ПРОГРАММИРОВАНИЕ II

Модели данных и базы данных

План курса

- Требования к моделям данных и средствам их хранения и обработки.
- Обзор XML.
- Реляционная модель данных.
- SQL.
- СУБД/программный интерфейс.

Литература

1. Гарсиа-Молина Г., Ульман Дж.Д., Уидом Д. Системы баз данных. Полный курс. – М.: Издательский дом “Вильямс”. 2003.
2. Грабер М. Введение в SQL. – М.: Лори, 1996.
3. Дейт К. Введение в системы баз данных. – М.: Наука, 1980. Имеется 6-е и 7-е издание, – М.: Издательский дом “Вильямс”, 1999 и 2001.
4. Технологии XML. World Wide Web Consortium Home Page. http://www.w3.org.

Требования

- Долговременное (persistent) хранение больших объемов данных (логическая и физическая организация).
- Данные могут быть типизированы и структурированы. Необходимо иметь средства для формального описания и программного использования этой информации о данных (модель/схема данных).
- Эффективное манипулирование данными (добавление, модификация, выбор, удаление), сохраняющее корректность, в том числе, в конкурентной среде исполнения. Реализуется СУБД (DBMS).
- Программный интерфейс доступа к данным, в том числе, конкурентные средства доступа. Реализуется прикладными программными интерфейсами (API) для разных языков программирования.

Транзакции. Свойство ACID

Транзакция (transaction, TR) —группа последовательных операций (добавление, модификация, выбор, удаление), которая представляет собой логическую единицу работы с данными.

- Атомарность (**a**tomicity) – TR либо выполняется полностью, либо не выполняется совсем.
- Согласованность (**c**onsistency) – результат выполнения TR не нарушает ограничений, налагаемых моделью данных.
- Изолированность (**i**solation) – TR выполняется так, как будто другие TRs при этом не выполняются. В частности, другие исполняемые в этот же промежуток времени TRs не видят промежуточных результатов этой TR.
- Устойчивость (**d**urability) – результат выполненной TR не должен быть утрачен ни при каких обстоятельствах (включая физические причины, например, отключение электричества).

Пример: файловая система

- Умеет хранить терабайты двоичных данных.
- Информация о типе файла (смысле хранимых в нем данных) скудна. Связи между элементами данных не поддерживаются.
- В основном управлением занимается менеджер файловой системы ОС. Транзакционность по минимуму.
- API – текстовые/бинарные операторы ввода/вывода и/или библиотечные функции языков программирования. Параллельная работа возможна в ограниченном виде.

e**X**tensible **M**arkup **L**anguage

- Разработан World Wide Web Consortium (W3C). Версия 1.0 спецификации – 1998 год. Вторая редакция версии 1.1 спецификации – 2006 год. XML – упрощенная версия SGML.
- Стандарт для разработки языков разметки:
  - HTML/xhtml – язык разметки для гипертекста;
  - MathML – язык разметки математических формул;
  - CML – язык описания химических соединений;
  - TTML – язык описания данных для составления расписаний...
- Определяет стандартные механизмы обработки.
- Сопутствующие стандарты и технологии: DTD/XSchema (модель данных), XPath (адресация элементов документа), XSL/XSLT (преобразование xml-документов), XQuery (организация запросов к документам), DOM (объектное представление документа и доступ к нему)...

XML – основные понятия

- **Элементы.** Задают структуру документа. Документ обязан иметь единственный корневой элемент. Элемент может не иметь атрибутов и/или вложенных элементов.
- **Атрибуты.** Связаны с элементами и описывают их свойства. Могут иметь умолчательные значения.
- **Сущности.** Существуют предопределенные, могут определяться вне документа.
- **Символьные данные.** Произвольный текст.
- **Модель/схема данных.**
- **Инструкции обработки.**

Пример XML-данных

```
<?xml version="1.0" encoding="windows-1251" ?>
<University>
  <Groups>
    <Group ID="9871" />
    <Group ID="G123" Name="3134">
      <Student ID="S1256" FirstName="Анна" LastName="Иванова" Gender="true">
        Я родилась <Date>12 февраля</Date> в городе Кимры.
      </Student>
      <Student ID='S78' NickName='d&amp;apos;Artanjan'/>
      <Student ID="S502" FirstName="Василий" LastName="Кузькин" Gender="false">
        &amp;lt; &amp;gt; &amp;amp; &amp;quot; &amp;apos;
        &amp;#1105;
      </Student>
    </Group>
    <Group ID="G125" Name="3132">
      <!-- студенты -->
    </Group>
  </Groups>
  <Lectures>
    <Lecture Title="Программирование II" />
    <Lecture Title="Математическая логика" Lecturer="проф. Гончаров С.С.">
      <GroupItem GroupID="G123" />
      <GroupItem GroupID="G125" />
      <GroupItem GroupID="G12598723423" />
    </Lecture>
  </Lectures>
</University>
```

DTD – I

- DTD - Document Type Definition (определение типов документа).
- Используется проверяющим XML-процессором (validating processor) для проверки структуры документа, множества значений атрибутов и определения сущностей.
- Документ может как содержать DTD в самом себе, так и ссылаться на внешний файл:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE University SYSTEM "./University.dtd">
<University>
…
</University>
```

DTD – II

- Элемент:

```
<!ELEMENT body ANY>          - любое корректное содержимое
<!ELEMENT br   EMPTY>        - элемент не может иметь вложенных элементов
<!ELEMENT a    (#PCDATA)>    - элемент может содержать только текст
<!ELEMENT p    (#PCDATA|br|a)* > - любая последовательность (в том числе
                 и пустая) из данных элементов в любом порядке
<!ELEMENT elem (elem?,a+,p)+ > - последовательность элементов и
                                 групп элементов в заданном порядке
```

- Атрибут:

```
<!ATTLIST elem
          id      ID           #REQUIRED    - обязательный, уникальный
          name    CDATA        #IMPLIED     - необязательный, символьный
          visible (true|false) "false"      - перечислимый с умолчанием
          spouse  IDREF        #REQUIRED    - обязательная ссылка
          related IDREFS       #IMPLIED     - необязательный список ссылок
  >
```

- Сущности:

```
<!ENTITY % AND '-&amp;-' >
<!ENTITY RnR "Rock%AND;Roll" >
```

Пример DTD

```
<!ELEMENT University (Groups,Lectures)>   <!-- …University ANY… возможно, но плохо -->
<!ELEMENT Groups     (Group)* >
<!ELEMENT Group      (Student)+ >
<!ELEMENT Student    (#PCDATA,Date?,#PCDATA)* >
<!ELEMENT Date       (#PCDATA) >
<!ELEMENT Lectures   (Lecture)* >
<!ELEMENT Lecture    (GroupItem)* >
<!ELEMENT GroupItem  EMPTY >

<!ATTLIST Group
          ID         ID    #REQUIRED
          Name       CDATA #REQUIRED>

<!ATTLIST Student
          ID         ID    #REQUIRED
          FirstName  CDATA #IMPLIED
          LastName   CDATA #REQUIRED
          Gender     (true|false)    “true”>

<!ATTLIST Lecture
          Title      CDATA #REQUIRED
          Lecturer   CDATA #IMPLIED>

<!ATTLIST GroupItem
          GroupID      IDREF #REQUIRED>
```

XPath – основные понятия

- Средства описания подмножеств элементов/атрибутов/… XML-документа, удовлетворяющих заданным условиям.
- Рассматривая XML-документ как дерево, состоящее из узлов разного типа, XPath оперирует понятием пути и шага, который в свою очередь состоит из точки отсчета, теста узла и предиката, проверяющего свойства узла.

```
/University//parent::Group[position()=1 or @LastName='Ли']
```

- Имеются библиотечные функции, манипулирующие с числами, строками, ...

```
//Student[local-name()!='Student']
```

XPath: описание пути

- `.` – описание пути, совпадающее собственно с текущей вершиной (контекстная вершина).
- `..` – описание пути к отцу текущей вершины.
- `/` – описание пути до детей текущей вершины.
- `//` – путь до всех наследников текущей вершины.

Примеры:

- `/*` или `/University` – корневой элемент документа.
- `//*` – все элементы документа.
- `/University/Lectures//GroupItem`
- `./../*` – все братья текущего элемента и он сам.

XPath: описание точки отсчета

- `child` – ребенок контекстной вершины.
- `descendant` – все наследники контекстной вершины.
- `parent` – отец контекстной вершины (если она существует).
- `ancestor` – все предшественники контекстной вершины.
- `following-sibling` – все следующие братья контекстной вершины.
- `preceding-sibling` – все предшествующие братья контекстной вершины.
- `following` – все следующие братья и их наследники.
- `preceding` – все предшествующие братья и их наследники.
- `attribute` – атрибуты контекстной вершины.
- `self` – собственно сама контекстная вершина.
- `descendant-or-self` – ...
- `ancestor-or-self` – ...

ЗАМЕЧАНИЕ: ancestor, descendant, following, preceding и self – являются разбиением документа, т.е. множества, определяемые ими, не пересекаются, а их объединение дает весь документ.

Примеры описаний точек отсчета

- `. -» self::node()`
- `.. -» parent::node()`
- `// -» /descendant-or-self::node()/`
- `../title -» parent::node()/child::title`
- `.//author -» self::node()/descendant-or-self::node()/child::author`
- `chapter/section -» child::chapter/child::section`
- `//@ID -» /descendant-or-self::node()/attribute::ID`
- `following-sibling::*[attribute::Name='Петя'] -» following-sibling::node()[@Name='Петя']`

Примеры описаний предикатов

- `//section[paragraph]`
- `//section[not(@title)]`
- `./chapter[3]/section[position()=2]`
- `.//section[position()=1 or position()=last()]`
- `//*[local-name()='Student' and @FirstName!='Уи']`
- `./chapter[count(section/paragraph)!=0]`
- `//paragraph[id('P2345')]`
- `//author[sum(book/@price) > 1000]`
- `//Student[starts-with(@LastName, 'Я')]`

DOM – Document Object Model

Levels 1-3, последняя версия 2004

- Node
  - Document
  - Element
  - Attr
  - …
- NodeList
- NamedNodeMap
- DOMString
- DOMError
- DOMException
- …

Node интерфейс

```
interface Node {
  // NodeType
  const unsigned short   ELEMENT_NODE     = 1;
  const unsigned short   ATTRIBUTE_NODE   = 2;
  ...
  const unsigned short   COMMENT_NODE     = 8;
  const unsigned short   DOCUMENT_NODE    = 9;
  ...

  readonly attribute DOMString       nodeName;
           attribute DOMString       nodeValue;
  readonly attribute unsigned short  nodeType;
  readonly attribute Node            parentNode;
  readonly attribute NodeList        childNodes;
  readonly attribute Node            firstChild;
  readonly attribute Node            lastChild;
  readonly attribute Node            previousSibling;
  readonly attribute Node            nextSibling;
  readonly attribute NamedNodeMap    attributes;
  readonly attribute Document        ownerDocument;
  Node               insertBefore(in Node newChild, in Node refChild);
  Node               replaceChild(in Node newChild, in Node oldChild);
  Node               removeChild(in Node oldChild);
  Node               appendChild(in Node newChild);
  boolean            hasChildNodes();
  Node               cloneNode(in boolean deep);
  ...
};
```

NodeList и NamedNodeMap интерфейсы

```
interface NodeList {
  Node                  item(in unsigned long index);
  readonly attribute unsigned long   length;
};


interface NamedNodeMap {
  Node                  getNamedItem(in DOMString name);
  Node                  setNamedItem(in Node arg);
  Node                  removeNamedItem(in DOMString name);
  Node                  item(in unsigned long index);
  readonly attribute unsigned long   length;
  ...
};
```

Document интерфейс

```
interface Document : Node {
  ...
  readonly attribute Element documentElement;
  Element createElement(in DOMString tagName);
  Comment createComment(in DOMString data);
  Attr createAttribute(in DOMString name);
  EntityReference createEntityReference(in DOMString
  name);
  ...
  NodeList getElementsByTagName(in DOMString
  tagname);
  Element getElementById(in DOMString elementId);
  readonly attribute DOMString xmlEncoding;
  attribute boolean xmlStandalone;
  attribute DOMString xmlVersion;
  attribute DOMString documentURI;
  ...
};
```

Element и Attr интерфейс

```
interface Element : Node {
  readonly attribute DOMString   tagName;
  DOMString  getAttribute(in DOMString name);
  void       setAttribute(in DOMString name, in DOMString value);
  void       removeAttribute(in DOMString name);
  ...
  Attr       getAttributeNode(in DOMString name);
  Attr       setAttributeNode(in Attr newAttr);
  Attr       removeAttributeNode(in Attr oldAttr);
  ...
  void       setIdAttribute(in DOMString name, in boolean isId);
  void       setIdAttributeNode(in Attr idAttr, in boolean isId);
  ...
  boolean    hasAttribute(in DOMString name);

  NodeList   getElementsByTagName(in DOMString name);
  ...
};
```

```
interface Attr : Node {
  readonly attribute DOMString       name;
  readonly attribute boolean         specified;
           attribute DOMString       value;
  readonly attribute Element         ownerElement;
  readonly attribute TypeInfo        schemaTypeInfo;
  readonly attribute boolean         isId;
};
```

DOMError интерфейс

```
interface DOMError {

  // ErrorSeverity
  const unsigned short   SEVERITY_WARNING     = 1;
  const unsigned short   SEVERITY_ERROR       = 2;
  const unsigned short   SEVERITY_FATAL_ERROR = 3;

  readonly attribute unsigned short  severity;
  readonly attribute DOMString       message;
  readonly attribute DOMString       type;
  readonly attribute DOMObject       relatedException;
  readonly attribute DOMObject       relatedData;
  readonly attribute DOMLocator      location;
};
```

Пример: C++ (VS v6.0)

```
#include <iostream>
#import <msxml.dll> named_guids
using namespace std;
using namespace MSXML;

void main()
{
     ::CoInitialize(NULL);
     IXMLDOMDocumentPtr doc;
     doc.CreateInstance(CLSID_DOMDocument);
     doc->loadXML("<a><b><c>ddd</c></b><c>eee</c></a>");
     cout<<(char*)doc->Getxml();
     IXMLDOMElementPtr el=doc->selectSingleNode("//b");
     cout<<(char*)el->Getxml();
     IXMLDOMNodeListPtr lst=doc->documentElement->selectNodes("//*");
     cout<<lst->length<<endl;
     IXMLDOMNodePtr n=doc->documentElement->firstChild;
     while (n) {
          cout<<(char*)n->GetnodeName()<<endl;
          n=n->nextSibling;
     }
}
```

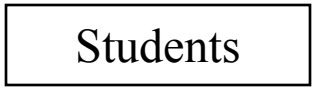
Entity**R**elationship-модель данных

Peter Chen, 1976

- ER-модель – семантическая модель данных, т.е. модель данных, главным предназначением которой удобное и адекватное моделирование смысла моделируемой предметной области (высокоуровневое моделирование).
- В общем случае она не описывает способов хранения данных и средств манипуляции ими.
- Разработаны формальные методы преобразования ER-моделей в другие модели данных.
- Преобразования поддерживаются программными средствами.

Элементы ER-модели

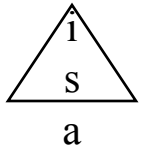
- Сущности – классы элементов моделируемой семантической области.



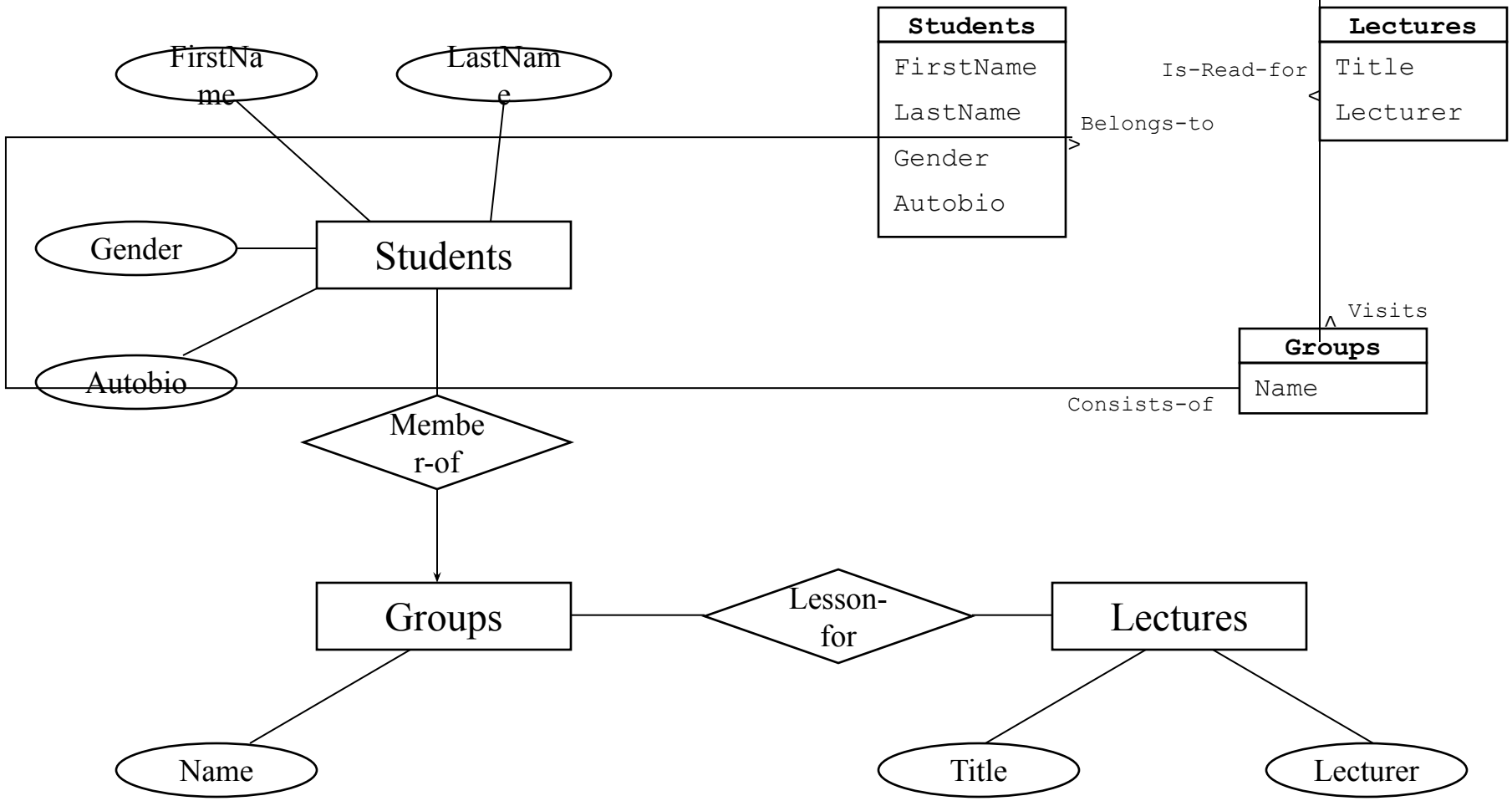
- Атрибуты – индивидуальные характеристики сущностей.

- Отношения – описания взаимодействий моделируемых сущностей.

  – Отношение общее-частное



Пример ER-модели

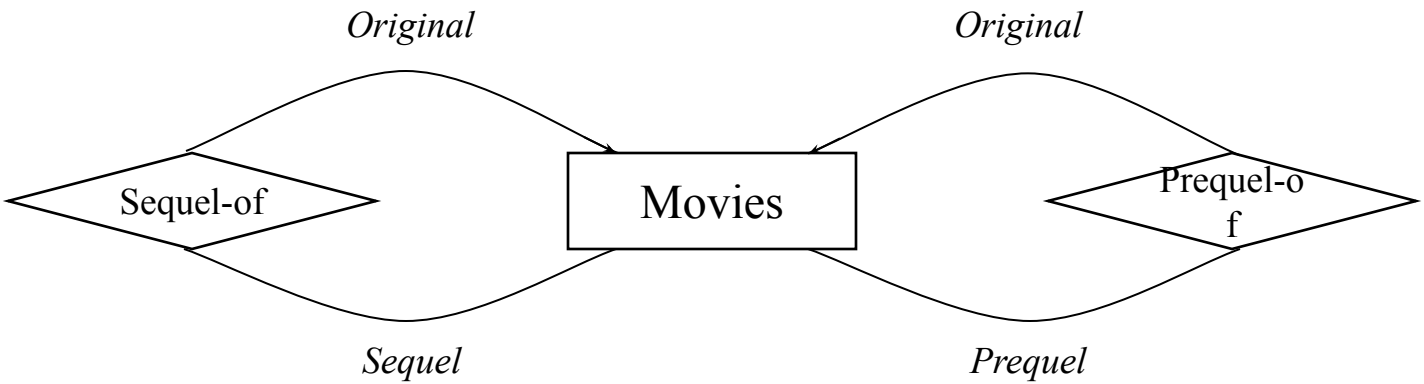


Типы связей в ER-моделях

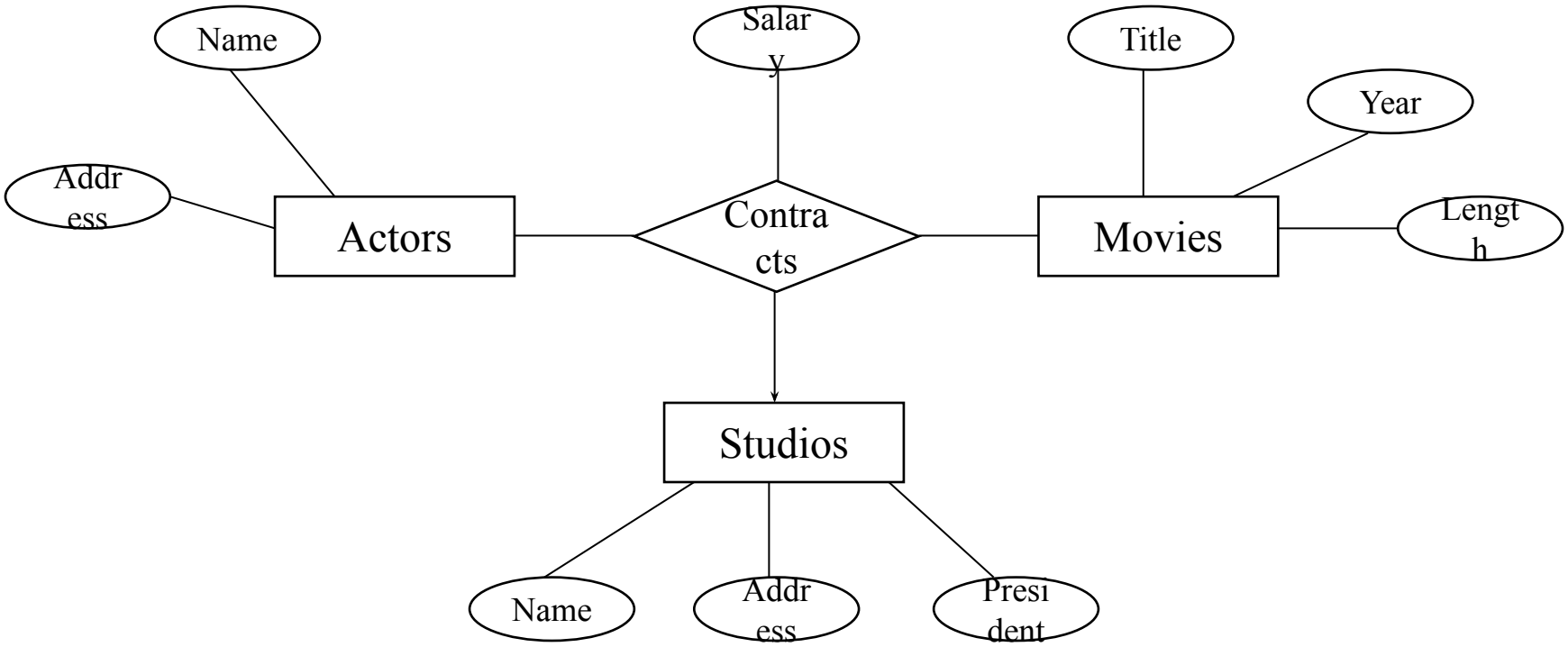
- Если каждый член множества A посредством связи R может быть связан не более чем с одним членом множества B, то R является связью типа **«многие к одному»** (many-one relationship). Эта же связь, рассматриваемая в обратном направлении, имеет тип «один ко многим».
- Если связь R в обоих направлениях (т.е. от A к B и от B к A) является связью «многие к одному», то это связь имеет тип **«один к одному»** (one-one relationship).
- Если связь R ни в одном из направлений не является связью «многие к одному», то эта связь имеет тип **«многие ко многим»** (many-many relationship).

Связи и роли

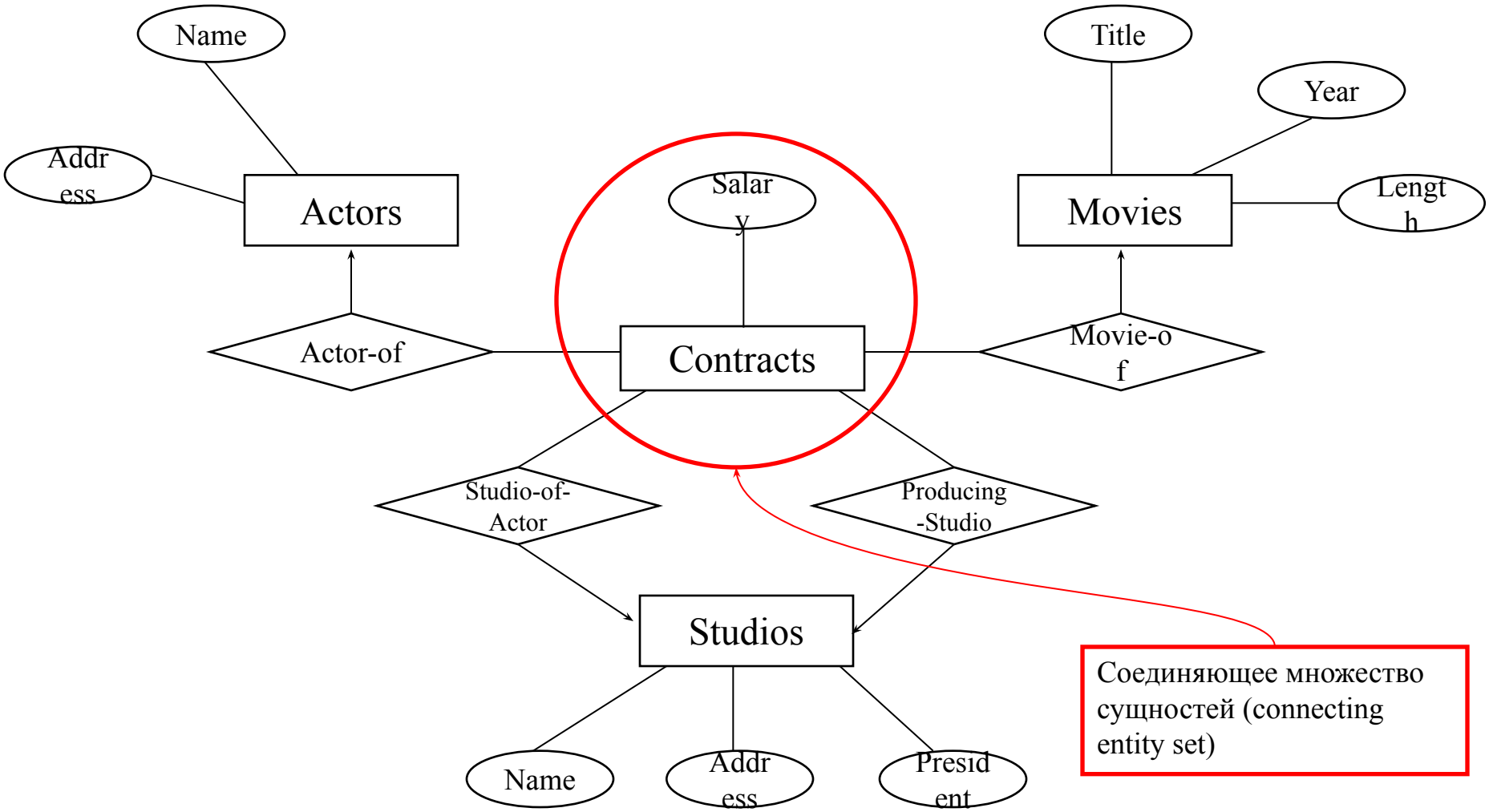
Если одна и та же сущность используется несколько раз в контексте одной и той же связи, то говорят, что разные концы связи описывают разные роли, в которых выступает данная сущность в этом случае.



Многосторонние связи и атрибуты связей

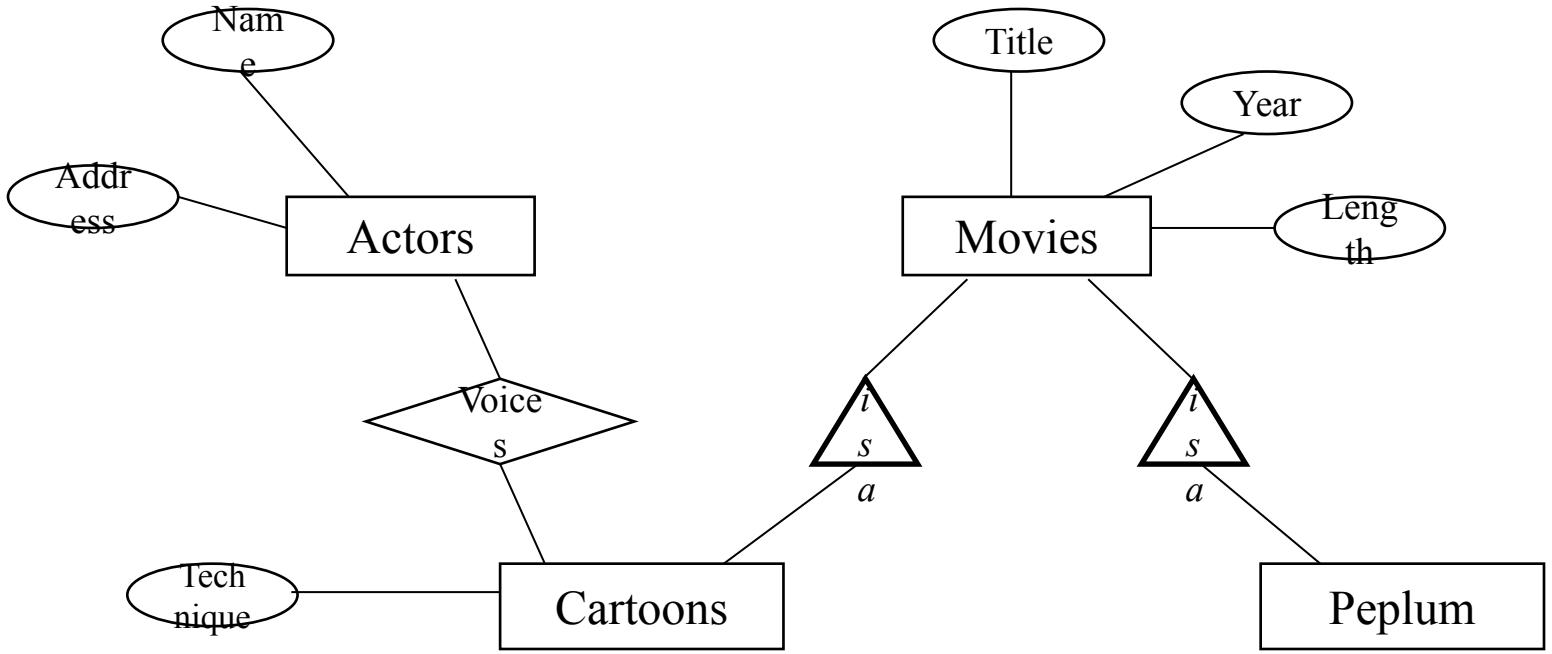


Преобразование многосторонних связей в бинарные



Подклассы в ER-модели

Подклассы (subclasses) служат для выделения в базовых классах (superclasses) сущностей, обладающих собственными атрибутами и/или связями. Используется для моделирования отношения «общее-частное»



Моделирование ограничений

- Ключ (key) – атрибут или подмножество атрибутов, уникальным образом определяющее экземпляр сущности среди множества других.
- Ограничение единственности (single-value constraint) – атрибут(ы)/связ(ь/и) в некотором контексте должны иметь единственное значение или не иметь его вообще.
- Ссылочная целостность (referential integrity constraint) – тот, на кого кто-то ссылается, должен обязательно существовать.
- Ограничение области значений (domain constraint) – значение атрибута принадлежит определенной области значений.
- Ограничение общего вида (general constraint).

Ограничение области значений

- Типы значений атрибутов (логические, диапазоны чисел, перечисления, длины строк) должны адекватно представлять моделируемую предметную область. Вводимые ограничения должны нести семантическую нагрузку.
- ER-модель не имеет специальных средств представления этих ограничений. Допускаются произвольные сопроводительные тексты.

Ограничение единственности

- Некоторый атрибут сущности может обладать не более чем единственным значением. Если допускается отсутствие значения атрибута, то появляется необходимость представлять этот факт каким-либо образом (выделенное «нулевое» значение). Наоборот, если некоторый атрибут обязан всегда иметь осмысленное значение (например, атрибут, входящий в ключ), то «нулевое» значение для него недопустимо.
- Связь R типа many-one между сущностями E_1 и E_2 демонстрирует ограничение уникальности, указывающее, что для любого экземпляра E_1, если эта связь существует, соответствует не более одного экземпляра E_2.

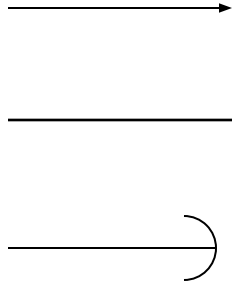
Ограничение ссылочной целостности

- Это ограничение (всюду определенное – total) требует, чтобы значение, выступающее в некоторой роли, имело в точности одно значение. Моделирует ситуацию отсутствия «висячих» (т.е. не определенных в данном контексте) ссылок.
- Если добавляется экземпляр А некоторой сущности, который обязан ссылаться на что-то (экземпляр В той же самой или другой сущности), то это что-то обязано уже существовать.
- Экземпляр В не может быть удален, пока не удалены все ссылающиеся на него экземпляры. Если удаление В обязательно, то должны быть удалены и все ссылающиеся на него экземпляры.

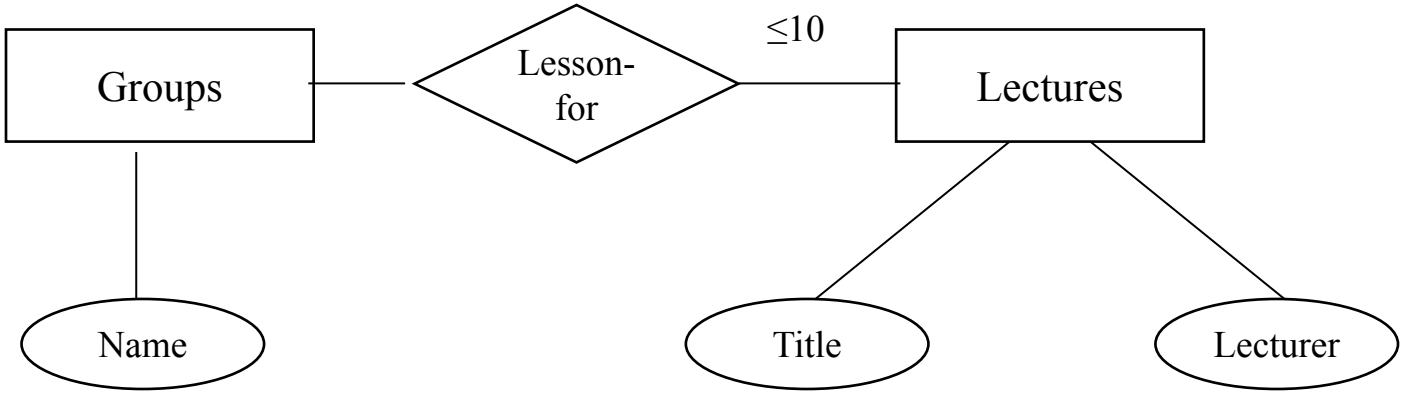
Ограничения общего вида

- Единственность
- Множественность
- Обязательность

представляется на ER-диаграмме



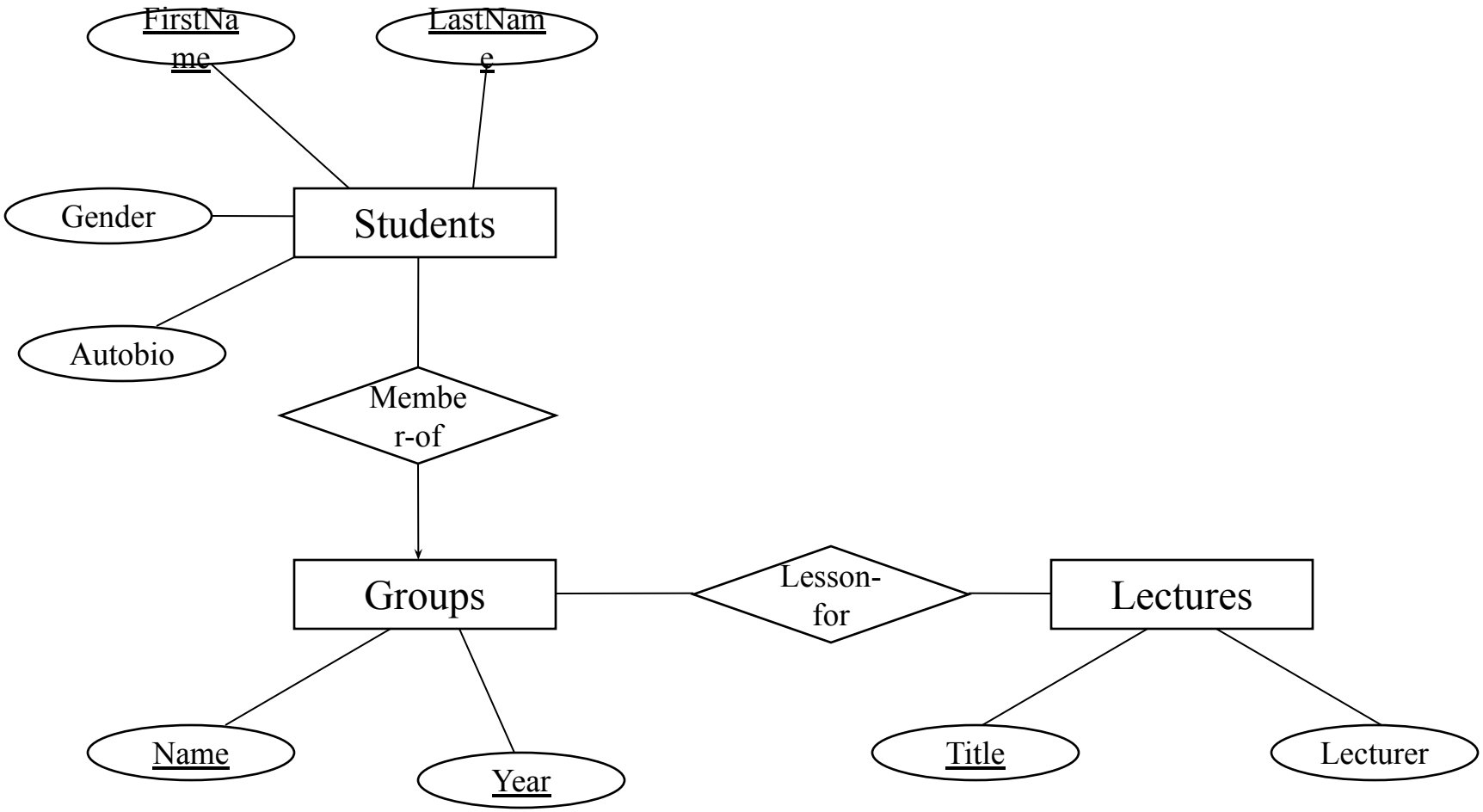
- Общего вида



Ключи в ER-моделях

- Каждая сущность должна обладать ключом. Сущность без ключа вызывает вопросы о правильности модели предметной области.
- Ключ может состоять из нескольких атрибутов.
- Сущность может обладать несколькими ключами. Тем не менее целесообразно выделять один – первичный ключ (primary key) и далее полагать, что эта сущность обладает единственным ключом.
- Если некоторая сущность участвует в иерархии связей isa, необходимо гарантировать, чтобы корневая сущность обладала всеми атрибутами, необходимыми для формирования ключа, и ключ для каждой сущности из иерархии может быть определен на основе «корневого» ключа.

Пример: ключи



Слабые сущности

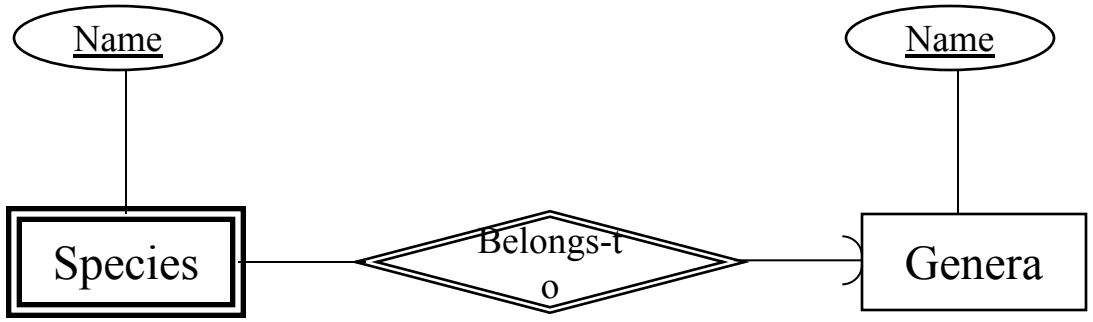
- В предметной области выделяется некоторое содержательное понятие – сущность с набором атрибутов. Однако оказывается, что на основе только этих атрибутов нельзя сформировать ключ для этой сущности. А с использованием атрибутов другой сущности (называемой владельцем) можно. Такая «невыразительная» сущность называется ***слабой.***
- Ограничения:
  - Между слабой сущностью и сущностью, используемой для ключа, должно быть отношение *many-one*.
  - Это отношение должно быть *обязательным* (total).
- Пример: связывающее множество сущностей обычно не имеет атрибутов. Их ключи определяются на основе сущностей, которые они связывают.

Выбор ключевых атрибутов для слабых сущностей

- Подмножества собственных атрибутов сущности E.
- Ключевые атрибуты сущностей, которые могут быть достигнуты посредством связей, соединяющих E с другими сущностями; такие связи называются поддерживающими (supporting relationships) для E. Они должны удовлетворять следующим условиям (ведут к сущности F):
  - это бинарная связь типа many-one, ведущая от E к F и реализующая ограничения ссылочной целостности;
  - атрибуты F, используемые для построения ключа E, должны быть ключом для F;
  - если F само по себе является слабой сущностью, то для него аналогично отыскиваются его поддерживающее множество связей.
- От E к F может вести несколько различных поддерживающих связей и каждая может поставлять свою копию ключевых атрибутов F. Таким образом некоторый экземпляр E может иметь ключ, порожденный разными экземплярами F.

Пример: слабые сущности

Биологический вид именуется парой – именем рода, которому принадлежит вид, и собственно именем вида. Пример: Homo erectus, Homo habilis, Homo sapiens – названия видов рода «человек» (сответственно человек прямоходящий, человек умелый, человек разумный). Имя вида может быть неуникальным.



Реляционная модель данных

- $D_1, D_2, \ldots, D_n$ – множества (атомарных) значений (domains).
- $R \subseteq D_1 \times D_2 \times \ldots \times D_n$ – отношение (relation), подмножество произведения доменов.
- Кортеж – отдельный элемент подмножества, определяемого отношением R.

FirstName	**LastName**	**Gender**	**Age**
Петр	Иванов	male	21
Петр	Сергеев	male	*null*
Анна	Павлова	female	18
Василий	Ломовой	male	123
Любовь	Яровая	female	18

Number	**Year**	**Speciaity**
7113	2007	Инф-тика
7114	2007	Механика
2114	2002	Механика
7114	1977	Механика

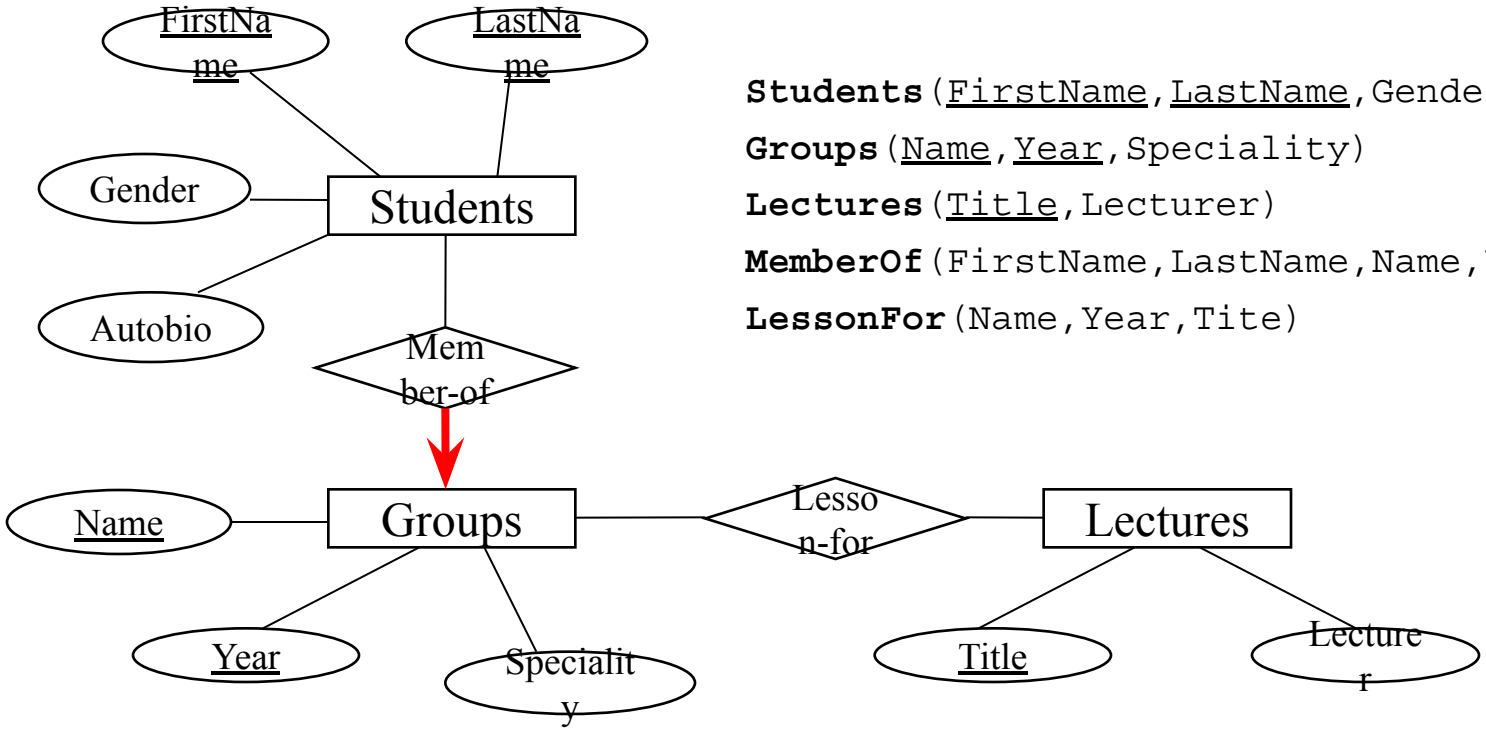
FirstName	**LastName**	**Number**	**Year**
Петр	Иванов	7113	2007
Петр	Сергеев	2114	2002
Анна	Павлова	7114	2007
Василий	Ломовой	7114	1977
Любовь	Яровая	7113	2007

Схема данных:

Students(FirstName,LastName,Gender,Age)
Groups(Number,Year,Speciality)
MemberOf(FirstName,LastName,Number,Year)

Преобразование ER-модели в реляционную. «Простая» часть

- Преобразовать каждую «простую» сущность (т.е. которые не являются слабыми и не участвуют в иерархии isa) в отношение (таблицу) с тем же набором атрибутов.
- Преобразовать каждую «простую» связь (не-isa) в отношение, атрибутами которого являются ключи сущностей, соединяемых этой связью. Добавить собственные атрибуты связи в это отношение



Students(FirstName,LastName,Gender,Autobio)

Groups(Name,Year,Speciality)

Lectures(Title,Lecturer)

MemberOf(FirstName,LastName,Name,Year)

LessonFor(Name,Year,Tite)

Преобразование ER-модели в реляционную. Объединение отношений

Имеется сущность E, соединенная связью R типа many-one с сущностью F в направлении от E к F. Можно выполнить объединение отношений, соответствующих E и R. Новое отношение получается объединением следующих атрибутов:

1. все атрибуты E;
2. ключевые атрибуты F;
3. собственные атрибуты связи R.

Если некоторый экземпляр E не имеет связи с экземплярами R, то атрибуты из пунктов 2 и 3 принимают значение `NULL`. Основное соображение в пользу преобразования – экономия памяти и эффективность манипулирования.

Пример: объединение отношений

FirstName	LastName	Gender	Age	Number	Year
Петр	Иванов	male	21	7113	2007
Петр	Сергеев	male	*null*	2114	2002
Анна	Павлова	female	18	7114	2007
Жугдермидин	Гуррагча	male	*null*	*null*	*null*
Василий	Ломовой	male	123	7114	1977
Любовь	Яровая	female	18	7113	2007
Анжела	Дэвис	female	*null*	*null*	*null*

Number	Year	Speciaity
7113	2007	Инф-тика
7114	2007	Механика
2114	2002	Механика
7114	1977	Механика

Students(FirstName,LastName,Gender,
Autobio,Name,Year)

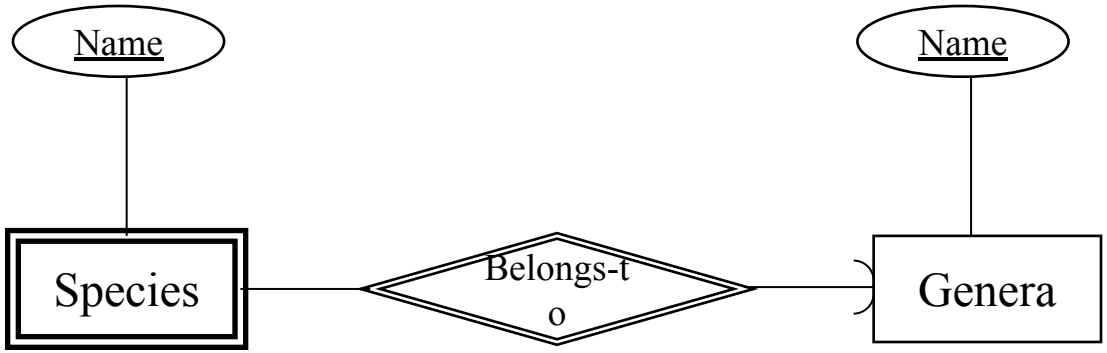
Groups(Name,Year,Speciality)

~~**MemberOf**(FirstName,LastName,Name,Year)~~

Преобразование ER-модели в реляционную. Слабые сущности

- Если W – слабая сущность, отношение для W строится следующим образом:
  - включаются все атрибуты W;
  - включаются все атрибуты поддерживающих связей для W;
  - включаются все ключевые атрибуты каждого множества сущностей, соединенных с W поддерживающими связями.
- Любые поддерживающие связи для W игнорируются.

Пример: преобразование слабых сущностей

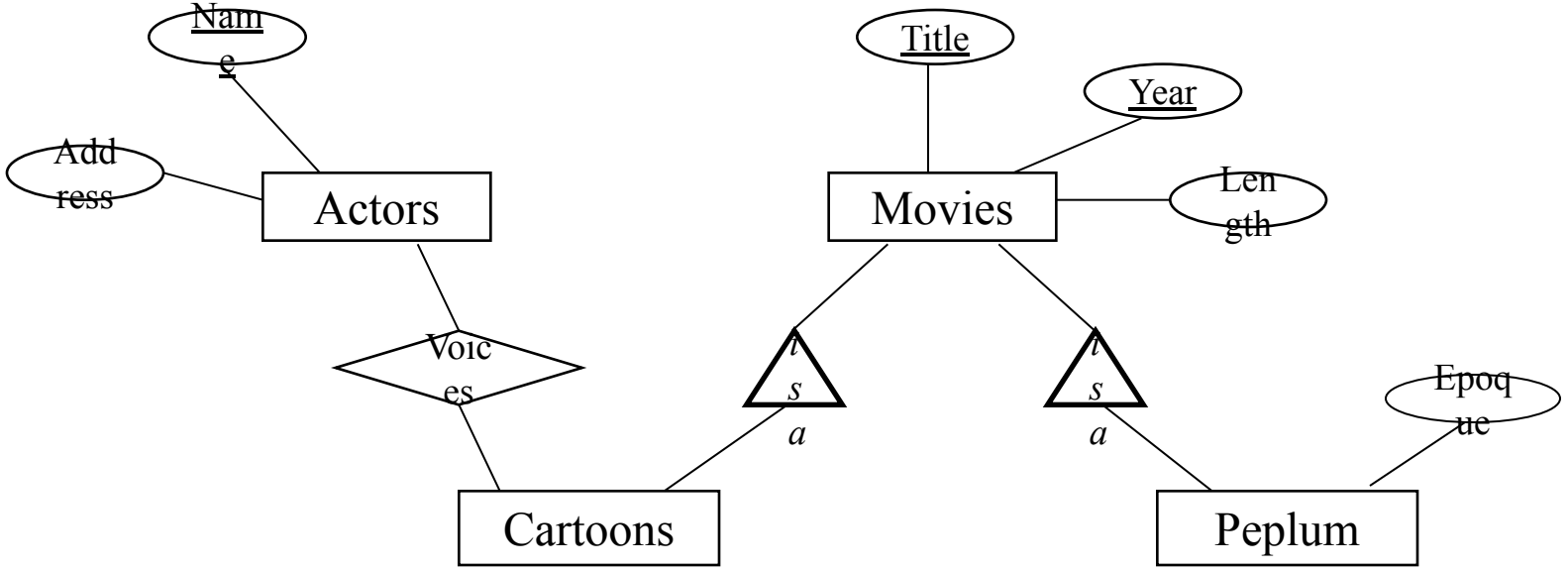


Genera(Name)

Species(Name, NameOfGenera)

Преобразование ER-модели в реляционную ISA в стиле «сущность-связь»

Для каждой сущности в иерархии создается отношение. Если сущность не является корневой, соответствующее ей отношение, помимо собственных атрибутов, должно содержать ключевые атрибуты корневого множества (они также участвуют в связывании этой сущности с другими)



Movies(Title,Year,Length)
Cartoons(Title,Year)
Peplum(Title,Year,Epoque)

Actors(Name,Address)

Voices(Title,Year,ActorName)

Преобразование ER-модели в реляционную ISA в объектно-ориентированном стиле

Метод состоит в перечислении всевозможных поддеревьев ER-диаграммы, включающих корневую сущность, на основе которых создаются отношения, представляющих сущности. Они обладают всеми атрибутами поддерева.

Movies(Title,Year,Length)
MoviesCartoons(Title,Year,Length)
MoviesPeplum(Title,Year,Length,Epoque)
MoviesCartoonsPeplum(Title,Year,Length,Epoque)

Actors(Name,Address)
Voices(Title,Year,ActorName)

Преобразование ER-модели в реляционную
ISA в NULL-стиле

Все сущности иерархии объединяются в одно отношение. При этом если для некоторого кортежа отношения (экземпляра некоторой сущности) значение какого-то атрибута не определено, оно представляется `NULL`.

Movies(Title,Year,Length,Epoque)
Actors(Name,Address)
Voices(Title,Year,ActorName)

Title	Year	Length	Epoque
Star Wars	1977	240	*null*
Titanic	1997	200	*null*
Lion King	1995	60	*null*
Cinderella	1950	74	*null*
Spartacus	1960	120	Rome
Gone with the Wind	1939	120	Civil War

Преобразования отношений в реляционной модели

Функциональная зависимость между атрибутами отношения: если два кортежа отношения совпадают в атрибутах $A_1,A_2,\ldots,A_n$, то они должны совпадать и в атрибутах $B_1,B_2,\ldots,B_m$ (функционально их обуславливают $A_1,A_2,\ldots,A_n \rightarrow B_1,B_2,\ldots,B_m$).

Movies(Title,Year,Length,StudioName,Actor)

Title Year ⟶ Length

Title Year ⟶ StudioName

Title Year $\overset{?}{\longrightarrow}$ Actor

- Множество функциональных зависимостей S следует из множества ФЗ T, если каждый экземпляр отношения, удовлетворяющий всем ФЗ T, также удовлетворяет всем ФЗ S.
- Множества функциональных зависимостей S и T являются эквивалентными, если они следуют одно из другого и наоборот.

Замыкание множества атрибутов

Замыканием (closure) $\{A_i\}^+$ множества атрибутов $\{A_i\}$ обусловленным множеством функциональных зависимостей S называется множество атрибутов $\{B_i\}$, такое что ФЗ $A \rightarrow B$ следует из ФЗ S.

Алгоритм построения замыкания:

1. Инициализировать переменную X множеством $\{A_i\}$.
2. Если существует ФЗ $B_1,\ldots,B_n \rightarrow C$ из S, такая, что $\{B_1,\ldots,B_n\} \subseteq X$, но $C \notin X$, то добавить C в X.
3. Выполнять шаг 2, пока множество X не стабилизируется.

Пример:

Отношение имеет множество атрибутов A, B, C, D, E, F и удовлетворяет ФЗ $A,B \rightarrow C$, $B,C \rightarrow A,D$, $D \rightarrow E$, $C,F \rightarrow B$.

Замыкание $\{A,B\}^+ = \{A,B,C,D,E\}$

Свойство:

ФЗ $A_1,A_2,\ldots,A_n \rightarrow B_1,B_2,\ldots,B_m$ следует из ФЗ S, тогда и только тогда $B_1,B_2,\ldots,B_m \notin \{A_1,A_2,\ldots,A_n\}$ обусловленным ФЗ S.

Ключи и суперключи отношений

- Множество атрибутов $\{A_i\}$ называется ключом отношения R, если:
  - эти атрибуты функционально обуславливают все остальные атрибуты; совпадение двух кортежей отношения R в этих атрибутах невозможно;
  - ни одно из подмножеств $\{A_i\}$ не обуславливает функционально все остальные атрибуты отношения R.
- Суперключ отношения – всякое множество атрибутов, содержащее в качестве подмножества ключ отношения.
- Чтобы определить, формирует ли множество атрибутов $\{A_i\}$ ключ отношения, надо проверить совпадает ли $\{A_i\}^+$ относительно известного Вам множества ФЗ со всем множеством атрибутов отношения, а любое подмножество $\{A_i\}^+$ – нет.

Аномалии отношений

- Избыточность (redundancy) данных.
- Аномалии изменения (update anomalies).
- Аномалии удаления (delete anomalies).

Title	Year	Length	FilmType	StudioName	ActorName	ActorPhoto
Star Wars	1977	124	color	20thC Fox	C. Fisher	BMP-data1
Star Wars	1977	124	color	20thC Fox	M. Hamill	BMP-data2
Star Wars	1977	124	color	20thC Fox	H. Ford	BMP-data3
Star Wars	2003	134	color	20thC Fox	C. Fisher	BMP-data1
Indiana Jones	1989	110	color	Lucasfilm	H. Ford	BMP-data3
Indiana Jones	1989	110	color	Lucasfilm	S. Connery	BMP-data4
From Russia with Love	1964	115	color	Eon Productions	S. Connery	BMP-data4
King Kong	1933	100	bw	Culver Studios	null	null
King Kong	1976	134	color	Dino De Laurentis	J. Lange	BMP-data5
King Kong	2005	187	color	Big Primate Pictures	A. Brody	BMP-data6

Декомпозиция отношений

- Отношения $S(s_1,\ldots,s_n)$ и $T(t_1,\ldots,t_m)$ являются декомпозицией отношения $R(r_1,\ldots,r_k)$, если:
- $\{r_1,\ldots,r_k\}=\{s_1,\ldots,s_n\}\cup\{t_1,\ldots,t_m\}$.
- кортежи отношений S и T являются проекциями всех кортежей отношения R на их множества атрибутов (на $\{s_1,\ldots,s_n\}$ и $\{t_1,\ldots,t_m\}$ соответственно).

Title	Year	Length	FilmType	StudioName
Star Wars	1977	124	color	20thC Fox
Star Wars	2003	134	color	20thC Fox
Indiana Jones	1989	110	color	Lucasfilm
From Russia with Love	1964	115	color	Eon Productions
King Kong	1933	100	bw	Culver Studios
King Kong	1976	134	color	Dino De Laurentis
King Kong	2005	187	color	Big Primate Pics

Title	Year	ActorName	ActorPhoto
Star Wars	1977	C. Fisher	BMP-data1
Star Wars	1977	M. Hamill	BMP-data2
Star Wars	1977	H. Ford	BMP-data3
Star Wars	2003	C. Fisher	BMP-data1
Indiana Jones	1989	H. Ford	BMP-data3
Indiana Jones	1989	S. Connery	BMP-data4
From Russia with Love	1964	S. Connery	BMP-data4
~~King Kong~~	~~1936~~	~~null~~	~~null~~
King Kong	1976	J. Lange	BMP-data5
King Kong	2005	A. Brody	BMP-data6

Нормальная форма Бойса-Кодда

- Отношение R удовлетворяет BCNF тогда и только тогда, когда для R существует нетривиальная ФЗ $A_1,A_2, \ldots,A_n \rightarrow B_1,B_2, \ldots,B_m$ (т.е. $\exists\ B_k \notin \{A_1,A_2, \ldots,A_n\}$) такая, что множество атрибутов $\{A_1,A_2, \ldots,A_n\}$ является суперключом для R.
- Отношение, удовлетворяющее BCNF, не содержит аномалий.
- Отношение, содержащее только два атрибута, удовлетворяет BCNF.

Пример:

{Title,Year,ActorName} – ключ исходного отношения. Однако это отношение не удовлетворяет BCNF, т.к. содержит ФЗ Title,Year→Length,FilmType,StudioName, у которой левая часть не является суперключом. Первое отношение декомпозиции, имеющее в качестве ключа {Title,Year}, удовлетворяет BCNF.

Пример: нормализованные отношения

NormMovies

Title	Year	Length	FilmType	StudioName
Star Wars	1977	124	color	20thC Fox
Star Wars	2003	134	color	20thC Fox
Indiana Jones	1989	110	color	Lucasfilm
From Russia with Love	1964	115	color	Eon Productions
King Kong	1933	100	bw	Culver Studios
King Kong	1976	134	color	Dino De Laurentis
King Kong	2005	187	color	Big Primate Pics

StarsIn

Title	Year	Name
Star Wars	1977	C. Fisher
Star Wars	1977	M. Hamill
Star Wars	1977	H. Ford
Star Wars	2003	C. Fisher
Indiana Jones	1989	H. Ford
Indiana Jones	1989	S. Connery
From Russia with Love	1964	S. Connery
King Kong	1976	J. Lange
King Kong	2005	A. Brody

Actors

Name	Photo
C. Fisher	BMP-data1
M. Hamill	BMP-data2
H. Ford	BMP-data3
S. Connery	BMP-data4
J. Lange	BMP-data5
A. Brody	BMP-data6

Третья нормальная форма

- BCNF иногда является слишком строгим условием и требует декомпозиции даже в тех случаях, когда наличие некоторых «ненормализованных» ФЗ целесообразно.
- Отношение R удовлетворяет 3NF, если всякий раз, когда для R существует нетривиальная ФЗ $A_1, A_2, \ldots, A_n \rightarrow B$, множество атрибутов $\{A_1, A_2, \ldots, A_n\}$ является суперключом для R либо B является членом некоторого ключа.
- Если отношение R удовлетворяет 3NF, но не удовлетворяет BCNF, существует некоторая вероятность, что отношение будет содержать избыточные данные.

Реляционная алгебра

- Переменные, соответствующие неограниченным отношениям, и константы, являющиеся конечными отношениями.
- Операции:
- теоретико-множественные операции: объединение (union), пересечение (intersection) и разность (difference);
- операции удаления частей отношения: выбор (selection) и проекции (projection);
- операции сочетания кортежей отношений: декартово произведение (Cartesian product) и различные виды соединений (joins);
- операции переименования: атрибутов и отношений.
- Агрегирование, группировка и сортировка данных

Пример выбора и проекции

$$\pi_{\text{ActorName}}(\sigma_{\text{Year}>1976}(\text{Movies}))$$

ActorName
C. Fisher
M. Hamill
H. Ford
S. Connery
A. Brody

$$\sigma_{\text{Year}>1976}(\pi_{\text{Title,Year}}(\text{Movies}))$$

Title	Year
Star Wars	1977
Star Wars	2003
Indiana Jones	1989
King Kong	2005

Пример декартова произведения и естественного соединения

S

A	B
1	2
3	4

T

B	C	D
2	5	8
4	7	8
9	10	11

S×T – декартово произведение

A	S.B	T.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

S⋈T – естественное соединение (natural join)

A	B	C	D
1	2	5	6
3	4	7	8

$$\mathbf{S} \bowtie \mathbf{T} = \pi_{S.A,S.B,T.C,T.D}(\sigma_{S.B=T.B}(\mathbf{S} \times \mathbf{T}))$$

Другие виды соединений

- Тэта-соединение (Θ-join): $\mathbf{S} \bowtie_{\Theta} \mathbf{T} = \sigma_{\Theta}(\mathbf{S} \times \mathbf{T})$.
- Внешние соединения (outerjoins, left/right):

S

A	B	C
1	2	3
4	5	6
7	8	9

T

B	C	D
2	3	10
2	3	11
6	7	12

S ⋈ T

A	B	C	D
1	2	3	10
1	2	3	11

S ⟗ T

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	null
7	8	9	null
null	6	7	12

S ⟕ T

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	null
7	8	9	null

S ⟖ T

A	B	C	D
1	2	3	10
1	2	3	11
null	6	7	12

- Вненшние тэта-соединения.

SQL – Structured Query Language
первая версия – конец 1970-х (IBM)
текущий стандарт – 1999 (ANSI)

- Создание/удаление таблиц (`create/drop table`)
- Изменение таблиц (`alter table`)
- Добавление/удаление данных (`insert into/ delete from`)
- Изменение данных (`update`)
- Выборка данных (`select-from-where`)
- Специальные возможности.

Создание/удаление/изменение таблиц

Типы данных SQL:

- INT/INTEGER, SHORTINT
- CHAR(n), VARCHAR(n)
- BOOLEAN
- FLOAT/REAL, DOUBLE PECISION, DECIMAL(n,d)/NUMERIC
- DATE, TIME, TIMESTAMP
- BLOB/MEMO

```
CREATE TABLE Movies (
  Title       CHAR(200),
  Year        INTEGER,
  Length      INTEGER,
  FilmType    CHAR(5) DEFAULT 'color',
  StudioName  VARCHAR(100) DEFAULT 'unknown',
  ActorName   VARCHAR(1000),
  Birthdate   DATE DEFAULT DATE '0000-00-00',
  Gender      BOOLEAN,
  PRIMARY KEY (Title, Year)
);
```

```
DROP TABLE Movies;
```

```
ALTER TABLE Movies
ADD ActorEmail VARCHAR(512)
DEFAULT 'admin@nsu.ru';

ALTER TABLE Movies
DROP Gender;
```

Добавление/удаление/обновление данных

```
INSERT INTO Movies(Title,Year,ActorName)
VALUES('Titanic',1997,'L. Di Caprio');

INSERT INTO Movies
VALUES('Titanic',1997,240,'color','Universal',
       'K. Winslet','1972-03-08',true);


DELETE FROM Movies
WHERE Year<1970 AND StudioName LIKE '%war%';

DELETE FROM Movies;


UPDATE Movies
SET    ActorName='Mister '||ActorName
WHERE  NOT Gender AND Birthdate<'1990-01-01';
```

Выборка данных I

- Дальнейшая обработка пользователем.
- Использование в проверке условий.
- Использование при добавлении данных.
- Создание временных/виртуальных таблиц (например, денормализация отношений).

```
SELECT *
FROM   Movies;
```

```
SELECT Title, ActorName
FROM   Movies;
WHERE  Year>1970;
```

$$\pi_{\text{Title,ActorName}}(\sigma_{\text{Year>1970}}(\text{Movies}))$$

```
INSERT INTO Actors(Name)
  SELECT DISTINCT ActorName
  FROM Movies
  WHERE ActorName NOT IN
    (SELECT Name FROM Actors);
```

Выборка данных II

Movies(Title,Year,Length,FilmType,Studio)
Actors(Name,Address,Gender,Birthdate,Photo)
Studios(Name,Address)
StarsIn(Title,Year,Name)

```
SELECT DISTINCT
       Movies.Title, Std.Name AS StudioName, A.Name AS ActorName

FROM   Movies, Studios Std, Actors A, StarsIn S, StarsIn SS

WHERE  Movies.Title=S.Title AND Movies.Year=S.Year
                            AND  A.Name=S.Name
  AND  Movies.Studio=Std.Name
  AND  NOT (S.Name=SS.Name AND (S.Title<>SS.Title OR
                                S.Year<>SS.Year))
  AND  Std.Address NOT LIKE '%USA'

ORDER BY Movies.Title, Std.Name ASC, A.Name DESC;
```

Выборка данных III

```
SELECT Address
FROM   Studios, Movies
WHERE  Name=Studio
  AND  Title='Indiana Jones';
```

```
SELECT Address
FROM   Studios
WHERE  Name=
       (SELECT Studio
        FROM   Movies
        WHERE  Title=
               'Indiana Jones');
```

- `EXISTS (SELECT …) / NOT EXISTS (SELECT …)`
- `IN (SELECT …) / NOT IN (SELECT …)`
- С операциями сравнения `=, <>, <, >, <=, >=` применяется:
  - `X > ALL (SELECT …)`
  - `X > ANY (SELECT …)`

Выборка данных IV

```
SELECT Address
FROM   Studios, (SELECT Studio
                 FROM   Movies
                 WHERE  Title='Indiana Jones') Indy
WHERE  Name=Indy.Studio;
```

```
SELECT NormMovies.*,StarsInMovies.Name,StarsInMovies.Photo,
FROM   NormMovies, (SELECT S.Title, S.Year, A.Name, A.Photo
                    FROM   StarsIn S, Actors A
                    WHERE  S.Name=A.Name) StarsInMovies
WHERE  NormMovies.Title=StarsInMovies.Title
  AND  NormMovies.Year =StarsInMovies.Year;
```

Значения NULL и UNKNOWN

- NULL используется, когда:
  - Значение не является известным на данный момент,
  - Значение не является применимым в данном контексте,
  - Значение закрыто для общего доступа.
- NULL не является константой и не может быть явно использован в выражениях.
- Для проверки, равно ли ли значение NULL, используется специальный предикаты SQL: `IS NULL` и `IS NOT NULL` (например, `ActorPhoto IS NOT NULL`).
- Если NULL встречается в арифметических выражениях, то результатом будет NULL. В логических – UNKNOWN.
- Логика SQL трехзначная: `FALSE (0), TRUE (1), UNKNOWN (½)`. Интерпретация логических связок: AND – минимум, OR – максимум, NOT – дополнение до единицы.

Операторы агрегирования

- SUM, AVG, MIN, MAX – операторы над числовыми атрибутами (сумма, среднее, минимум, максимум).
- COUNT – количество записей, которые получены из таблиц, указанных в предложении FROM, и удовлетворяют условию, указанному в предложении WHERE. COUNT(DISTINCT Attr) подсчитывает записи, различающиеся в атрибуте Attr.
- Предложение GROUP BY <список атрибутов> позволяет получить группы, в которых значения атрибутов из <список атрибутов> одинаковы, и применить операторы агрегирования в рамках групп. Условие на записи в группе указывается с помощью предложения HAVING.

```
SELECT   Studio, SUM(Length)
FROM     Movies
WHERE    FilmType='bw'
GROUP BY Studio;
HAVING   Max(Year)<1960;
```

Соединения в SQL

- `T1 CROSS JOIN T2` – декартово произведение таблиц.
- `T1 NATURAL JOIN T2` – естественное соединение таблиц.
- `T1 JOIN T2 ON <условие на атрибутах T1 и T2>` - тета-соединение таблиц.
- `T1 NATURAL FULL OUTER JOIN T2,`
  `T1 NATURAL LEFT OUTER JOIN T2,`
  `T1 NATURAL RIGHT OUTER JOIN T2` – внешние соединения.
- `T1 FULL  OUTER JOIN T2 ON <условие>,`
  `T1 LEFT  OUTER JOIN T2 ON <условие>,`
  `T1 RIGHT OUTER JOIN T2 ON <условие>` – внешние тета-соединения.
- Первые три вида соединений можно выразить с помощью `SELECT-FROM-WHERE`.

Описание схемных ограничений – ключи

- Объявление ключей с помощью `UNIQUE`
  - может быть несколько;
  - атрибуты, входящие в такой ключ, могут иметь значение `NULL`.
- Объявление внешних ключей:

```
CREATE TABLE NormMovies (
  Title       CHAR(200),
  Year        INTEGER,
  ...
  PRIMARY KEY (Title, Year)
);
```

```
CREATE TABLE Actors (
  Name  CHAR(1000) PRIMARY KEY,
  ...
);
```

```
CREATE TABLE StarsIn (
 Title  CHAR(200),
 Year   INTEGER,
 Name   CHAR(1000) REFERENCES Actors(Name),
 FOREIGN KEY (Title,Year) REFERENCES NormMoveis(Title,Year),
 PRIMARY KEY (Title,Year,Name)
);
```

- Стратегии добавления записей, содержащих внешние ключи.

Пример: C++ (VS v6.0)

```
CDatabase  db;      CRecordset rs(&db);
db.Open(NULL);
db.ExecuteSQL("DROP TABLE myTable");
db.ExecuteSQL("CREATE TABLE myTable (myID INT PRIMARY KEY, myNum INT, myName VARCHAR(20))");
db.ExecuteSQL("INSERT INTO Tbl1 (Name) VALUES ('AAA')");
db.ExecuteSQL("UPDATE Tbl1 SET Tbl1.Number=Tbl1.Number+1 WHERE Tbl1.ID>Tbl1.Number");
db.ExecuteSQL("DELETE FROM Tbl1 WHERE Tbl1.Number=Tbl1.ID");
if (rs.Open(CRecordset::forwardOnly,
            "SELECT t1.*, (t1.Number+t2.Number) as SUM FROM Tbl1 t1, Tbl1 t2 WHERE t1.Number+t2.Number>t1.ID",
            CRecordset::none)) {
     cout<<"SQL query: "<<endl<<(LPCTSTR)rs.GetSQL()<<endl;
     cout.width(10*rs.GetODBCFieldCount()+1); cout.fill('='); cout<<' '<<endl; cout.fill(' ');
     for(int i=0; i<rs.GetODBCFieldCount(); ++i) {
          CODBCFieldInfo info;
          rs.GetODBCFieldInfo(i,info);
          cout.width(10); cout<<(LPCTSTR)info.m_strName;
     }
     cout<<endl;
     cout.width(10*rs.GetODBCFieldCount()+1); cout.fill('-'); cout<<' '<<endl; cout.fill(' ');
     while (!rs.IsEOF()) {
          for(k=0; k<rs.GetODBCFieldCount(); ++k) {
               CString strVal;
               rs.GetFieldValue(k,strVal);
               cout.width(10); cout<<(LPCTSTR)strVal;
          }
          cout<<endl;
          rs.MoveNext();
     }
     cout.width(10*rs.GetODBCFieldCount()+1); cout.fill('='); cout<<' '<<endl; cout.fill(' ');
     cout<<"Total records: "<<rs.GetRecordCount()<<endl;
}
rs.Close();  db.Close();
```