**Лекция 7:**

**Модели организации баз данных**

**Аннотация:**В лекции рассматриваются модели организации баз данных, дается характеристика каждой модели. Описываются достоинства и недостатки существующих моделей баз данных. Даются определения атрибута, записи и отношений в различных моделях БД.

**Цель лекции:** Уяснить разницу между моделями организации *БД*. Ознакомиться с их достоинствами и недостатками. Понять, как организовываются связи в этих моделях, как применяются *операции* изменения в той или иной модели.

Различают три *основные модели* *базы данных* - это иерархическая, сетевая и реляционная. Эти модели отличаются между собой по способу установления связей между данными.

**1. Иерархический подход к организации баз данных.** *Иерархические базы данных* имеют форму деревьев с дугами-связями и узлами-элементами данных. Иерархическая структура предполагала неравноправие между данными - одни жестко подчинены другим. Подобные структуры, безусловно, четко удовлетворяют требованиям многих, но далеко не всех реальных задач.

**2. Сетевая модель данных.** В *сетевых БД* наряду с вертикальными реализованы и горизонтальные связи. Однако унаследованы многие недостатки иерархической и главный из них, необходимость четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей при запросах к базе.

**3. Реляционная модель.** Реляционная модель появилась вследствие стремления сделать базу данных как можно более гибкой. Данная модель предоставила простой и *эффективный механизм* поддержания связей данных.

**Во-первых**, все данные в модели представляются в виде таблиц и только таблиц. Реляционная модель - единственная из всех обеспечивает единообразие представления данных. И сущности, и связи этих самых сущностей представляются в модели совершенно одинаково - **таблицами**. Правда, такой подход усложняет понимание смысла хранящейся в базе данных информации, и, как следствие, манипулирование этой информацией.

Избежать трудностей манипулирования позволяет **второй элемент** модели - реляционно-полный язык (отметим, что язык является неотъемлемой частью любой модели данных, без него модель не существует). *Полнота* языка в приложении к реляционной модели означает, что он должен выполнять любую *операцию реляционной алгебры* или *реляционного исчисления* (*полнота* последних доказана математически Э.Ф. Коддом). Более того, язык должен описывать любой *запрос* в виде операций с таблицами, а не с их строками. Одним из таких языков является *SQL*.

**Третий элемент** реляционной модели требует от реляционной модели поддержания некоторых *ограничений целостности*. Одно из таких ограничений утверждает, что каждая строка в таблице должна иметь некий уникальный *идентификатор*, называемый первичным ключом. Второе ограничение накладывается на *целостность* ссылок между таблицами. Оно утверждает, что атрибуты таблицы, ссылающиеся на первичные ключи других таблиц, должны иметь одно из значений этих первичных ключей.

**4. Объектно-ориентированная модель.** Новые области использования вычислительной техники, такие как научные исследования, автоматизированное проектирование и *автоматизация* учреждений, потребовали от баз данных способности хранить и обрабатывать новые объекты - текст, аудио- и видеоинформацию, а также документы. Основные трудности *объектно-ориентированного моделирования* данных проистекают из того, что такого развитого математического аппарата, на который могла бы опираться общая *объектно-ориентированная модель данных*, не существует. В большой степени, поэтому до сих пор нет базовой объектно-ориентированной модели. С другой стороны, некоторые авторы утверждают, что общая *объектно-ориентированная модель данных* в классическом смысле и не может быть определена по причине непригодности классического понятия модели данных к парадигме объектной ориентированности. Несмотря на преимущества объектно-ориентированных систем - реализация *сложных типов данных*, *связь* с языками программирования и т.п. - на ближайшее время превосходство реляционных *СУБД* гарантировано.

Рассмотрим более подробно эти модели данных далее.

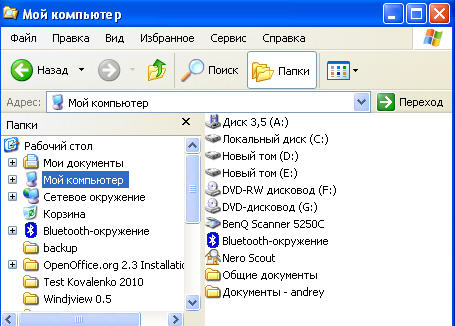
**Иерархическая модель базы данных**

*Иерархические базы данных* - самая ранняя модель представления сложной структуры данных. *Информация* в иерархической базе организована по принципу древовидной структуры, в виде отношений "предок-*потомок*". Каждая *запись* может иметь не более одной родительской записи и несколько подчиненных. Связи записей реализуются в виде физических указателей с одной записи на другую. Основной недостаток *иерархической структуры базы данных* - невозможность реализовать отношения "*многие-ко-многим*", а также ситуации, когда *запись* имеет несколько предков.

*Иерархические базы данных*. *Иерархические базы данных* графически могут быть представлены как перевернутое *дерево*, состоящее из объектов различных уровней. Верхний уровень (*корень дерева*) занимает один *объект*, второй - объекты второго уровня и так далее.

Между объектами существуют связи, каждый *объект* может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (*объект*, более близкий к корню) к потомку (*объект* более низкого уровня), при этом *объект*-предок может не иметь потомков или иметь их несколько, тогда как *объект*-*потомок* обязательно имеет только одного предка. Объекты, имеющие общего предка, называются близнецами.

Иерархической базой данных является Каталог папок *Windows*, с которым можно работать, запустив Проводник. Верхний уровень занимает *папка* *Рабочий стол*. На втором уровне находятся папки Мой *компьютер*, Мои документы, Сетевое окружение и *Корзина*, которые являются потомками папки *Рабочий стол*, а между собой является близнецами. В свою *очередь*, *папка* Мой *компьютер* является предком по отношению к папкам третьего уровня -папкам дисков (*Диск* 3,5(А:), (С:), (D:), (Е:), (F:)) и системным папкам (*сканер*, *bluetooth* и.т.д.) - на [рис. 4.1](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.1).



**Рис. 4.1.**Иерархическая база данных Каталог папок Windows

Организация данных в *СУБД* иерархического типа определяется в терминах: элемент, агрегат, *запись* (*группа*), групповое *отношение*, *база данных*.

|  |  |
| --- | --- |
| **Атрибут** (элемент данных) | - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем. |
| **Запись** | - именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. *Экземпляр записи* - конкретная запись с конкретным значением элементов. |
| **Групповое отношение** | - *иерархическое отношение* между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры. |

**Корневая запись** каждого дерева обязательно должна содержать *ключ* с уникальным значением. Ключи некорневых записей должны иметь уникальное *значение* только в рамках группового отношения. Каждая *запись* идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой, по иерархическому пути.

При графическом изображении групповые отношения изображают дугами ориентированного графа, а *типы записей* - вершинами (*диаграмма* Бахмана).

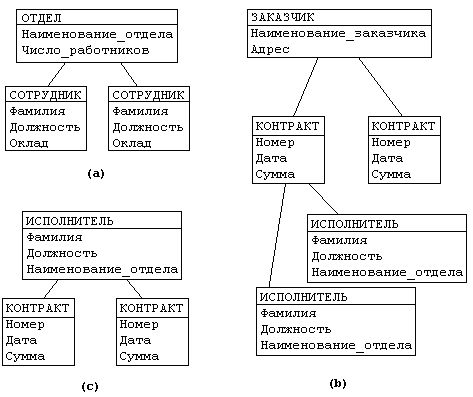
Для групповых отношений в *иерархической модели* обеспечивается автоматический режим включения и фиксированное членство. Это означает, что для запоминания любой некорневой записи в *БД* должна существовать ее родительская *запись*.

**Пример**

Рассмотрим следующую модель данных предприятия (см. [рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2)): предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

Поэтому, для *информационной системы управления* персоналом необходимо создать групповое отношение, состоящее из родительской записи ОТДЕЛ (НАИМЕНОВАНИЕ\_ОТДЕЛА, ЧИСЛО\_РАБОТНИКОВ) и дочерней записи СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД). Это отношение показано на [рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2) (а) (Для простоты полагается, что имеются только две дочерние записи).

Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры: заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи ЗАКАЗЧИК (НАИМЕНОВАНИЕ\_ЗАКАЗЧИКА, АДРЕС), КОНТРАКТ(НОМЕР, ДАТА,СУММА), ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ\_ОТДЕЛА) ([рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2) b).



**Рис. 4.2.**Пример иерархической базы данных

Из этого примера видны недостатки иерархических *БД*:

Частично дублируется *информация* между записями СОТРУДНИК и *ИСПОЛНИТЕЛЬ* (такие записи называют парными), причем в *иерархической модели данных* не предусмотрена *поддержка* соответствия между парными записями.

*Иерархическая модель* реализует *отношение* между исходной и дочерней записью по схеме 1:N, то есть одной родительской записи может соответствовать любое число дочерних.

Допустим теперь, что *исполнитель* может принимать участие более чем в одном контракте (т.е. возникает *связь* типа M:N). В этом случае в базу данных необходимо ввести еще одно групповое *отношение*, в котором *ИСПОЛНИТЕЛЬ* будет являться исходной записью, а КОНТРАКТ - дочерней ([рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2) c). Таким образом, мы опять вынуждены дублировать информацию.

*Операции* над данными, определенные в *иерархической модели*:

* **Добавить** в базу данных новую запись. Для корневой записи обязательно формирование значения ключа.
* **Изменить** значение данных предварительно извлеченной записи. Ключевые данные не должны подвергаться изменениям.
* **Удалить** некоторую запись и все подчиненные ей записи.
* **Извлечь** корневую запись по ключевому значению, допускается также последовательный просмотр корневых записей.
* **Извлечь** следующую запись (следующая запись извлекается в порядке левостороннего *обхода дерева*).

В *операции* **ИЗВЛЕЧЬ** допускается задание условий выборки (например, извлечь сотрудников с окладом более 10 тысяч руб.)

Как видим, все *операции* изменения применяются только к одной "текущей" записи (которая предварительно извлечена из *базы данных*). Такой подход к манипулированию данных получил название "навигационного".

**Ограничения целостности**

Поддерживается только целостность связей между владельцами и членами группового отношения (никакой потомок не может существовать без предка). Как уже отмечалось, не обеспечивается автоматическое поддержание соответствия парных записей, входящих в разные иерархии.

**Сетевая модель базы данных**

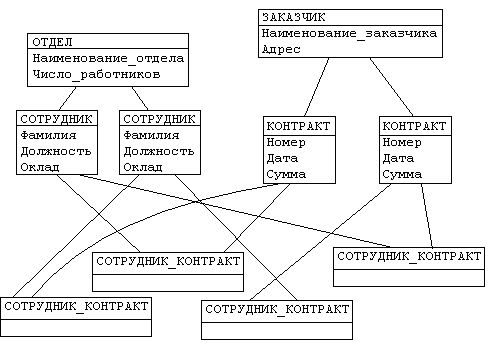
На разработку этого стандарта большое влияние оказал американский ученый Ч.Бахман. Основные принципы *сетевой модели данных* были разработаны в середине 60-х годов, эталонный вариант *сетевой модели данных* описан в отчетах рабочей группы по языкам баз данных (*COnference* on *DAta SYstem* Languages) CODASYL (1971 г.).

*Сетевая модель данных* определяется в тех же терминах, что и *иерархическая*. Она состоит из *множества* записей, которые могут быть владельцами или членами групповых отношений. *Связь* между записью-владельцем и записью-членом также имеет вид 1:N.

**Основное различие этих моделей** состоит в том, что в *сетевой модели* *запись* может быть членом более чем одного группового отношения. Согласно этой модели каждое групповое *отношение* именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: *экземпляр записи* не может быть членом двух экземпляров групповых отношений одного типа (т.е. сотрудник из примера в п..1, например, не может работать в двух отделах).

Иерархическая структура [рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2) преобразовывается в сетевую модель, следующим образом (см. [рис. 4.3](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=2#image.4.3)):

* деревья (a) и (b), показанные на [рис. 4.2](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=1#image.4.2), заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
* для отображения типа M:N вводится запись СОТРУДНИК\_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, (см. [рис. 4.3](https://intuit.ru/studies/courses/3439/681/lecture/14023?page=2#image.4.3)). Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту.



**Рис. 4.3.**Сетевая модель базы данных

Каждый экземпляр группового отношения характеризуется следующими признаками:

**Способ упорядочения подчиненных записей:**

* произвольный,
* хронологический /очередь/,
* обратный хронологический /стек/,
* сортированный.

Если *запись* объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

**Режим включения подчиненных записей:**

* автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
* ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем.

**Режим исключения.**

Принято выделять три класса членства подчиненных записей в групповых отношениях:

* **Фиксированное.** Подчиненная запись *жестко связана* с записью владельцем и ее можно исключить из группового отношения только удалив. При *удалении записи*-владельца все подчиненные записи автоматически тоже удаляются. В рассмотренном выше примере фиксированное членство предполагает групповое отношение "ЗАКЛЮЧАЕТ" между записями "КОНТРАКТ" и "ЗАКАЗЧИК", поскольку контракт не может существовать без заказчика.
* **Обязательное.** Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи "СОТРУДНИК" и "ОТДЕЛ". Если отдел расформировывается, все его сотрудники должны быть либо переведены в другие отделы, либо уволены.
* **Необязательное.** Можно исключить запись из группового отношения, но сохранить ее в базе данных не прикрепляя к другому владельцу. При *удалении записи*-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа. Примером такого группового отношения может служить "ВЫПОЛНЯЕТ" между "СОТРУДНИКИ" и "КОНТРАКТ", поскольку в организации могут существовать работники, чья деятельность не связана с выполнением каких-либо договорных обязательств перед заказчиками.

**Операции над данными в сетевой модели БД**

|  |  |
| --- | --- |
| **Добавить** | - внести запись в БД и, в зависимости от режима включения, либо включить ее в групповое отношение, где она объявлена подчиненной, либо не включать ни в какое групповое отношение. |
| **Включить в групповое отношение** | - связать существующую подчиненную запись с записью-владельцем. |
| **Переключить** | - связать существующую подчиненную запись с другой записью-владельцем в том же групповом отношении. |
| **Обновить** | - изменить значение элементов предварительно извлеченной записи. |
| **Извлечь** | - извлечь записи последовательно по значению ключа, а также используя групповые отношения - от владельца можно перейти к записям - членам, а от подчиненной записи к владельцу набора. |
| **Удалить** | - убрать из БД запись. Если эта запись является владельцем группового отношения, то анализируется класс членства подчиненных записей. Обязательные члены должны быть предварительно исключены из группового отношения, фиксированные удалены вместе с владельцем, необязательные останутся в БД. |
| **Исключить из группового отношения** | - разорвать связь между записью-владельцем и записью-членом. |

**Ограничения целостности**

Как и в *иерархической модели* обеспечивается только поддержание целостности по ссылкам (владелец отношения - член отношения).

**Достоинства и недостатки ранних СУБД**

**Достоинства ранних СУБД:**

* развитые средства управления данными во внешней памяти на низком уровне;
* возможность построения вручную эффективных прикладных систем;
* возможность экономии памяти за счет разделения подобъектов (в сетевых системах)

**Недостатки ранних СУБД:**

* сложность использования;
* высокий уровень требований к знаниям о физической организации БД;
* зависимость прикладных систем от физической организации БД;
* перегруженность логики прикладных систем деталями организации доступа к БД.

Как иерархическая, так и *сетевая модель данных* предполагает наличие высококвалифицированных программистов. И даже в таких случаях реализация пользовательских запросов часто затягивается на длительный срок.

**Объектно-ориентированные СУБД**

Появление объектно-ориентированных *СУБД* вызвано потребностями программистов на ОО-языках, которым были необходимы средства для хранения объектов, не помещавшихся в оперативной памяти компьютера. Также важна была задача сохранения состояния объектов между повторными запусками прикладной программы. Поэтому, большинство ООСУБД представляют собой библиотеку, процедуры управления данными которой включаются в прикладную программу. Примеры реализации ООСУБД как *выделеного сервера* *базы данных* крайне редки.

Сразу же необходимо заметить, что общепринятого определения "*объектно-ориентированной модели данных*" не существует. Сейчас можно говорить лишь о неком "объектном" подходе к логическому представлению данных и о различных объектно-ориентированных способах его реализации.

Мы знаем, что любая *модель данных* должна включать три аспекта: структурный, целостный и манипуляционный. Посмотрим, как они реализуются на основе объектно-ориентированная *парадигмы программирования*.

**Структура**

Структура объектной модели описывается с помощью трех ключевых понятий:

|  |  |
| --- | --- |
| **инкапсуляция** | - каждый объект обладает некоторым внутренним состоянием (хранит внутри себя запись данных), а также набором методов - процедур, с помощью которых (и только таким образом) можно получить доступ к данным, определяющим внутреннее состояние объекта, или изменить их. Таким образом, объекты можно рассматривать как самостоятельные сущности, отделенные от внешнего мира; |
| **наследование** | - подразумевает возможность создавать из классов объектов новые классы объекты, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков); |
| **полиморфизм** | - различные объекты могут по разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы. |

**Целостность данных**

Для поддержания целостности объектно-ориентированный подход предлагает использовать следующие средства:

* автоматическое поддержание отношений наследования возможность объявить некоторые поля данных и методы объекта как "скрытые", не видимые для других объектов; такие поля и методы используются только методами самого объекта создание процедур контроля целостности внутри объекта

**Средства манипулирования данными**

К сожалению, в объектно-ориентированном программировании отсутствуют общие средства манипулирования данными, такие как реляционная алгебра или реляционное счисление. Работа с данными ведется с помощью одного из объектно-ориентированных языков программирования общего назначения, обычно это SmallTalk, C++ или Java.

**Подведем теперь некоторые итоги**

В объектно-ориентированных базах данных, в отличие от реляционных, хранятся не записи, а объекты. ОО-подход представляет более совершенные средства для отображения реального мира, чем реляционная модель, естественное представление данных. В реляционной модели все отношения принадлежат одному уровню, именно это осложняет преобразование иерархических связей модели "сущность-связь" в реляционную модель. ОО - модель можно рассматривать послойно, на разных уровнях абстракции. Имеется возможность определения новых типов данных и операций с ними.

В то же время, ОО - *модели присущ и ряд недостатков*:

* отсутствуют мощные непроцедурные средства извлечения объектов из базы. Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста;
* вместо чисто декларативных *ограничений целостности* (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов **PRIMARY KEY** и **REFERENCES**) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код.

Очевидно, что оба эти недостатка связаны с отсутствием развитых средств манипулирования данными. Эта задача решается двумя способами - расширение ОО-языков в сторону управления данными (стандарт ODMG), либо добавление объектных свойств в реляционные СУБД (SQL-3, а также так называемые объектно-реляционных СУБД).

**Объектно-реляционные СУБД**

Разница между объектно-реляционными и объектными *СУБД*: первые являют собой надстройку над реляционной схемой, вторые же изначально объектно-ориентированы. Главная особенность и отличие объектно-реляционных, как и объектных, *СУБД* от реляционных заключается в том, что О(Р)*СУБД* интегрированы с Объектно-Ориентированным (OO) языком программирования, внутренним или внешним как C++, *Java*. Характерные свойства OРСУБД - 1) комплексные данные, 2) *наследование* типа, и 3) объектное поведение.

Комплексные данные могут быть реализованы через постоянно-хранимые объекты (*persistent* objects). Создание комплексных данных в большинстве существующих ОРСУБД основано на предварительном определении схемы через определяемый пользователем тип (*UDT* - *user-defined* type). Используются также встроенные конструкторы составных типов, например *массив* (*ARRAY*).

*Иерархия* структурных комплексных данных предлагает дополнительное свойство, *наследование* типа. То есть *структурный тип* может иметь подтипы, которые используют все его атрибуты и содержат дополнительные атрибуты, специфицированные в подтипе.

Объектное поведение закладывается через описание программных объектов. Такие объекты должны быть сохраняемыми и переносимыми для обработки в базе данных, поэтому они называются обычно как постоянные (или долговременные) объекты. Внутри *базы данных* все отношения с постоянным программным объектом есть отношения с его объектным идентификатором (*OID*).

Объектно-реляционными *СУБД* являются, к примеру, широко известные *Oracle* *Database*, Microsoft *SQL* *Server* 2005, PostgreSQL, а также Sav Zigzag, *IBM* Cloudscape.

**Краткие итоги**

Рассмотрены модели организации *БД*. Различают три *основные модели* *базы данных* - это иерархическая, сетевая и реляционная. Эти модели отличаются между собой по способу установления связей между данными.

Достоинства и недостатки ранних *СУБД*.

Рассмотрены более поздние модели *СУБД* такие как объектно-ориентированные и объектно-реляционные.

**Вопросы для самопроверки**

* *Иерархическая модель* БД ее характеристики.
* *Сетевая модель* БД ее характеристики.
* Реляционная модель БД ее характеристики.
* Понятие атрибута.
* Понятие записи.
* Понятие групповых отношений.
* Перечислите достоинства и недостатки ранних СУБД.
* Дайте характеристику объектно-ориентированным СУБД.
* Структура объектно-ориентированным СУБД.
* Дайте характеристику объектно-реляционным СУБД