

Організація баз даних

Fundamentals of databases

Копп Андрій Михайлович

Andrii M. Kopp

Асистент кафедри ПІІТУ

Assistant Lecturer of the Department of SEMIT

kopp93@gmail.com @andriikopp

Програма курсу / Course program

1. Вступ / Intro
2. Зберігання даних та файлова структура / Storage and file structure
3. Модель сутність-зв'язок / Entity-relationship model
4. Нормалізація бази даних / Database normalization
5. Реляційна модель / Relational model
6. Мова запитів SQL / SQL query language

- Вступ / Intro
 - Загальні поняття / Overview
 - Архітектура / Architecture
 - Моделі даних / Data models
 - Схеми даних / Data schemas
 - Незалежність даних / Data independence
- Зберігання даних та файлова структура / Storage and file structure
- Модель сутність-зв'язок / Entity-relationship model
 - Основні поняття / Basic concepts
 - Подання діаграм сутність-зв'язок / ER diagram representation
 - Узагальнення та агрегація / Generalization and aggregation
- Нормалізація бази даних / Database normalization
- Реляційна модель / Relational model
 - Правила Кодда / Codd's rules
 - Реляційна модель даних / Relational data model
 - Реляційна алгебра / Relational algebra
 - Перетворення моделі сутність-зв'язок у реляційну модель / ER to relational model
- Мова запитів SQL / SQL query language

Лекція 01: Вступ

Lecture 01: Intro

Дані / Data

- Дані – це сукупність/набір/колекція фактів та уявлень/понять, які можуть бути оброблені для отримання інформації.
- 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 даних / Database

- База даних – це сукупність/набір/колекція пов'язаних даних.
- Database is a collection of related data.

Data (facts) -> Information (conclusion)

 \Rightarrow

Students -> Top
-> Average

Система управління базами даних / Database management system

- Система управління базами даних (СУБД) дозволяє зберігати дані таким чином, що стає легше отримувати, маніпулювати та створювати інформацію
- A database management system (DBMS) stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Характеристики сучасної СУБД / Characteristics of a modern DBMS

- Предметна область / Real-world entity
- Відношення таблиць / Relation-based tables
- Ізоляція даних та застосунку / Isolation of data and application
- Зменшення надмірності / Less redundancy
- Узгодженість / Consistency
- Мова запитів / Query language
- Властивості ACID / ACID properties
- Багатокористувацький та одночасний доступ / Multiuser and concurrent access
- Декілька виглядів / Multiple views
- Безпека / Security

Предметна область / Real-world entity

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

- множина сутностей / set of entities
- властивості / attributes
- відношення / relationships

Відношення таблиць / 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

Ізоляція даних та застосунку / Isolation of data and application

Data – records

Application – DBMS

Metadata (DBMS)

id

name

grade

Data (records)

1

“John Doe”

2

Зменшення надмірності / Less redundancy

Правила нормалізації / Normalization rules

Надмірність / 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

Узгодженість / Consistency

- Узгодженість – це стан, при якому кожне відношення у базі даних залишається узгодженим.
- Consistency is a state where every relation in a database remains consistent.

Metadata

id : int

name : string

grade : int

Data

“A1”

1234

2,5

Мова запитів / Query language

- Отримання, обробка даних
- 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

Властивості ACID / ACID properties

- Атомарність / Atomicity
- Узгодженість / Consistency
- Ізоляція / Isolation
- Довговічність / Durability

Застосовуються до **транзакцій**, які маніпулюють даними в базі даних

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

Транзакцію можна визначити як групу завдань.

Єдине завдання – це мінімальний блок обробки, який більше не можна розділити.

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)

Багатокористувацький та одночасний доступ / Multiuser and concurrent access

- СУБД підтримує багатокористувацьке середовище та дозволяє їм одночасно отримувати доступ до даних і керувати ними.
- DBMS supports multi-user environment and allows them to access and manipulate data in parallel.

Декілька виглядів / Multiple views

- СУБД пропонує декілька виглядів для різних користувачів. Ця функція дозволяє користувачам переглядати базу даних відповідно до їхніх вимог.
- 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

Робота з даними
про постачання
Work with
supply data

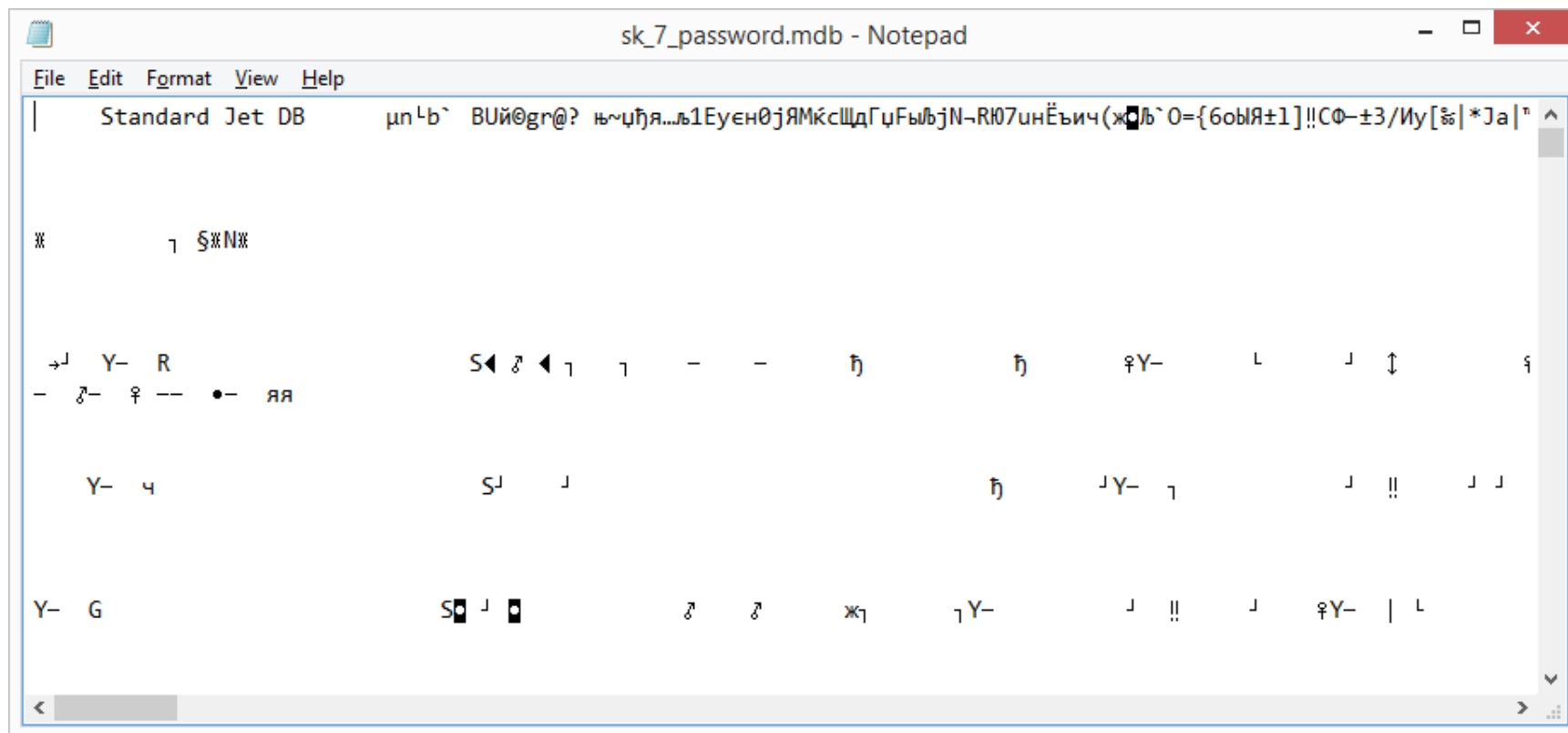
Робота з даними
про залишки
продукції
Work with data
about stock

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	

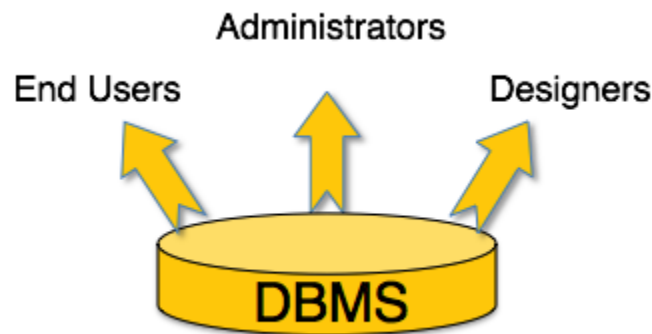
Безпека / Security

- Різні вигляди з різними функціями для різних користувачів / Different views with different features for multiple views
- Обмеження введення та отримання даних / Data entering and retrieving constraints
- СУБД не зберігається на диску як звичайний текстовий файл / DBMS is not saved on the disk as traditional text file



Спроба відкрити файл бази даних як текстовий файл
Trying to open the database file as the text file

Користувачі / Users



- Адміністратори підтримують СУБД і несуть відповідальність за адміністрування бази даних.
- Administrators maintain the DBMS and are responsible for administering the database.

- Дизайнери – це група людей, які безпосередньо працюють над проектуванням бази даних.
- End users are the group of people who actually work on the designing part of the database.
- Кінцеві користувачі – це ті, хто безпосередньо користується перевагами роботи СУБД.
- End users are those who actually reap the benefits of having a DBMS.

Архітектура / Architecture

- Однорівнева / Single-tier

- Багаторівнева / Multi-tier

Пов'язані але незалежні модулі / Related but independent modules

- Модифікувати / Modify
- Доповнювати / Alter
- Змінювати / Change
- Замінити / Replace

1-рівнева архітектура / 1-tier architecture

Користувач використовує єдиний об'єкт, яким є безпосередньо СУБД

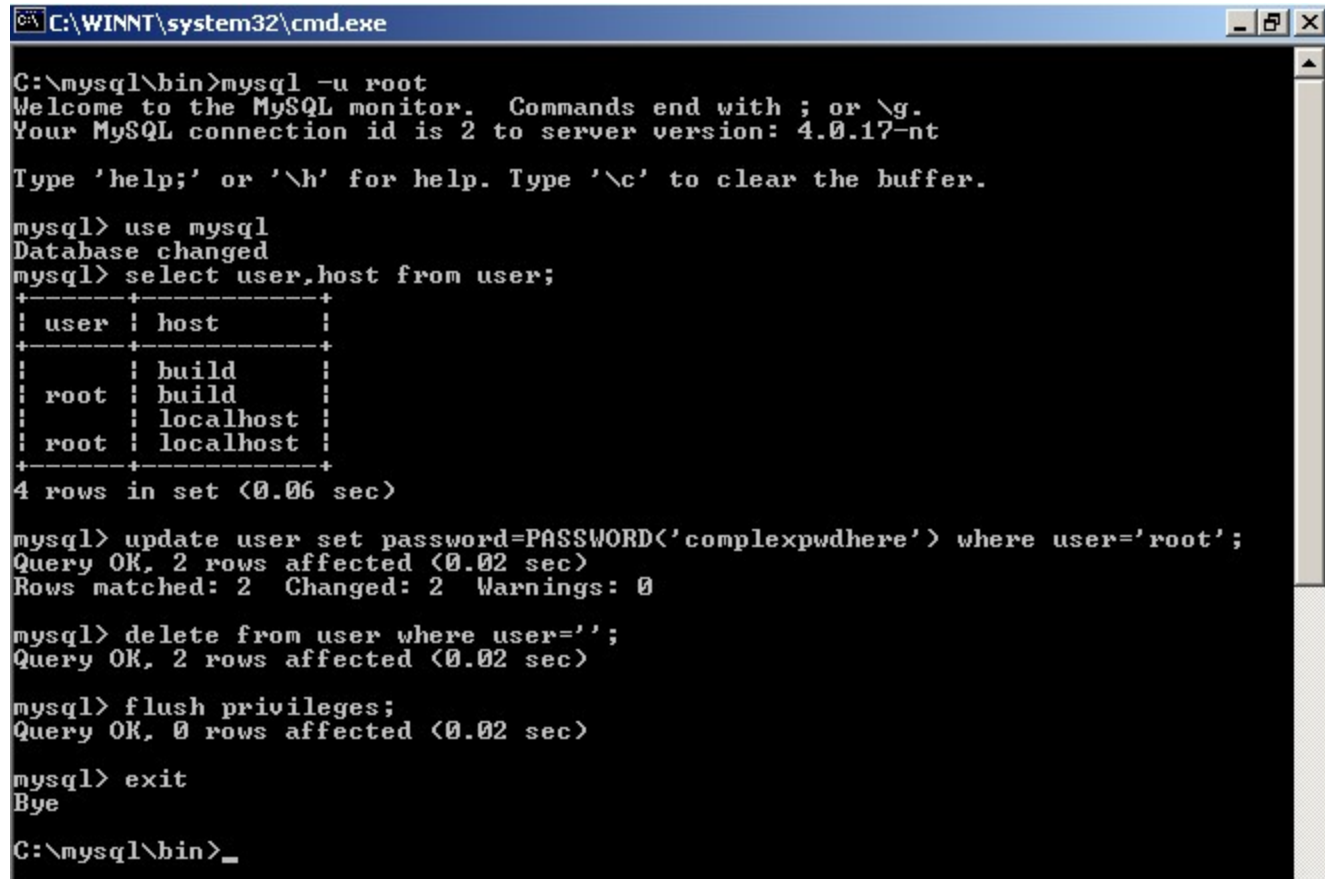
DBMS is the only entity where the user directly sits and uses it

2-рівнева архітектура / 2-tier architecture

СУБД має застосунок, через який можна отримати доступ до 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 |
+-----+-----+
|      |      |
| root | build |
|      | 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)

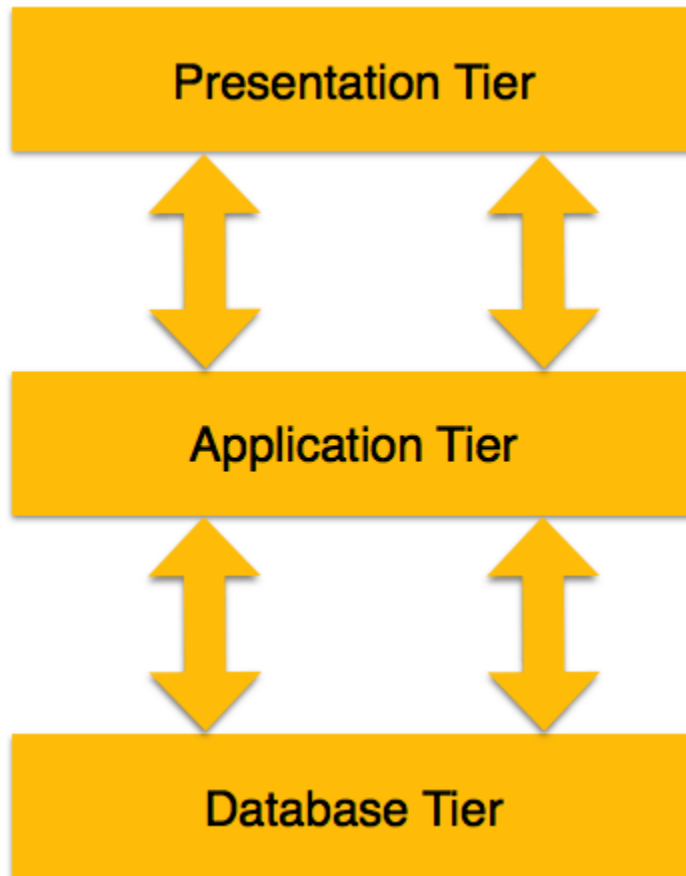
mysql> exit
Bye
C:\mysql\bin>_
```

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

Tier ?

The screenshot displays the phpMyAdmin web interface. On the left, a sidebar shows a list of databases including 'Usuarios', 'Usuarios1', 'VSet', 'VsetAdmin', 'Xss', 'uam', 'uam2', 'ube_db', 'victoria_base', 'vseti', 'world', and a 'New' section with 'City', 'Country', and 'CountryLanguage'. The main panel shows the 'Structure' tab for the 'City' table in the 'world' database. A table lists the 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 table, there are options to 'Check All', 'With selected', and buttons for 'Browse', 'Change', 'Drop', 'Primary', 'Unique', and 'Index'. At the bottom, the 'Information' tab is active, showing 'Space usage' (Data: 266.9 KiB, Index: 42 KiB, Total: 308.9 KiB) and '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/>



← End-users ???

1-tier
architecture
Database
administrators

2-tier
architecture
Database
designers

3-рівнева архітектура / 3-tier architecture

- Рівень даних / Database tier
 - Відношення / Relations
 - Мова запитів / Query language
- Рівень застосунків / Application tier
 - Програми, які мають доступ до бази даних / Programs that access the database
 - Посередник між кінцевим користувачем та базою даних / A mediator between the end-user and the database
- Рівень презентації / Presentation tier
 - Програми надають різні перегляди бази даних / Multiple views of the database are provided by applications
 - Кінцеві користувачі не знають про існування бази даних за межами цього рівня / 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

Моделі даних / Data models

- Визначають логічну структуру бази даних, що моделюється
- Define how the logical structure of a database is modeled
- Яким чином дані зв'язуються між собою / How data is connected to each other
- Яким чином дані обробляються та зберігаються у системі / How data is processed and stored inside the system

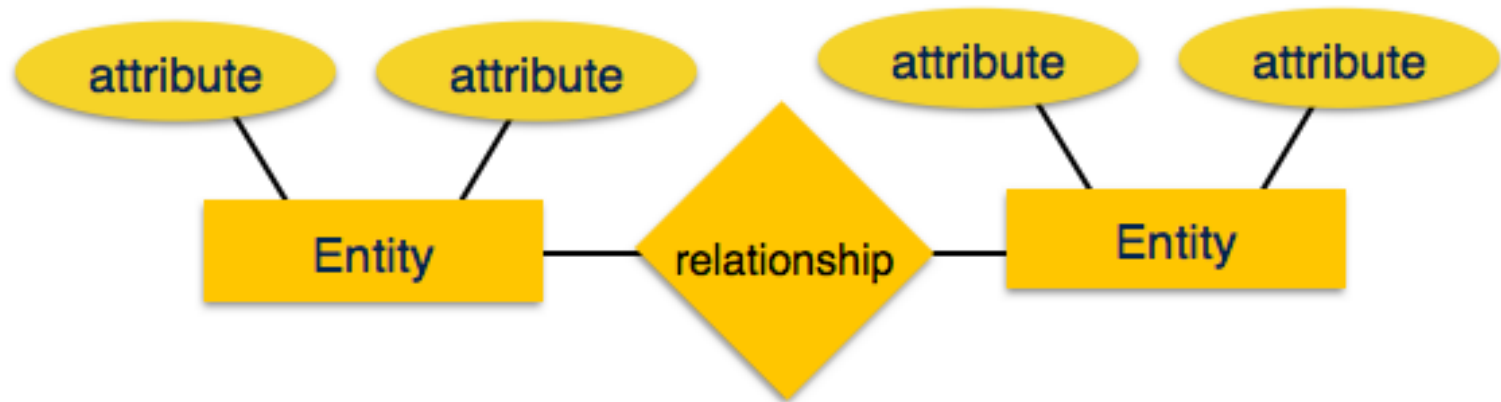
Модель сутність-зв'язок / Entity-relationship model

- Заснована на відомостях про реальні сутності та відношення між ними
- Is based on the notion of real-world entities and relationships among them

Використовується для концептуального проектування бази даних / is used for the conceptual design of a database

ER модель включає / ER model is based on:

- **Сутності** та їх **атрибути** / **Entities** and their **attributes**
- **Відношення** між сутностями / **Relationships** among entities



Кожний атрибут визначається набором значень, який називається **доменом**

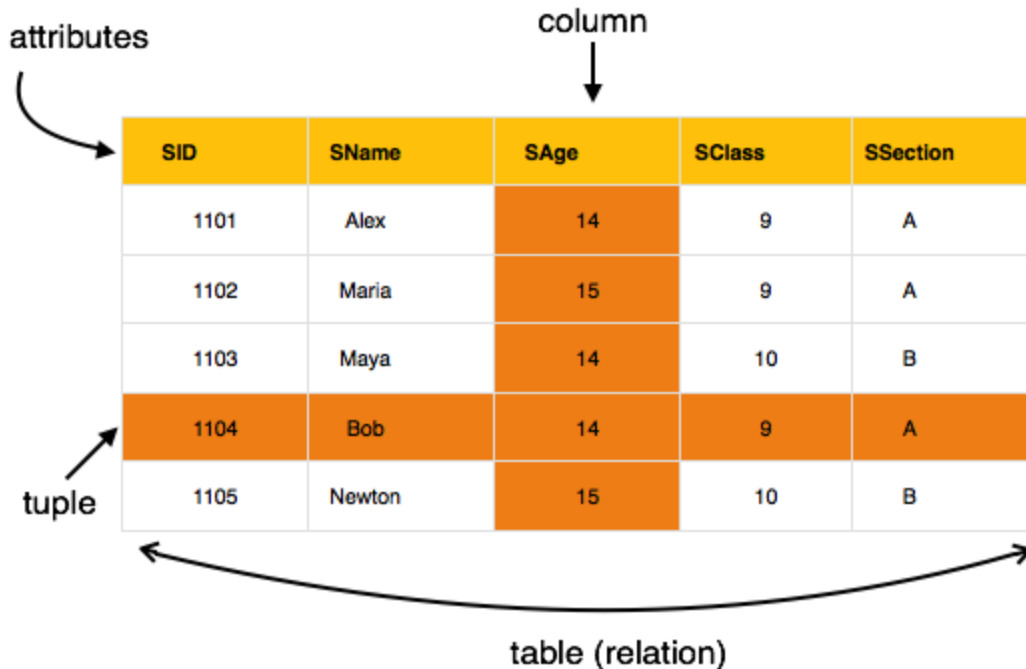
Every attribute is defined by its set of values called **domain**

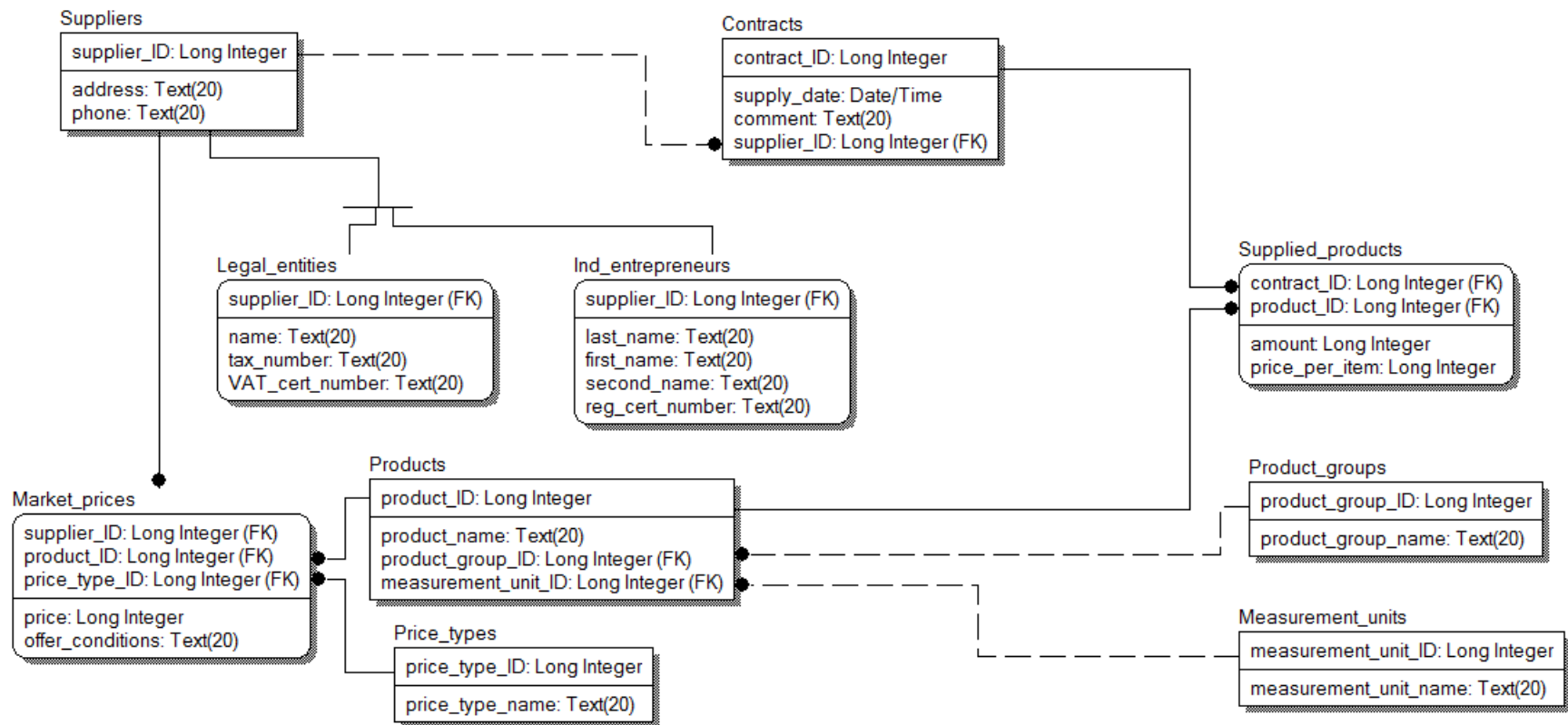
Потужність відношення визначає число асоціацій між двома сутностями
Mapping **cardinalities** define the number of association between two entities

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

Реляційна модель / Relational model

- Визначає таблиці у вигляді n-арного відношення
- Defines table as an n-ary relation





Приклад моделі даних з лабораторного практикуму
The example of the data model from lab classes

Основні особливості реляційної моделі / The main highlights of the relational model:

- Дані зберігаються в таблицях – **відношеннях** / Data is stored in tables called **relations**
- Відношення можуть бути **нормалізовані** / Relations can be **normalized**
- У нормалізованих відношеннях збережені значення є **атомарними** / In normalized relations, values saved are **atomic** values
- Кожен рядок у відношенні містить **унікальне** значення / Each row in a relation contains a **unique** value
- Кожен стовпець у відношенні містить значення одного і того ж **домену** / Each column in a relation contains values from a same **domain**

Нотація IDEF1X / IDEF1X notation

Сутність / Entity

Представлення класу реальних або абстрактних речей

The representation of a class of real or abstract things

Домен / Domain

Іменована множина значень даних одного й того ж типу

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

Атрибут / Attribute

Властивість або характеристика, яка є загальною для деяких або усіх екземплярів сутності

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

Ключ / Key

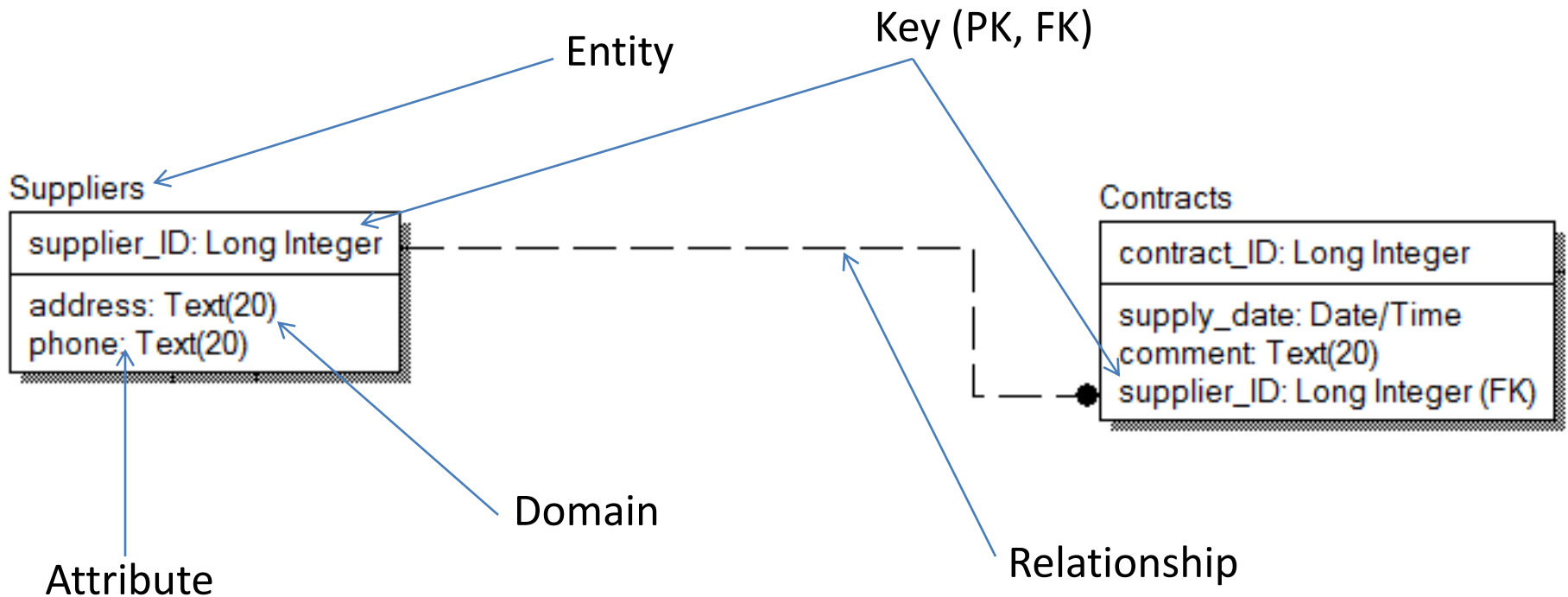
Атрибут або комбінація атрибутів сутності, значення якого однозначно ідентифікує кожен екземпляр сутності. Кожен такий набір являє собою потенційний ключ.

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

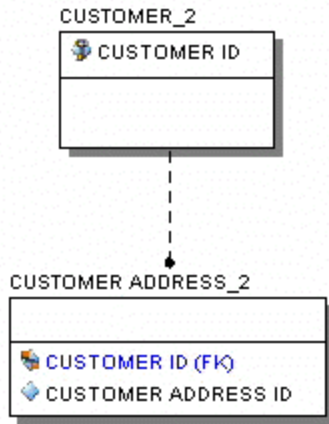
Відношення / Relationship

Асоціація між екземплярами двох сутностей або між екземплярами одної й тієї ж сутності.

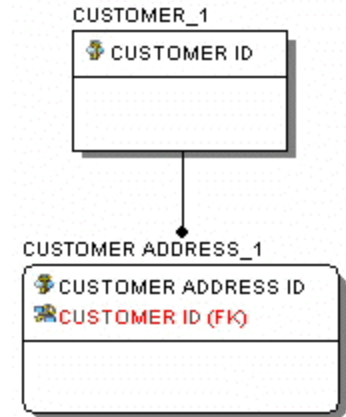
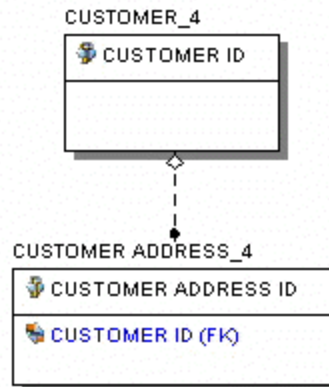
An association between the instances of two entities or between instances of the same entity.



Ідентифікуюче відношення / Identifying relationship



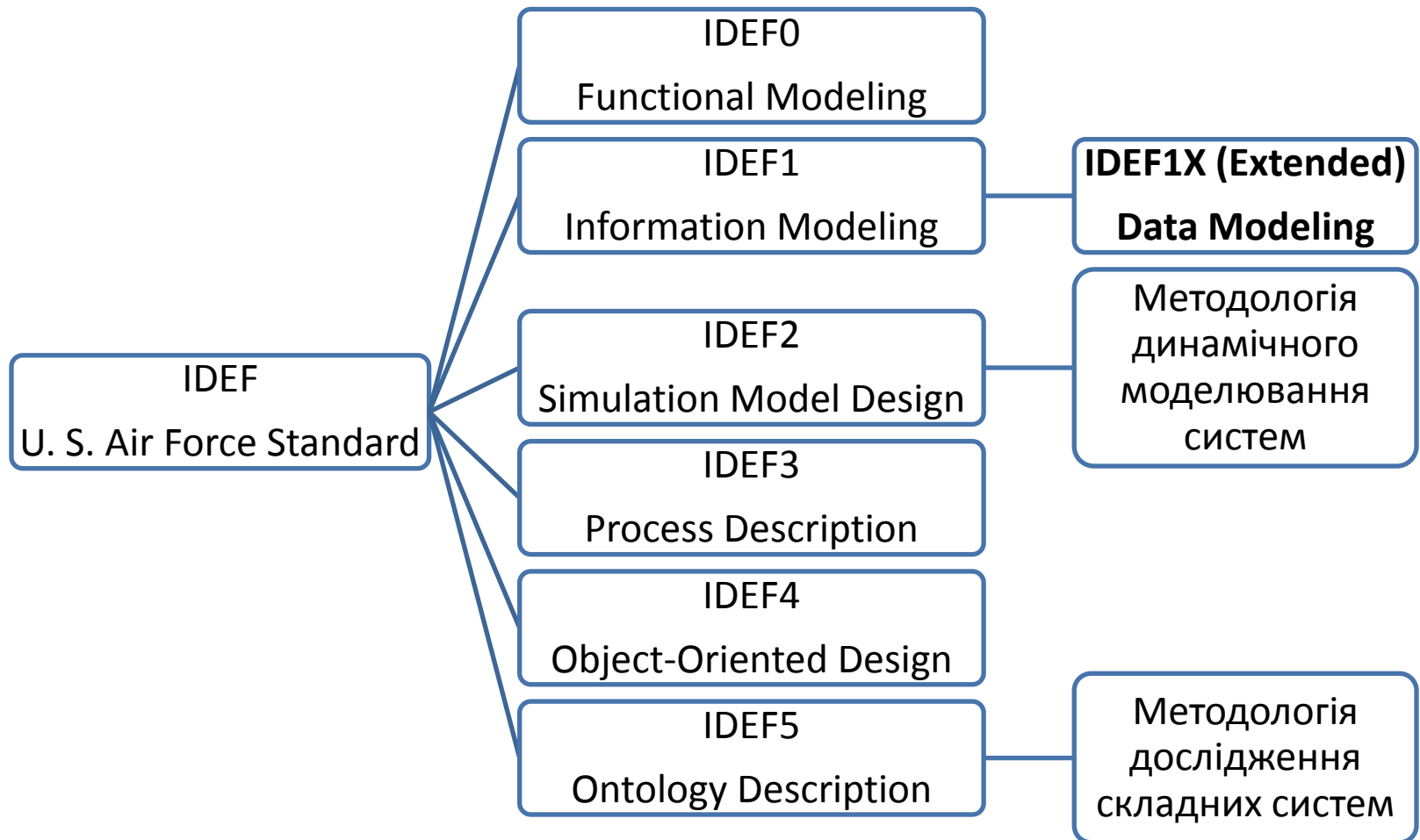
**Non-Identifying, Optional
Relationship**



Неідентифікуюче відношення / Non-Identifying relationship

- обов'язкове / mandatory
- необов'язкове / optional

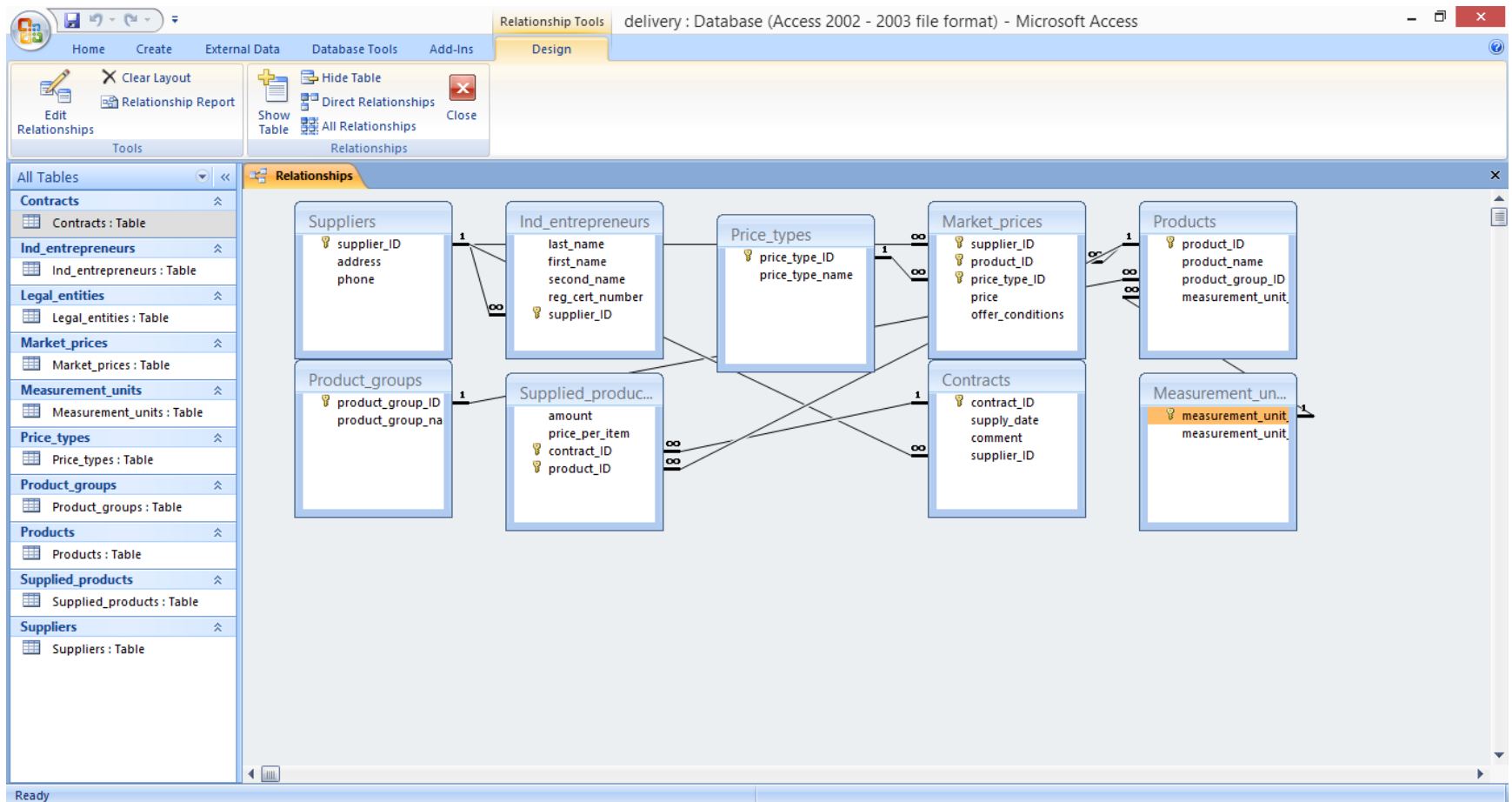
IDEF (Integrated DEFinition)



... IDEF6 – IDEF14

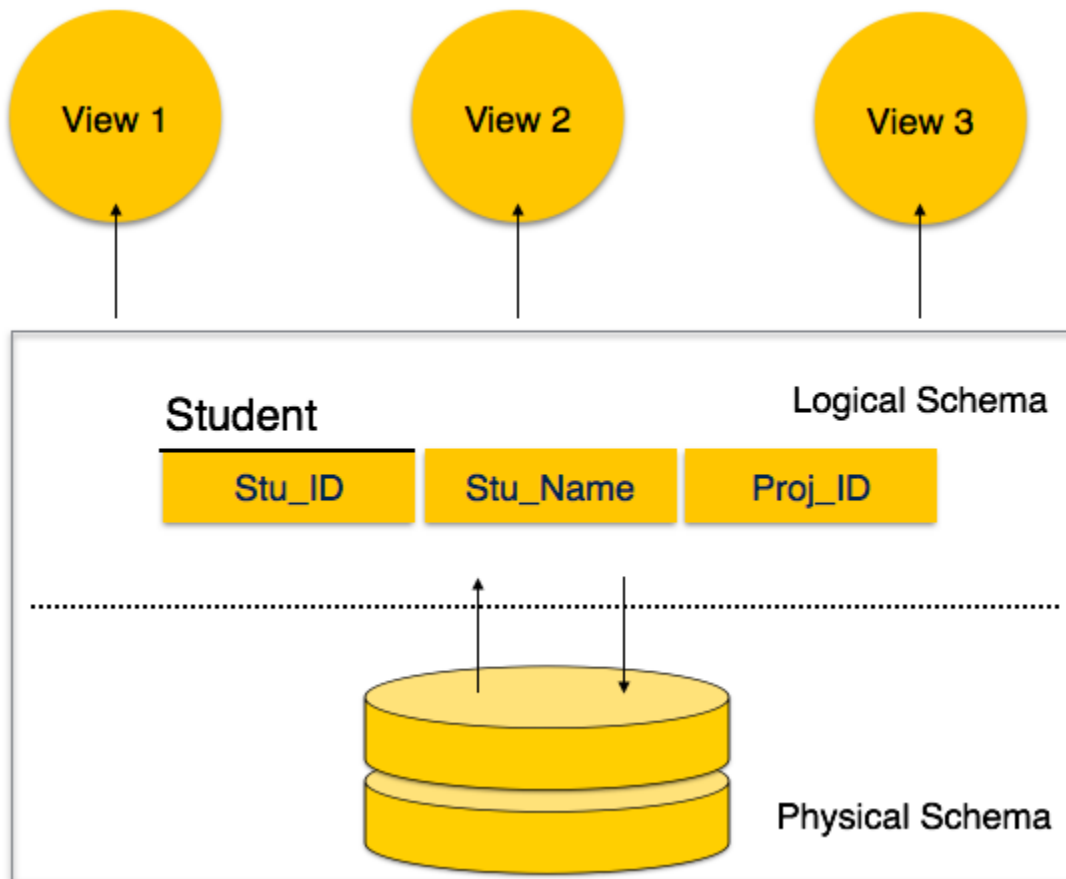
Схема бази даних / Database schema

- Схема бази даних – це “скелет” структури, що представляє логічний вигляд усієї бази даних, визначає яким чином організовані дані та якими відношеннями вони зв’язані, визначає усі обмеження, які накладаються на дані
- 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.



Приклад схеми даних у СУБД Microsoft Access

The example of the database schema in DBMS Microsoft Access



Таблиці, уявлення та обмеження
цілісності
Tables, views, and integrity
constraints

Фактичне зберігання даних
у формі файлів, індексів тощо
Actual storage of data in the
form of files, indices, etc.

Екземпляр бази даних != Схема бази даних

Database instance != Database scheme

Екземпляр бази даних – стан операційної бази даних, що містить дані, у будь-який час.

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

Екземпляри бази даних можуть змінюватись з часом.

Database instances tend to change with time.

Незалежність даних / Data independence

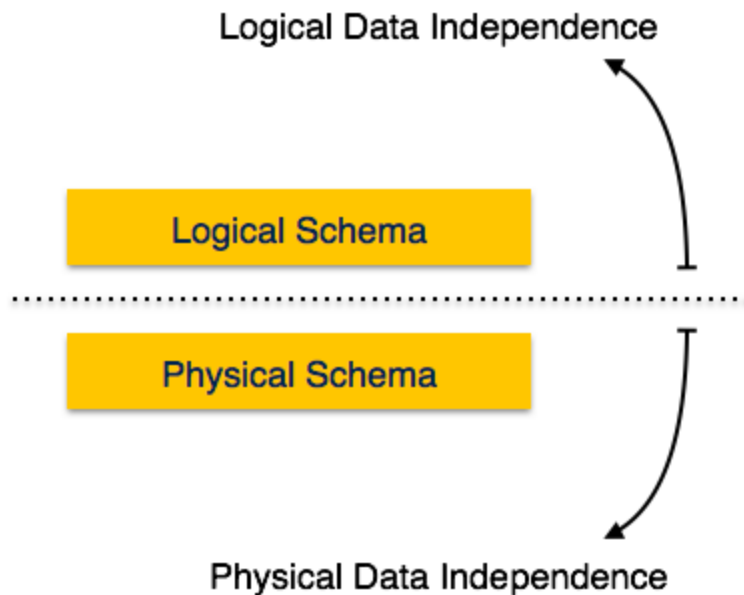
СУБД містить / DBMS contains:
дані користувачів / users' data
метадані / metadata

Логічна незалежність даних – механізм, що відокремлює метадані від фактичних даних, які зберігаються на диску.

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

Фізична незалежність даних – механізм, який забезпечує можливість зміни фізичних даних без впливу на схему бази даних або логічні дані.

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



Внесення змін у формат таблиці не повинно вплинути на дані, які знаходяться на диску
If we do some changes on table format, it should not change the data residing on the disk

Заміна HDD на SSD не повинна вплинути на логічні дані або схеми
Suppose we want to replace hard-disks with SSD – it should not have any impact on the logical data or schemas

Лекція 02: Зберігання даних та
файлова структура

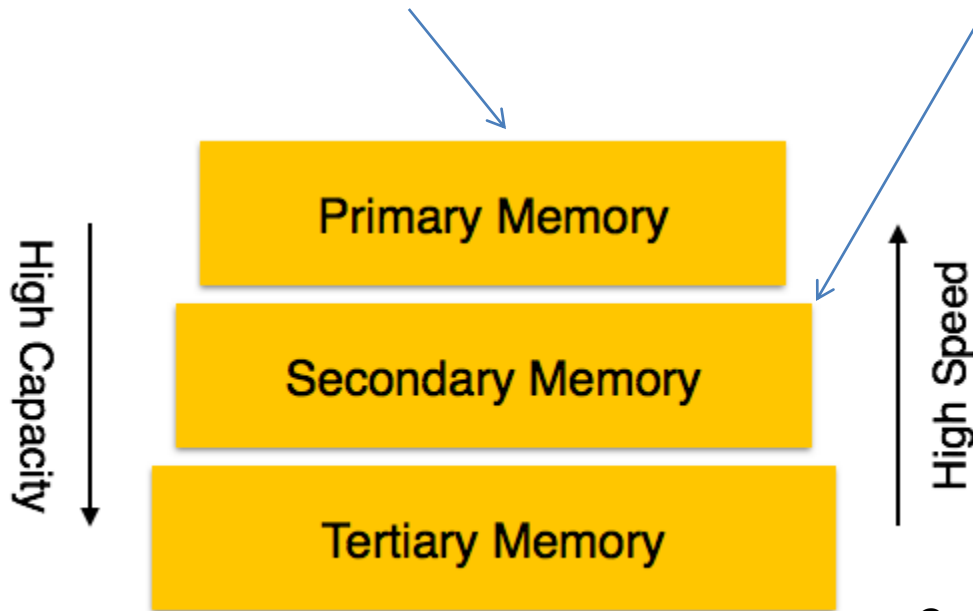
Lecture 02: Storage and file
structure

Система зберігання даних / Storage system

- Бази даних зберігаються у форматах файлів, що містять записи. На фізичному рівні фактичні дані зберігаються на деякому пристрої. Ці пристрої зберігання даних можна розділити на три типи.
- 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.

Первинна пам'ять – безпосередньо доступна для CPU
The memory storage that is directly accessible to the CPU.

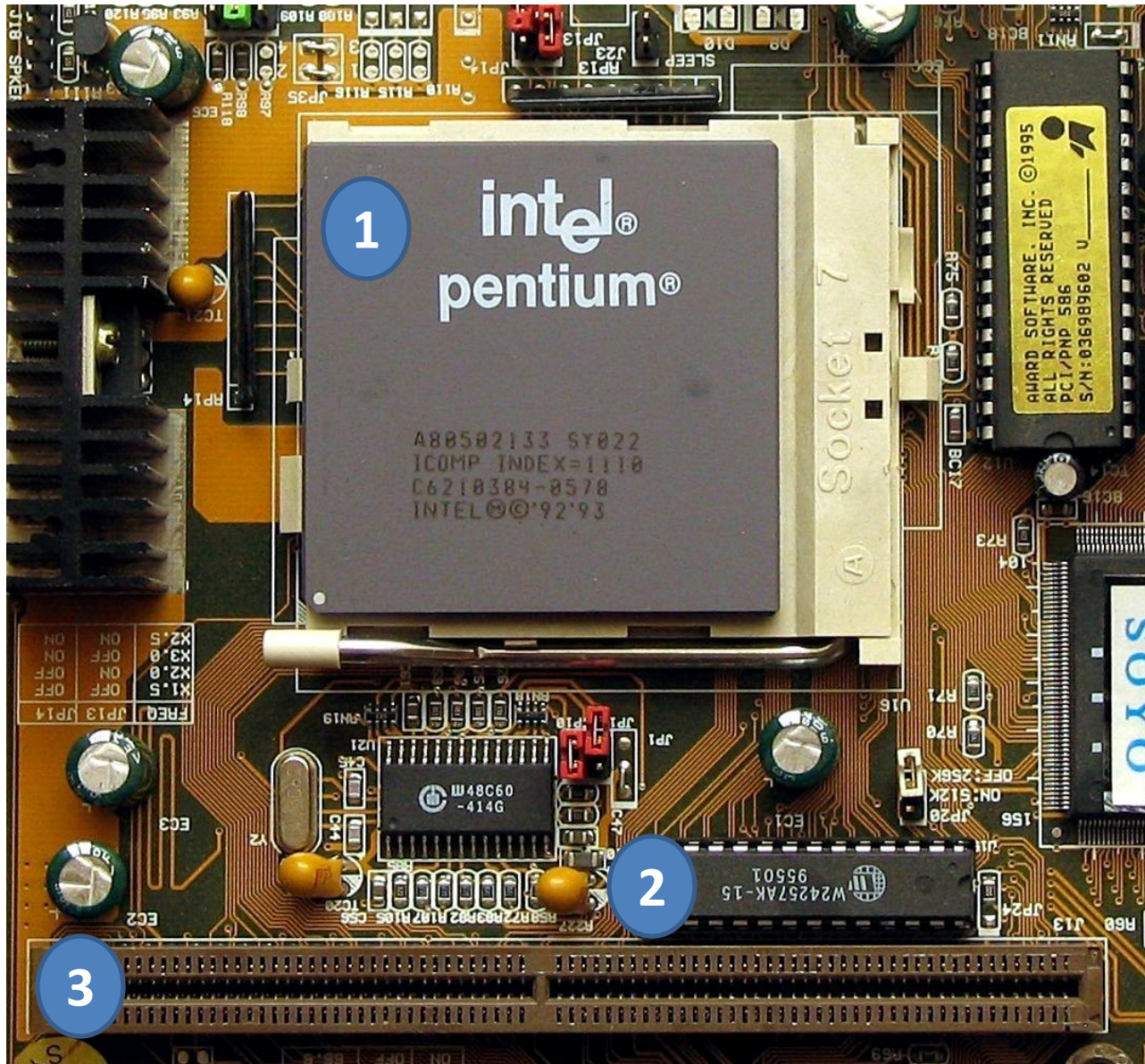
Вторинна пам'ять – використовується для збереження даних з метою їх подальшого використання, або в якості бекапу.
Secondary storage devices are used to store data for future use or as backup.



Стороння пам'ять – використовується для зберігання великих обсягів даних, наприклад, для резервного копіювання всієї системи.
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.

Первинна пам'ять / Primary storage

- Пам'ять, що безпосередньо доступна для CPU, потрапляє до цієї категорії. Внутрішня пам'ять процесора (registers), швидка пам'ять (cache) та основна пам'ять (RAM) безпосередньо доступні для CPU, оскільки всі вони розміщені на материнській платі чи на чіпсеті CPU. Цей накопичувач, як правило, дуже малий, надшвидкий та нестабільний. Основне сховище вимагає постійного джерела живлення, щоб підтримувати його стан. У випадку відключення живлення всі його дані втрачаються.
- 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.



Вторинна пам'ять / Secondary storage

- Вторинні накопичувачі використовуються для зберігання даних для подальшого використання або резервного копіювання. Вторинна пам'ять включає в себе пристрої пам'яті, які не є частиною чіпсета CPU або материнської плати, наприклад, магнітні диски та стрічки, оптичні диски (DVD, CD та ін.), HDD, USB-накопичувачі та 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>

Стороння пам'ять / Tertiary storage

- Сторонній накопичувач використовується для зберігання величезних обсягів даних. Оскільки такі пристрої зберігання знаходяться поза комп'ютером, вони є найбільш повільними. Ці пристрої зберігання даних найчастіше використовуються для резервного копіювання всієї системи.
- 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.



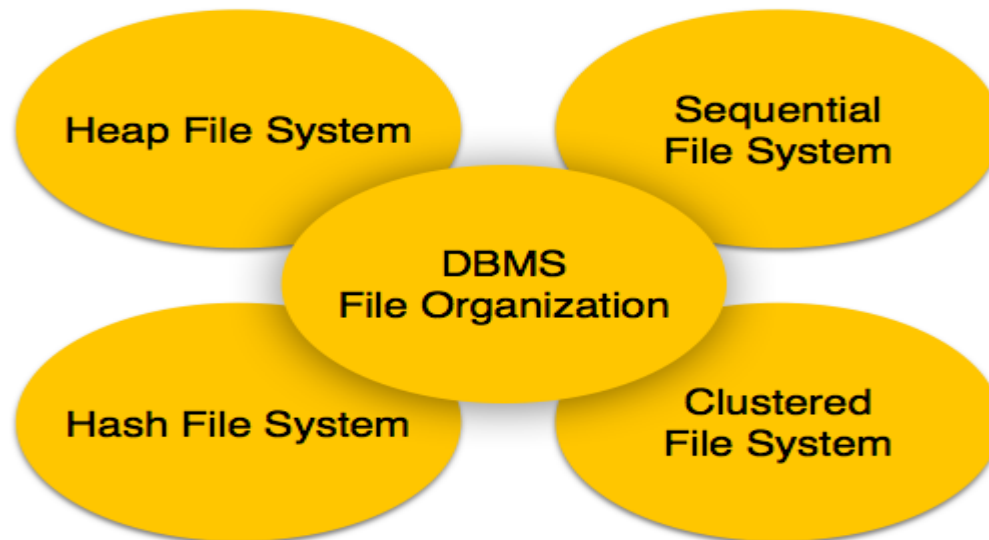
<https://www.bhphotovideo.com/explora/computers/buying-guide/recommended-external-hard-drives-photo-video-and-audio-production>

Файлова структура / File structure

- Пов'язані дані та інформація зберігаються разом у форматах файлів. Файл – це послідовність записів, що зберігаються у двійковому форматі. Дисковий привід відформатовано в кілька блоків, які можуть зберігати записи. Файлові записи відображаються на ці дискові 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.

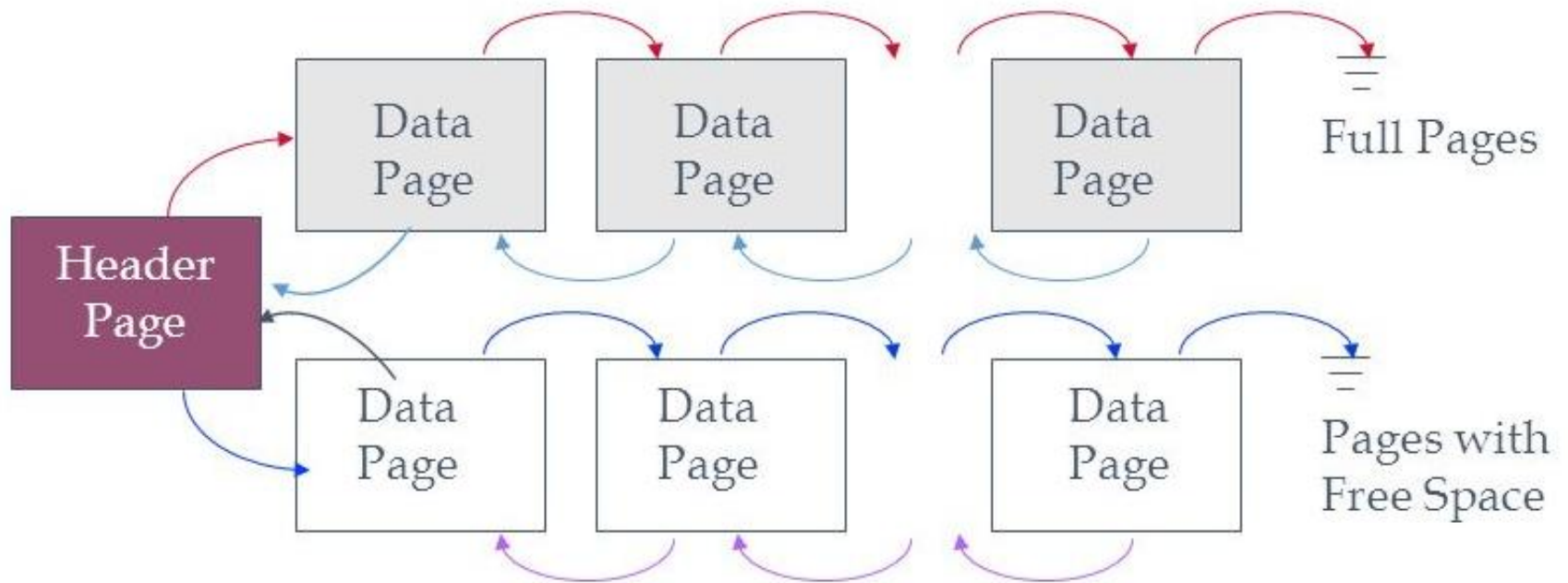
Організація файлів / File organization

- Організація файлів визначає спосіб записування файлів на блоки дисків. Існує чотири типи організації файлів для організації файлових записів.
- File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records.



Організація файлів у купі / Heap file organization

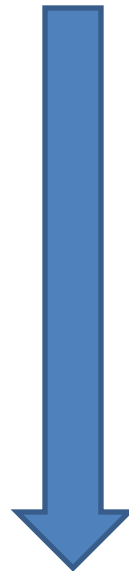
- Коли файл створюється за допомогою купи, операційна система виділяє область пам'яті для цього файлу без будь-яких додаткових облікових даних. Файлові записи можна розміщувати в будь-якій частині виділеної області пам'яті. Відповідальність за керування записами несе програмне забезпечення. Файлова купа не підтримує сортування, упорядкованість чи індексування самотійно.
- 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.



Послідовна організація файлів / Sequential file organization

- Кожний запис файлу містить поле даних (атрибут), щоб однозначно ідентифікувати цей запис. У послідовній організації файлів записи розміщуються у файлі в деякому послідовному порядку на основі унікального ключового поля або ключа пошуку. Практично неможливо зберігати всі записи послідовно у фізичній формі.
- 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 файлі
впорядковані
за допомогою
ключа пошуку

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

Хеш організація файлів / Hash file organization

- Хеш організація файлів використовує обчислення хеш-функцій на деяких полях записів. Вивід хеш-функції визначає розташування блоку на диску, де потрібно розмістити 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).

Кластерна організація файлів / Clustered file organization

- Кластерна організація файлів не вважається корисною для великих баз даних. У цьому механізмі пов'язані записи з одного або декількох відношень зберігаються в тому ж дисковому блоці, тобто упорядковування записів не ґрунтується на первинному ключі або ключі пошуку.
- 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.

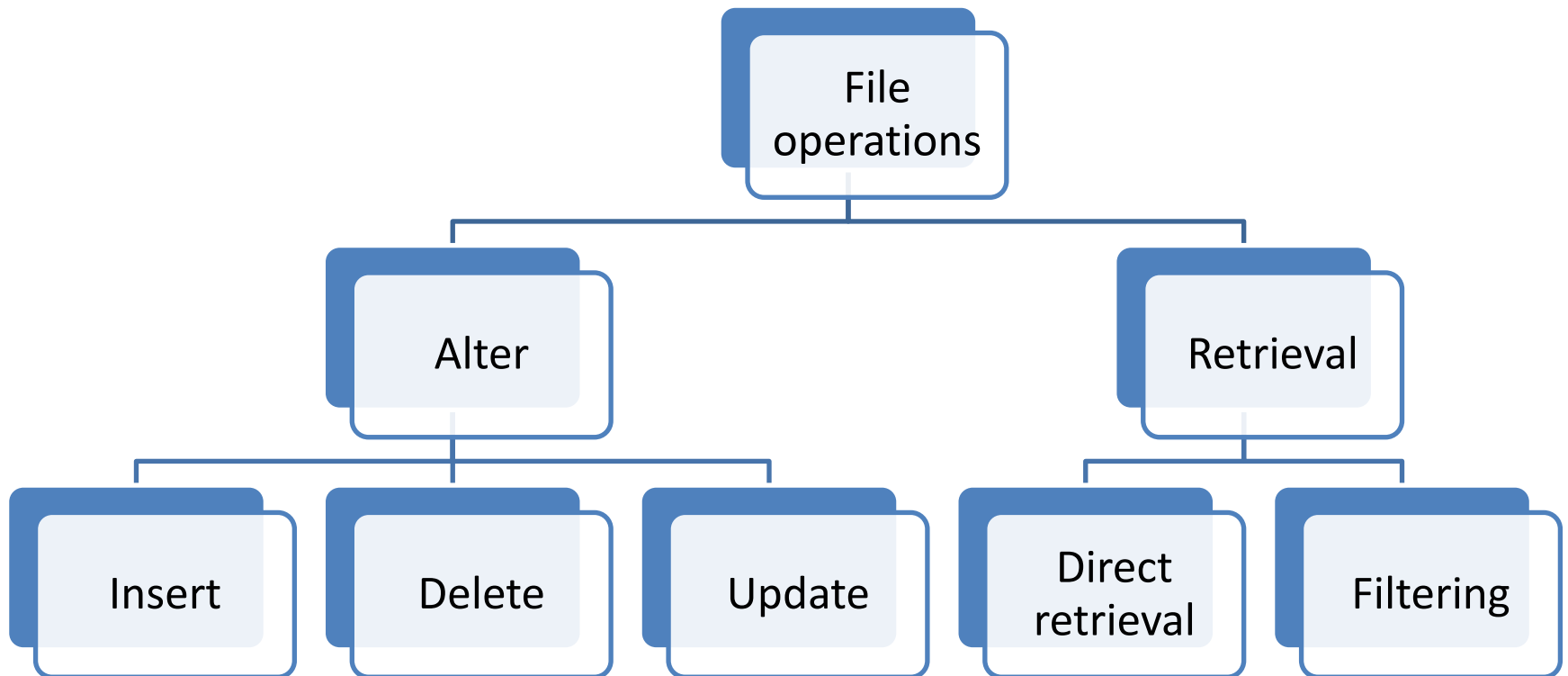
CLUSTER KEY



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

Файлові операції / File operations



Відкриття – файл може бути відкритий в одному з двох режимів: режим **читання** або режим **запису**. У режимі читання операційна система не дозволяє нікому змінювати дані. Інакше кажучи, дані лише для читання. Файли, відкриті в режимі читання, можуть бути розподілені між декількома сутностями. Режим запису дозволяє змінювати дані. Файли, відкриті в режимі запису, можна читати, але їх не можна поділити.

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.

Переміщення – кожен файл має покажчик файлу, який показує поточну позицію, де дані потрібно читати чи записувати. Цей покажчик можна відповідно скорегувати. Використовуючи операцію пошуку, покажчик можна пересувати вперед або 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.

Читання – за замовчуванням, коли файли відкриваються в режимі читання, покажчик файлу вказує на початок файлу. Існують варіанти, за яких користувач може повідомити операційній системі, де знаходити покажчик файлу під час відкриття файлу. Зчитуються наступні до покажчика файлу дані.

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.

Запис – користувач може вибрати, щоб відкрити файл у режимі запису, який дає їм змогу редагувати його вміст. Це може бути видалення, вставка або модифікація. Показчик файлів може знаходитися під час відкриття або може бути динамічно змінений, якщо операційна система це дозволяє.

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.

Закриття – це найважливіша операція з точки зору операційної системи. Коли створюється запит на закриття файлу, операційна система:

- видалляє всі замки (якщо в режимі спільного використання);
- зберігає дані (якщо вони змінені) на вторинні носії інформації;
- випускає всі буфери та обробники файлів, пов'язані з 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.

Лекція 03: Модель сутність-зв'язок

Lecture 03: Entity-relationship model

Модель ER / ER model

- Модель ER визначає концептуальний вигляд бази даних. Вона визначає реальні сутності та асоціації поміж них. На рівні перегляду ER модель вважається прийнятним варіантом для проектування баз даних. Представлена Пітером Ченом у 1976 році.
- 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.

Сутність / Entity

- Сутність може бути реальним об'єктом, живим або неживим, який можна легко ідентифікувати.
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable.
- Набір сутностей – сукупність сутностей одного й того ж типу.
- An entity set is a collection of similar types of entities.

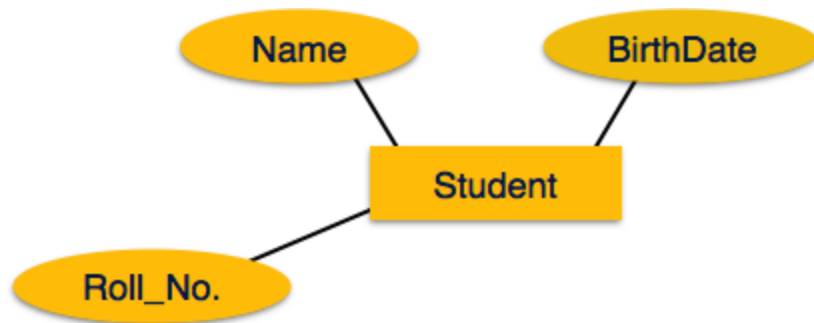
Атрибути / Attributes

- Об'єкти представляються за допомогою їх властивостей, які називаються атрибутами. Всі атрибути мають значення.
- Entities are represented by means of their properties, called attributes. All attributes have values.
- Існує **домен** або діапазон значень, які можуть бути призначені для атрибутів.
- There exists a **domain** or range of values that can be assigned to attributes.

Сутності представляються за допомогою прямокутників.

Прямокутники називаються сутністю, яку вони представляють.

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



Атрибути – це властивості сутностей. Атрибути подаються за допомогою еліпсів. Кожен еліпс представляє один атрибут і безпосередньо пов'язаний з його сутністю (прямокутником).

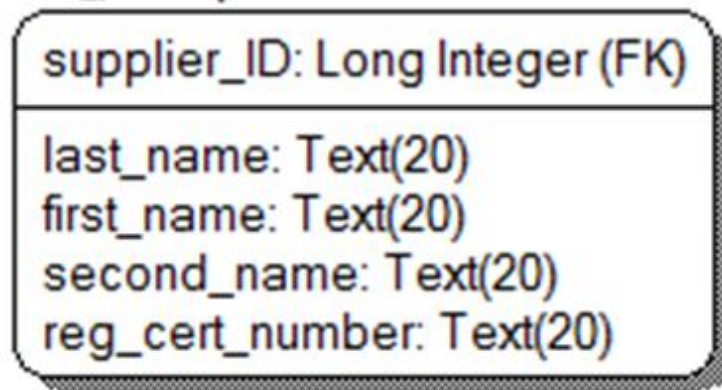
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

Подання моделей даних / Data models representation

Entity

Attributes

Ind_entrepreneurs



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

Типи атрибутів / Types of attributes

Прості атрибути – це атомарні значення, які неможливо розділити далі.

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

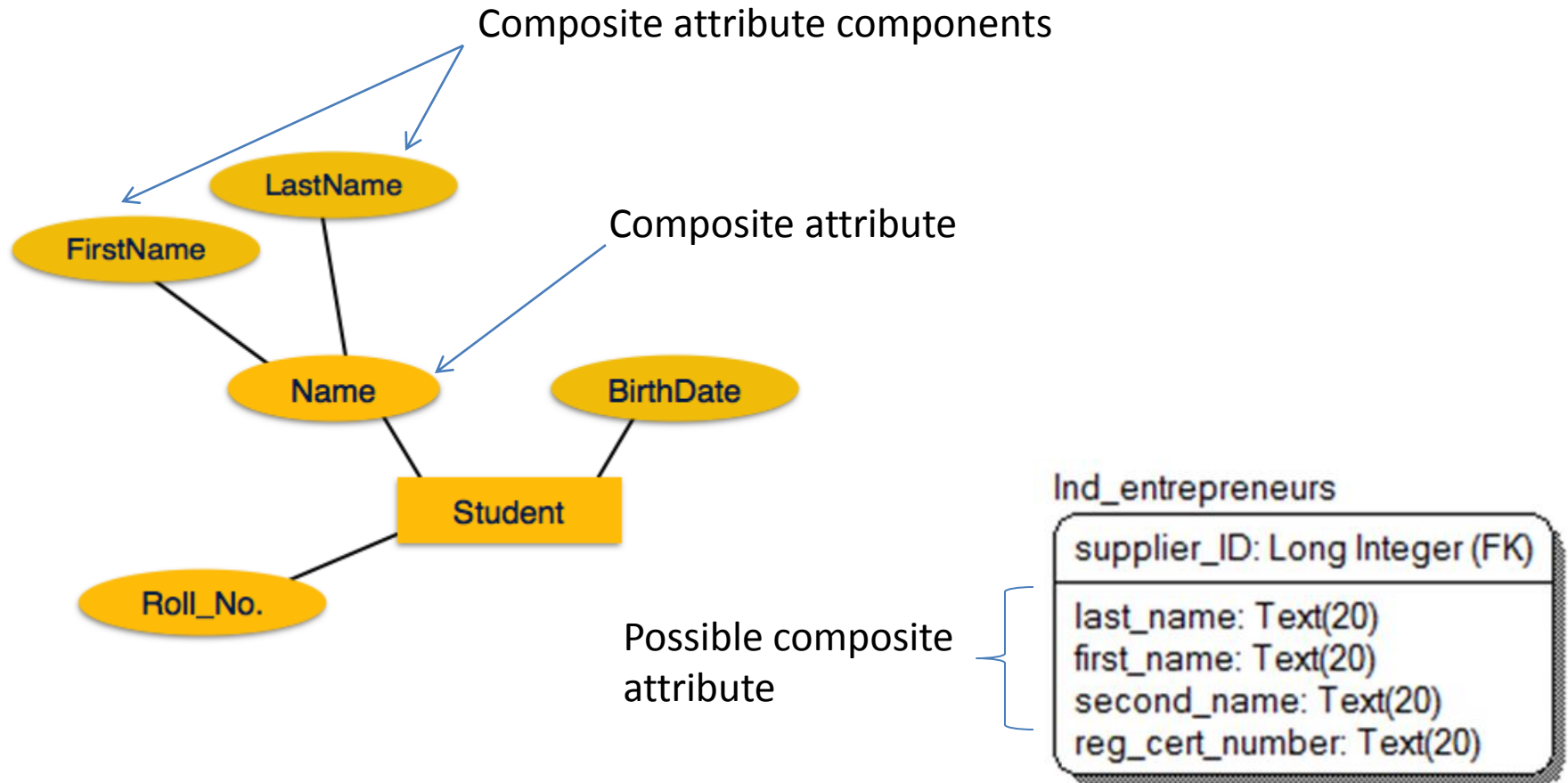
Student ID	First Name	Phone Number
1	Jonh Doe	+38 012 345 6789
...
999	Petrenko Petro Petrovych	+38 098 765 4321

- **Складені атрибути** складаються з більш ніж 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

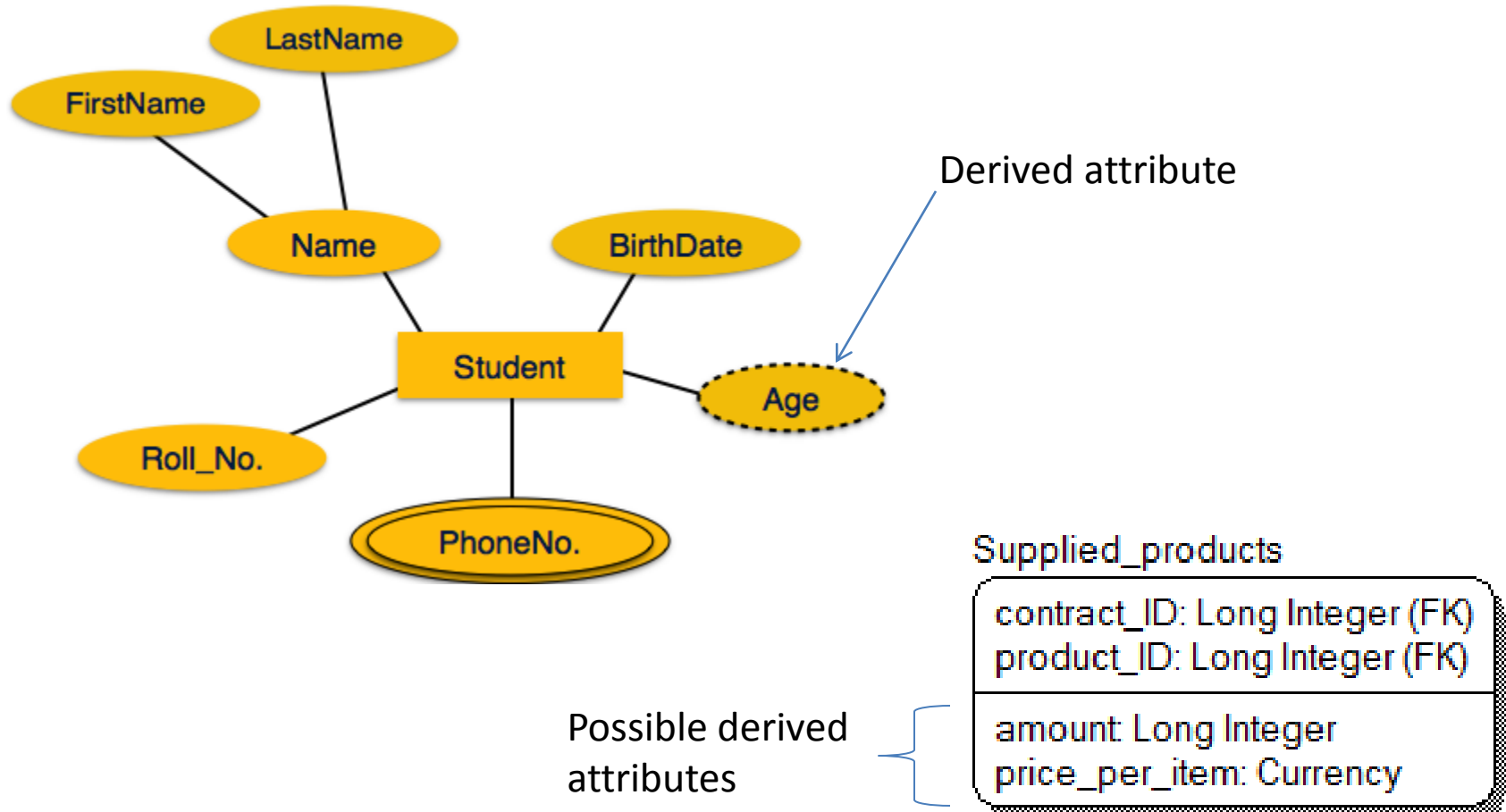
- Наприклад, повне ім'я студента може мати ім'я first_name і last_name.
- For example, a student's complete name may have first_name and last_name.

Подання складених атрибутів / Composite attributes representation



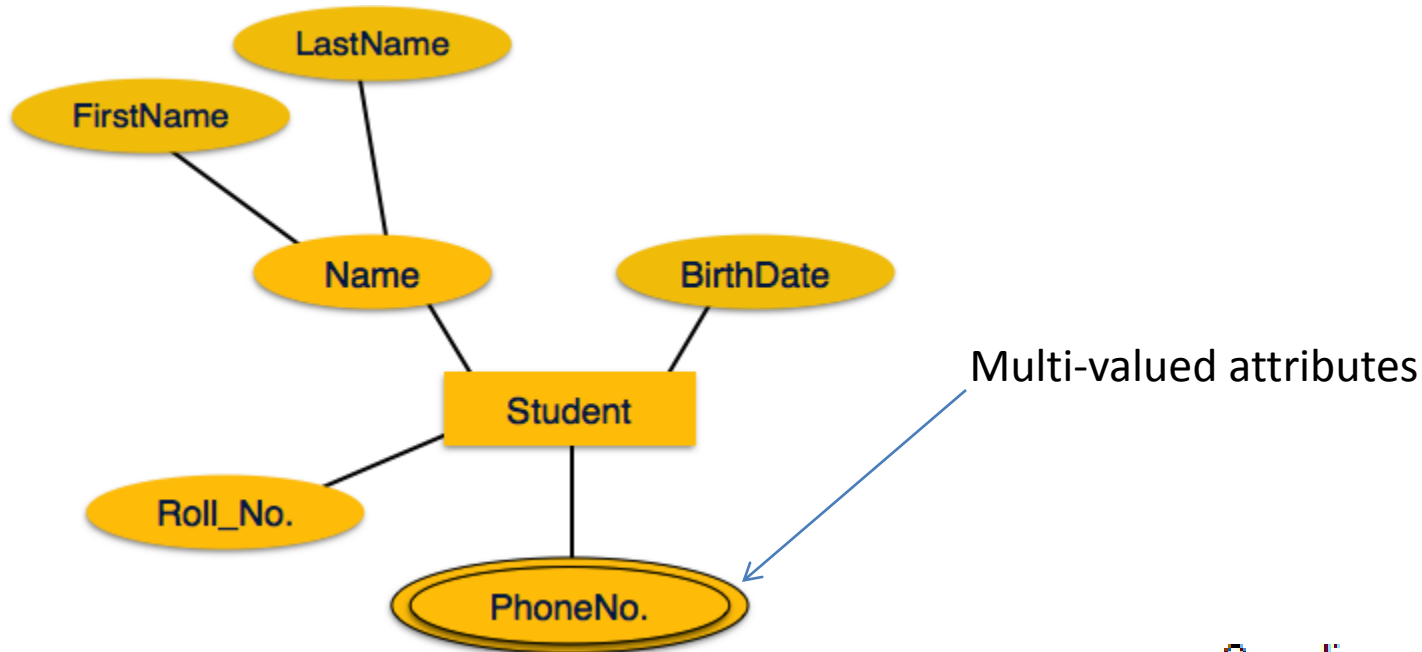
- **Похідні атрибути** – це атрибути, які не існують у фізичній базі даних, але їх значення походять від інших атрибутів, присутніх у базі даних.
- **Derived attributes** are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
- Наприклад, середня зарплата у відділі не повинна зберігатися безпосередньо в базі даних, замість цього вона може бути виведена.
- For example, average salary in a department should not be saved directly in the database, instead it can be derived.

Подання похідних атрибутів / Derived attributes representation



- **Однозначні атрибути** містять одне значення. **Багатозначні атрибути** можуть містити декілька значень.
- **Single -value attributes** contain single value. **Multi-value attributes** may contain more than one values.
- Наприклад – Social_Security_Number. Наприклад, людина може мати більше одного номера телефону, адреси електронної пошти та інше.
- For example – Social_Security_Number. For example, a person can have more than one phone number, email address, etc.

Подання багатозначних атрибутів / Multi-valued attributes representation



Possible multi-valued
attributes

Suppliers

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

Ці типи атрибутів можуть комбінуватись у такий спосіб:

- прості однозначні атрибути;
- прості багатозначні атрибути;
- складені однозначні атрибути;
- складені багатозначні атрибути.

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.

Ключові атрибути / Keys

Ключ – це атрибут або набір атрибутів, які однозначно ідентифікують сутність поміж набором сутностей.

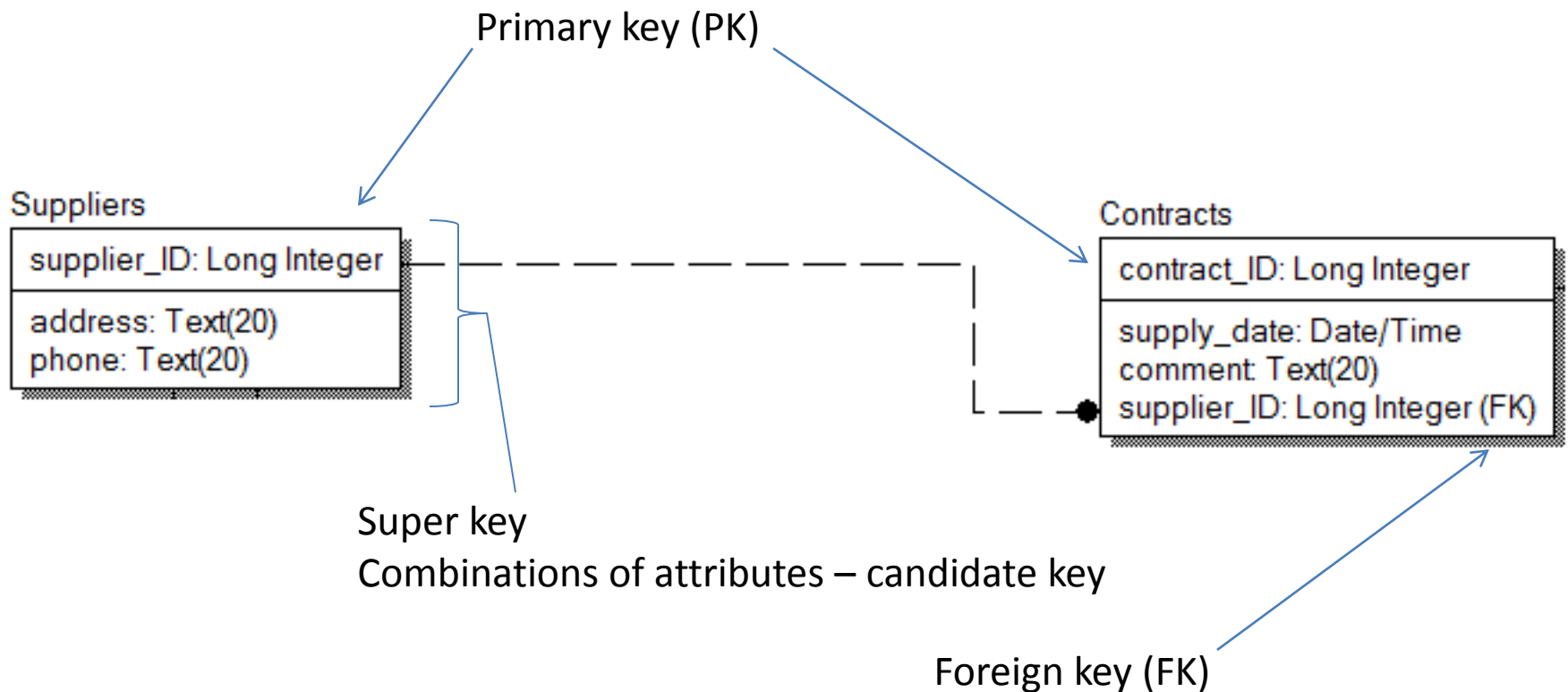
Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

Наприклад, номер залікової книжки студента ідентифікує його/її серед усіх студентів.

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

- **Супер-ключ** – набір атрибутів (один або декілька), який сукупно ідентифікує сутність у наборі сутностей.
- **Super Key** – A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Потенційний ключ** – мінімальний супер-ключ називається потенційним ключем. Набір сутностей може містити декілька потенційних ключів.
- **Candidate Key** – A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Первинний ключ** – це один з потенційних ключів, обраних дизайнером баз даних для того, щоб однозначно ідентифікувати набір сутностей.
- **Primary Key** – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Подання ключових атрибутів / Key attributes representation



Відношення / Relationship

- Асоціація поміж сутностями називається відношенням.
- The association among entities is called a relationship.
- Наприклад, співробітник працює на кафедрі, студент зараховується до курсу. Тут, «працює» та «зараховується» – відношення.
- For example, an employee works_at a department, a student enrolls in a course. Here, works_at and Enrolls are called relationships.

Набір відношень / Relationship set

- Множина відношень одного й того ж самого типу називається набором відношень. Як і сутності, відношення теж можуть мати атрибути. Ці атрибути називаються описовими 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.

Ступінь відношення / Degree of relationship

Кількість сутностей, що беруть участь у відношенні, визначає ступінь відношення:

- бінарне = ступінь 2;
- тернарне = ступінь 3;
- n-арне = ступінь 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.

Потужності відображення / Mapping cardinalities

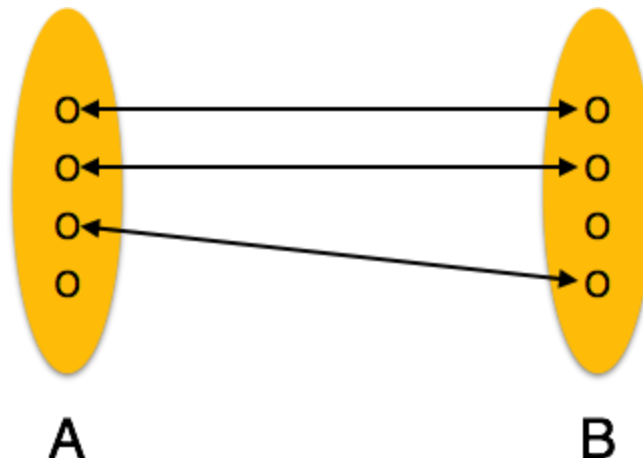
Потужність визначає кількість сутностей в одному наборі сутностей, які можуть бути пов'язані з кількістю сутностей іншого набору через набір відношень.

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 до одного / One-to-one

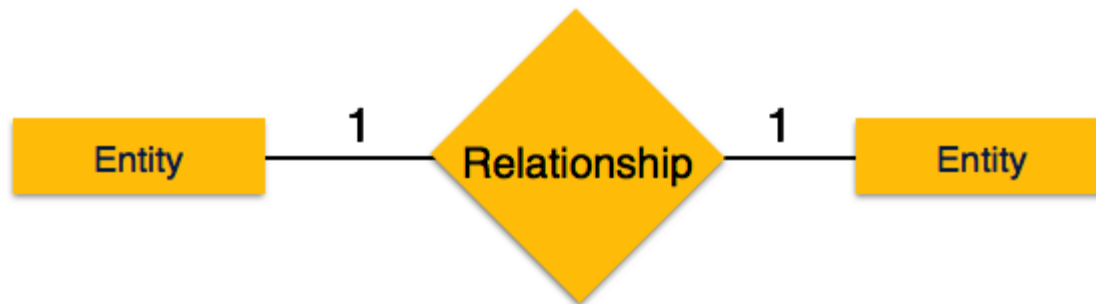
Одна сутність з набору сутностей A, може бути пов'язана не більше ніж з однією сутністю з набору сутностей B, і навпаки.

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

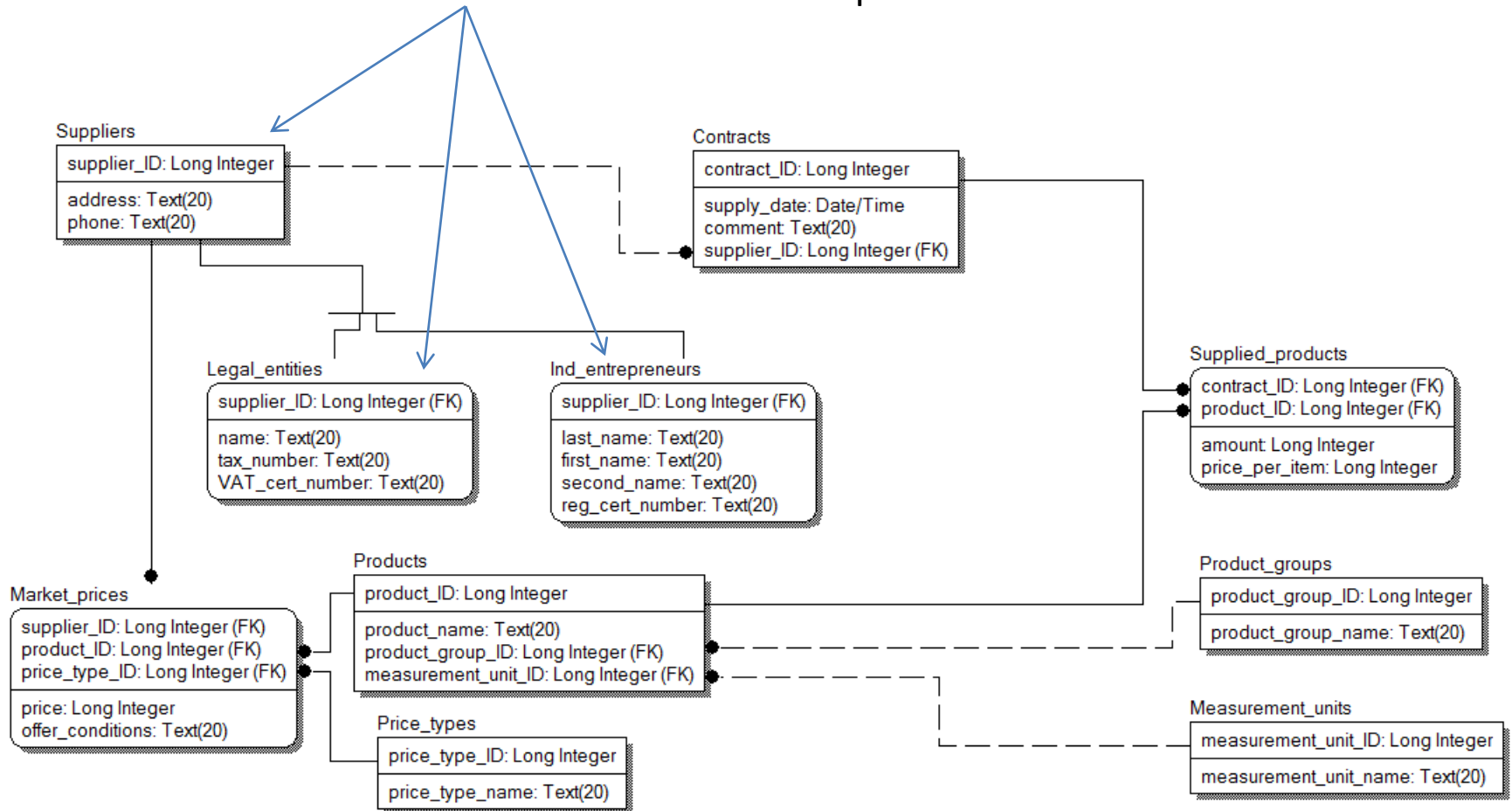


Один-до-одного – лише один екземпляр сутності пов'язаний з відношенням, він позначається як "1:1". Наступне зображення показує, що лише один екземпляр кожної сутності повинен бути пов'язаний з відношенням.

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



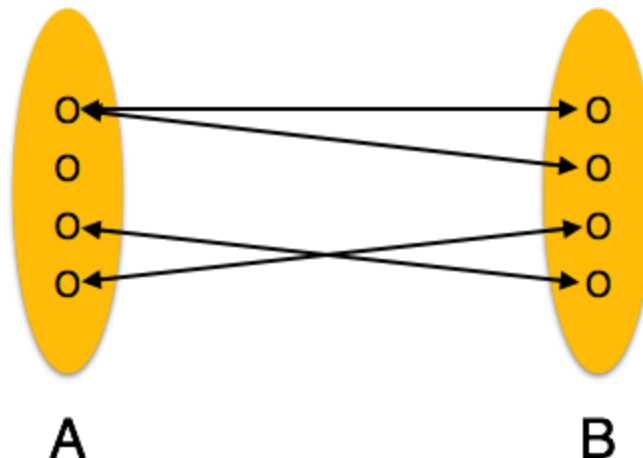
Each supplier entry should be provided with the additional information depending on its type.

Для кожного постачальника повинна бути визначена додаткова інформація в залежності від його типу.

Один до багатьох / One-to-many

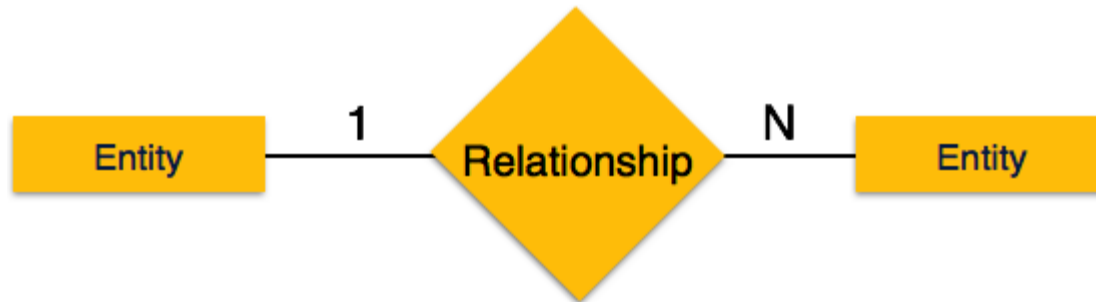
Одна сутність з набору сутностей A може бути асоційована з кількома сутностями з набору сутностей B, однак сутність з набору сутностей B може бути асоційована не більше ніж з однією сутністю.

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.

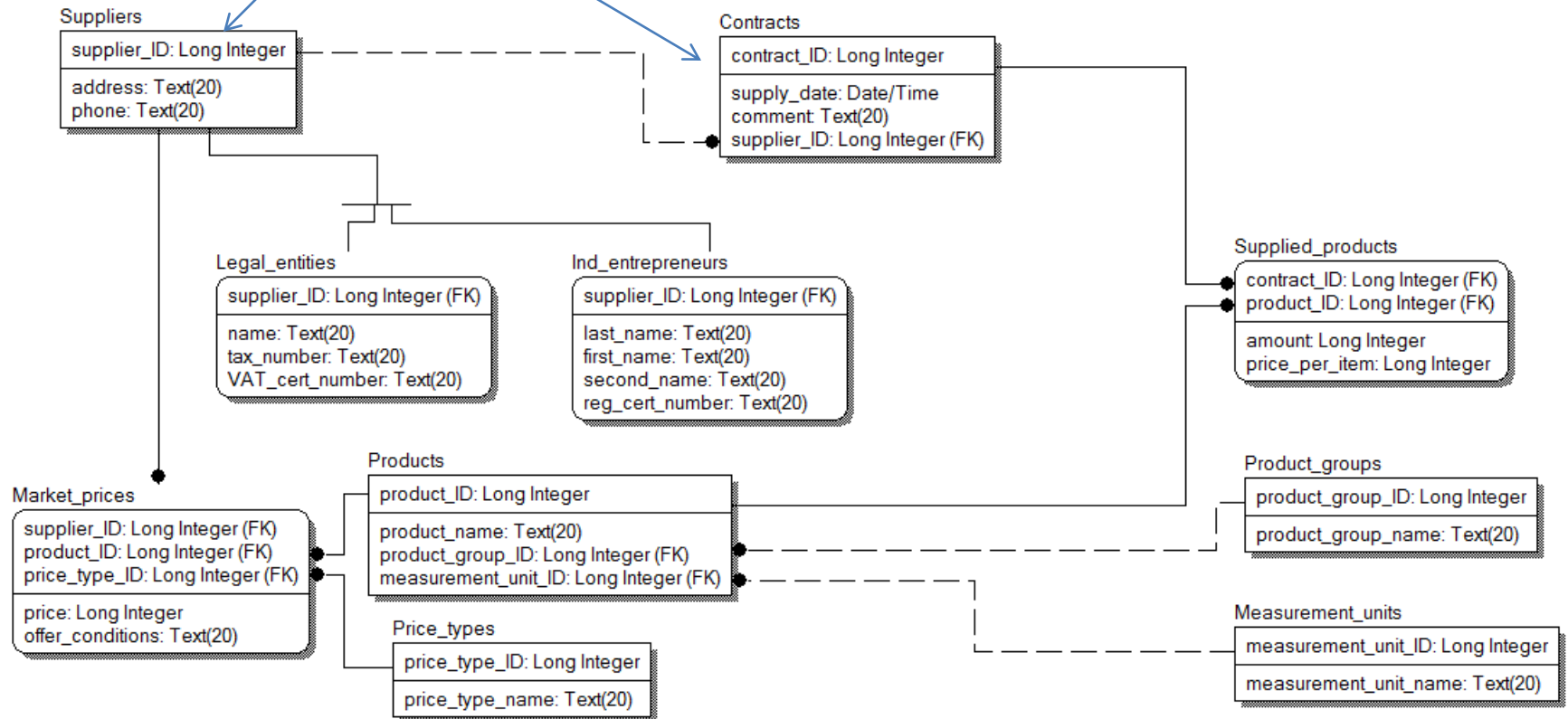


Один-до-багатьох – якщо більше одного екземпляра сутності пов'язано з відношенням, він позначається як '1:N'. Наступне зображення відображає, що лише один екземпляр сутності зліва і більше, ніж один екземпляр сутності справа, може бути пов'язаний з цим відношенням.

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



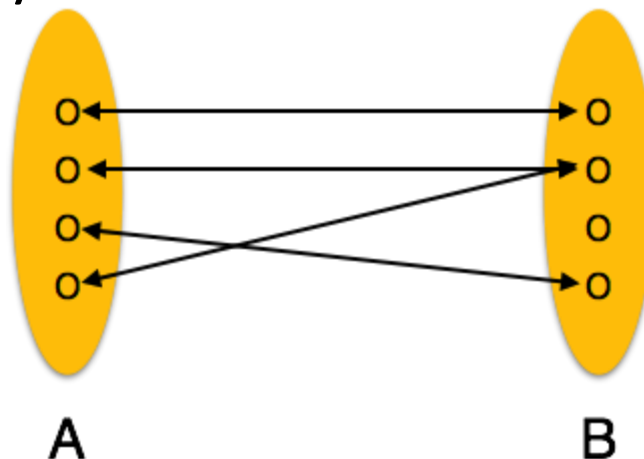
Each supplier can participate in multiple contracts. While a single contract might be assigned to a single supplier.

Кожний постачальник може бути позначений в багатьох договорах. Однак один договір може бути пов'язаний лише з одним постачальником.

Багато до одного / Many-to-one

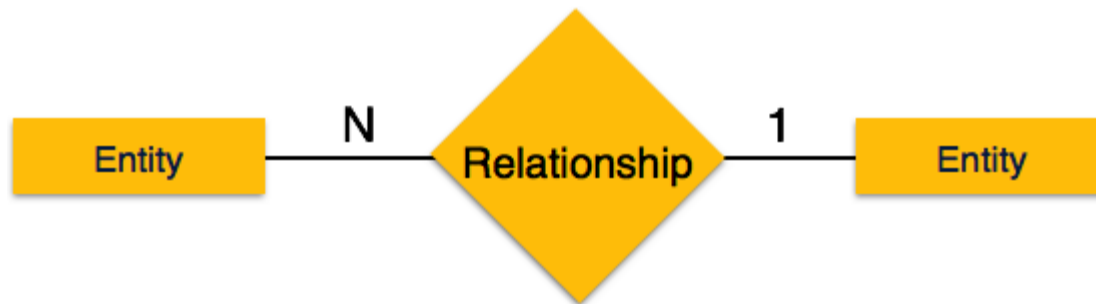
Більше однієї сутності з набору сутностей A можуть бути асоційовані не більше ніж з однією сутністю набору сутностей B, однак сутність з набору сутностей B може бути пов'язана з кількома сутностями з набору сутностей A.

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.



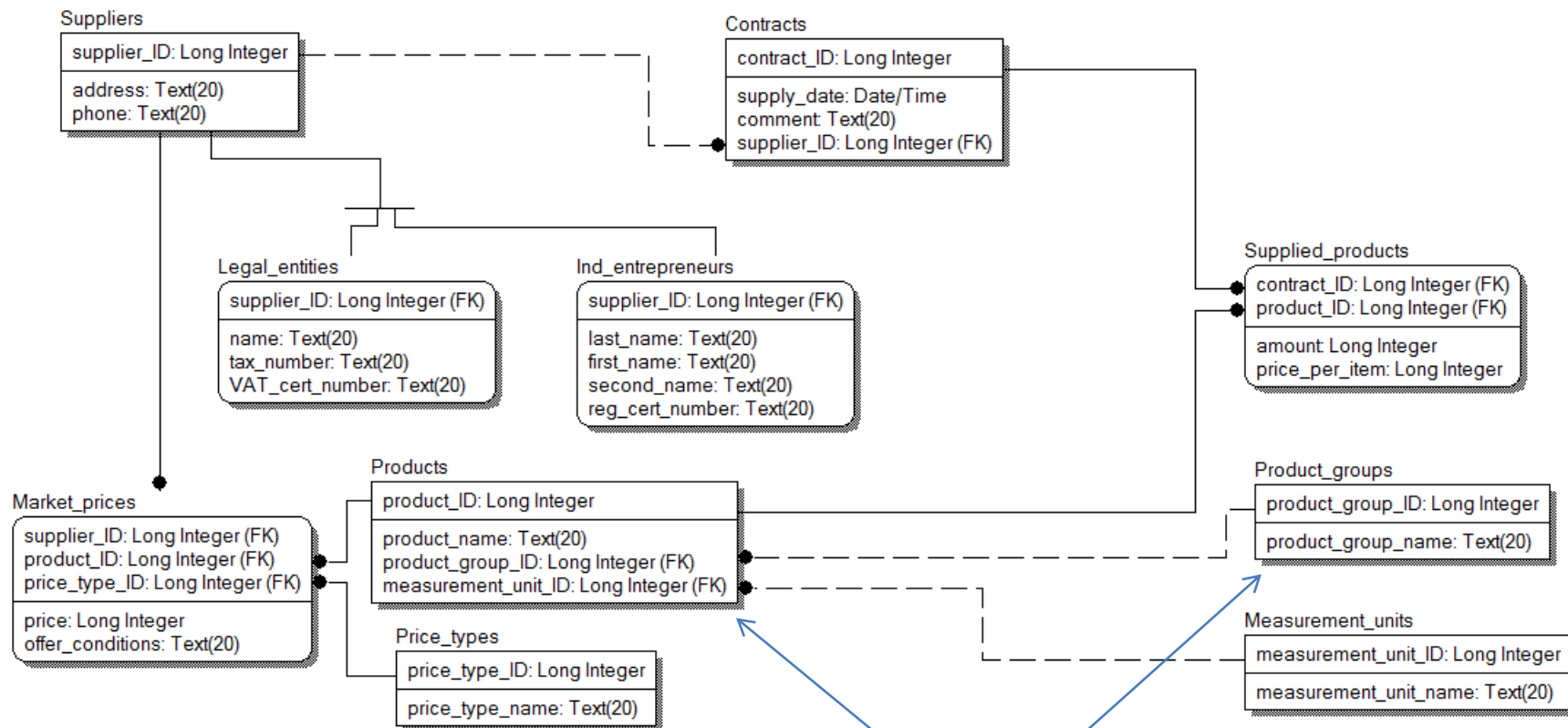
Багато-до-одного – якщо більше одного екземпляру сутності пов'язано з відношенням, воно позначається як "N: 1". Наступне зображення відображає, що більше ніж один екземпляр сутності ліворуч і лише один екземпляр сутності праворуч може бути пов'язаний з цим відношенням.

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 багатьох товарів може бути визначена одна продуктова група. Однак для одного товару може бути визначена лише одна продуктова група.

For multiple products might be defined a single product category. But only one category might be defined for a single product.

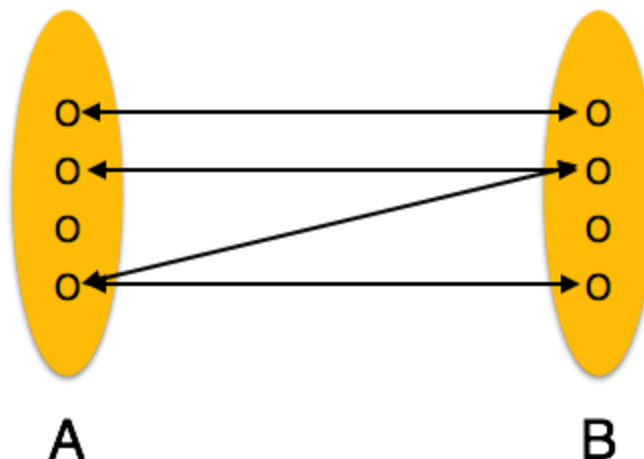


Many-to-one

Багато до багатьох / Many-to-many

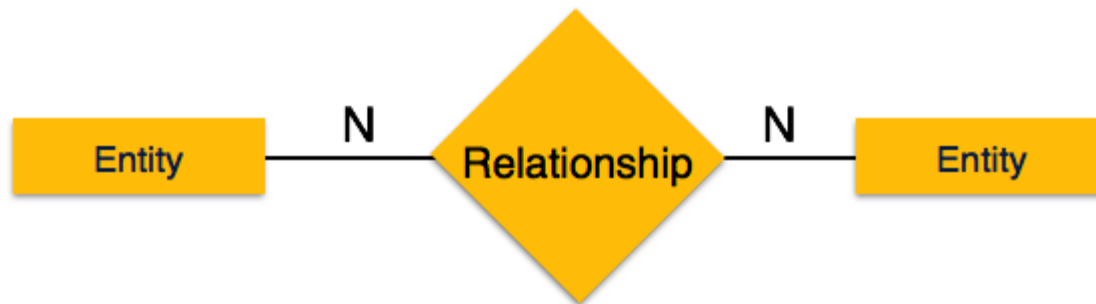
Одна сутність з А може бути пов'язана з кількома сутностями з В і навпаки.

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



Багато-до-багатьох. Наступне зображення відображає, що більше ніж один екземпляр сутності зліва і більше одного екземпляра сутності праворуч може бути пов'язаний з цим відношенням.

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.



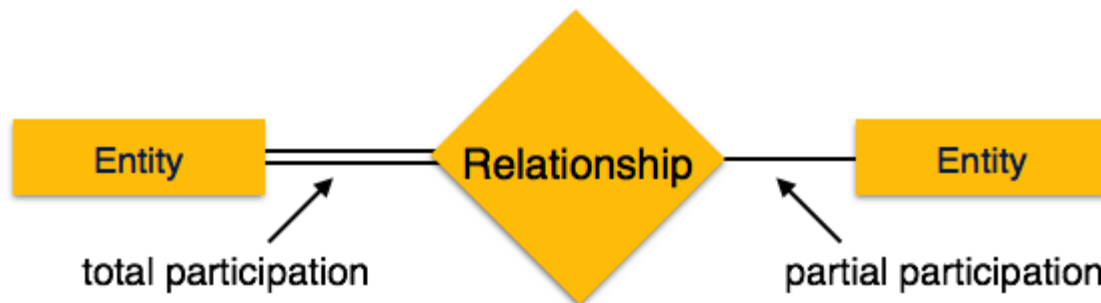
Обмеження участі / Participation constraints

Загальна участь – кожна сутність бере участь у відношенні. Загальна участь представлена подвійними лініями.

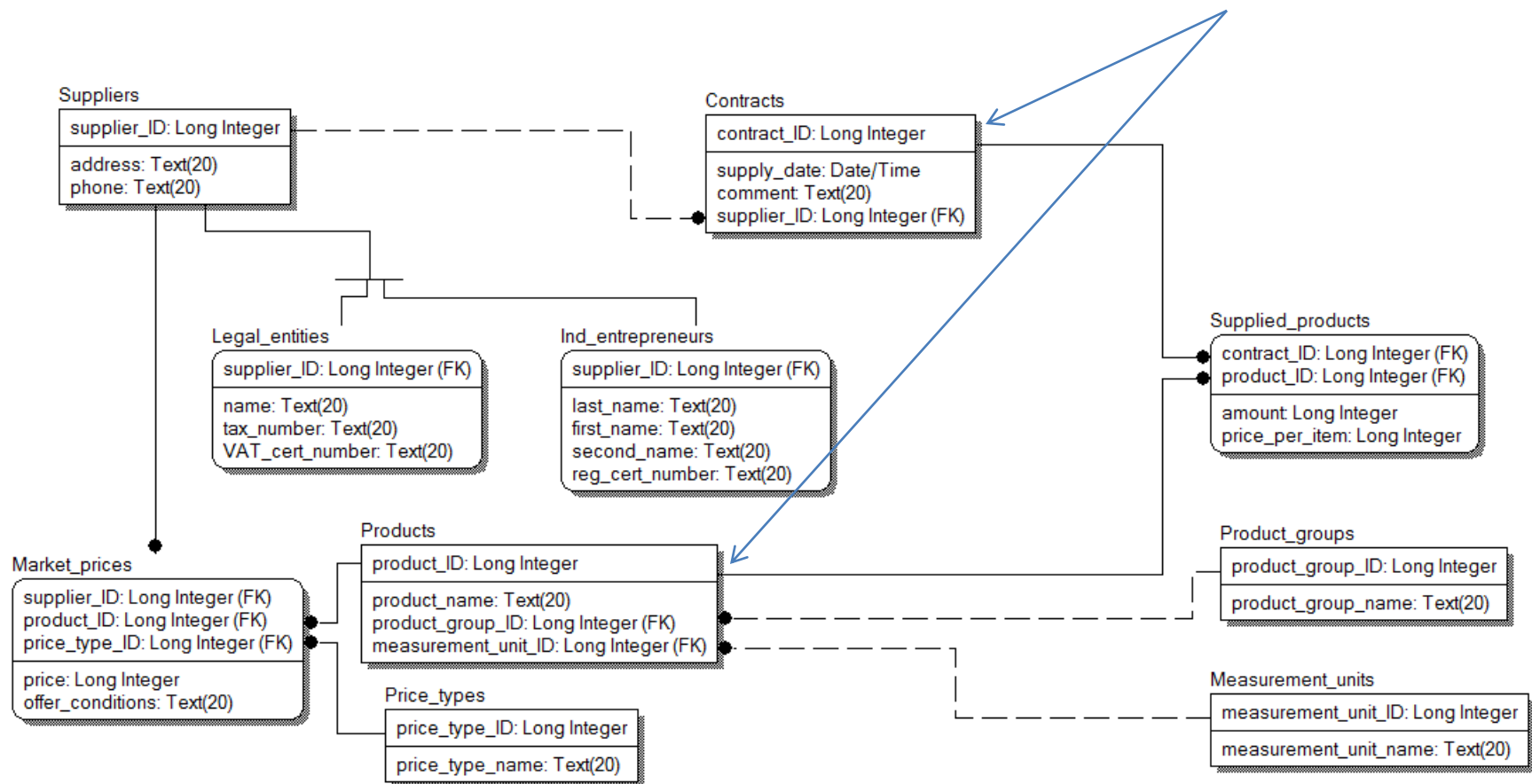
Total Participation – Each entity is involved in the relationship. Total participation is represented by double lines.

Часткова участь – у відношенні беруть участь не всі сутності. Часткова участь представлена лініями.

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



Many-to-many



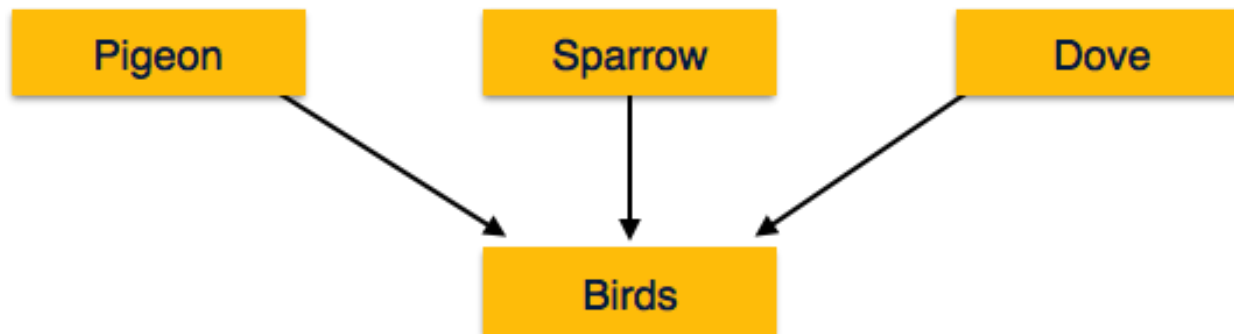
Each contract can be composed of multiple products. Each product can be assigned to multiple contracts.

Кожний договір може складатися з багатьох товарів. Кожний товар може бути зазначений у багатьох договорах.

Узагальнення / Generalization

В узагальненні ряд сутностей об'єднуються в одну узагальнену сутність на основі їх аналогічних характеристик.

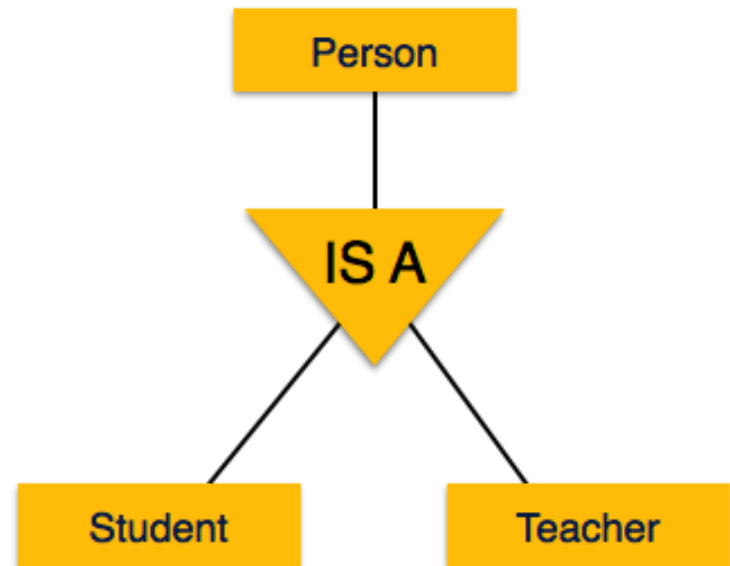
In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics.



Спеціалізація / Specialization

Спеціалізація – це протилежність узагальненню. У спеціалізації група сутностей поділяється на підгрупи на основі їх характеристик.

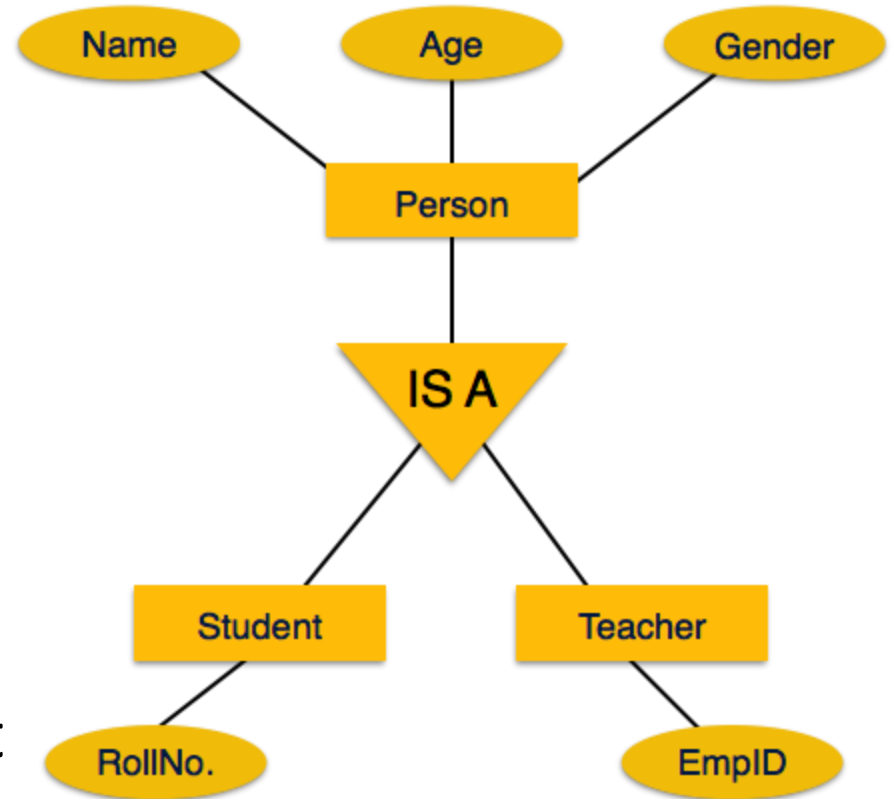
Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics.



Спадкування / Inheritance

Спадкування є важливою ознакою узагальнення та спеціалізації. Воно дозволяє сутностям нижчого рівня успадковувати атрибути сутностей вищого рівня.

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



Завдання для тренування 01 / Training task 01

Деяке **підприємство** виробляє певні товари. Для кожного **товару** вказується кількість, ціна та дата виготовлення. У відомостях про підприємство зазначаються адреса та телефон. **Адреса** вказується із зазначенням області та міста.

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.

Лекція 04: Нормалізація бази даних

Lecture 04: Database normalization

Нормалізація / Normalization

Якщо база даних ненормалізована, вона може містити **аномалії**, які є нічним кошмаром для кожного адміністратора баз даних. Управління базою даних, що містить аномалії, майже неможливо.

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.

Аномалії оновлення / Modification anomalies

Якщо дані розкидані і не пов'язані між собою належним чином, це може призвести до дивних ситуацій. Наприклад, коли ми намагаємося оновити один елемент даних, який має копії, розкидані по декількох місцях, декілька екземплярів оновлюються правильно, тоді як декілька інших залишаються зі старими значеннями. Такі випадки залишають базу даних у неузгодженому стані.

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

Якщо телефон студента А зміниться, його необхідно оновити у всіх відповідних записах таблиці.

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

Аномалії видалення / Deletion anomalies

- В результаті спроби видалити запис, деякі його частини не були видалені через незнання того, що дані також зберігаються десь в іншому місці.
- 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

Якщо видалити записи про біологію з бази даних, будуть також втрачені усі дані про студента B.

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


Аномалії вставки / Insert anomalies

Спроба вставити дані в запис, який взагалі не існує.

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 базі даних неможливо зберегти дані про новий предмет, тому що він не буде пов'язаний з жодним студентом.

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

Перша нормальна форма / First normal form

Це правило визначає, що всі атрибути у відношенні повинні мати атомарні домени. Значення в атомарному домені є неподільними одиницями. Кожен атрибут повинен містити лише одне значення з його попередньо визначеного домену.

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

Друга нормальна форма / Second normal form

- Для дотримання другої нормальної форми, кожен не первинний атрибут повинен повністю функціонально залежати від первинного ключа. Тобто, якщо $X \rightarrow A$ справедливо, то не повинно бути жодної підмножини Y множини X , для якої $Y \rightarrow A$ також 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.

Функціональна залежність / Functional Dependency

Функціональна залежність (FD) – це сукупність обмежень між двома атрибутами у відношенні. Функціональна залежність говорить, що якщо два кортежі мають однакові значення для атрибутів A_1, A_2, \dots, A_n , то ці два кортежі повинні мати однакові значення для атрибутів 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 .

Функціональна залежність представлена знаком стрілки (\rightarrow), тобто $X \rightarrow Y$, де X функціонально визначає Y . Атрибути лівої частини визначають значення атрибутів у правій частині.

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.

- **Рефлексивність.** Якщо A – це набір атрибутів і $B \in$ підмножиною A , то A містить B .
- **Reflexive rule.** If A is a set of attributes and B is subset of A , then A holds B .

- **Доповнення.** Якщо $A \rightarrow B$ виконується, та $Y \in$ множиною атрибутів, то $AY \rightarrow BY$ також виконується. Це додавання атрибутів до залежностей, не змінює основних залежностей.
- **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.
- **Транзитивність.** Те ж, що і транзитивне правило в алгебрі, якщо виконується $A \rightarrow B$, і $B \rightarrow C$, то $A \rightarrow C$ також має місце. $A \rightarrow B$ функціонально визначає 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
--------	---------	----------	-----------



Часткова залежність 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
---------	-----------

Третя нормальна форма / Third normal form

Відношення, яке має бути у третій нормальній формі, має бути у другій нормальній формі, а також має виконуватися:

- немає нетривіальних атрибутів, які транзитивно залежать від первинного ключа;
- для будь-якої нетривіальної функціональної залежності $X \rightarrow A$, X має бути супер-ключем або A має бути первинним ключем.

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.

Тривіальна функціональна залежність / Trivial Functional Dependency

- Якщо виконується функціональна залежність $X \rightarrow Y$, де Y – підмножина X , то вона називається **тривіальною**.
- If a functional dependency $X \rightarrow Y$ holds, where Y is a subset of X , then it is called a **trivial**.
- Якщо виконується $X \rightarrow Y$, де Y не є підмножиною X , то функціональну залежність називають **нетривіальною**.
- If an functional dependency $X \rightarrow Y$ holds, where Y is not a subset of X , then it is called a **non-trivial**.
- Якщо виконується $X \rightarrow Y$, де перетин X та $Y = \emptyset$, то функціональна залежність вважається **абсолютно нетривіальною**.
- If an functional dependency $X \rightarrow Y$ holds, where x intersect $Y = \emptyset$, it is said to be a **completely non-trivial**.

Student_Detail

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

Транзитивна залежність
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

Нормальна форма Бойса-Кодда / Boyce-Codd Normal Form

Нормальна форма Бойса-Кодда (BCNF) є продовженням третьої нормальної форми на більш строгих умовах. BCNF стверджує, що для будь-якої нетривіальної функціональної залежності, $X \rightarrow A$, X має бути супер-ключем.

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.

Завдання для тренування 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 відношення до 3 НФ / Bring the relation into 3 NF

Декомпозиція схеми відношення / Relation scheme decomposition

Декомпозицією схеми відношення $R = \{A_1, A_2, \dots, A_n\}$ називається заміна її сукупністю підмножин R , таких що їх об'єднання дає R . При цьому допускається, що підмножини можуть перетинатися.

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

$\text{Product_ID} \rightarrow \text{Product}$

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

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

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

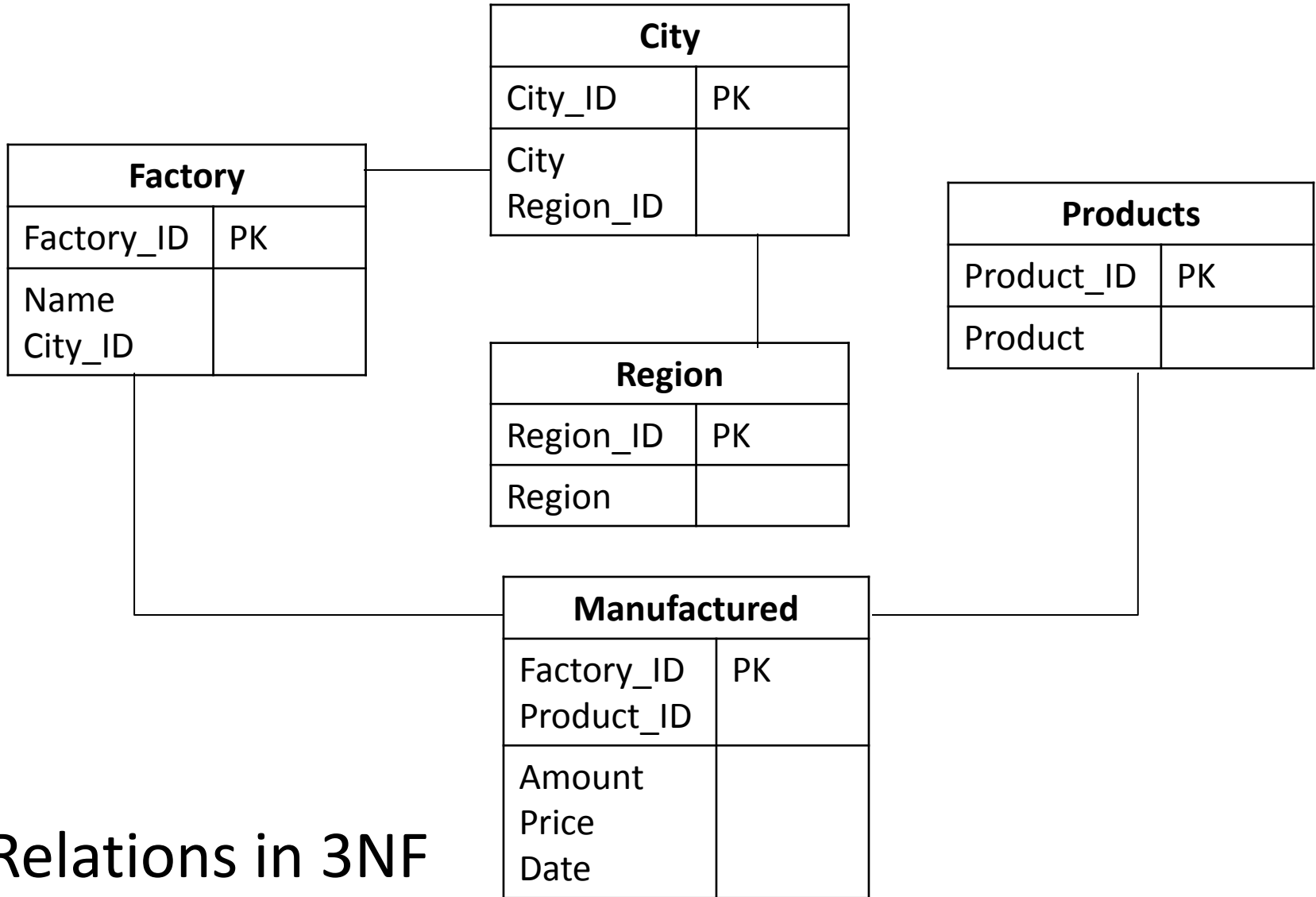
SOLUTION: Split R_{Factory} into several relations

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

Четверта нормальна форма / Fourth normal form

Відношення знаходиться в четвертій нормальній формі, якщо воно знаходиться у нормальній формі Бойса-Кодда та не містить нетривіальних **багатозначних залежностей**.

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**.

П'ята нормальна форма / Fifth normal form

Відношення знаходиться у п'ятій нормальній формі тоді та тільки тоді, коли кожна нетривіальна **залежність з'єднання** у ній визначається потенційним ключем (ключами) цього відношення.

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-ключова нормальна форма / Domain-key normal form

Відношення знаходиться у доменно-ключовій нормальній формі тоді та тільки тоді, коли кожне накладене на неї обмеження є логічним наслідком **обмежень доменів і обмежень ключів**, накладених на дане відношення.

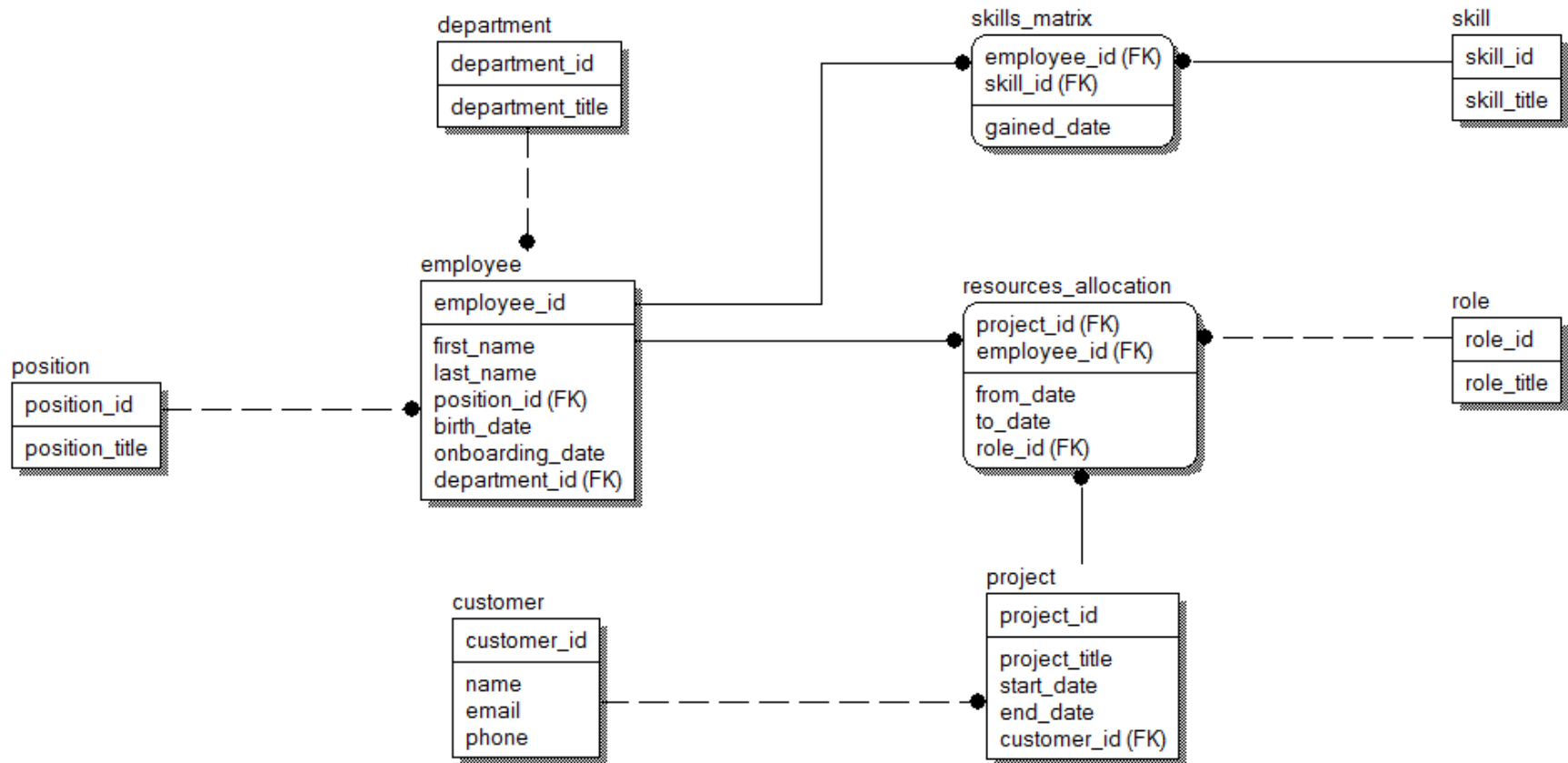
A relation is in the domain-key normal form if and only if it contains no constraints other than **domain constraints** and **key constraints**.

Шоста нормальна форма / Sixth normal form

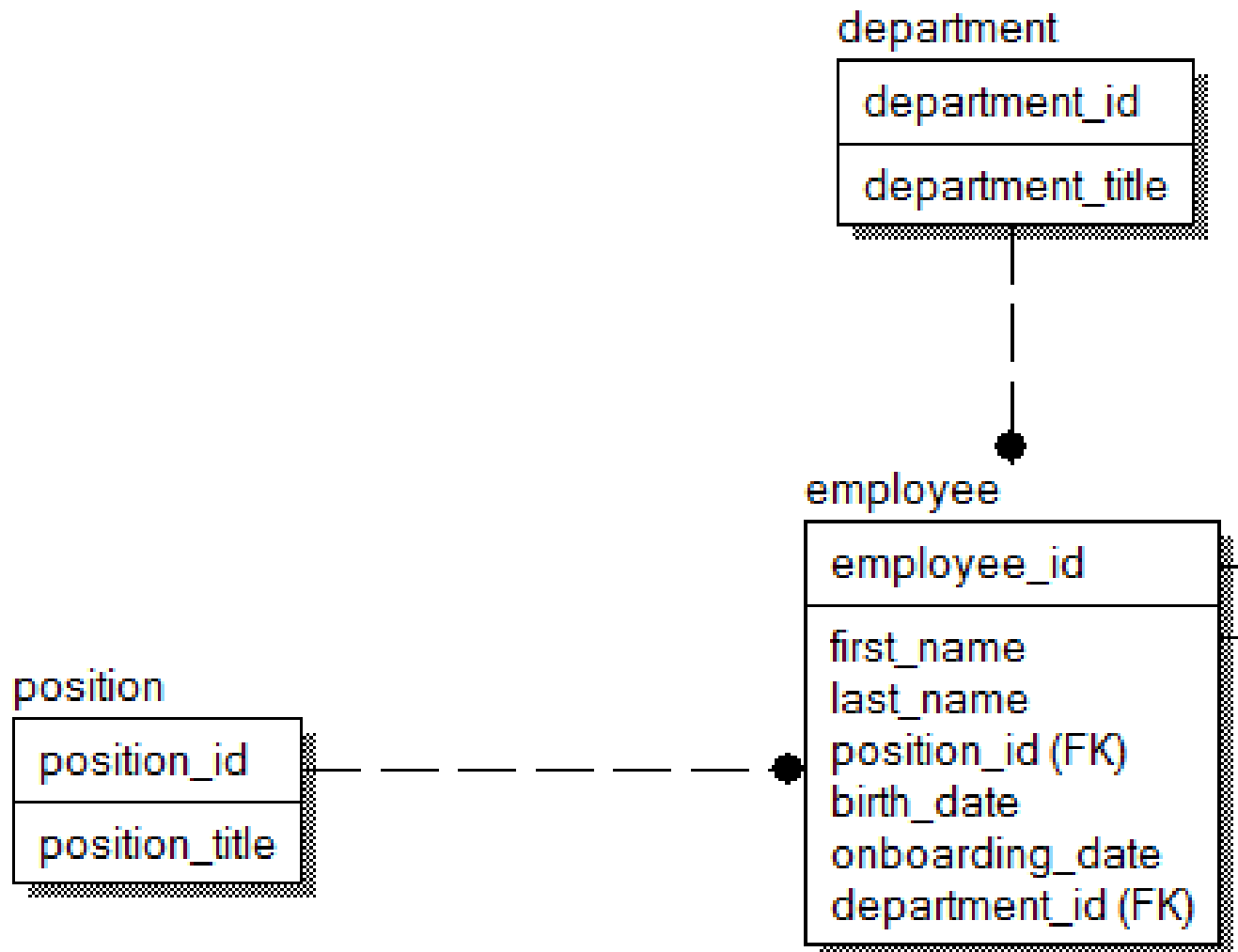
Відношення знаходиться у шостій нормальній формі тоді і тільки тоді, коли вона задовольняє **усім нетривіальним залежностям з'єднання** (тобто, не може бути піддана подальшій декомпозиції без втрат).

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).

Предметна область для контролю: IT компанія / Subject area for tests: IT company



Співробітники / Employees

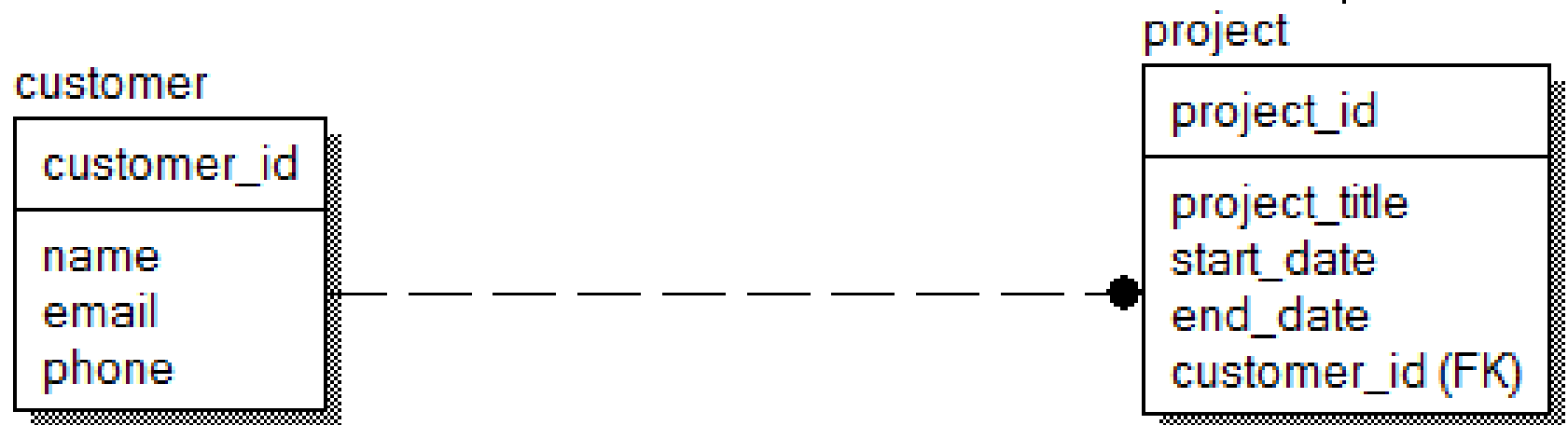


employee							
	employee_i ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾	department ▾
+		1 Jim	Halpert	2	5/12/1990	3/2/2014	1
+	2	Pamela	Beesly	1	11/2/1992	9/23/2015	2
+	3	Dwight	Schrute	3	2/11/1991	12/3/2013	1
+	4	Kelly	Kapoor	4	6/4/1993	3/2/2014	3
+	5	Michael	Scott	5	12/9/1989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

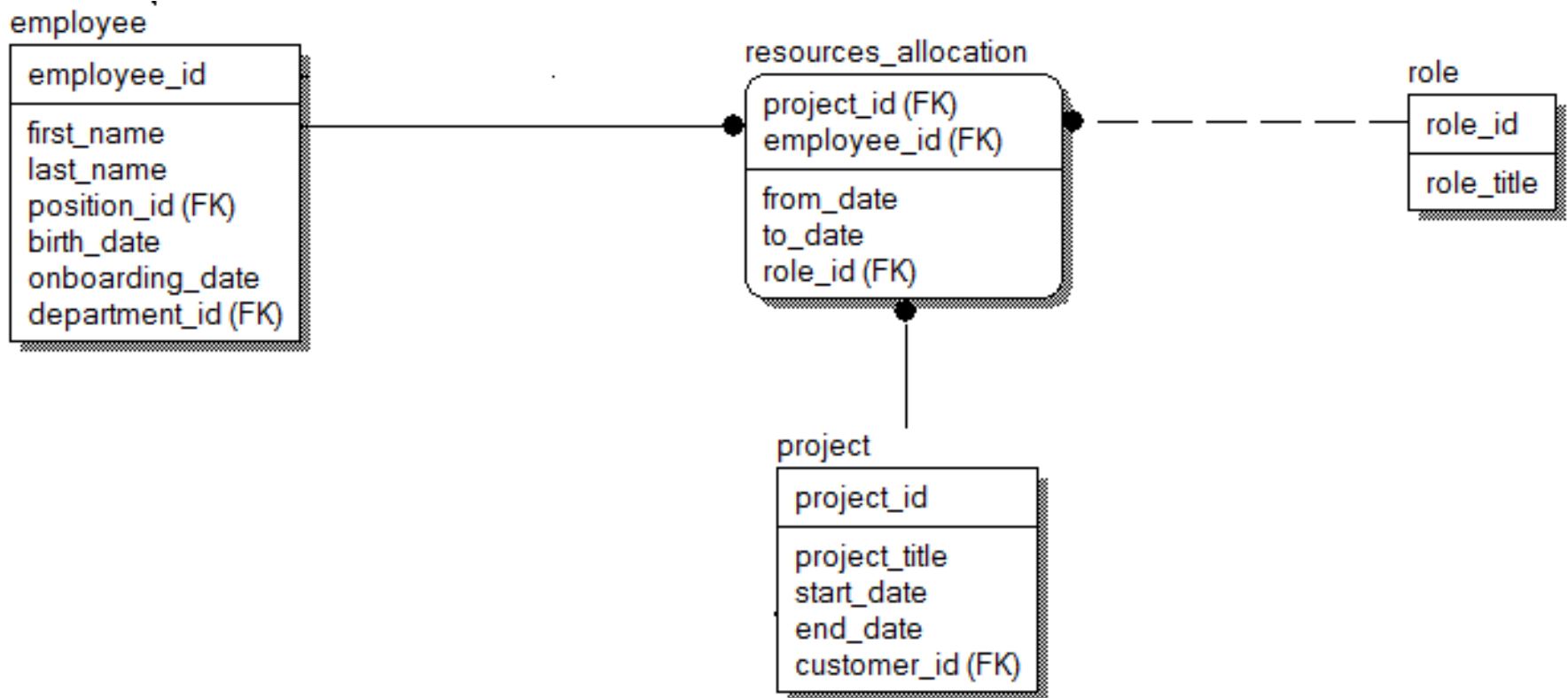
Проекты / Projects



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

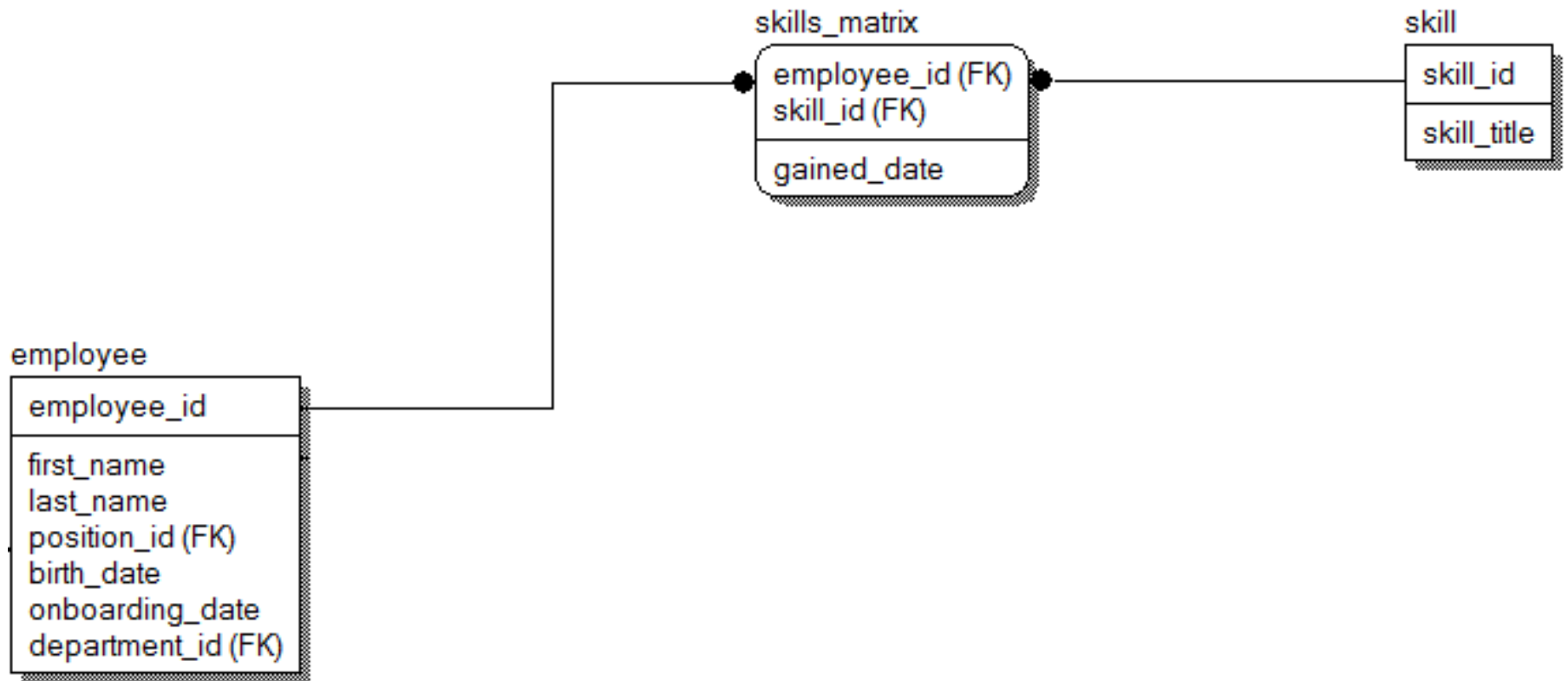
Відношення співробітники-проекти / Employees-projects relation



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

Відношення співробітники-вміння / 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

Зберігати дані про вміння,
необхідні для займання
певної посади, вказуючи
рівень кожного вміння

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

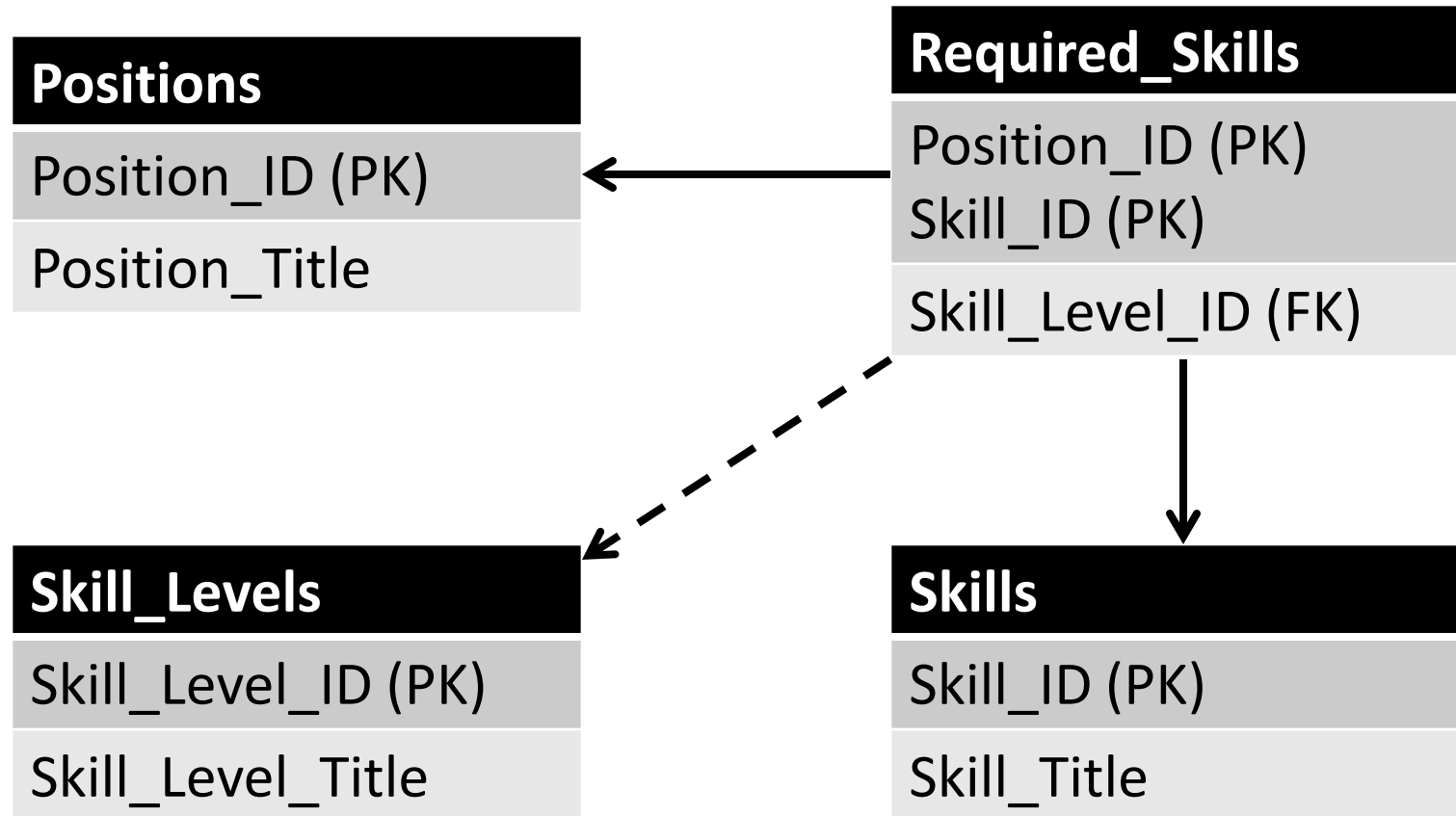
Position_ID	Skill_ID	Level
DEV	JSE	Intermediate
DEV	JSP	Elementary
...
MT	DLC	Advanced



Modification anomalies

What if we need to:

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



Лекція 05: Реляційна модель даних

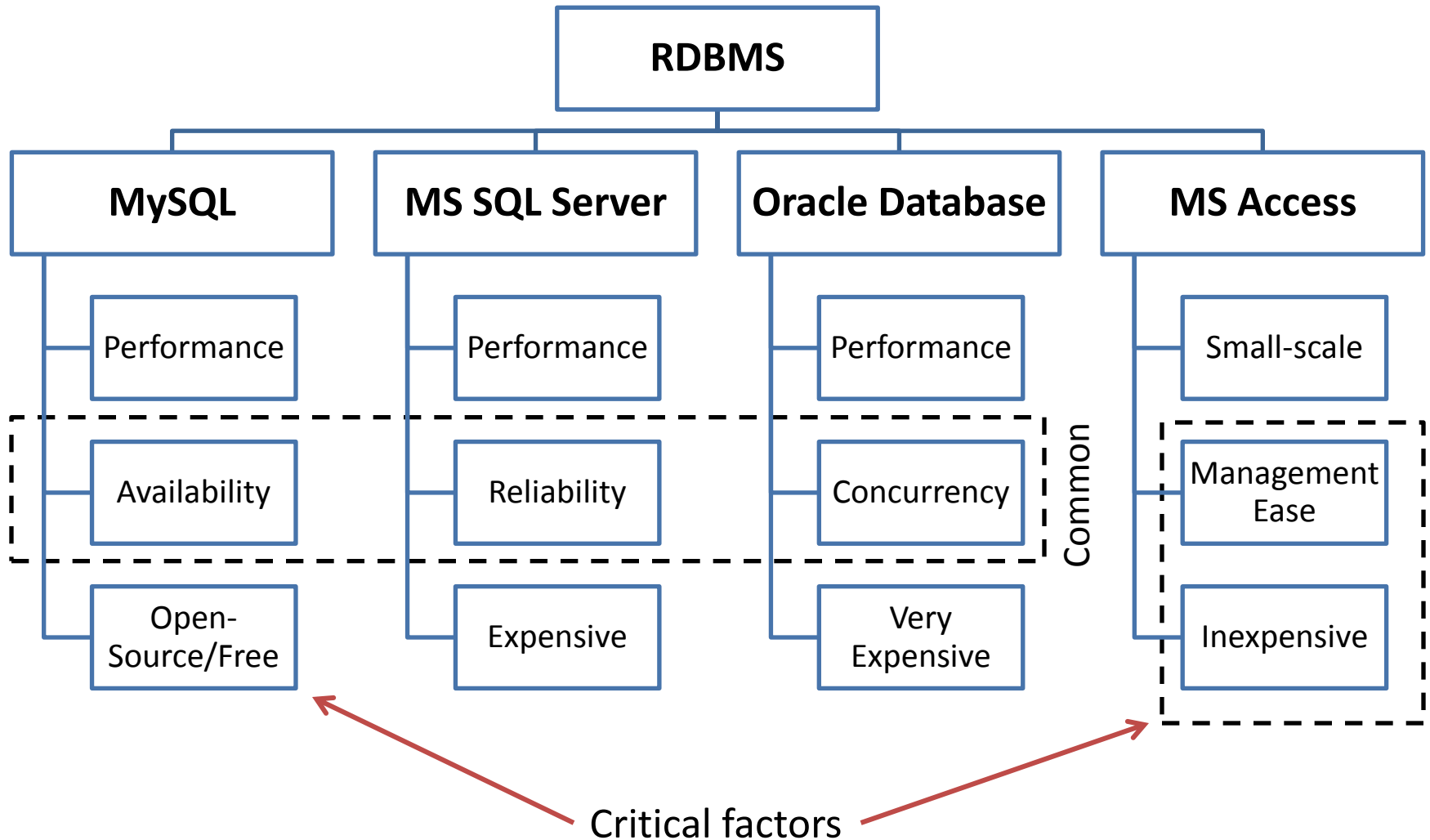
Lecture 05: Relational data model

Реляційна модель даних / Relational data model

Реляційна модель даних є основною моделлю даних, яка широко використовується в усьому світі для зберігання та обробки даних. Ця модель проста і має всі властивості та можливості, необхідні для обробки даних з ефективністю зберігання.

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.

Реляційні СУБД / Relational DBMS



MySQL



MySQL – це база даних з відкритим вихідним кодом, розроблена шведською компанією MySQL AB.

MySQL підтримує багато різних платформ, включаючи Microsoft Windows, основні дистрибутиви Linux, UNIX і Mac OS.

MySQL має безкоштовні та платні версії залежно від його використання (не комерційні або комерційні) та функцій. MySQL поставляється з дуже швидким багатопотоковим, багатокористувацьким і надійним сервером баз даних 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 – це система управління реляційними базами даних, розроблена компанією Microsoft Inc. Окрім високої продуктивності та доступності, надає можливості щодо бізнес-аналітики (SQL Server Analytics Services) та хмарних обчислень (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



Це дуже велика багатокористувачка система управління базами даних. Oracle Database – це реляційна система управління базами даних, розроблена корпорацією Oracle.

Це відмінний вибір сервера баз даних для клієнтських або серверних обчислень. Oracle Database підтримує всі основні операційні системи для клієнтів і серверів.

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



Це один з найпопулярніших продуктів Microsoft. Microsoft Access – це СУБД початкового рівня. База даних MS Access не тільки недороге, але потужне рішення для малих проектів.

MS Access використовує певний діалект мови SQL (іноді називається Jet SQL).

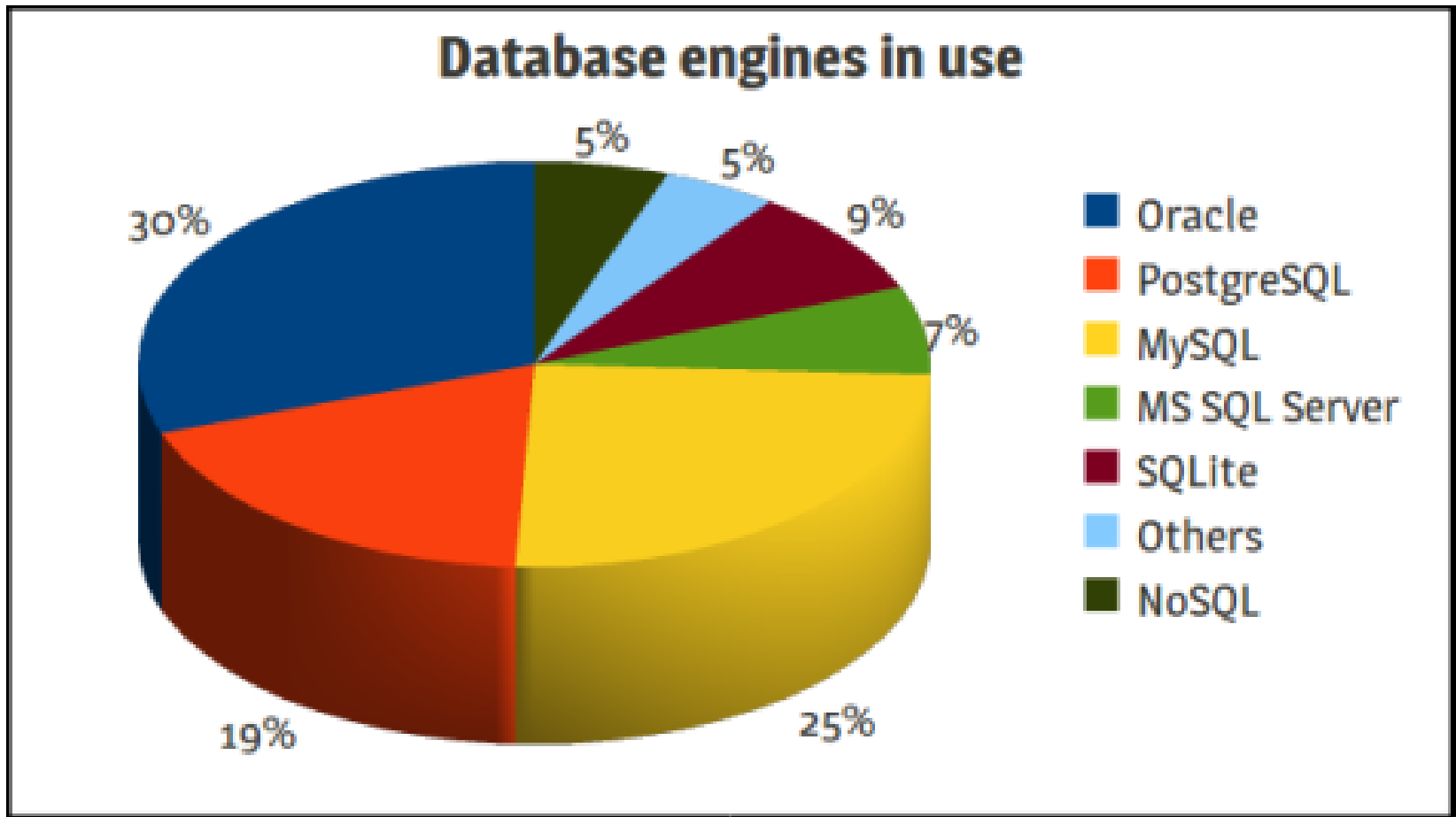
MS Access постачається з професійним виданням пакета MS Office. MS Access має простий у використанні інтуїтивно зрозумілий графічний інтерфейс.

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).

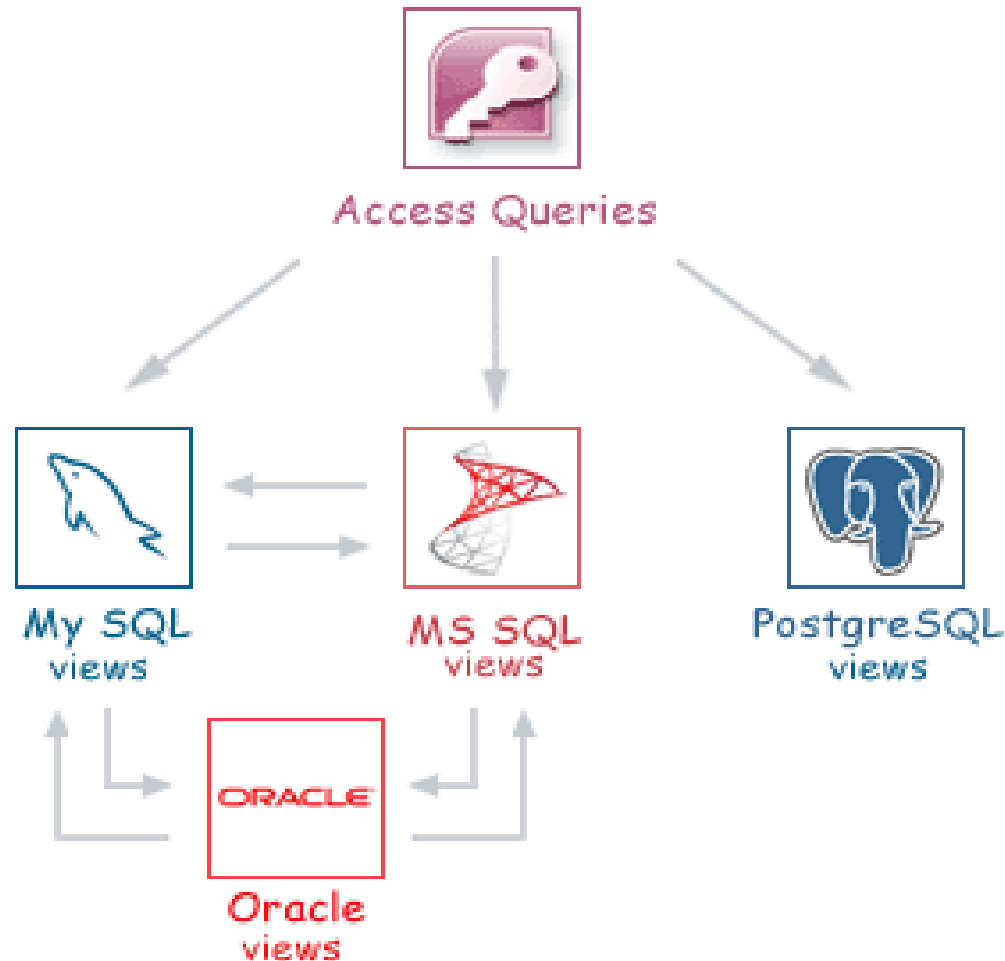
MS Access comes with the professional edition of MS Office package. MS Access has easy-to-use intuitive graphical interface.

RDBMS Market Share 2018



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

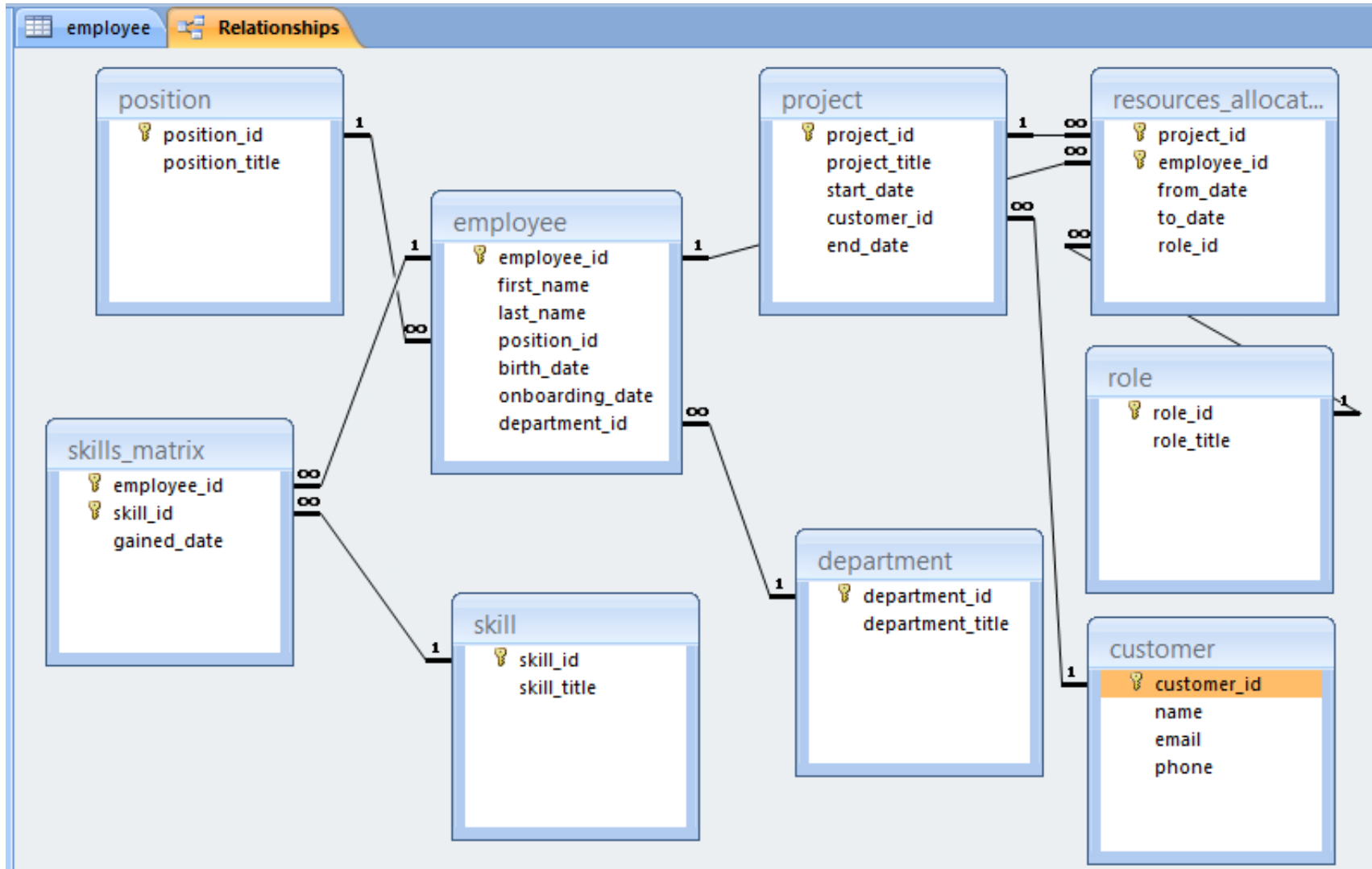
Подібність синтаксису SQL / Similarity of SQL syntax



Основні поняття / Concepts

- **Таблиці** – у реляційній моделі даних відношення зберігаються у форматі таблиць. Цей формат зберігає відношення між суб'єктами. У таблиці є рядки та стовпці, де рядки представляють записи, а стовпці – атрибути.
- **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.

Таблиці / Tables



- **Кортеж** – окремий рядок таблиці, що містить єдиний запис для цього відношення, називається кортежем.
- **Tuple** – A single row of a table, which contains a single record for that relation is called a tuple.
- **Екземпляр відношення** – кінцевий набір кортежів у реляційній базі даних представляє екземпляр відношення. Екземпляри відношення не мають дублікатів кортежів.
- **Relation instance** – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Кортежі та екземпляр відношення / 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 відношення** – схема відношення описує ім'я відношення (назву таблиці), атрибути та їхні імена.
- **Relation schema** – A relation schema describes the relation name (table name), attributes, and their names.
- **Ключ відношення** – кожен рядок має один або декілька атрибутів, відомих як ключ відношення, який може однозначно ідентифікувати рядок у відношенні (таблиці).
- **Relation key** – Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Схема та ключ відношення / 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 визначення атрибутів** – кожен атрибут має певну заздалегідь визначену область визначення, відому як домен атрибута.
- **Attribute domain** – Every attribute has some pre-defined value scope, known as attribute domain.

Правила Кодда / Codd's rules

Д-р **Едгар Кодд**, після тривалих досліджень щодо реляційної моделі систем баз даних, висунув дванадцять власних правил, які, на його думку, повинна виконувати база даних для того, щоб вважатися справжньою реляційною базою даних.

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.

Ці правила можуть бути застосовані до будь-якої системи бази даних, яка управляє збереженими даними, використовуючи лише свої реляційні можливості. Це основне правило, яке є основою для всіх інших 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: Інформаційне правило / Rule 1: Information

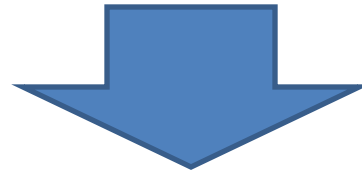
Дані, що зберігаються в базі даних, які можуть бути користувацькими даними або метаданими, повинні бути значенням деякої комірки таблиці. Все в базі даних повинно зберігатися у форматі таблиці.

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

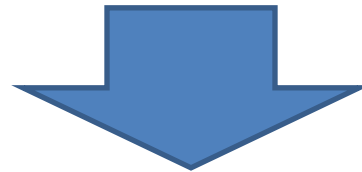
```
> SHOW DATABASES
```



Database
delivery
information_schema
mysql
performance_schema
phpmyadmin
test

```
> SHOW TABLES
```

```
>
```



Tables_in_delivery

contract

legal_supplier

private_supplier

product

supplier

Правило 2: Гарантований доступ /

Rule 2: Guaranteed Access

Для кожного елемента даних (значення) гарантований логічний доступ за комбінацією імені таблиці, первинного ключа (значення рядка) та імені атрибута (значення стовпця). Інших засобів, таких як покажчики, не можна використовувати для доступу до даних.

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: Систематична обробка NULL / Rule 3: Systematic Treatment of NULL Values

Ставлення до значень NULL в базі даних повинне бути систематичним та однорідним. Це дуже важливе правило, оскільки значення NULL може бути інтерпретовано як: дані відсутні, дані невідомі або дані не можуть бути застосовні.

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: Активний онлайн-каталог

/ Rule 4: Active Online Catalog

Опис структури всієї бази даних має зберігатися в онлайн-каталозі, відомому як словник даних, до якого можуть отримати доступ авторизовані користувачі. Користувачі можуть використовувати ту ж мову запитів для доступу до каталогу, який вони використовують для доступу до самої бази даних.

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: Повнота підмножини мови /

Rule 5: Comprehensive Data Sub-Language

Доступ до БД можна отримати лише за допомогою мови, що має лінійний синтаксис, що підтримує визначення даних, маніпулювання даними та операції з управління транзакціями. Цю мову можна використовувати безпосередньо або за допомогою певної програми. Якщо база даних дозволяє отримати доступ до даних без допомоги цієї мови, то це вважається порушенням.

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: Оновлення представлення / Rule 6: View Updating

Всі представлення бази даних, які теоретично можуть бути оновлені, також повинні бути придатні до оновлення системою.

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

Правило 7: Вставка, оновлення та видалення високого рівня / Rule 7: High-Level Insert, Update, and Delete

База даних повинна підтримувати вставку, оновлення та видалення високого рівня. Це не повинно обмежуватися одним рядком, тобто вона також повинна підтримувати операції об'єднання, перетину та доповнення для отримання множини записів даних.

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
```

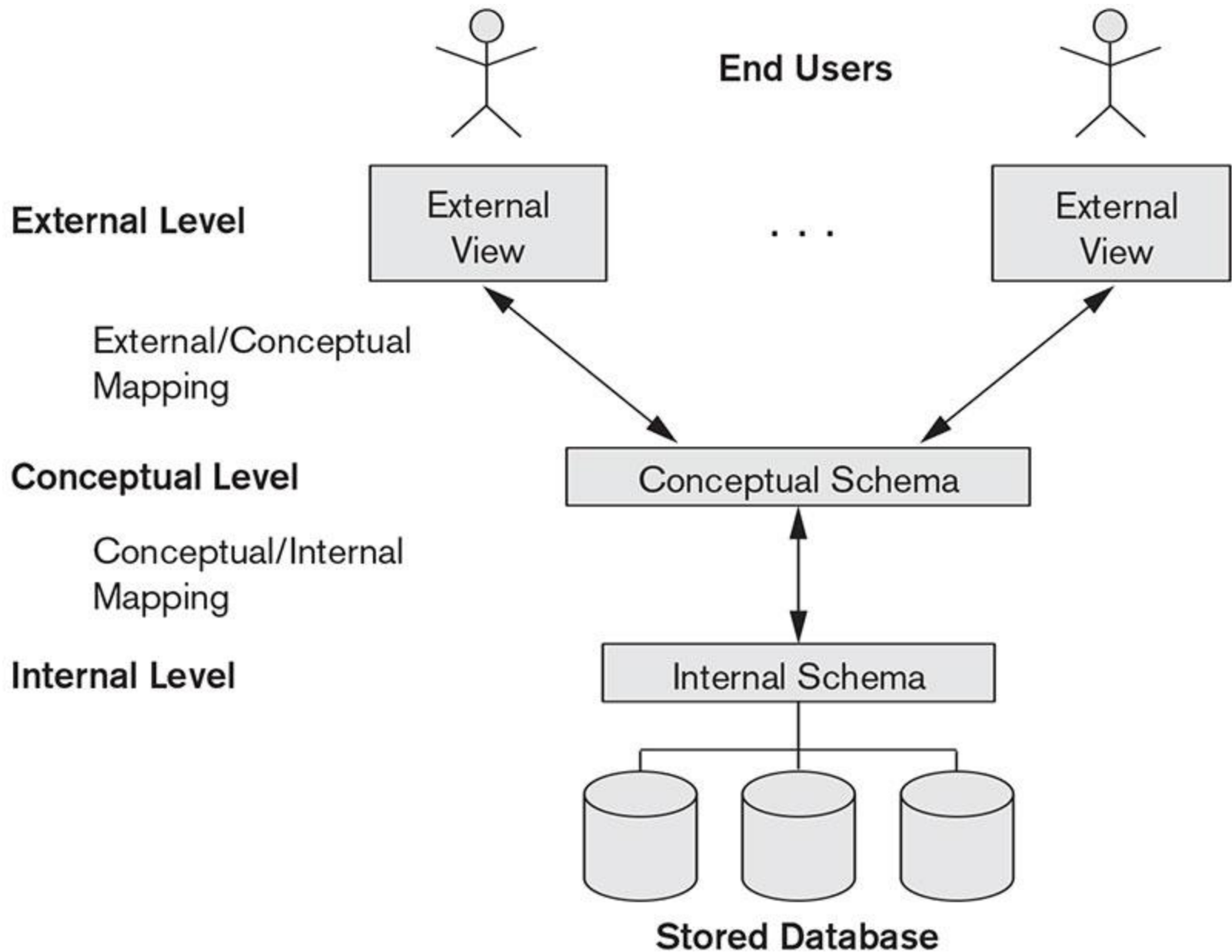


id	name	address
2	Ivanov I. I. PE	Kharkiv, Pushkinska Str., 77 (tel. 33-33-44, 12-...
5	"Transservis" LLC	Odesa, Deribasivska Str., 75
6	Sidorov S. S. PE	Poltava, Svobody Str., 15, apt. 43

Правило 8: Фізична незалежність даних / Rule 8: Physical Data Independence

Дані, що зберігаються в базі даних, повинні бути незалежними від програм, які мають доступ до бази даних. Будь-які зміни в фізичній структурі бази даних не повинні мати ніякого впливу на те, як зовнішні програми отримують доступ до даних.

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.

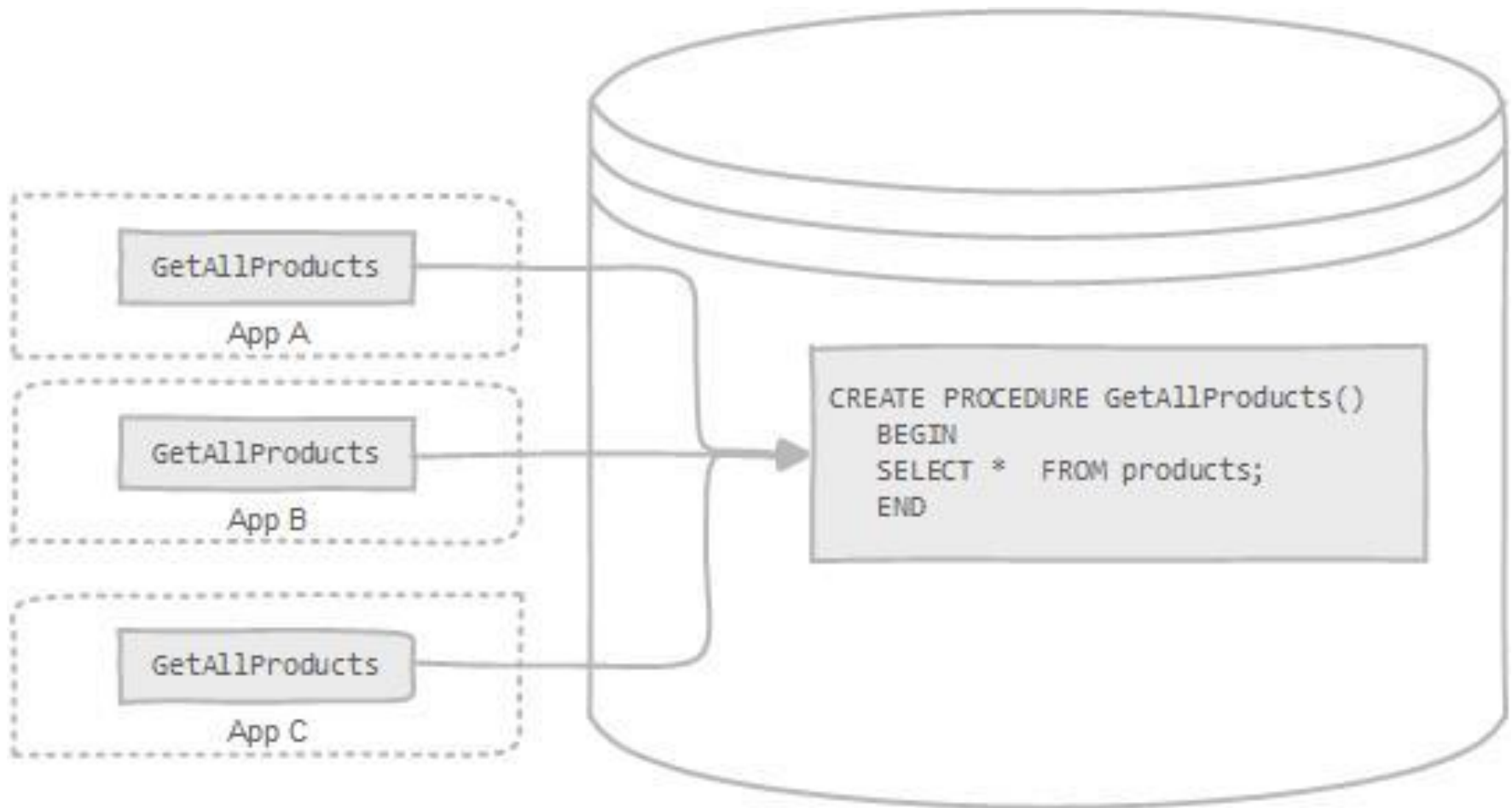


Правило 9: Логічна незалежність даних /

Rule 9: Logical Data Independence

Логічні дані в базі даних повинні бути незалежними від перегляду користувача (програми). Будь-яка зміна логічних даних не повинна впливати на програми, що їх використовують. Наприклад, якщо дві таблиці об'єднуються або одна поділяється на дві різні таблиці, користувацькі застосунки не повинні зазнавати впливу або змін. Це одне з найважчих правил щодо виконання.

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: Незалежність контролю цілісності / Rule 10: Integrity Independence

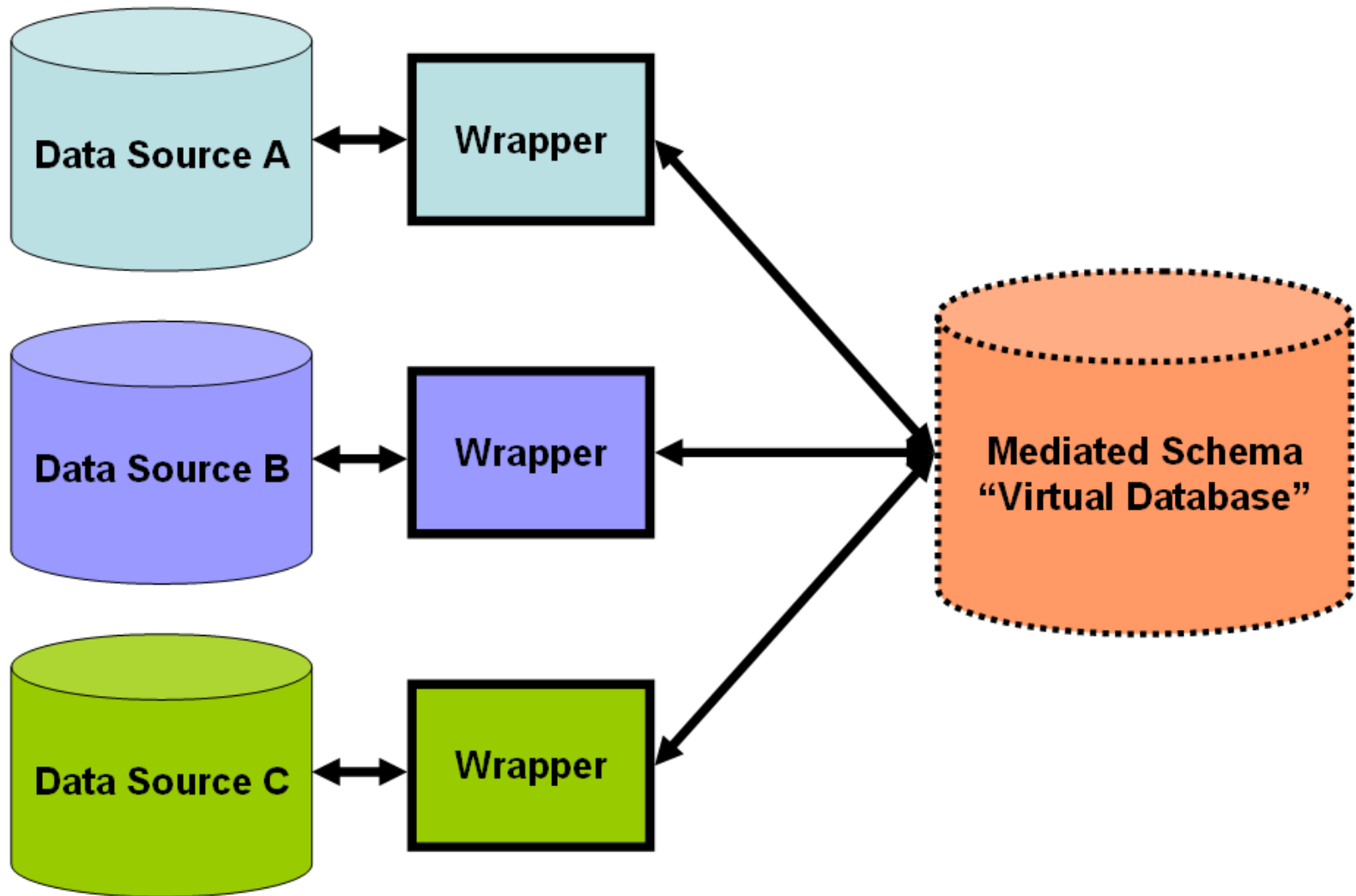
База даних повинна бути незалежною від програми, яка її використовує. Усі її обмеження цілісності можуть бути незалежно модифіковані без необхідності будь-яких змін у додатку. Це правило створює базу даних, незалежну від зовнішньої програми та її інтерфейсу.

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: Незалежність від розміщення / Rule 11: Distribution Independence

Кінцевий користувач не повинен мати змоги бачити, що дані розподіляються по різних місцях. Користувачам завжди потрібно створити враження, що дані розміщуються лише в одному місці. Це правило розглядається як основа розподілених систем баз даних.

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: Узгодженість мовних рівнів / Rule 12: Non-Subversion

Якщо в системі є інтерфейс, що забезпечує доступ до записів низького рівня, то інтерфейс не повинен мати змоги зламати систему та обходити обмеження безпеки та цілісності.

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.

Обмеження / Constraints

Для кожного відношення існують деякі умови, які повинні для нього виконуватися. Ці умови називаються **реляційними обмеженнями цілісності**. Існує три основних обмеження цілісності:

- ключів;
- доменів;
- посилань.

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.

Обмеження ключів / Key constraints

У відношенні повинна бути принаймні одна мінімальна підмножина атрибутів, яка може однозначно ідентифікувати кортеж. Ця мінімальна підмножина атрибутів називається **ключем** для цього відношення. Якщо існує декілька таких мінімальних підмножин, вони називаються **потенційними 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**.

Обмеження ключів визначають, що:

- у відношенні з ключовим атрибутом, два кортежі не можуть мати ідентичні значення для ключових атрибутів.
- ключовий атрибут не може мати значення NULL.

Обмеження ключів також називаються обмеженнями сутності.

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.

Обмеження доменів / Domain constraints

В реальності атрибути мають певні значення. Наприклад, вік може бути лише позитивним цілим числом. Ті самі обмеження спробували застосувати до атрибутів відношення. Кожен атрибут повинен мати певний діапазон значень. Наприклад, вік не може бути менше нуля, а номери телефонів не можуть містити цифри за межами 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.

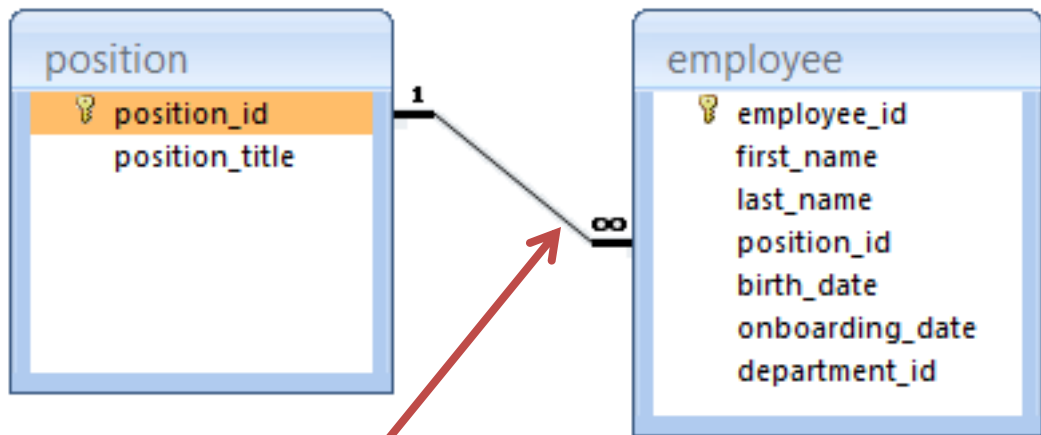
Обмеження цілісності посилань / Referential integrity constraints

Обмеження цілісності посилань оперують з концепцією **зовнішніх ключів**. Зовнішній ключ є ключовим атрибутом відношення, на який можна посилатися в іншому відношенні.

Обмеження цілісності посилань вказує, що якщо відношення посилається на ключовий атрибут іншого або того ж самого відношення, то цей ключовий атрибут повинен існувати.

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

Реляційна алгебра / Relational algebra

Реляційна алгебра є процедурною мовою запитів, яка приймає екземпляри відношень та повертає екземпляри відношень. Вона використовує оператори для виконання запитів. Оператор може бути як **унарним**, так і **бінарним**. Вони приймають відношення та повертають також відношення. Реляційна алгебра виконується рекурсивно на відношенні, а проміжні результати також розглядаються як відношення.

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.

Операція вибору / Select operation

Він вибирає кортежі, які задовольняють даному предикату, з відношення.

Позначення – $\sigma_p(r)$.

Де σ означає предикат вибору, а r – відношення. p – формула логіки препозиції, яка може використовувати з'єднувачі, такі як AND, OR, та NOT. Ці умови можуть використовувати реляційні оператори, такі як: $=$, \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?

Операція проєкції / Project operation

Вона проєктує стовпчик(и), що задовольняє заданому предикату.

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

Де A_1, A_2, \dots, A_n – назви атрибутів відношення r .

Дубльовані рядки автоматично виключаються, оскільки відношення є множиною.

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?

Операція об'єднання / Union operation

Вона виконує об'єднання двох відношень і визначається як:

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

Де **r** та **s** – це відношення бази даних або результуючі відношення (тимчасові відношення).

Для того, щоб операція об'єднання була дійсною, **r** та **s** повинні мати однакову кількість атрибутів; домени атрибутів повинні бути сумісними; дубльовані кортежі автоматично виключаються.

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?

Різниця множин / Set difference

Результатом операції різниці множин є кортежі, які присутні в одному відношенні, але не в другому відношенні.

Позначення – $r - s$.

Знаходить всі кортежі, які присутні в r , але 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 .

Декартів добуток / Cartesian product

Поєднує інформацію з двох різних відношень в одне.

Позначення – $r \times s$.

Де r та s – відношення, і їх результат буде визначений як:

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

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?

Операція перейменування / Rename operation

Результати реляційної алгебри - це також відношення, але без назви. Операція перейменування дозволяє нам перейменувати вихідне відношення.

Позначення – $\rho_x(E)$.

Де результат виразу **E** зберігається з назвою **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**.

Реляційне обчислення / Relational Calculus

На відміну від реляційної алгебри, реляційне обчислення є не процедурною мовою запитів, тобто вона визначає, що робити, але ніколи не пояснює, як це зробити.

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.

Реляційне обчислення кортежів / Tuple Relational Calculus (TRC)

Фільтрація кортежів за змінними
Filtering variable ranges over tuples

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

Повертає усі кортежі **T**, що задовольняють умові
Returns all tuples **T** that satisfies a condition

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

Яким буде результат?
What will be the output?

Реляційне обчислення доменів / Domain Relational Calculus (DRC)

Змінні фільтрації використовують домен атрибутів замість усіх значень кортежу (як це реалізовано в TRC)

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

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

Де **a1, a2, ..., an** – атрибути, **P** – формула, заснована на атрибутах.

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

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

Яким буде результат?

What will be the output?

Лекція 06: Мова запитів SQL

Lecture 06: SQL query language

SQL

SQL – це структурована мова запитів для зберігання, обробки та отримання даних, що зберігаються в реляційній базі даних. SQL є стандартною мовою ANSI (American National Standards Institute), але існує безліч різних версій мови 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.

Історія SQL

1970 – Доктор Кодд з IBM описав реляційну модель для баз даних.

1974 – З'явилася мова SQL.

1978 – IBM працював над розробкою ідей Кодда і випустив продукт під назвою System/R.

1986 – IBM розробив перший прототип реляційної бази даних, стандартизований ANSI. Перша реляційна база даних була випущена компанією Relational Software, яка згодом стала відома як 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.

Використання SQL / SQL usage

Всі реляційні СУБД, такі як MySQL, MS Access, Oracle, Sybase, Informix, PostgreSQL та SQL Server використовують SQL як стандартну мову бази даних.

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

Переваги SQL

- Дозволяє користувачам отримувати доступ до даних в реляційних СУБД.
- Дозволяє користувачам описувати дані.
- Дозволяє користувачам визначати дані в базі даних та маніпулювати ними.
- Дозволяє вбудовувати інші мови за допомогою модулів SQL, бібліотек та пре-компіляторів.
- Дозволяє користувачам створювати та видаляти бази даних та таблиці.
- Дозволяє користувачам створювати представлення, збережені процедури, функції в базі даних.
- Дозволяє користувачам встановлювати права на таблиці, процедури та представлення.

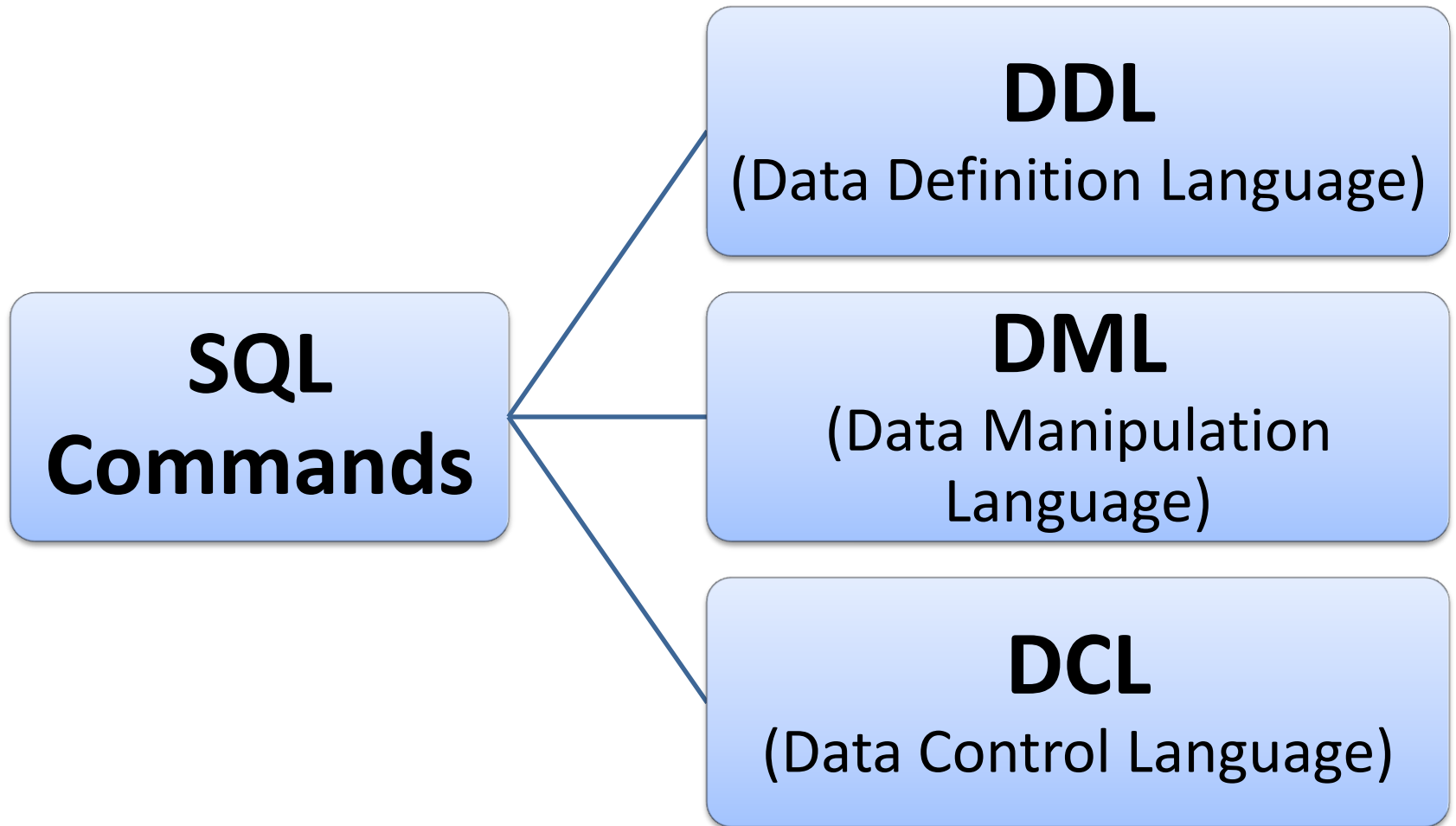
Advantages of SQL

- 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.

Команди SQL / SQL commands

Стандартні команди SQL для взаємодії з реляційними базами даних – CREATE, SELECT, INSERT, UPDATE, DELETE і DROP. Ці команди можна розділити на наступні групи залежно від їх характеру.

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 та опис
1	CREATE Створює нову таблицю, представлення таблиці або інший об'єкт у базі даних Creates a new table, a view of a table, or other object in the database
2	ALTER Модифікує існуючий об'єкт бази даних, такий як таблиця Modifies an existing database object, such as a table
3	DROP Видаляє цілу таблицю, представлення таблиці або інші об'єкти в базі даних Deletes an entire table, a view of a table or other objects in the database

Data Manipulation Language (DML)

Номер	Команда та опис
1	SELECT Отримує певні записи з однієї або декількох таблиць Retrieves certain records from one or more tables
2	INSERT Створює запис Creates a record
3	UPDATE Модифікує записи Modifies records
4	DELETE Видаляє записи Deletes records

Data Control Language (DCL)

Номер	Команда та опис
1	GRANT Дає привілей користувачу Gives a privilege to user
2	REVOKE Відміняє привілеї, надані користувачу Takes back privileges granted from user

Процес виконання команди SQL / SQL command execution process

Коли ви виконуєте команду SQL для будь-якої реляційної СУБД, система визначає найкращий спосіб виконання вашого запиту, і движок SQL обчислює, як інтерпретувати завдання.

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.

У цьому процесі приймають участь різні
компоненти.

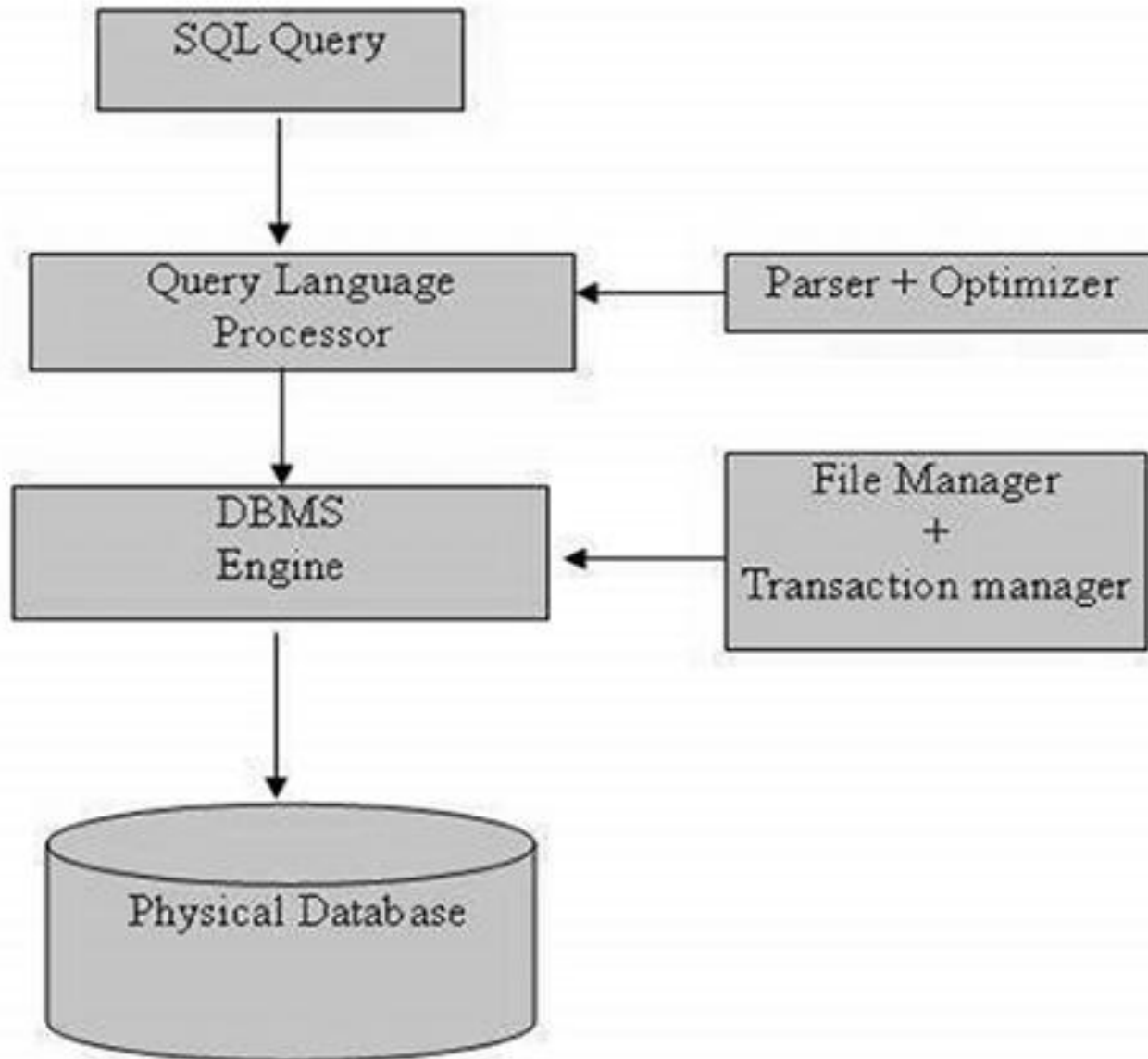
There are various components included in this
process.

Query Dispatcher (диспетчер запитів)

Optimization Engines (інструмент оптимізації)

Classic Query Language (движок запитів)

SQL Query Engine (движок запитів)



SQL data types

Тип даних SQL – це атрибут, який визначає тип даних будь-якого об'єкта. Кожен стовпець, змінна та вираз мають відповідний тип даних в SQL. Ці типи даних використовуються при створенні таблиць. Тип даних для стовпця таблиці обирається відповідно до вимог.

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

Усі оператори SQL починаються з будь-якого з ключових слів, таких як SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW, і всі твердження закінчуються крапкою з комою (;).

SQL є нечутливим до регістру, тобто SELECT і select мають однакове значення 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

Ключове слово SQL DISTINCT використовується разом із оператором SELECT для виключення всіх дубльованих записів та отримання лише унікальних записів.

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

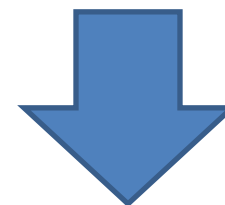
SQL DISTINCT clause

```
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

ЯКИЙ БУДЕ РЕЗУЛЬТАТ ЯКЩО ЗАМІСТЬ
DISTINCT БУДЕ **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

За замовчуванням SQL ORDER BY

використовується для сортування даних в порядку зростання або зменшення на основі одного або декількох стовпців. Деякі бази даних сортують результати запиту за зростанням.

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

Позиція SQL WHERE використовується для вказування умови для отримання даних з однієї таблиці або шляхом об'єднання з кількома таблицями. Якщо ця умова виконується, то тільки вона повертає певне значення з таблиці. WHERE використовують щоб відфільтрувати записи та отримувати лише необхідні 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 є зарезервованим словом або символом, який використовується, перш за все, в позиції WHERE оператора SQL для виконання операцій, таких як порівняння, та арифметичних операцій. Ці оператори використовуються для того, щоб вказувати умови в операторі SQL та виступати в якості сполучень для декількох умов у твердженні.

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

Позиція SQL LIKE використовується для порівняння значення з аналогічними значеннями, використовуючи оператори підстановки. Є два символи підстановки, які використовуються спільно з оператором LIKE:

- * – замість одного або декількох символів (або %);
- ? – замість одного символу (або _).

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

Оператори SQL AND та OR використовуються для об'єднання декількох умов для фільтрації даних у операторі SQL. Ці два оператори називаються кон'юнктивними 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

Позиція SQL GROUP BY використовується у співпраці з оператором SELECT для організації однакових даних у групи. Ця позиція GROUP BY слідує за позицією WHERE в операторі SELECT і передує позиції 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

Функції агрегації / Aggregate functions

Number	Function	Description
1	Avg	Calculates the arithmetic mean of a set of values contained in a specified field on a query. Середнє арифметичне.
2	Count	Calculates the number of records returned by a query. Кількість записів.
3	First	Return a field value from the first or last record in the result set returned by a query. Значення з першого/останнього запису.
4	Last	
5	Min	Return the minimum or maximum of a set of values contained in a specified field on a query. Максимальне/мінімальне значення.
6	Max	
7	Sum	Returns the sum of a set of values contained in a specified field on a query. Сума значень.

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

Позиція HAVING дозволяє визначити умови, які фільтрують групи, що з'являються у результатах.

Позиція WHERE містить умови для вибраних стовпців, тоді як позиція HAVING містить умови для груп, створених у пункті 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

Позиція SQL JOIN використовується для об'єднання записів з двох або більше таблиць у базі даних. JOIN використовується для об'єднання атрибутів з двох таблиць, використовуючи спільні для кожної таблиці значення.

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?

Декартове з'єднання / Cartesian or cross join

Декартове з'єднання повертає декартов добуток з наборів записів з двох або більше об'єднаних таблиць. Таким чином, він прирівнюється до внутрішнього з'єднання, де умова приєднання завжди оцінюється як істина, або де умова з'єднання відсутня у твердженні.

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

Найбільш важливим і часто використовуваним з'єднанням є INNER JOIN. Воно також називається EQUIJOIN.

INNER JOIN створює нову таблицю результатів, об'єднуючи значення стовпчиків з двох таблиць на основі предиката приєднання. Запит порівнює кожен рядок першої таблиці з кожним рядком другої таблиці, щоб знайти всі пари рядків, які задовольняють предикату приєднання. Коли предикат приєднання виконується, значення стовпців для кожної зіставленої пари рядків об'єднуються в рядок результату.

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?**

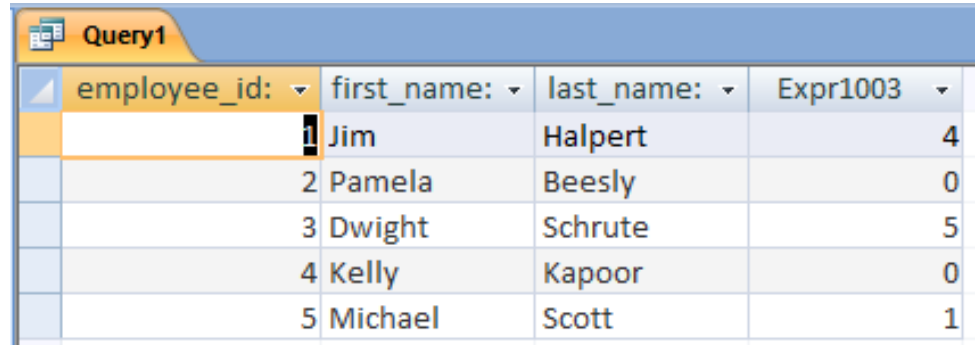
Query 04a

```
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;
```

Query1			
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

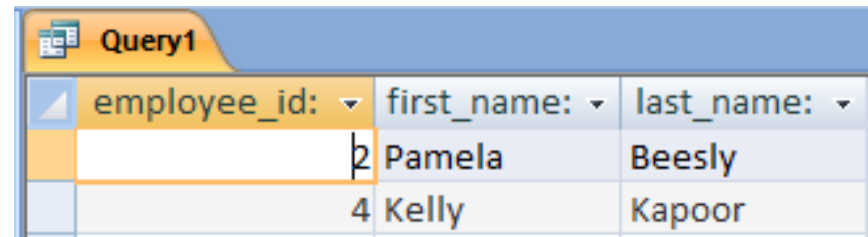
SELECT

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 повертає всі рядки лівої таблиці, навіть якщо у правій таблиці немає відповідностей. Це означає, що якщо умова ON відповідає 0 (нульовим) записам у правій таблиці; з'єднання все одно поверне рядок в результаті, але з NULL у кожному стовпці з правої таблиці.

Це означає, що LEFT JOIN повертає всі значення з лівої таблиці, а також відповідні значення з правої таблиці або NULL у випадку відсутності відповідності для предиката з'єднання.

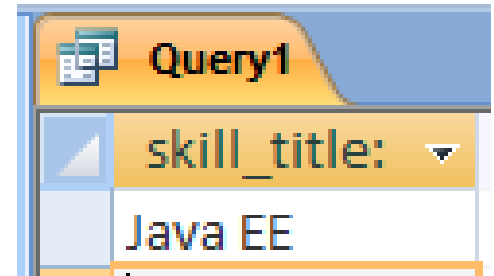
SQL SELECT – LEFT JOIN

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 повертає всі рядки з правої таблиці, навіть якщо у лівій таблиці немає відповідностей. Це означає, що якщо умова ON відповідає 0 (нульовим) записам у лівій таблиці; з'єднання все-таки поверне рядок в результаті, але з NULL у кожному стовпці лівої таблиці.

Це означає, що RIGHT JOIN повертає всі значення з правої таблиці, а також відповідні значення з лівої таблиці або NULL у випадку відсутності відповідності для предиката з'єднання.

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.

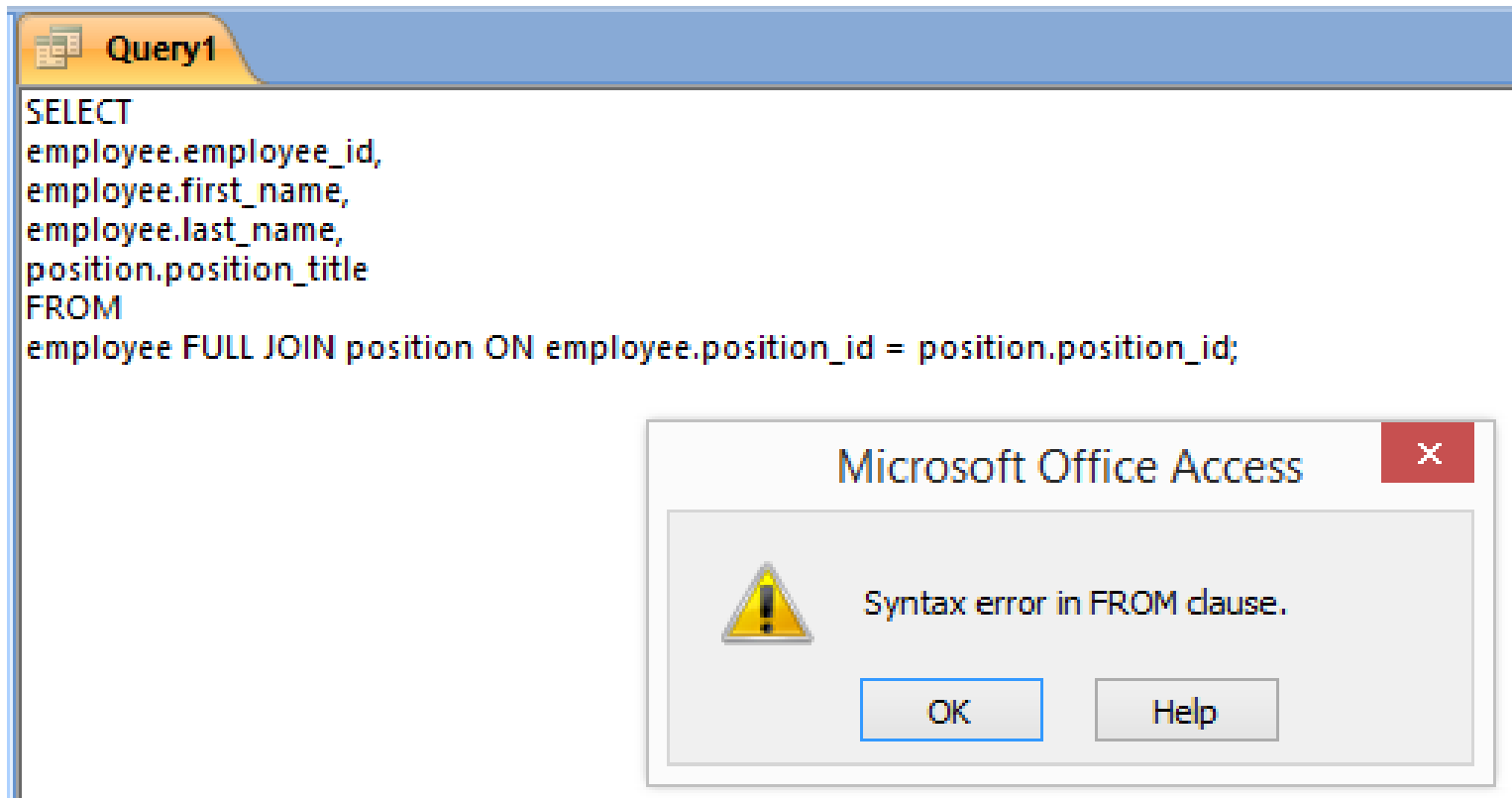
SQL SELECT – FULL JOIN

SQL FULL JOIN поєднує результати як з'єднання LEFT JOIN, так і з'єднання RIGHT JOIN.

Об'єднана таблиця буде містити всі записи з обох таблиць, а також значення NULL для відсутніх відповідностей з обох боків.

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;

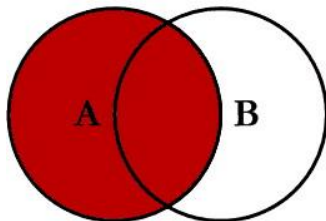
employee							
	employee_id: ▾	first_name: ▾	last_name: ▾	position_id: ▾	birth_date: ▾	onboarding ▾	department ▾
+		1 Jim	Halpert	2	5/12/1990	3/2/2014	1
+		2 Pamela	Beesly	1	11/2/1992	9/23/2015	2
+		3 Dwight	Schrute	3	2/11/1991	12/3/2013	1
+		4 Kelly	Kapoor	4	6/4/1993	3/2/2014	3
+		5 Michael	Scott	5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

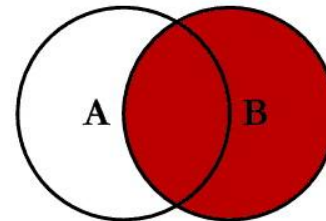
Query1			
employee_id: ▾	first_name: ▾	last_name: ▾	position_tit ▾
			CEO
			Sales Manager
1 Jim	Halpert		SW Eng.
2 Pamela	Beesly		QA Eng.
3 Dwight	Schrute		Sr. SW Eng.
4 Kelly	Kapoor		Syst. Eng.
5 Michael	Scott		PM

SQL JOINS

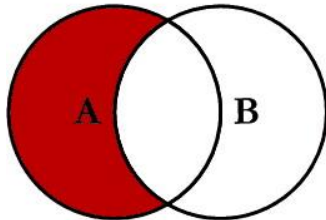
SQL JOINS



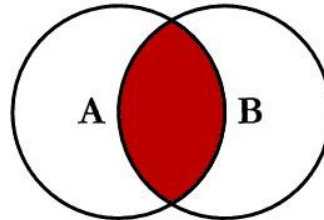
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



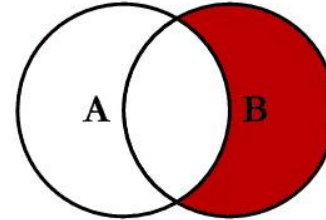
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



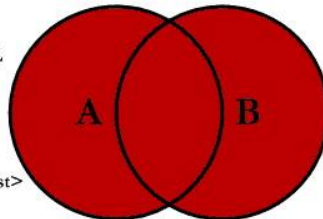
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



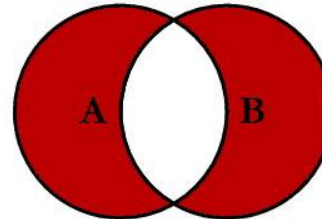
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
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 використовується для об'єднання результатів двох або більше операторів SELECT без повернення дублікатів рядків.

Для коректного використання оператора UNION, кожен оператор SELECT має відповідати умовам:

- однакова кількість стовпців;
- однакова кількість виразів;
- однакові типи даних;
- однаковий порядок стовпців та виразів.

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 використовується для об'єднання результатів двох операторів SELECT, включаючи дублікати рядків.

Ті самі правила, які застосовуються до оператору UNION, застосовуються і до оператора 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 використовується для об'єднання двох операторів SELECT, але повертає рядки тільки з першого оператора SELECT, які ідентичні рядку у другому операторі SELECT. Це означає, що INTERSECT повертає лише загальні рядки, які повертаються двома операторами SELECT.

Як і оператор UNION, застосовуються ті ж правила, коли використовується 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 використовується для об'єднання двох операторів SELECT і повертає рядки з першого оператора SELECT, які не повертаються другим оператором SELECT. Це означає, що EXCEPT повертає лише рядки, які відсутні у результаті другого оператора SELECT.

Як і оператор UNION, застосовуються ті ж правила, коли використовується 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

Вираз SQL INSERT INTO використовується для додавання нових рядків даних до таблиці в базі даних.

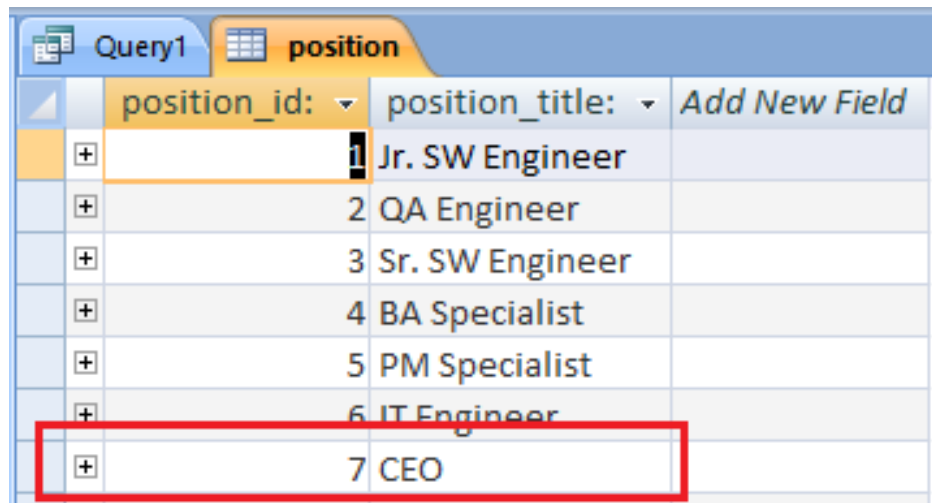
The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

```
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');

Наповнення таблиці з використанням іншої таблиці / Populate one table using another table

Можна заповнити дані в таблиці за допомогою оператора `select`, який звертається до іншої таблиці; якщо інша таблиця має набір полів, які необхідні для заповнення першої таблиці.

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

Запит SQL UPDATE використовується для зміни існуючих записів у таблиці. Вираз WHERE можна використовувати з запитом UPDATE, щоб оновити вибрані рядки, інакше всі рядки будуть також змінені.

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

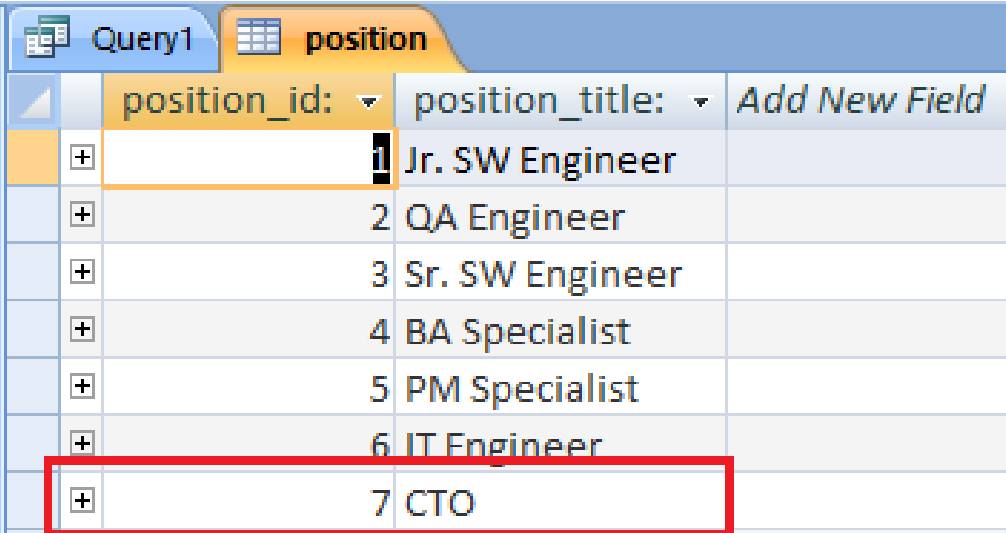
```
UPDATE table_name  
SET   column1 = value1,  
      column2 = value2,  
      ...,  
      columnN = valueN  
WHERE [condition];
```

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

Запит SQL DELETE використовується для видалення існуючих записів з таблиці.

Вираз WHERE можна використовувати у запиті DELETE щоб видалити необхідні рядки, інакше всі записи будуть видалені.

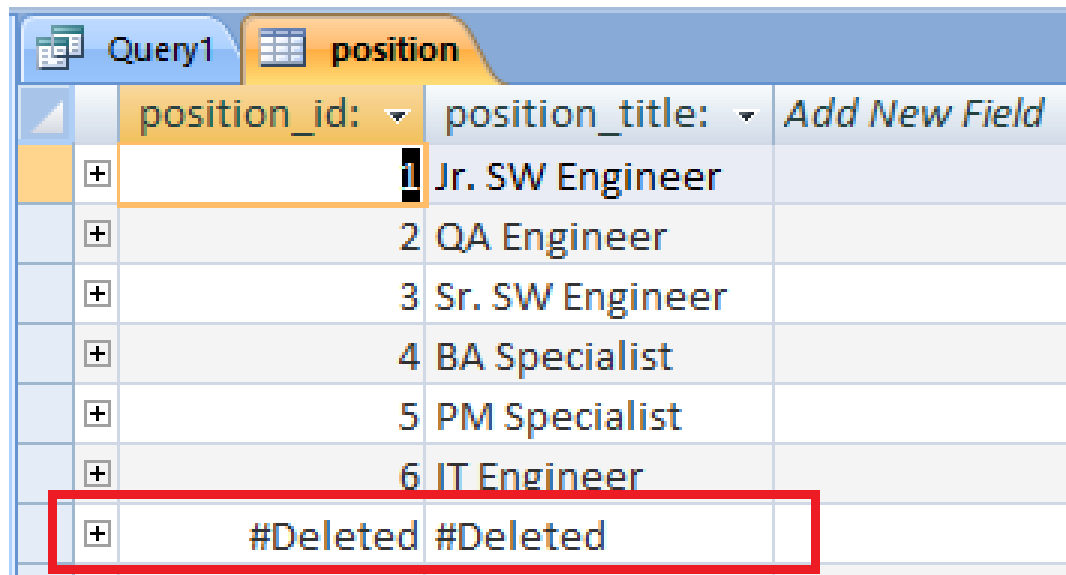
The SQL DELETE Query is used to delete the existing records from a table.

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;



Query1		position	
	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 ...