

Организация баз данных

Fundamentals of databases

Копп Андрей Михайлович

Andrii M. Kopp

Ассистент кафедры ПИИТУ

Assistant Lecturer of the Department of SEMIT

kopp93@gmail.com @andriikopp

Программа курса / Course program

1. Введение / Intro
2. Хранение данных и файловая структура / Storage and file structure
3. Модель сущность-связь / Entity-relationship model
4. Нормализация базы данных / Database normalization
5. Реляционная модель / Relational model
6. Язык запросов SQL / SQL query language

- Введение / Intro
 - Общие понятия / Overview
 - Архитектура / Architecture
 - Модели данных / Data models
 - Схемы данных / Data schemas
 - Независимость данных / Data independence
- Хранение данных и файловая структура / Storage and file structure
- Модель сущность-связь / Entity-relationship model
 - Основные понятия / Basic concepts
 - Представление диаграмм сущность-связь / ER diagram representation
 - Обобщение и агрегация / Generalization and aggregation
- Нормализация базы данных / Database normalization
- Реляционная модель / Relational model
 - Правила Кодда / Codd's rules
 - Реляционная модель данных / Relational data model
 - Реляционная алгебра / Relational algebra
 - Преобразование модели сущность-связь в реляционную модель / ER to relational model
- Язык запросов SQL / SQL query language

Лекция 01: Введение

Lecture 01: Intro

Данные / Data

- Данные – это совокупность / набор / коллекция фактов и представлений / понятий, которые могут быть обработаны для получения информации.
- Data is a collection of facts and figures that can be processed to produce information.

Student

- Student ID / Credit book number
- Full name
- Group number
- ...
- Birth date
- Contacts
- etc.

How to store data?

```
int id = 1;
```

```
string name = "John Doe";
```

```
int grade = 2;
```

```
int id = 1;  
string name = "John Doe";  
int grade = 2;
```

=>

```
int[] ids = { 1, 2, ..., N }  
string[] names = { "John Doe", "Petro Petrenko",  
    ... }  
int[] grades = { 2, 1, ... }
```



```
int[] ids = { 1, 2, ..., N };  
string[] names = { "John Doe", "Petro Petrenko", ... };  
int[] grades = { 2, 1, ... };
```

=>

```
struct Student {  
    int id;  
    string name;  
    int grade;  
};
```

```
Student[] students = { {1, "Jonh Doe", 2}, ... };
```

```
ofstream out("students.txt");
for (int i = 0; i < N; i++) {
    out << students[i].id << "," << students[i].name << "," <<
    students[i].grade << endl;
}
out.close();
```

=>

students.txt

1,JohnDoe,2

2,Petro Petrenko,1

...

N,Firstname Lastname,X

Student

id

name

grade

+

How to store **RELATED DATA**?

Courses

id

title

hours

credits

База данных / Database

- База данных – это совокупность / набор / коллекция связанных данных.
- Database is a collection of related data.

Data (facts) -> Information (conclusion)

 \Rightarrow

Students -> Top
 -> Average

Система управления базами данных / Database management system

- Система управления базами данных (СУБД) позволяет хранить данные таким образом, что становится легче получать, манипулировать и создавать информацию
- A database management system (DBMS) stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Характеристики современной СУБД / Characteristics of a modern DBMS

- Предметная область / Real-world entity
- Отношение таблиц / Relation-based tables
- Изоляция данных и приложения / Isolation of data and application
- Уменьшение избыточности / Less redundancy
- Согласованность / Consistency
- Язык запросов / Query language
- Свойства ACID / ACID properties
- Многопользовательский и одновременный доступ / Multiuser and concurrent access
- Множество представлений / Multiple views
- Безопасность / Security

Предметная область / Real-world entity

Student (сущность / entity)
id
name
grade (атрибут / attribute)
...
age
etc.

- множество сущностей / set of entities
- свойства / attributes
- отношение / relationships

Отношения таблиц / Relation-based tables

Table "Students"

id

name

grade

=>

1,John Doe,2

2,Petro Petrenko,1

Table "Courses"

id

title

credits

=>

1,Programming,5

2,Databases,6

Relation "Students-Courses"

Students.id

Courses.id

=>

1,2

2,1

2,2

Изоляция данных и приложения / Isolation of data and application

Data – records

Application – DBMS

Metadata (DBMS)

id

name

grade

Data (records)

1

“John Doe”

2

Уменьшение избыточности / Less redundancy

Правила нормализации / Normalization rules
Избыточность / Redundancy

Table “Students”

<u>id</u>	<u>name</u>	<u>grade</u>	<u>course</u>	<u>credits</u>
1	John Doe	2	OOP	5
2	John Doe	2	Math	3
3	John Doe	2	DB	6

Согласованность / Consistency

- Согласованность – это состояние, при котором каждое отношение в базе данных остается согласованным.
- Consistency is a state where every relation in a database remains consistent.

Metadata

id : int

name : string

grade : int

Data

“A1”

1234

2,5

Язык запросов / Query language

- Получение, обработка данных
- Retrieve and manipulate data

SELECT * FROM Students

=>

<u>id</u>	<u>name</u>	<u>grade</u>
1	John Doe	2
2	Petro Petrenko	1

DELETE FROM
Students WHERE id = 1

=>

<u>id</u>	<u>name</u>	<u>grade</u>
2	Petro Petrenko	1

Свойства ACID / ACID properties

- Атомарность / Atomicity
- Согласованность / Consistency
- Изоляция / Isolation
- Долговечность / Durability

Применяются к **транзакциям**, которые манипулируют данными в базе данных

These concepts are applied on **transactions**, which manipulate data in a database

Транзакцию можно определить как группу задач.

Единственная задача – это минимальный блок обработки, который больше нельзя разделить.

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Money transfer transaction	
Account A	Account B
open(A) oldBalance = A.balance newBalance = oldBalance - 100 A.balance = newBalance close(A)	open(B) oldBalance = B.balance newBalance = oldBalance + 100 B.balance = newBalance close(B)

Многопользовательский и одновременный доступ / Multiuser and concurrent access

- СУБД поддерживает многопользовательскую среду и позволяет одновременно получать доступ к данным и управлять ими.
- DBMS supports multi-user environment and allows them to access and manipulate data in parallel.

Множество представлений / Multiple views

- СУБД предлагает несколько представлений для различных пользователей. Эта функция позволяет пользователям просматривать базу данных в соответствии с их требованиями.
- DBMS offers multiple views for different users. This feature enables the users to have a concentrate view of the database according to their requirements.

Supply information

ContractNumber:

ContractDate:


SupplierID:

ContractName:

Comment:

Suppliers

Supplier



Date/time

Products

ContractNumber	Product	Amount	PricePerItem
1	TV	10	1253.45
1	Audio Player	25	655.12
1	Video Player	12	722.33
1	Stereo System	12	220.45
1	PC	24	1554.22

Total

Работа с данными
о поставках
Work with
supply data

Работа с
данными об
остатках
продукции
Work with data
about stock

Available Products

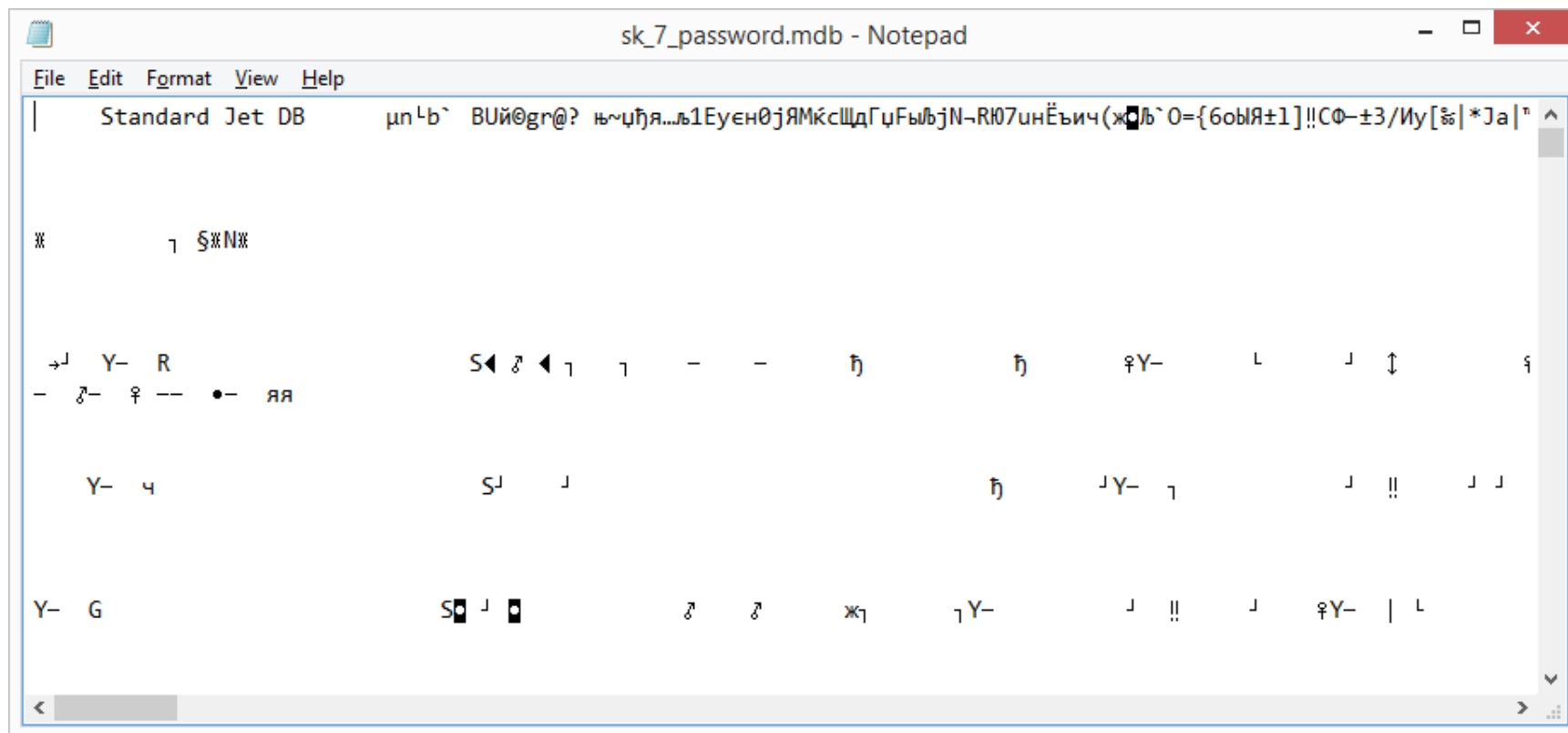
Product	SupplierName	Amount	PricePerItem	Cost
Audio Player				
	Petrov P. P. PE	11	544	5984.00
	Petrov P. P. PE	22	323.19	7110.18
	"Interfrut" LLC	33	585.67	19327.11
	Ivanov I. I. PE	15	455.14	6827.10
	Ivanov I. I. PE	35	655.12	22929.20
Summary for 'Product' = Audio Player (5 detail records)				
	Total	116	Total Cost	62177.59 UAH
Product	Week 36	Week 37	Week 39	Week 40
Audio Player	35	26	55	

Безопасность / Security

- Различные представления с различными функциями для различных пользователей / Different views with different features for multiple views
- Ограничение ввода и получения данных / Data entering and retrieving constraints
- СУБД не хранится на диске как обычный текстовый файл / DBMS is not saved on the disk as traditional text file

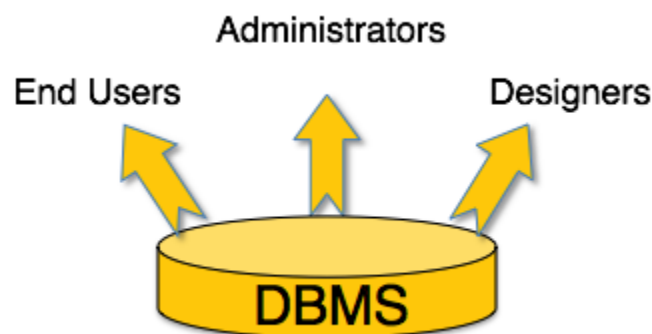
All Tables	Suppliers	LegalEntities	IndividualEntrepreneurs	Contracts		
Suppliers	ContractNumber	ContractDate	SupplierID	ContractName	Comment	Add New Field
Suppliers : Table	1	9/1/1999	1	Contract 1	Invoice 34 from 8/30/99	
LegalEntities	2	9/10/1999	1	Contract 2	Invoice 08-78 from 8/28/99	
LegalEntities : Table	3	9/10/1999	3	Contract 3	Invoice 08-78 from 8/28/99	
IndividualEntrepreneurs	4	9/23/1999	3	Contract 4	Order 56 from 8/28/99	
IndividualEntrepreneurs : Table	5	9/24/1999	2	Contract 5	Invoice 74 from 9/11/99	
Contracts	6	10/1/1999	1	Contract 6	Invoice 9-12 from 9/28/99	
Contracts : Table	HELLO		2	Contract 7	Invoice 85 from 9/21/99	
Supplied						
Supplied : Table						

Попытка ввести строковое значение в поле для ввода даты
Trying to enter the string into the data field



Попытка открыть файл базы данных как текстовый файл
Trying to open the database file as the text file

Пользователи / Users



- Администраторы поддерживают СУБД и несут ответственность за администрирование базы данных.
- Administrators maintain the DBMS and are responsible for administering the database.

- Дизайнеры – это группа людей, которые непосредственно работают над проектированием базы данных.
- Designers are the group of people who actually work on the designing part of the database.
- Конечные пользователи – это те, кто непосредственно пользуется преимуществами работы СУБД.
- End users are those who actually reap the benefits of having a DBMS.

Архитектура / Architecture

- Одноуровневая / Single-tier

- Многоуровневая / Multi-tier

Связанные но независимые модули / Related but independent modules

- Модифицировать / Modify
- Дополнять / Alter
- Изменять / Change
- Заменять / Replace

1-уровневая архитектура / 1-tier architecture

Пользователь использует единственный объект, которым является непосредственно СУБД

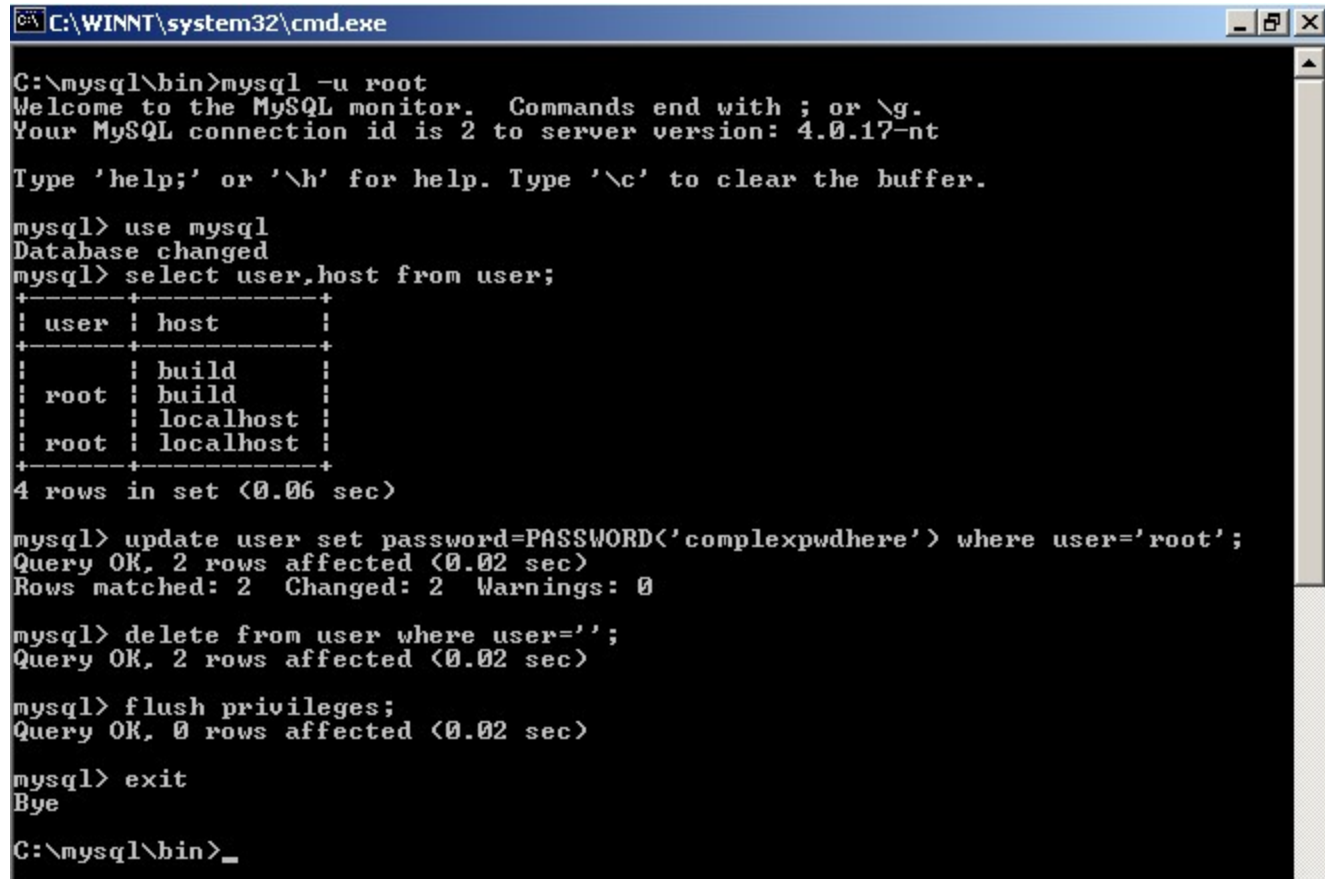
DBMS is the only entity where the user directly sits and uses it

2-уровневая архитектура / 2-tier architecture

СУБД имеет приложение, через которое можно получить доступ к СУБД

DBMS must have an application through which the DBMS can be accessed

Tier ?



```
C:\WINNT\system32\cmd.exe

C:\mysql\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.17-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use mysql
Database changed
mysql> select user,host from user;
+-----+-----+
| user | host |
+-----+-----+
|      | build |
| root | build |
|      | localhost |
| root | localhost |
+-----+-----+
4 rows in set (0.06 sec)

mysql> update user set password=PASSWORD('complexpwdhere') where user='root';
Query OK, 2 rows affected (0.02 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> delete from user where user='';
Query OK, 2 rows affected (0.02 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.02 sec)

mysql> exit
Bye
C:\mysql\bin>_
```

<https://www.netikus.net/documents/MySQLServerInstallation/setupsecurity.htm>

Tier ?

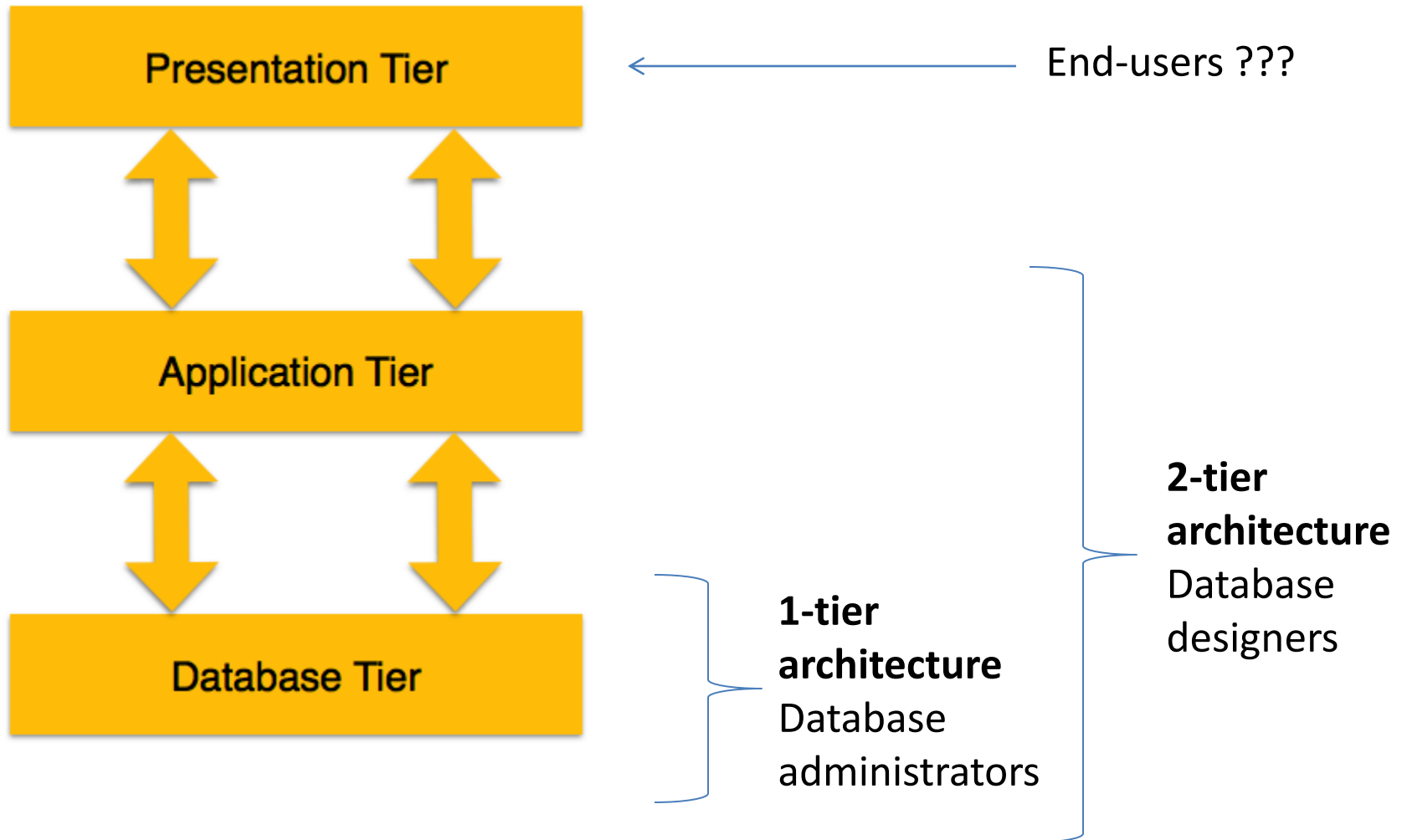
The screenshot displays the phpMyAdmin web interface. On the left, a sidebar shows a tree of databases including 'Usuarios', 'VSet', 'Xss', 'uam', 'uam2', 'ube_db', 'victoria_base', 'vseti', 'world', and 'New'. The 'world' database is selected, and the 'City' table is highlighted. The main panel shows the 'Structure' tab for the 'City' table. It lists five columns: 'ID' (int(11), primary key), 'Name' (char(35), latin1_swedish_ci), 'CountryCode' (char(3), latin1_swedish_ci), 'District' (char(20), latin1_swedish_ci), and 'Population' (int(11)). Below the column list, there are options to 'Check All', 'With selected', and buttons for 'Browse', 'Change', 'Drop', 'Primary', 'Unique', and 'Index'. At the bottom, there is a section for 'Indexes' and a table of 'Space usage' and 'Row Statistics'.

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	ID	int(11)			No	None	AUTO_INCREMENT	Change Drop More
2	Name	char(35)	latin1_swedish_ci		No			Change Drop More
3	CountryCode	char(3)	latin1_swedish_ci		No			Change Drop More
4	District	char(20)	latin1_swedish_ci		No			Change Drop More
5	Population	int(11)			No	0		Change Drop More

Space usage	
Data	266.9 KiB
Index	42 KiB
Total	308.9 KiB

Row Statistics	
Format	static
Collation	latin1_swedish_ci
Rows	4,079
Row length	67
Row size	78 B
Next autoindex	4,080
Creation	Apr 03, 2013 at 01:30 PM
Last update	Apr 03, 2013 at 01:30 PM

<https://sourceforge.net/projects/phpmyadmin/>



3-уровневая архитектура / 3-tier architecture

- Уровень данных / Database tier
 - Отношения / Relations
 - Язык запросов / Query language
- Уровень приложения / Application tier
 - Программы, которые имеют доступ к базе данных / Programs that access the database
 - Посредник между конечным пользователем и базой данных / A mediator between the end-user and the database
- Уровень презентации / Presentation tier
 - Программы предоставляют различные представления базы данных / Multiple views of the database are provided by applications
 - Конечные пользователи не знают о существовании базы данных за пределами этого уровня / End-users know nothing about any existence of the database beyond this layer

Tier ?



Packages:	Introductory (private)	Group of Mat classes	Semi-private sessions	Private sessions
individual class	FREE	\$20	\$55	\$80
5 classes	\$148	\$95	\$250	\$375
10 classes	-	\$180	\$450	\$700
20 classes	-	\$340	\$860	\$1300

Модели данных / Data models

- Определяют логическую структуру базы данных, которая моделируется
- Define how the logical structure of a database is modeled
- Каким образом данные связываются между собой / How data is connected to each other
- Каким образом данные обрабатываются и хранятся в системе / How data is processed and stored inside the system

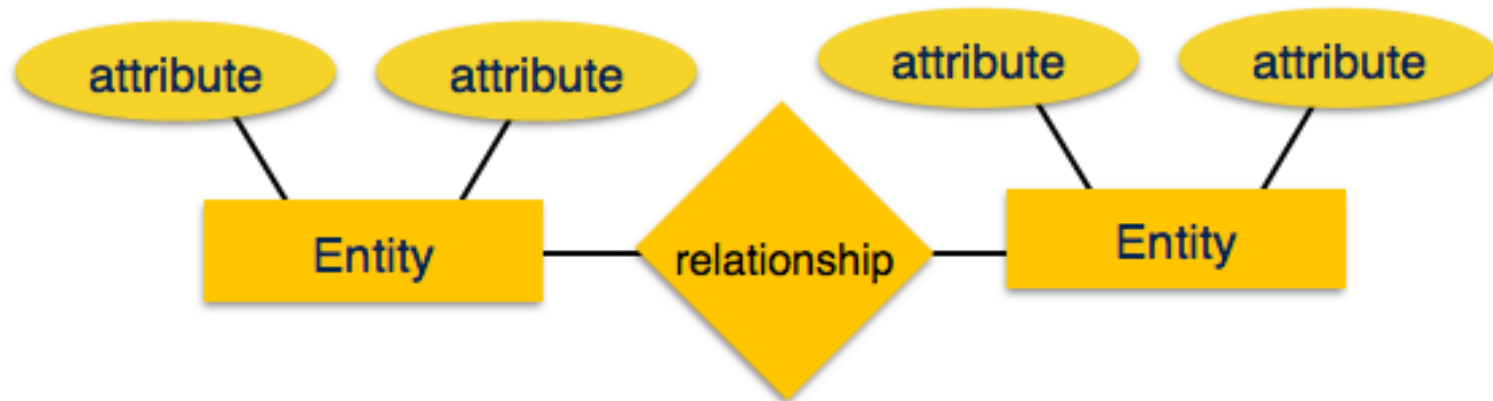
Модель сущность-связь / Entity-relationship model

- Основана на сведениях о реальных сущности и отношениях между ними
- Is based on the notion of real-world entities and relationships among them

Используется для концептуального проектирования базы данных / is used for the conceptual design of a database

ER модель включает / ER model is based on:

- **Сущности** и их **атрибуты** / **Entities** and their **attributes**
- **Отношения** между сущностями / **Relationships** among entities



Каждый атрибут определяется набором значений, который называется **доменом**

Every attribute is defined by its set of values called **domain**

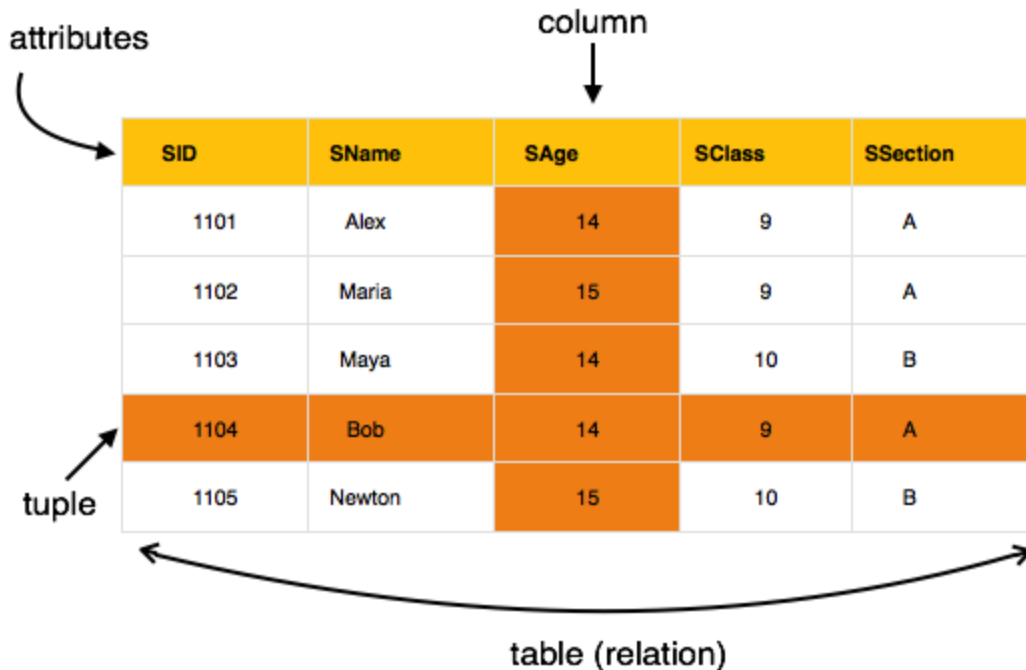
Мощность отношения определяет число ассоциаций между двумя сущностями

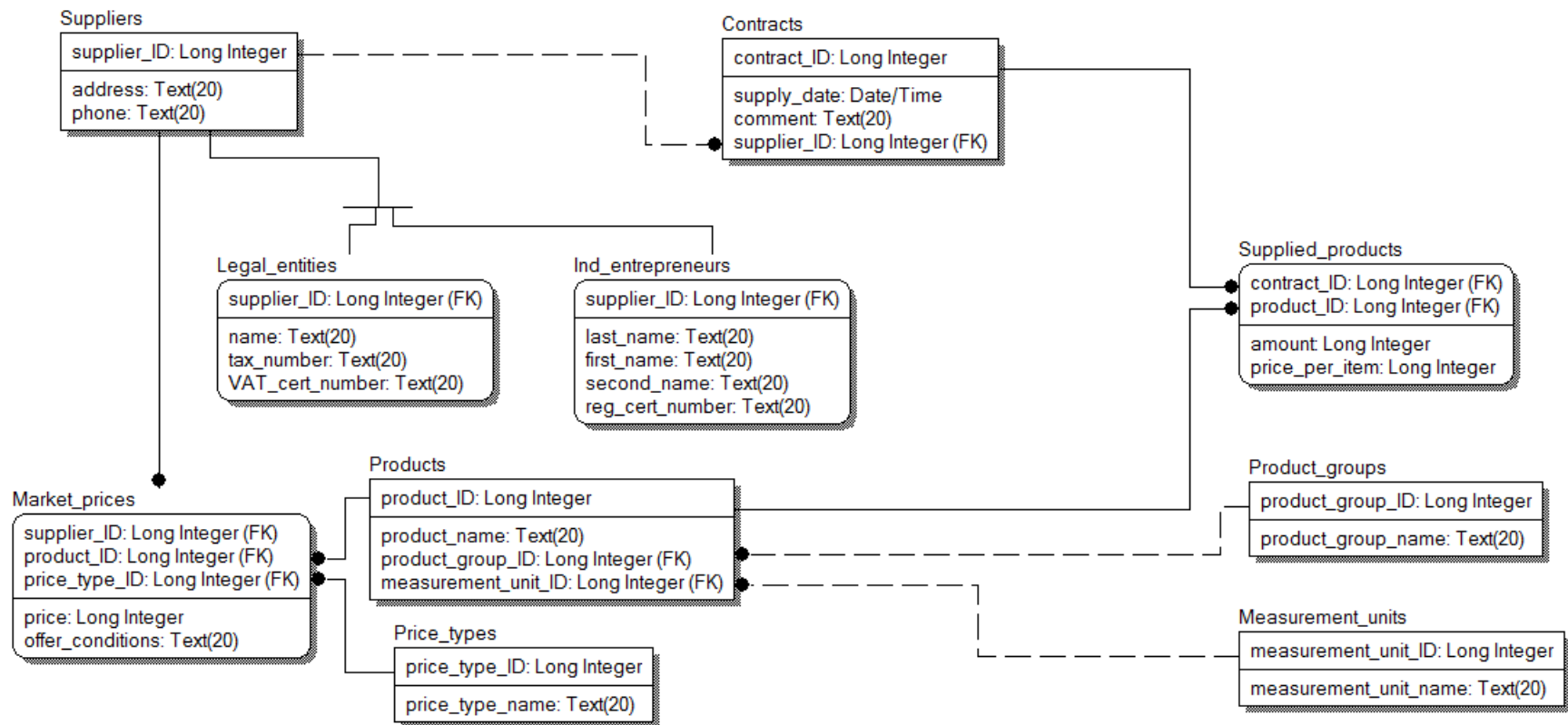
Mapping **cardinalities** define the number of association between two entities

1-to-1, 1-to-many, many-to-1, many-to-many

Реляционная модель / Relational model

- Определяет таблицы в виде n-арного отношения
- Defines table as an n-ary relation





Пример модели данных из лабораторного практикума
The example of the data model from lab classes

Основные особенности реляционной модели / The main highlights of the relational model:

- Данные хранятся в таблицах – **отношениях** / Data is stored in tables called **relations**
- Отношения могут быть **нормализованы** / Relations can be **normalized**
- В нормализованных отношениях хранимые значения являются **атомарными** / In normalized relations, values saved are **atomic** values
- Каждая строка в отношении содержит **уникальное** значение / Each row in a relation contains a **unique** value
- Каждый столбец в отношении содержит значения одного и того же **домена** / Each column in a relation contains values from a same **domain**

Нотация IDEF1X / IDEF1X notation

Сущность / Entity

Представление класса реальных или абстрактных вещей

The representation of a class of real or abstract things

Домен / Domain

Именованное множество значений данных одного и того же типа

A named set of data values all of the same data type

Атрибут / Attribute

Свойство или характеристика, которая является общей для некоторых или всех экземпляров сущности

A property or characteristic that is common to some or all of the instances of an entity

Ключ / Key

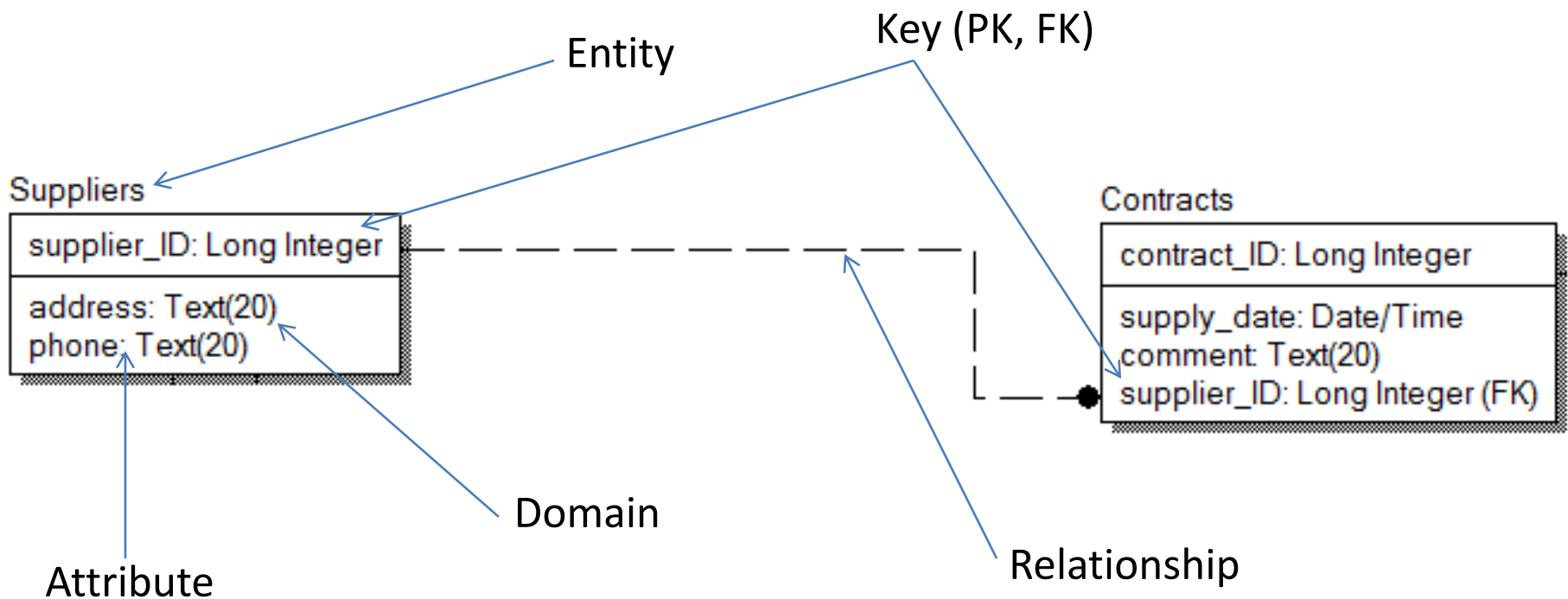
Атрибут или комбинация атрибутов сущности, значение которого однозначно идентифицирует каждый экземпляр сущности. Каждый такой набор представляет собой потенциальный ключ.

An attribute, or combination of attributes, of an entity whose values uniquely identify each entity instance. Each such set constitutes a candidate key.

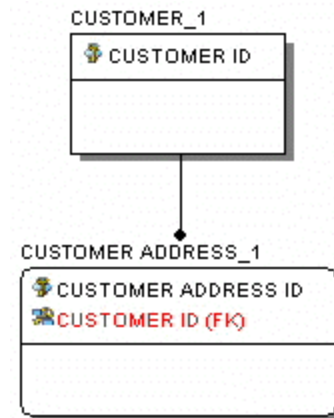
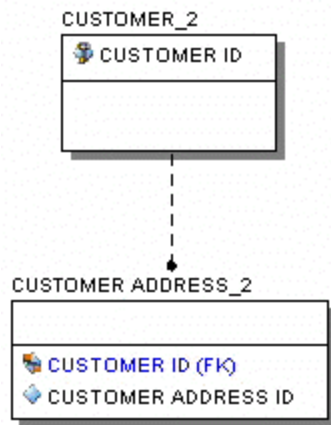
Отношение / Relationship

Ассоциация между экземплярами двух сущностей или между экземплярами одной и той же сущности.

An association between the instances of two entities or between instances of the same entity.

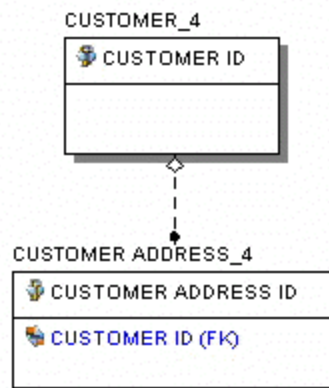


Идентифицирующее отношение / Identifying relationship



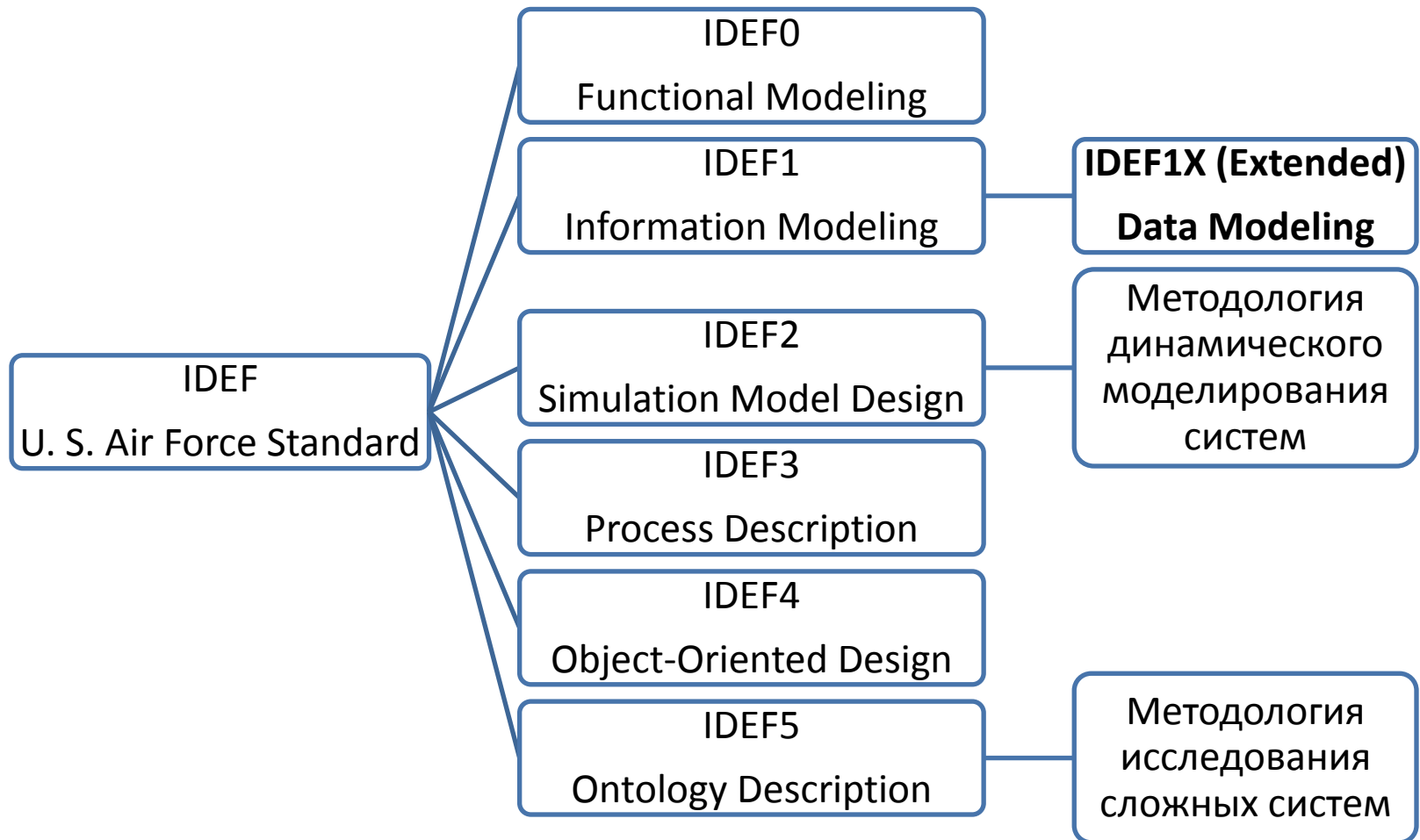
Неидентифицирующее отношение / Non-Identifying relationship

**Non-Identifying, Optional
Relationship**



- обязательное / mandatory
- необязательное / optional

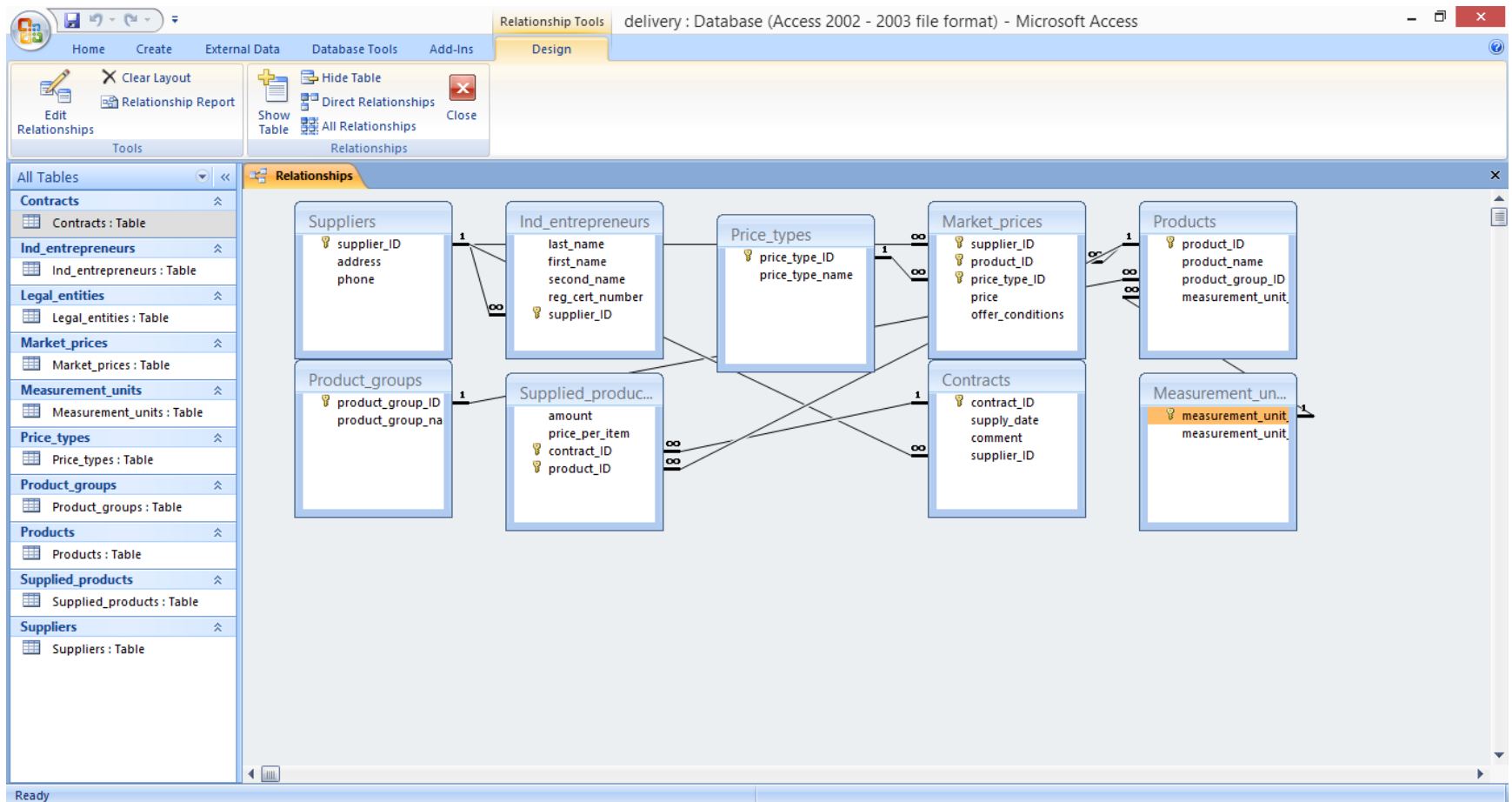
IDEF (Integrated DEFinition)



... IDEF6 – IDEF14

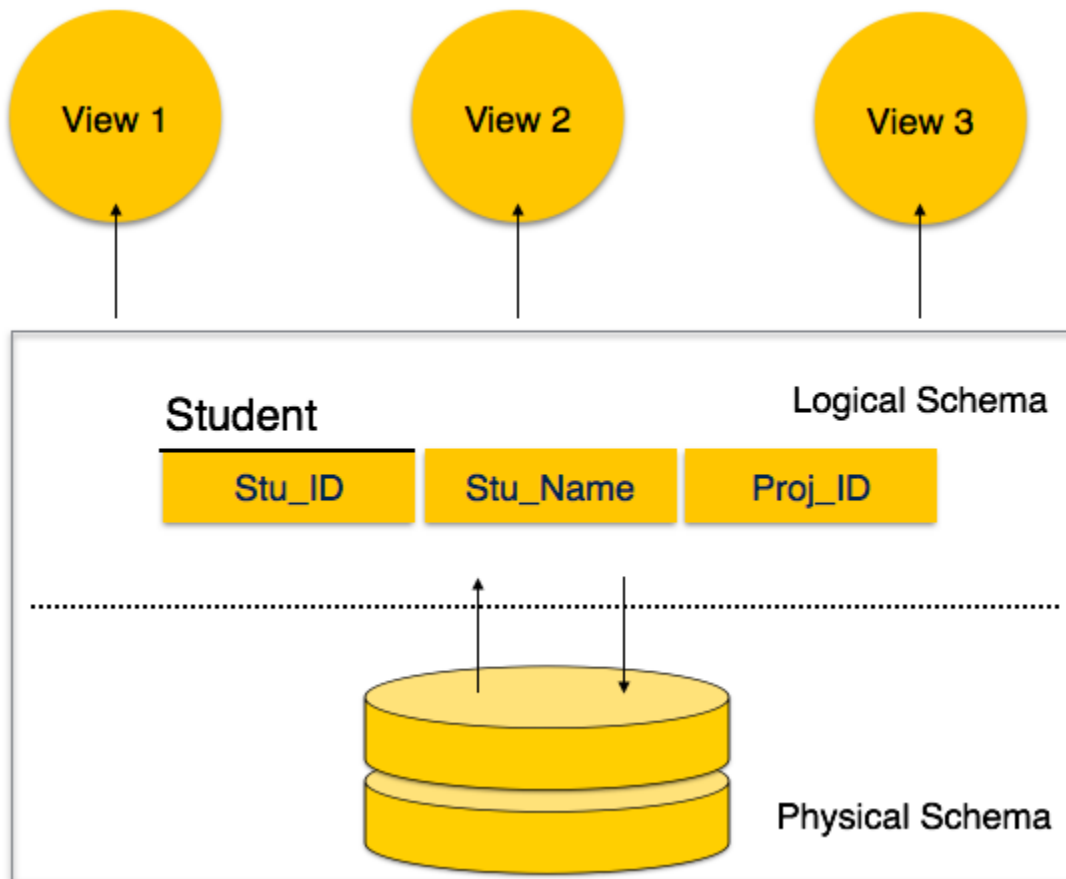
Схема базы данных / Database schema

- Схема базы данных – это "скелет" структуры, которая представляет логический вид всей базы данных, определяет каким образом организованы данные и какими отношениями они связаны, определяет все ограничения, которые накладываются на данные
- A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.



Пример схемы данных в СУБД Microsoft Access

The example of the database schema in DBMS Microsoft Access



Таблицы, представления и ограничения целостности
Tables, views, and integrity constraints

Фактическое хранения данных
в форме файлов, индексов и т.д.
Actual storage of data in the
form of files, indices, etc.

Экземпляр базы данных != Схема базы данных
Database instance != Database scheme

Экземпляр базы данных – состояние операционной базы данных, содержащей данные, в любое время.

Database instance is a state of operational database with data at any given time.

Экземпляры базы данных могут меняться со временем.

Database instances tend to change with time.

Независимость данных / Data independence

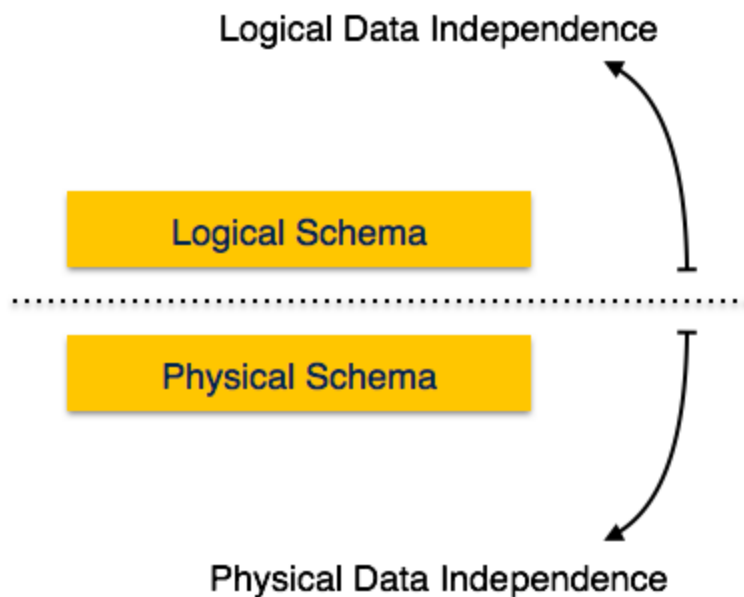
СУБД содержит / DBMS contains:
данные пользователей / users' data
метаданные / metadata

Логическая независимость данных – механизм, отделяющий метаданные от фактических данных, которые хранятся на диске.

Logical data independence is a kind of mechanism, which liberalizes data about database from actual data stored on the disk.

Физическая независимость данных – механизм, который обеспечивает возможность изменения физических данных без влияния на схему базы данных или логические данные.

Physical data independence is the power to change the physical data without impacting the schema or logical data.



Внесение изменений в формат таблицы не должно повлиять на данные, которые находятся на диске
If we do some changes on table format, it should not change the data residing on the disk

Замена HDD на SSD не должна повлиять на логические данные или схемы
Suppose we want to replace hard-disks with SSD – it should not have any impact on the logical data or schemas

Лекция 02: Хранение данных и
файловая структура

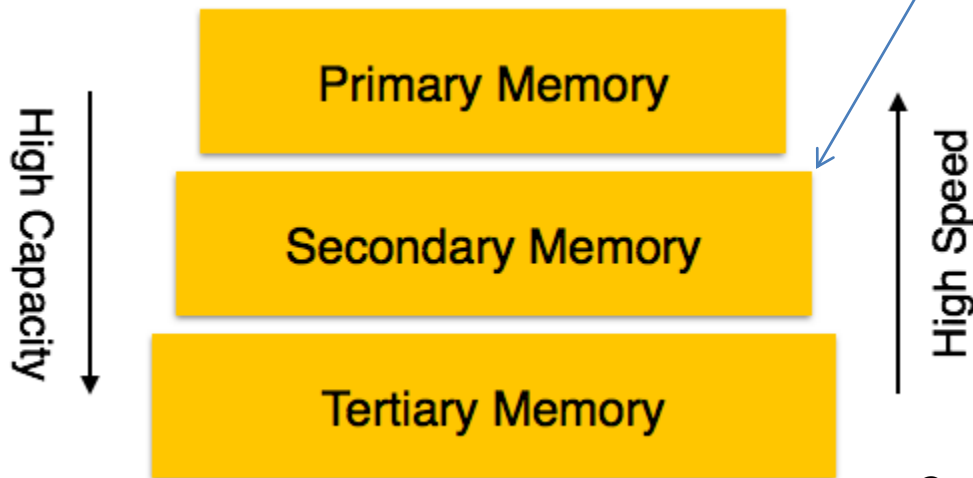
Lecture 02: Storage and file
structure

Система хранения данных / Storage system

- Базы данных хранятся в форматах файлов, содержащих записи. На физическом уровне фактические данные хранятся на некотором устройстве. Эти устройства хранения данных можно разделить на три типа.
- Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types.

Первичная память – непосредственно доступна для CPU
The memory storage that is directly accessible to the CPU.

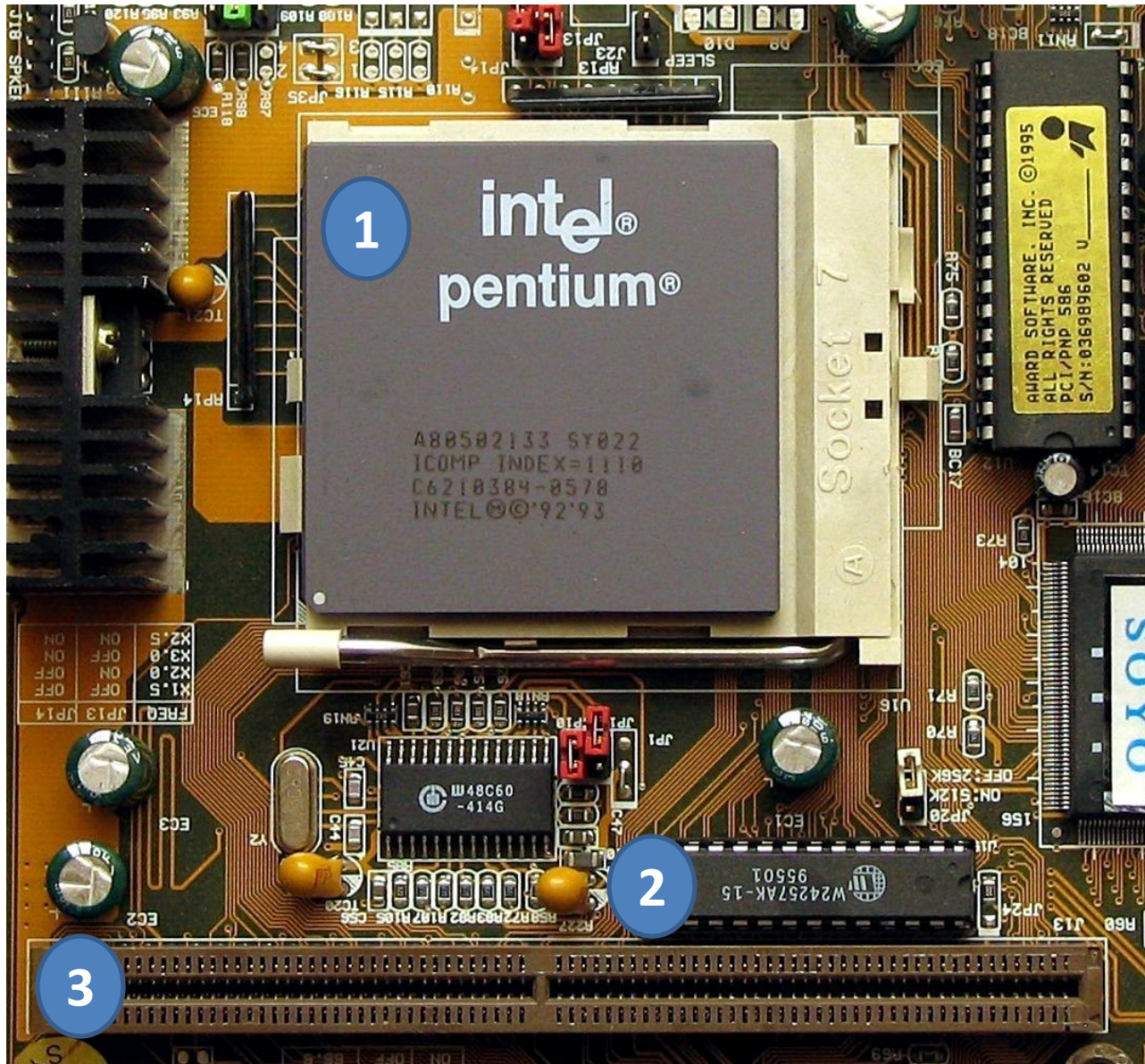
Вторичная память – используется для хранения данных с целью их дальнейшего использования, или в качества бэкапа.
Secondary storage devices are used to store data for future use or as backup.



Сторонняя память – используется для хранения больших объемов данных, например, для резервного копирования всей системы.
Tertiary storage is used to store huge volumes of data. These storage devices are mostly used to take the back up of an entire system.

Первичная память / Primary storage

- Память, непосредственно доступна для CPU, попадает в эту категорию. Внутренняя память процессора (registers), быстрая память (cache) и основная память (RAM) непосредственно доступны для CPU, поскольку все они размещены на материнской плате или на чипсете CPU. Этот накопитель, как правило, очень мал, сверхбыстрый и нестабилен. Основное хранилище требует постоянного источника питания, чтобы поддерживать его состояние. В случае отключения питания все его данные теряются.
- The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.



Вторичная память / Secondary storage

- Вторичные накопители используются для хранения данных для дальнейшего использования или резервного копирования. Вторичная память включает в себя устройства памяти, которые не являются частью чипсета CPU или материнской платы, например, магнитные диски и ленты, оптические диски (DVD, CD и др.), HDD, USB-накопители и SSD.
- Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks and tapes, optical disks (DVD, CD, etc.), HDD, flash drives, and SSD.



<https://www.pcworld.com/article/2864385/are-pc-hard-drives-destined-to-die-at-the-hand-of-the-cloud-maybe-analysts-say.html>

Сторонняя память / Tertiary storage

- Сторонний накопитель используется для хранения огромных объемов данных. Поскольку такие устройства хранения находятся вне компьютера, они являются наиболее медленными. Эти устройства хранения данных чаще всего используются для резервного копирования всей системы.
- Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system.



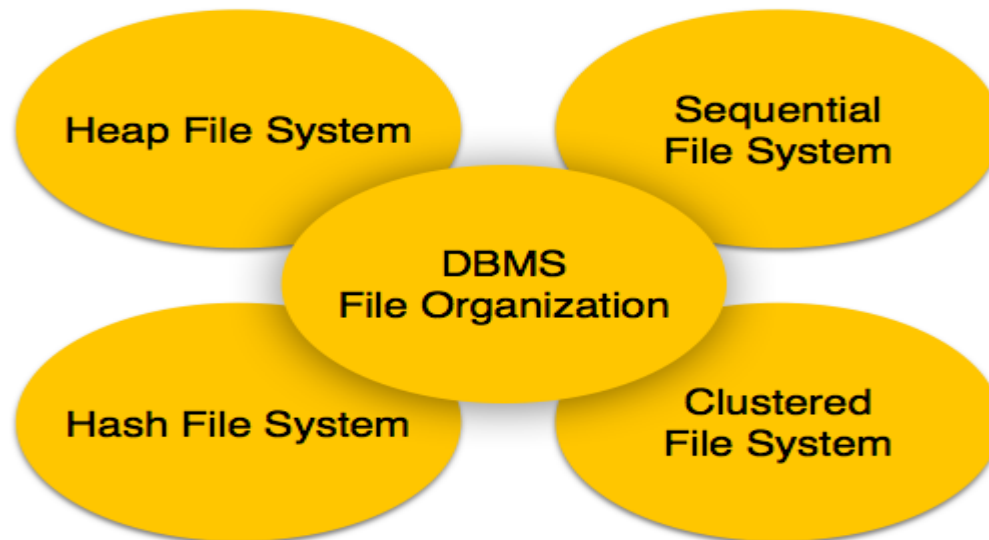
<https://www.bhphotovideo.com/explora/computers/buying-guide/recommended-external-hard-drives-photo-video-and-audio-production>

Файловая структура / File structure

- Связанные данные и информация хранятся вместе в форматах файлов. Файл – это последовательность записей, хранящихся в двоичном формате. Дисковый привод отформатирован в несколько блоков, которые могут хранить данные. Файловые записи отображаются на эти дисковые блоки.
- Relative data and information is stored collectively in file formats. A file is a sequence of records stored in binary format. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.

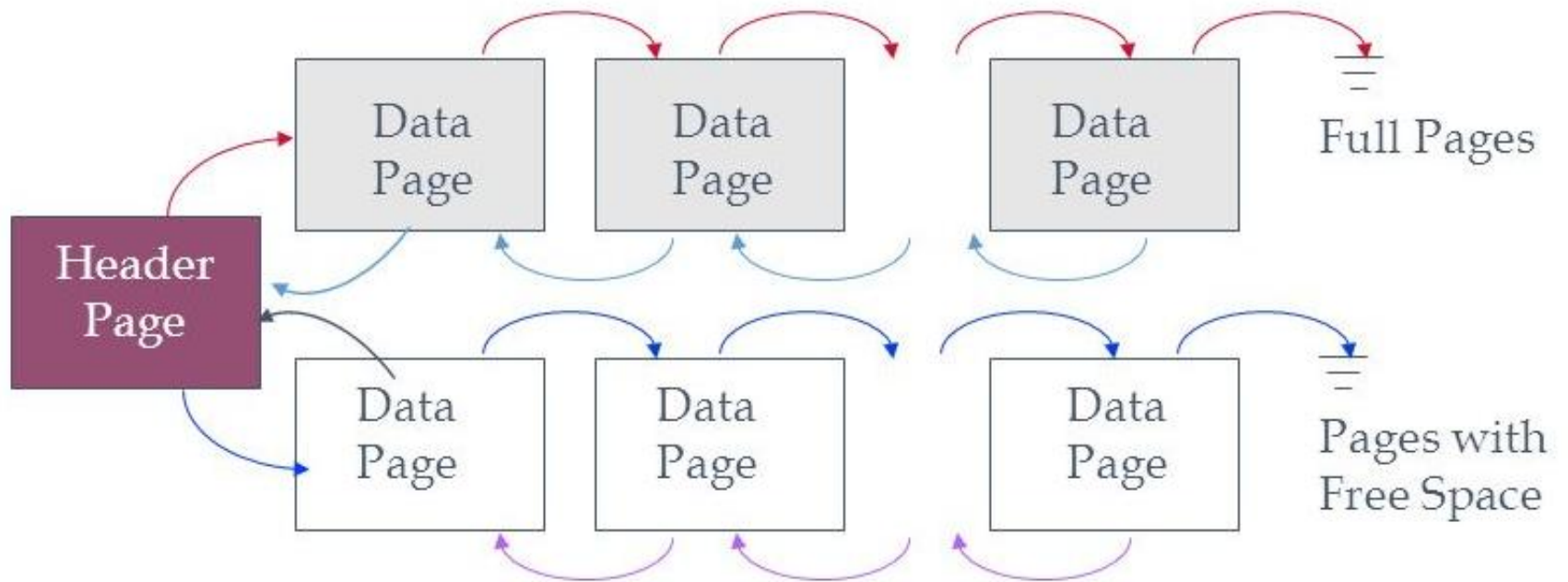
Организация файлов / File organization

- Организация файлов определяет способ записи файлов на блоки дисков. Существует четыре типа организации файлов для организации файловых записей.
- File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records.



Организация файлов в куче / Heap file organization

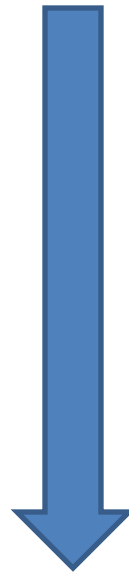
- Когда файл создается с помощью кучи, операционная система выделяет область памяти для этого файла без каких-либо дополнительных учетных данных. Файловые записи можно размещать в любой части выделенной области памяти. Ответственность за управление записями несет программное обеспечение. Файловая куча не поддерживает сортировку, упорядоченность или индексирования самостоятельно.
- When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.



Последовательная организация файлов / Sequential file organization

- Каждая запись файла содержит поле данных (атрибут), чтобы однозначно идентифицировать эту запись. В последовательной организации файлов записи размещаются в файле в некотором последовательном порядке на основе уникального ключевого поля или ключа поиска. Практически невозможно хранить все записи последовательно в физической форме.
- Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	



Записи в файле
упорядоченные
с помощью
ключа поиска

The records in the file
are ordered by
a search-key

Хэш организация файлов / Hash file organization

- Хэш организация файлов использует вычисления хэш-функций на некоторых полях записей. Вывод хэш-функции определяет расположение блока на диске, где нужно разместить записи.
- Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hash file organization of *instructor* file, using *dept_name* as key
(see previous slide for details).

Кластерная организация файлов / Clustered file organization

- Кластерная организация файлов не считается полезной для больших баз данных. В этом механизме связанные записи из одного или нескольких отношений сохраняются в том же дисковом блоке, то есть приведение записей не основывается на первичном ключе или ключи поиска.
- Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

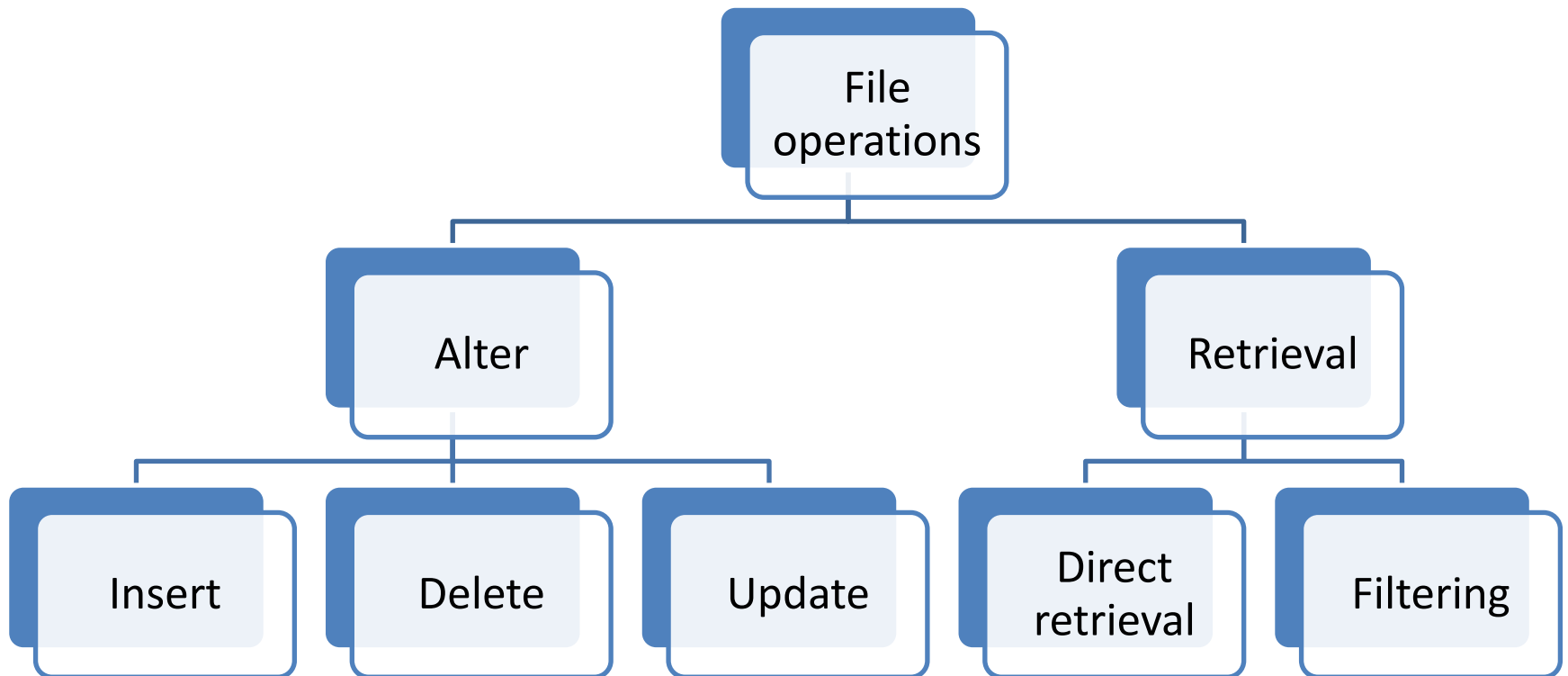
CLUSTER KEY



DEP_ID	DEP_NAME	EMP ID	EMP_NAME	EMP_ADD
D_101	ECO	01	JOE	CAPE TOWN
		02	PETER	CROY CITY
		03	MARY	NOVI
D_102	CS	04	JOHN	FRANSISCO
D_103	JAVA	05	ANNIE	FRANSISCO
D_104	MATHS	06	SAKACHI	TOKYO
D_105	BIO	07	SONI	W.LAND
D_106	CIVIL	08	LUNA	TOKYO

DEPARTMENT + EMPLOYEE

Файловые операции / File operations



Открытие – файл может быть открыт в одном из двух режимов: режим **чтения** или режим **записи**. В режиме чтения операционная система не позволяет никому менять данные. Иначе говоря, данные только для чтения. Файлы, открытые в режиме чтения, могут быть распределены между несколькими сущностями. Режим записи позволяет изменять данные. Файлы, открытые в режиме записи, можно читать, но их нельзя разделить.

Open – A file can be opened in one of the two modes, **read** mode or **write** mode. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.

Перемещение – каждый файл имеет указатель файла, который показывает текущую позицию, где данные нужно читать или записывать. Этот указатель можно соответственно скорректировать. Используя операцию поиска, указатель можно передвигать вперед или назад.

Locate – Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.

Чтение – по умолчанию, когда файлы открываются в режиме чтения, указатель файла указывает на начало файла. Существуют варианты, при которых пользователь может сообщить операционной системе, где искать указатель файла при открытии файла. Считываются следующие до указателя файла данные.

Read – By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.

Запись – пользователь может выбрать, чтобы открыть файл в режиме записи, который дает им возможность редактировать его содержимое. Это может быть удаление, вставка или модификация. Указатель файлов может находиться во время открытия или может быть динамически изменен, если операционная система это позволяет.

Write – User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.

Заккрытие – это важнейшая операция с точки зрения операционной системы.

Когда создается запрос на закрытие файла, операционная система:

- удаляет все замки (если в режиме совместного использования);
- сохраняет данные (если они изменены) на вторичные носители информации;
- выпускает все буферы и обработчики файлов, связанные с файлом.

Close – This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system

- removes all the locks (if in shared mode);
- saves the data (if altered) to the secondary storage media;
- releases all the buffers and file handlers associated with the file.

Лекция 03: Модель сущность-
связь

Lecture 03: Entity-relationship
model

Модель ER / ER model

- Модель ER определяет концептуальный вид базы данных. Она определяет реальные сущности и ассоциации между ними. На уровне просмотра ER модель считается приемлемым вариантом для проектирования баз данных. Представлена Питером Ченом в 1976 году.
- The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases. Proposed by Peter Chen in 1976.

Сущность / Entity

- Сущность может быть реальным объектом, живым или неживым, который можно легко идентифицировать.
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable.
- Набор сущностей – совокупность сущностей одного и того же типа.
- An entity set is a collection of similar types of entities.

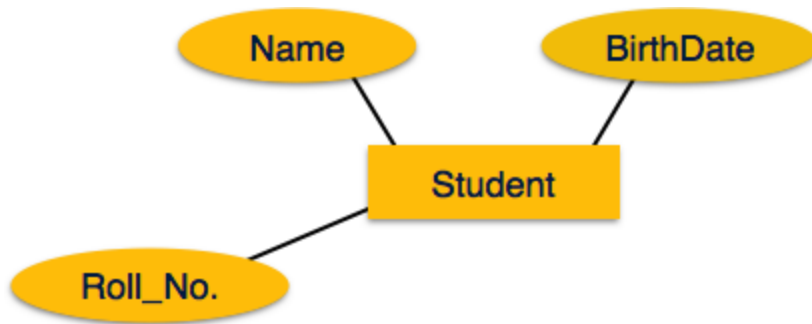
Атрибуты / Attributes

- Объекты представляются с помощью их свойств, которые называются атрибутами. Все атрибуты имеют значения.
- Entities are represented by means of their properties, called attributes. All attributes have values.
- Существует **домен** или диапазон значений, которые могут быть назначены для атрибутов.
- There exists a ***domain*** or range of values that can be assigned to attributes.

Сущности представляются с помощью прямоугольников.

Прямоугольники называются сущностью, которую они представляют.

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



Атрибуты – это свойства сущностей.

Атрибуты представляются с помощью эллипсов. Каждый эллипс представляет один атрибут и непосредственно связан с его сущностью (прямоугольником).

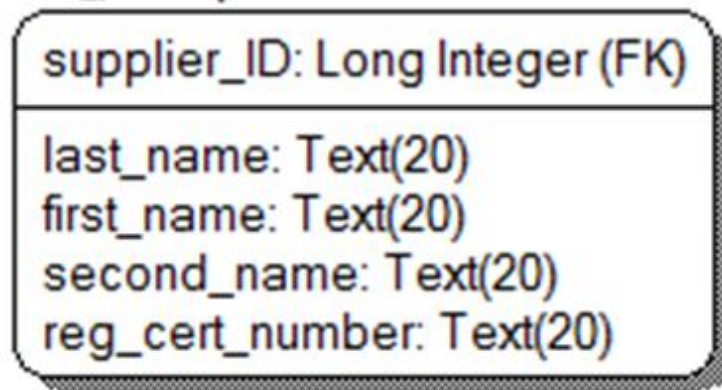
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

Представление моделей данных / Data models representation

Entity

Attributes

Ind_entrepreneurs



- Supplier ID (PK)
- Supplier full name
- Registration number

Типы атрибутов / Types of attributes

Простые атрибуты – это атомарные значения, которые невозможно разделить далее.

Simple attributes are atomic values, which cannot be divided further.

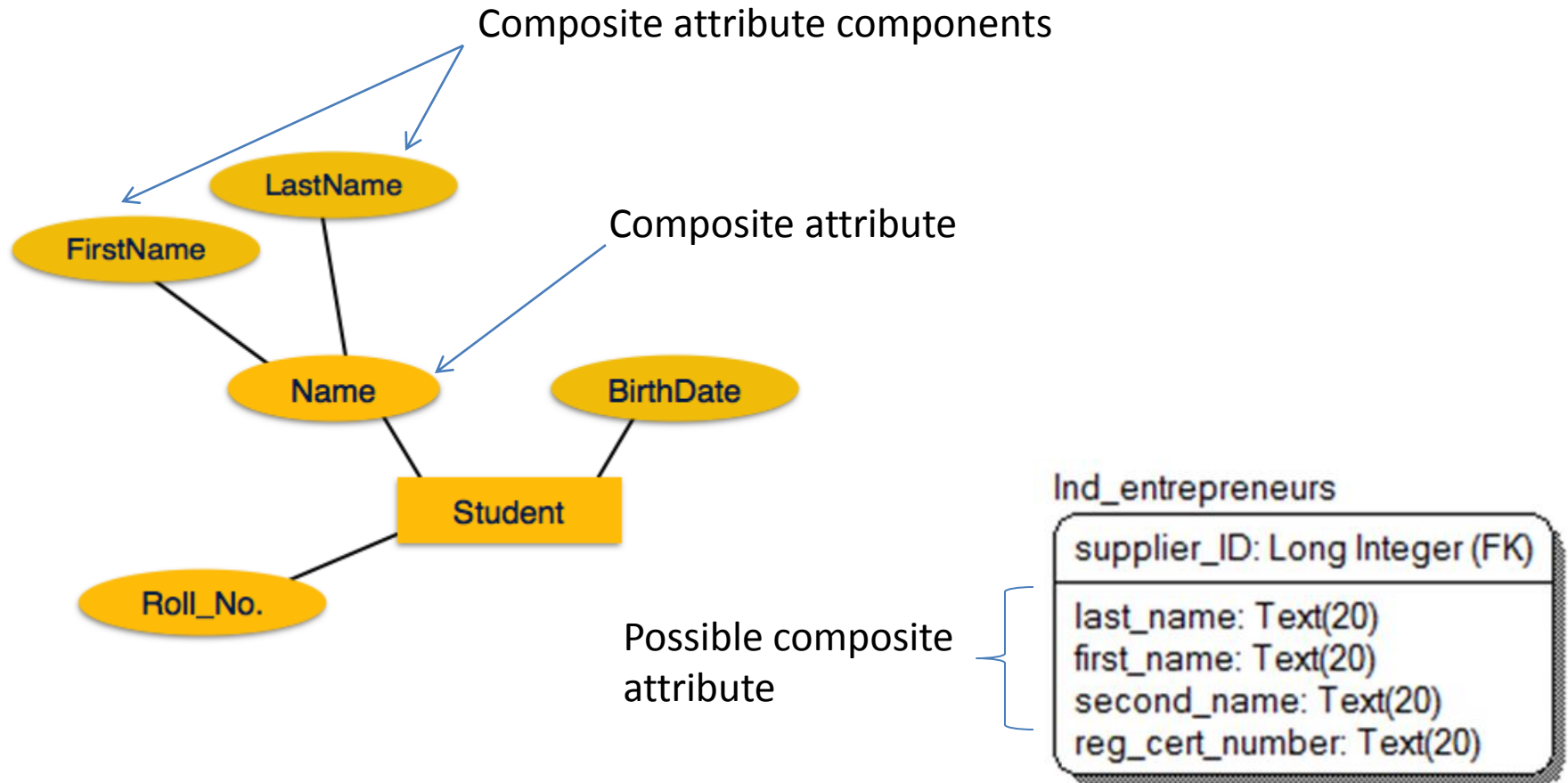
Student ID	First Name	Phone Number
1	Jonh Doe	+38 012 345 6789
...
999	Petrenko Petro Petrovych	+38 098 765 4321

- **Составные атрибуты** состоят из более чем одного простого атрибута.
- **Composite attributes** are made of more than one simple attribute.

Student ID	First Name	Last Name
1	John	Doe
...
999	Petro	Petrenko

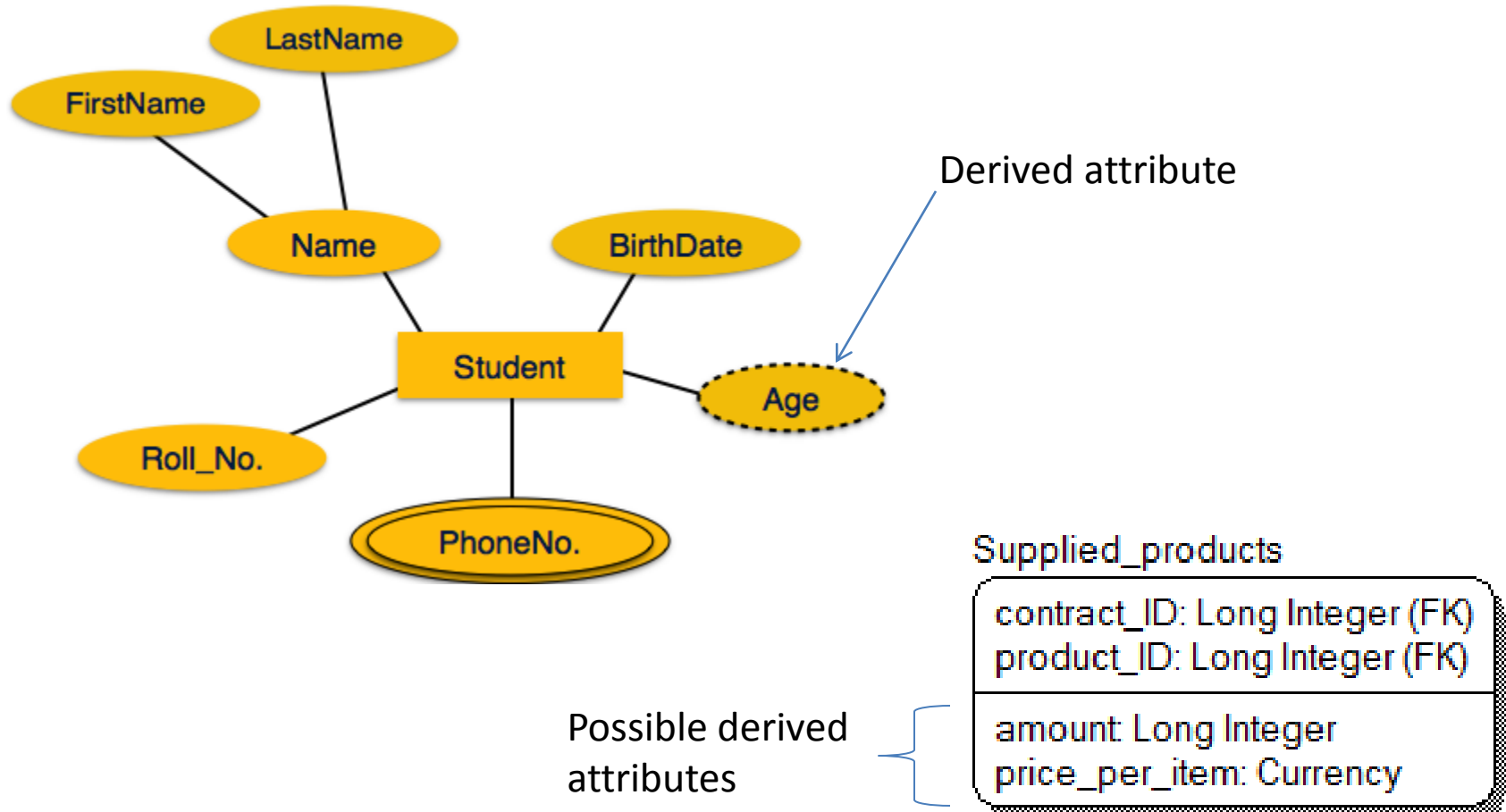
- Например, полное имя студента может иметь имя `first_name` и `last_name`.
- For example, a student's complete name may have `first_name` and `last_name`.

Представление составных атрибутов / Composite attributes representation



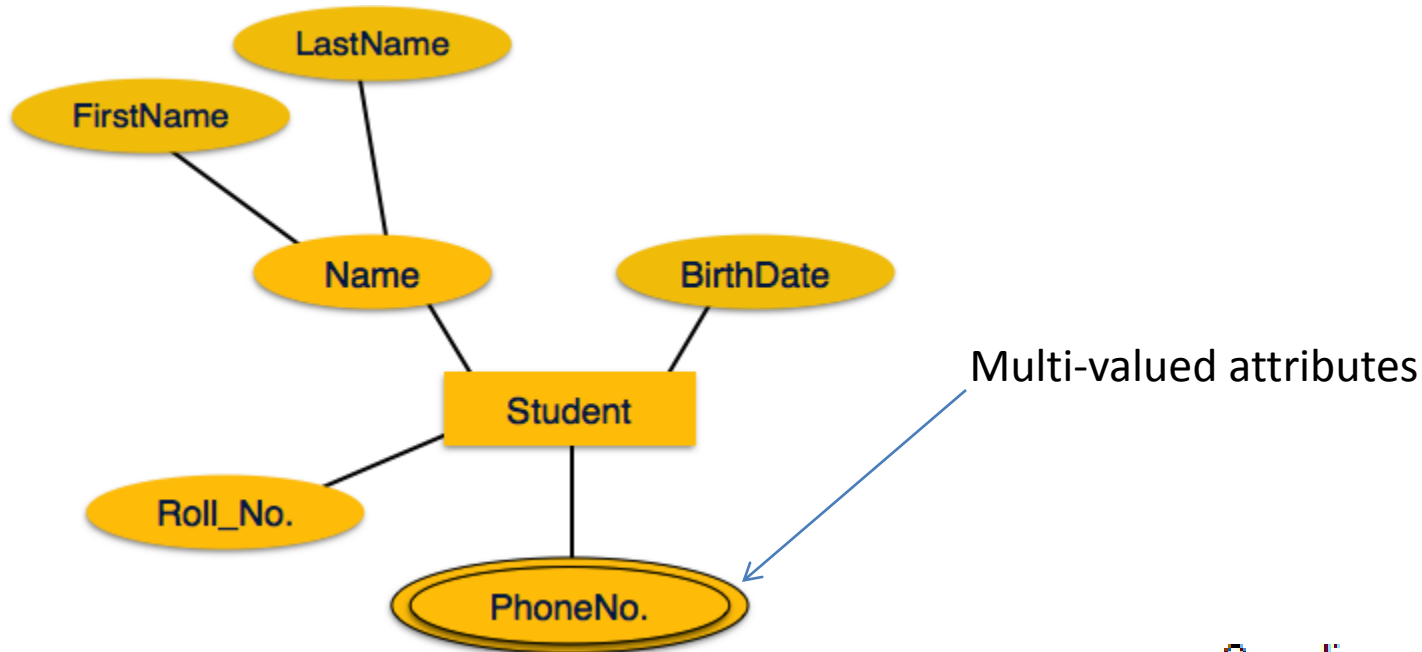
- **Производные атрибуты** – это атрибуты, которые не существуют в физической базе данных, но их значение происходят от других атрибутов, присутствующих в базе данных.
- **Derived attributes** are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
- Например, средняя зарплата в отделе не должна храниться непосредственно в базе данных, вместо этого она может быть выведена.
- For example, average salary in a department should not be saved directly in the database, instead it can be derived.

Представление производных атрибутов / Derived attributes representation



- **Однозначные атрибуты** содержат одно значение. **Многозначные атрибуты** могут содержать несколько значений.
- **Single -value attributes** contain single value. **Multi-value attributes** may contain more than one values.
- Например – Social_Security_Number. Например, человек может иметь несколько номеров телефона, адреса электронной почты и прочее.
- For example – Social_Security_Number. For example, a person can have more than one phone number, email address, etc.

Представление многозначных атрибутов / Multi-valued attributes representation



Possible multi-valued
attributes

Suppliers

supplier_ID: Long Integer
address: Text(20)
phone: Text(20)

Эти типы атрибутов могут комбинироваться таким образом :

- простые однозначные атрибуты;
- простые многозначные атрибуты;
- составные однозначные атрибуты;
- составные многозначные атрибуты.

These attribute types can come together in a way like:

- simple single-valued attributes;
- simple multi-valued attributes;
- composite single-valued attributes;
- composite multi-valued attributes.

Ключевые атрибуты / Keys

Ключ – это атрибут или набор атрибутов, которые однозначно идентифицируют сущность между набором сущностей.

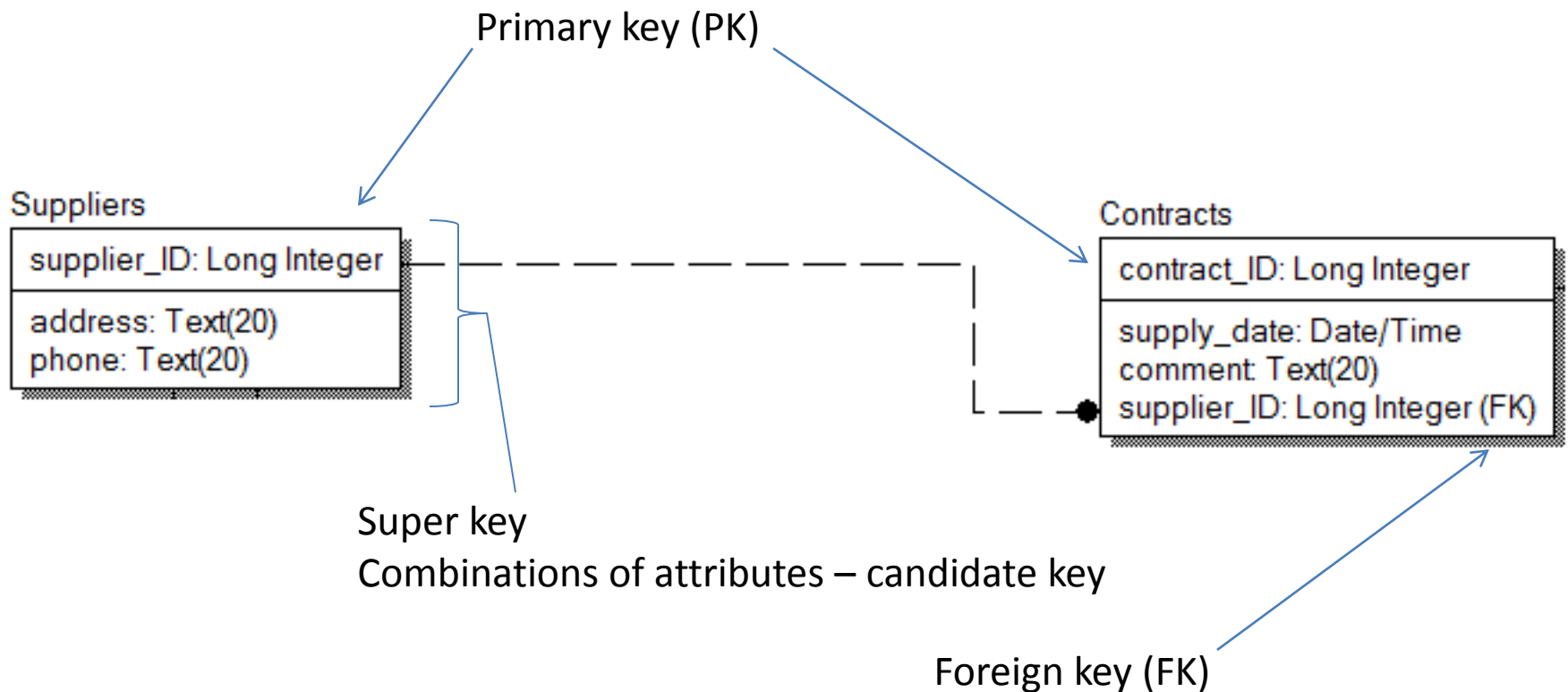
Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

Например, номер зачетной книжки студента идентифицирует его / ее среди всех студентов.

For example, the roll_number of a student makes him/her identifiable among students.

- **Супер-ключ** – набор атрибутов (один или несколько), который совокупно идентифицирует сущность в наборе сущностей.
- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Потенциальный ключ** – минимальный супер-ключ называется потенциальным ключом. Набор сущностей может содержать несколько потенциальных ключей.
- **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Первичный ключ** – это один из потенциальных ключей, избранных дизайнером баз данных для того, чтобы однозначно идентифицировать набор сущностей.
- **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Представление ключевых атрибутов / Key attributes representation



Отношение / Relationship

- Ассоциация между сущностями называется отношением.
- The association among entities is called a relationship.
- Например, сотрудник работает на кафедре, студент зачисляется к курсу. Здесь, «работает» и «засчитывается» – отношение.
- For example, an employee works_at a department, a student enrolls in a course. Here, works_at and Enrolls are called relationships.

Набор отношений / Relationship set

- Множество отношений одного и того же типа называется набором отношений. Как и сущности, отношения тоже могут иметь атрибуты. Эти атрибуты называются описательными атрибутами.
- A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Степень отношения / Degree of relationship

Количество сущностей, участвующих в отношении, определяет степень отношения:

- бинарное = степень 2,
- тернарное = степень 3;
- n-арное = степень n.

The number of participating entities in a relationship defines the degree of the relationship:

- Binary = degree 2;
- Ternary = degree 3;
- n-ary = degree n.

Мощности отображения / Mapping cardinalities

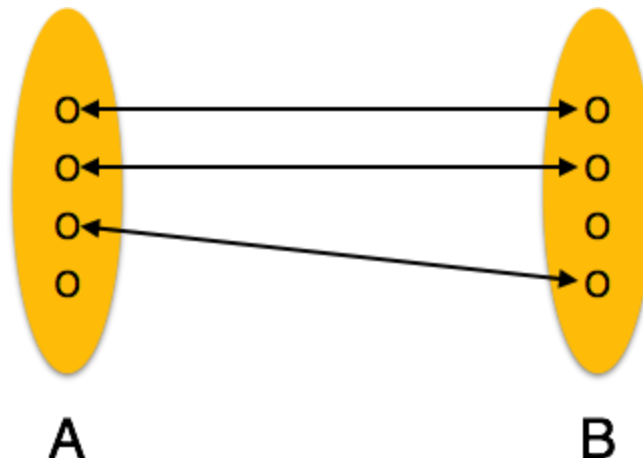
Мощность определяет количество сущностей в одном наборе сущностей, которые могут быть связаны с количеством сущностей другого набора через набор отношений.

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

Один-к-одному / One-to-one

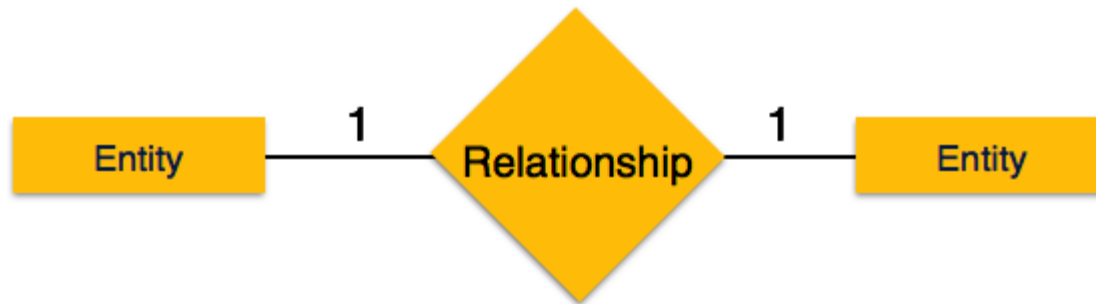
Одна сущность из набора сущностей A, может быть связана не более чем с одной сущностью из набора сущностей B, и наоборот.

One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

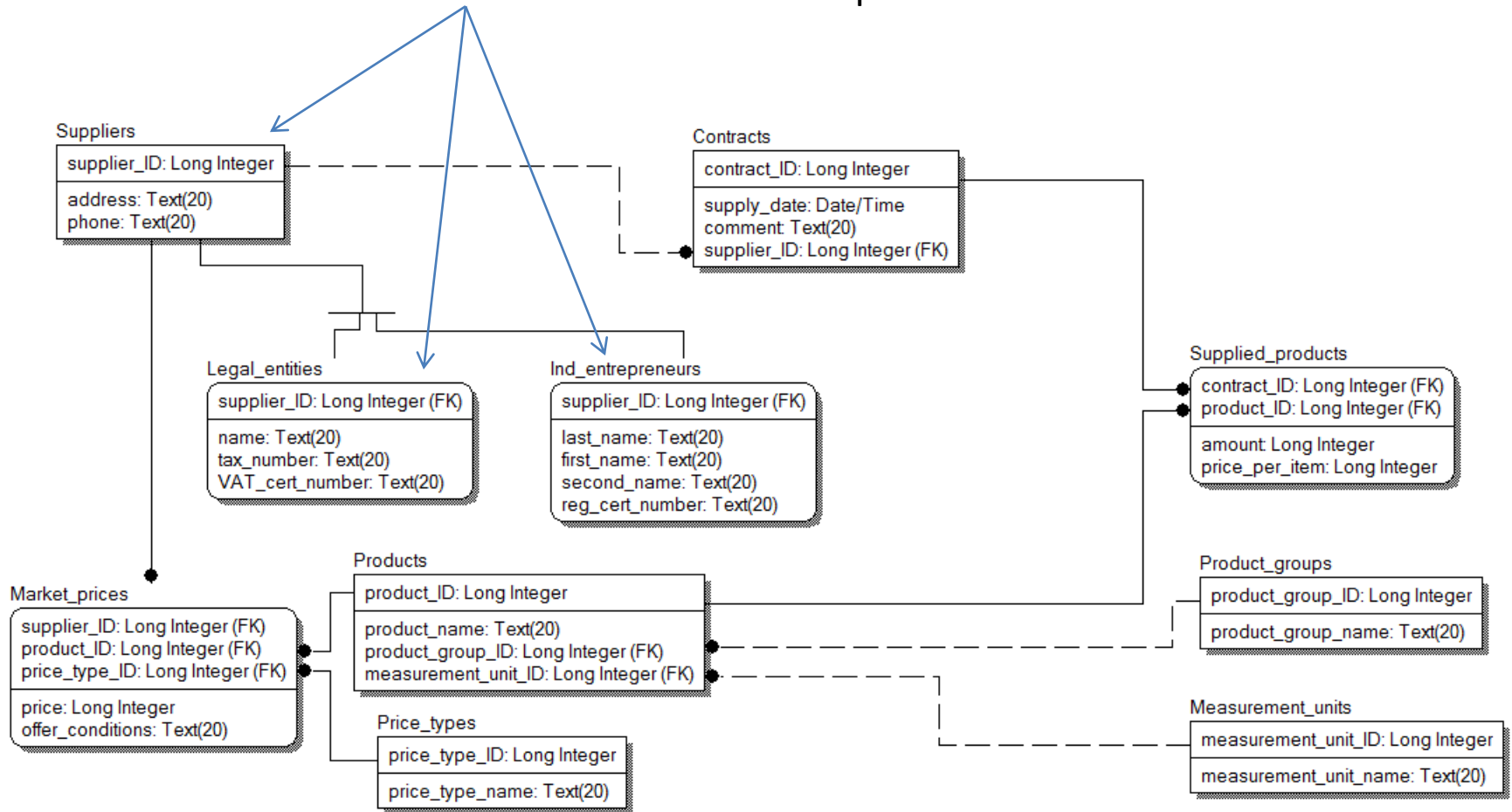


Один-к-одному – только один экземпляр сущности связан с отношением, он обозначается как "1: 1". Следующее изображение показывает, что только один экземпляр каждой сущности должен быть связан с отношением.

One-to-one – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship.



One-to-one relationship



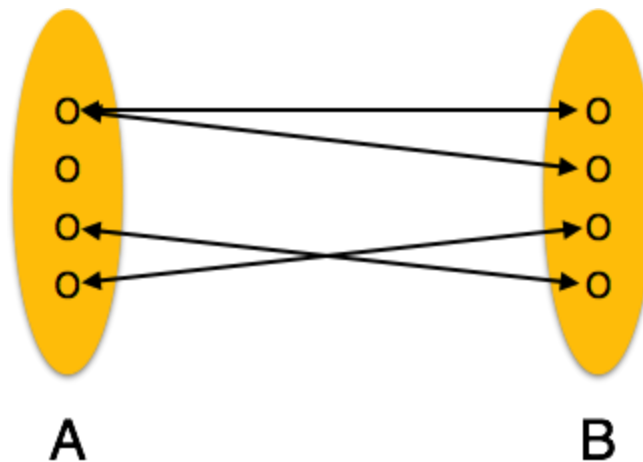
Each supplier entry should be provided with the additional information depending on its type.

Для каждого поставщика должна быть определена дополнительная информация в зависимости от его типа.

Один-ко-многим / One-to-many

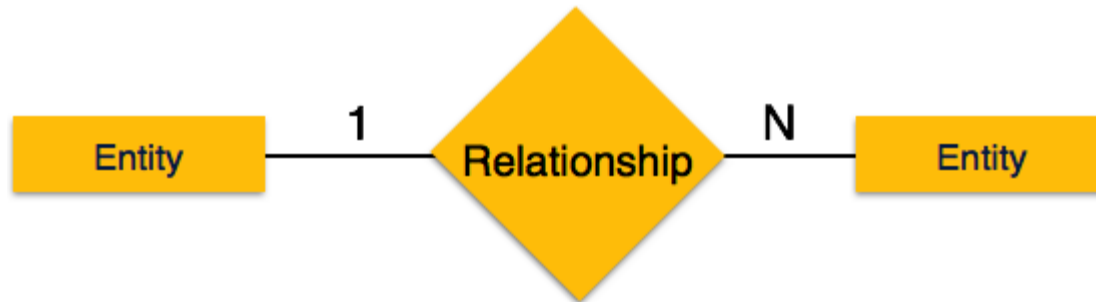
Одна сущность из набора сущностей A может быть ассоциирована с несколькими сущностями из набора сущностей B, однако сущность из набора сущностей B может быть ассоциирована не более чем с одной сущностью.

One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

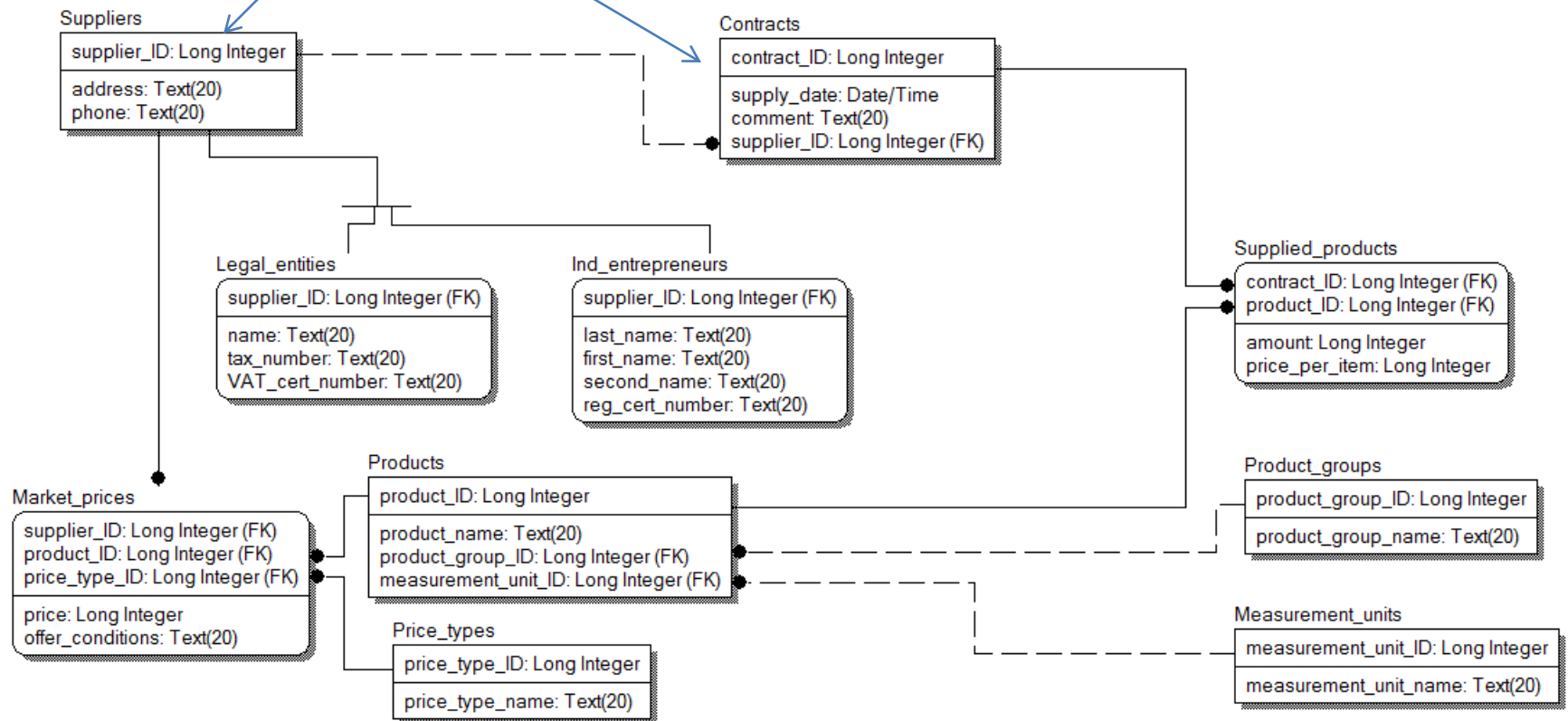


Один-ко-многим – если более одного экземпляра сущности связано с отношением, оно обозначается как '1: N'.
Следующее изображение отражает, что только один экземпляр сущности слева и больше, чем один экземпляр сущности справа, может быть связан с этим отношением.

One-to-many – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship.



One-to-many



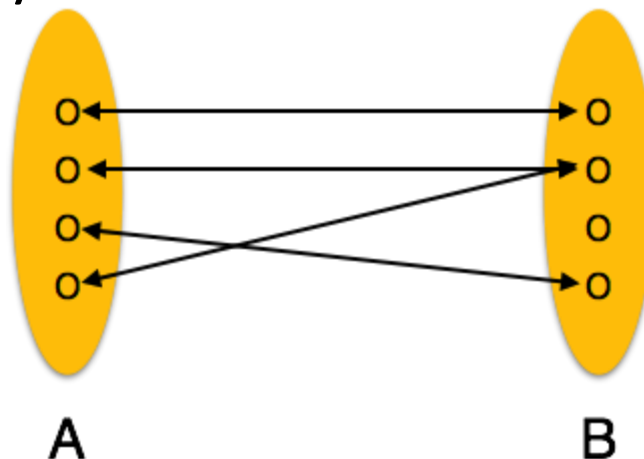
Each supplier can participate in multiple contracts. While a single contract might be assigned to a single supplier.

Каждый поставщик может быть обозначен во многих договорах. Однако один договор может быть связан только с одним поставщиком.

Многие-к-одному / Many-to-one

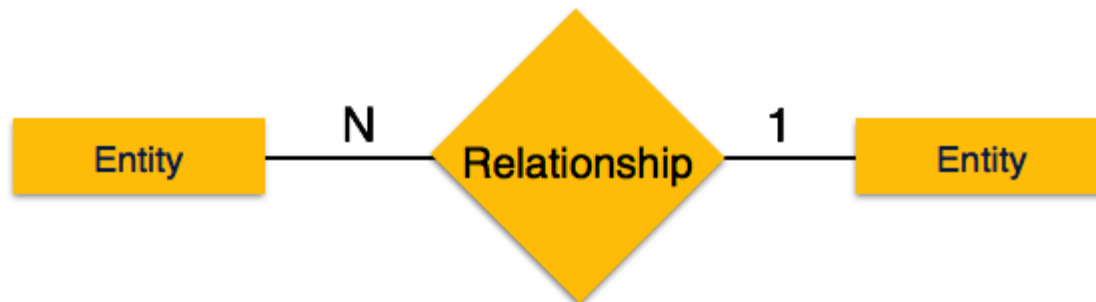
Более одной сущности из набора сущностей A могут быть ассоциированы не более чем из одной сущности набора сущностей B, однако сущность из набора сущностей B может быть связана с несколькими сущностями из набора сущностей A.

More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

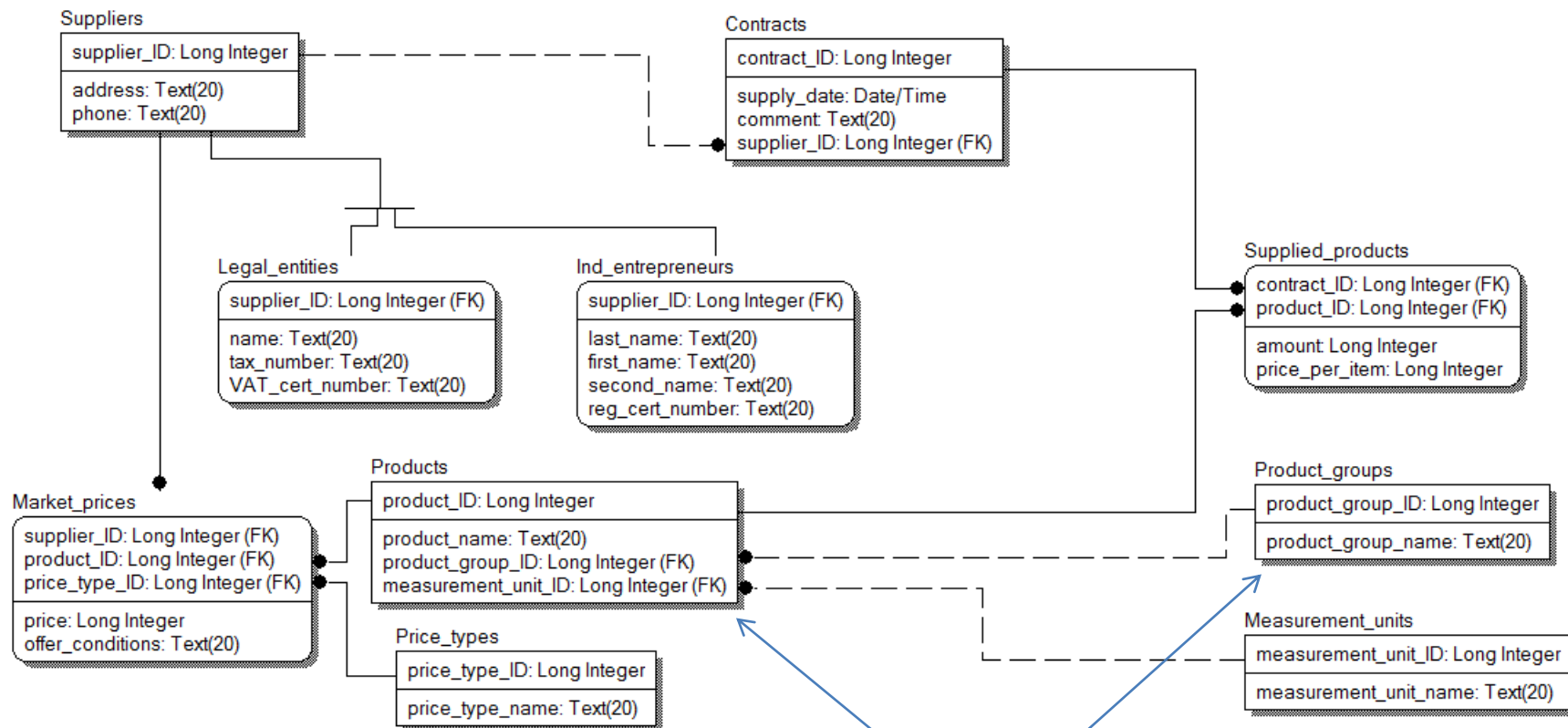


Многие-к-одному – если больше одного экземпляра сущности связано с отношением, оно обозначается как "N:1". Следующее изображение отражает, что больше чем один экземпляр сущности слева и только один экземпляр сущности справа может быть связан с этим отношением.

Many-to-one – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship.



Для многих товаров может быть определена одна продуктовая группа. Однако для одного товара может быть определена только одна продуктовая группа.
 For multiple products might be defined a single product category. But only one category might be defined for a single product.

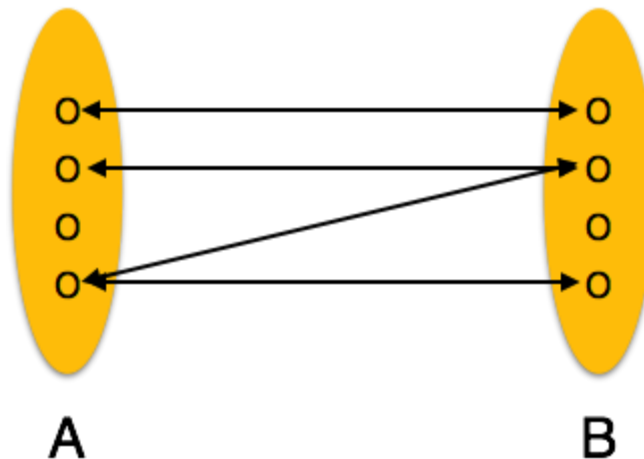


Many-to-one

Многие-ко-многим / Many-to-many

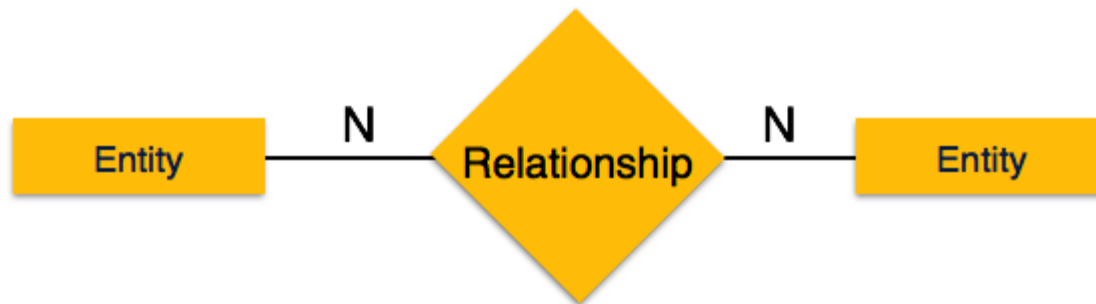
Одна сущность из A может быть связана с несколькими сущностями из B и наоборот.

One entity from A can be associated with more than one entity from B and vice versa.



Многие-ко-многим. Следующее изображение отражает, что больше чем один экземпляр сущности слева и более одного экземпляра сущности справа может быть связан с этим отношением.

Many-to-many – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



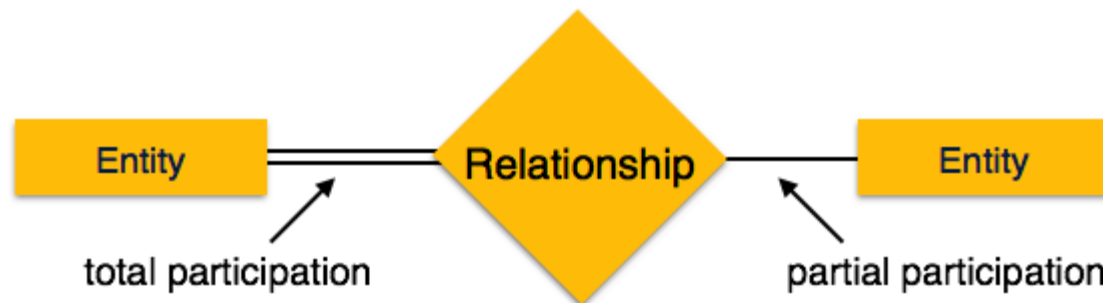
Ограничения участия / Participation constraints

Общее участие – каждая сущность участвует в отношения. Общее участие представлено двойными линиями.

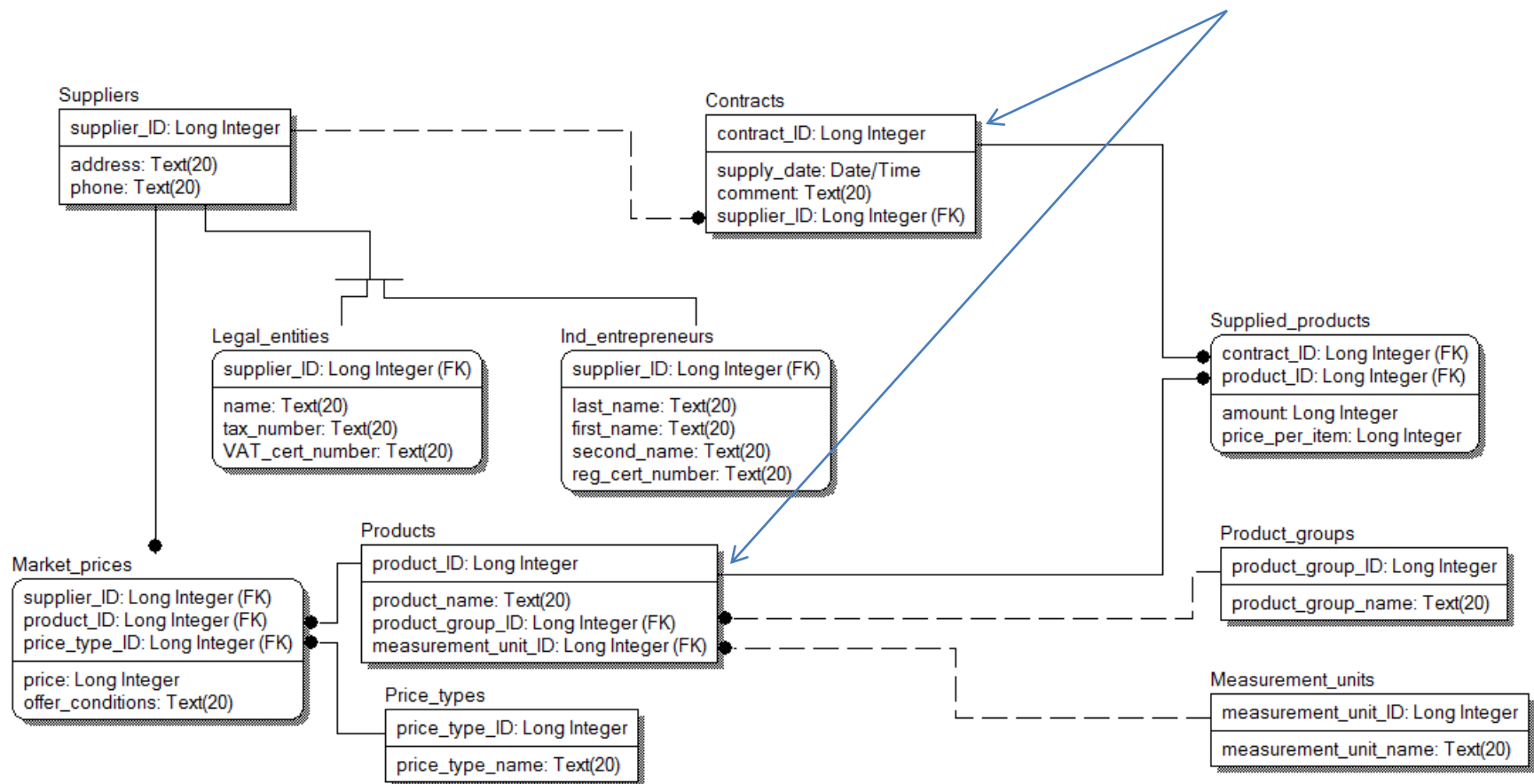
Total Participation – Each entity is involved in the relationship. Total participation is represented by double lines.

Частичное участие – в отношении участвуют не все сущности. Частичное участие представлено линиями.

Partial participation – Not all entities are involved in the relationship. Partial participation is represented by single lines.



Many-to-many



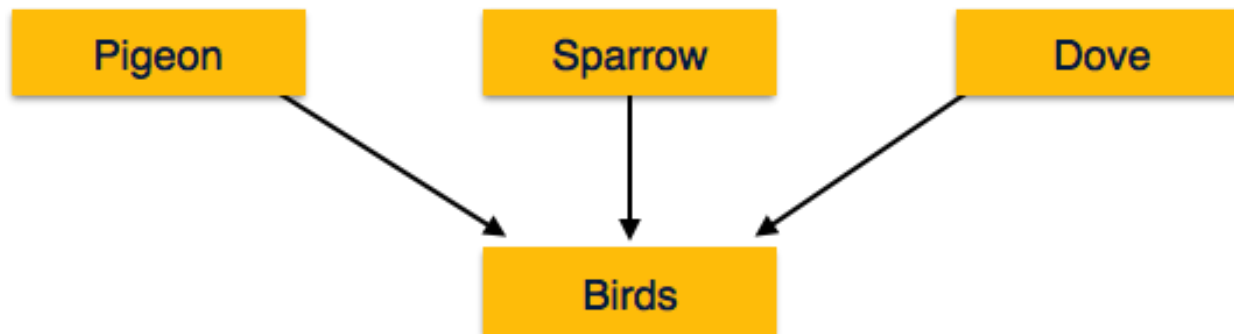
Each contract can be composed of multiple products. Each product can be assigned to multiple contracts.

Каждый договор может состоять из многих товаров. Каждый товар может быть указанный во многих договорах.

Обобщение / Generalization

В обобщении ряд сущностей объединяются в одну обобщенную сущность на основе их аналогичных характеристик.

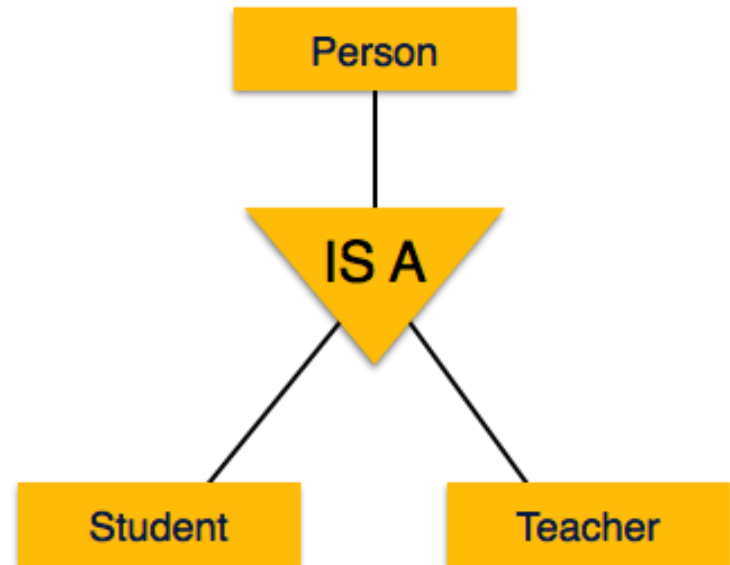
In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics.



Специализация / Specialization

Специализация – это противоположность обобщению. В специализации группа сущностей делится на подгруппы на основе их характеристик.

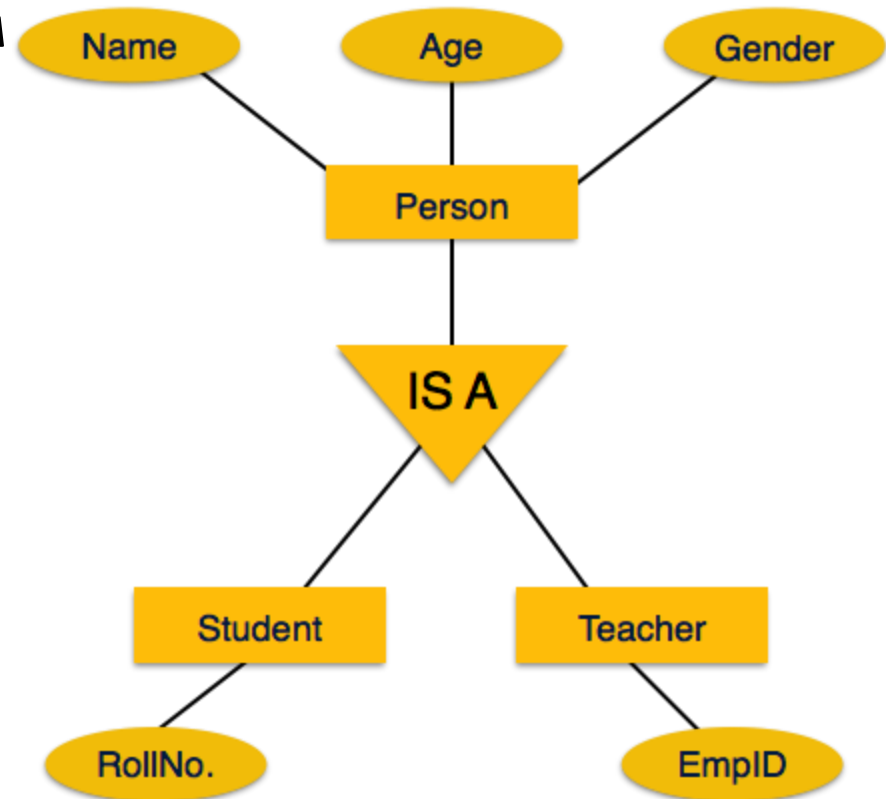
Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics.



Наследование / Inheritance

Наследование является важным признаком обобщения и специализации. Оно позволяет сущностям низшего уровня наследовать атрибуты сущностей высшего уровня.

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



Задание для тренировки 01 / Training task 01

Некоторое **предприятие** производит определенные товары. Для каждого **товара** указывается количество, цена и дата изготовления. В сведениях о предприятии указываются адрес и телефон. **Адрес** указывается с указанием области и города.

Some **factory** produces certain products. Each **product** is described by 'amount', 'price', and 'manufacturing date' properties. Information about factory includes 'address' and 'phone' properties. **Address** includes region and city.

Лекция 04: Нормализация базы данных

Lecture 04: Database normalization

Нормализация / Normalization

Если база данных ненормализованная, она может содержать **аномалии**, которые являются «ночным кошмаром» для каждого администратора баз данных. Управления базой данных, содержащей аномалии, почти невозможно.

If a database design is not perfect, it may contain ***anomalies***, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

Аномалии обновления / Modification anomalies

Если данные разбросаны и не связаны между собой определенным образом, это может привести к странным ситуациям. Например, когда мы пытаемся обновить один элемент данных, который имеет копии, разбросанные по нескольким местам, несколько экземпляров обновляются правильно, тогда как несколько других остаются со старыми значениями. Такие случаи оставляют базу данных в несогласованном состоянии.

If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

Student ID	Student Name	Student Phone	Assigned Subject
1	A	1234	Programming
2	A	1234	Databases
3	B	2345	Biology
4	C	3456	Math
5	D	4567	Physics

Если телефон студента А изменится, его необходимо обновить во всех соответствующих записях таблицы.

If student's A phone changes, must change it in each table record assigned to this student.

Аномалии удаления / Deletion anomalies

- В результате попытки удалить запись, некоторые его части не были удалены из-за незнания того, что данные также хранятся в другом месте.
- We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

Student ID	Student Name	Student Phone	Assigned Subject
1	A	1234	Programming
2	A	1234	Databases
3	B	2345	Biology
4	C	3456	Math
5	D	4567	Physics

Если удалить записи о биологии из базы данных, будут также потеряны все данные о студенте B.

If we delete Biology entries from the database, we also loose all data about student B.


Аномалии вставки / Insert anomalies

Попытка вставить данные в запись, которая вообще не существует.

We tried to insert data in a record that does not exist at all.

Student ID	Student Name	Student Phone	Assigned Subject
1	A	1234	Programming
2	A	1234	Databases
3	B	2345	Biology
4	C	3456	Math
5	D	4567	Physics

English



В базе данных невозможно сохранить данные о новом предмете, потому что он не будет связан ни с одним студентом.

It is impossible to store data about the new subject, since it won't be associated to any student.

Первая нормальная форма / First normal form

Это правило определяет, что все атрибуты в отношении должны иметь атомарные домены. Значения в атомарном домене являются неделимыми единицами. Каждый атрибут должен содержать только одно значение из его предварительно определенного домена.

This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units. Each attribute must contain only a single value from its pre-defined domain.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

← Non-atomic attributes



Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Вторая нормальная форма / Second normal form

- Для соблюдения второй нормальной формы, каждый не первичный атрибут должен полностью функционально зависеть от первичного ключа. То есть, если $X \rightarrow A$ справедливо, то не должно быть никакого подмножества Y множества X , для которого $Y \rightarrow A$ также справедливо.
- If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X , for which $Y \rightarrow A$ also holds true.

Функциональная зависимость / Functional Dependency

Функциональная зависимость (FD) – это совокупность ограничений между двумя атрибутами в отношении. Функциональная зависимость говорит, что если два кортежа имеют одинаковые значения для атрибутов A_1, A_2, \dots, A_n , то эти два кортежа должны иметь одинаковые значения для атрибутов B_1, B_2, \dots, B_n .

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A_1, A_2, \dots, A_n , then those two tuples must have to have same values for attributes B_1, B_2, \dots, B_n .

Функциональная зависимость представлена знаком стрелки (\rightarrow), то есть $X \rightarrow Y$, где X функционально определяет Y . Атрибуты левой части определяют значения атрибутов в правой части.

Functional dependency is represented by an arrow sign (\rightarrow) that is, $X \rightarrow Y$, where X functionally determines Y . The left-hand side attributes determine the values of attributes on the right-hand side.

- **Рефлексивность.** Если A – это набор атрибутов и B является подмножеством A , то A содержит B .
- **Reflexive rule.** If A is a set of attributes and B is subset of A , then A holds B .

- **Дополнение.** Если $A \rightarrow B$ выполняется, и Y является множеством атрибутов, то $AY \rightarrow BY$ также выполняется. Это добавление атрибутов к зависимостям, не меняет основных зависимостей.
- **Augmentation rule.** If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Транзитивность.** То же, что и транзитивное правило в алгебре, если выполняется $A \rightarrow B$, и $B \rightarrow C$, то $A \rightarrow C$ также имеет место. $A \rightarrow B$ функционально определяет B .
- **Transitivity rule.** Same as transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds. $A \rightarrow B$ is called as a functionally that determines b .

Student_Project

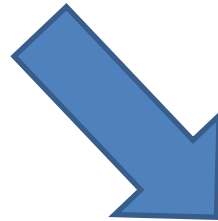
Stu_ID	Proj_ID	Stu_Name	Proj_Name
--------	---------	----------	-----------



Частичная зависимость Partial dependency

$A = \{Stu_Name, Proj_Name\}$
 $X = \{Stu_ID, Proj_ID\}$

$Stu_ID \rightarrow Stu_Name$
 $Proj_ID \rightarrow Proj_Name$



$A1 = \{Stu_Name, Proj_ID\}$
 $X1 = \{Stu_ID\}$
 $X1 \rightarrow A1$

$A2 = \{Proj_Name\}$
 $X2 = \{Proj_ID\}$
 $X2 \rightarrow A2$

Student

Stu_ID	Stu_Name	Proj_ID
--------	----------	---------

Project

Proj_ID	Proj_Name
---------	-----------

Третья нормальная форма / Third normal form

Отношение, которое должно быть в третьей нормальной форме, должно быть во второй нормальной форме, а также выполняться:

- Есть нетривиальных атрибутов, которые транзитивно зависят от первичного ключа;
- для любой нетривиальной функциональной зависимости $X \rightarrow A$, X должно быть супер-ключом или A должно быть первичным ключом.

For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy:

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, $X \rightarrow A$, then either X is a super key or, A is prime attribute.

Тривиальная функциональная зависимость / Trivial Functional Dependency

- Если выполняется функциональная зависимость $X \rightarrow Y$, где Y – подмножество X , то она называется **тривиальной**.
- If a functional dependency $X \rightarrow Y$ holds, where Y is a subset of X , then it is called a **trivial**.
- Если выполняется $X \rightarrow Y$, где Y не является подмножеством X , то функциональную зависимость называют **нетривиальной**.
- If an functional dependency $X \rightarrow Y$ holds, where Y is not a subset of X , then it is called a **non-trivial**.
- Если выполняется $X \rightarrow Y$, где пересечение X и $Y = \Phi$, то функциональная зависимость считается **абсолютно нетривиальной**.
- If an functional dependency $X \rightarrow Y$ holds, where x intersect $Y = \Phi$, it is said to be a **completely non-trivial**.

Student_Detail

Stu_ID	Stu_Name	City	Zip
--------	----------	------	-----

Транзитивная зависимость
Transitive dependency

Zip -> City ???

Zip IS NOT A SUPER KEY

City IS NOT A PRIMARY KEY

Stu_ID -> Zip -> City



Student_Detail

Stu_ID	Stu_Name	Zip
--------	----------	-----

Stu_ID -> Stu_Name, Zip

ZipCodes

Zip	City
-----	------

Zip -> City

Super Key

Нормальная форма Бойса-Кодда / Boyce-Codd Normal Form

Нормальная форма Бойса-Кодда (BCNF) является продолжением третьей нормальной формы на более строгих условиях. BCNF утверждает, что для любой нетривиальной функциональной зависимости, $X \rightarrow A$, X должно быть супер-ключом.

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that for any non-trivial functional dependency, $X \rightarrow A$, X must be a super-key.

Задание для тренировки 02 / Training task 02

Name	Address	Product	Amount	Price	Date
ABC Group	Lozova, Kharkiv Oblast	TV	5000	5999	10.10.2018
XYZ Corp	Sumy, Sumska Oblast	Laptop	2000	8999	11.10.2018
ABC Group	Lozova, Kharkiv Oblast	Laptop	3000	8499	10.10.2018
XYZ Corp	Sumy, Sumska Oblast	Phone	1000	2999	12.10.2018

Привести отношение к 3 НФ / Bring the relation into 3 NF

Декомпозиция схемы отношения / Relation scheme decomposition

Декомпозицией схемы отношения $R = \{A_1, A_2, \dots, A_n\}$ называется замена ее совокупностью подмножеств R , таких что их объединение дает R . При этом допускается, что подмножества могут пересекаться.

Relation $R = \{A_1, A_2, \dots, A_n\}$ scheme decomposition is a procedure of replacement of R with the set of sub-relations which union gives R . Moreover, the intersection of such sub-sets is allowed.

Input: $R_0 = \{\text{Name, Address, Product, Amount, Price, Date}\}$

Address
Lozova, Kharkiv Oblast
Sumy, Sumska Oblast

← **PROBLEM**

Address is non-atomic attribute $\Rightarrow R$ is not in 1NF

SOLUTION: Address $\Rightarrow \{\text{City, Region}\}$

Output: $R_{1NF} = \{\text{Name, City, Region, Product, Amount, Price, Date}\}$

Input: $R_{1NF} = \{\text{Name, City, Region, Product, Amount, Price, Date}\}$

Name $\rightarrow \{\text{City, Region}\}$

Name $\rightarrow \{\text{Amount, Price, Date}\}$

Product $\rightarrow \{\text{Amount, Price, Date}\}$

PROBLEM: R_{1NF} has partial dependencies $\Rightarrow R_{1NF}$ is not in 2NF

SOLUTION: Split R_{1NF} into several relations

Output: $R_{2NF} = R_{\text{Factory}} \text{ AND } R_{\text{Product}} \text{ AND } R_{\text{PF}}$
 $R_{\text{Factory}} = \{\text{Factory_ID, Name, City, Region}\}$
 $R_{\text{Product}} = \{\text{Product_ID, Product}\}$
 $R_{\text{Manufactured}} = \{\text{Factory_ID, Product_ID, Amount, Price, Date}\}$

Factory	
Factory_ID	PK
Name City Region	

Products	
Product_ID	PK
Product	

Manufactured	
Factory_ID Product_ID	PK
Amount Price Date	

Relations in 2NF

Input: $R_{2NF} = R_{\text{Factory}} \text{ AND } R_{\text{Product}} \text{ AND } R_{\text{Manufactured}}$

Product_ID \rightarrow Product

{Factory_ID, Product_ID} \rightarrow {Amount, Price, Date}

Factory_ID \rightarrow {Name, City, Region}, City \rightarrow Region

PROBLEM: R_{2NF} has transitive dependency Factory_ID \rightarrow City \rightarrow Region $\Rightarrow R_{2NF}$ is not in 3NF

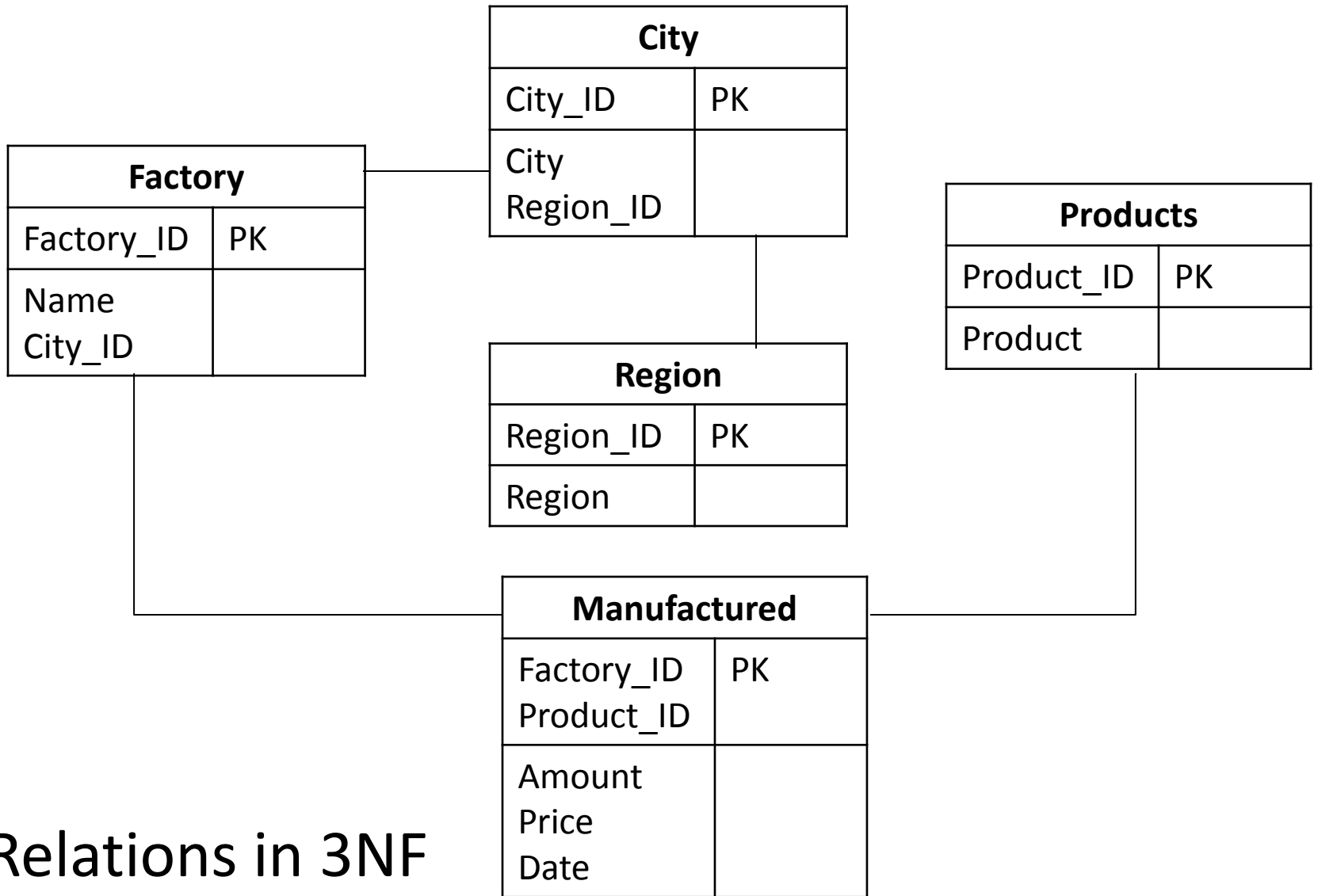
SOLUTION: Split R_{Factory} into several relations

Output: $R_{3NF} = R_{2NF} \text{ AND } R_{\text{City}} \text{ AND } R_{\text{Region}}$

$R_{\text{Factory}} = \{\text{Factory_ID}, \text{City_ID}\}$

$R_{\text{City}} = \{\text{City_ID}, \text{City}, \text{Region_ID}\}$

$R_{\text{Region}} = \{\text{Region_ID}, \text{Region}\}$



Relations in 3NF

Четвертая нормальная форма / Fourth normal form

Отношение находится в четвертой нормальной форме, если оно находится в нормальной форме Бойса-Кодда и не содержит нетривиальных **многозначных зависимостей**.

A relation is in the fourth normal form if it is already in the Boyce-Codd normal form and does not contain any non-trivial **multivalued dependencies**.

Пятая нормальная форма / Fifth normal form

Отношение находится в пятой нормальной форме тогда и только тогда, когда каждая нетривиальная **зависимость соединения** в ней определяется потенциальным ключом (ключами) этого отношения.

A relation is in the fifth normal form if and only if every non-trivial **join dependency** in that relation is implied by the candidate keys.

Доменно-ключевая нормальная форма / Domain-key normal form

Отношение находится в доменно-ключевой нормальной форме тогда и только тогда, когда каждое наложенное на нее ограничение является логическим следствием **ограничений доменов** и **ограничений ключей**, наложенных на это отношение.

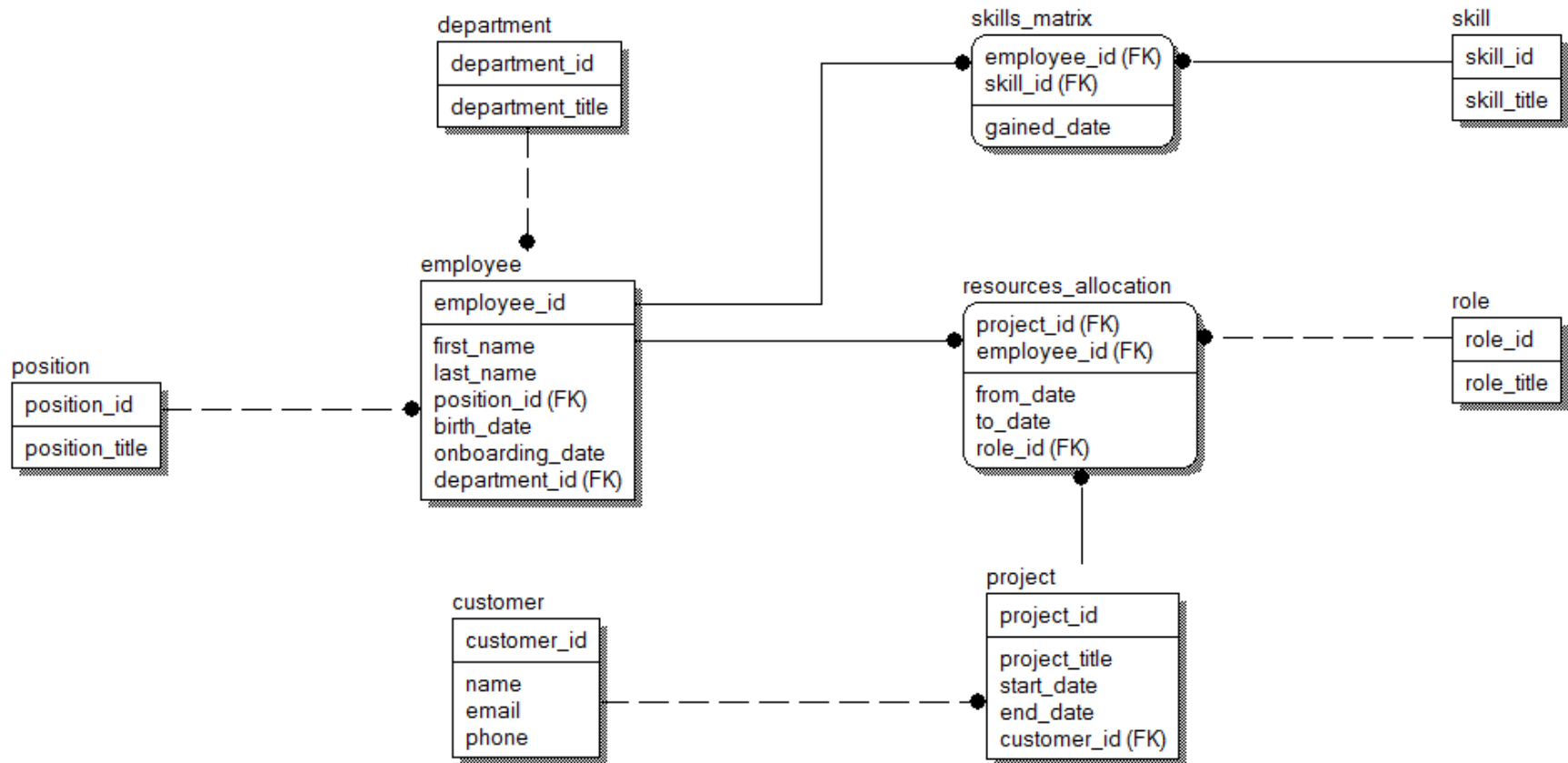
A relation is in the domain-key normal form if and only if it contains no constraints other than **domain constraints** and **key constraints**.

Шестая нормальная форма / Sixth normal form

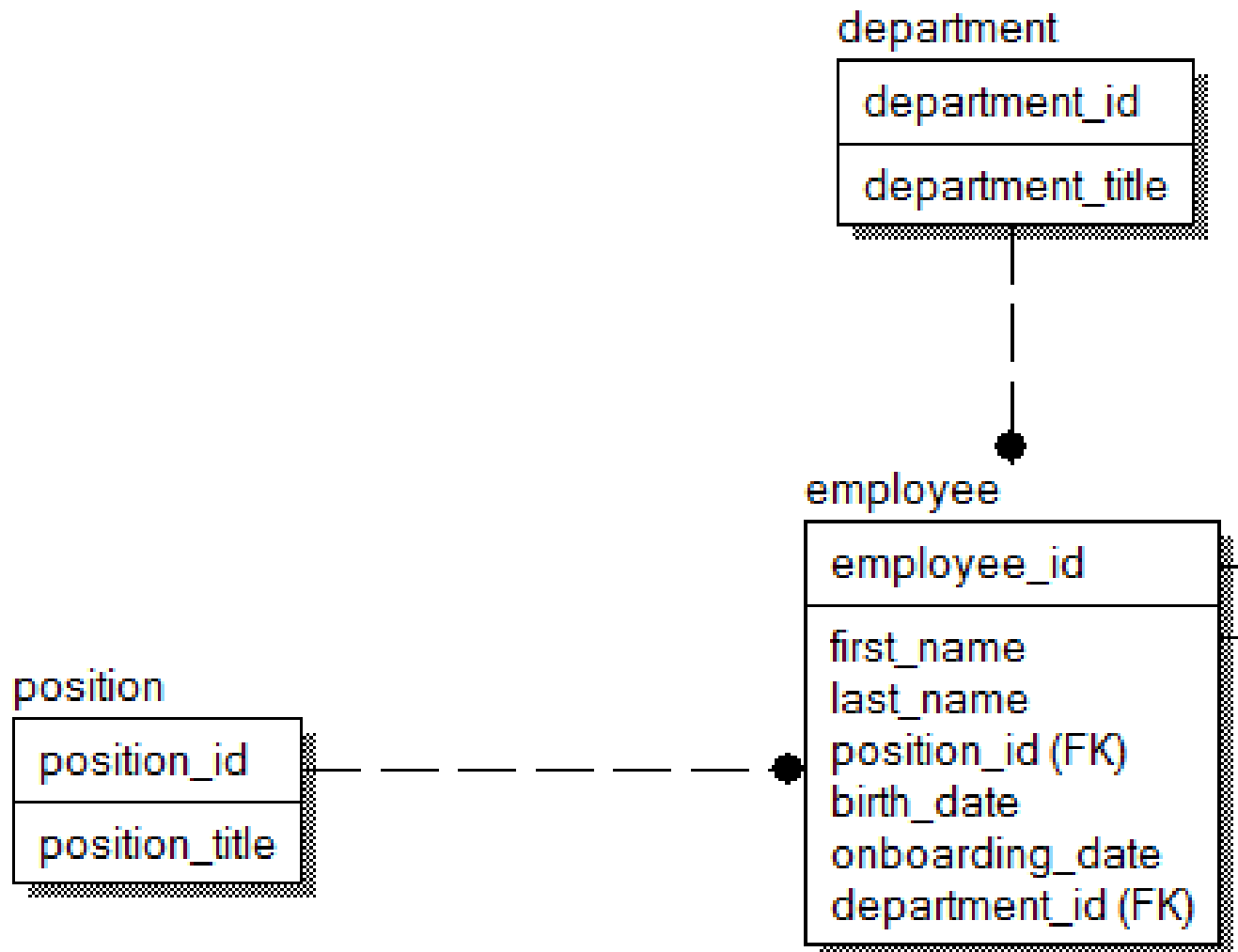
Отношение находится в шестой нормальной форме тогда и только тогда, когда оно удовлетворяет **всем нетривиальным зависимостям соединения** (то есть, не может быть подвергнута дальнейшей декомпозиции без потерь).

A relation is in the sixth normal form if and only if it satisfies **all non-trivial join dependencies** (that is it cannot be further decomposed without loss).

Предметная область для контроля: IT компания / Subject area for tests: IT company



Сотрудники / Employees

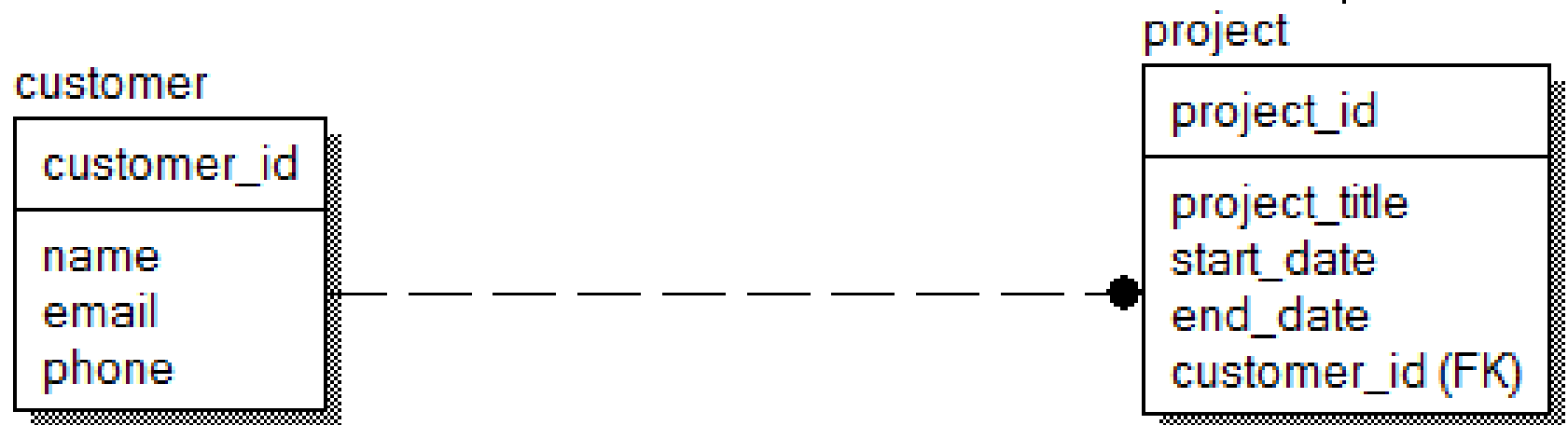


employee							
	employee_i ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾	department ▾
+		1 Jim	Halpert	2	5/12/1990	3/2/2014	1
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3
+	5	Michael	Scott	5	12/9/1989	4/28/2014	4

position		
	position_id: ▾	position_tit ▾
+		1 QA Eng.
+	2	SW Eng.
+	3	Sr. SW Eng.
+	4	Syst. Eng.
+	5	PM

department		
	department ▾	department_title: ▾
+	1	Dev. Dept.
+	2	QA Dept.
+	3	Maintenance Dept.
+	4	Management Dept.
+	5	Accounting Dept.
+	6	HR Dept.

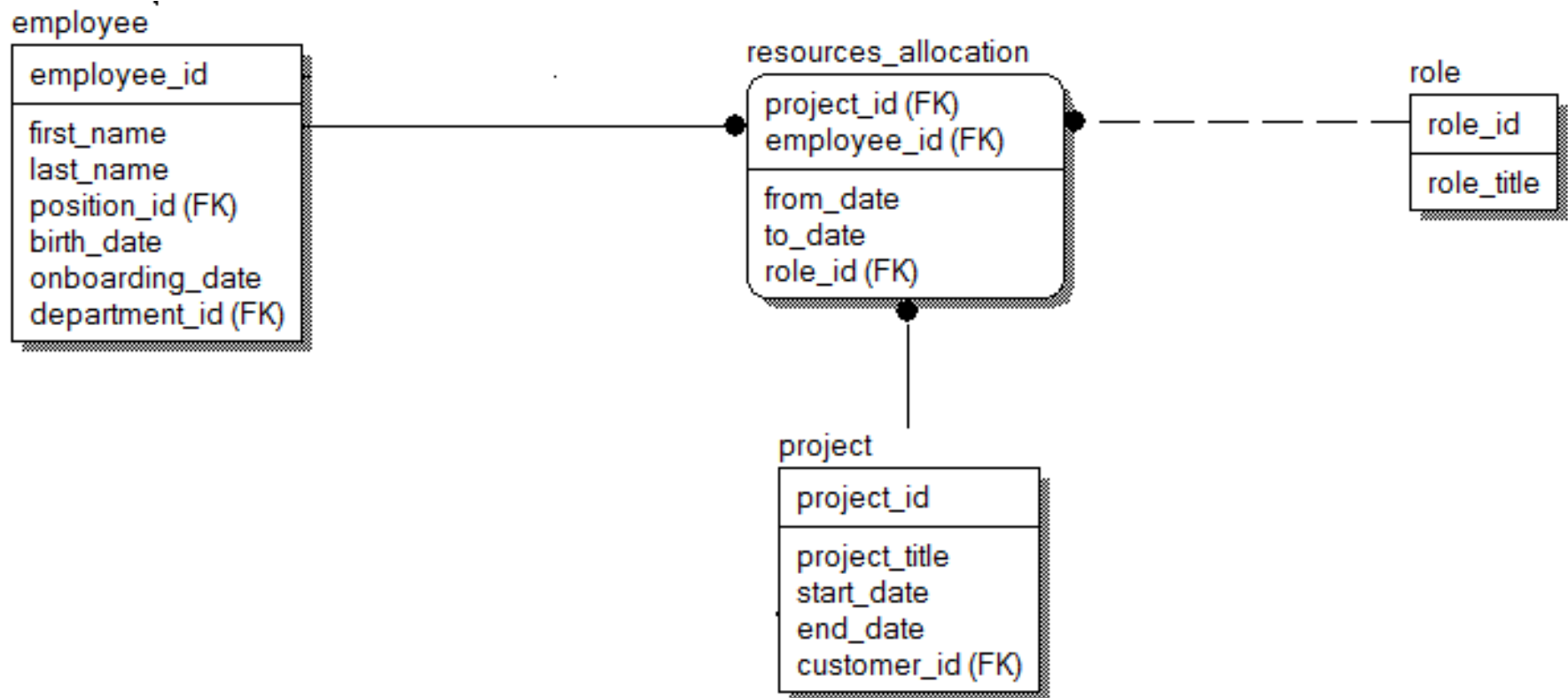
Проекты / Projects



project					
	project_id: ▾	project_title ▾	start_date: ▾	customer_id ▾	end_date: ▾
+	1	MEB-ATM	2/14/2016	1	12/19/2016
+	2	MEB-INS	10/17/2016	1	4/22/2017
+	3	NAE-PRT	3/3/2016	2	5/20/2016
+	4	ICN-SRV	5/11/2017	3	12/31/2018
+	5	LKH-ERP	12/5/2017	4	2/20/2018

customer				
	customer_id: ▾	name: ▾	email: ▾	phone: ▾
+	1	ME Bank	me@bank.com	+442938457601
+	2	North AE	atomnorth@energy.com	+17356745692
+	3	IrishCons LLC	egi@consult.com	+433569184822
+	4	Lucky Holding	itdept@lucky.com	+410223415617
+	5	Southoil	contact@soil.com	+13120023475

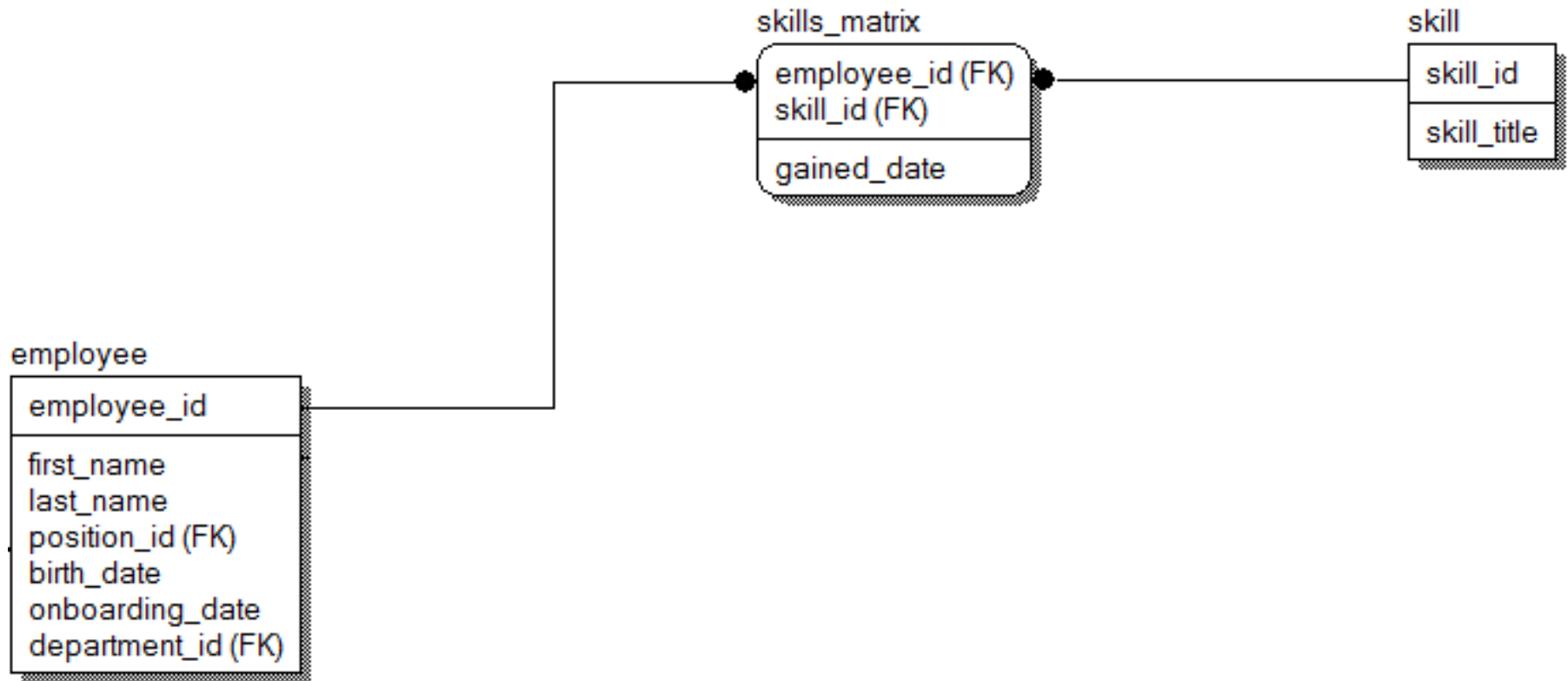
Отношение сотрудники-проекты / Employees-projects relation



resources_allocation					
project_id: ▾	employee_id: ▾	from_date: ▾	to_date: ▾	role_id: ▾	
1	1	2/28/2016	12/18/2016	1	
1	2	4/20/2016	12/18/2016	3	
1	3	2/25/2016	12/15/2016	2	
1	4	4/20/2016	12/19/2016	4	
1	5	2/14/2016	12/19/2016	5	
2	5	10/17/2016	4/22/2017	5	
4	3	5/15/2017	12/31/2018	4	
4	5	5/11/2017	12/31/2018	5	
5	4	12/5/2017	2/20/2018	4	

role		
	role_id: ▾	role_title: ▾
+	1	Developer
+	2	Team Lead
+	3	Tester
+	4	Support Specialist
+	5	Manager

Отношение сотрудники-навыки / Employees-skills relation



skills_matrix			
employee_id: ▾	skill_id: ▾	gained_date: ▾	
1	1	5/30/2012	
1	2	5/30/2012	
1	5	4/12/2013	
1	6	5/30/2012	
3	1	8/10/2011	
3	2	8/10/2011	
3	4	12/5/2011	
3	5	1/20/2012	
3	6	12/5/2011	
5	7	5/21/2008	

skill		
	skill_id: ▾	skill_title: ▾
+	1	Java SE
+	2	Maven
+	3	Java EE
+	4	Servlets
+	5	JSP
+	6	Git
+	7	UML

Positions

Position_ID (PK)

Position_Title

Хранить данные о навыках,
необходимые для
замещения определенной
должности, указывая
уровень каждого навыка

Skills

Skill_ID (PK)

Skill_Title

Store data about skills
required to obtain a certain
position with considering
a level of a certain skill

Positions
Position_ID (PK)
Position_Title

Required_Skills
Position_ID (PK)
Skill_ID (PK)
Level



Skills
Skill_ID (PK)
Skill_Title

IS THIS SOLUTION
PERFECT?

Relation

Required_Skills

Position_ID (PK)

Skill_ID (PK)

Level



Table in DB

Position_ID	Skill_ID	Level
DEV	JSE	Intermediate
DEV	JSP	Elementary
...
MT	DLC	Advanced

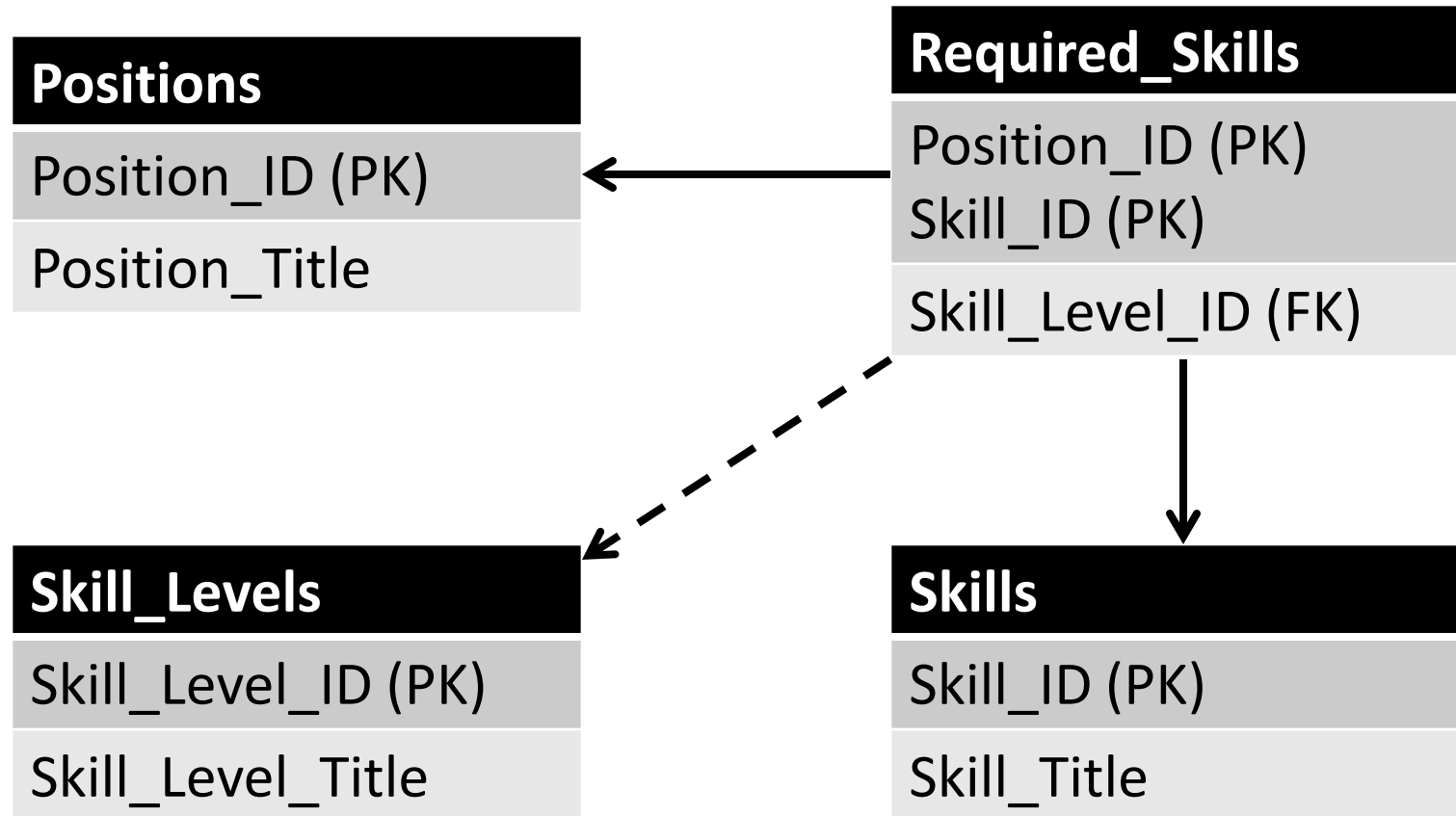
Position_ID	Skill_ID	Level
DEV	JSE	Intermediate
DEV	JSP	Elementary
...
MT	DLC	Advanced



Modification anomalies

What if we need to:

- Update levels ?
- Insert levels ?
- Delete levels ?



Лекция 05: Реляционная модель данных

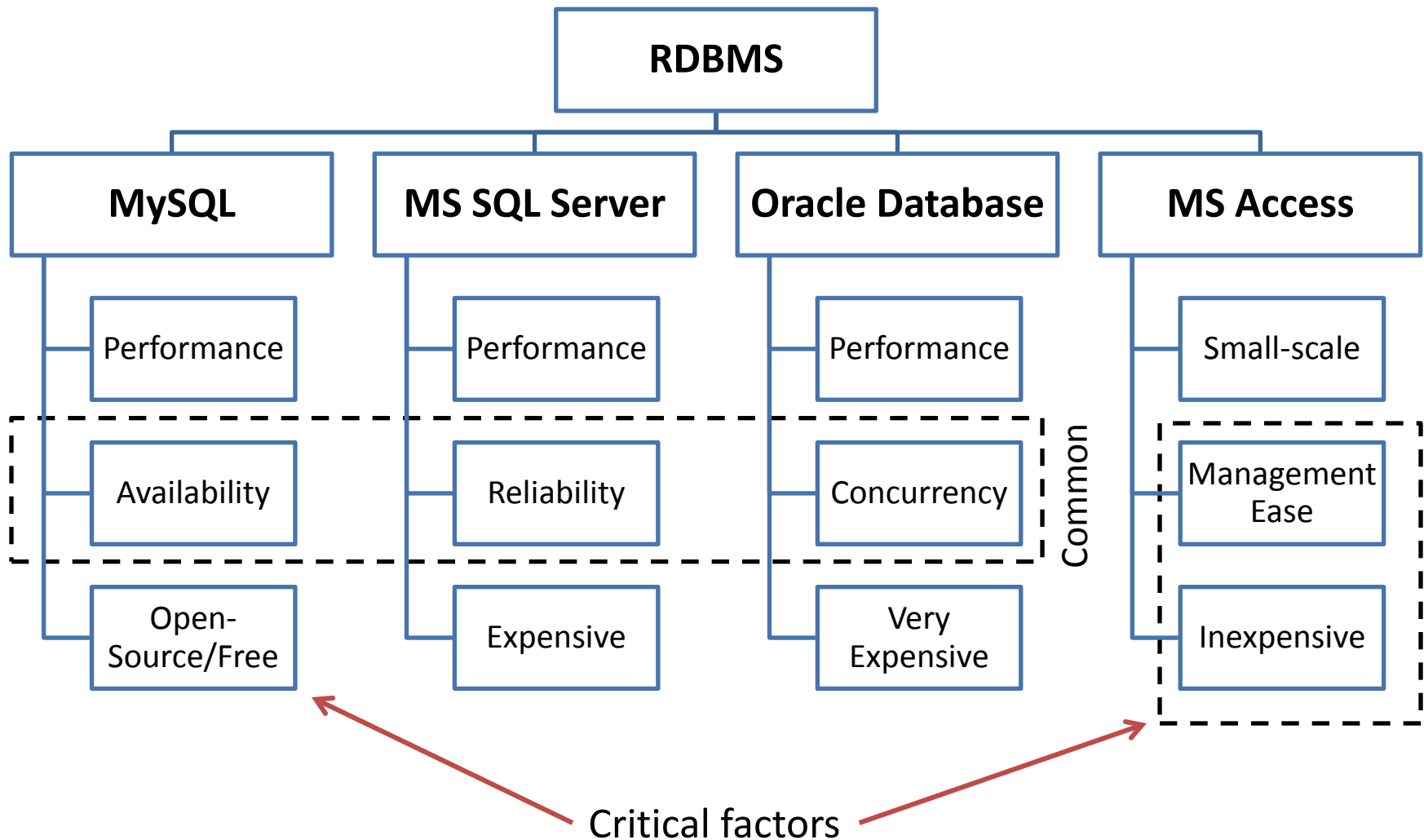
Lecture 05: Relational data model

Реляционная модель данных / Relational data model

Реляционная модель данных является основной моделью данных, которая широко используется во всем мире для хранения и обработки данных. Эта модель проста и имеет все свойства и возможности, необходимые для обработки данных с эффективностью хранения.

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Реляционные СУБД / Relational DBMS



MySQL



MySQL – это база данных с открытым исходным кодом, разработанная шведской компанией MySQL AB.

MySQL поддерживает много различных платформ, включая Microsoft Windows, основные дистрибутивы Linux, UNIX и Mac OS.

MySQL имеет бесплатные и платные версии в зависимости от его использования (не коммерческие или коммерческие) и функций.

MySQL поставляется с очень быстрым многопоточным, многопользовательским и надежным сервером баз данных SQL.

MySQL is an open source SQL database, which is developed by a Swedish company – MySQL AB.

MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS.

MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user and robust SQL database server.

MS SQL Server



MS SQL Server – это система управления реляционными базами данных, разработанная компанией Microsoft Inc. Кроме высокой производительности и доступности, предоставляет возможности бизнес-аналитики (SQL Server Analytics Services) и облачных вычислений (Azure SQL Database).

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Besides high performance and availability it provides Business Intelligence (SQL Server Analytics Services) and Cloud Computing features (Azure SQL Database).

Oracle Database



Это очень большая многопользовательская система управления базами данных. Oracle Database – это реляционная система управления базами данных, разработанная корпорацией Oracle.

Это отличный выбор сервера баз данных для клиентских или серверных вычислений. Oracle Database поддерживает все основные операционные системы для клиентов и серверов.

It is a very large multi-user based database management system. Oracle Database is a relational database management system developed by 'Oracle Corporation'.

It is an excellent database server choice for client/server computing. Oracle Database supports all major operating systems for both clients and servers.

MS Access



Это один из самых популярных продуктов Microsoft. Microsoft Access – это СУБД начального уровня. База данных MS Access не только недорогое, но мощное решение для малых проектов.

MS Access использует определенный диалект языка SQL (иногда называется Jet SQL).

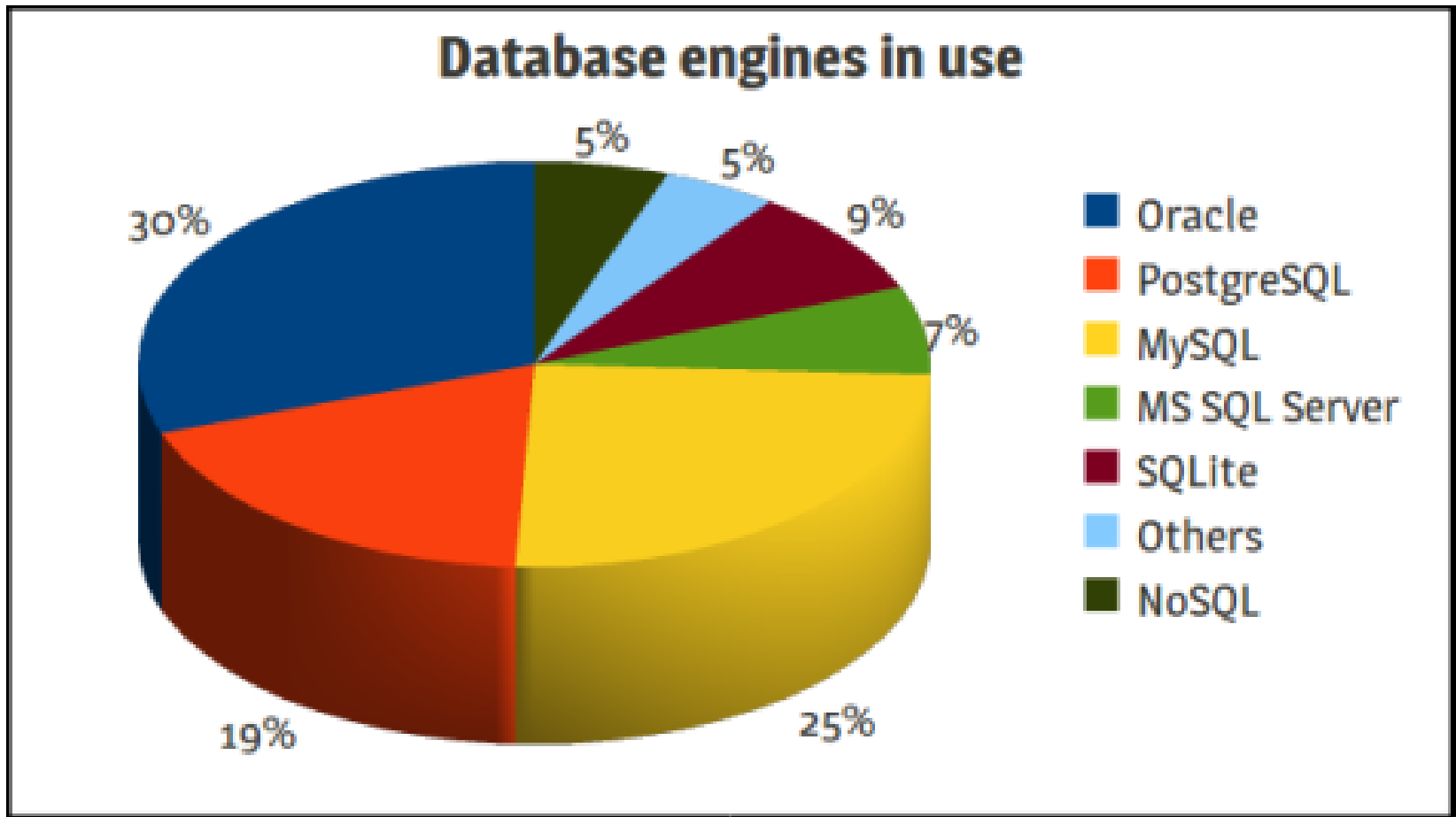
MS Access поставляется с профессиональным изданием пакета MS Office. MS Access имеет простой в использовании интуитивно понятный графический интерфейс.

This is one of the most popular Microsoft products. Microsoft Access is an entry-level database management software. MS Access database is not only inexpensive but also a powerful database for small-scale projects.

MS Access uses a specific SQL language dialect (sometimes referred to as Jet SQL).

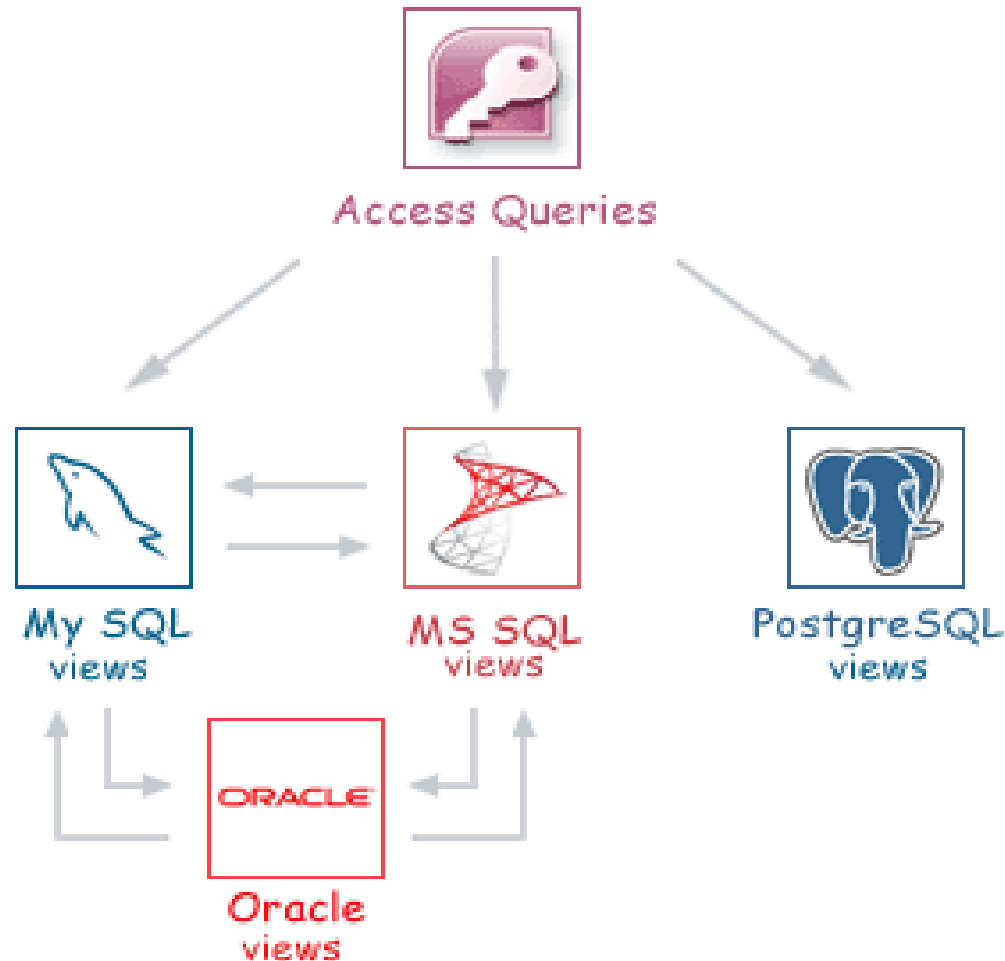
MS Access comes with the professional edition of MS Office package. MS Access has easy-to-use intuitive graphical interface.

RDBMS Market Share 2018



<https://terraencounters.wordpress.com/2018/02/02/learning-relational-databases/>

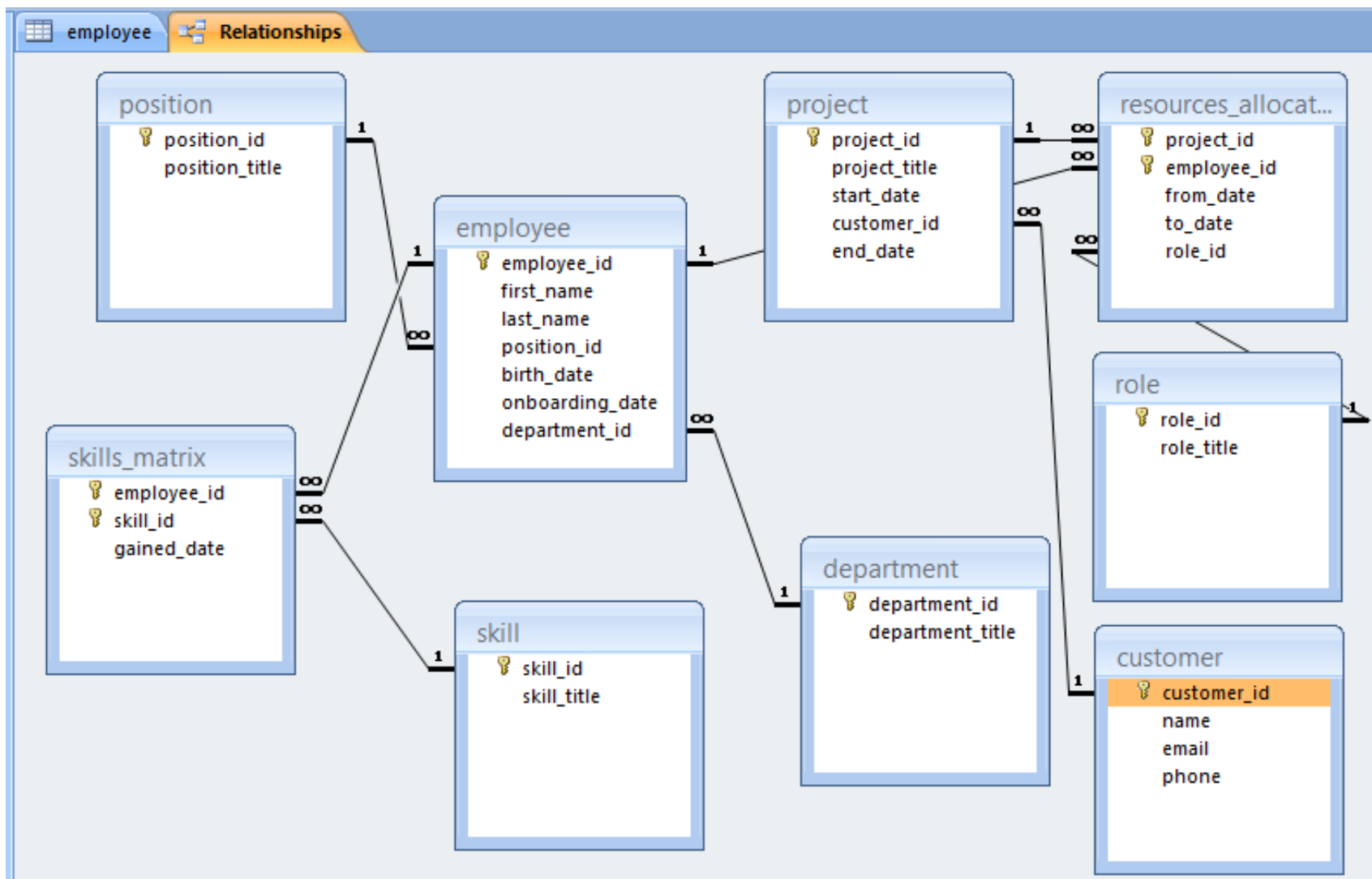
Подобие синтаксиса SQL / Similarity of SQL syntax



Основные понятия / Concepts

- **Таблицы** – в реляционной модели данных отношение хранятся в формате таблиц. Этот формат сохраняет отношение между субъектами. В таблице есть строки и столбцы, где строки представляют записи, а столбцы – атрибуты.
- **Tables** – In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Таблицы / Tables



- **Кортеж** – отдельная строка таблицы, содержащая единственную запись для этого отношения, называется кортежем.
- **Tuple** – A single row of a table, which contains a single record for that relation is called a tuple.
- **Экземпляр отношения** – конечный набор кортежей в реляционной базе данных представляет экземпляр отношения. Экземпляры отношения не имеют дубликатов кортежей.
- **Relation instance** – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Кортежи и экземпляр отношения / Tuple and relation instance

employee							
employee_i	first_name:	last_name:	position_id:	birth_date:	onboarding:	department	
1	Jim	Halpert	2	5/12/1990	3/2/2014	1	
2	Pamela	Beesly	1	11/2/1992	9/23/2015	2	
3	Dwight	Schrute	3	2/11/1991	12/3/2013	1	
4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3	
5	Michael	Scott	5	12/9/1989	4/28/2014	4	

← Tuple

Relation instance

- **Схема отношения** – схема отношения описывает имя отношения (название таблицы), атрибуты и их имена.
- **Relation schema** – A relation schema describes the relation name (table name), attributes, and their names.
- **Ключ отношения** – каждая строка имеет один или несколько атрибутов, известных как ключ отношения, который может однозначно идентифицировать строку в отношении (таблицы).
- **Relation key** – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Схема и ключ отношения / Relation schema and key

Key →

employee		
	Field Name	Data Type
Key	employee_id	Number
	first_name	Text
	last_name	Text
	position_id	Number
	birth_date	Date/Time
	onboarding_date	Date/Time
	department_id	Number

Relation schema ↗

- **Область определения атрибутов** – каждый атрибут имеет определенную заранее определенную область определения, известную как домен атрибута.
- **Attribute domain** – Every attribute has some pre-defined value scope, known as attribute domain.

Правила Кодда / Codd's rules

Д-р **Эдгар Кодд**, после длительных исследований реляционной модели систем баз данных, выдвинул двенадцать собственных правил, которые, по его мнению, должна выполнять база данных для того, чтобы считаться настоящей реляционной базой данных.

Dr **Edgar Codd**, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

Эти правила могут быть применены к любой системы базы данных, которая управляет сохраненными данными, используя только свои реляционные возможности. Это основное правило, которое является основой для всех других правил.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Правило 1: Информационное правило / Rule 1: Information

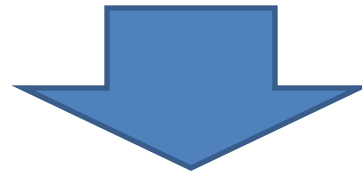
Данные, хранящиеся в базе данных, которые могут быть пользовательскими данными или метаданными, должны быть значением некоторой ячейки таблицы. Все в базе данных должно храниться в формате таблицы.

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Console

Press Ctrl+Enter to execute query

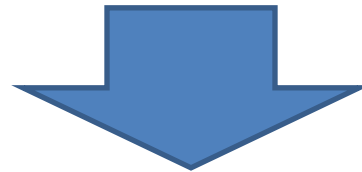
```
> SHOW DATABASES
```



Database
delivery
information_schema
mysql
performance_schema
phpmyadmin
test

```
> SHOW TABLES
```

```
>
```



Tables_in_delivery

contract

legal_supplier

private_supplier

product

supplier

Правило 2: Гарантированный доступ к данным / Rule 2: Guaranteed Access

Для каждого элемента данных (значения) гарантируется логический доступ по комбинации имени таблицы, первичного ключа (значение строки) и имени атрибута (значение столбца). Другие средства, такие как указатели, нельзя использовать для доступа к данным.

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

```
USE delivery;  
SELECT supplier.name, supplier.address  
FROM supplier  
WHERE supplier.id = 3
```



name

address

"Interfrut" LLC Kyiv, Peremohy Av., 154, office 3

Правило 3: Систематическая обработка NULL / Rule 3: Systematic Treatment of NULL Values

Отношение к значениям NULL в базе данных должно быть систематическим и однородным. Это очень важное правило, поскольку значение NULL может быть интерпретировано как: данные отсутствуют, данные неизвестны или данные не могут быть применимы.

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Правило 4: Активный онлайн-каталог

/ Rule 4: Active Online Catalog

Описание структуры всей базы данных должно храниться в онлайн-каталоге, известном как словарь данных, к которому могут получить доступ авторизованные пользователи. Пользователи могут использовать тот же язык запросов для доступа к каталогу, который они используют для доступа к самой базе данных.

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

SQL query language statements

Your SQL query has been executed successfully.

SHOW DATABASES

+ Options

Database

delivery

information_schema

mysql

performance_schema

phpmyadmin

test

Your SQL query has been executed successfully.

SHOW TABLES

+ Options

Tables_in_delivery

contract

legal_supplier

private_supplier

product

supplier

Правило 5: Полнота подмножества языка /

Rule 5: Comprehensive Data Sub-Language

Доступ к БД можно получить только с помощью языка, который имеет линейный синтаксис, поддерживает определение данных, манипулирования данными и операции по управлению транзакциями. Этот язык можно использовать непосредственно или с помощью определенной программы. Если база данных позволяет получить доступ к данным без помощи этого языка, то это считается нарушением.

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

WHEN?

name	address
Ivanov I. I. PE	Kharkiv, Pushkinska Str., 77 (tel. 33-33-44, 12-...
"Interfrut" LLC	Kyiv, Peremohy Av., 154, office 3
Petrov P. P. PE	Kharkiv, Nauky Av., 55, office 108, tel. 32-18-44
"Transservis" LLC	Odesa, Deribasivska Str., 75
Sidorov S. S. PE	Poltava, Svobody Str., 15, apt. 43
John Doe	Kharkiv, Kyrpychova st., 2, 61002

```
START TRANSACTION;
```

```
INSERT INTO supplier (name, address)
VALUES ('John Doe', 'Kharkiv, Kyrpychova st., 2, 61002');
```

```
SELECT name, address FROM supplier;
```

```
ROLLBACK;
```

```
SELECT name, address FROM supplier;
```

WHEN?

name	address
Ivanov I. I. PE	Kharkiv, Pushkinska Str., 77 (tel. 33-33-44, 12-...
"Interfrut" LLC	Kyiv, Peremohy Av., 154, office 3
Petrov P. P. PE	Kharkiv, Nauky Av., 55, office 108, tel. 32-18-44
"Transservis" LLC	Odesa, Deribasivska Str., 75
Sidorov S. S. PE	Poltava, Svobody Str., 15, apt. 43

Правило 6: Обновление представлений / Rule 6: View Updating

Все представления базы данных, которые теоретически могут быть обновлены, также должны быть пригодны к обновлению системы.

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Правило 7: Вставка, обновление и удаление высокого уровня / Rule 7: High-Level Insert, Update, and Delete

База данных должна поддерживать вставку, обновления и удаления высокого уровня. Это не должно ограничиваться одной строкой, то есть она также должна поддерживать операции объединения, пересечения и дополнения для получения множества записей данных.

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

```
SELECT * FROM supplier WHERE id < 3  
UNION  
SELECT * FROM supplier WHERE id > 4
```

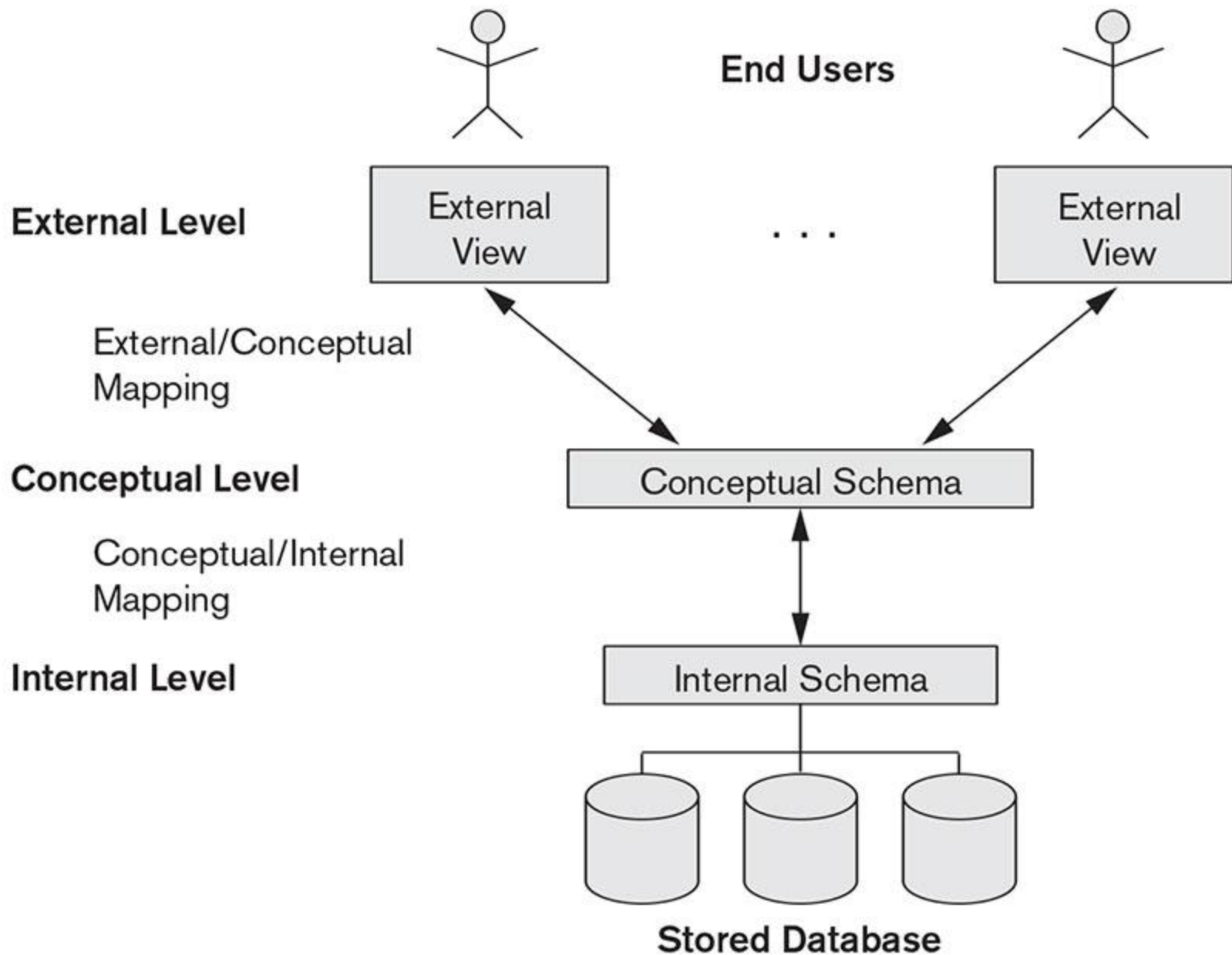


id	name	address
2	Ivanov I. I. PE	Kharkiv, Pushkinska Str., 77 (tel. 33-33-44, 12-...
5	"Transservis" LLC	Odesa, Deribasivska Str., 75
6	Sidorov S. S. PE	Poltava, Svobody Str., 15, apt. 43

Правило 8: Физическая независимость данных / Rule 8: Physical Data Independence

Данные, хранящиеся в базе данных, должны быть независимыми от программ, которые имеют доступ к базе данных. Любые изменения в физической структуре базы данных не должны иметь никакого влияния на то, как внешние программы получают доступ к данным.

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

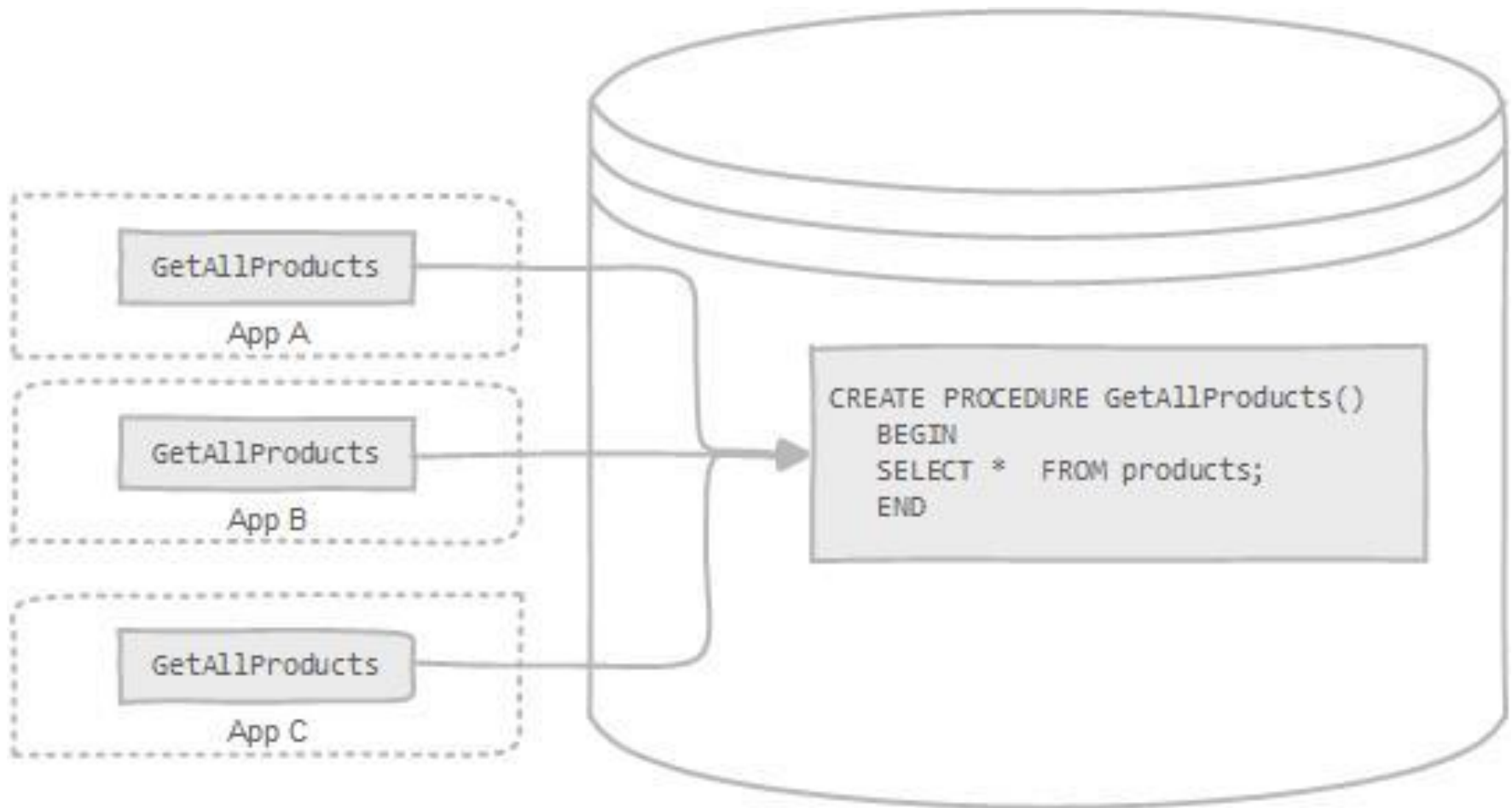


Правило 9: Логическая независимость данных

/ Rule 9: Logical Data Independence

Логические данные в базе данных должны быть независимыми от просмотра пользователя (программы). Любое изменение логических данных не должна влиять на программы, которые используют. Например, если две таблицы объединяются или одна делится на две разные таблицы, пользовательские приложения не должны подвергаться воздействию или изменений. Это одно из самых тяжелых правил выполнения.

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.



Правило 10: Независимость контроля целостности / Rule 10: Integrity Independence

База данных должна быть независимой от программы, которая ее использует. Все ее ограничения целостности могут быть независимо модифицированы без необходимости каких-либо изменений в приложения. Это правило создает базу данных, независимую от внешней программы и ее интерфейса.

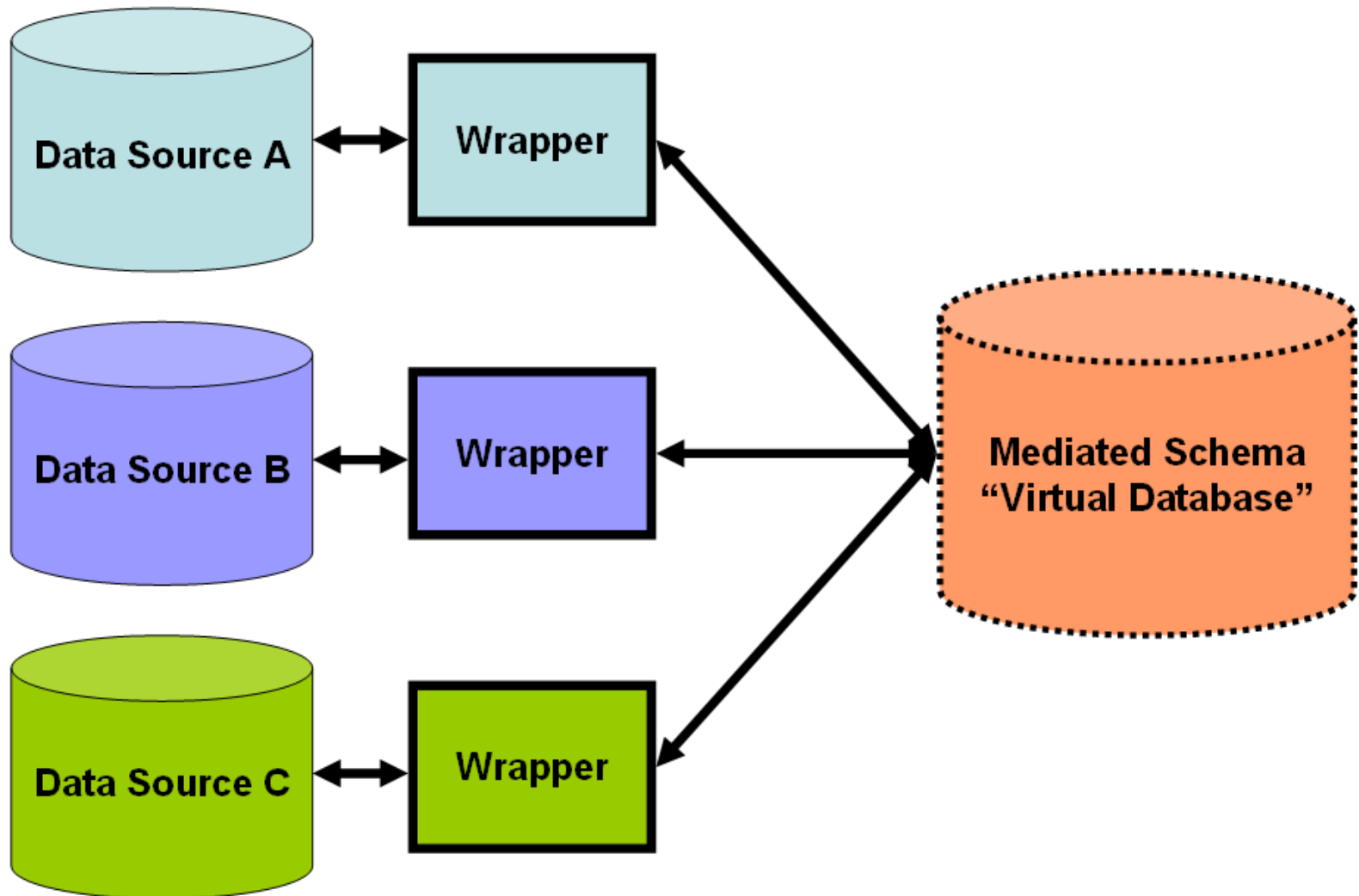
A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Правило 11: Независимость от расположения

/ Rule 11: Distribution Independence

Конечный пользователь не должен иметь возможности видеть, что данные распределяются по разным местам. Пользователям всегда нужно создать впечатление, что данные размещаются только в одном месте. Это правило рассматривается как основа распределенных систем баз данных.

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.



Правило 12: Согласование языковых уровней / Rule 12: Non-Subversion

Если в системе есть интерфейс, обеспечивающий доступ к записям низкого уровня, то интерфейс не должен иметь возможности сломать систему и обходить ограничения безопасности и целостности.

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Ограничения / Constraints

Для каждого отношения существуют некоторые условия, которые должны ему выполняться. Эти условия называются **реляционными ограничениями целостности**. Существует три основных ограничения целостности:

- ключей;
- доменов;
- ссылок.

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints:

- Key constraints;
- Domain constraints;
- Referential integrity constraints.

Ограничения ключей / Key constraints

В отношении должна быть по крайней мере одна минимальная подмножество атрибутов, которая может однозначно идентифицировать кортеж. Эта минимальная подмножество атрибутов называется **ключом** для этого отношения. Если существует несколько таких минимальных подмножеств, они называются **потенциальными ключами**.

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called **candidate keys**.

Ограничения ключей определяют, что:

- в отношении с ключевым атрибутом, два кортежа не могут иметь идентичные значения для ключевых атрибутов.
- ключевой атрибут не может иметь значение NULL.

Ограничения ключей также называются ограничениями сущности.

Key constraints force that:

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Ограничения доменов / Domain constraints

В реальности атрибуты имеют определенные значения. Например, возраст может быть только положительным целым числом. Те же ограничения попытались применить к атрибутам отношения. Каждый атрибут должен иметь определенный диапазон значений. Например, возраст не может быть меньше нуля, а номера телефонов не могут содержать цифры за пределами 0-9.

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

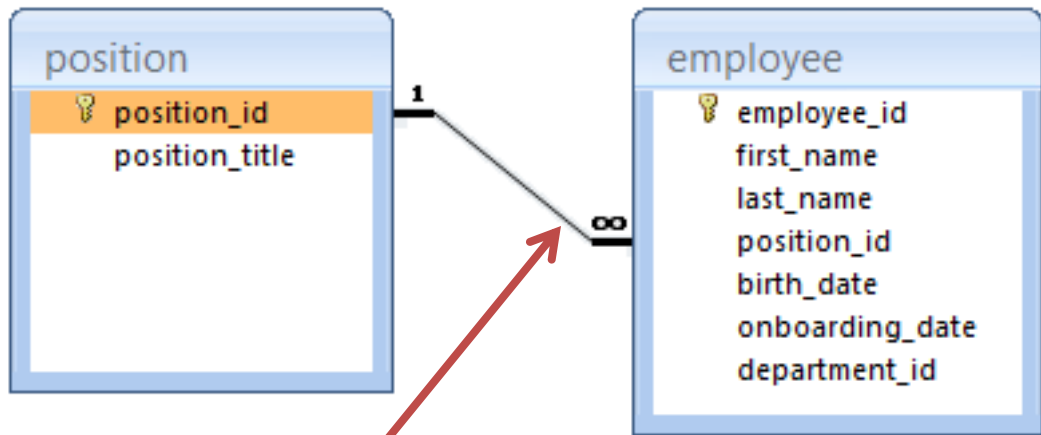
Ограничения ссылочной целостности / Referential integrity constraints

Ограничения целостности ссылок оперируют с концепцией **внешних ключей**. Внешний ключ является ключевым атрибутом отношения, на который можно сослаться в другом отношении.

Ограничения целостности ссылок указывает, что если отношение ссылается на ключевой атрибут другого или того же самого отношения, то этот ключевой атрибут должен существовать.

Referential integrity constraints work on the concept of **Foreign Keys**. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.



1) Double-click on the relation

2) Check the first box to enforce referential integrity

The 'Edit Relationships' dialog box shows the relationship between 'position' and 'employee' tables. The 'Table/Query' is 'position' and the 'Related Table/Query' is 'employee'. The primary key 'position_id' in 'position' is linked to the 'position_id' field in 'employee'. The 'Enforce Referential Integrity' checkbox is checked. The 'Relationship Type' is 'One-To-Many'. Buttons for 'OK', 'Cancel', 'Join Type..', and 'Create New..' are on the right. A red arrow points from the text '2) Check the first box to enforce referential integrity' to the 'Enforce Referential Integrity' checkbox.

Table/Query:	Related Table/Query:
position	employee
position_id	position_id

☒ Enforce Referential Integrity
☐ Cascade Update Related Fields
☐ Cascade Delete Related Records

Relationship Type: One-To-Many

Реляционная алгебра / Relational algebra

Реляционная алгебра является процедурным языком запросов, которая принимает экземпляры отношений и возвращает экземпляры отношений. Она использует операторы для выполнения запросов. Оператор может быть как **унарным**, так и **бинарным**. Они принимают отношение и возвращают также отношения. Реляционная алгебра выполняется рекурсивно в отношении, а промежуточные результаты также рассматриваются как отношения.

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Операция выбора / Select operation

Он выбирает кортежи, которые удовлетворяют данному предикату, из отношения.

Обозначение – $\sigma_p(r)$.

Где σ означает предикат выбора, а r – отношение. p – формула логики препозиции, которая может использовать соединители, такие как AND, OR, и NOT. Эти условия могут использовать реляционные операторы, такие как: $=$, \neq , \geq , $<$, $>$, \leq .

It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$.

Where σ stands for selection predicate and r stands for relation. p is propositional logic formula which may use connectors like AND, OR, and NOT. These terms may use relational operators like: $=$, \neq , \geq , $<$, $>$, \leq .

r = 'literature'

Book	Pages	Type
SQL Cookbook	218	Technical
Harry Potter and bla-bla-bla	411	Novel
LotR	593	Fantasy
C in Examples	299	Technical
The Walking Dead #39	18	Comics

$\sigma_p = \text{Pages} < 300 \text{ AND Type} = \text{'Comics'} \text{ (r = 'literature')}$

RESULT?

Операция проекции / Project operation

Она проектирует столбец(цы), удовлетворяющие заданному предикату.

Обозначение – $\Pi_{A_1, A_2, A_n}(r)$.

Где A_1, A_2, \dots, A_n – названия атрибутов отношения r .

Дублированные строки автоматически исключаются, поскольку отношение является множеством.

It projects column(s) that satisfy a given predicate.

Notation – $\Pi_{A_1, A_2, A_n}(r)$.

Where A_1, A_2, \dots, A_n are attribute names of relation r .

Duplicate rows are automatically eliminated, as relation is a set.

r = 'literature'

Book	Pages	Type
SQL Cookbook	218	Technical
Harry Potter and bla-bla-bla	411	Novel
LotR	593	Fantasy
C in Examples	299	Technical
The Walking Dead #39	18	Comics

$\Pi_{\text{Book, Pages}} (r = \text{'literature'})$

RESULT?

Операция объединения / Union operation

Она выполняет объединение двух отношений и определяется как :

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}.$$

Где **r** и **s** – это отношения базы данных или результирующие отношения (временные отношения).

Для того, чтобы операция объединения была действительной, **r** и **s** должны иметь одинаковое количество атрибутов домены атрибутов должны быть совместимыми; дублированные кортежи автоматически исключаются.

It performs binary union between two given relations and is defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}.$$

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, **r** and **s** must have the same number of attributes; attribute domains must be compatible; duplicate tuples are automatically eliminated.

$r1 = \text{'technical'}$

Book	Pages	Type
SQL Cookbook	218	Technical
C in Examples	299	Technical

$r2 = \text{'other'}$

Book	Pages	Type
Harry Potter and bla-bla-bla	411	Novel
LotR	593	Fantasy
The Walking Dead #39	18	Comics

$r1 \cup r2 = \{t \mid t \in r1 \text{ or } t \in r2\}$

RESULT?

Разница отношений / Set difference

Результатом операции разницы множеств является кортежи, которые присутствуют в одном отношении, но не во втором отношении.

Обозначение – $r - s$.

Находит все кортежи, которые присутствуют в r , но не в s .

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – $r - s$.

Finds all the tuples that are present in r but not in s .

Декартово произведение / Cartesian product

Объединяет информацию из двух различных отношений в одно.

Обозначение – $r \times s$.

Где r и s – отношения, и их результат будет определен как:

$$r \times s = \{q \ t \mid q \in r \text{ and } t \in s\}.$$

Combines information of two different relations into one.

Notation – $r \times s$.

Where r and s are relations and their output will be defined as:

$$r \times s = \{q \ t \mid q \in r \text{ and } t \in s\}.$$

$r1 = \text{'books'}$

Book	Pages
SQL Cookbook	218
LotR	593

$r2 = \text{'book types'}$

Book	Type
SQL Cookbook	Technical
LotR	Fantasy

$r1 \times r2 = \{q \ t \mid q \in r1 \text{ and } t \in r2\}$

RESULT?

Операция переименования / Rename operation

Результаты реляционной алгебры – это также отношения, но без названия. Операция переименования позволяет нам переименовать исходное отношение.

Обозначение – $\rho_x(E)$.

Где результат выражения E сохраняется с названием x .

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation.

Notation – $\rho_x(E)$.

Where the result of expression E is saved with name of x .

Реляционное исчисление / Relational Calculus

В отличие от реляционной алгебры, реляционное исчисление является не процедурным языком запросов, то есть оно определяет, что делать, но никогда не объясняет, как это сделать.

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Реляционное исчисление кортежей / Tuple Relational Calculus (TRC)

Фильтрация кортежей по переменным
Filtering variable ranges over tuples

$$\{ T \mid \text{Condition} \}$$

Возвращает все кортежи **T**, удовлетворяющие условию
Returns all tuples **T** that satisfies a condition

$$\{ T.\text{name} \mid \text{Supplier}(T) \text{ AND } T.\text{id} = '3' \}$$

Какой будет результат?
What will be the output?

Реляционное исчисление доменов / Domain Relational Calculus (DRC)

Переменные фильтрации используют домен атрибутов вместо всех значений кортежа (как это реализовано в TRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC)

$$\{ a1, a2, \dots, an \mid P(a1, a2, \dots, an) \}$$

Где **a1, a2, ..., an** – атрибуты, **P** – формула, основанная на атрибутах.

Where **a1, a2, ..., an** are attributes and **P** stands for formulae built by inner attributes.

$$\{ \langle \text{name}, \text{address} \rangle \mid \in \text{Supplier} \wedge \text{id} = '3' \}$$

Какой будет результат?

What will be the output?

Лекция 06: Язык запросов SQL

Lecture 06: SQL query language

SQL

SQL – это структурированный язык запросов для хранения, обработки и получения данных, хранящихся в реляционной базе данных. SQL является стандартным языком ANSI (American National Standards Institute), но существует множество различных версий языка SQL.

SQL is Structured Query Language, used for storing, manipulating and retrieving data stored in a relational database. SQL is an ANSI (American National Standards Institute) standard language, but there are many different versions of the SQL language.

История SQL

- 1970** – Доктор Кодд с IBM описал реляционную модель для баз данных.
- 1974** – Появился язык SQL.
- 1978** – IBM работал над разработкой идей Кодда и выпустил продукт под названием System / R.
- 1986** – IBM разработал первый прототип реляционной базы данных, стандартизированный ANSI. Первая реляционная база данных была выпущена компанией Relational Software, которая впоследствии стала известна как Oracle.

History of SQL

- 1970** – Dr. Edgar Codd of IBM described a relational model for databases.
- 1974** – Structured Query Language appeared.
- 1978** – IBM worked to develop Codd's ideas and released a product named System/R.
- 1986** – IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

Использование SQL / SQL usage

Все реляционные СУБД, такие как MySQL, MS Access, Oracle, Sybase, Informix, PostgreSQL и SQL Server используют SQL как стандартный язык базы данных.

All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, PostgreSQL and SQL Server use SQL as their standard database language.

Vendor	Relational DBMS	SQL Dialects
Microsoft	SQL Server	T-SQL (Transact-SQL)
	Access	JET SQL (Native format)
Oracle	Oracle Database	PL/SQL (Procedural Language/SQL)
SAP	Sybase IQ	ANSI SQL (Native format) T-SQL
Open-Source	PostgreSQL	ANSI SQL PL/pgSQL (similar to PL/SQL) SQL:2011
Oracle	MySQL	SQL:2003

Преимущества SQL

- Позволяет пользователям получать доступ к данным в реляционных СУБД.
- Позволяет пользователям описывать данные.
- Позволяет пользователям определять данные в базе данных и манипулировать ими.
- Позволяет встраивать другие языки с помощью модулей SQL, библиотек и прекомпиляторов.
- Позволяет пользователям создавать и удалять базы данных и таблицы.
- Позволяет пользователям создавать представления, хранимые процедуры, функции в базе данных.
- Позволяет пользователям устанавливать права на таблицы, процедуры и представления.

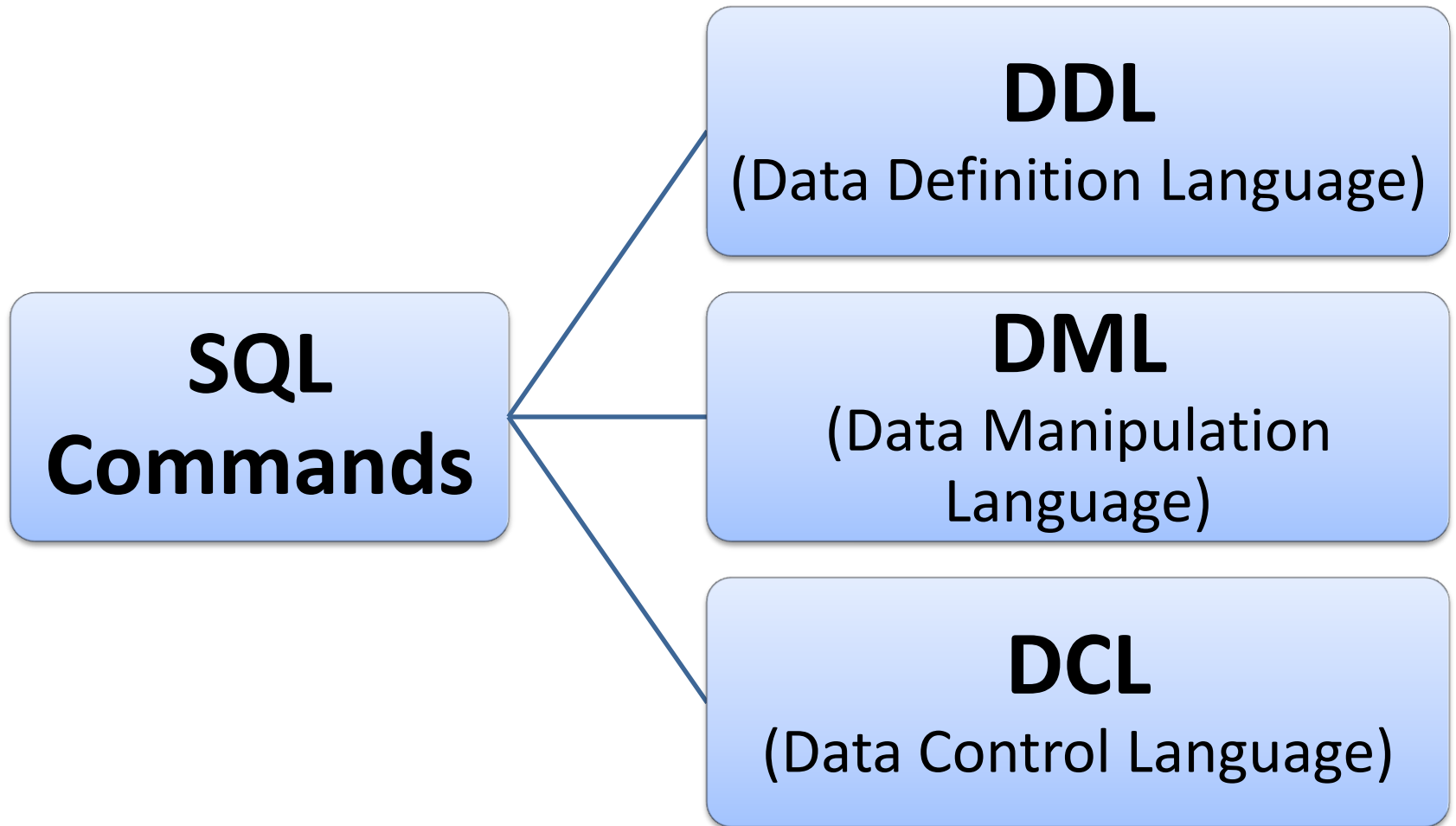
Advantages of SQL

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

Команды SQL / SQL commands

Стандартные команды SQL для взаимодействия с реляционными базами данных – CREATE, SELECT, INSERT, UPDATE, DELETE и DROP. Эти команды можно разделить на следующие группы в зависимости от их характера.

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature.



Data Definition Language (DDL)

Номер	Команда и описание
1	CREATE Создает новую таблицу, представление таблицы или другой объект в базе данных Creates a new table, a view of a table, or other object in the database
2	ALTER Модифицирует существующий объект базы данных, такой как таблица Modifies an existing database object, such as a table
3	DROP Удаляет всю таблицу, представление таблицы или другие объекты в базе данных Deletes an entire table, a view of a table or other objects in the database

Data Manipulation Language (DML)

Номер	Команда и описание
1	SELECT Получает определенные записи из одной или нескольких таблиц Retrieves certain records from one or more tables
2	INSERT Создает запись Creates a record
3	UPDATE Модифицирует записи Modifies records
4	DELETE Удаляет записи Deletes records

Data Control Language (DCL)

Номер	Команда и описание
1	GRANT Дает преимущество пользователю Gives a privilege to user
2	REVOKE Отменяет привилегии, предоставленные пользователю Takes back privileges granted from user

Процесс выполнения команды SQL / SQL command execution process

Когда вы выполняете команду SQL для любой реляционной СУБД, система определяет лучший способ выполнения вашего запроса, и движок SQL вычисляет, как интерпретировать задачи.

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

В этом процессе принимают участие различные компоненты.

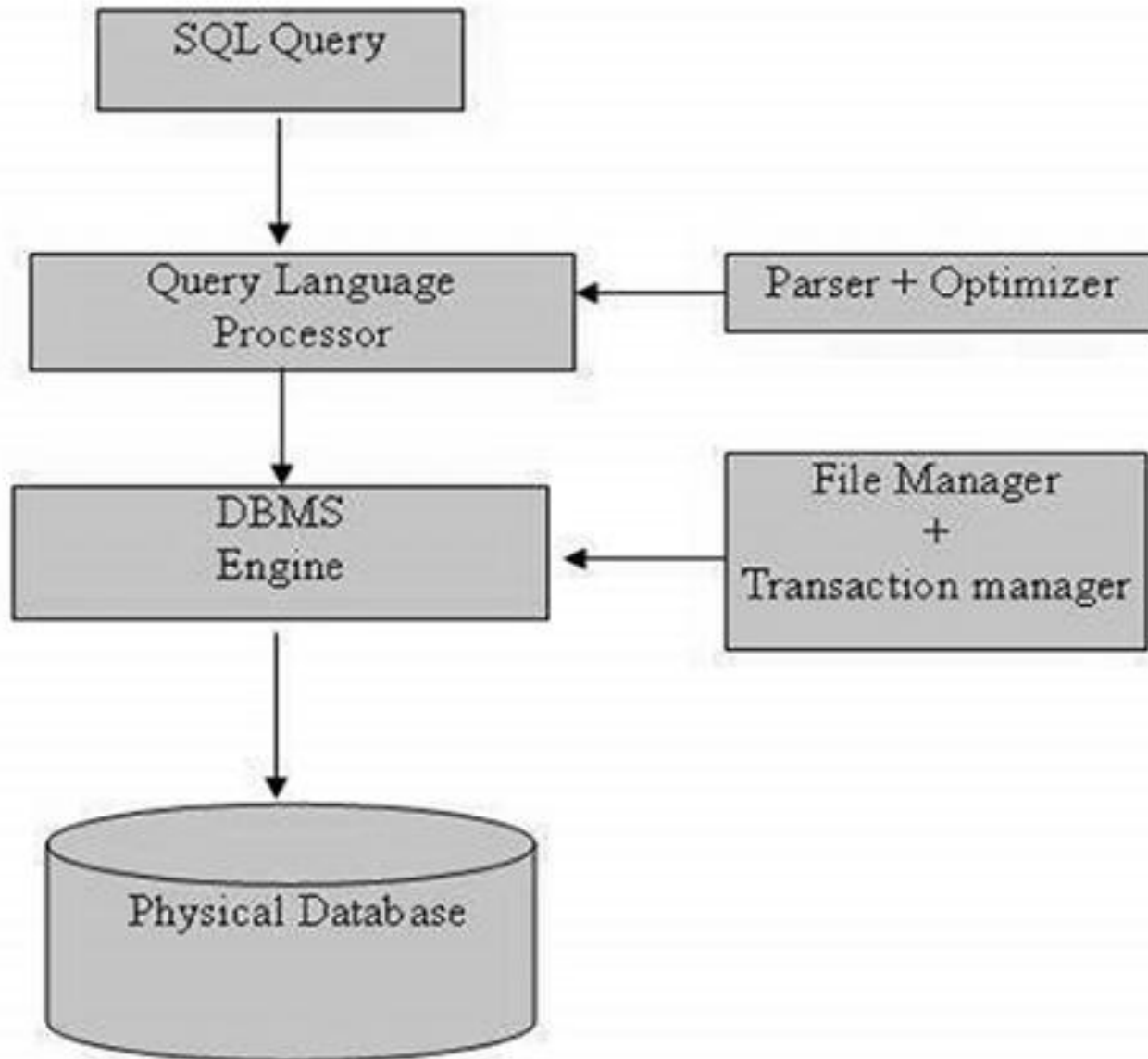
There are various components included in this process.

Query Dispatcher (диспетчер запросов)

Optimization Engines (инструмент оптимизации)

Classic Query Language (движок запросов)

SQL Query Engine (движок запросов)



SQL data types

Тип данных SQL – это атрибут, который определяет тип данных любого объекта. Каждый столбец, переменная и выражение имеют соответствующий тип данных в SQL. Эти типы данных используются при создании таблиц. Тип данных для столбца таблицы избирается в соответствии с требованиями.

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

Numeric data types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	$-10^{38} + 1$	$10^{38} - 1$
numeric	$-10^{38} + 1$	$10^{38} - 1$
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647
float	$-1.79E + 308$	$1.79E + 308$
real	$-3.40E + 38$	$3.40E + 38$

Date and time data types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Character strings data types

No.	DATA TYPE & Description
1	char Maximum length of 8,000 characters. (Fixed length non-Unicode characters)
2	varchar Maximum of 8,000 characters. (Variable-length non-Unicode data).
3	varchar(max) Maximum length of 2^{E+31} characters. (Variable-length non-Unicode data (SQL Server 2005 only)).
4	text Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode character strings data type

No.	DATA TYPE & Description
1	nchar Maximum length of 4,000 characters. (Fixed length Unicode)
2	nvarchar Maximum length of 4,000 characters. (Variable length Unicode)
3	nvarchar(max) Maximum length of 2E + 31 characters (SQL Server 2005 only). (Variable length Unicode)
4	ntext Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary data types

No.	DATA TYPE & Description
1	binary Maximum length of 8,000 bytes. (Fixed-length binary data)
2	varbinary Maximum length of 8,000 bytes. (Variable length binary data)
3	varbinary(max) Maximum length of 2E + 31 bytes (SQL Server 2005 only). (Variable length Binary data)
4	image Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Синтаксис SQL / SQL syntax

Все операторы SQL начинаются с любого из ключевых слов, таких как SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW, и все утверждения заканчиваются точкой с запятой (;).

SQL является нечувствительным к регистру, то есть SELECT и select имеют одинаковое значение в операторах SQL.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

SQL is case insensitive, which means SELECT and select have same meaning in SQL statements.

SQL SELECT

SELECT

[**DISTINCT** | **ALL** | **TOP** {unsigned_integer}]
select_expression,...

FROM table_references

[**WHERE** where_definition]

[**GROUP BY** {unsigned_integer | col_name |
formula}]

[**HAVING** where_definition]

[**ORDER BY** {unsigned_integer | col_name |
formula} [**ASC** | **DESC**], ...]

SQL SELECT statement

SELECT

column1, column2, ..., columnN

FROM table_name;

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



SELECT

first_name, last_name, birth_date

FROM employee;



	first_name: ▾	last_name: ▾	birth_date: ▾
	Jim	Halpert	5/12/1990
	Pamela	Beesly	11/2/1992
	Dwight	Schrute	2/11/1991
	Kelly	Kapoor	6/4/1993
	Michael	Scott	12/9/1989

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



SELECT

*

FROM employee;



	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾	department ▾
	1	Jim	Halpert	2	5/12/1990	3/2/2014	1
	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1
	4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3
	5	Michael	Scott	5	12/9/1989	4/28/2014	4

SQL SELECT – DISTINCT

Ключевое слово SQL DISTINCT используется вместе с оператором SELECT для исключения всех дублированных записей и получения только уникальных записей.

The SQL DISTINCT keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

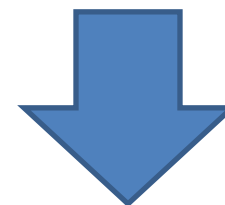
SQL DISTINCT clause

```
SELECT DISTINCT  
column1, column2, ..., columnN  
FROM table_name;
```

employee_id: ▾	skill_id: ▾	gained_date: ▾
1	1	5/30/2012
1	2	5/30/2012
1	5	4/12/2013
1	6	5/30/2012
3	1	8/10/2011
3	2	8/10/2011
3	4	12/5/2011
3	5	1/20/2012
3	6	12/5/2011
5	7	5/21/2008



SELECT DISTINCT
employee_id
FROM skills_matrix;



employee_id: ▾
1
3
5

КАКОЙ БУДЕТ РЕЗУЛЬТАТ ЕСЛИ ВМЕСТО
DISTINCT БУДЕТ **ALL**?

WHAT WILL BE THE OUTPUT IF WE USE
THE KEYWORD **ALL** INSTEAD OF **DISTINCT**?

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



SELECT TOP 3

FROM employee;



	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾	department: ▾
	1	Jim	Halpert	2	5/12/1990	3/2/2014	1
	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1

SQL SELECT – ORDER BY

По умолчанию SQL ORDER BY используется для сортировки данных в порядке возрастания или убывания на основе одного или нескольких столбцов. Некоторые базы данных сортируют результаты запроса по возрастанию.

The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

SQL ORDER BY clause

SELECT

column1, column2, ..., columnN

FROM table_name

ORDER BY column_name {ASC | DESC};

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```

SELECT TOP 3
first_name, last_name, onboarding_date
FROM employee
ORDER BY onboarding_date;

```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Dwight	Schrute	12/3/2013
	Kelly	Kapoor	3/2/2014
	Jim	Halpert	3/2/2014

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```
SELECT TOP 3
first_name, last_name, onboarding_date
FROM employee
ORDER BY onboarding_date DESC;
```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Pamela	Beesly	9/23/2015
	Michael	Scott	4/28/2014
	Kelly	Kapoor	3/2/2014
	Jim	Halpert	3/2/2014

SQL SELECT – WHERE

Позиция SQL WHERE используется для указания условия для получения данных из одной таблицы или путем объединения с несколькими таблицами. Если это условие выполняется, то только она возвращает определенное значение из таблицы. WHERE используют чтобы отфильтровать записи и получать только необходимые записи.

The SQL WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. WHERE is used to filter the records and fetching only the necessary records.

SQL WHERE clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE CONDITION;

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```

SELECT
first_name, last_name, onboarding_date
FROM employee
WHERE first_name = "Jim"

```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Jim	Halpert	3/2/2014

SQL – Operators

Оператор является зарезервированным словом или символом, который используется, прежде всего, в позиции WHERE оператора SQL для выполнения операций, таких как сравнение, и арифметических операций. Эти операторы используются для того, чтобы указывать условия в операторе SQL и выступать в качестве сообщений для нескольких условий в заявлении.

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

SQL arithmetic operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	$a + b$ will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	$a - b$ will give -10
* (Multiplication)	Multiplies values on either side of the operator.	$a * b$ will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	$b \% a$ will give 0

SQL comparison operators (1)

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.

SQL comparison operators (2)

Operator	Description	Example
<code>>=</code>	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	<code>(a >= b)</code> is not true.
<code><=</code>	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	<code>(a <= b)</code> is true.
<code>!<</code>	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	<code>(a !< b)</code> is false.
<code>!></code>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	<code>(a !> b)</code> is true.

SQL logical operators (1)

No.	Operator & Description
1	ALL The ALL operator is used to compare a value to all values in another value set.
2	AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	BETWEEN The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.

SQL logical operators (2)

No.	Operator & Description
6	IN The IN operator is used to compare a value to a list of literal values that have been specified.
7	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
8	NOT The NOT operator reverses the meaning of the logical operator with which it is used.
9	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	IS NULL The NULL operator is used to compare a value with a NULL value.
11	UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL SELECT – LIKE

Позиция SQL LIKE используется для сравнения значения с аналогичными значениями, используя операторы подстановки. Есть два символа подстановки, которые используются совместно с оператором LIKE:

- * – вместо одного или нескольких символов (или %);
- ? – вместо одного символа (или _).

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

- * – Matches one or more characters (or %).
- ? – Matches one character (or _)

SQL LIKE clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE column_name LIKE { pattern };

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```

SELECT
first_name, last_name, onboarding_date
FROM employee
WHERE last_name LIKE "S*";

```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Dwight	Schrute	12/3/2013
	Michael	Scott	4/28/2014

SQL SELECT – AND & OR

Операторы SQL AND и OR используются для объединения нескольких условий для фильтрации данных в операторе SQL. Эти два оператора называются конъюнктивные операторами.

The SQL AND & OR operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

SQL AND/OR clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE CONDITION1 { AND|OR }

CONDITION 2;

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```

SELECT
first_name, last_name, onboarding_date
FROM employee
WHERE first_name = "Jim" OR first_name = "Kelly";

```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Jim	Halpert	3/2/2014
	Kelly	Kapoor	3/2/2014

	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014
+	5	Michael	Scott	5	12/9/1989	4/28/2014



```

SELECT
first_name, last_name, onboarding_date
FROM employee
WHERE department_id = 1 AND position_id = 3;

```



	first_name: ▾	last_name: ▾	onboarding_date: ▾
	Dwight	Schrute	12/3/2013

SQL IN clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE column_name IN (val-1, val-2,
..., val-N);

SQL BETWEEN clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE column_name BETWEEN val-1
AND val-2;

SQL SELECT – GROUP BY

Позиция SQL GROUP BY используется в сотрудничестве с оператором SELECT для организации одинаковых данных в группы. Эта позиция GROUP BY следует за позицией WHERE в операторе SELECT и предшествует позиции ORDER BY.

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

SQL GROUP BY clause

SELECT

column1, column2, ..., columnN

FROM table_name

WHERE CONDITION

GROUP BY column_name;

project_id: ▾	employee_i ▾	from_date: ▾	to_date: ▾	role_id: ▾
1	1	2/28/2016	12/18/2016	1
1	2	4/20/2016	12/18/2016	3
1	3	2/25/2016	12/15/2016	2
1	4	4/20/2016	12/19/2016	4
1	5	2/14/2016	12/19/2016	5
2	5	10/17/2016	4/22/2017	5
4	3	5/15/2017	12/31/2018	4
4	5	5/11/2017	12/31/2018	5
5	4	12/5/2017	2/20/2018	4



```

SELECT
employee_id, COUNT(project_id) AS [number_of_projects]
FROM resources_allocation
GROUP BY employee_id;

```



employee_id: ▾	number_of_projects ▾
1	1
2	1
3	2
4	2
5	3

Функции агрегации / Aggregate functions

Number	Function	Description
1	Avg	Calculates the arithmetic mean of a set of values contained in a specified field on a query. Среднее арифметическое.
2	Count	Calculates the number of records returned by a query. Количество записей.
3	First	Return a field value from the first or last record in the result set returned by a query. Значение первой / последней записи.
4	Last	
5	Min	Return the minimum or maximum of a set of values contained in a specified field on a query. Максимальное / минимальное значение.
6	Max	
7	Sum	Returns the sum of a set of values contained in a specified field on a query. Сумма значений.

SQL COUNT clause

```
SELECT  
COUNT(column_name)  
FROM table_name  
WHERE CONDITION;
```

project_id: ▾	employee_i ▾	from_date: ▾	to_date: ▾	role_id: ▾
1	1	2/28/2016	12/18/2016	1
1	2	4/20/2016	12/18/2016	3
1	3	2/25/2016	12/15/2016	2
1	4	4/20/2016	12/19/2016	4
1	5	2/14/2016	12/19/2016	5
2	5	10/17/2016	4/22/2017	5
4	3	5/15/2017	12/31/2018	4
4	5	5/11/2017	12/31/2018	5
5	4	12/5/2017	2/20/2018	4



```

SELECT
employee_id, Sum(to_date - from_date) AS [days_on_projects]
FROM resources_allocation
GROUP BY employee_id;

```



employee_id: ▾	days_on_projects ▾
1	294
2	242
3	889
4	320
5	1095

SQL SELECT – HAVING

Позиция HAVING позволяет определить условия, которые фильтруют группы, появляющиеся в результатах.

Позиция WHERE содержит условия для выбранных столбцов, тогда как позиция HAVING содержит условия для групп, созданных в пункте GROUP BY.

The HAVING Clause enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

SQL HAVING clause

```
SELECT  
SUM(column_name)  
FROM table_name  
WHERE CONDITION  
GROUP BY column_name  
HAVING CONDITION;
```

project_id: ▾	employee_i ▾	from_date: ▾	to_date: ▾	role_id: ▾
1	1	2/28/2016	12/18/2016	1
1	2	4/20/2016	12/18/2016	3
1	3	2/25/2016	12/15/2016	2
1	4	4/20/2016	12/19/2016	4
1	5	2/14/2016	12/19/2016	5
2	5	10/17/2016	4/22/2017	5
4	3	5/15/2017	12/31/2018	4
4	5	5/11/2017	12/31/2018	5
5	4	12/5/2017	2/20/2018	4



```

SELECT
employee_id, Sum(to_date - from_date) AS [days_on_projects]
FROM resources_allocation
GROUP BY employee_id
HAVING Sum(to_date - from_date) >= 300;

```



employee_id: ▾	days_on_projects ▾
3	889
4	320
5	1095

SQL SELECT – JOIN

Позиция SQL JOIN используется для объединения записей из двух или более таблиц в базе данных. JOIN используется для объединения атрибутов из двух таблиц, используя общие для каждой таблицы значения.

The SQL JOIN clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

employee							
	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding: ▾	department: ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014	1
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3
+	5	Michael	Scott	5	12/9/1989	4/28/2014	4

position	
position_id: ▾	position_title: ▾
1	QA Eng.
2	SW Eng.
3	Sr. SW Eng.
4	Syst. Eng.
5	PM

department	
department: ▾	department_title: ▾
1	Dev. Dept.
2	QA Dept.
3	Maintenance Dept.
4	Management Dept.
5	Accounting Dept.
6	HR Dept.

Query 01

SELECT

employee.employee_id,
employee.first_name,
employee.last_name,
position.position_title,
department.department_title

FROM

employee, position, department;

Query1				
employee_id: ▾	first_name: ▾	last_name: ▾	position_tit ▾	department_title: ▾
1	Jim	Halpert	QA Eng.	Dev. Dept.
2	Pamela	Beesly	QA Eng.	Dev. Dept.
3	Dwight	Schrute	QA Eng.	Dev. Dept.
4	Kelly	Kapoor	QA Eng.	Dev. Dept.
5	Michael	Scott	QA Eng.	Dev. Dept.
1	Jim	Halpert	SW Eng.	Dev. Dept.
2	Pamela	Beesly	SW Eng.	Dev. Dept.
3	Dwight	Schrute	SW Eng.	Dev. Dept.
4	Kelly	Kapoor	SW Eng.	Dev. Dept.
5	Michael	Scott	SW Eng.	Dev. Dept.
1	Jim	Halpert	Sr. SW Eng.	Dev. Dept.
2	Pamela	Beesly	Sr. SW Eng.	Dev. Dept.
3	Dwight	Schrute	Sr. SW Eng.	Dev. Dept.
4	Kelly	Kapoor	Sr. SW Eng.	Dev. Dept.
5	Michael	Scott	Sr. SW Eng.	Dev. Dept.
1	Jim	Halpert	Syst. Eng.	Dev. Dept.
2	Pamela	Beesly	Syst. Eng.	Dev. Dept.
3	Dwight	Schrute	Syst. Eng.	Dev. Dept.
4	Kelly	Kapoor	Syst. Eng.	Dev. Dept.
5	Michael	Scott	Syst. Eng.	Dev. Dept.
1	Jim	Halpert	PM	Dev. Dept.
2	Pamela	Beesly	PM	Dev. Dept.
3	Dwight	Schrute	PM	Dev. Dept.
4	Kelly	Kapoor	PM	Dev. Dept.
5	Michael	Scott	PM	Dev. Dept.

Record: 1 of 150 No Filter Search

Query result:
150 records (!!!)
Instead of 5

WHY?

Декартово соединение / Cartesian or cross join

Декартово соединение возвращает декартово произведение из наборов записей из двух или более объединенных таблиц. Таким образом, оно приравнивается к внутреннему соединению, где условие присоединения всегда оценивается как истина, или где условие соединения отсутствует в заявлении.

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

SQL SELECT – INNER JOIN

Наиболее важным и часто используемым соединением является INNER JOIN. Оно также называется EQUIJOIN.

INNER JOIN создает новую таблицу результатов, объединяя значение столбиков из двух таблиц на основе предиката присоединения. Запрос сравнивает каждую строку первой таблицы с каждой строкой второй таблицы, чтобы найти все пары строк, удовлетворяющих предикату присоединения. Когда предикат присоединения выполняется, значения столбцов для каждой сопоставленной пары строк объединяются в строку результата.

SQL SELECT – INNER JOIN

The most important and frequently used of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.

The INNER JOIN creates a new result table by combining column values of two tables based upon the join-predicate. The query compares each row of the first table with each row of the second table to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows are combined into a result row.

Query 02

SELECT

employee.employee_id,
employee.first_name,
employee.last_name,
position.position_title,
department.department_title

FROM

(employee INNER JOIN position ON employee.position_id
= position.position_id) INNER JOIN department ON
employee.department_id =
department.department_id;

Query1				
employee_id: ▾	first_name: ▾	last_name: ▾	position_tit ▾	department_title: ▾
1	Jim	Halpert	SW Eng.	Dev. Dept.
2	Pamela	Beesly	QA Eng.	QA Dept.
3	Dwight	Schrute	Sr. SW Eng.	Dev. Dept.
4	Kelly	Kapoor	Syst. Eng.	Maintenance Dept.
5	Michael	Scott	PM	Management Dept.

position	
position_id: ▾	position_tit ▾
1	QA Eng.
2	SW Eng.
3	Sr. SW Eng.
4	Syst. Eng.
5	PM

department	
department ▾	department_title: ▾
1	Dev. Dept.
2	QA Dept.
3	Maintenance Dept.
4	Management Dept.
5	Accounting Dept.
6	HR Dept.

Query1				
employee_id: ▾	first_name: ▾	last_name: ▾	position_tit ▾	department_title: ▾
1	Jim	Halpert	SW Eng.	Dev. Dept.
2	Pamela	Beesly	QA Eng.	QA Dept.
3	Dwight	Schrute	Sr. SW Eng.	Dev. Dept.
4	Kelly	Kapoor	Syst. Eng.	Maintenance Dept.
5	Michael	Scott	PM	Management Dept.

skill	
skill_id: ▾	skill_title: ▾
1	Java SE
2	Maven
3	Java EE
4	Servlets
5	JSP
6	Git
7	UML

skills_matrix		
employee_id: ▾	skill_id: ▾	gained_date: ▾
1	1	5/30/2012
1	2	5/30/2012
1	5	4/12/2013
1	6	5/30/2012
3	1	8/10/2011
3	2	8/10/2011
3	4	12/5/2011
3	5	1/20/2012
3	6	12/5/2011
5	7	5/21/2008

Query 03

SELECT

skills_matrix.employee_id,
employee.first_name,
employee.last_name,
skill.skill_title

FROM

(skills_matrix INNER JOIN employee ON
skills_matrix.employee_id =
employee.employee_id) INNER JOIN skill ON
skills_matrix.skill_id = skill.skill_id;

Query1				
	employee_id: ▾	first_name: ▾	last_name: ▾	skill_title: ▾
	1	Jim	Halpert	Java SE
	1	Jim	Halpert	Maven
	1	Jim	Halpert	Git
	1	Jim	Halpert	JSP
	3	Dwight	Schrute	Java SE
	3	Dwight	Schrute	Maven
	3	Dwight	Schrute	Git
	3	Dwight	Schrute	Servlets
	3	Dwight	Schrute	JSP
	5	Michael	Scott	UML

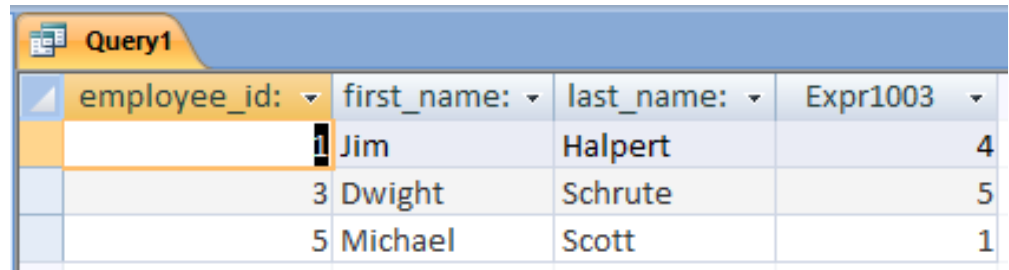
Query result:

skills have been assigned to 3 employees of 5

**HOW WE CAN SEE WHICH EMPLOYEES ARE NOT RELATED TO SKILLS?
WHICH SKILLS ARE NOT RELATED TO EMPLOYEES?**

Query 04a

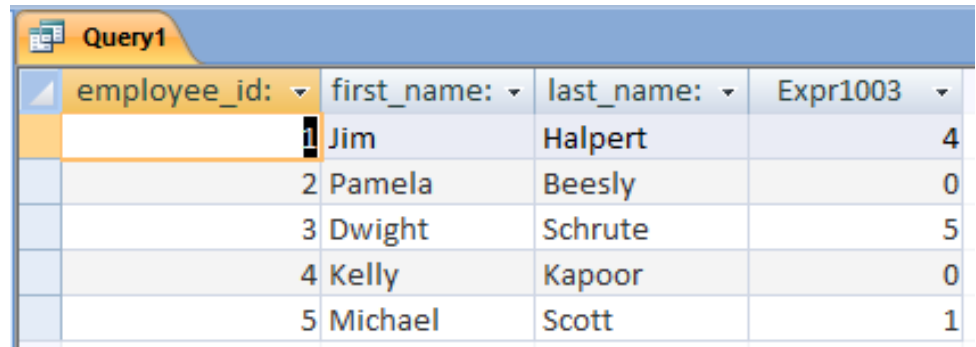
```
SELECT
employee.employee_id,
employee.first_name,
employee.last_name,
COUNT(skills_matrix.skill_id)
FROM
employee INNER JOIN skills_matrix ON
    skills_matrix.employee_id = employee.employee_id
GROUP BY employee.employee_id, employee.first_name,
    employee.last_name;
```



employee_id: ▾	first_name: ▾	last_name: ▾	Expr1003 ▾
1	Jim	Halpert	4
3	Dwight	Schrute	5
5	Michael	Scott	1

Query 04b

```
SELECT
employee.employee_id,
employee.first_name,
employee.last_name,
COUNT(skills_matrix.skill_id)
FROM
employee LEFT JOIN skills_matrix ON
    skills_matrix.employee_id = employee.employee_id
GROUP BY employee.employee_id, employee.first_name,
employee.last_name;
```



employee_id: ▾	first_name: ▾	last_name: ▾	Expr1003 ▾
1	Jim	Halpert	4
2	Pamela	Beesly	0
3	Dwight	Schrute	5
4	Kelly	Kapoor	0
5	Michael	Scott	1

Query 04c

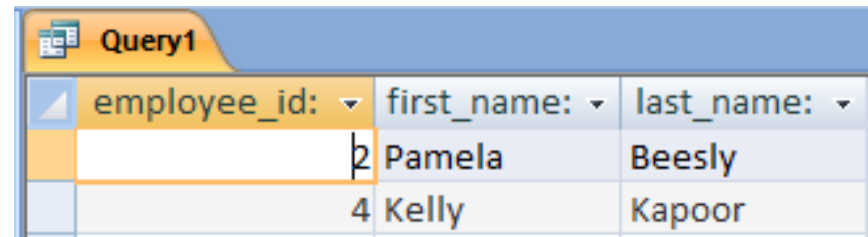
SELECT

employee.employee_id,
employee.first_name,
employee.last_name

FROM

employee LEFT JOIN skills_matrix ON
skills_matrix.employee_id =
employee.employee_id

WHERE skills_matrix.skill_id IS NULL;



employee_id: ▾	first_name: ▾	last_name: ▾
2	Pamela	Beesly
4	Kelly	Kapoor

SQL SELECT – LEFT JOIN

SQL LEFT JOIN возвращает все строки левой таблицы, даже если в правой таблице нет соответствий. Это означает, что если условие ON соответствует 0 (нулевой) записям в правой таблицы; соединение все равно вернет строку в результате, но NULL в каждом столбце из правой таблицы.

Это означает, что LEFT JOIN возвращает все значения из левой таблицы, а также соответствующие значения из правой таблицы или NULL в случае отсутствия соответствия предиката соединения.

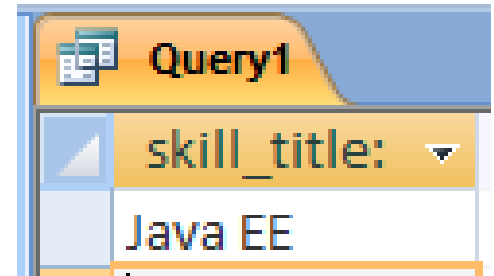
SQL SELECT – LEFT JOIN

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Query 05

```
SELECT  
skill.skill_title  
FROM  
skills_matrix RIGHT JOIN skill ON  
    skills_matrix.skill_id = skill.skill_id  
WHERE skills_matrix.employee_id IS NULL;
```



SQL SELECT – RIGHT JOIN

SQL RIGHT JOIN возвращает все строки из правой таблицы, даже если в левой таблице нет соответствий. Это означает, что если условие ON соответствует 0 (нулевой) записям в левой таблице; соединение все-таки вернет строку в результате, но NULL в каждом столбце левой таблицы.

Это означает, что RIGHT JOIN возвращает все значения из правой таблицы, а также соответствующие значения из левой таблицы или NULL в случае отсутствия соответствия предиката соединения.

SQL SELECT – RIGHT JOIN

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

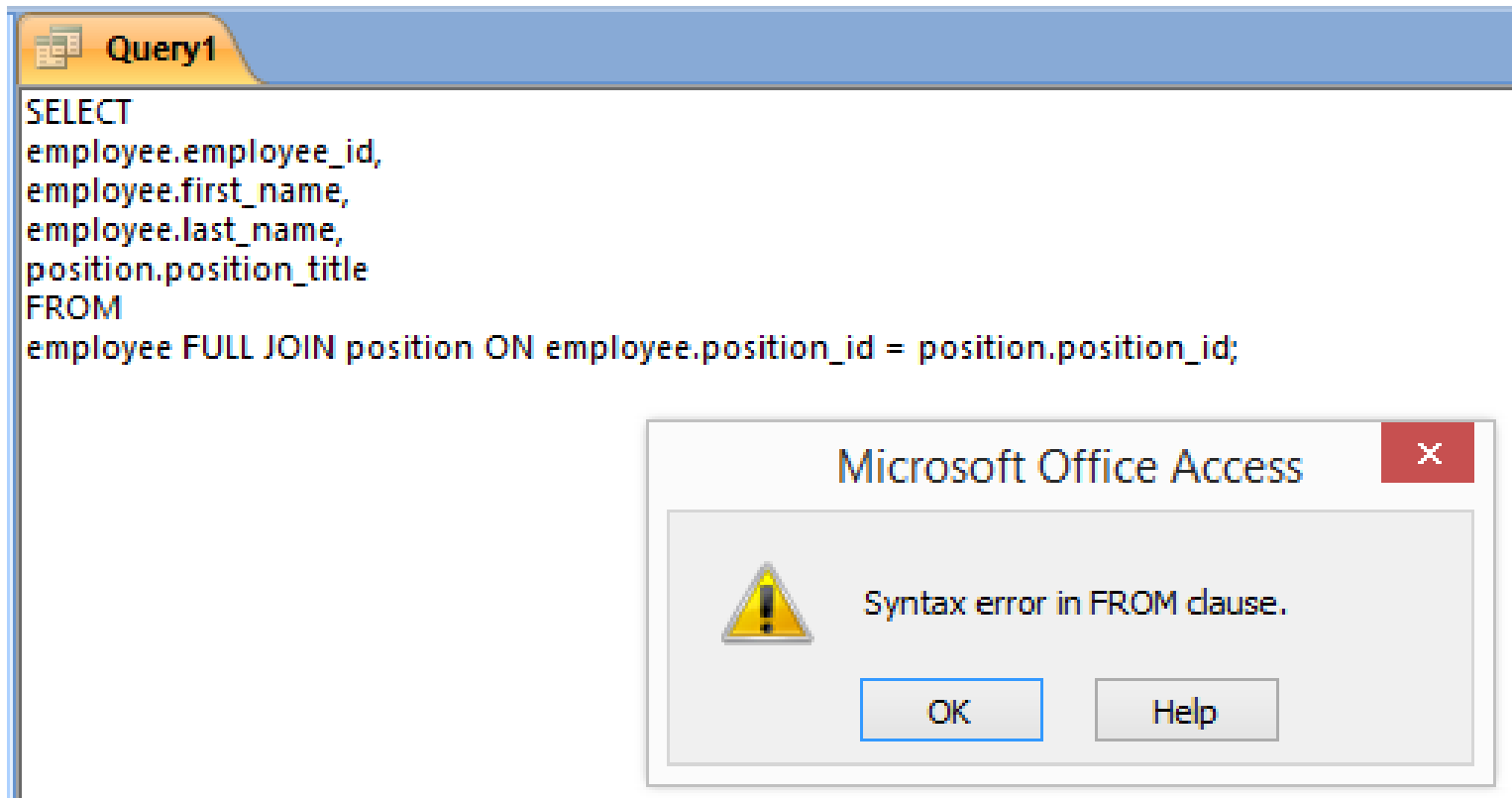
SQL SELECT – FULL JOIN

SQL FULL JOIN объединяет результаты как соединения LEFT JOIN, так и соединения RIGHT JOIN.

Объединенная таблица будет содержать все записи из обеих таблиц, а также значение NULL для отсутствующих соответствий с обеих сторон.

The SQL FULL JOIN combines the results of both left and right outer joins.

The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.



Some database management systems do not support FULL JOINS.

Query 06

SELECT

employee.employee_id,
employee.first_name,
employee.last_name,
position.position_title

FROM

employee LEFT JOIN position ON employee.position_id = position.position_id

UNION

SELECT

employee.employee_id,
employee.first_name,
employee.last_name,
position.position_title

FROM

employee RIGHT JOIN position ON employee.position_id = position.position_id;

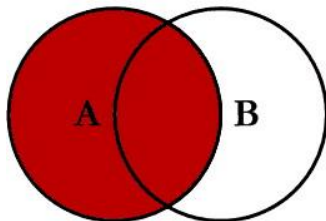
employee							
	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾	department ▾
+	1	Jim	Halpert	2	5/12/1990	3/2/2014	1
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3
+	5	Michael	Scott	5	12/9/1989	4/28/2014	4

position	
position_id: ▾	position_tit ▾
1	QA Eng.
2	SW Eng.
3	Sr. SW Eng.
4	Syst. Eng.
5	PM

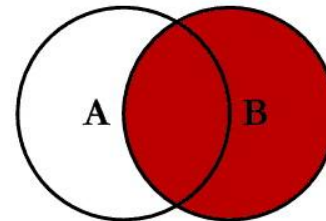
Query1			
employee_id: ▾	first_name: ▾	last_name: ▾	position_tit ▾
			CEO
			Sales Manager
1	Jim	Halpert	SW Eng.
2	Pamela	Beesly	QA Eng.
3	Dwight	Schrute	Sr. SW Eng.
4	Kelly	Kapoor	Syst. Eng.
5	Michael	Scott	PM

SQL JOINS

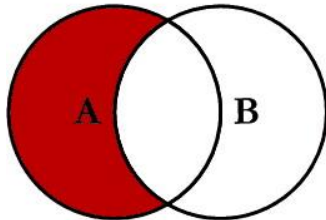
SQL JOINS



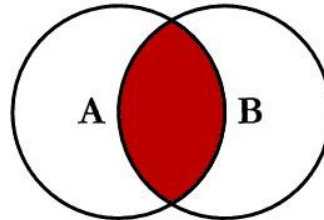
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



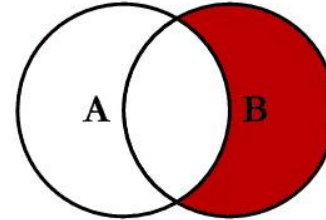
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



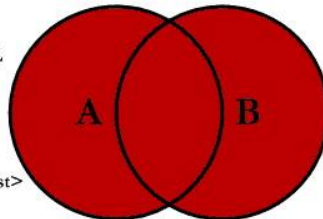
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



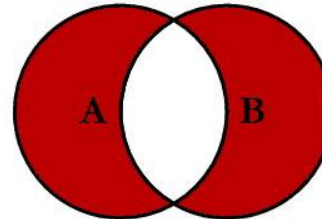
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

SQL SELECT – UNION

Оператор SQL UNION используется для объединения результатов двух или более операторов SELECT без возврата дубликатов строк.

Для корректного использования оператора UNION, каждый оператор SELECT должно соответствовать условиям:

- одинаковое количество столбцов;
- одинаковое количество выражений;
- одинаковые типы данных;
- одинаковый порядок столбцов и выражений.

SQL SELECT – UNION

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have:

- The same number of columns selected.
- The same number of column expressions.
- The same data type.
- Have them in the same order.

UNION syntax

```
SELECT column1 [, column2 ]
```

```
    FROM table1 [, table2 ]
```

```
    [WHERE condition]
```

```
UNION [ALL]
```

```
SELECT column1 [, column2 ]
```

```
    FROM table1 [, table2 ]
```

```
    [WHERE condition]
```

SQL SELECT – UNION ALL

Оператор UNION ALL используется для объединения результатов двух операторов SELECT, включая дубликаты строк.

Те же правила, которые применяются к оператору UNION, применяются и к оператору UNION ALL.

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

SQL SELECT – INTERSECT

Оператор SQL INTERSECT используется для объединения двух операторов SELECT, но возвращает строки только с первого оператора SELECT, которые идентичны строке во втором операторе SELECT. Это означает, что INTERSECT возвращает только общие строки, возвращаемые двумя операторами SELECT.

Как и оператор UNION, применяются те же правила, когда используется оператор INTERSECT.

SQL SELECT – INTERSECT

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator.

INTERSECT syntax

```
SELECT column1 [, column2 ]  
    FROM table1 [, table2 ]  
    [WHERE condition]
```

INTERSECT

```
SELECT column1 [, column2 ]  
    FROM table1 [, table2 ]  
    [WHERE condition]
```


SQL SELECT – EXCEPT

Оператор SQL EXCEPT используется для объединения двух операторов SELECT и возвращает строки из первого оператора SELECT, которые не возвращаются вторым оператором SELECT. Это означает, что EXCEPT возвращает только строки, которые отсутствуют в результате второго оператора SELECT.

Как и оператор UNION, применяются те же правила, когда используется оператор EXCEPT.

SQL SELECT – EXCEPT

The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.

Just as with the UNION operator, the same rules apply when using the EXCEPT operator.

EXCEPT syntax

```
SELECT column1 [, column2 ]  
    FROM table1 [, table2 ]  
    [WHERE condition]
```

EXCEPT

```
SELECT column1 [, column2 ]  
    FROM table1 [, table2 ]  
    [WHERE condition]
```

SQL – INSERT

Выражение SQL INSERT INTO используется для добавления новых строк данных в таблицы в базе данных.

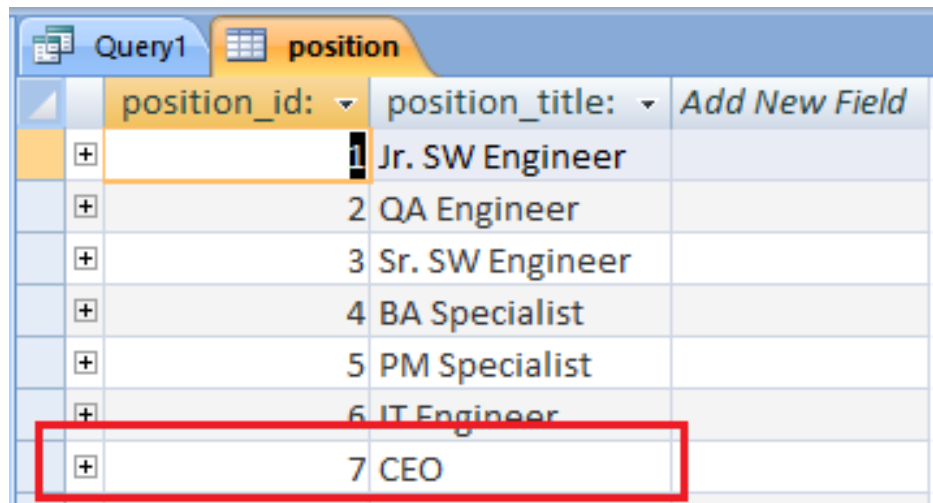
The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

```
INSERT INTO TABLE_NAME [(column1, column2,  
    column3, ... columnN)]
```

```
VALUES (value1, value2, value3, ... valueN);
```

Query 07

INSERT INTO position VALUES (7, 'CEO');



	position_id: ▾	position_title: ▾	Add New Field
+	1	Jr. SW Engineer	
+	2	QA Engineer	
+	3	Sr. SW Engineer	
+	4	BA Specialist	
+	5	PM Specialist	
+	6	IT Engineer	
+	7	CEO	

INSERT INTO position (position_id,
position_title) VALUES (7, 'CEO');

Наполнение таблицы с использованием другой
таблицы / Populate one table using another table

Можно заполнить данные в таблице с помощью оператора `select`, который обращается к другой таблице; если другая таблица имеет набор полей, которые необходимы для заполнения первой таблицы.

You can populate the data into a table through the `select` statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Table population syntax

```
INSERT INTO first_table_name [(column1,  
    column2, ... columnN)]  
SELECT column1, column2, ...columnN  
FROM second_table_name  
[WHERE condition];
```

SQL – UPDATE

Запрос SQL UPDATE используется для изменения существующих записей в таблице. Выражение WHERE можно использовать с запросом UPDATE, чтобы обновить выбранные строки, иначе все строки будут также изменены.

The SQL UPDATE Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

SQL UPDATE syntax

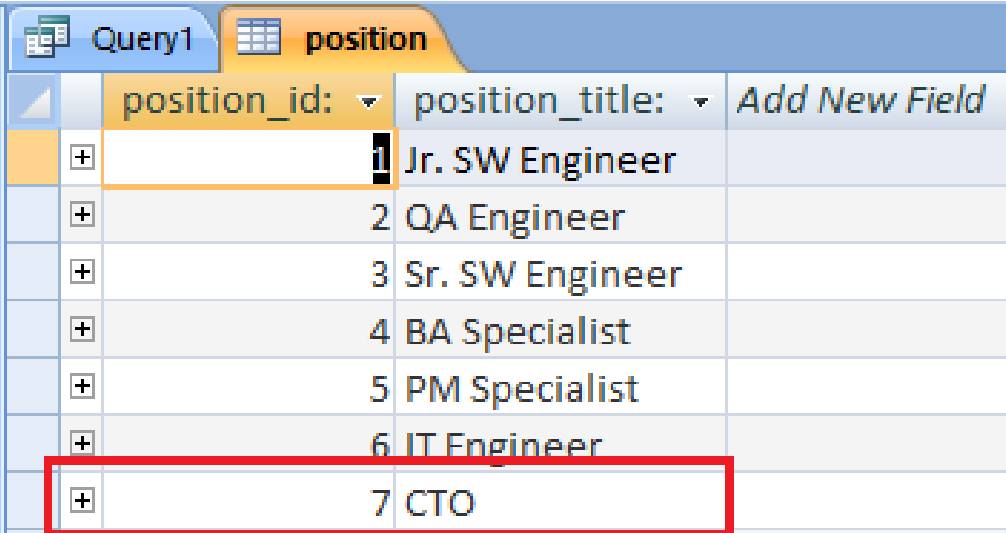
```
UPDATE table_name  
SET  column1 = value1,  
      column2 = value2,  
      ...,  
      columnN = valueN  
WHERE [condition];
```

Query 08

UPDATE position

SET position.position_title = 'CTO'

WHERE position.position_id = 7;



	position_id: ▾	position_title: ▾	Add New Field
+	1	Jr. SW Engineer	
+	2	QA Engineer	
+	3	Sr. SW Engineer	
+	4	BA Specialist	
+	5	PM Specialist	
+	6	IT Engineer	
+	7	CTO	

SQL – DELETE

Запрос SQL DELETE используется для удаления существующих записей из таблицы.

Выражение WHERE можно использовать в запросе DELETE чтобы удалить необходимые строки, иначе все записи будут удалены.

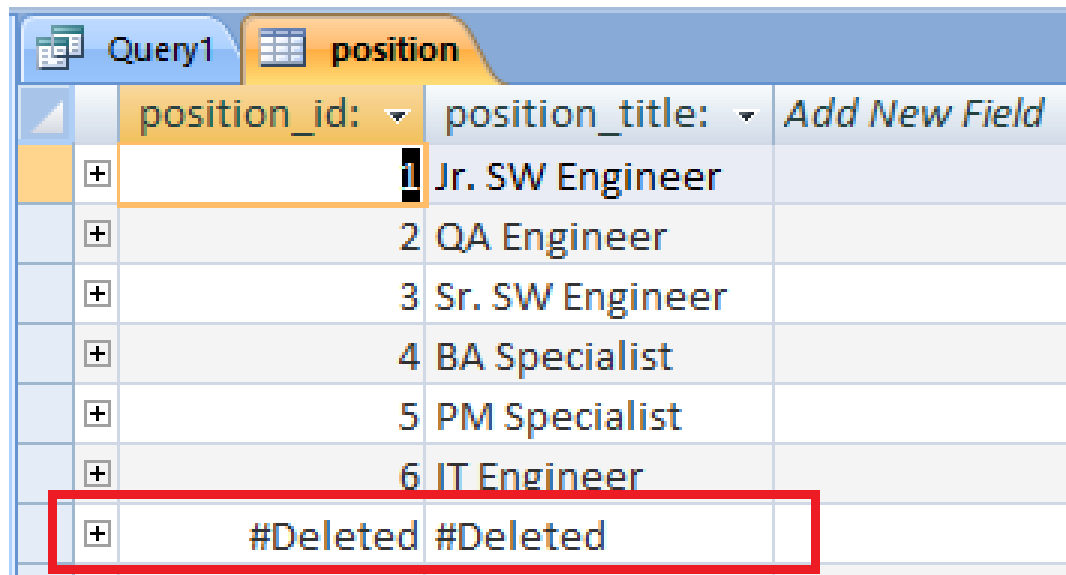
The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

SQL DELETE syntax + Query 09

DELETE FROM table_name WHERE [condition];

DELETE FROM position WHERE position_id = 7;



	position_id: ▾	position_title: ▾	Add New Field
+	1	Jr. SW Engineer	
+	2	QA Engineer	
+	3	Sr. SW Engineer	
+	4	BA Specialist	
+	5	PM Specialist	
+	6	IT Engineer	
+	#Deleted	#Deleted	

To be continued ...