Лекция 10:

**Моделирование информационного обеспечения**

### Моделирование данных

Одной из основных частей **информационного обеспечения** является **информационная база**. Как было определено выше (см. лекцию 9), *информационная база* (ИБ) представляет собой совокупность данных, организованную определенным способом и хранимую в памяти вычислительной системы в виде файлов, с помощью которых удовлетворяются информационные потребности управленческих процессов и решаемых задач. Разработка *БД* выполняется с помощью моделирования данных. **Цель моделирования данных** состоит в обеспечении разработчика ИС *концептуальной схемой базы* данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных. Наиболее распространенным **средством моделирования** данных являются **диаграммы "сущность-связь"** (*ERD*). С помощью *ERD* осуществляется *детализация* *накопителей данных* *DFD* – диаграммы, а также документируются информационные аспекты *бизнес-системы*, включая идентификацию объектов, важных для *предметной области* ( *сущностей* ), свойств этих объектов ( *атрибутов* ) и их *связей* с другими объектами (отношений).

#### Базовые понятия ERD

***Сущность*** (Entity) — множество экземпляров реальных или абстрактных объектов (людей, событий, состояний, идей, предметов и др.), обладающих общими *атрибутами* или характеристиками. Любой объект системы может быть представлен только одной *сущностью*, которая должна быть уникально идентифицирована. При этом имя *сущности* должно отражать тип или класс объекта, а не его конкретный экземпляр (например, АЭРОПОРТ, а не ВНУКОВО).

Каждая *сущность* должна обладать уникальным **идентификатором**. Каждый экземпляр *сущности* должен однозначно идентифицироваться и отличаться от всех других экземпляров данного *типа сущности*. Каждая *сущность* должна обладать некоторыми свойствами:

* иметь уникальное имя; к одному и тому же имени должна всегда применяться одна и та же интерпретация; одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
* иметь один или несколько *атрибутов*, которые либо принадлежат *сущности*, либо наследуются через *связь* ;
* иметь один или несколько *атрибутов*, которые однозначно идентифицируют каждый экземпляр *сущности*.

Каждая *сущность* может обладать любым количеством *связей* с другими *сущностями* модели.

***Связь*** (*Relationship*) — поименованная ассоциация между двумя *сущностями*, значимая для рассматриваемой предметной области. *Связь* — это ассоциация между *сущностями*, при которой каждый экземпляр одной *сущности* ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй *сущности*, и наоборот.

***Атрибут*** (Attribute) — любая характеристика *сущности*, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния *сущности* . *Атрибут* представляет тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, предметов и т.д.). Экземпляр *атрибута* — это определенная характеристика отдельного элемента множества. **Экземпляр***атрибута* определяется типом характеристики и ее значением, называемым **значением***атрибута*. На диаграмме "*сущность-связь*" *атрибуты* ассоциируются с конкретными *сущностями*. Таким образом, экземпляр *сущности* должен обладать единственным определенным значением для ассоциированного *атрибута*.

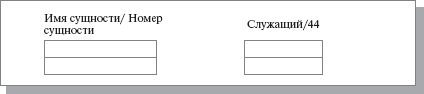
#### Метод IDEFI

Наиболее распространенными методами для построения *ERD*-диаграмм являются метод Баркера и метод IDEFI.

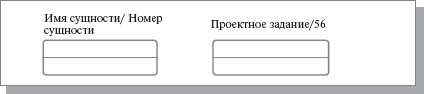
Метод Баркера основан на нотации, предложенной автором, и используется в case-средстве Oracle Designer.

**Метод IDEFI** основан на подходе Чена и позволяет построить *модель данных*, эквивалентную реляционной модели в *третьей нормальной форме*. На основе совершенствования метода IDEFI создана его новая версия — метод IDEFIX, разработанный с учетом таких требований, как простота для изучения и возможность автоматизации. IDEFIX-диаграммы используются в ряде распространенных CASE-средств (в частности, ERwin, Design/*IDEF*).

В методе IDEFIX *сущность* является независимой от идентификаторов или просто независимой, если каждый экземпляр *сущности* может быть однозначно идентифицирован без определения его отношений с другими *сущностями*. *Сущность* называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра *сущности* зависит от его отношения к другой *сущности* ([рис. 10.1](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.1), [10.2](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.2)).



**Рис. 10.1.**Независимые от идентификации сущности



**Рис. 10.2.**Зависимые от идентификации сущности

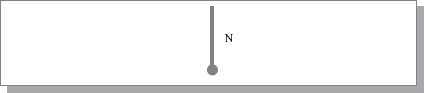
Каждой *сущности* присваиваются уникальные имя и номер, разделяемые косой чертой "/" и помещаемые над блоком.

*Связь* может дополнительно определяться с помощью указания степени или мощности (количества *экземпляров сущности*-потомка, которое может порождать каждый *экземпляр сущности*-родителя). В IDEFIX могут быть выражены следующие *мощности связей*:

* каждый *экземпляр сущности*-родителя может иметь ноль, один или более одного связанного с ним *экземпляра сущности*-потомка;
* каждый *экземпляр сущности*-родителя должен иметь не менее одного связанного с ним *экземпляра сущности*-потомка;
* каждый *экземпляр сущности*-родителя должен иметь не более одного связанного с ним *экземпляра сущности*-потомка;
* каждый *экземпляр сущности*-родителя связан с некоторым фиксированным числом *экземпляров сущности*-потомка.

Если *экземпляр сущности*-потомка однозначно определяется своей *связью* с сущностью-родителем, то *связь* называется идентифицирующей, в противном случае — неидентифицирующей.

*Связь* изображается линией, проводимой между сущностью-родителем и сущностью-потомком, с точкой на конце линии у сущности-потомка ([рис. 10.3](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.3)). *Мощность связей* может принимать следующие значения: N — ноль, один или более, Z — ноль или один, Р — один или более. По умолчанию *мощность связей* принимается равной N.



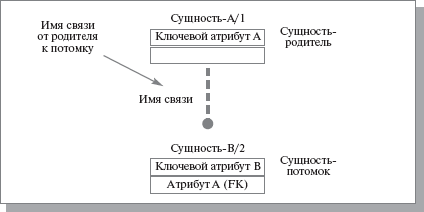
**Рис. 10.3.**Графическое изображение мощности связи

Идентифицирующая *связь* между сущностью-родителем и сущностью-потомком изображается сплошной линией. Сущность-потомок в идентифицирующей *связи* является зависимой от идентификатора *сущностью*. Сущность-родитель в идентифицирующей *связи* может быть как независимой, так и зависимой от идентификатора *сущностью* (это определяется ее *связями* с другими *сущностями* ).

Пунктирная линия изображает неидентифицирующую *связь* ([рис. 10.4](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.4)). Сущность-потомок в неидентифицирующей *связи* будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей *связи*.

*Атрибуты* изображаются в виде списка имен внутри блока *сущности*. *Атрибуты*, определяющие *первичный ключ*, размещаются наверху списка и отделяются от других *атрибутов* горизонтальной чертой ([рис. 10.4](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.4) ).

*Сущности* могут иметь также **внешние ключи** (*Foreign* Key), которые могут использоваться в качестве части или целого *первичного ключа* или неключевого *атрибута*. Для обозначения внешнего ключа внутрь блока *сущности* помещают имена *атрибутов*, после которых следуют буквы FK в скобках ([рис. 10.4](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=1#image.10.4)).



**Рис. 10.4.**Неидентифицирующая связь

### Отображение модели данных в инструментальном средстве ERwin

ERwin имеет два уровня *представления* модели — *логический* и физический.

**Логический уровень — это абстрактный взгляд на данные, когда данные представляются так, как выглядят в реальном мире**, и могут называться так, как они называются в реальном мире, например "Постоянный клиент", "Отдел" или "Фамилия сотрудника". Объекты модели, представляемые на логическом уровне, называются *сущностями* и *атрибутами*. *Логическая модель данных* может быть построена на основе другой *логической модели*, например на основе модели процессов. *Логическая модель данных* является универсальной и никак не связана с конкретной реализацией *СУБД*.

*Физическая модель данных*, напротив, **зависит от конкретной СУБД, фактически являясь отображением системного каталога**. В *физической модели* содержится *информация* обо всех объектах *БД*. Поскольку стандартов на объекты *БД* не существует (например, нет стандарта на типы данных), *физическая модель* зависит от конкретной реализации *СУБД*. Следовательно, одной и той же *логической модели* могут соответствовать несколько разных физических моделей. Если в *логической модели* не имеет значения, какой конкретно *тип данных* имеет *атрибут*, то в *физической модели* важно описать всю информацию о конкретных физических объектах — таблицах, колонках, *индексах*, процедурах и т.д.

#### Документирование модели

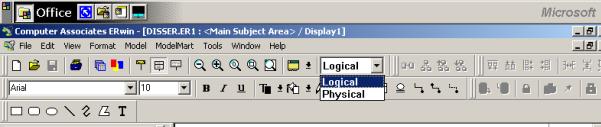
Многие СУБД имеют ограничение на именование объектов (например, ограничение на длину имени таблицы или запрет использования специальных символов — пробела и т. п.). Зачастую разработчики ИС имеют дело с нелокализованными версиями СУБД. Это означает, что объекты БД могут называться короткими словами, только латинскими символами и без использования специальных символов (т. е. нельзя назвать таблицу, используя предложение — ее можно назвать только одним словом). Кроме того, проектировщики БД нередко злоупотребляют "техническими" наименованиями, в результате таблица и колонки получают наименования типа RTD\_324 или CUST\_A12 и т.д. Полученную в результате структуру могут понять только специалисты (а чаще всего — только авторы модели), ее невозможно обсуждать с экспертами предметной области. Разделение модели на логическую и физическую позволяет решить эту проблему. На физическом уровне объекты БД могут называться так, как того требуют ограничения СУБД. На логическом уровне можно этим объектам дать синонимы — имена более понятные неспециалистам, в том числе на кириллице и с использованием специальных символов. Например, таблице **CUST\_A12** может соответствовать *сущность* **Постоянный клиент**. Такое соответствие позволяет лучше документировать модель и дает возможность обсуждать структуру данных с экспертами предметной области.

#### Масштабирование

Создание *модели данных*, как правило, начинается с разработки *логической модели*. После описания *логической модели* проектировщик может выбрать необходимую СУБД, и ERwin автоматически создаст соответствующую *физическую модель*. На основе *физической модели* ERwin может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Этот процесс называется *прямым проектированием* (*Forward Engineering*). Тем самым достигается масштабируемость — создав одну *логическую модель данных*, можно сгенерировать физические модели под любую поддерживаемую ERwin СУБД. С другой стороны, ERwin способен по содержимому системного каталога или SQL-скрипту воссоздать физическую и *логическую модель данных* (*Reverse Engineering*). На основе полученной *логической модели данных* можно сгенерировать *физическую модель* для другой СУБД и затем создать ее системный каталог. Следовательно, ERwin позволяет решить задачу по переносу структуры данных с одного сервера на другой. Например, можно перенести структуру данных с Oracle на *Informix* (или наоборот) или перенести структуру dbf-файлов в реляционную СУБД, тем самым облегчив переход от *файл-серверной* к клиент-серверной ИС. Однако, формальный перенос структуры "плоских" таблиц на реляционную СУБД обычно неэффективен. Для того чтобы извлечь выгоды от перехода на клиент-серверную технологию, структуру данных следует модифицировать.

Для переключения между логической и *физической моделью данных* служит список выбора в центральной части панели инструментов ERwin ([рис. 10.5](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=2#image.10.5)).

Если при переключении *физической модели* еще не существует, она будет создана автоматически.



**Рис. 10.5.**Переключение между логической и физической моделью

Интерфейс ERwin. Уровни отображения модели

Интерфейс выполнен в стиле *Windows-приложений*, достаточно прост и интуитивно понятен. Рассмотрим кратко основные функции ERwin по отображению модели.

Каждому уровню отображения модели соответствует своя палитра инструментов. На **логическом уровне** палитра инструментов имеет следующие кнопки:

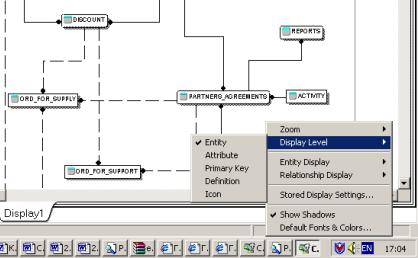
* кнопку указателя (режим мыши) — в этом режиме можно установить фокус на каком-либо объекте модели;
* кнопку внесения *сущности* ;
* кнопку категории (категория, или категориальная *связь*, — специальный тип *связи* между *сущностями*, которая будет рассмотрена ниже);
* кнопку внесения текстового блока;
* кнопку перенесения *атрибутов* внутри *сущностей* и между ними;
* кнопки создания *связей*: идентифицирующую, "многие-ко-многим" и неидентифицирующую.

На **физическом уровне** палитра инструментов имеет:

* вместо кнопки категорий — кнопку внесения *представлений* (view);
* вместо кнопки *связи* "многие-ко-многим" — кнопку *связей* *представлений*.

Для создания *моделей данных* в ERwin можно использовать две нотации: IDEFIX и IE (Information Engineering). В дальнейшем будет рассматриваться нотация IDEFIX.

ERwin имеет несколько уровней отображения диаграммы: уровень *сущностей*, уровень *атрибутов*, уровень определений, уровень *первичных ключей* и уровень иконок. Переключиться между первыми тремя уровнями можно с использованием кнопок панели инструментов. Переключиться на другие уровни отображения можно при помощи контекстного меню, которое появляется, если "кликнуть" по любому месту диаграммы, не занятому объектами модели. В контекстном меню следует выбрать пункт Display Level ([рис. 10.6](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=2#image.10.6)) и затем — необходимый уровень отображения.



**Рис. 10.6.**Выбор уровней отображения диаграммы

### Создание логической модели данных

#### Уровни логической модели

Различают три уровня *логической модели*, отличающихся по глубине *представления* информации о данных:

* *диаграмма сущность-связь* (Entity *Relationship* Diagram, *ERD*);
* *модель данных*, *основанная на ключах* (Key *Based model*, KB);
* *полная атрибутивная модель* (Fully Attributed model, FA).

***Диаграмма сущность-связь*** представляет собой *модель данных* верхнего уровня. Она включает *сущности* и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные *сущности* и *связи* между ними, которые удовлетворяют основным требованиям, предъявляемым к ИС. *Диаграмма сущность-связь* может включать *связи* "многие-ко-многим" и не включать описание ключей. Как правило, *ERD* используется для презентаций и обсуждения структуры данных с экспертами предметной области.

***Модель данных***, *основанная на ключах*, — более подробное *представление* данных. Она включает описание всех *сущностей* и *первичных ключей* и предназначена для *представления* структуры данных и ключей, которые соответствуют предметной области.

***Полная атрибутивная модель*** — наиболее детальное *представление* структуры данных: представляет данные в *третьей нормальной форме* и включает все *сущности*, *атрибуты* и *связи* .

#### Сущности и атрибуты

Основные компоненты диаграммы ERwin — это *сущности*, *атрибуты* и *связи*. Каждая *сущность* является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. *Атрибут* выражает определенное свойство объекта. С точки зрения БД (*физическая модель*) *сущности* соответствует таблица, экземпляру *сущности* — строка в таблице, а *атрибуту* — колонка таблицы.

Построение *модели данных* предполагает определение *сущностей* и *атрибутов*, т. е. необходимо определить, какая информация будет храниться в конкретной *сущности* или *атрибуте*. *Сущность* можно определить **как объект, событие или концепцию, информация о которых должна сохраняться**. *сущности* должны иметь наименование с четким смысловым значением, именоваться существительным в единственном числе, не носить "технических" наименований и быть достаточно важными для того, чтобы их моделировать. Именование *сущности* в единственном числе облегчает в дальнейшем чтение модели. Фактически имя *сущности* дается по имени ее экземпляра. Примером может быть *сущности* Заказчик (но не Заказчики!) с *атрибутами* Номер заказчика, Фамилия заказчика и Адрес заказчика. На уровне *физической модели* ей может соответствовать таблица Customer с колонками Customer\_number, Customer\_name и Customer\_address. Каждая *сущность* должна быть полностью определена с помощью текстового описания. Для внесения дополнительных комментариев и определений к *сущности* служат свойства, определенные пользователем (UDP). Использование (UDP) аналогично их использованию в BPwin.

Как было указано выше, каждый *атрибут* хранит информацию об **определенном свойстве***сущности*, а каждый экземпляр *сущности* должен быть уникальным. *Атрибут* или группа *атрибутов*, которые идентифицируют *сущность*, называется ***первичным ключом*** .

Очень важно дать *атрибуту* правильное имя. *Атрибуты* должны именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения *атрибутов*. Например, создание в *сущности* Сотрудник *атрибута* Телефоны сотрудника противоречит требованиям нормализации, поскольку *атрибут* должен быть атомарным, т. е. не содержать множественных значений. Согласно синтаксису IDEFIX имя *атрибута* должно быть уникально в рамках модели (а не только в рамках *сущности*!). По умолчанию при попытке внесения уже существующего имени *атрибута* ERwin переименовывает его.

Каждый *атрибут* должен быть определен, при этом следует избегать циклических определений, например, когда термин 1 определяется через термин 2, термин 2 — через термин 3, а термин 3 в свою очередь — через термин 1. Часто приходится создавать производные *атрибуты*, т. е. *атрибуты*, значение которых можно вычислить из других *атрибутов*. Примером производного *атрибута* может служить Возраст сотрудника, который может быть вычислен из *атрибута* Дата рождения сотрудника. Такой *атрибут* может привести к конфликтам; действительно, если вовремя не обновить значение *атрибута* Возраст сотрудника, он может противоречить значению *атрибута* Дата рождения сотрудника. Производные *атрибуты* — ошибка нормализации, однако их вводят для повышения производительности системы, чтобы не проводить вычисления, которые на практике могут быть сложными.

#### Связи

*Связь* является логическим соотношением между *сущностями*. Каждая *связь* должна именоваться глаголом или глагольной фразой. *Имя связи* выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы. По умолчанию *имя связи* на диаграмме не показывается. На логическом уровне можно установить идентифицирующую *связь* "один-ко-многим", *связь* "многие-ко-многим" и неидентифицирующую *связь* "один-ко-многим".

В IDEFIX различают зависимые и независимые *сущности*. *Тип сущности* определяется ее *связью* с другими *сущностями*. Идентифицирующая *связь* устанавливается между независимой (родительский конец *связи* ) и зависимой (дочерний конец *связи* ) *сущностями*. Когда рисуется идентифицирующая *связь*, ERwin автоматически преобразует дочернюю *сущность* в зависимую. Зависимая *сущность* изображается прямоугольником со скругленными углами. Экземпляр зависимой *сущности* определяется только через отношение к родительской *сущности*. При установлении идентифицирующей *связи**атрибуты* *первичного ключа* родительской *сущности* автоматически переносятся в состав *первичного ключа* дочерней *сущности*. Эта *операция дополнения* *атрибутов* дочерней *сущности* при создании *связи* называется миграцией *атрибутов*. В дочерней *сущности* новые *атрибуты* помечаются как внешний ключ — FK.

При установлении неидентифицирующей *связи* дочерняя *сущность* остается независимой, а *атрибуты* *первичного ключа* родительской *сущности* мигрируют в состав неключевых компонентов дочерней *сущности*. Неидентифицирующая *связь* служит для связывания независимых *сущностей*.

Идентифицирующая *связь* показывается на диаграмме сплошной линией с жирной точкой на дочернем конце *связи*, неидентифицирующая – пунктирной (см. [рис. 10.6](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=2#image.10.6)).

***Мощность связей*** (*Cardinality*) — служит для обозначения отношения числа экземпляров родительской *сущности* к числу экземпляров дочерней.

Различают четыре *типа сущности*:

* общий случай, когда одному экземпляру родительской *сущности* соответствуют 0, 1 или много экземпляров дочерней *сущности* ; не помечается каким-либо символом;
* символом Р помечается случай, когда одному экземпляру родительской *сущности* соответствуют 1 или много экземпляров дочерней *сущности* (исключено нулевое значение);
* символом Z помечается случай, когда одному экземпляру родительской *сущности* соответствуют 0 или 1 экземпляр дочерней *сущности* (исключены множественные значения);
* цифрой помечается случай точного соответствия, когда одному экземпляру родительской *сущности* соответствует заранее заданное число экземпляров дочерней *сущности*.

***Имя связи*** (*Verb* Phrase) — фраза, характеризующая отношение между родительской и дочерней *сущностями* . Для *связи* "один-ко-многим", идентифицирующей или неидентифицирующей, достаточно указать имя, характеризующее отношение от родительской к дочерней *сущности* (Parent-to-Child). Для *связи* многие-ко-многим следует указывать имена как Parent-to-Child, так и Child-to-Parent.

#### Типы сущностей и иерархия наследования

Как было указано выше, *связи* определяют, является ли *сущность* независимой или зависимой. Различают несколько **типов зависимых***сущностей*.

**Характеристическая** — зависимая дочерняя *сущность*, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской *сущности* ([рис. 10.7](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.7)).

Пример характеристической сущности "Хобби"

**Рис. 10.7.**Пример характеристической сущности "Хобби"

**Ассоциативная** — *сущность*, связанная с несколькими родительскими *сущностями*. Такая *сущность* содержит информацию о *связях* *сущностей*.

**Именующая** — частный случай ассоциативной *сущности*, не имеющей собственных *атрибутов* (только *атрибуты* родительских *сущностей*, мигрировавших в качестве внешнего ключа).

**Категориальная** — дочерняя *сущность* в *иерархии наследования*.

***Иерархия наследования*** (или иерархия категорий) представляет собой особый тип объединения *сущностей*, которые разделяют общие характеристики. Например, в организации работают служащие, занятые полный рабочий день (постоянные служащие), и совместители. Из их общих свойств можно сформировать обобщенную *сущность* (родовой предок) Сотрудник ([рис. 10.8](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.8)), чтобы представить информацию, общую для всех типов служащих. Специфическая для каждого типа информация может быть расположена в категориальных *сущностях* (потомках) Постоянный сотрудник и Совместитель.

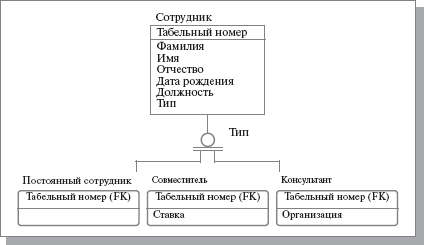
Обычно *иерархию наследования* создают, когда несколько *сущностей* имеют общие по смыслу *атрибуты*, либо когда *сущности* имеют общие по смыслу *связи* (например, если бы Постоянный сотрудник и Совместитель имели сходную по смыслу *связь* "работает в" с *сущностью* Организация ), либо когда это диктуется бизнес-правилами.

Для каждой категории можно указать дискриминатор — *атрибут* родового предка, который показывает, как отличить одну категориальную *сущность* от другой ( *атрибут* Тип на [рис. 10.8](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.8)).



**Рис. 10.8.**Иерархия наследования. Неполная категория

Иерархии категорий делятся на два типа — полные и неполные. В полной категории одному экземпляру родового предка ( *сущность* Cjn, [рис. 10.9](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.9)) обязательно соответствует экземпляр в каком-либо потомке, т. е. в примере сотрудник обязательно является либо совместителем, либо консультантом, либо постоянным сотрудником.



**Рис. 10.9.**Иерархия наследования. Полная категория

Если категория еще не выстроена полностью и в родовом предке могут существовать экземпляры, которые не имеют соответствующих экземпляров в потомках, то такая категория будет неполной. На [рис. 10.8](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.8) показана неполная категория — сотрудник может быть не только постоянным или совместителем, но и консультантом, однако *сущность* Консультант еще не внесена в *иерархию наследования*.

#### Ключи

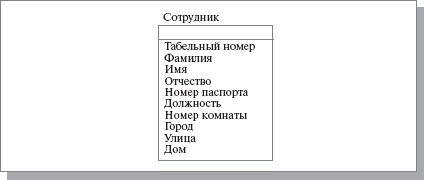
Как было сказано выше, каждый экземпляр *сущности* должен быть уникален и должен отличаться от других *атрибутов*.

***Первичный ключ*** (primary key) — это *атрибут* или группа *атрибутов*, однозначно идентифицирующая экземпляр *сущности* . *атрибуты* *первичного ключа* на диаграмме не требуют специального обозначения — это те *атрибуты*, которые находятся в списке *атрибутов* выше горизонтальной линии (см., например, [рис. 10.9](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.9)).

В одной *сущности* могут оказаться несколько *атрибутов* или наборов *атрибутов*, претендующих на роль *первичного ключа*. Такие претенденты называются ***потенциальными ключами*** (candidate key).

Ключи могут быть сложными, т. е. содержащими несколько *атрибутов*. Сложные *первичные ключи* не требуют специального обозначения — это список *атрибутов*, расположенных выше горизонтальной линии.

Рассмотрим кандидатов на роль *первичного ключа* *сущности* Сотрудник ([рис. 10.10](https://intuit.ru/studies/courses/2195/55/lecture/1636?page=4#image.10.10)).



**Рис. 10.10.**Определение первичного ключа для сущности "Сотрудник"

Здесь можно выделить следующие *потенциальные ключи*:

1. Табельный номер ;
2. Номер паспорта ;
3. Фамилия + Имя + Отчество.

Для того чтобы стать первичным, *потенциальный ключ* должен удовлетворять ряду требований:

**Уникальность**. Два экземпляра не должны иметь одинаковых значений *возможного ключа*. *потенциальный ключ* № 3 ( Фамилия + Имя + Отчество ) является плохим кандидатом, поскольку в организации могут работать полные тезки.

**Компактность**. Сложный возможный ключ не должен содержать ни одного *атрибута*, удаление которого не приводило бы к утрате уникальности. Для обеспечения *уникальности ключа* № 3 дополним его *атрибутами* Дата рождения и Цвет волос. Если бизнес-правила говорят, что сочетания *атрибутов* Фамилия + Имя + Отчество + Дата рождения достаточно для однозначной идентификации сотрудника, то Цвет волос оказывается лишним, т. е. ключ Фамилия + Имя + Отчество + Дата рождения + Цвет волос не является компактным.

При выборе *первичного ключа* предпочтение должно отдаваться более простым ключам, т. е. ключам, содержащим меньшее количество *атрибутов*. В приведенном примере ключи № 1 и 2 предпочтительней ключа № 3.

*Атрибуты* ключа не должны содержать нулевых значений. Значение *атрибутов* ключа не должно меняться в течение всего времени существования экземпляра *сущности*. Сотрудница организации может выйти замуж и сменить как фамилию, так и паспорт. Поэтому ключи № 2 и 3 не подходят на роль *первичного ключа*.

Каждая *сущность* должна иметь по крайней мере один *потенциальный ключ*. Многие *сущности* имеют только один *потенциальный ключ*. Такой ключ становится первичным. Некоторые *сущности* могут иметь более одного *возможного ключа*. Тогда один из них становится первичным, а остальные — *альтернативными ключами*.

***Альтернативный ключ*** (Alternate Key) — это *потенциальный ключ*, не ставший первичным.

#### Нормализация данных

Нормализация данных — **процесс проверки и реорганизации***сущностей* **и***атрибутов* **с целью удовлетворения требований к реляционной***модели данных*. Нормализация позволяет быть уверенным, что каждый *атрибут* определен для своей *сущности*, а также значительно сократить объем памяти для хранения информации и устранить аномалии в организации хранения данных. В результате проведения нормализации должна быть создана структура данных, при которой информация о каждом факте хранится только в одном месте. *Процесс нормализации* сводится к последовательному приведению структуры данных к нормальным формам — формализованным требованиям к организации данных. Известны шесть нормальных форм.

На практике обычно ограничиваются приведением данных к *третьей нормальной форме*. Для углубленного изучения нормализации рекомендуется книга К. Дж. Дейта "Введение в системы баз данных" (Киев; М.: *Диалектика*, 1998).

ERwin не содержит полного алгоритма нормализации и не может проводить нормализацию автоматически, однако его возможности облегчают создание нормализованной *модели данных*. Запрет на присвоение неуникальных имен *атрибутов* в рамках модели (при соответствующей установке опции Unique Name) облегчает соблюдение правила "один факт — в одном месте". Имена ролей *атрибутов* внешних ключей и унификация *атрибутов* также облегчают построение нормализованной модели.

#### Домены

***Домен*** можно определить как совокупность значений, из которых берутся значения *атрибутов* . Каждый *атрибут* может быть определен только на одном *домене*, но на каждом *домене* может быть определено множество *атрибутов*. В понятие *домена* входит не только тип данных, но и область значений данных. Например, можно определить *домен* "Возраст" как положительное целое число и определить *атрибут* Возраст сотрудника как принадлежащий этому *домену*.

В ERwin *домен* может быть определен только один раз и использоваться как в логической, так и в *физической модели*.

*Домены* позволяют облегчить работу с данными как разработчикам на этапе проектирования, так и администраторам БД на этапе эксплуатации системы. На логическом уровне *домены* можно описать без конкретных физических свойств. На физическом уровне они автоматически получают специфические свойства, которые можно изменить вручную. Так, *домен* "Возраст" может иметь на логическом уровне тип Number, на физическом уровне колонкам *домена* будет присвоен тип INTEGER.

Каждый *домен* может быть описан, снабжен комментарием или свойством, определенным пользователем (UDP).

#### Создание физической модели данных

*Физическая модель* содержит всю информацию, необходимую для реализации конкретной БД. Различают два уровня *физической модели*:

* трансформационную модель;
* модель СУБД.

Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. Данная модель позволяет проектировщикам и администраторам БД лучше представить, какие объекты БД хранятся в словаре данных, и проверить, насколько *физическая модель* удовлетворяет требованиям к ИС.

**Модель СУБД** автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД.

*Физический уровень* *представления* модели зависит от выбранного сервера. ERwin поддерживает более 20 реляционных и нереляционных БД.

По умолчанию ERwin генерирует имена таблиц и *индексов* по шаблону на основе имен соответствующих *сущностей* и ключей *логической модели*, которые в дальнейшем могут быть откорректированы вручную. Имена таблиц и колонок будут сгенерированы по умолчанию на основе имен *сущностей* и *атрибутов* *логической модели*.

#### Правила валидации и значения по умолчанию

ERwin поддерживает *правила валидации* для колонок, а также значение, присваиваемое колонкам по умолчанию.

*Правило валидации* задает список допустимых значений для конкретной колонки и/или правила *проверки допустимых* значений. В список допустимых значений можно вносить новые значения. ERwin позволяет сгенерировать *правила валидации* соответственно синтаксису выбранной СУБД с учетом границ диапазона или списка значений.

**Значение по умолчанию** – значение, которое нужно ввести в колонку, если никакое другое значение не задано явным образом во время ввода данных. С каждой колонкой или *доменом* можно связать значение по умолчанию. Список значений можно редактировать.

После создания *правила валидации* и значения по умолчанию их можно присвоить одной или нескольким колонкам или *доменами*.

#### Индексы

В БД данные обычно хранятся в том порядке, в котором их ввели в таблицу. Многие реляционные СУБД имеют *страничную организацию*, при которой таблица может храниться фрагментарно в разных областях диска, причем строки таблицы располагаются на страницах неупорядоченно. Такой способ позволяет быстро вводить новые данные, но затрудняет поиск данных.

Чтобы решить проблему поиска, СУБД используют объекты, называемые *индексами*. ***Индекс*** содержит отсортированную по колонке или нескольким колонкам информацию и указывает на строки, в которых хранится конкретное значение колонки. Поскольку значения в *индексе* хранятся в определенном порядке, при поиске просматривать нужно значительно меньший объем данных, что существенно уменьшает время выполнения запроса. *Индекс* рекомендуется создавать для тех колонок, по которым часто производится поиск.

При генерации схемы физической БД ERwin автоматически создает *индекс* на основе *первичного ключа* каждой таблицы, а также на основе всех *альтернативных ключей* и внешних ключей, поскольку эти колонки наиболее часто используются для поиска данных. Можно отказаться от генерации *индексов* по умолчанию и создать собственные *индексы*. Для увеличения эффективности поиска *администратор БД* должен анализировать часто выполняемые запросы и на основе анализа создавать собственные *индексы*.

#### Триггеры и хранимые процедуры

*Триггеры* и *хранимые процедуры* **– это именованные блоки кода SQL, которые заранее откомпилированы и хранятся на сервере для того, чтобы быстро производить обработку запросов, валидацию данных и другие часто выполняемые функции**. Хранение и выполнение кода на сервере позволяет создавать код только один раз, а не в каждом приложении, работающем с БД. Это экономит время при написании и сопровождении программ. При этом гарантируется, что целостность данных и бизнес-правила поддерживаются независимо от того, какое именно клиентское приложение обращается к данным. *Триггеры* и *хранимые процедуры* не требуется пересылать по сети из клиентского приложения, что значительно снижает сетевой трафик.

***Хранимой процедурой*** называется именованный набор предварительно откомпилированных команд SQL, который может вызываться из клиентского приложения или из другой *хранимой процедуры* .

***Триггером*** называется процедура, которая выполняется автоматически как реакция на событие. Таким событием может быть вставка, изменение или удаление строки в существующей таблице. *Триггер* сообщает СУБД, какие действия нужно выполнить при выполнении команд SQL INSERT, UPDATE или DELETE для обеспечения дополнительной функциональности, выполняемой на сервере.

*Триггер* **ссылочной целостности** – это особый вид *триггера*, используемый для поддержания целостности между двумя таблицами, которые связаны между собой. Если строка в одной таблице вставляется, изменяется или удаляется, то *триггер* *ссылочной целостности* сообщает СУБД, что нужно делать с теми строками в других таблицах, у которых значение внешнего ключа совпадает со значением *первичного ключа* вставленной строки (измененной или удаленной строки).

Для генерации *триггеров* ERwin использует механизм шаблонов – специальных скриптов, использующих *макрокоманды*. При генерации кода *триггера* вместо *макрокоманд* подставляются имена таблиц, колонок, переменные и другие фрагменты кода, соответствующие синтаксису выбранной СУБД. Шаблоны *триггеров* *ссылочной целостности*, генерируемые ERwin по умолчанию, можно изменять.

Для создания и редактирования *хранимых процедур* ERwin располагает специальными редакторами, аналогичными редакторам, используемым для создания *триггеров*. В отличие от *триггера* *хранимая процедура* не выполняется в ответ на какое-то событие, а вызывается из другой программы, которая передает на сервер имя процедуры. *Хранимая процедура* более гибкая, чем *триггер*, поскольку может вызывать другие *хранимые процедуры*. Ей можно передавать параметры, и она может возвращать параметры, значения и сообщения.

### Проектирование хранилищ данных

В хранилища данных помещают данные, которые редко меняются. Хранилища ориентированы на выполнение аналитических запросов, обеспечивающих поддержку *принятия решений* для руководителей и менеджеров. При проектировании хранилищ данных необходимо выполнять следующие требования:

* хранилище должно иметь понятную для пользователей структуру данных;
* должны быть выделены статические данные, которые модифицируются по расписанию (ежедневно, еженедельно, ежеквартально);
* должны быть упрощены требования к запросам для исключения запросов, требующих множественных утверждений SQL в традиционных реляционных СУБД;
* должна обеспечиваться поддержка сложных запросов SQL, требующих обработки миллионов записей.

Как видно из этих требований, по своей структуре реляционные *СУБД* существенно отличаются от хранилищ данных. *Нормализация* данных в реляционных *СУБД* приводит к созданию *множества* связанных между собой таблиц. Выполнение сложных запросов неизбежно приводит к объединению многих таблиц, что значительно увеличивает время отклика. Проектирование хранилища данных подразумевает создание денормализованной структуры данных, ориентированных в первую *очередь* на высокую *производительность* при выполнении аналитических запросов. *Нормализация* делает модель хранилища слишком сложной, затрудняет ее понимание и снижает скорость выполнения запроса. Для эффективного проектирования хранилищ данных ERwin использует размерную модель – методологию проектирования, предназначенную специально для разработки хранилищ данных. Размерное *моделирование* сходно с моделированием *связей* и *сущностей* для реляционной модели, но имеет другую цель. Реляционная модель акцентируется на целостности и эффективности ввода данных. Размерная модель ориентирована в первую *очередь* на выполнение сложных запросов

В размерном моделировании принят стандарт модели, называемый схемой "*звезда*", которая обеспечивает высокую скорость выполнения запроса посредством денормализации и разделения данных. Невозможно создать универсальную структуру данных, обеспечивающую высокую скорость обработки любого запроса, поэтому схема "*звезда*" строится для обеспечения наивысшей производительности при выполнении самого важного запроса (или группы запросов).

Схема "*звезда*" обычно содержит одну большую таблицу, называемую таблицей факта, помещенную в центре. Ее окружают меньшие таблицы, называемые таблицами размерности, которые связаны с таблицей факта радиальными *связями*.

Для создания *БД* со схемой "*звезда*" необходимо проанализировать бизнес-правила *предметной области* для выяснения центрального запроса. Данные, обеспечивающие выполнение этого запроса, должны быть помещены в центральную таблицу. При проектировании хранилища важно определить *источник данных*, метод, которым данные извлекаются, преобразуются и фильтруются, прежде чем они импортируются в хранилище. Знания об источнике данных позволяют поддерживать регулярное обновление и проверку качества данных.

#### Вычисление размера БД

ERwin позволяет рассчитать приблизительный размер БД в целом, а также таблиц, *индексов* и других объектов через определенный период времени после начала эксплуатации ИС. Расчет строится на основе следующих параметров: начальное количество строк; максимальное количество строк; прирост количества строк в месяц. Результаты расчетов сводятся в отчет.

#### Прямое и обратное проектирование

***Прямым проектированием*** называется процесс генерации физической схемы БД из *логической модели*. При генерации физической схемы ERwin включает *триггеры* *ссылочной целостности*, *хранимые процедуры*, *индексы*, ограничения и другие возможности, доступные при определении таблиц в выбранной СУБД.

***Обратным проектированием*** называется процесс генерации *логической модели* из физической БД. *Обратное проектирование* позволяет конвертировать БД из одной СУБД в другую. После создания *логической модели* БД путем *обратного проектирования* можно переключиться на другой сервер и произвести *прямое проектирование*.

Кроме режима прямого и *обратного проектирования* программа обеспечивает синхронизацию между логической моделью и системным каталогом СУБД на протяжении всего жизненного цикла создания ИС.

### Генерация кода клиентской части с помощью ERwin

#### Расширенные атрибуты

ERwin поддерживает не только проектирование сервера БД, но и автоматическую генерацию клиентского приложения в средах разработки MS Visual Basic и Power Builder. Технология генерации состоит в том, что на этапе разработки *физической модели данных* каждой колонке присваиваются расширенные *атрибуты*, содержащие информацию о свойствах объектов клиентского приложения (в том числе и визуальных), которые будут отображать информацию, хранящуюся в соответствующей колонке. Эта информация записывается в файле модели. На основе информации, содержащейся в расширенных *атрибутах*, генерируются экранные формы. Полученный код может быть откомпилирован и выполнен без дополнительного ручного кодирования.

Каждой колонке в модели ERwin можно задать предварительно описанные и именованные свойства:

* *правила валидации* (проверка значений);
* начальные значения, устанавливаемые по умолчанию;
* стиль визуального объекта (например, радиокнопка, поле ввода и др.);
* формат изображения.

Для описания каждого свойства ERwin содержит соответствующие редакторы.

#### Генерация кода в Visual Basic

ERwin поддерживает генерацию кода в Visual Basic версий 4.0 и 5.0. В качестве источника информации при генерации форм служит модель ERwin. С помощью ERwin можно одновременно описывать как клиентскую часть (объекты, отображающие данные на экране), так и сервер БД (процедуры и *триггеры* ), тем самым оптимально распределяя функциональность ИС между клиентской и серверной частью. Компонент ERwin Form Wizard автоматически проектирует формы с дочерними объектами – кнопками, списками, полями, радиокнопками и т. д., используя расширенные *атрибуты*. Совместное использование ERwin и Visual Basic позволяет сократить жизненный цикл разработки ИС путем употребления для каждой задачи наиболее эффективного инструмента. Visual Basic может быть использован для проектирования визуального интерфейса, а ERwin – для разработки физической и *логической модели данных* с последующей генерацией системного каталога сервера. Если БД уже существует, то с помощью ERwin можно провести *обратное проектирование*, полученную модель дополнить расширенными *атрибутами* и сгенерировать клиентское приложение.

#### Создание отчетов

Для генерации отчетов в ERwin имеется простой и эффективный инструмент – Report Browser. По умолчанию Report Browser содержит предварительно определенные отчеты, позволяющие наглядно представить информацию об основных объектах *модели данных* – как логической, так и физической. С помощью специального редактора существующие отчеты можно изменить или создать собственный отчет. Каждый отчет может быть настроен индивидуально, данные в нем могут быть отсортированы и отфильтрованы. Browser Report позволяет сохранять результаты выполнения отчетов, печатать и экспортировать их в распространенные форматы.

#### Генерация словарей

Для управления большими проектами ERwin имеет специальный инструмент – ERwin Dictionary, который обеспечивает коллективную работу над диаграммами и позволяет сохранять и документировать различные версии *моделей данных*. ERwin Dictionary представляет собой специальную БД, которая позволяет решить проблемы документирования и хранения моделей, однако не полностью отвечает требованиям многопользовательской работы.