживаться несколькими поставщиками, но любой поставщик обслуживает только один проект.

Для приведения к НФБК необходимо выполнить

1. ;
2. .

Хотя существует НФ более высокого уровня, которые накладывают даже более сильные ограничения на отношения, на практике обычно стараются получить отношение НФБК.

Переход к НФБК происходит не по схеме , а с использованием более общего подхода к декомпозиции отношения.

IV нормальная форма

Отношение, находится в IV НФ, если оно находится в НФБК, но в нем отсутствуют многозначные зависимости. IV НФ показывает, что отношение может находиться в НФБК, но тем не менее могут существовать аномалии, особенно при добавлении.

Например, для отношения преподаватель Препод (ФИО, группа, предмет) при появлении у преподавателя новой группы в отношение приходится добавлять не один кортеж, а столько, сколько предметов он читает в этой группе. Аналогичная ситуация возникает при добавлении в отношение нового предмета.

Устранение аномалий достигается разложением исходного отношения на несколько отношений с многозначной зависимостью от одного и того же ключа.

ФИО

Предмет

Группа

В нашем случае "Препод" разбивается на:

Предмет (ФИО, предмет)

Группа (ФИО, группа)

* 1. Достоверность и безопасность информации

Поскольку первичное заполнение таблиц и ввод их в машину ведет человек, ошибки в данных являются не исключением, а правилом, и любая ИС должна иметь средство для диагностики и исправления ошибок.

Нарушение логической взаимосвязи – это логические (семантические) ошибки, ошибки смысла, которые могут быть обнаружены аппаратом формального логического контроля, построенным для ИС. Кроме того, конкретная ИС может иметь собственные средства дополнительного ("нестандартного") контроля, так как стандартные средства не могут охватить все возможные случаи. В современных СУБД имеются средства поддержания целостности данных. Кроме того, в современных ИС можно указать условия, которым должны удовлетворять значения некоторых полей (условия верификации данных).

Гораздо сложнее дело обстоит с ошибками в допустимых значениях данных. Такие ошибки условно называются арифметическими, хотя это не совсем точно, так как ошибочно может быть записано значение текстового данного: например, Иванов И.П. вместо Иванов А.П. Существует ряд средств для выявления арифметических ошибок, однако на пользовательском уровне ограничиваются простым визуальным контролем.

Термин "безопасность" относится к защите базы данных от несанкционированного доступа. Под безопасностью БД подразумевается, что пользователям разрешается выполнять некоторые действия. Под целостностью подразумевается, что это действие выполняется корректно.

Существуют различные аспекты безопасности:

* правовой;
* физический;
* аппаратное обеспечение;
* безопасность БД.

СУБД поддерживает два подхода доступа к данным: избирательный и обязательный. В случае избирательного управления пользователь обладает различными правами при работе с разными объектами. В случае обязательного управления каждому объекту данных присваивается некоторый уровень доступа, а каждый пользователь обладает определенным уровнем. Доступом к определенному объекту обладает только пользователь с соответствующим уровнем доступа.

Избирательное управление доступом.

Для определения правил безопасности нужно использовать некоторый язык:

CREATE SECURITY RULE <правило>

GRANT <список привилегий>

ON <выражение>

TO <список пользователей>

[ON ATTENTED VIOLATION <действие>]

<правило> — имя нового правила безопасности;

<список привилегий> может быть представлен:

RETRIEVE [<список атрибутов>]

INSERT

UPDATE [<список атрибутов>]

DELETE

ALL

<выражение> — подразумевает выражение реляционного исчисления, в котором задано диапазоном действие данного правила. Диапазоном является некоторое подмножество кортежей единственного отношения.

<действие> — процедура произвольной сложности, например, заблокировать клавиатуру, записать нарушение правил безопасности. Т.е. выполнить отслеживание угроз.

Пример:

CREATE SECURITY RULE ПС3 // создание правил секретности

GRANT RETRIEVE (У#, ЛФУ, Разр), DELETE

ON Узел WHERE ЛФУ<>"Сумматор"

TO Иванов, Петров, Сидоров // для пользователей

ON ATTENTED VIOLATION REJECT // не выполнять при нарушении этого правила

В системе должен быть предусмотрен способ устранения существующих правил безопасности.

DESTROY SECURITY RULE

Обязательное управление доступом

Каждый объект имеет некоторый уровень классификации. Эти уровни образуют строгую иерархию. Например, "совершенно секретно", "секретно", "для служебного пользования". Каждый пользователь имеет уровень доступа с такими же градациями, что и уровни классификации.

Можно сформулировать два простых правила:

1. Пользователь А имеет доступ к объекту В, если его уровень доступа больше или равен уровню классификации объекта;
2. Пользователь А может модифицировать объект В, если его уровень доступа равен уровню классификации объекта. Таким образом, любая информация, записанная пользователем А, приобретает классификацию, равную уровню пользователя А. Необходимо предотвратить запись секретных данных в файлы с меньшим уровнем классификации.

До сих пор предполагалось, что пользователь пытается незаконно проникнуть в БД с помощью обычных средств доступа, имеющихся в системе. Теперь рассмотрим случай, когда такой пользователь пытается проникнуть в БД, минуя систему, то есть физически перемещая часть БД или подключаясь к коммуникационному каналу. Наиболее эффективным методом борьбы является шифрование данных.

Исходные (незакодированные) данные называются открытым текстом. Открытый текст шифруется с помощью специального алгоритма шифрования. В качестве входных данных поступает открытый текст и ключ шифрования, на выводе — зашифрованный текст. Если алгоритм может не утаиваться, то ключ хранится в секрете.

Обычно выделяются следующие методы шифрования: процедуры подстановки, перестановки, комбинация указанных методов.

Рассмотрим алгоритм шифрования на примере использования процедуры подстановки.

1. Разбейте открытый текст на блоки, длина которых равна длине ключа шифрования. пусть в качестве открытого текста дана строка:

AS KINGFISHERS CATCH FIRE

ключ: ELIOT

разбиение: "AS\_KI", "NGFIS", "HERS\_", "CATCH", "\_FIRE"

1. Замените каждый символ открытого текста целым числом в диапазоне от 0 до 26, используя для пробелов число 00. В результате получится строка чисел

0119001109 1407060919 0805181900 0301200308 0006091805,

ключ: 0512091520.

1. Значение для каждого символа в каждом блоке открытого текста просуммируйте с соответствующими значениями каждого символа ключа шифрования по модулю 27.

Для нашего примера получим

0604092602 1919152412 1317100620 0813021801 0518180625

1. Замените каждое число в строке соответствующим текстом символов:

"FDIZB", "SSOXL", "MQJFT", "HMBRA", "ERRGY"

зашифрованный текст: FDIZBSSOXLMQJFTHMBRAERRGY

Если ключ известен, расшифровка может быть выполнена достаточно быстро.

# Глава 2. Основы разработки базы данных

Прежде чем приступать к работе с базой данных, в первую очередь необходимо выбрать модель представления данных. Она должна отвечать следующим требованиям:

* Наглядность представления информации;
* Простота ввода информации;
* Удобство поиска и отбора информации;
* Возможность использования информации, введенной в другую базу;
* Возможность быстрой перенастройки базы данных (добавление новых полей, новых записей, их удаление).

При разработке БД можно выделить следующие этапы работы.

I этап. Постановка проблемы

На этом этапе формируется задание по созданию БД. В нем подробно описывается состав базы, назначение и цели ее создания, а также перечисляется, какие виды работ предполагается осуществлять в этой базе данных (отбор, дополнение, изменение данных, печать или вывод отчета и т.д.).

II этап. Анализ объекта

На этом этапе необходимо рассмотреть, из каких объектов может состоять ваша БД, каковы свойства этих объектов. После разбиения БД на отдельные объекты необходимо рассмотреть свойства каждого из этих объектов, другими словами, установить, какими параметрами описывается каждый объект. Все эти сведения можно располагать в виде отдельных записей и таблиц. Далее необходимо рассмотреть тип данных каждой отдельной единицы записи (текстовый, числовой и т.д.). Сведения о типах данных также следует занести в составляемую таблицу.

III этап. Синтез модели

На этом этапе по проведенному выше анализу необходимо выбрать определенную модель БД. Далее рассматриваются достоинства и недостатки каждой модели, сопоставить их с требованиями и задачами вашей БД и выбрать ту модель, которая сможет максимально обеспечить реализацию поставленной задачи. После выбора модели необходимо нарисовать ее схему с указанием связей между таблицами или узлами.

IV этап. Способы представления информации, программный инструментарий

После создания модели необходимо, в зависимости от выбранного программного продукта, определить форму представления информации. В большинстве СУБД данные можно хранить в двух видах:

* С использованием форм;
* Без использования форм.

Форма – созданный пользователем графический интерфейс для ввода данных в базу.

V этап. Синтез компьютерной модели объекта и технология его создания

После рассмотрения инструментальных возможностей выбранного программного продукта можно приступить к реализации БД на компьютере. В процессе создания компьютерной модели можно выделить некоторые стадии, типичные для любой СУБД.

Стадия 1. Запуск СУБД, создание нового файла базы данных или открытие созданной ранее базы

В процессе выполнения данной стадии необходимо запустить СУБД, создать новый файл (новую базу) или открыть существующую.

Стадия 2. Создание исходной таблицы или таблиц.

Создавая исходную таблицу, необходимо указать имя и тип каждого поля. Имена полей не должны повторяться внутри одной таблицы. В процессе работы с БД можно дополнять таблицу новыми полями. Созданную таблицу необходимо сохранить, дав ей имя, уникальное в пределах создаваемой базы.

Стадия 3. Создание экранных форм.

Первоначально необходимо указать таблицу, на базе которой будет создаваться форма. Ее можно создавать при помощи Мастера форм или самостоятельно, указав, какой вид она должна иметь (например, в виде столбца или таблицы). При создании формы можно указывать не все поля, которые содержит таблица, а только некоторые из них. Имя формы может совпадать с именем таблицы, на базе которой она создана. На основе одной таблицы можно создать несколько форм, которые могут отличаться видом или количеством используемых из данной таблицы полей. После создания форму необходимо сохранить. Созданную форму можно редактировать, изменяя местоположение, размеры и формат полей.

Стадия 4. Заполнение БД.

Процесс заполнения БД может проводиться в двух видах: в виде таблицы и в виде формы. Числовые и текстовые поля можно заполнять в виде таблицы, а поля типа МЕМО и OLE – в виде формы.

VI этап. Работа с созданной базой данных

Работа с БД включает в себя такие действия, как:

* Поиск необходимых сведений;
* Сортировка данных;
* Отбор данных;
* Вывод на печать;
* Изменение и дополнение данных.

Рассмотрим все этапы создания и принципы работы с базами данных на примере СУБД Microsoft Access.

2.1 Методы проектирования реляционной БД

Когда перечень атрибутов составлен, очередная задача состоит в их агрегации, то есть компоновке атрибутов в объекты. Это и есть суть процесса проектирования реляционной базы данных.

Существует несколько подходов к проектированию РБД.

1. Декомпозиция (для небольших БД);
2. Синтез;
3. Использование модели объекта связи (метод ER-диаграмм).

2.1.1 Метод декомпозиции

Характеризуется тем, что сначала создается одно УО, которое затем разбивается на ряд отношений, находящихся в НФ. Использование таблицы кажется на первый взгляд удобным, на самом деле при работе с такой таблицей возникают аномалии обновления.

Возможный ключ — представляет собой атрибут или набор атрибутов, который может быть использован для данного отношения в качестве первичного ключа.

Детерминант — если А=>В — ФЗ и В не зависит функционально от любого подмножества А, то говорят, что А представляет собой детерминант В.

Большинство потенциальных аномалий в БД будет устранено при декомпозиции любого отношения в НФБК.

Отношение находится в НФБК, если и только если каждый детерминант отношения является возможным ключом.

Пример:

1. Функциональные зависимости:



1. Диаграмма ФЗ:

Из диаграммы видно, что УО на находится ни в 2НФ, ни в 3НФ.

Таб №

Имя реб.

Возраст

Таб №

Таб №

Таб №

Расположим рядом перечень всех детерминантов и всех возможных ключей.

|  |  |
| --- | --- |
| Возможный ключ | Детерминант |
| 1. Таб№, Имя\_Р | Таб№, Имя\_Р,ФИООтделТаб№Телефон |

Поскольку не каждый детерминант в данном отношении является возможным ключом, следовательно, отношение "сотрудник" не находится в НФБК и его нужно подвергнуть декомпозиции.

Общий подход декомпозиции заключается в следующих шагах:

1. Разработка универсального отношения БД; для построенного УО находится первичный ключ.
2. Определение всех ФЗ между атрибутами отношения;
3. Определение того, находится ли отношение в НФБК, если да, то проектирование завершается, если нет, то отношение должно быть разбито на два.
4. Повторение шагов 2) и 3) для каждого нового отношения, построенного в результате декомпозиции.

Разбиение отношения на два отношения на шаге 2) осуществляется по следующему правилу.

Пусть отношение  не находится в НФБК. Определяется ФЗ, например, , которое является причиной того, что r не находится в НФБК, т.е. С является детерминантом но не является возможным ключом.

Создаются два новых отношения,  и , где зависимая часть была выделена из отношения и опущена при формировании отношения r1, и полностью использована при формировании отношения r2. Теперь нужно проверить, находится ли полученное отношение в НФБК. Про отношение говорят, что оно является проекцией отношения r. Этот тип декомпозиции называется декомпозицией без потерь.

На этом же этапе должно быть принято решение, какую ФЗ выбрать для проведения первой проекции. Не исключено, что в итоге выбора той или иной I (начальной) проекции будут получены разные БД. Простым правилом выбора … , с последующим использованием для I проекции правой зависимости (). Если таких цепочек нет, то безразлично, с чего начинать.

Это правило можно сформулировать и так: следует избегать выбора ФЗ, правая часть которой сама (целиком или частично), является детерминантом другой ФЗ.

Отношение, получаемое в результате декомпозиции, должно удовлетворять требованиям:

1. Возможность восстановить в точности исходное отношение путем естественного соединения отношений — результатов декомпозиции;
2. Сохранение всех ФЗ исходного отношения.

Пусть в отношении r со схемой R имеется множество ФЗ. Говорят, что схема R разложима без потерь на отношения  с сохранением ФЗ, если для любого кортежа . Кортеж может быть восстановлен

соединением его проекций. Невыполнение этих требований приведет к получению набора отношений, порождающих не согласующиеся данные.

Условия разложения без потерь:

а) для разложения, состоящего из двух отношений. Если отношение R1 и R2 являются разложением с сохранением множества ФЗ, это разложение обеспечивает соединение без потерь, если либо . Здесь операции " ∩ " и " – " определены над списками атрибутов отношений. Это правило можно сформулировать по-другому: отсутствие потерь при декомпозиции гарантируется, если от общего атрибута двух получаемых отношений зависит хотя бы один атрибут из двух оставшихся.

б) для разложения, состоящего более чем из двух отношений (метод табло). Процедура состоит в построении таблицы, строками которой являются имена, полученные при декомпозиции отношений, а столбцы — список атрибутов этих отношений без повторений. Таблица заполняется символами , если элемент i-й строки в j-м столбце соответствует атрибуту Aj в отношении Ri, в противном случае ничего не ставится. После заполнения таблицы следует итеративный просмотр всех ФЗ . Если для атрибутов из Х найдутся строки, где в соответствующих местах стоят aj, то пустые элементы этих строк, соответствующие столбцам атрибутов из Y, заменяются на aj\*. Если в результате появится строка таблицы, заполненная полностью aj и aj\*, то данное соединение без потерь. В противном случае — с потерями.

Метод декомпозиции применим для задач с малым числом атрибутов и отношений.

2.1.2 Метод синтеза

Отметим еще одно важное правило, лежащее в основе метода синтеза.

Все ФЗ с одинаковыми детерминантами нужно выделять в одну группу и каждой такой группе отводить свои собственные отношения. Получаемые отношения проверяются на соответствие с НФБК.

Этот прием следует применять в том случае, когда использование метода декомпозиции может привести к потере ФЗ, например, при выделении из  зависимости  будут получены отношения и , ни одно из которых не будет содержать зависимость . В этом случае метод декомпозиции не работает.

Правильно (метод синтеза): , 

Использование описанного выше правила декомпозиции может быть осложнено присутствием избыточных ФЗ (ИФЗ).

ИФЗ — зависимость, не заключающая в себе такой информации, которая не могла бы быть получена на основе других зависимостей, использованных при проектировании БД.

Поскольку ИФЗ не содержит уникальной информации, она может быть удалена из набора ФЗ. ИФЗ удаляется до начала проектирования (т.е. декомпозиции).

Возникновение ИФЗ связано с:

1. наличием транзитивных зависимостей;
2. добавлением.

Пример:

a) Исходный набор ФЗ После удаления ИФЗ

b) Эта форма избыточности имеет несколько видов:

1. Если , то является правильной, но избыточной;
2. Если , то  избыточна.

Рассмотренные ПВ ФЗ из исходных ФЗ входят в состав так называемых аксиом вывода (АВ).

АВ — правило, согласно которому, если отношение удовлетворяет определенным ФЗ, то оно должно удовлетворять и другим определенным ФЗ.

Рассмотрим 6 основных правил вывода (АВ).

1. Рефлексивность. А→А,

А

А

1. Добавление (расширение)

А

В

С

А

В

а) если А→В, то А,С→В

б) если А🡪В, то А, С 🡪 В,С ИФЗ

С

А

B

1. Транзитивность. Если А→B и B→C, то А→С ИФЗ

А

В

С

4) Псевдотранзитивность. Если А→B и B,С→D, то А,С→D ИФЗ

С

В

А

С

В

А

С

В

D

А

5) Объединение (аддитивность). Если А→B и А→С, то А→В,С

6) Декомпозиция (проективность). Если А→В,С, то А→B и А→С

С

В

А

С

В

А

Используя аксиомы вывода f1-f6 можно получить другие правила вывода для ФЗ. Например, с использованием аксиом f1 и f2 можно получить A,B→B. Первые три аксиомы называются аксиомами Армстронга, оставшиеся три следуют из них. Использование полной системы аксиом позволяет вывести все функциональные зависимости, допустимые в множестве ФЗ. Пусть F – множество ФЗ для схемы отношений R. Множество ФЗ, которые логически следуют из F, называют замыканием F (F+). Если F=F+, то говорят, что F — полное семейство зависимости. Вычисление F+ для F являются трудоемкой задачей, поскольку мощность F+ может быть велика даже при небольшой мощности F. Вычисление же X+ для данного множества атрибутов Х не представляет трудности. Х+— это замыкание Х относительно F, если есть множество атрибутов А таких, что зависимость Х→А может быть выведена из F по аксиомам f1,f2, f4.

2.1.3 Метод объектной связи

Отличается от метода декомпозиции тем, что ФЗ привлекаются не на начальном, а на конечном этапе проектирования.

Предположим, что проектируется БД, предназначенная для хранения информации о преподавателях и курсах, которые они читают. Двумя главными сущностями здесь являются преподаватель и курс. Они соотносятся с помощью связи "читает", что позволяет нам сказать "преподаватель читает курс". Связь "читает" может быть представлена двумя способами.

1. Метод ER-диаграмм в ER-экземплярах;
2. Метод ER-типов.

п1

п2

п3

п4

к1

к2

к3

к4

препод.

курс

читает

1

1

П#, … класс принадлежности: К#, …

необязательный

1) 2)

Необходимо различать понятия сущности и отношения. Сущность переходит в отношение путем выделения ее из ER-диаграмм. Отсюда различие между ключом сущности и ключом отношения, они не всегда совпадают. В процессе перехода сущность → отношение ключ сущности может быть дополнен некоторым атрибутом и стать ключом отношения. В большинстве случаев для определения набора отношений используется диаграмма ER-типа (вторая схема).

Необязательный класс принадлежности

Обязательный класс принадлежности

Нетрудно подсчитать, что для двух объектов общее число возможных состояний . 4 типов соответствия; 2 класса принадлежности, 2 объекта.

Общий подход к проектированию

Связь "читает" — бинарная, так как соединяет две сущности.

1. Строится диаграмма ER-типа, включающая в себя все сущности и связи;
2. Строится набор предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения;
3. Составляется список всех атрибутов (тех из них, которые не были перечислены в ER-диаграмме в качестве ключей сущности), каждый из этих атрибутов приписывается одному из предварительных отношений с тем условием, чтобы эти отношения находились в НФБК. Для каждого отношения должны быть определены межатрибутные функциональные зависимости, с помощью которых проверяется соответствие отношений НФБК. Если полученные в итоге отношения не находятся в НФБК или если некоторым атрибутам не находятся логически обоснованных мест в предварительных отношениях, необходимо пересмотреть ER-диаграмму.

Предварительные отношения для бинарных связей с типом соответствия 1:1.

Перечень общих правил генерации отношений можно получить, опираясь на:

1. Тип соответствия;
2. Класс принадлежности;

как определяющие факторы.

Правило 1: Если степень бинарной связи равна 1:1 и класс принадлежности обеих сущностей обязательный, то требуется только одно отношение. Первичным ключом этого отношения может быть ключ любой из этих двух сущностей. В этом случае гарантируется однократное появление каждого значения ключа в любом экземпляре отношения.

Правило 2: Если степень бинарной связи равна 1:1 и класс одной из сущностей необязательный, то необходимо построение двух отношений, под каждую сущность необходимо выделение одного отношения. Ключ сущности, для которого класс принадлежности является необязательным, добавляется в качестве атрибута в отношение, выделенное для сущности с обязательным классом принадлежности.

В том случае, если класс принадлежности ни одной из сущностей не является необязательным, недостаточно использования и двух отношений, т.к. возникают проблемы с внесением ключа сущности в отношение, выделенное под другую сущность.

Правило 3: Если степень бинарной связи равна 1:1 и класс принадлежности ни одной из сущностей не является необязательным, то используется три отношения — по одному для каждой сущности — ключи которых служат в качестве первичных в соответствующих отношениях и одного для связи. Отношение, выделенное для связи, будет иметь по одному ключу сущности от каждой сущности. Читает (Пр#,К#, …)

Предварительные отношения для бинарных связей с типом соответствия 1:M.

Для них требуется два правила. Фактором, определяющим выбор и использование одного из этих правил, является класс принадлежности многосвязной сущности. Класс принадлежности односвязной сущности не влияет на конечный результат в обоих случаях.

|  |  |  |  |
| --- | --- | --- | --- |
| K#препод.курсчитает1MП#, … К#, …Проблемы: пустые поля и дублирование | Курс | Пр# | ФИО |
| 12110301— | историяполитологияфизикаматематика— | 33456 | ИвановИвановПетровСидоровАндреев |

Правило 4: Если степень бинарной связи равна 1:М и класс принадлежности М-связной сущности обязательный, то достаточно использовать два отношения: по одному на каждую сущность, при условии что ключ сущности служит в качестве первичного ключа для соответствующего отношения. Ключ же односвязной сущности должен быть добавлен как атрибут в отношение, отводимое М-связной сущности.

То есть получим таблицы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| К# | Курс | Пр# | Пр# | ФИО |
|  |  |  |  |  |

Если односвязная сущность вырождена в атрибут, то есть имеет в своем составе один атрибут, то возникает проблема.

Пример:

1. Товар хранится в ячейках на складе. В одной ячейке могут храниться товары разного рода, но каждый товар только в одной ячейке.

Ячейка вырождена.

товар

ячейка

хранится

М

1

Т#, … Яч#, …

1. Сотрудники занимают должность

сотрудник

должность

занимает

М

1

Анализ примеров позволяет сделать вывод: если односвязная сущность представлена только ключом, то, в зависимости от класса принадлежности односвязной сущности, применяют разные правила:

а) Когда класс принадлежности односвязной сущности обязателен, то в "товар" добавляем "номер ячейки", получаем одно отношение.

б) Если класс принадлежности односвязной сущности необязателен, то имеем два отношения, так как "должность", которую никто не занимает, нужно хранить в БД.

Правило 5: Если степень бинарной связи равна 1:М и класс принадлежности М-связной сущности необязателен, то необходимо использовать три отношения: по одному на сущность и одно для связи. Связь должна иметь среди своих атрибутов ключ сущности от каждой сущности.

Предварительные отношения для бинарных связей с типом соответствия М:М.

Правило 6: Если степень бинарной связи равна М:М, то для хранения данных необходимо три отношения: по одному на сущность и одно для связи. Ключи сущности входят в связь. Если одна из сущностей вырождена, то — два отношения (т.е. достаточно будет двух таблиц).

Предварительные отношения для многосторонних связей.

Правило 7: В случае трехсторонней связи необходимо использовать четыре отношения: по одному на сущность и одно для связи. Отношение, порождаемое связью, имеет в себе среди атрибутов ключи сущности от каждой сущности.

Если связь n-сторонняя, требуется N+1 отношений: N для сущностей и 1 для связи.

С1

С2

 связь

С3

С1#, …

С3#, …

С2#, …

По правилу 6 имеем три отношения. Количество характеристика связи.

Имеем четыре отношения.

МС

Узел

 состав

 узла

Блок

С1#, …

У#, …

М

М

 явл-ся

 эл-ом

1

МС#, У#, К

М

Использование ролей в ER-моделях.

Одних только сущностей и связей может оказаться недостаточно для моделирования. Одна из таких ситуаций возникает тогда, когда экземпляры некоторой сущности должны играть разные роли. Рассмотрим на примере.

Пример:

Пусть необходима база данных для хранения информации о персонале предприятия. Различают две категории служащих: мастеров и сборщиков, мастера получают фиксированный оклад, у сборщиков почасовая оплата. Построим ER-диаграмму.

мастер

сборщик

руководит

1

М

М#, … С#, …

Используем правило 4, получим отношение:

Мастер (М#, …)

Сборщик(C#, …, M#)

Проблема возникает при добавлении не ключевых атрибутов в предварительные отношения. Предположим, что такими не ключевыми атрибутами являются ФИО служащего, телефон рабочий, телефон домашний, адрес, ставка, оклад, разряд. Нет проблем при размещении следующих атрибутов к отношениям:

Мастер (М#, тел. рабочий, оклад)

Сборщик(C#, ставка, разряд, М#)

Не размещенными остались атрибуты: ФИО, домашний телефон, адрес. Для полноты их следовало бы поместить в оба отношения, однако, как это было в общем правиле проектирования, каждый ключевой атрибут следует размещать только в одном отношении. Можно, конечно, оставшиеся три атрибута преобразовать в шесть атрибутов. Однако это не будет удачным решением, поскольку может привести к следующей проблеме: предположим, необходимо найти номер домашнего телефона служащего X. Т.к. неизвестно: Х – мастер или сборщик, необходимо просмотреть оба отношения с целью нахождения именно Х. А если существуют два служащих с именами Х (мастер и сборщик), то может быть выбран неправильный номер телефона. Для решения этой проблемы необходимо завести отношение "служащие" (то есть построить таблицу служащих). Мастер и сборщик — это те роли, которые они могут играть. Тогда ER-диаграмма имеет вид:

Шифр М#, C# и Сл# определены на одном домене.

мастер

сборщик

 руководит

1

М

М#, … С#, …

служащий

Сл#, …

Правило: исходная сущность служит источником генерации одного отношения. Ролевые элементы и связи, их соединяющие, порождают такое число отношений, которое определяется ранее описанными правилами. Причем каждая роль трактуется как обычная сущность.

команда-

хозяин

команда-

гость

расписание

1

М

КХ#, … КГ#, …

команда

К#, …

Приведенный пример характеризуется

а) наличием связи между ролевыми объектами;

б) ролевые объекты не вырождены;

Особый случай возникает при наличии связи между вырожденными сущностями. Рассмотрим пример со спортивными командами, которые соревнуются друг с другом.

Получим два отношения:

Команда (K#, …)

Расписание (КХ#, КГ#, дата, счет)

Пример:

2.2 Организация СУБД

Понятие СУБД относится к набору средств ПО, необходимых для использования фактографических БД. Различают документальные и фактографические БД.

Документальные БД хранят совокупность произвольных текстовых документов.

Фактографические БД хранят множество сведений и фактов, хранящихся в информационной системе, удовлетворяющих фиксированной совокупности форматов.

СУБД — это набор ПС, позволяющий:

1. Описать и манипулировать данными, для чего предназначены соответствующие языки: язык описания данных (ЯОД) и язык манипулирования данными (ЯМД). Термин ЯД означает либо оба, либо один из названных языков. ЯД может быть включен в универсальный язык, либо представлять собой оригинальное языковое средство. В первом случае, включаемый язык называется подъязыком данных, во втором автономным ЯД.
2. Поддерживать модели данных пользователя.
3. Обеспечить защиту и целостность данных. Защита — это использование БД, пользователями имеющими на это право. Целостность — поддержка согласованности данных.

Логически в современных СУБД можно выделить:

1. Внутренняя часть — ядро СУБД (Data Base Engine — DBE).
2. Компилятор языка БД (SQL).
3. Набор утилит.

Ядро отвечает за следующие процессы:

1. Управление данными во внешней памяти.
2. Управление буферами оперативной памяти.
3. Управление транзакциями.
4. Журнализация.

Выделяют следующие компоненты ядра (DBE):

1. Менеджер данных.
2. Менеджер буфера.
3. Менеджер транзакций.
4. Менеджер журнала.

### БД

#### Ядро СУБД

Утилиты БД

Программа SQL

Ядро СУБД обладает собственным интерфейсом, недоступным пользователю напрямую. Этот интерфейс используется программами, производимыми компилятором SQL и утилитами БД.

При использовании архитектуры "клиент-сервер" ядро является основной составляющей сервера.

Основная функция компилятора — компиляция операторов языка БД в некоторую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (SQL) являются непроцедурными.

2.2.1 Требования к современной СУБД

Традиционные файловые системы характеризуются тесной связью между физическими данными и прикладной программой. В ней отсутствуют практически все средства, предлагаемые СУБД.

Перечислим основные проблемы, возникающие в файловых системах:

1. Зависимость данных;
2. Жесткость и статичность;
3. Дублирование данных;
4. Отсутствие интеграции;
5. Невозможность обработки нетипичных запросов.

Современные СУБД разрабатываются с целью устранения этих недостатков.

Основные требования:

1. Независимость данных. Одна из самых важных задач при разработке БД — спроектировать ее так, чтобы изменение БД можно было выполнить без модификации прикладных программ. Для реализации этого требования структура БД должна отвечать требованиям физической и логической независимости данных. Физическая независимость данных — когда в файловой организации данных и в аппаратных средствах вносятся изменения, они должны быть отражены в ПО БД (т.е. в СУБД), но не должны затрагивать прикладные программы. Логическая независимость данных — представление данных в прикладной программе должно быть защищено от изменения:
* в глобальной логической структуре;
* в требованиях данных других прикладных программ.

Примеры возможных изменений:

а) модификация старых прикладных программ;

б) добавление новых прикладных программ, использующих новые типы данных;

в) добавление новых полей и создание новых связей.

Для достижения такой независимости необходимо отделить представление данных в каждой прикладной программе от общего логического представления и обеспечить возможность добавления новых полей в запись без перезаписи тех прикладных программ, которые используют эту запись. Тогда перечисленные изменения приведут к модификации общего логического представления и не коснутся прикладной программы.

Должны существовать три отдельных представления организации БД:

* физическое;
* общелогическое (концептуальная модель);
* представление данных в прикладных программах.

Итак, при логической независимости данных изменения ни в СУБД, ни в других прикладных программах не должны привести, в идеале, к изменению программы пользователя.

1. Универсальность. СУБД должна поддерживать разные модели данных.
2. Совместимость. Сохранение работоспособности при развитии программного и аппаратного обеспечения.
3. Минимальная избыточность данных.
4. Целостность данных.

Журнал

Копия 1

Сервисная программа

восстановления

БД

Копия 2

(восстановление вперед)

а) физическая — защита данных от физических разрушений. Обеспечивается средствами ведения журнального файла, в котором регистрируются все изменения БД с некоторого момента времени. На момент начала ведения журнального файла создается копия БД.

б) логическая — предупреждает неверное использование данных. Обеспечивается механизмом управления доступом к данным. Ограничение доступа ко всей БД, доступ не ко всей записи, а только к ее части, заданной областью допустимых значений.

в) семантическая — поддерживает осмысленное сочетание разных данных.

1. Защита данных от несанкционированного доступа.

а) конфиденциальность —защита от несанкционированного получения данных;

б) целостность — защита от несанкционированного изменения данных;

в) доступность — защита от несанкционированного удержания данных.

1. Обеспечение коллективного доступа к данным. Поскольку данные интегрированы, возникает проблема синхронизации параллельного доступа к одним и тем же данным многих пользователей. Обычно проблема возникает при записи. Также проблема возникает, когда процедура обновления завершается аварийно. В этом случае до разблокировки необходимо выполнить откат, или восстановление назад. При откате будут уничтожены все изменения БД, инициированные процедурой обновления.
2. СУБД должна поддерживать как централизованные, так и распределенные БД.

2.2.2 Архитектура СУБД

Существует несколько уровней представления данных:

1. Описание конкретного конечного пользователя, имеющего локальное описание;
2. Общее логическое описание, интегрирующее описание локальных пользователей;
3. Описание физической организации БД.

Описание модели на каком-либо языке называется схемой. В соответствии с уровнями представления различают:

1. Подсхема, или внешняя схема — представление локального пользователя;
2. Концептуальная схема, или модель описания логической структуры БД на языке СУБД.
3. Физическая, или внутренняя схема.

Указанные виды схем связывают больше с этапом эксплуатации БД, когда БД уже спроектирована, и общая логическая схема нашла свое отражение в конкретной СУБД.

На этапе проектирования можно выделить логическое пользовательское представление и информационную схему предметной области.

Локальное пользовательское представление 1

ЛПП №

Инфологическая схема

Концептуальная модель

Внутренняя схема

Внешн. Схема 1

Внешн. Схема 2

Физич.

БД

Уровни абстракции,

поддерживаемые СУБД

Все эти разновидности уровней описания принято связывать с понятием архитектура СУБД.

Описание предметной области, выполненное без ориентации на используемые в дальнейшем программные и технические средства, называется инфологической моделью предметной области, а сам этап проектирования — инфологическим проектированием. Концептуальная модель БД является моделью логического уровня, то есть обрабатывает логические связи между элементами данных безотносительно к их содержанию и среде хранения.

Эта модель строится в соответствии терминам той конкретной СУБД, в среде которой проектируется БД. Этот этап называется даталогическим проектированием.

Описание физической структуры БД называется схемой хранения. Этот этап называется физическим проектированием. На этом этапе могут выполняться следующие работы:

1. Выбор типа носителя;
2. Способ организации данных;
3. Метод доступа;
4. Определение размера физического блока;
5. Выбор методов сжатия или отказ от них;
6. Проблема утилизации и т.д.

В большинстве настольных СУБД этот этап проектирования скрыт от пользователя.

Описание логической структуры БД с точки зрения конкретного пользователя называется подсхемой. Это внешняя модель БД. Если СУБД поддерживает схему, схему хранения и подсхему, то она является СУБД с трехуровневой архитектурой. Если СУБД поддерживает уровень подсхем, то перед проектировщиком возникает задача их определения. Это можно рассматривать как еще один этап проектирования. Если определена подсхема, то пользователь имеет доступ только к тем данным, которые отражены в соответствующей подсхеме, что является одним из способов защиты информации от несанкционированного доступа к данным. В подсхеме могут также задаваться допустимые режимы обработки, что служит дополнительным механизмом защиты информации от разрушения.

В тех случаях, когда СУБД не поддерживает подсхемы, перечисленные функции могут выполнять другие компоненты системы. Близким к понятию подсхем является понятие "представление".

ПП1

ПП2

ПП3

ПП4

ПП5

Подсхема А

Подсхема В

Схема

Физическое описание БД

Физическая БД

Отображение подсхемы в схему

Отображение на физическую память

Поддерживается СУБД

2.2.3 Работа СУБД

Прикладная программа А

Рабочая

область

Системные

буферы

Операционная

система

СУБД

Подсхемная исполняемая

программа А

Схема

Описание

физической

организации БД

Физическая БД

1

2

3

4

5

6

7

9

10

11

На рисунке представлена последовательность основных действий, реализуемых СУБД в процессе считывания записи для прикладной программы.

1. Прикладная программа А выдает запрос СУБД на чтение записи.
2. СУБД получает в распоряжение подсхему, исполняемую программой А, и осуществляет в ней поиск описания данных, на которые выдан запрос.
3. СУБД получает в распоряжение схему (глобальное логическое описание данных) и с ее помощью определяет необходимый тип логических данных.
4. СУБД просматривает описание физической организации БД и определяет, какую физическую запись требуется считать.
5. СУБД выдает операционной системе команду чтения требуемой записи.
6. Операционная система взаимодействует с физической памятью, в которой хранятся данные.
7. Запрошенные данные передаются из памяти в системный буфер.
8. СУБД осуществляет сравнение схемы и подсхемы, выделяет ту логическую запись, которая запрошена прикладной программой. Любое преобразование данных, необходимость которого возникает из-за различий в описании одних и тех же данных в схеме и подсхеме, выполняется СУБД.
9. СУБД передает данные из системного буфера в рабочую область прикладной программы А.
10. СУБД передает прикладной программе информацию о результатах выполнения различных процедур по обслуживанию ее запросов. Эта информация содержит также сведения о возможных ошибках.
11. Прикладная программа обрабатывает данные, помещенные в рабочую область.

2.3 Организация данных

2.3.1 Физическая организация данных

Проблема физической организации данных: каким образом можно представить структуры данных в памяти компьютера в виде последовательной цепи?

Аспекты проблемы:

1. Как можно найти необходимую запись среди множества данных? Группа битов, которую можно прочитать с помощью одной машинной инструкции, называется одной физической записью. Программы идентифицируют логическую запись с помощью ключа.

Ключ→ машинный адрес

Переход от ключа к адресу определяет сущность способа адресации, которые будут рассмотрены ниже.

1. Каким образом организовать данные, чтобы поиск был эффективным, а выборку записей можно было бы осуществлять по нескольким ключам?
2. Каким образом древовидные сетевые структуры можно представить в виде последовательности битов?
3. Как добавить новую запись к данным, уничтожить старые записи, не нарушая структуры адресации и поиска, а также структуру данных?

Ключ БД — RID (Record IDentificator)

Каждой записи СУБД присваивается внутренний идентификатор. Ключ БД не следует приравнивать ключу записи. Если значение ключа записей задается пользователем, то RID устанавливается системой при размещении.

Пример:

1. RID состоит из номера страницы и номера записи на странице;
2. Последовательный номер записи в файле.

### В некоторых системах пользователь не знает RID, в других — он доступен пользователю, который может его использовать.

При описании типа записи (имя таблицы) администратором БД определяется область хранения. ОХ — участок памяти, т.е. совокупность страниц, внутри которого могут размещаться записи описываемого типа. Вне области хранения записи размещаться не могут.

В некоторых системах для любой записи используется своя область хранения, в других они могут пересекаться, и тогда на одной странице могут оказаться записи разных типов.

Характеристики файла данных: Параметры блокирования страниц

##### Формат и длина страницы

Запись может быть сегментирована, если она не умещается на одной странице.

Файл БД:


#### Как по первичному ключу определить месторасположение записи с данным ключом?

1. Последовательное сканирование файла — сканируется файл с проверкой ключа каждой записи. Эффективен только для файлов с последовательным доступом. Записи должны быть упорядочены по ключу.
2. Блочный поиск — записи упорядочиваются по ключу. И при сканировании файла можно рассматривать не каждую запись, а каждую сотую в последовательности возрастающих ключей. Затем область поиска сужается.
3. Двоичный поиск — рассматривается запись в середине области, в которой выполняется поиск и ее запись сравнивается с поисковым ключом. Делится пополам.
4. Индекс на последовательный файл — последовательное сканирование файла для нахождения записи требует много времени, если оно выполняется над всем файлом. Но сканирование можно использовать для небольшой области, которая представляет весь файл. Пусть файл упорядочен по ключу. Для адресации к такому упорядоченному файлу используется таблица, называемая индексом.
5. Индекс на произвольные файлы — записи располагаются в произвольной последовательности, как правило, в порядке ввода в БД. Непоследовательный файл можно индексировать точно также, как и последовательный. Но для этого требуется значительно больший по размеру индекс, т.к. он должен содержать по одному элементу для каждой записи файла, а не для блока записей. Кроме того, в нем должны содержаться полные абсолютные адреса.
6. Адресация с помощью ключа, эквивалентного адресу. Известны много методов преобразования ключа непосредственно в адрес в файле. Когда такое преобразование возможно, оно обеспечивает самую быструю адресацию. Например, в некоторых банковских системах номера счетов изменялись так, чтобы номер счета или его часть являлись бы адресом записи в БД.
7. Алгоритм преобразования ключа в адрес дает почти ту же скорость, что и предыдущий. Но характеризуется неэффективным использованием памяти. Поскольку ключи преобразуются в непрерывное множество адресов, в файле остаются свободные участки.
8. Хеширование. При запоминании новой записи специальная программа ставит в соответствие значению первичного ключа номер страницы, куда следует поместить запись. Таким образом, перемешиваются по страницам все записи. Записи, для которых хеширование выдает одинаковые страницы, называются синонимами. Синонимы объединяются в связанный список, начало которого находится в заголовке таблицы. Если при заполнении очередного синонима для него не хватило места на одной странице, он размещается на другой странице, но включается в цепь синонимов той области, куда был хеширован. При извлечении записи по первичному ключу СУБД подключает программу хеширования, вычисляет номер страницы, а затем просматривает цепь синонимов на этой странице до нахождения нужной записи.

2.3.2 Организация индексных таблиц

Индекс можно определять как таблицу, воспринимающую на входе информацию о некоторых значениях атрибутов и выдающую на выходе информацию, соответствующую быстрому поиску записей, которые имеют заданное значение атрибутов.

Индексная таблица должна быть упорядочена по входному индексу.

Следует различать индексирование по первичному ключу и вторичное индексирование.

Первичное индексирование

СУБД, размещая записи на странице данных, формирует специальные индексные таблицы. Они используются для нахождения адреса записи по значению первичного ключа. При этом СУБД производит поиск записи не в файле, а в индексе. Если записи файла упорядочены по ключу, индекс обычно содержит не ссылки на каждую запись, а ссылки на блоки записи, внутри которых можно выполнять поиск или сканирование.

Рассмотрим один из известных механизмов индексирования. Пусть записи имеют ключи со значениями из натурального ряда. Перед загрузкой записи должны быть упорядочены по значению ключей.

Особенности первичного индексирования:

1. Ключ индексирования должен иметь уникальное неизменное значение;
2. Первоначальная загрузка должна выполняться обязательно с предварительной сортировкой;
3. Формирование индексной таблицы происходит одновременно с загрузкой записей файлов, а значение ключей индексирования влияют на размещение записей в памяти.

Вторичное индексирование

При обработке БД возникает потребность извлечения записи по значению данных, отличных от первичного ключа, например, извлечь данные из записи о студентах данной специальности или по специальности — номер группы. Последовательный просмотр большого числа записей может оказаться неэффективным. Другой вариант состоит в использовании вторичного индексирования.

Вторичные ключи могут идентифицировать записи неединственным образом. Одному вторичному ключу могут соответствовать несколько записей

Входной индекс

Вторичный ключ

Выходной индекс

КБД, первичный ключ

КБД — ключ БД

Особенности вторичного индексирования:

1. Вторичные ключи могут иметь неуникальное значение. Допускается изменение значений вторичных ключей. При этом системой автоматически будут корректироваться соответствующие таблицы.
2. Вторичное индексирование служит только для выборки данных и никак не используется при размещении данных в памяти.
3. Таблицы вторичного индексирования можно создавать не только при первоначальной загрузке, но и в любое другое время.

Организация индексных таблиц должна обеспечивать быстрое нахождение адреса записи по значению ключа. Один из механизмов индексирования нам уже известен, это иерархия индексных таблиц с последовательным просмотром внутри таблицы. Этот механизм работает только если записи перед загрузкой накапливаются и упорядочиваются. Если же загрузка БД происходит в реальном времени, то этот механизм малоэффективен. Рассмотрим другие механизмы индексирования.

Бинарное деление

Индексная таблица при этом механизме индексирования состоит из множества записей, связанных адресными ссылками в иерархическую структуру. Такие записи называют индексными.

Каждая индексная запись (ИЗ) содержит

1. Значение ключа записи;
2. Адрес этой записи (ключ БД);
3. Адресная ссылка на ИЗ с ключом меньше данного;
4. Адресная ссылка на ИЗ с ключом больше данного.

При этом механизму не требуется, чтобы записи предварительно накапливались и упорядочивались. Каждой записи БД соответствует одна ИЗ. Первая ИЗ становится корневой.

Пример:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # записи | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Ключ записи | 12 | 8 | 4 | 9 | 6 | 13 | 14 | 16 | 100 | 10 | 25 | 31 |

Поиск записи по значению ключа выполняется по тому же алгоритму, начиная с корня БД. Механизм основан на известном дихотомическом поиске. Наряду с достоинствами, бинарные деревья имеют значительные недостатки. В нашем примере для извлечения записи с ключом записи, равным 31, требуется 7 шагов, а для записи, с КЗ=6 — 4 шага, хотя обе записи конечны. Такое дерево называется несбалансированным.

Сбалансированные деревья — деревья, в которых все конечные вершины равноудалены от корня. Если бинарное дерево растет вниз и корень неизменен, то В-дерево растет вверх и корень меняется.

В-дерево n-ого порядка должно удовлетворять следующим условиям:

1. Любая вершина может содержать n адресных ссылок и n-1 ключей. Ссылка влево от ключа обеспечивает переход к вершинам дерева с меньшими значениями ключа, вправо — с большими.
2. Любая неконечная вершина может иметь не менее n/2 подчиненных вершин.
3. Если неконечные вершины содержат k ключей, то им подчинена k+1 вершина на следующем уровне иерархии.
4. Все конечные вершины В-дерева расположены на одном уровне.

Алгоритм формирования В-дерева предполагает первоначальное заполнение корня до тех пор, пока не будут сформированы все ее n-1 ключей. Затем при появлении очередной записи выделяется новая корневая вершина и несколько подчиненных ей вершин. При запоминании нового ключа поиск места для него начинается с корня В-дерева. При этом используется алгоритм, рассмотренный для бинарного дерева.

Пример:

Пусть сгружается та же последовательность записей с формированием бинарного дерева с n=3. Тогда каждая запись такого дерева рассчитана на хранение 2-х ключей и 3-х ссылок. Первоначально корневая вершина содержит только ключ 12.



Затем он сдвигается вправо, уступая место ключу 8, так как внутри записи ключи расположены в порядке возрастания значений.



Далее, поскольку для ключа 4 нет записи в корневой записи, происходит ее деление. Из трех ключей (4, 8, 12) выбирается средний ключ (8), чтобы ключи 4 и 12 попали в две подчиненные записи.



При этом будет выполнено ограничение 2) и 3). Это общее правило для деревьев 3-го порядка.

В итоге получим следующее дерево:



Связанные списки

В сетевых иерархических моделях данных связь с данными

C4

C2

C3

C1

A

B

M

H

E

E1 → E2 → E3  E4 E5→ E6 E7→ E8

 H1 H2 H3 H4

 C4 C4 C4 C4

A1

C3

 B1 B2 B3

C1

M1 → M2 M3→ M4 → M5

 C2 C2 C2

поддерживается групповыми отношениями. Наиболее распространенный способ реализации группового отношения — построение связанных списков.

Последняя подчиненная запись содержит либо ссылку на владельца, либо признак конца цепи (замкнутый или разомкнутый список). Рассмотренный тип ссылок называется ссылками на следующий. В цепном связанном списке могут использоваться и другие типы ссылок:

* ссылка на владельца обеспечивает движение по групповому отношению вверх;
* ссылка на предыдущий обеспечивает просмотр в обратном направлении и повышает эффективность процедур удаления записей из группового отношения. Ссылка на предыдущий формируется у всех участников списка, включая запись владельца. Владелец при этом будет содержать кроме ссылки на первую подчиненную запись еще и ссылку на последнюю.

Пусть, например, необходимо удалить В2. Удаление будет корректным, если В1 будет указывать на В3. Для этого необходимо сделать шаг назад, для чего и нужна ссылка на предыдущий узел.

Существует несколько видов отображения групповых отношений на физическую память. Чаще всего — левосторонний обход структуры дерева, за последней записью первого дерева размещается корень следующего и т.д.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Стр.1 | А1 | В1 | М1 | М2 | В2 |
| Стр.2 | В3 | М3 | М4 | М5 | Н1 |

2.4 Обновление и восстановление данных

При динамической реорганизации страниц записи плотно размещаются в начале каждой страницы, а в конце расположен свободный участок. Тогда адрес начала свободного участка хранится в начале таблицы.

При удалении некоторой записи таблица сжимается и увеличивается длина свободного участка. Для учета свободных участков на странице СУБД поддерживает инвентарные страницы. Одна инвентарная страница создается для группы страниц данных и содержит сведения о наличии на них свободных участков памяти.

Запоминание новых записей

При запоминании СУБД ищет место для записи в области хранения. Поиск ведется сначала через инвентарные страницы, а затем — на страницах данных. Если для учета свободных участков используется цепи участков, то СУБД выбирает первый свободный участок, пригодный по длине. Если длина записи меньше длины свободного участка, то участок оформляется в виде нового свободного участка. В результате может образоваться несколько небольших свободных участков, не позволяющих запомнить новую запись.

При динамической реорганизации страниц такого случиться не может, так как на каждой странице поддерживается один свободный участок. После запоминания записи СУБД корректирует содержимое соответствующей инвентарной страницы.

Корректировка

Если элементы данных имеют фиксированную длину, то обновленные значения помещаются на место прежних. Если система допускает переменную длину, то обновленная запись может иметь длину отличную от прежней. В этом случае либо реорганизуется страница, либо перемещается запись на другой участок памяти.

Транзакция

Восстановление данных в СУБД означает возвращение БД в согласованное, непротиворечивое состояние, если какой-нибудь сбой сделал текущее состояние противоречивым. Принцип, на котором строится такое восстановление, это избыточность, которая образуется на физическом уровне. Если любая часть информации содержащейся в БД, может быть восстановлена из другой (хранимой в БД или извне), то такая БД восстановимая. Проблемы восстановления и параллелизма являются аспектами одной проблемы — проблемы транзакции.

Транзакция — это последовательность операций над БД, рассматриваемая СУБД как единое целое.

Особенности транзакции:

1. Всегда связана с изменениями в БД, вызываемых операциями INSERT, DELETE, ACCEPT в SQL;
2. Транзакция — логически связанная последовательность одной или нескольких таких операций, которые преобразуют одно непротиворечивое состояние БД в другое, но не гарантируют этого в промежуточные моменты времени.

Пример:

Пусть мы хотим изменить значение первичного ключа какого-либо кортежа. Для этого используется команда UPDATE. По ней за один раз можно обновить только одну таблицу. Поэтому одной операцией UPDATE не обойтись, так как кортеж — это этап в других отношениях. В данном примере мы сталкиваемся с проблемой целостности по ссылкам. БД становится противоречивой после выполнения первой команды UPDATE. Тогда транзакция будет включать в себя столько операций UPDATE, сколько раз этот кортеж входит в другие отношения.

Что будет, если внутри транзакции будет аварийный отказ? Все зависит от того, поддерживает ли система обработку транзакций. Система, обрабатывающая транзакции, гарантирует, что если в транзакцию входит обновление базы данных, а затем произошла ошибка (до того, как транзакция достигла нормального завершения), то эти обновления будут аннулированы.

Таким образом, возможны два исхода выполнения транзакции:

1. Транзакция полностью исполняется;
2. Транзакция полностью аннулируется.

С точки зрения конечного пользователя транзакция кажется атомарной — ее обслуживают системные компоненты — монитор (диспетчер) транзакций (МТ). МТ не является частью СУБД, наоборот, СУБД подчиняется МТ. МТ определяет две команды:

1. COMMIT — фиксация;
2. ROLLBACK — откат.

В зависимости от типа МТ они могут выглядеть по-разному.

Точка синхронизации (ТС) представляет собой граничную точку между двумя последовательными транзакциями. ТС устанавливается при инициализации программы и при выполнении команд МТ:

1. COMMIT — сообщает об успешной транзакции и устанавливает ТС. Все обновления, сделанные транзакцией, фиксируются. Снимаются все блокировки записей, все открытые курсоры закрываются.
2. ROLLBACK — сообщает о неудачном завершении транзакции и устанавливает ТС. Все обновления программы (после установки последней ТС) аннулируются. Снимаются все блокировки записей, все открытые курсоры закрываются.

Восстановление транзакции

Производится в ситуациях:

1. Индивидуальный откат транзакции;
2. явное завершение оператором ROLLBACK,
3. откат производится самой системой (выбор транзакции как "жертвы" в синхронизационном тупике);
4. Восстановление после внезапной потери содержимого ОЗУ (мягкий сбой);
5. Восстановление после поломки основного внешнего носителя (жесткий сбой).

Во всех трех случаях основа восстановления — избыточное хранение данных. Они хранятся в журнале, содержащем последовательность записей об изменении БД. Журнализация изменений часто связана не только с управлением транзакциями, но и с буферизацией страниц БД в оперативной памяти.

Если бы запись об изменении БД сразу записывалась во внешнюю память, это привело бы к замедлению работы системы. Поэтому записи журнала тоже буферизируются.

Существуют два вида буферов:

1. Буфер журнала;
2. Буфер страниц БД.

Оба буфера всегда восстанавливаются во внешнюю память. Проблема состоит в выработке восстановления буфера во внешнюю память. Основным принципом такого восстановления является то, что запись об изменении объекта БД должна попадать во внешнюю память раньше, чем измененный объект помещается во внешнюю память. Такой протокол называется "пиши сначала в журнал"

t1

t0

Внешняя память

Журнал

Данные

Буфера ОЗУ

Журнал

Данные

Транзакция

Операция модификации

Модификация БД

Индивидуальный откат транзакции

1. Выбирается очередная запись из списка данной транзакции;
2. Выбирается противоположная по смыслу операция;
3. Любая из этих обратных операций также журнализируется;
4. При успешном завершении отката заносится в журнал запись о конце транзакции.

Восстановление после мягкого сбоя

1. Страницы БД буферизируются в ОЗУ и восстанавливаются независимо;
2. Несмотря на применение WAL после мягкого сбоя набор атрибутов БД во внешней памяти может оказаться несогласованным, т.е. часть страницы во внешней памяти соответствует БД до изменения, часть — после изменения.

Состояние внешней памяти БД называется физически согласованным, если наборы всех страниц согласованны, т.е. соответствуют состоянию объекта либо до, либо после изменений.

Будем считать, что в журнале отмечаются точки согласованности БД, т.е. моменты времени, в которые во внешней памяти создаются согласованные результаты операций, завершенных до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память. Тогда к моменту мягкого сбоя будут следующие состояния транзакций:

Внешняя память

Т1

Т2

Т3

Т4

Т5

Журнал

Пусть удалось восстановить внешнюю память БД к согласованному состоянию и времени tфс, тогда для:

T1 — никаких действий не производить.

T2 — нужно повторно выполнить оставшуюся часть операций, так как во внешней памяти полностью отсутствуют следы операций, которые выполнялись транзакцией после времени tфс.

Это приведет к согласованному состоянию БД, так как транзакция Т2 успешно завершилась до момента мягкого сбоя и в журнале содержатся все действия (REDO).

T3 — выполняется в обратном направлении первая часть операций (UNDO).

Во внешней памяти БД полностью отсутствуют результаты операций транзакций Т3, которые были выполнены после tфс. С другой стороны, во внешней памяти существуют результаты операций Т3, которые были выполнены до tфс.

Т4 — выполняются повторно полностью все операции (REDO).

Т5 — никакие действия не предпринимаются.

Во внешней памяти БД полностью отсутствуют результаты операций транзакции Т5.

Восстановление после жесткого сбоя

При восстановлении последнего состояния БД после жесткого сбоя журналы и данные явно недостаточны. Основой восстановления в этом случае является журнал и архивные копии БД. Восстановление начинается с обратного копирования БД из архивной копии. Затем для всех закончившихся транзакций выполняется REDO, т.е. операции повторно выполняются в прямом смысле. Более точно происходит следующее:

1. По журналу в прямом направлении выполняются все операции;
2. Для транзакций, которые не закончились к моменту сбоя, выполняется откат.

Так как жесткий сбой не сопровождается утратой буферов ОЗУ, можно восстановить БД до такого уровня, что даже можно выполнить незавершенные транзакции.

Хотя к ведению журнала предъявляются особые требования по части надежности, возможна его утрата. Тогда единственным способ восстановления БД является возврат к архивной копии. В этом случае не удается получить последнее согласованное состояние БД.

2.5 БД в сетях

По характеру организации хранения данных и обращения к ним различают персональные (локальные), централизованные и распределенные базы данных.

1. Персональные — все части СУБД размещаются на компьютере пользователя локальной БД. Если к одной БД обращаются несколько пользователей одновременно, то каждый пользователь должен иметь свою копию. Этот вариант в корпоративной работе практически не встречается, т.к. в нем чрезвычайно трудно синхронизировать содержимое нескольких копий БД. Это проблема согласования (репликации) БД.
2. Централизованные — такие БД, которые хранятся на одном компьютере, находящемся в узле сети, с помощью которой различные подразделения получают доступ к данным. Централизованные базы данных обычно организуются в рамках локальной вычислительной сети.
3. Распределенные — такие БД, в которых данные распространяются по сети.

Приведенные понятия следует отличать от понятия распределенной обработки данных, которая может быть организована и при централизованном хранении БД. Пусть большая БД расположена на мощном компьютере. Можно организовать доступ к этой БД из других компьютеров, подключенных к сети. При этом выборка и предварительная обработка данных будет выполняться мощной машиной, а окончательная обработка и представление данных — менее мощным ПК пользователя-клиента.

Архитектура взаимодействия клиента и сервера

БД

ЯДРО

ПРОГРАММА

Клиент 2

Клиент 1

ФАЙЛ-СЕРВЕР

БД

ЯДРО

ПРОГРАММА

ЯДРО

ПРОГРАММА

1. Локальные и файл-серверные БД.

В каждый момент времени клиент работает с некоторой локальной копией БД, причем управление данными целиком возлагается на клиентские программы. Именно они должны заботиться о синхронизации копий данных на каждой машине. В обоих случаях ядро располагается на машине клиента и вместе с программой образует локальную СУБД, количество которых равно количеству пользователей.

Архитектура "файл-сервер" обладает следующими недостатками:

1. Вся тяжесть вычислительной работы ложится на компьютер клиента, например, если в результате запроса клиент должен получить две записи из таблицы объемом 100.000 записей все они будут скопированы с файл-сервера на ПК клиента.
2. Так как БД — набор файлов на сетевом сервере, то доступ к таблице регулируется только сетевой ОС, что делает такие БД беззащитными.
3. Бизнес-правила в системах файл-сервера реализуются в программе клиента, что не исключает проектирование противоречащих друг другу бизнес-правил в различных программах.
4. Недостаточно развитый аппарат транзакции локальной СУБД служит потенциальным источником ошибок при одновременном внесении изменений.
5. Клиент-серверные БД.

Клиент 2

Клиент 1

МАШИНА СЕРВЕРА

БД

ЯДРО

ПРОГРАММА

ЯДРО

ПРОГРАММА

СЕРВЕР БД

В архитектуре "клиент-сервер" между ядром и БД появляется сервер БД.

Сервер БД — специальная программа, управляющая БД. Клиент формирует запрос к серверу на языке SQL. Большинство существующих серверов умеют обрабатывать язык SQL.

SQL-сервер обеспечивает интерпретацию и выполнение запроса, формирование результата и выдачу этого результата клиенту.

При этом сам клиент не участвует в физическом выполнении запроса. ПК клиента лишь посылает запрос серверу и получает результат. В итоге снижается нагрузка на сеть, т.к. по сети передаются только те данные, которые нужны клиенту. Обычно такой сервер работает в среде многозадачной ОС.

1. Трехзвенная архитектура клиент-сервер.

Теперь клиентские машины могут не иметь ядра, а клиентские программы уже не включают в себя громоздкие коды компонентов-источников. Реализация большей части клиентских бизнес-правил переводится на сервер приложений. Поэтому такая клиентская программа называется облегченным, или тонким, клиентом. Такой клиент не требует больших ресурсов памяти и может загружаться с сервера. Это главное преимущество трехзвенной архитектуры.

На рисунке показан вариант размещения сервера приложений на машине сервера БД. Это наиболее популярный вариант, но не обязательный. Сервер приложений может размещаться на любой машине, оснащенной ядром

МАШИНА СЕРВЕРА

БД

СЕРВЕР БД

ЯДРО

СЕРВЕР

ПРИЛОЖЕНИЙ

ПРИЛОЖЕНИЕ 1

ПРИЛОЖЕНИЕ N

2.6 Доступ к данным в Windows

На сегодняшний день существуют две параллельно развивающиеся конкурирующие технологии взаимодействия между объектами и программами. Это COM (Component Object Model) и CORBA (Command Object Require Broker Architecture). COM развивается компанией Microsoft, CORBA — другими.

Технология COM

Предназначена для того, чтобы одна программа (клиент) смогла заставить работать объект, который является частью другой программы (сервера) так, как если бы этот объект был частью клиента, причем обе программы в общем случае могут быть:

1. Расположены на разных машинах;
2. Написаны на разных языках;
3. Использоваться в разных ОС;
4. Располагаться на машинах разного типа.

Ключевое понятие COM — это интерфейс. Интерфейс имеет уникальный идентификатор и набор параметров, описывающих методы, события и свойства общего объекта. Идентификатор интерфейса является частным случаем глобального системного идентификатора. В Windows входят функции, генерирующие эти идентификаторы. Вероятность совпадения двух идентификаторов ничтожно мала.

Параметры идентификатора в общем случае описывают типы и имена используемых полей, количество и типы параметров обращения к доступным методам и свойствам, имена методов и свойства и т.д. Получив идентификатор внешнего СОМ-объекта, клиент может его использовать также, как и свои собственные.

Сервер СОМ представляет собой исполняемую программу или .DLL, содержащую один или несколько объектов СОМ. В зависимости от местоположения клиента и сервера возможны 3 варианта:

1. Клиент и сервер расположены на одной машине и запускаются в одном процессе. В этом случае сервер представляет собой .DLL.

Клиент с помощью интерфейса объекта непосредственно обращается к методам объектов в своем собственном адресном пространстве.

Процесс клиент

СЕРВЕР

СОМ объект

КЛИЕНТ

1. Клиент и сервер располагаются на одной машине, но запускаются в разных процессах. Например, таблица Excel вставляется в документ Word. В этом случае сервер представляет собой программу.

COM

Процесс клиента

КЛИЕНТ

PROXY

ПРОЦЕСС СЕРВЕРА

STUB

Объект

1. Клиент и сервер располагаются на разных машинах. Сервером может быть как программа, так и .DLL. В этом случае используют распространенный вариант COM — DCOM.

УДАЛЕННАЯ МАШИНА

DCOM

Процесс клиента

КЛИЕНТ

PROXY

ПРОЦЕСС СЕРВЕРА

DCOM

Сервер

STUB

Объект

Если сервер запускается в другом процессе или на другой машине, между объектом и клиентом располагаются два посредника: Proxy (уполномоченный) и Stub (заглушка).

Клиент посещает параметры вызова в стек и обращается к методу интерфейса объекта. Это обращение перехватывается Proxy, упаковывает параметры вызова в пакет СОМ и направляет его в Stub другого процесса. Stub распаковывает параметры, помещает в стек и делает вызов нужного метода объекта. Таким образом, метод объекта выполняется в собственном адресном пространстве процесса сервера.

Понятие открытого интерфейса Windows

Прикладная

программа

API

Windows

SPI

Источник

данных

Для унификации связи Microsoft поддерживает так называемую открытую архитектуру. Целью открытой архитектуры является соединение ПК и информационными службами. WOSA представляет собой некоторый изолирующий слой между прикладной программой и источником данных.

Такой архитектурой определены два стандартных интерфейса:

1. API — интерфейс прикладных программ. Разработчики приложений используют только этот интерфейс для доступа к любому числу информационных служб.
2. SPI — интерфейс, обеспечения информационных служб. Поставщики служб пишут драйверы, обеспечивающие доступ к этим службам.

Преимущества:

1. Существует доступ пользователя к информационным службам без изучения различных приложений;
2. Поставщики могут сделать доступными свои приложения для большого числа пользователей;
3. Разработчики. Их приложения получают новые возможности без использования различных интерфейсов.

# Глава 3. Работа с таблицами базы данных на примере СУБД Microsoft Access

Таблицы – фундаментальные объекты реляционной базы данных, в которых хранится основная часть данных приложения. Отдельная таблица чаще всего хранит информацию по конкретной теме (например, сведения о служащих компании или адреса заказчиков). Информация в таблице организуется в строки (записи) и столбцы (поля). Таблице присущи два компонента: структура таблицы и данные таблицы.

## 3.1 Структура таблицы, ее создание

Структура таблицы (также называется определением таблицы) специфицируется при создании таблицы. Структура таблицы должна быть спроектирована и создана перед вводом в таблицу каких-либо данных. Она определяет, какие данные таблица будет хранить, а также правила, ассоциированные с вводом, изменением или удалением данных (бизнес-правила, или ограничения). Структура таблицы доступна через окно конструктора таблиц. Чтобы открыть это окно для существующей таблицы, нужно открыть вкладку Таблицы окна базы данных, выбрать таблицу и нажать кнопку Конструктор.

Структура таблицы включает следующую информацию:

|  |  |
| --- | --- |
| Имя таблицы | Имя, по которому к таблице можно обратиться в свойствах, методах и операторах SQL. |
| Столбцы таблицы | Категории информации, сохраненной в таблице. Каждый столбец имеет имя и тип данного. |
| Табличные и столбовые ограничения | Ограничения целостности, определенные на уровне таблицы или на уровне столбца. |

Окно конструктора таблиц используется как для определения структуры таблицы при ее создании, так и для последующего изменения структуры таблицы. Эти операции обсуждаются ниже.

Данные таблицы – информация, которая сохранена в таблице. Все данные таблицы хранятся в строках, каждая из которых содержит порции информации в столбцах, определенных в структуре таблицы. Данные – та часть таблицы, к которой обычно должны иметь доступ пользователи приложения (например, данные таблицы могут выводиться в элементах управления, размещенных в формах и отчетах, либо предоставляться в режиме таблицы).

Так как таблицы – объекты, которые будут хранить большую часть данных приложения, подходить к проектированию таблиц необходимо со всей тщательностью. Правильная разработка таблиц включает много аспектов, которые достойны глубокого рассмотрения. Ниже приводятся некоторые базовые принципы разработки таблиц.

1. Необходимо избегать дублирования информации. Для каждой категории информации следует использовать отдельную таблицу. Например, не стоит хранить описания отделов в таблице, которая хранит информацию о служащих. Процесс проектирования таблиц, позволяющий исключать дублирование данных, называется нормализацией. Нормализация также позволяет экономить пространство базы данных и помогает предотвратить ошибки, которые могут возникать при наличии дублированной информации. В Access процедуры нормализации помогает выполнить мастер анализа.
2. Не следует хранить значения, которые могут быть легко вычислены из существующих значений. Например, не нужно хранить сумму всех позиций товарного заказа, так как ее можно вычислить с помощью простой формулы.
3. Для полей необходимо выбирать подходящий тип данных. Это поможет уменьшить размеры базы данных и увеличит скорость выполнения операций. При описании поля следует устанавливать для него тип данных наименьшего размера, позволяющий хранить нужные данные.
4. В каждой создаваемой таблице нужно стараться включать столбец или набор столбцов первичного ключа. Первичные ключи необходимы для установления между таблицами отношений один-к-многим. Кроме того, многие базы данных поддерживают ограничения по первичному ключу, используя индекс, который может значительно повышать скорость поиска и операций сортировки. В составном ключе, включающем несколько полей, нужно использовать ровно столько полей, сколько для него необходимо.

Имеются специальные случаи, когда первичный ключ целесообразнее не создавать. Например, для некоторых таблиц индекс, ассоциированный с первичным ключом, может неприемлемо снижать производительность ввода и модификации данных. После того как проект таблицы готов, можно приступать к ее созданию. Создать таблицу можно двумя способами. Для ввода новых данных можно создать пустую таблицу. Можно также создать таблицу, используя уже существующие данные из другого источника.

### 3.1.1 Создание новой пустой таблицы

В Microsoft Access существует несколько способов создания новой таблицы:

1. Использование мастера баз данных для создания новой стандартной базы данных из числа предоставляемых Access. Созданная за одну операцию база данных будет содержать все требуемые отчеты, таблицы и формы. Мастер баз данных создает новую базу данных целиком, его нельзя использовать для добавления новых таблиц, форм и отчетов в уже существующую базу данных.
2. Мастер таблиц позволяет выбрать поля для данной таблицы из числа определенных ранее таблиц.
3. Ввод данных непосредственно в пустую таблицу в режиме таблицы. При сохранении новой таблицы в Microsoft Access данные анализируются и каждому полю присваивается необходимый тип данных и формат.
4. Определение всех параметров структуры таблицы в режиме конструктора.
5. Импорт в текущую базу данных структур таблиц и данных из внешнего источника.
6. Создание в текущей базе данных таблиц, связанных с таблицами внешнего источника.

Независимо от метода, примененного для создания таблицы, всегда имеется возможность использовать режим конструктора для дальнейшего изменения структуры таблицы, например для добавления новых полей, установки значений по умолчанию или для создания масок ввода.

### 3.1.2 Создание таблицы в режиме конструктора

Режим конструктора позволяет наиболее гибко управлять всеми создаваемыми и уже имеющимися компонентами определения таблицы. Чтобы создать таблицу в режиме конструктора:

1. Перейти в окно базы данных. Переключиться из другого окна в окно базы данных можно, нажав клавишу "F 11".
2. Выбрав вкладку Таблицы, нажать кнопку Создать.

Прошу обратить внимание: Если данная таблица является связанной, то добавлять в нее новое поле в текущей базе данных невозможно. Если связанная таблица является таблицей Microsoft Access, для добавления поля необходимо открыть исходную базу данных. Если связанная таблица является таблицей другого приложения, то для добавления поля необходимо открыть исходный файл в этом приложении.

1. В окне Новая таблица выбрать элемент Конструктор.
2. Определить в таблице каждое поле. Поля таблицы содержат данные, представляющие порции записи. Пользователь имеет возможность определять формат отображения данных, указывать значения по умолчанию и ускорять операции поиска и сортировки, задавая значения свойств полей в разделе Свойства поля в режиме конструктора таблицы. В Microsoft Access свойства полей используются при просмотре или изменении данных пользователем. Например, заданные пользователем значения свойств Формат поля, Маска ввода и Подпись определяют вид базы данных таблицы и запроса. Элементы управления в новых формах и отчетах, присоединенные к полям таблицы, наследуют эти свойства полей базовой таблицы по умолчанию. Другие свойства позволяют определить условия на значения полей или задать обязательный ввод данных в поле. Microsoft Access будет проверять выполнение этих условий при каждом добавлении или изменении данных в таблице. Для добавления поля в конец структуры таблицы нужно выбрать первую пустую строку структуры. Для вставки поля в середину структуры следует выбрать строку, над которой требуется добавить новое поле, и нажать кнопку Добавить строки на панели инструментов. В столбец Имя столбца ввести имя поля; в столбце Тип данных выбрать требуемый тип данных в раскрывающемся списке либо оставить настройку по умолчанию (Текстовый). В столбце Описание можно ввести необязательное краткое описание поля. Текст описания будет выводится в строке состояния при добавлении данных в поле, а также будет включен в описание объекта таблицы. При необходимости можно задать значения свойств поля в бланке свойств в нижней части окна.
3. Назначить ключевые поля таблицы. Наличие в таблице ключевых полей не обязательно. Однако если они не были определены, то при сохранении таблицы выдается вопрос, нужно ли их создавать.
4. Для сохранения таблицы нажать кнопку Сохранить на панели инструментов, введя допустимое имя таблицы.

Выбор для поля таблицы типа данного

Тип данного поля таблицы можно выбрать в раскрывающемся списке в столбце Тип данных. При выборе типа данных, используемых в поле, необходимо учитывать следующее:

1. какие значения должны отображаться в поле. Например, нельзя хранить текст в поле, имеющем числовой тип данных, и нецелесообразно хранить числовые данные в текстовом виде;
2. сколько места необходимо для хранения значений в поле;
3. какие операции должны производиться со значениями в поле. Например, суммировать
4. значения можно в числовых полях и в полях, имеющих валютный формат, а в текстовых полях и полях объектов OLE, – нельзя;
5. нужна ли сортировка или индексирование поля. Сортировать и индексировать поля МЕМО, гиперссылки и объекты OLE невозможно;
6. будут ли поля использоваться в группировке записей в запросах или отчетах. Поля МЕМО, гиперссылки и объекты OLE использовать для группировки записей нельзя;
7. каким образом должны быть отсортированы значения в поле. Числа в текстовых полях сортируются как строковые значения (1, 10, 100, 2, 20, 200 и т. д.), а не как числовые значения. Для сортировки чисел как числовых значений необходимо использовать числовые поля или поля, имеющие денежный формат. Также многие форматы дат невозможно надлежащим образом отсортировать, если они введены в текстовое поле. Для обеспечения сортировки дат и времен следует использовать поле типа Дата/Время.

В следующей таблице представлены все типы данных Microsoft Access и их применение.

|  |  |  |
| --- | --- | --- |
| Тип данных | Применение | Размер |
| Текстовый | Текст или комбинация текста и чисел, например адрес, а также числа, не требующие вычислений, например номера телефонов, номенклатурные номера или почтовый индекс. | До 255 символов. Хранятся только введенные в поле символы; позиции, не использованные в текстовом поле, не хранятся. Для управления максимальным числом вводимых символов следует определить свойство Размер поля. |
| ПолеМЕМО | Длинный текст или числа, например комментарии или описание. Поля МЕМО не могут быть индексированы или отсортированы. Для хранения форматированного текста или длинных документов, вместо поля МЕМО, следует создать поле объекта OLE. | До 64 000 символов. |
| Числовой | Числовые данные, используемые для математических вычислений, за исключением вычислений, включающих денежные операции (для которых используется денежный тип). Тип и размер значений, которые могут находиться в числовом поле, можно изменить в свойстве Размер поля. Например, в поле, занимающее на диске 1 байт, допускается ввод только целых чисел (без десятичных знаков) от 0 до 255. | 1, 2, 4 или 8 байт. 16 байт только для кодов репликации. |
| Дата/Время | Даты и время. Хранение значений дат и времени в поле типа Дата/Время обеспечивает правильную сортировку. Все изменения, внесенные в форматы дат и времени в окне Язык и стандарты Панели управления Windows, будут автоматически отражены в полях типа Дата/Время. | 8 байт. |
| Денежный | Значения валют. Денежный тип используется для предотвращения округлений во время вычислений. Предполагает до 15 символов в целой части числа и 4 – в дробной. | 8 байт. |
| Счетчик | Автоматическая вставка последовательных (отличающихся на 1) или случайных чисел при добавлении записи. Для создания возрастающего счетчика следует оставить все настройки свойства в нижней части окна прежними, по умолчанию, в свойстве Размер поля задано значение Длинное целое, а в свойстве Новые значения – Последовательные. Для создания счетчика случайных чисел для свойства Новые значения нужно установить значение Случайные. | 4 байта. Для кодов репликации – 16 байт. |
| Логический | Содержащие только одно или два значения, такие как Да/Нет, Истина/Ложь, Вкл/Выкл. | 1 бит. |
| Объекты OLE | Объекты (например, документы Microsoft Word, электронные таблицы Microsoft Excel, рисунки, звуки и другие данные), созданные в других программах, использующих протокол OLE. Объекты могут быть связанными или внедренными в таблицу Microsoft Access. Для отображения объекта OLE в форме или отчете необходимо использовать элемент управления Присоединенная рамка объекта. | До 1 гигабайта |
| Гиперссылка | Поле, в котором хранятся гиперссылки. Гиперссылка может быть либо типа UNC (стандартный формат пути файла с включением сетевого сервера), либо URL (адрес объекта в 1п1егпе1 или внутренней сети с включением типа протокола доступа). | До 64 000 символов |
| Мастер подстановок | Создается поле, позволяющее выбрать значение. Из другой таблицы или из списка значений, используя поле со списком. При выборе данного параметра в списке типов данных для их определения загружается мастер. | Размер такой же, как и размер ключевого поля |

Важно: Числовые, денежные и логические типы данных, а также Дата/Время обеспечивают стандартные форматы отображения. Для выбора форматов для каждого типа данных следует определить свойство Формат. Для всех данных, кроме объектов OLE, можно также создать пользовательский формат отображения. Подробнее см. ниже, в разделе "Свойство Формат поля".

Свойство Размер поля

Свойство Размер поля определяет максимальный размер данных, которые могут сохраняться в полях с типом данных Текстовый, Числовой или Счетчик.

Если свойство Тип данных имеет значение "Текстовый", значением данного свойства должно быть целое число в диапазоне от 0 до 255. По умолчанию задается размер 50.

Если свойство Тип данных имеет значение "Счетчик", то допустимыми значениями свойства Размер поля будут "Длинное целое" или "Код репликации".

Если поле имеет тип данных "Числовой", то допустимыми являются следующие значения свойства Размер поля:

|  |  |  |  |
| --- | --- | --- | --- |
| Значение | Описание | Дробная часть  | Размер |
| Байт | Числа от 0 до 255 | Отсутствует | 1 байт |
| Целое | Числа от -32 768 до 32 767 | Отсутствует | 2 байта |
| Длинное целое | (Значение по умолчанию). Числа от -2 147 483 648 до 2 147 483 647 | Отсутствует | 4 байта |
| С плавающей точкой (4 байта) | Числа от -3.402823Е38 до -1.401298Е-45 для отрицательных значений и от 1.401298Е-45 до 3.402823Е38 для положительных. | 7 знаков | 4 байта |
| С плавающей точкой (8 байт) | Числа от -1.79769313486232Е308 до -4.94065645841247Е для отрицательных значений и от 1.79769313486231Е308 до4.94065645841247Е-324 для положительных. | 15 знаков | 8 байт |
| Код репликации | Глобальный уникальный идентификатор (GUID) при репликации объектов данных | Не определено | 16 байт |

Для получения или задания максимального размера текстового поля в программе Visual Basic следует использовать свойство Size объектов доступа к данным (DAO). Для полей других типов значение свойства Size автоматически определяется значением свойства Туре.

Важно: Пользователь имеет возможность указать стандартные размеры текстовых и числовых полей в группе Размеры полей по умолчанию на вкладке Таблицы/запросы (в диалоговом окне Параметры, которое открывается командой Параметры в меню Сервис). Рекомендую задавать минимально допустимое значение свойства Размер поля, поскольку обработка данных меньшего размера выполняется быстрее и требует меньше памяти. Преобразование большего значения свойства Размер поля к меньшему в таблице, которая уже содержит данные, может привести к потере данных. Например, при уменьшении размера текстового поля с 255 до 50 все значения, длина которых превышает 50 символов, будут усечены. Данные в числовом поле, которые выходят за пределы диапазона, соответствующего новому размеру поля, округляются или заменяются пустыми значениями. Например, при замене значения "С плавающей точкой (4 байта)" на "Целое" дробные числа будут округлены до ближайшего целого числа, а значения вне диапазона от -32 768 до 32 767 будут преобразованы в пустые значения. Отменить изменения данных, произошедших при модификации свойства Размер поля, после его сохранения в режиме конструктора таблицы будет невозможно. Для полей, в которых планируется хранить числовые значения с одним – четырьмя знаками в дробной части, рекомендуется использовать денежный тип данных. При обработке числовых значений из полей типа "С плавающей точкой (4 байта)" и "С плавающей точкой (8 байт)" применяются вычисления с плавающей точкой. При обработке числовых значений из денежных полей используются более быстрые вычисления с фиксированной точкой.

Поле типа Счетчик

Для создания полей, в которые при добавлении записи автоматически вводится число, в Microsoft Access существует тип данных Счетчик. При этом созданный для записи номер уже не может быть удален или изменен. В поле счетчика могут быть использованы три типа чисел: последовательно возрастающие на один, случайные числа, а также коды репликации (также называются GUID – глобальные уникальные идентификаторы). Наиболее часто используется счетчик последовательно возрастающих чисел. Такой тип счетчика удобно использовать как ключевое поле таблицы. Счетчик случайных чисел создает уникальный номер для каждой записи в таблице.

Поле счетчика и репликация

Код репликации используется при репликации базы данных для создания уникальных идентификаторов, обеспечивающих синхронизацию реплик. При репликации базы данных необходимо определить подходящий размер для поля типа Счетчик, используемого в качестве ключевого поля таблицы. При использовании поля типа Счетчик как ключевого поля для таблицы в реплицированной базе данных для его свойства Размер поля можно установить значение либо Длинное целое, либо Код репликации. Если между операциями синхронизации реплик добавляется, как правило, менее 100 записей, то с целью экономии дискового пространства для свойства Размер поля следует использовать значение Длинное целое. Ну а если между операциями синхронизации добавляется более 100 записей, то с целью предотвращения повторения значений в ключевых полях в разных репликах следует использовать значение Код репликации. Однако следует иметь в виду, что в поле типа Счетчик с размером Код репликации генерируются 128-байтовые значения, требующие больше места на диске.

Свойство Формат поля

Свойство Формат поля позволяет указать форматы вывода текста, чисел, дат и значений времени на экран и на печать. Например, для поля Цена разумно указать в свойстве Формат поля формат Денежный и установить для его свойства - Число десятичных знаков - значение 2 или Авто. В этом случае введенное в поле значение 4321,678 будет отображаться как 4 321,68р. Допустимо использование как встроенных, так и специальных форматов, созданных при помощи символов форматирования. Для элементов управления значение свойства Формат поля задается в окне свойств. Для поля в таблице или запросе значение данного свойства задается в режиме конструктора таблицы (в разделе свойств поля) или в окне запроса (в окне свойств поля). Форматы можно выбирать из списка встроенных форматов для полей, имеющих числовой, денежный, логический типы данных, а также типы данных счетчика и даты/времени. Также для любых типов данных полей, отличных от объектов OLE; есть возможность создания собственных специальных форматов. Кроме того, значение данного свойства можно задать в макросе или в программе.

Свойство Формат поля определяет только способ отображения данных, не оказывая воздействия на способ их сохранения. В Microsoft Access определены стандартные форматы для полей с типами данных Числовой, Дата/Время, Логический, Текстовый и Поле МЕМО. В качестве стандартных используются национальные форматы, выбираемые в окне Язык и стандарты Панели управления Windows. Набор форматов определяется настройками для конкретной страны. Например, если на вкладке Язык и стандарты указать Английский (США), то число 1234.56 в денежном формате будет выглядеть как $1,234.56. Но если указать на этой вкладке Русский, то это число будет выглядеть так: 1 234,56р. Настройка Формат поля, заданная в режиме конструктора таблицы, используется для отображения данных в режиме таблицы. Эта же настройка применяется при создании связанных с этим полем новых элементов управления в форме или отчете.

Ниже перечисляются символы, используемые при определении специальных форматов для любого типа данных.

|  |  |
| --- | --- |
| Символ | Значение |
| (Пробел) | Выводит пробел как символьную константу. |
| "АВС" | Все символы внутри кавычек считаются символьными константами. |
| ! | Выравнивает символы по левому краю. |
| \* | Заполняет доступное пустое пространство следующим символом. |
| \ | Выводит следующий символ как символьную константу. Для этой же цели можно использовать кавычки. |
| [цвет] | Задает цвет, название которого указано в скобках. Допустимые имена цветов: Черный, Синий, Зеленый, Бирюзовый , Красный, Лиловый, Желтый, Белый. |

Не разрешается смешивать в одном формате специальные символы, предназначенные для определения числовых форматов, форматов даты/времени и текстовых форматов. Если для поля определено значение свойства Маска ввода, а в свойстве Формат поля задается другое форматирование тех же данных, то приоритет имеют настройки, задаваемые в свойстве Формат поля, а значение Маска ввода игнорируется. В свойстве Формат поля задаются разные настройки для различных типов данных. Ниже приводится описание конкретных настроек.

Свойство Формат поля для даты/времени

Свойство Формат поля позволяет указать использование встроенных или специальных числовых форматов для полей даты/времени. В следующей таблице приводятся встроенные значения свойства Формат поля для полей даты/времени.

|  |  |
| --- | --- |
| Значение | Описание |
| Полный формат даты | (Значение по умолчанию). Если значение содержит только дату, то время не отображается; если значение содержит только время, то дата не отображается. Данный формат является комбинацией двух: "Краткий формат даты" и "Длинный формат времени". Примеры: 01.11.95 1:07:19 и 23.01.96 23:01:04. |
| Длинный формат даты | Совпадает с настройкой "Полный формат", задающейся в окне Язык и стандарты Панели управления Windows. Пример: 1 Июнь 1995 г. |
| Средний формат даты | Пример: 03-апр-95. |
| Краткий формат даты | Совпадает с настройкой "Краткий формат даты", задающейся в окне Язык и стандарты Панели управления Windows. Пример: 11.06.95. Значения краткого формата даты предполагают, что даты из диапазона 01.01.00 и 31.12.29 относятся к двадцать первому веку (то есть, предполагаются годы с 2000 по 2029). Предполагается также, что даты из промежутка 01.01.30 и 31.12.99 относятся к двадцатому веку (то есть годы с 1930 по 1999). |
| Длинный формат времени | Совпадает с форматом времени, задающимся в окне Язык и стандарты на вкладке Время панели управления Windows.Пример: 20:58:10. |
| Средний формат времени | Пример: 05:34 РМ. |
| Краткий формат времени | Пример: 17:34. |

Также существуют специальные форматы даты и времени. Специальные форматы выводятся в соответствии со значениями, установленными в окне Язык и стандарты Панели управления Windows. Специальные форматы, противоречащие настройкам окна Язык и стандарты, игнорируются.

Свойство Формат поля для числовых и денежных полей

Свойство Формат поля позволяет указать использование встроенных и специальных числовых форматов для числовых и денежных типов данных. В следующей таблице приводятся встроенные значения свойства Формат поля для числовых полей.

|  |  |
| --- | --- |
| Значение | Описание |
| Основной | (Значение по умолчанию). Числа отображаются так, как они были введены. |
| Денежный | Используются разделители групп разрядов; отрицательные числа выводятся в круглых скобках; свойство Число десятичных знаков по умолчанию получает значение 2. |
| Фиксированный | Выводится по крайней мере один разряд; свойство Число десятичных знаков по умолчанию получает значение 2. |
| С разделителями | Числа выводятся с разделителями групп разрядов; свойство Число разрядов десятичных знаков по умолчанию получает значение 2. |
| Процентный | Значение умножается на 100; добавляется символ процентов (%); свойство Число десятичных знаков по умолчанию получает значение 2. |
| Экспоненциальный | Числа выводятся в экспоненциальной (научной) нотации. |

Специальные числовые форматы могут включать в себя от одного до четырех разделов, отделенных друг от друга точкой с запятой (;). Каждый формат содержит спецификацию для различных разделов (типов) числовых данных.

|  |  |
| --- | --- |
| Раздел | Описание |
| Первый | Формат положительных чисел. |
| Второй | Формат отрицательных чисел. |
| Третий | Формат нулевых значений. |
| Четвертый | Формат пустых значений. |

Свойство Формат поля для текстовых и МЕМО-полей

Свойство Формат поля позволяет создавать специальные форматы для текстовых и МЕМО-полей с помощью специальных символов. Для этого используются следующие символы:

|  |  |
| --- | --- |
| Символ | Описание |
| @ | Обязательный текстовый символ или пробел. |
| & | Необязательный текстовый символ. |
| < | Преобразует все символы в строчные. |
| > | Преобразует все символы в прописные. |

Специальные форматы для текстовых полей и полей МЕМО могут включать один или два раздела, разделяемых точкой с запятой (;). Эти разделы описывают спецификации формата различных порций данных поля.

|  |  |
| --- | --- |
| Раздел | Описание |
| Первый | Формат отображения текста. |
| Второй | Формат отображения строк нулевой длины и пустых значений. |

Формат поля и маска ввода данных

В Microsoft Access к похожим результатам приводит изменение двух свойств полей: свойство Формат поля и свойство Маска ввода. Свойство Формат поля используется для отображения данных в постоянном формате. Например, если свойство Формат поля для полей типа Дата/Время установлено на Средний формат даты, то все вводимые данные будут отображаться в следующем формате: 12-янв-97. Если же пользователь базы данных введет число в виде 12.01.97 (или в другом определенном виде), то при сохранении записи формат даты будет преобразован в Средний формат даты. При установке свойства Формат поля изменяется только отображение значения, однако, данное свойство никак не влияет на хранение значения в таблице. Изменения в формате отображения применяются только после сохранения введенных данных, до этого момента определить, в каком формате были введены данные в поле, невозможно. Если же вводом данных необходимо управлять, в дополнение к формату отображения данных или вместо него используется маска ввода. Если требуется, чтобы данные отображались так, как они были введены, свойство Формат поля вообще не устанавливается. Маска ввода обеспечивает соответствие данных определенному формату, а также заданному типу значений, вводимых в каждую позицию. Например, для поля Номер телефона требуется, чтобы все вводимые значения телефонного номера содержали точное число только цифровых знаков и составляли полный номер телефона (например, в США это код штата, код города и номер абонента). Если для поля определены как формат отображения, так и маска ввода, то при добавлении и редактировании данных используется маска ввода, а параметр Формат поля определяет отображение данных при сохранении записи. Если используется как свойство Формат поля, так и свойство Маска ввода, необходимо обеспечить, чтобы результаты их действия не противоречили друг другу. Маска ввода для поля таблицы создается в режиме конструктора с помощью мастера.

## 3.2 Ключи и индексы

Мощь реляционных баз данных заключается в том, что с их помощью можно быстро найти и связать данные из разных таблиц при помощи запросов; форм и отчетов. Для этого каждая таблица должна содержать одно или несколько полей, однозначно идентифицирующих каждую запись в таблице. Эти поля называются ключевыми полями таблицы. Если для таблицы обозначены ключевые поля, то процессор базы данных (в Access – Microsoft Jet) предотвращает дублирование или ввод пустых значений в ключевое поле.

### 3.2.1 Типы ключевых полей

В Microsoft Access можно выделить три типа ключевых полей: счетчик, простой ключ и составной ключ.

Ключевые поля счетчика

Поле счетчика можно задать таким образом, чтобы при добавлении каждой записи в таблицу в это поле автоматически вносилось порядковое число (см. выше, раздел "Поле счетчика"). Указание такого поля в качестве ключевого – наиболее простой способ создания ключевых полей. Если до сохранения созданной таблицы ключевые поля не были определены, то при сохранении будет выдано сообщение о создании ключевого поля. При нажатии кнопки Да будет автоматически создано ключевое поле счетчика.

Простой ключ

Если поле содержит уникальные значения, такие как коды или инвентарные номера, то это поле можно определить как ключевое. Если выбранное поле содержит повторяющиеся или пустые значения, то оно не будет определено как ключевое. Для определения записей, содержащих повторяющиеся данные, можно выполнить запрос на поиск повторяющихся записей. Если устранить повторы путем изменения значений невозможно, то следует либо добавить в таблицу поле счетчика и сделать его ключевым, либо определить составной ключ.

Составной ключ

В случаях, когда невозможно гарантировать уникальность значений каждого отдельного поля, можно создать ключ, состоящий из нескольких полей. Чаще всего такая ситуация возникает для таблицы, используемой для связывания двух таблиц в отношении "многие-к-многим". Примером такой таблицы может служить таблица Книги базы данных Картотека, связывающая таблицы Авторы и Издатели. В этой таблице можно назначить ключ, состоящий из двух полей: NoАвтора и NoИздателя. В таблице Книги может быть представлено много авторов и много издателей, но каждая книга присутствует в картотеке только один раз, поэтому комбинация значений полей NoАвтора и NoИздателя достаточна для образования ключа.

Другим примером может служить складская база данных, в инвентарной книге которой используются один основной и один или несколько вспомогательных инвентарных номеров.

Важно: Если определить подходящий набор полей для составного ключа сложно, можно просто добавить поле счетчика и сделать его ключевым. Например, не рекомендую определять ключ по полям Имена и Фамилии, поскольку нельзя исключить повторения этой пары значений для разных людей.

### 3.2.2 Индексы

Индексы – объекты базы данных, которые обеспечивают быстрый доступ к отдельным строкам в таблице. Индекс создается с целью повышения производительности операций запросов и сортировки данных таблицы. Индексы также используются для поддержания в таблицах некоторых типов ключевых ограничений; эти индексы часто создаются автоматически при определении ограничения. Индекс – независимый объект, логически отдельный от индексированной таблицы; создание или удаление индекса никак не воздействует на определение или данные индексированной таблицы. Он хранит высоко оптимизированные версии всех значений одного или больше столбцов таблицы. Когда значение запрашивается из индексированного столбца, процессор (ядро) базы данных использует индекс для быстрого нахождения требуемого значения. Индексы должны постоянно поддерживаться, чтобы отражать последние изменения индексированных столбцов таблицы. Процедуры обновления индекса при вставке, модификации или удалении значения в индексированный столбец автоматически выполняются процессором базы данных. Хотя эти операции не требуют никаких действий со стороны пользователя, они, однако, снижают эффективность некоторых операций манипулирования данными (кроме запросов на выборку). Однако уменьшение производительности, ассоциированное с поддержанием индекса, в большинстве случаев с лихвой компенсируется преимуществами повышения быстродействия доступа к данным, которое обеспечивает индекс. Индексы обеспечивают наибольшие выгоды для относительно статичных таблиц, по которым часто выполняются запросы.

### 3.2.3 Создание и изменение ключевых полей

Для создания ключевых полей таблицы:

1. В режиме конструктора выделить одно или несколько полей, которые необходимо определить как ключевые. Для выделения одного поля нужно щелкнуть область выделения строки нужного поля (кнопка слева строки). Выделить несколько полей можно, удерживая при выборе каждого поля клавишу "Сtrl".
2. Нажать кнопку Ключевое поле на панели инструментов.

Создание индекса

Создать индексы, как и ключи, можно по одному или нескольким полям. Составные индексы позволяют при отборе данных группировать записи, в которых первые поля могут иметь одинаковые значения. Индексировать поля требуется для выполнения частых поисков, сортировок или объединений с полями из других таблиц в запросах. Ключевые поля таблицы индексируются автоматически. Нельзя индексировать поля с типом данных поле МЕМО, гиперссылка или объект OLE. Для остальных полей индексирование используется, если поле имеет текстовый, числовой, денежный тип или тип даты/времени и требуется осуществлять поиск и сортировку значений в поле. Если предполагается, что будет часто выполняться сортировка или поиск одновременно по двум и более полям, можно создать составной индекс. Например, если для одного и того же запроса часто устанавливается критерий для полей Имя и Фамилия, то для этих двух полей имеет смысл создать составной индекс. При сортировке таблицы по составному индексу сначала осуществляется сортировка по первому полю, определенному для данного индекса. Если в первом поле содержатся записи с повторяющимися значениями, то сортировка осуществляется по второму полю и т. д.

Чтобы создать индекс для одного поля надо:

1. В режиме конструктора в панели структуры таблицы (верхняя часть окна) выбрать поле, для которого требуется создать индекс.
2. В панели свойств (нижняя часть окна) для свойства Индексированное поле установить значение "Да (Допускаются совпадения)" или "Да (Совпадения не допускаются)".

Убедиться, что в данном поле совпадающих записей нет, можно, выбрав значение "Да (Совпадения не допускаются)".

Чтобы создать составной индекс:

1. В режиме конструктора на панели инструментов нажать кнопку Индексы.
2. В первой пустой строке поля Индекс ввести имя индекса. Для индекса можно использовать либо имя одного из индексируемых полей, либо другое подходящее имя.
3. В поле Имя поля нажать стрелку и выбрать в списке первое поле, для которого требуется создать индекс.
4. В следующей строке поля Имя поля указать второе индексируемое поле (для данной строки поле Индекс следует оставить пустым). Повторить эти действия для всех полей, которые необходимо включить в индекс. В составном индексе может быть до 10 полей.

Важно: По умолчанию, установлен порядок сортировки "По возрастанию". Для сортировки данных полей по убыванию в поле Порядок сортировки в окне индексов нужно указать значение "По убыванию". Хочу заметить, что поля индекса могут не быть ключевыми.

Ограничение Onigue

Ограничение Unigue предотвращает ввод в поле повторяющихся значений. Этот тип ограничения может быть установлен как для одного поля, так и для нескольких полей составного ключа. Назначение ключевого поля (для одного поля) автоматически запрещает ввод в него повторяющихся значений, тем самым обеспечивая для каждой записи уникальный идентификатор. Однако запрет на ввод повторяющихся значений может потребоваться и для других, не ключевых, полей.

Чтобы установить ограничение Unigue для одного поля таблицы:

1. В режиме конструктора в панели структуры таблицы выбрать поле, в котором допускается ввод только уникальных значений.
2. В панели свойств для свойства Индексированное поле установить значение "Да (Совпадения не допускаются)".

Чтобы установить ограничение Unigue для нескольких полей таблицы:

1. В режиме конструктора открыть окно индексов и создать составной индекс, включив в него поля, в которые должен быть, запрещен ввод повторяющихся значений.
2. Выбрав имя индекса, в панели свойств индекса в ячейке свойства Уникальный индекс установить значение "Да".

##

## 3.3 Общая картина ограничений и поддержания целостности данных

Ограничение – некоторое ограничивающее условие. В базе данных – общее понятие, охватывающее широкий круг аспектов управления базой данных: ключи, значения, типы и форматы данных и т. д. Ограничения устанавливают для пользователя некоторые рамки при вводе, изменении или удалении данных приложения. Вся система ограничений при создании приложения базы данных строится с целью обеспечения целостности данных.

Целостность данных представляет собой набор правил, используемых процессором базы данных для поддержания связей между записями в связанных таблицах, а также для защиты от случайного удаления или изменения связанных данных. Например, ограничение можно использовать, чтобы гарантировать, что каждый служащий в базе данных будет относиться к какому-либо отделу или что пользователи не смогут случайно ввести отрицательное значение для цены товара.

Ограничения можно определять на двух уровнях:

1. В базе данных. Ограничения в базе данных ассоциируются с определениями объектов-таблиц. Например, для таблицы может быть установлено ограничение, которое требует, чтобы каждое значение в столбце было уникальным.
2. В приложении Access (в программном коде или свойствах объектов). Ограничения в приложении ассоциируются с объектами приложения, которые формируют интерфейс к информации базы данных. Например, текстовое поле может иметь ограничение, которое требует, чтобы все вводимые в него значения были больше 20.

###

### 3.3.1 Ограничения в базе данных

Ограничение в базе данных – декларативно определенное правило, ограничивающее значения, которые могут быть введены в столбец или набор столбцов в таблице. Ограничения базы данных являются декларативно определяемыми, так как определяют ограничения как часть структуры таблицы при ее создании или изменении. Будучи однажды ассоциировано с таблицей, ограничение всегда поддерживается, если его явно не удалить или не деактивировать. Размещение ограничений в базе данных имеет следующие преимущества:

* централизация. Ограничение базы данных определяется только один раз и может автоматически использоваться всеми клиентами, обращающимися к базе данных. Определение ограничения в базе данных освобождает разработчика от необходимости вносить одни и те же ограничения в каждую форму, которая использует данную информацию. Кроме того, при необходимости модифицировать ограничение изменения вносятся только в один объект;
* защита. Ограничения базы данных всегда поддерживаются, независимо от того, какой инструмент доступа к данным используется. С другой стороны, ограничения, определенные в приложении, могут быть нарушены пользователем, использующим для доступа к тем же таблицам другое приложение или инструмент;
* простота. Ограничения базы данных просты в определении и не требуют никакого программного кода.

### 3.3.2 Типы ограничений в базе данных

Типы ограничений, которые можно ассоциировать с таблицей, варьируются в зависимости от базы данных, в которой хранится эта таблица. Описанные ниже категории ограничений поддерживаются большинством реляционных баз данных, в том числе и Microsoft Access.

Ограничения Not Null

Ограничение Not Null запрещает ввод в столбец таблицы пустых значений. Оно всегда применяется к отдельным столбцам. Ограничения Not Null используются, чтобы гарантировать, что для важных данных всегда имеются значения. Например, это ограничение можно использовать, чтобы гарантировать, что в записи каждого служащего в базе данных проставлено его жалованье. При определении структуры таблицы это ограничение задается установкой значений свойств Обязательное поле и Пустые строки поля таблицы. Необходимо различать два типа пустых значений: пустые значения и пустые строки. В некоторых ситуациях поле может быть оставлено пустым потому, что данные для него либо существуют, но пока неизвестны, либо их не существует вовсе. В связи с этим и различают два типа пустых строк. Например, если в таблице есть поле "Номер факса", то оно может быть пустым потому, что пользователь не знает, есть ли у клиента номер факса или нет, или потому, что он знает, что номера факса у клиента нет. Таким образом, если поле имеет пустое значение, то это означает, что его значение неизвестно. Если же введена пустая строка (два знака прямых кавычек (" ")), то это означает, что строкового значения нет.

Ограничения Unique

Ограничение Unique запрещает пользователю ввод в столбец или набор столбцов дублированных значений. Ограничение Unique может активироваться для отдельного столбца или для комбинации столбцов. В последнем случае ограничение Unique иногда называется составным ограничением Unique. Ограничения Unique используются, чтобы гарантировать, что в таблице не будет дублированных значений столбцов. Например, оно может гарантировать, что каждому служащему в базе данных будет присвоен уникальный номер. Ограничение Unique не запрещает пользователю ввод в таблицу нескольких пустых значений – пустое значение в столбце всегда удовлетворяет ограничению Unique. Чтобы предотвратить ввод в столбец с ограничением Unigue пустых значений, к столбцу необходимо также добавить ограничение Unique. В Access ограничение Unique инициируется установкой значения "Да (Совпадения не допускаются)" для свойства Индексированное поле, либо установкой значения "Да" для свойства Уникальный индекс.

Ограничения Primary Key

Ограничение Primary Key гарантирует, что каждая строка в таблице будет уникально идентифицирована значением в столбце или наборе столбцов первичного ключа. Ограничение по первичному ключу объединяет черты ограничения 0пiцие и ограничения Unigue и Not Null.

Обычно рекомендуется включать ограничение Primary Key в каждой создаваемой таблице. Использование первичного ключа может значительно повысить быстродействие доступа к строкам таблицы. Ограничение Primary Key также используется для поддержания ссылочной целостности, когда в базе данных определены отношения один-к-многим. Установка ссылочной целостности позволяет поддерживать соответствие между главной и подчиненной таблицами. Для поддержания ссылочной целостности ограничения Primary Key используются в комбинации с ограничениями Primary Key, описанными ниже.

Ограничения Foreign Key

Ограничение Foreign Key (внешний ключ) гарантирует, что каждое значение, введенное в столбец, уже существует в некотором другом столбце (обычно в другой таблице). Ограничения Foreign Key обычно используются для поддержания ссылочной целостности, когда в базе данных определены отношения один-к-многим. Ограничения Foreign Key всегда используются вместе с ограничениями Foreign Key (описанными в предыдущем разделе). В отношении один-к-многим внешний ключ – столбец в подчиненной таблице, которая содержит идентификатор строки в главной таблице. Значение в столбце внешнего ключа равно значению в столбце первичного ключа в другой таблице. Также, в отношении один-к-одному каждая строка в подчиненной таблице соответствует уникальной строке в главной таблице, одной строке в главной таблице может соответствовать любое количество строк в подчиненной таблице.

3.3.3 Поддержание целостности данных

При поддержании ссылочной целостности между главной и подчиненной таблицами часто используются следующие правила:

* подчиненная строка не может быть вставлена, пока не существует главная строка. Например, нельзя ввести записи позиций счет-фактуры, пока в главной таблице не появится запись счета. Однако в поле внешнего ключа возможен ввод пустых значений, показывающих, что записи только подготавливаются и пока не являются связанными;
* главная строка не может быть удалена до удаления всех подчиненных строк. Например, нельзя удалить запись счет-фактуры, если в подчиненной таблице имеются записи позиций счета;
* если значение первичного ключа в главной строке- изменено, все значения внешнего ключа, которые обращаются к этому значению первичного ключа, должны быть также обновлены; и наоборот, нельзя изменить значение ключевого поля в главной таблице, если существуют связанные записи.

Чтобы наложить эти правила на конкретную связь, при ее создании следует установить флажок Обеспечение целостности данных в окне Связи. Если данный флажок установлен, то любая попытка выполнить действие, нарушающее одно из перечисленных выше правил, приведет к выводу на экран предупреждения, а само действие выполнено не будет.

Каскадное обновление и каскадное удаление

Для связей, для которых определена целостность данных, пользователь имеет возможность указать, следует ли автоматически выполнять для связанных записей операции каскадного обновления и каскадного удаления. Если включить данные параметры, станут возможными операции удаления и обновления, в противном случае запрещенные условиями целостности данных. Чтобы обеспечить целостность данных при удалении записей или изменении значения ключевого поля в главной таблице, автоматически вносятся необходимые изменения в связанные таблицы. Если при определении связи в окне Связи установить флажок Каскадное обновление связанных полей, любое изменение значения в ключевом поле главной таблицы приведет к автоматическому обновлению соответствующих значений во всех связанных записях. Например, при изменении кода клиента в таблице "Клиенты" будет автоматически обновлено поле "Код Клиента" во всех записях таблицы "Заказы" для заказов каждого клиента, поэтому целостность данных не будет нарушена. Access выполнит каскадное обновление без ввода предупреждающих сообщений.

Важно: Если в главной таблице ключевым полем является поле счетчика, то установление флажка Каскадное обновление связанных полей не приведет к каким-либо результатам, так как изменить значение поля счетчика невозможно.

Если при определении связи установить флажок Каскадное удаление связанных записей, любое удаление записи в главной таблице приведет к автоматическому удалению связанных записей в подчиненной таблице. Например, при удалении из таблицы "Клиенты" записи конкретного клиента будут автоматически удалены все связанные записи в таблице "Заказы". Если записи удаляются из формы или таблицы при установленном флажке Каскадное удаление связанных записей, Access выводит предупреждение о возможности удаления связанных записей. Если же записи удаляются с помощью запроса на удаление записей, то удаление осуществляется автоматически, без вывода предупреждения.

# Заключение

Microsoft Access – самая популярная в мире база данных для операционной системы Microsoft Windows . Кроме того, система управления базами данных Access – также мощная платформа разработки с чрезвычайно гибкой и функциональной интегрированной средой. Access – это инструмент, предназначенный для разработки и развертывания широкопредметных информационных бизнес-систем. Возможности разработчиков программного обеспечения, а также методы и технологии решения этих задач постоянно изменяются и совершенствуются. Как только появляется какое-нибудь перспективное решение для обеспечения быстрой разработки приложений, технология и инструментальные средства изменяются на базе этого новшества практически мгновенно. С каждой новой версией Access такие решения становятся достоянием самого широкого сообщества разработчиков. Access 2007 для Windows позволяет для обработки информации и быстрого формирования деловых решений привлекать мощь реляционной базы данных, интегрировать данные из электронных таблиц и других баз данных, компоненты других приложений, а также использовать информацию совместного доступа во внутренних сетях и Internet. Среда Access может с успехом использоваться начинающими пользователями для познания секретов реляционных баз данных и увлекательных занятий по созданию несложных (поначалу) приложений и в то же время предоставляет мощные инструменты разработки опытным программистам. Чрезвычайно развитые справочная система, средства обучения, мастера и программы-надстройки позволяют при построении приложения и работе в Access 07 найти выход из любой ситуации и получить ответ на любой вопрос. Начинать работать с Access можно практически с любым уровнем подготовки. Access 07 – это масштабируемая система. Создаваемые прикладные решения могут легко расширяться для реализации новых деловых задач и управления данными.