



Wprowadzenie do Systemów Baz Danych

Introduction to Database Systems

By Maciej Penar ©
mpenar@kia.prz.edu.pl

Agenda

1. AD 2019
2. Relacyjny model danych
3. Wprowadzenie do SQL (DQL)
4. SQL (DDL / DML)
5. Administracja Bazami Danych

AD 2019

Jaki problem rozwiązujemy

Istotną funkcjonalnością dla programisty aplikacji (webowych, mobilnych, desktopowych) jest efektywna* realizacja operacji:

- Zapisu danych
- Odczytu danych

I tak naprawdę nic więcej nie potrzebujemy
Więc.. w czym problem

* Zazwyczaj chodzi o szybkość działania

AD 2019

Problem pierwszy - dane

Nie do końca wiadomo czym **dane** są...

Oficjalna definicja: dane to nazwana wartość

Przykład dwóch danych:

Imię: Maciej

Wiek: 20 lat

Ale być może nasz dane nie mieszczą się w takiej definicji.

AD 2019

Problem drugi - ograniczenia

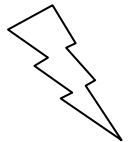
Drugi problem: nie do końca wiadomo czym jest sama **operacja**. Może istnieją jakieś ograniczenia które chcemy jako programiści żeby spełniała

Przykład: przy równoczesnym zapisie danych **osób** i ich **adresów** zamieszkania – w razie niepowodzenia **nie** doszło do zapisu **ani osób, ani adresów**

AD 2019

Problem trzeci – jesteśmy za głupi

Możemy po prostu nie wiedzieć jak napisać
program który zapisuje i odczytuje dane



Pisanie swojej bazy danych to **bardzo, bardzo, bardzo** zła praktyka

AD 2019

Na scenę wkraczają Bazy Danych

Baza Danych (ang. DB) służy do efektywnego przechowywania danych, związków pomiędzy danymi oraz do pilnowania by dostęp do danych mieli tylko uprawnieni użytkownicy.

Przykładem może być Krajowy Rejestr Karny

- Nie wiadomo gdzie składa się dane, ale składa się
- „Panienka z okienka” pilnuje dostępu do danych

Przykładem może być też... książka telefoniczna

Albo... sieć społeczna

AD 2019

Na scenę wkraczają Bazy Danych

System Zarządzania Baz Danych (ang. DBMS) to software zarządzający Bazą Danych

System Baz Danych (ang. DBS) – to połączenie Bazy Danych z Systemem Zarządzania Baz Danych

AD 2019

Na scenę wkraczają Bazy Danych: nowy problem

Możemy po prostu nie wiedzieć jak korzystać z programu z którego pomocą zapisujemy i odczytujemy dane...

Co rodzi szereg problemów: rosnący koszt w chmurach, przepisywanie DB w kodzie, komplikacje architektury (dodawanie cache'a), korzystanie z bibliotek ogólnego przeznaczenia, złe decyzje administracyjne

AD 2019

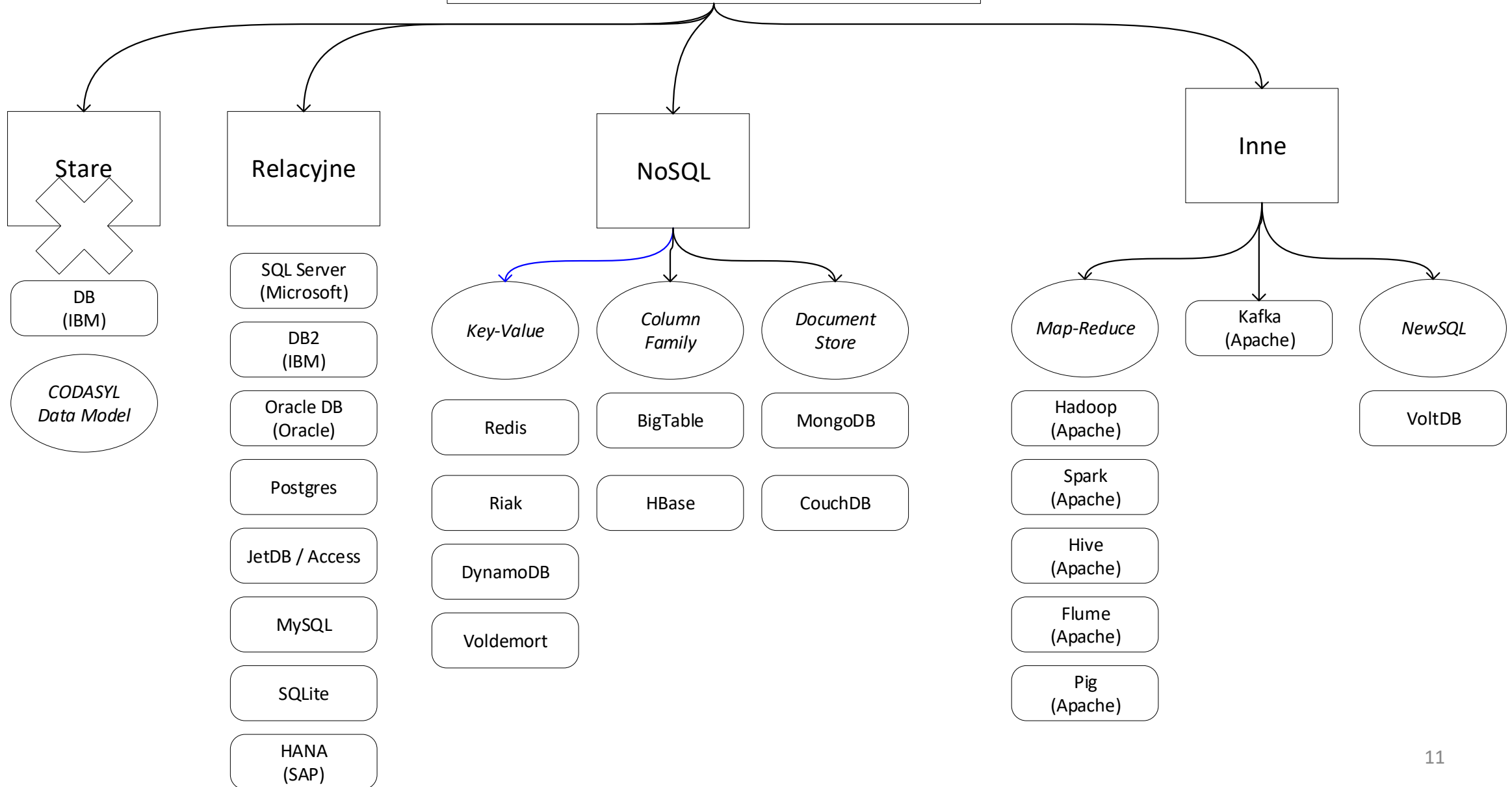
Na scenę wkraczają Bazy Danych: jeszcze jeden problem

Jeszcze jeden problem

**Jest dużo dostawców oprogramowania
Systemów Zarządzania Bazami Danych**

AD 2019

Systemy Zarządzania Bazami Danych



AD 2019

Każdy **rodzaj** bazy danych ma swoje konkretne zastosowanie.

Każda **baza danych** została zaprojektowana pod pewien konkretny cel.

Znowu pojawia się problem braku wiedzy co-do-czego-służy

Relacyjny model danych

Relacyjne bazy danych stanowią bazy ogólnego przeznaczenia

Nazwa pochodzi od **Relacji** z matematyki, a nie od synonimu „związku”.
Relacja to (bez)sensowny koncept którym nie będziemy się tu zajmować.

Relacyjna baza danych (ang. RDBMS) przechowuje dane w relacjach...
które będziemy od teraz nazywać **TABELAMI**

Relacyjny model danych

Tabela składa się z:

- Nazwy
- Nagłówek
- Krotek (danych)

Parę **Nazwa(Nagłówek)** nazywamy schematem

Nagłówek to „uporządkowana” lista kolumn

Relacyjny model danych

Tabela składa się z:

- Nazwy → Osoby
- Nagłówek
- Krotek (danych)

Numer Porządkowy	Imie	Nazwisko	Wiek
1	Jan	Abacki	10
2	Anna	Kabacka	15
3	Kamil	Babacki	25
4	Zuza	Abacka	14

Schemat tej tabeli to: **Osoby(Numer P., Imie, Nazwisko, Wiek)**

Relacyjny model danych

Z punktu widzenia Systemu Informatycznego – schemat to za mało. Chcielibyśmy wiedzieć np. ile pamięci trzeba przeznaczyć na wiersz. Z tego względu częstokroć schemat zapisuje się pionowo, oznaczając typ danych:

Osoby (

Numer P. INT,

Imie VARCHAR(20),

Nazwisko VARCHAR(20),

Wiek INT

)

Relacyjny model danych

Przypominajka typów danych

SQL Typ	Rozmiar w SQL [bajty]	Odpowiednik w Java/C#	Do czego
INT, TINYINT, BIGINT	4, 2, 8	Int, short, long	Liczby całkowite
FLOAT	4	Float	Liczby zmiennoprzecinkowe
DECIMAL(P,S) P – liczba wszystkich cyfr S – liczba cyfr po przecinku	Od 5 do 17	Double	Liczby zmiennoprzecinkowe (precyzja)
VARCHAR(X)	X + 1	String	Ciągi znaków
CHAR(X)	X	char[x]	Ciągi znaków o stałej długości
BIT	1 bit	boolean/bool	Prawda/fałsz
Obiekty (szukać pod UDT)	?	class/struct	Dowolna struktura

Relacyjny model danych

Normalizacja

Założmy że składować dane dotyczące adresu zamieszkania danej osoby. Moglibyśmy to zrobić tak:

Osoby

Numer Porządkowy	Imie	Nazwisko	Wiek	Adres
1	Jan	Abacki	10	Rzeszów, 35-000, ul. Złota 12
2	Anna	Kabacka	15	Stalowa Wola, 35-000, ul. Jakaś 1
3	Kamil	Babacki	25	Stalowa Wola, 35-000, pod Gruszą 111
4	Zuza	Abacka	14	Rzeszów, 35-000, ul. Złota 12

Relacyjny model danych

Normalizacja

(W OLTP) To nie jest dobre z kilku powodów:

- Dane w kolumnie Adres się powtarzają dla 1 i 4 rekordu
- Być może nie zawsze chcemy pokazywać adres
- Być może adres jest błędny i będziemy musieli poprawiać go w kilku miejscach

Relacyjny model danych

Normalizacja

Z pomocą przychodzi normalizacja: czyli podzielenie tabeli na kilka mniejszych

Relacyjny model danych

Normalizacja

Tabele:

Osoby(Numer P., Imie, Nazwisko, Wiek, Adres)

Podzielimy na:

Osoby(Numer P., Imie, Nazwisko, Wiek, IdAdres)

Adres(Id, Adres)

Relacyjny model danych

Normalizacja

Pomysł jest taki żeby rekordy z tabeli Osoby połączyć z rekordami z tabeli Adres w każdą możliwą parę i zostawić tylko te pary dla których spełniony jest warunek:

Adres.Id = Osoba.AdresId

Numer Porządkowy	Imie	Nazwisko	Wiek	IdAdres	ID	Adres
1	Jan	Abacki	10	1	1	Rzeszów, 35-000, ul. Złota 12
2	Anna	Kabacka	15	2	2	Stalowa Wola, 35-000, ul. Jakaś 1
3	Kamil	Babacki	25	3	3	Stalowa Wola, 35-000, pod Gruszą 111
4	Zuza	Abacka	14	1	1	Rzeszów, 35-000, ul. Złota 12

Relacyjny model danych

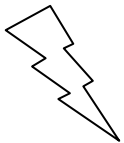
Normalizacja



Poprawa adresów wymaga poprawy w pojedynczym miejscu



Polepsza wydajność zapisu/aktualizacji danych



Normalizacja **pogarsza** wydajność odczytów

Relacyjny model danych

Normalizacja Osoby

Numer Porządkowy	Imie	Nazwisko	Wiek	IdAdres
1	Jan	Abacki	10	1
2	Anna	Kabacka	15	2
3	Kamil	Babacki	25	3
4	Zuza	Abacka	14	1

Adres

ID	Adres
1	Rzeszów, 35-000, ul. Żłota 12
2	Stalowa Wola, 35-000, ul. Jakaś 1
3	Stalowa Wola, 35-000, pod Gruszą 111

Relacyjny model danych

Normalizacja - dygresja

Adres

ID	Adres
1	Rzeszów, 35-000, ul. Złota 12
2	Stalowa Wola, 35-000, ul. Jakaś 1
3	Stalowa Wola, 35-000, pod Gruszą 111

Dygresja:

Tu ktoś może słusznie zauważyć że w wartości kolumny Adres mają skłonność do powtarzania się np. „Stalowa Wola”. Z tego względu **można** traktować tę tabelę jako zdenormalizowaną.

Na ogół schemat Adresu to: **Adres(Id, Miasto, Kod-Pocztowy, Województwo, Gmina, Ulica, ...)**

Na takim schemacie łatwiej można zauważyć powtarzalność danych np. w kolumnie Miasto dla różnych kodów pocztowych.

Relacyjny model danych

Normalizacja

Warto wprowadzić jeszcze dwa pojęcia które pomagają w skróceniu notacji:

- **Klucz główny** – czyli kolumna która posiada **obligatoryjnie uzupełnione, unikalne wartości**, które służą do identyfikacji wierszy (bytu). Na schemacie oznaczany jako podkreślenie.
- **Klucz obcy** – czyli kolumna której wartości stanowią klucz główny w innej tabeli. Na schemacie oznaczany jako #.

Czasem jest tak że jedna kolumna nie identyfikuje wiersza, wtedy wybierana jest lista kolumn – tzw. **klucz złożony**

Relacyjny model danych

Normalizacja

W tej notacji nasz schemat oznaczylibyśmy jako:

Osoby(**Numer P.**, Imie, Nazwisko, Wiek, **#IdAdres**)

Adres(**Id**, Adres)

Spostrzeżenie:

Osoba ma co najwyżej jeden adres

Adres może być przypisany do wielu Osób

Jest to przykład realizacji związku jeden-do-wielu

Relacyjny model danych

Rodzaje związków

- Związek jeden-do-jednego

Np. posiadanie paszportu przez osobę

- Związek jeden-do-wielu

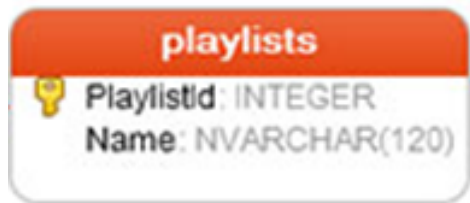
Np. pokoje w budynku

- Związek wiele-do-wielu

Np. Filmy i reżyserzy

Relacyjny model danych

W praktyce dostaje się diagramy obrazujące **schemat bazy danych**, czyli wszystkie schematy tabel. Na przykład taki diagram:



Odpowiada schematowi: playlists(PlaylistId, Name)

Relacyjny model danych

Taki diagram odpowiada schematom:

- `playlists(PlaylistId, Name)`
- `playlist_track(PlaylistId, TrackId)`



Linia łącząca oznajmia z iloma wierszami baza danych pozwala się połączyć tabelom.



Co najwyżej jeden



Dokładnie jeden



Co najmniej jeden



Dowolna liczba

Relacyjny model danych

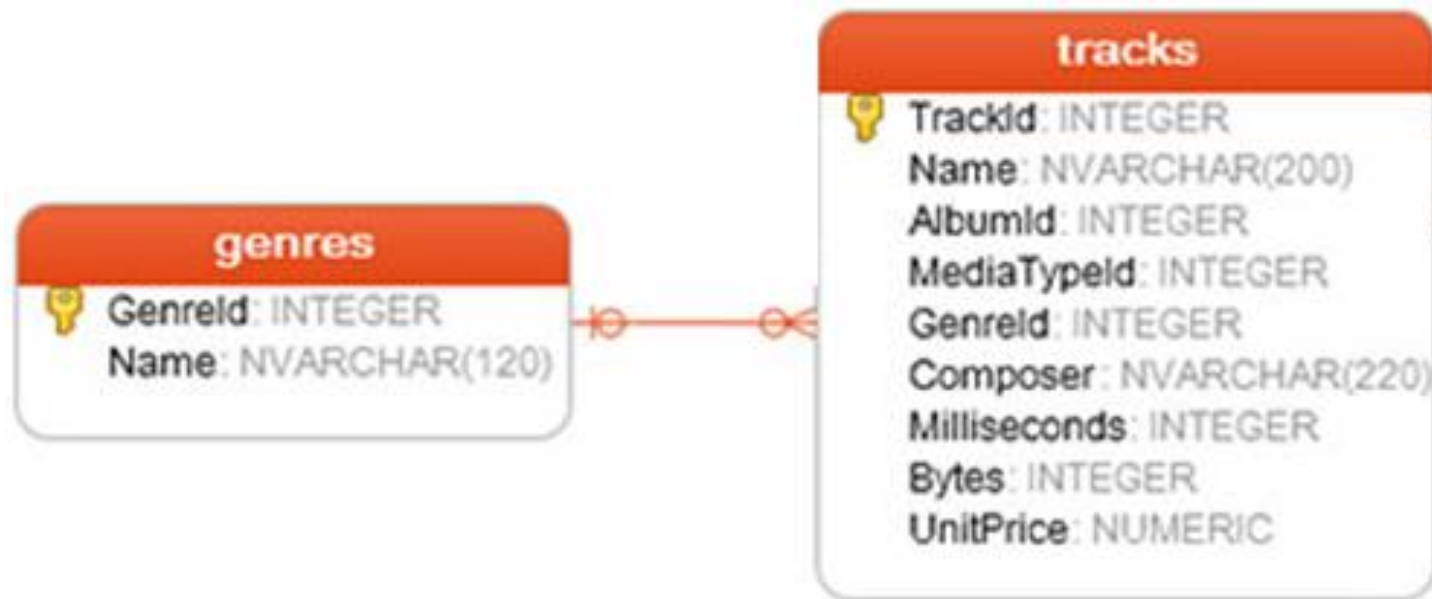
Czytamy zawsze od strony tabeli, pomijając sąsiadującą licznosc. Diagram obok możemy przeczytać na dwa sposoby:

1. Każdy wiersz w `playlists` łączy się z **co najwyżej jednym** wierszem `playlist_track`
2. Każdy wiersz w `playlist_track` łączy się z **dokładnie jednym** wierszem `playlist_track`



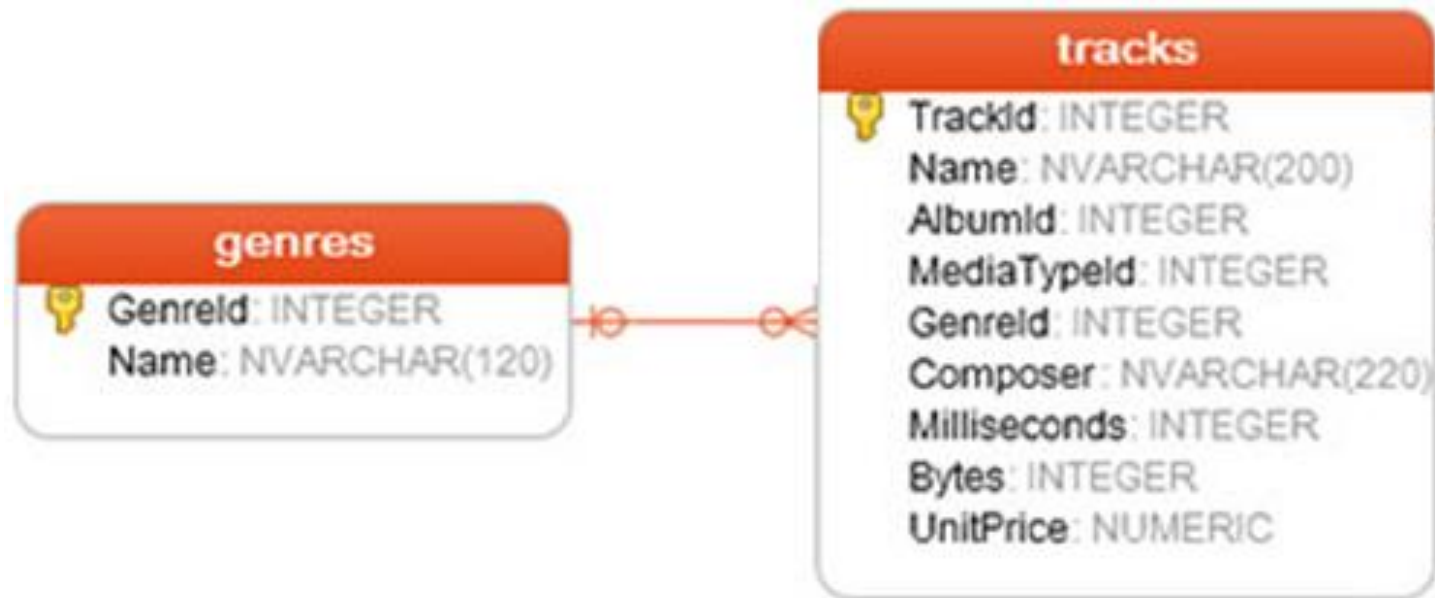
Relacyjny model danych

1. Każdy wiersz w genres łączy się **dowolną liczbą** wierszy z tracks
2. Każdy wiersz w tracks łączy się **z co najwyżej jednym** wierszem genres



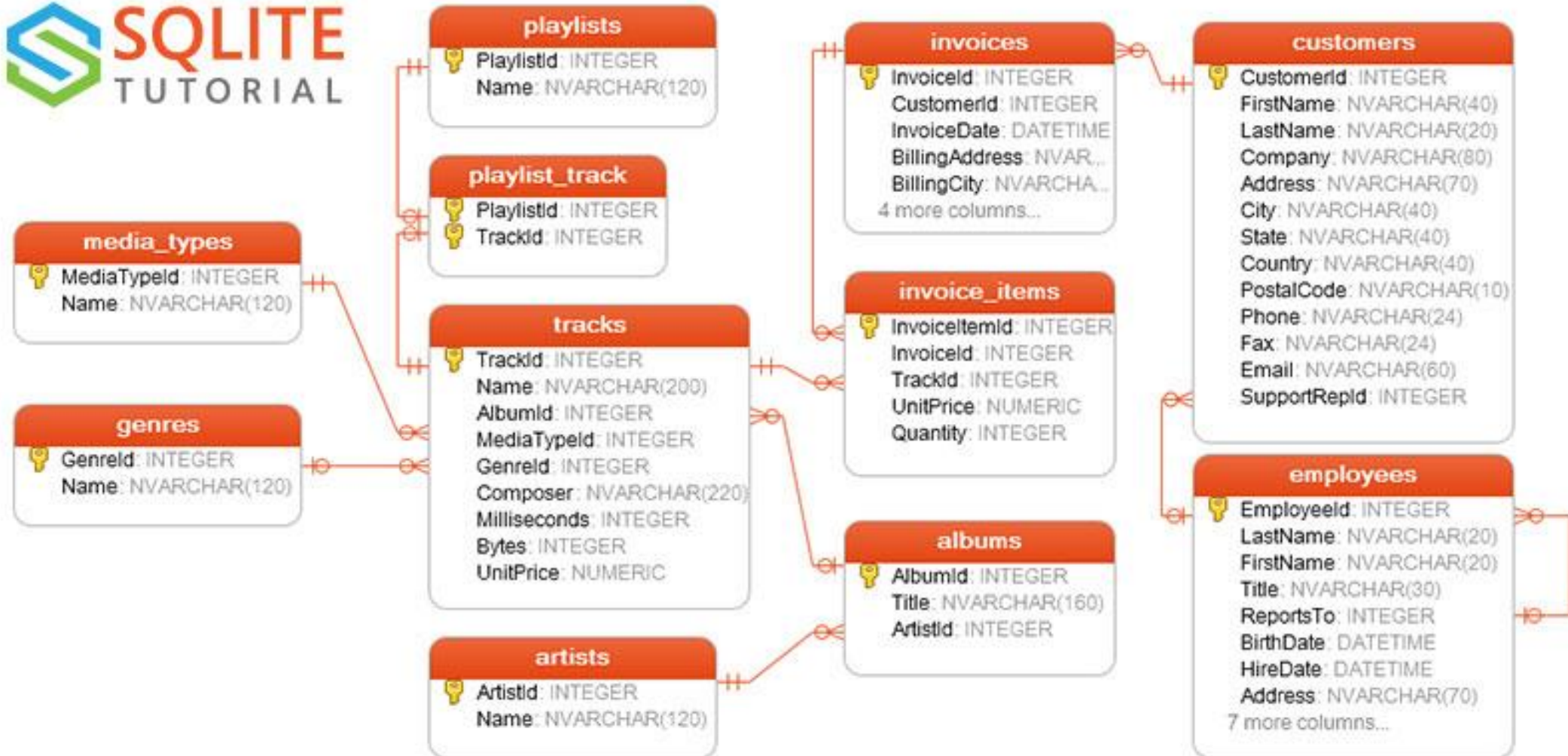
Relacyjny model danych

Z diagramu nie wynika jak łączyć tabele, ale ratuje nas konwencja:
„[nazwa pola]Id” oznacza klucze



Relacyjny model danych

ChinookDB



Wprowadzenie do SQL (DQL)

Mamy „z grubsza” koncepcję jak wyglądają tabele – ale nie potrafimy wyciągać z nich danych

W RDBMS służy za to
Strukturalizowany Język Zapytań (SQL).

Wprowadzenie do SQL (DQL)

SQL:

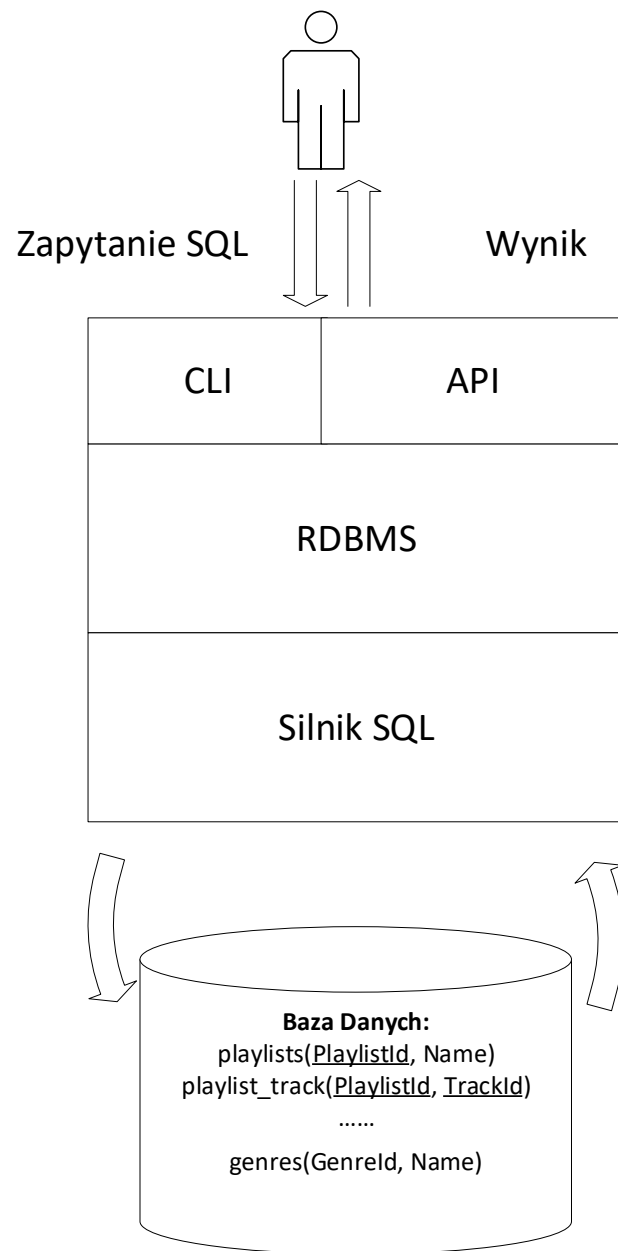
- Jest deklaratywnym językiem – nie piszemy algorytmu, tylko definiujemy logiczny wynik który potrzebujemy
- Zapytanie SQL można traktować jako przekształcenie tabel

Ogólnie rzecz biorąc SQL:

- Na wejściu dostaje wybrane schematy z Bazy danych
- Na wyjściu zwraca tabele o pewnym schemacie (tzw. **schemat wynikowy**)

Wprowadzenie do SQL (DQL)

Koncept



Wprowadzenie do SQL (DQL)

Kilka not:

- Projektanci SQL chcieli żeby język był zbliżony do języka naturalnego
- Najlepiej słowa kluczowe pisać KAPITALIKAMI

Wprowadzenie do SQL (DQL)

SELECT

Podstawowym słowem kluczowym jest **SELECT** (wybierz) które służy do definiowania schematu wynikowego zapytania.

Przykłady bardzo krótkich zapytań SQL to:

1. **SELECT 1**
2. **SELECT ,aaa'**

Oba zapytania zwracają jeden wiersz zawierający odpowiednio 1 oraz ,aaa'. Schemat wynikowy to: Wynik([bez nazwy])

Wprowadzenie do SQL (DQL)

ALIASOWANIE

Żeby uzyskać w schemacie wynikowym nazwy kolumny czytelne dla użytkownika końcowego wykorzystywane jest słowo kluczowe **AS**

1. **SELECT 1 AS ToBezsensownaLiczba;**
2. **SELECT ,aaa' AS ToBezsensownyCiagZnakow;**

Schematy wynikowe to:

1. Wynik(ToBezsensownaLiczba)
2. Wynik(ToBezsensownyCiagZnakow)

Wprowadzenie do SQL (DQL)

FROM

Bardzo istotnym słowem kluczowym jest **FROM** (z) które służy do definiowania źródła danych / schematów wejściowych.

Założmy że z tabeli utwory (**tracks**) chcemy wybrać: Nazwę (Name) oraz Kompozytora (Composer).

tracks	
	TrackId: INTEGER
	Name: NVARCHAR(200)
	AlbumId: INTEGER
	MediaTypeId: INTEGER
	GenreId: INTEGER
	Composer: NVARCHAR(220)
	Milliseconds: INTEGER
	Bytes: INTEGER
	UnitPrice: NUMERIC

Wprowadzenie do SQL (DQL)

FROM

Zapytanie przybierze postać:

SELECT

Name,

Composer

FROM

tracks;



Co w wolnym tłumaczeniu oznacza:

WYBIERZ nazwisko i kompozytora **Z** tracks

Wprowadzenie do SQL (DQL)

FROM

Jak ktoś chce to może aliasować

SELECT

Name AS NazwaUtworu,
Composer

FROM

tracks;

Co zmienia schemat wynikowy z: Wynik(**Name**, Composer)

Na: Wynik(**NazwaUtworu**, Composer)

Wprowadzenie do SQL (DQL)

SELECT – kalkulacje

Klauzuli **SELECT** użyliśmy na dwa sposoby:

1. Wsadziliśmy stałą: SELECT [stała]
2. Wybraliśmy kolumnę: SELECT [nazwa kolumny z FROM]

SELECT przyjmuje też obliczenia wykonywane na poziomie wiersza z wykorzystaniem dowolnych stałych i operatorów arytmetycznych np. SELECT 60 * [nazwa kolumny z FROM]

Wprowadzenie do SQL (DQL)

SELECT – kalkulacje

Założmy że z tabeli utwory (**tracks**) chcemy wybrać: Nazwę (Name) oraz długość utworu podaną w minutach – kolumna Milliseconds ma długość podaną w milisekundach.

SELECT

Name,
Milliseconds/(1000*60) AS Duration

FROM

tracks;

tracks	
	TrackId: INTEGER
	Name: NVARCHAR(200)
	AlbumId: INTEGER
	MediaTypeId: INTEGER
	GenreId: INTEGER
	Composer: NVARCHAR(220)
	Milliseconds: INTEGER
	Bytes: INTEGER
	UnitPrice: NUMERIC

Schemat Wynikowy: Wynik(Name, Duration)

Wprowadzenie do SQL (DQL)

WHERE

Kolejnym istotnym słowem kluczowym jest **WHERE** (gdzie) które służy do filtrowania wierszy.

Założmy że z tabeli utwory (**tracks**) chcemy wybrać: Nazwę (Name) tych utworów które są krótsze niż 1 minuta.

tracks	
	TrackId: INTEGER
	Name: NVARCHAR(200)
	AlbumId: INTEGER
	MediaTypeId: INTEGER
	GenreId: INTEGER
	Composer: NVARCHAR(220)
	Milliseconds: INTEGER
	Bytes: INTEGER
	UnitPrice: NUMERIC

Wprowadzenie do SQL (DQL)

WHERE

Zapytanie przybierze postać:

SELECT

Name

FROM

Tracks

WHERE

$\text{Milliseconds} / (1000 * 60) < 1;$



Co w wolnym tłumaczeniu oznacza:

WYBIERZ nazwisko **Z** tracks **GDZIE WIERSZE** mają mniej niż 1 w Minutach

Wprowadzenie do SQL (DQL)

WHERE – inne warunki

Operatory porównania dostępne w klauzuli WHERE to:

- =
- <
- >
- <=
- >=
- <> (różny)
- **LIKE** (wzorce)
- **IN** (wartość ze zbioru)
- **BETWEEN x AND y**
- **IS NULL**

Warunki można łączyć **AND** i **OR** oraz przeczyć **NOT**

Wprowadzenie do SQL (DQL)

SELECT – brzydkie *

Klauzuli **SELECT** użyliśmy na trzy sposoby:

1. Wsadziliśmy stałą: SELECT [stała]
2. Wybraliśmy kolumnę: SELECT [nazwa kolumny z FROM]
3. Wykonaliśmy kalkulacje

SELECT przyjmuje też specjalną wartość * oznaczającą – wszystkie kolumny wynikające z klauzuli FROM

Wprowadzenie do SQL (DQL)

SELECT – brzydkie *

SELECT

*

FROM

tracks;

tracks	
	TrackId: INTEGER
	Name: NVARCHAR(200)
	AlbumId: INTEGER
	MediaTypeId: INTEGER
	GenreId: INTEGER
	Composer: NVARCHAR(220)
	Milliseconds: INTEGER
	Bytes: INTEGER
	UnitPrice: NUMERIC

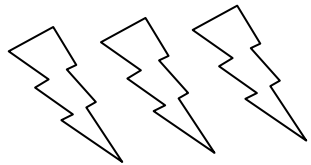
Schemat Wynikowy: Wynik(TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, Milliseconds, Bytes, UnitPrice)

Wprowadzenie do SQL (DQL)

SELECT – brzydkie *

**SELECT * prosi tabelę o wykonanie specjalnej operacji
tzw. skanowania**

**W aplikacjach w 99,9% przypadkach nie wolno
zostawiać kodu z SELECT ***



Wprowadzenie do SQL (DQL)

FROM – zapytanie jako źródło danych

Jeśli wynik zapytania jest tabelą, to czy można użyć go w klauzuli FROM? Tak, ale na ogół nie ma to sensu. Warunkiem jest to że trzeba nadać wynikowi alias:

SELECT

*

FROM

(SELECT Name FROM Tracks) AS Temp;

Schemat Wynikowy: Wynik(Name)

Ponieważ Schemat Wynikowy Temp to: Temp(Name)

Wprowadzenie do SQL (DQL)

WITH – źródło danych

Jeśli zachodzi potrzeba wykorzystywania wyników zapytań jako źródeł, to lepiej skorzystać ze składni tabel pomocniczych (ang. CTE):

WITH Tymczasowa AS(

SELECT

 Name,

 Milliseconds/(1000*60) **AS** DurationMin

FROM

 Tracks)

SELECT Name **FROM** Tymczasowa **WHERE** DurationMin < 1;

Wprowadzenie do SQL (DQL)

WITH – źródło danych

Dobre użycie CTE....:

```
WITH Tymczasowa AS(  
  SELECT  
    Name,  
    Milliseconds/(1000*60) AS DurationMin  
  FROM  
    Tracks)  
SELECT Name  
FROM Tymczasowa  
WHERE  
  DurationMin < 1  
  OR DurationMin > 10;
```

Wybiera nazwy utworów które trwają mniej niż 1 minuta lub dłużej niż 10 minut.

Wprowadzenie do SQL (DQL)

WITH – źródło danych

...bo tu kalkulacje musiałby być wykonywane dwukrotnie:

SELECT Name

FROM tracks

WHERE

Milliseconds/(1000*60) < 1

OR Milliseconds/(1000*60) > 10;

Wybiera nazwy utworów które trwają mniej niż 1 minuta lub dłużej niż 10 minut.

Wprowadzenie do SQL (DQL)

WITH – źródło danych

Przeciętne użycie CTE....:

```
WITH Tymczasowa AS(  
  SELECT  
    Name,  
    Milliseconds/(1000*60) AS DurationMin  
FROM  
  Tracks)  
SELECT Name  
FROM Tymczasowa  
WHERE DurationMin BETWEEN 1 AND 2;
```

Wybiera nazwy utworów które trwają od 1 do 2 minut.

Wprowadzenie do SQL (DQL)

WITH – źródło danych

...bo można pominąć CTE:

SELECT Name

FROM tracks

WHERE Milliseconds/(1000*60) **BETWEEN 1 AND 2;**

Wybiera nazwy utworów które trwają od 1 do 2 minut.

Wprowadzenie do SQL (DQL)

LIMIT/TOP – ograniczanie zbioru

Czasem zachodzi potrzeba zwrócenia pewnej znanej uprzednio liczby rekordów z Bazy Danych.

Standard SQL nie przewiduje takiej możliwości, ale dostawcy DBMS'ów zazwyczaj tak. Załóżmy że chcemy pobrać tylko 5 pierwszych rekordów. W różnych DBMS'ach wygląda to tak:

SQLite:

```
SELECT *  
FROM tracks  
LIMIT 5;
```

SQL Server:

```
SELECT TOP 5 *  
FROM tracks;
```

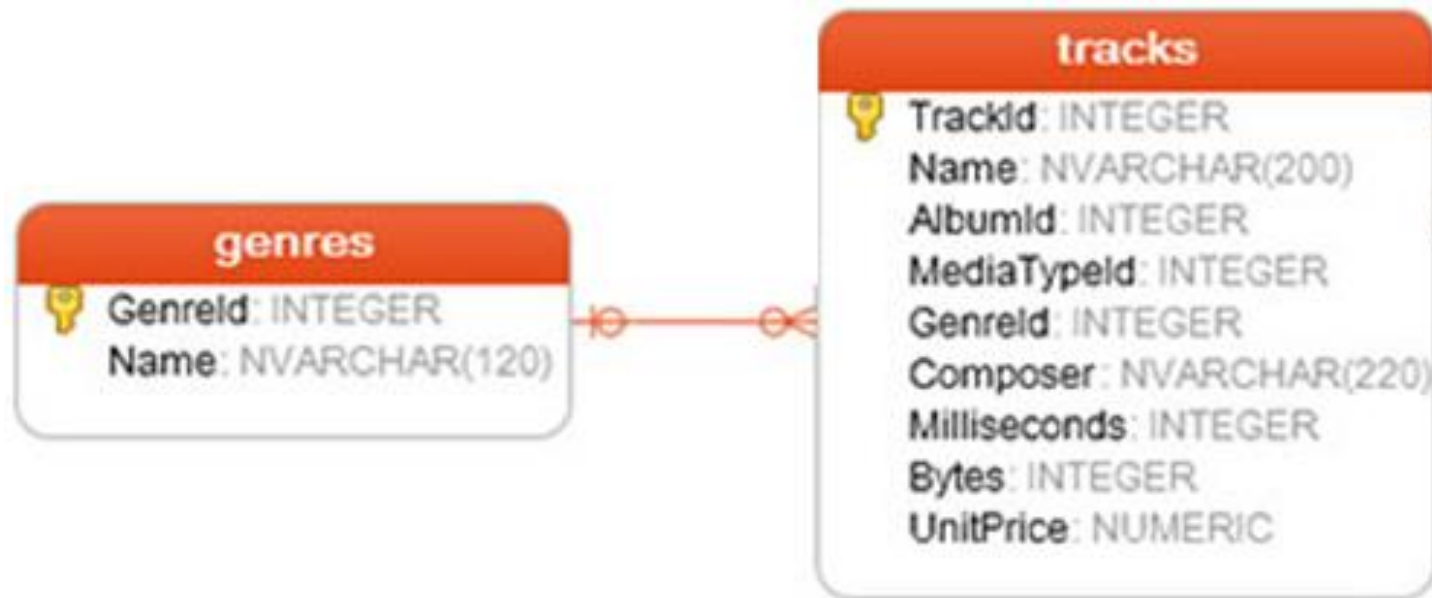
DB2:

```
SELECT *  
FROM tracks  
FETCH FIRST 5  
ROWS ONLY;
```

Wprowadzenie do SQL (DQL)

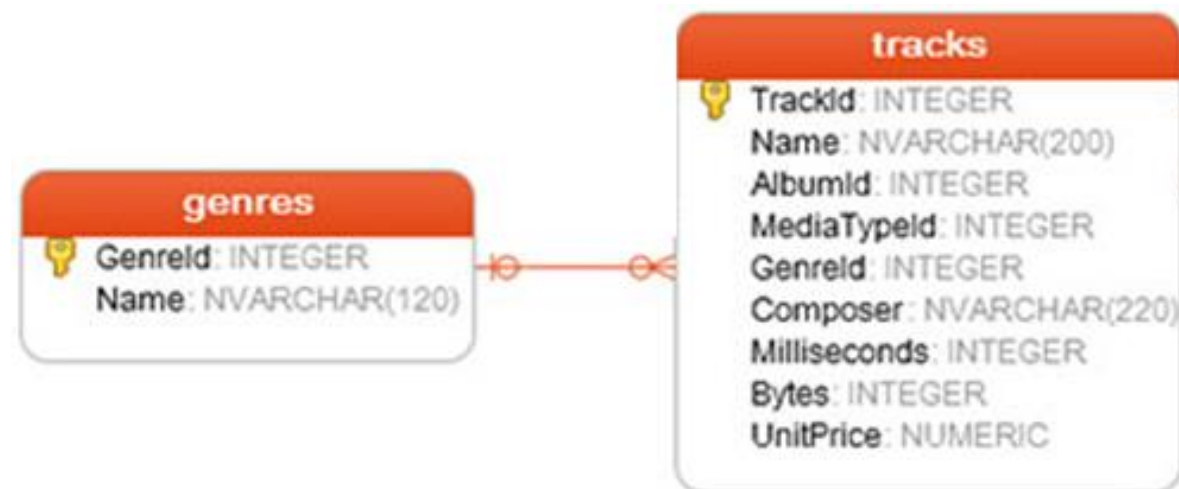
FROM – kilka źródeł danych

Założmy że chcemy wykonać zapytanie: wybierz wszystkie kolumny wyniku złączenia utworów (tracks) i gatunków (genres) – zwrócić tylko 10 pierwszych rekordów.



Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych



Pamiętamy że duża liczba tabel jest spowodowana **normalizacją** bazy danych

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych

Metoda 1

Stara składnia złączeń

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 1, krok 1

Pierwszy krok polega na wypisaniu w klauzuli FROM wszystkich tabel po przecinku:

SELECT

*

FROM

tracks,
genres;

Wynik takiego zapytania to **wszystkie kombinacje** wierszy z **tracks** i **genres** – także te bez sensu - nie tego szukamy

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 1, krok 2

Drugi krok polega na dodaniu odpowiedniego warunku w klauzuli WHERE

SELECT

*

FROM

tracks,

genres

WHERE

tracks.GenreId = genres.GenreId;

Wynik takiego zapytania to **wszystkie kombinacje** wierszy z **tracks** i **genres** – zgodne co do kolumn GenreId z **tracks** i **genres**

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 1

Ostateczna forma zapytania (z klauzulą LIMIT):

SELECT

*

FROM

tracks,
genres

WHERE

tracks.GenreId = genres.GenreId

LIMIT 10;

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych

Metoda 2

Nowa składnia złączeń

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 2

Spostrzeżenie – zauważmy że wszystkie klucze obce nie mają większego znaczenia dla samych aplikacji – stanowią tzw. metadane (dane opisujące dane).

Staramy się być uporządkowani – nie chcemy mieszać **warunków złączeń** z **warunkami filtrowania**:

- W klauzuli FROM chcielibyśmy napisać jak rekonstruować wiersze (warunki złączeń)
- W klauzuli WHERE chcielibyśmy napisać filtry odpowiednie dla aplikacji

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 2

Z pomocą przychodzi wyrażenie INNER JOIN dla klauzuli FROM.

Składnia to:

FROM

[tabela] [alias]

INNER JOIN [tabela] [alias] **ON** [warunek]

Można dodać dowolną liczbę złączeń:

[tabela] [alias]

INNER JOIN [tabela] [alias] **ON** [warunek]

...

INNER JOIN [tabela] [alias] **ON** [warunek]

Wprowadzenie do SQL (DQL)

FROM – kilka źródeł danych, metoda 2

Ostateczna forma zapytania (z klauzulą LIMIT):

SELECT

*

FROM

tracks

INNER JOIN genres **ON** tracks.GenreId = genres.GenreId

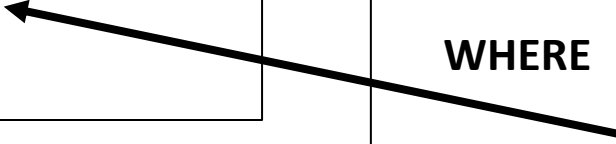
LIMIT 10;

Wprowadzenie do SQL (DQL)

FROM – porównanie metod złączeń

```
SELECT
    *
FROM
    tracks
    INNER JOIN genres ON tracks.GenreId = genres.GenreId
LIMIT 10;
```

```
SELECT
    *
FROM
    tracks,
    genres
WHERE
    tracks.GenreId = genres.GenreId
LIMIT 10;
```



Wprowadzenie do SQL (DQL)

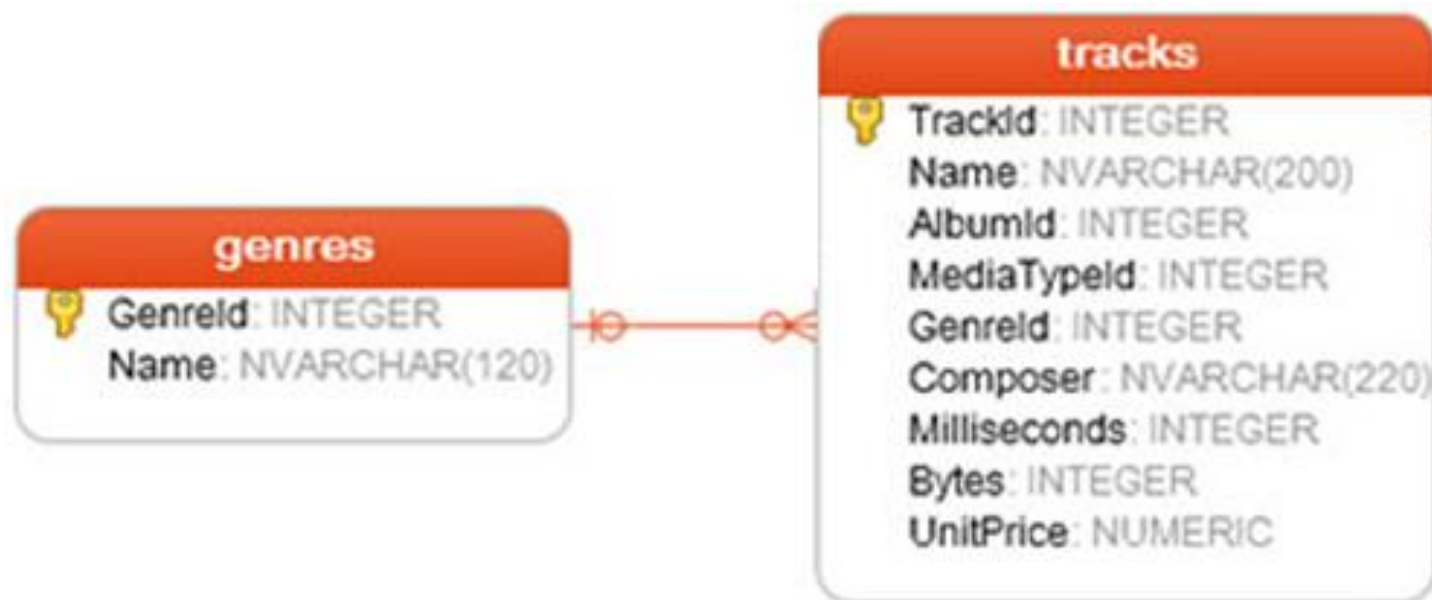
Wykonywanie złączeń

- Złączenia są **bardzo** kosztowne dla DBMS
- Istnieje kilka rodzajów złączeń (zajmiemy się nimi później):
 - INNER JOIN
 - LEFT JOIN
 - RIGHT JOIN
 - FULL OUTER JOIN
- Czasem złączeń można uniknąć

Wprowadzenie do SQL (DQL)

Unikanie złączeń

Założmy że chcemy wykonać zapytanie: wybierz nazwy (Name) utworów (tracks) z gatunku (genres) ,Pop'.



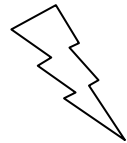
Wprowadzenie do SQL (DQL)

Unikanie złączeń

Intuicja podpowiada (bardzo źle podpowiada):

SELECT

Name



Nie zadziała bo nie wiadomo z której tabeli pochodzi Name

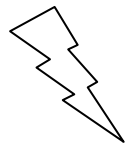
FROM

tracks

INNER JOIN genres **ON** tracks.GenreId = genres.GenreId

WHERE

Name = ,Pop';



Nie zadziała bo nie wiadomo z której tabeli pochodzi Name

Wprowadzenie do SQL (DQL)

Unikanie złączeń

Intuicja podpowiada (lepiej niż poprzednio) – dodajemy nazwy tabel:

SELECT

tracks.Name

FROM

tracks

INNER JOIN genres **ON** tracks.GenreId = genres.GenreId

WHERE

genres.Name = 'Pop';

Wprowadzenie do SQL (DQL)

Unikanie złączeń

Intuicja podpowiada (jeszcze lepiej niż poprzednio) – dodajemy aliasy:

SELECT

t.Name

FROM

tracks **AS** t

INNER JOIN genres **AS** g **ON** tracks.GenreId = genres.GenreId

WHERE

g.Name = 'Pop';

Wprowadzenie do SQL (DQL)

Unikanie złączeń

Ale można lepiej – zamienić złączenie na podzapytanie:

SELECT

Name

FROM

tracks

WHERE

GenreId **IN** (**SELECT** GenreId **FROM** genres **WHERE** g.Name = ,Pop');

Podzapytanie
(wykonane raz)

Wprowadzenie do SQL (DQL)

Unikanie złączeń – analiza 1

Mamy dwa zapytania:

1. Nadzapytanie – pobierz wszystkie nazwy utworów których identyfikatory gatunków to ?
2. Podzapytanie – pobierz identyfikatory gatunków które nazywają się Pop

```
SELECT
    Name
FROM
    tracks
WHERE
    GenreId IN (SELECT GenreId FROM genres WHERE g.Name = ,Pop');
```

Wprowadzenie do SQL (DQL)

Unikanie złączeń – analiza 2

Jeśli wiemy że podzapytanie zwróci co najwyżej 1 rekord to możemy zamiast IN użyć =

Jeśli z **danych** wynika że podzapytanie zwróci więcej niż 1 rekord to Baza Danych zgłosi wyjątek

SELECT

Name

FROM

tracks

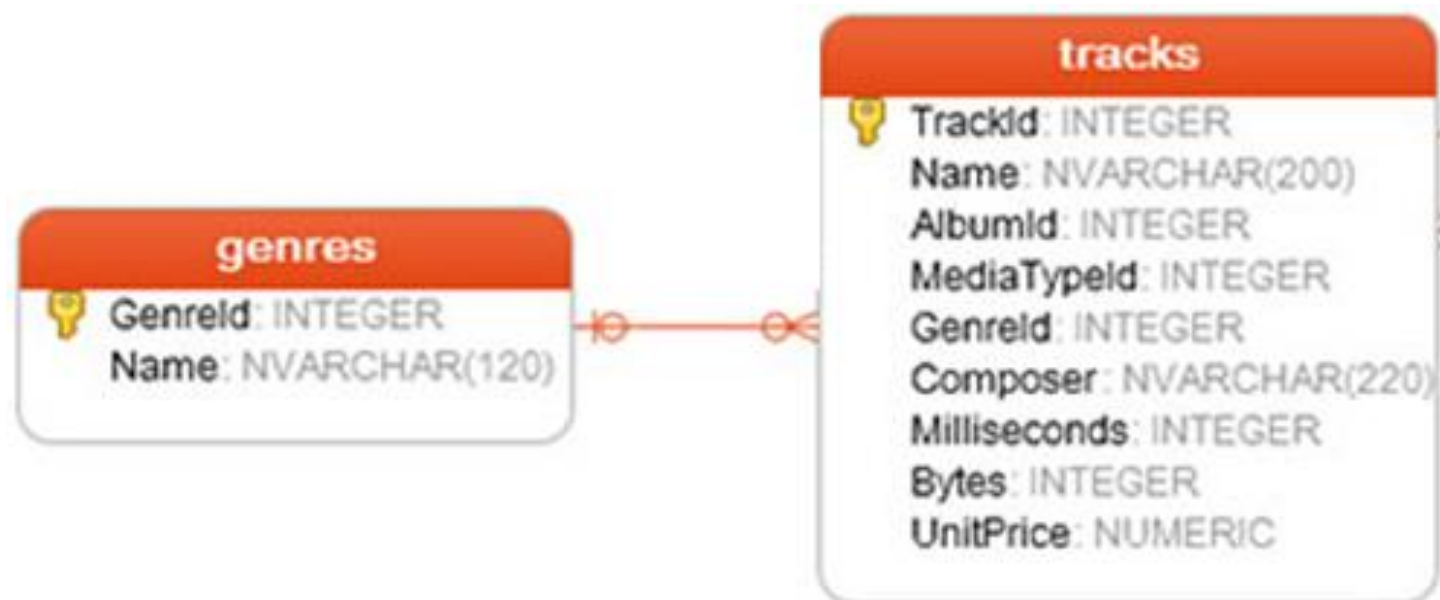
WHERE

GenreId = (**SELECT** GenreId **FROM** genres **WHERE** g.Name = ,Pop');

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Założmy że chcemy wykonać zapytanie: ile jest utworów w ramach gatunków



Służy do tego specjalne słowo kluczowe **GROUP BY**.

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Na przykład zapytanie:

SELECT

g.Name

FROM

tracks t

INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

GROUP BY

g.Name;

Zwróci tyle wierszy ile jest gatunków. Ale zaistnieje niuans...

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

... każdy wiersz z wyniku klauzuli **GROUP BY** jest powiązany z pewną kolekcją wierszy – zgodnych co do wartości zgrupowania.

Nie możemy dostać się bezpośrednio do wartości tych wierszy, ale z użyciem **funkcji agregujących** możemy wyliczyć wartość opisującą zgrupowanie.

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Przykłady dostępnych funkcji to:

- **COUNT(*)** – wyliczająca liczbę wszystkich wartości
- **COUNT(DISTINCT [kolumna])** – wyliczająca liczbę unikatowych wartości w kolumnie
- **SUM([kolumna])** – wyliczająca sumę
- **AVG([kolumna])** – wyliczająca średnią wartość
- **MIN([kolumna])** – wyliczająca minimalną wartość
- **MAX([kolumna])** – wyliczająca maksymalną wartość

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Wracając do przykładu:

```
SELECT
    g.Name,
    COUNT(*) AS NumberOfTracksPerGenres
FROM
    tracks t
    INNER JOIN genres g ON tracks.GenreId = genres.GenreId
GROUP BY
    g.Name;
```

Zwróci pary: **nazwa gatunku i liczba utworów per gatunek.**

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Bardziej szczegółowo:

FROM

tracks t

INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

To złączenie powoduje zwrócenie wyniku:

genres.GenreId	genres.Name	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
1	Metal	1		1
1	Metal	2		1
1	Metal	3		1
2	Rock	4		2
....

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Dodanie **GROUP BY genres.Name** powoduje utworzenie w pamięci:

genres.Name
Metal
Rock

genres.GenreId	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
1	1		1
1	2		1
1	3		1

genres.GenreId	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
2	4		2
....

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Dodanie **SELECT COUNT(*)** powoduje wyliczenie ile rekordów jest w zgrupowaniach – dla Metal byłoby to 3 rekordy.

genres.Name	genres.GenreId	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
Metal	1	1		1
	1	2		1
	1	3		1
Rock	2	4		2

Wprowadzenie do SQL (DQL)

GROUP BY – grupowanie danych

Funkcje agregujące takie jak **SUM([kolumna])** aplikujemy do zgrupowań np. **SUM(tracks.TrackId)**

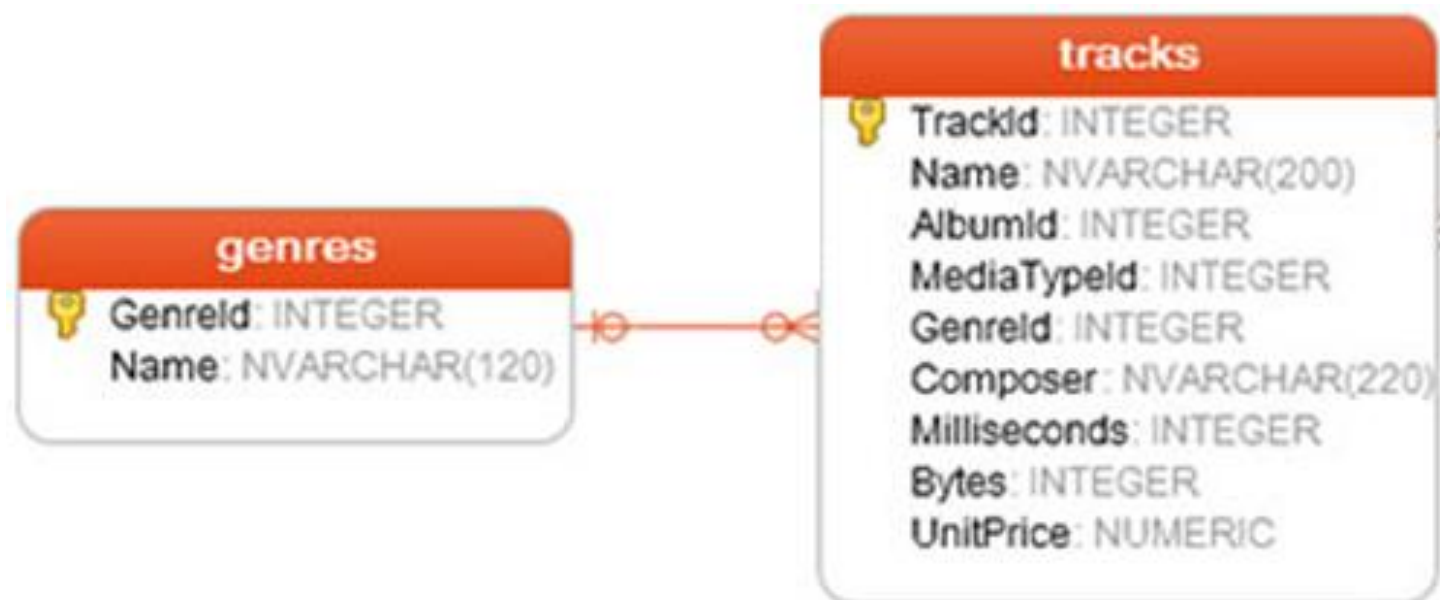
genres.Name	genres.GenreId	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
Metal	1	1		1
	1	2		1
	1	3		1

genres.Name	genres.GenreId	tracks.TrackId	pozostałe kolumny	Tracks.GenreId
Rock	2	4		2

Wprowadzenie do SQL (DQL)

Kolejność wykonywania sekcji

Założmy że chcemy wykonać zapytanie: ile jest utworów które mają kompozytora w ramach gatunków



Wprowadzenie do SQL (DQL)

Kolejność wykonywania sekcji

SELECT

g.Name,

COUNT(*) AS NumberOfTracksPerGenres

FROM

tracks t

INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

WHERE

t. Composer **IS NOT NULL**

GROUP BY

g.Name;

Wprowadzenie do SQL (DQL)

Kolejność wykonywania sekcji

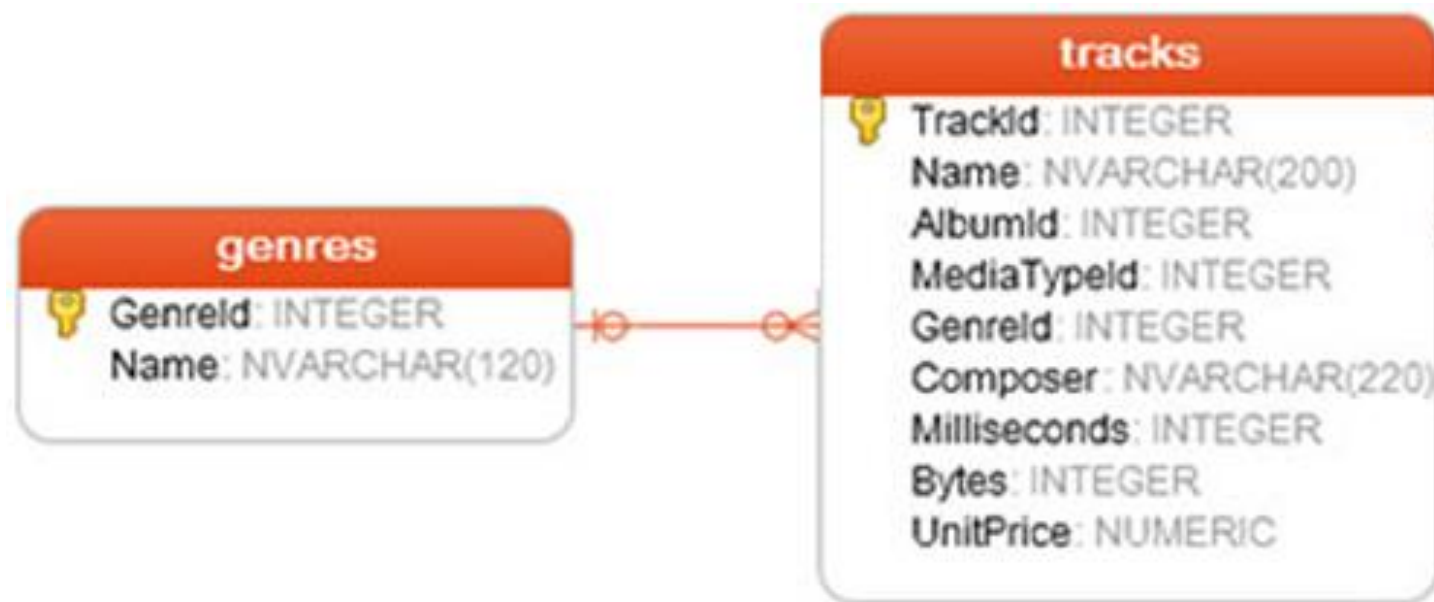
To zapytanie zadziała ponieważ klauzula **WHERE** jest wykonywana **przed** **GROUP BY**. Ogólna kolejność to:

1. **FROM**
2. **WHERE** – dostępne są wszystkie kolumny z **FROM**
3. **GROUP BY** – dostępne są wszystkie kolumny z **FROM**
4. **HAVING** – dostępne są wszystkie kolumny z **FROM**
5. **SELECT** – dostępne są kolumny z **GROUP BY** (jeśli użyte) lub **FROM** (jeśli nie ma **GROUP BY**)
6. **ORDER BY** – tak jak **SELECT**

Wprowadzenie do SQL (DQL)

HAVING – filtr na grupy

Założmy że chcemy wykonać zapytanie: ile jest utworów w ramach gatunków, wyświetlić tylko te grupy w których jest mniej niż 10 utworów



Wprowadzenie do SQL (DQL)

HAVING – filtr na grupy

Na pewno potrzebujemy **GROUP BY** bo potrzebujemy zgrupowania rekordów. Niestety nie zadziała **WHERE** ponieważ warunki w tej klauzuli są aplikowane przed grupowaniem.

Potrzebna nowego słowa kluczowego **HAVING** – filtrowanie po grupowaniu

HAVING wymaga żeby zapytanie SQL zawierało GROUP BY

Wprowadzenie do SQL (DQL)

HAVING – filtr na grupy

SELECT

g.Name,

COUNT(*) AS NumberOfTracksPerGenres

FROM

tracks t

INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

GROUP BY

g.Name

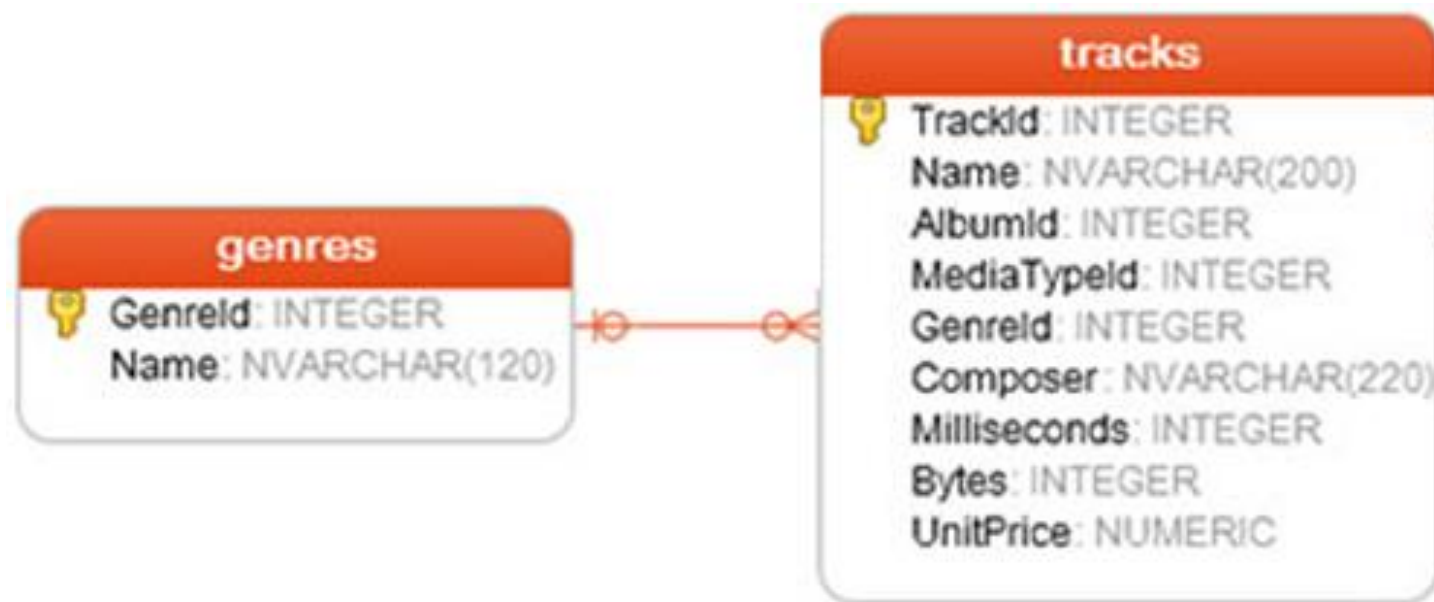
HAVING

COUNT(*) > 10;

Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

Założmy że chcemy wykonać zapytanie: ile jest utworów z niepustymi kompozytorami jest w ramach gatunków, wyświetlić tylko te grupy w których jest mniej niż 10 utworów



Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

SELECT

g.Name,
COUNT(*) AS NumberOfTracksPerGenres

FROM

tracks t
INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

WHERE

t.Composer IS NOT NULL

GROUP BY

g.Name

HAVING

COUNT(*) > 10;

Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

FROM

tracks t

INNER JOIN genres g **ON** tracks.GenreId = genres.GenreId

To złączenie powoduje zwrócenie wyniku:

genres.GenreId	genres.Name	tracks.Composer	pozostałe kolumny	Tracks.GenreId
1	Metal	NULL		1
1	Metal	Osoba A		1
1	Metal	Osoba B		1
2	Rock	Osoba C		2
....

Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

WHERE

t.Composer IS NOT NULL

To warunek usuwa wiersz:

genres.GenreId	genres.Name	tracks.Composer	pozostałe kolumny	Tracks.GenreId
1	Metal	NULL		1
1	Metal	Osoba A		1
1	Metal	Osoba B		1
2	Rock	Osoba C		2
....

Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

Dodanie **GROUP BY genres.Name** powoduje utworzenie w pamięci:

genres.Name
Metal
Rock

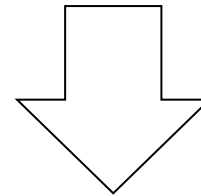
genres.GenreId	tracks.Composer	pozostałe kolumny	Tracks.GenreId
1	Osoba A		1
1	Osoba B		1

genres.GenreId	tracks.Composer	pozostałe kolumny	Tracks.GenreId
2	Osoba C		2
....

Wprowadzenie do SQL (DQL)

HAVING – kolejność jeszcze raz

HAVING jest aplikowane tu



genres.Name
Metal
Rock

genres.GenreId	tracks.Composer	pozostałe kolumny	Tracks.GenreId
1	Osoba A		1
1	Osoba B		1

genres.GenreId	tracks.Composer	pozostałe kolumny	Tracks.GenreId
2	Osoba C		2
....

Wprowadzenie do SQL

Algebra relacji

Działanie SQL'a jest podparte modelem matematycznym tzw. *algebrą relacji*. Algebra relacji definiuje jakie operacje można wykonać na **zbiorze krotek** (wierszy). Do dyspozycji są:

Operator algebry	Słowo kluczowe SQL	Notka
Projekcja: π	SELECT	Słowo kluczowe SELECT myli się z operatorem Selekcji
Selekcja: σ	WHERE	Nazwa selekcja myli się ze słowem kluczowym SQL: SELECT
Złączenie: \bowtie	(INNER) JOIN	

Wprowadzenie do SQL

Algebra relacji – dla ciekawskich

Zapytania SQL w matematyce przedstawia się w formie drzewa z użyciem operatorów algebry relacji.

SELECT

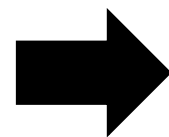
Nazwa

FROM

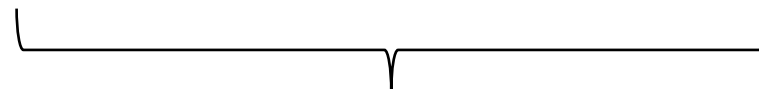
artysta

WHERE

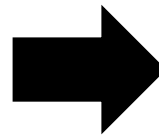
Wiek > 18



$\pi_{Nazwa}(\sigma_{Wiek > 18}(artysta))$



tzw. postać liniowa



π_{Nazwa}



$\sigma_{Wiek > 18}$



artysta

Wprowadzenie do SQL

To wszystko jest trudne

Zgadza się.

Na domiar złego SQL'a nie przerabia **dokładnie** się w szkołach/ na uczelniach. Umiejętność pisania SQL'a jest zazwyczaj „nabyta sama w sobie”.

Dlatego programiści często **ignorują** zapytania SQL'owe i korzystają z bibliotek które automatyzują pisanie zapytań.

Te biblioteki to tzw. **ORM'y**
(ang. Object-Relational Mapping)
(pl. Odwzorowanie obiektowo relacyjne)

DDL/DML

SQL służy nie tylko odpytywaniu, ale też umożliwia:

1. Tworzenie obiektów bazodanowych
2. Zmianę obiektów bazodanowych
3. Wstawianie danych
4. Aktualizację danych
5. Usuwanie danych

Punkty 1 i 2 wchodzą w skład Języka Definicji Danych (DDL)

Punkty 3,4,5 wchodzą w skład Języka Manipulacji Danymi (DML)

Operacje w DML/DDDL nazywamy **Komendami (ang. Command)**

DDL

Tworzenie obiektów

Ogólna zasada języka SQL-DDL brzmi:

Otwórz dokumentację bazy danych i stosuj się do zaleceń

Do wyboru mamy wiele obiektów, ale składnia tworzenia/modyfikowania jest ustandaryzowana **tylko dla tabel**

DDL

Tworzenie obiektów

Na ogół za tworzenie obiektów w SQL służy wyrażenie:

CREATE [obiekt]

Obiekty które na ogół mamy do dyspozycji to:

- Tabele (słowo TABLE)
- Widoki (słowo VIEW)
- Indeksy (słowo INDEX)
- Bazy danych (słowo DATABASE)
- Schemat (słowo SCHEMA)

DDL

Tworzenie tabel

Abstrakcyjna składnia komendy tworzącej tabelę danych:

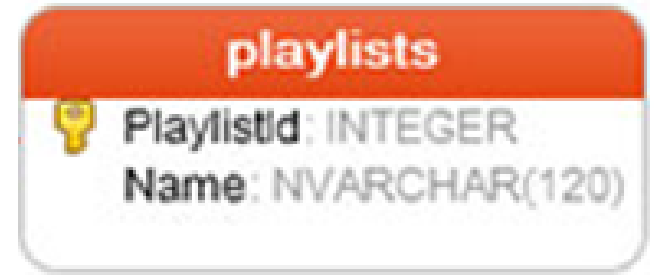
```
CREATE TABLE [nazwa tabeli]!(  
    [nazwa kolumny]! [typ]! [ograniczenia],  
    [nazwa kolumny]! [typ]! [ograniczenia],  
    ...  
    [nazwa kolumny]! [typ]! [ograniczenia],  
    [ograniczenia w innej składni]  
);
```

! – oznacza treść którą trzeba podać

DDL

Tworzenie tabel – przykład 1 – bez ograniczeń

```
CREATE TABLE playlists(  
    PlaylistId INTEGER,  
    Name NVARCHAR(120)  
);
```



DDL

Tworzenie tabel – przykład 2 – bez ograniczeń

```
CREATE TABLE tracks(  
    TrackId INTEGER,  
    Name NVARCHAR(200),  
    AlbumId INTEGER,  
    MediaTypeId INTEGER,  
    GenreId INTEGER,  
    Composer NVARCHAR(220),  
    Milliseconds INTEGER,  
    Bytes INTEGER,  
    UnitPrice NUMERIC  
);
```



DDL

Tworzenie tabel - ograniczenia

Najciekawsza z perspektywy projektanta bazy danych jest możliwość definiowania ograniczeń na danych. Służą do tego następujące słowa kluczowe:

- **PRIMARY KEY** – oznacza kolumnę jako klucz główny
- **UNIQUE** – oznacza kolumnę jako unikatową
- **NOT NULL** – oznacza kolumnę jako nieprzyjmującą wartości puste
- **NULL** – oznacza kolumnę jako przyjmującą wartości puste
- **FOREIGN KEY** lub **REFERENCES** – oznacza kolumnę jako przyjmującą wartości z innej tabeli z kolumny oznaczonej jako **PRIMARY KEY** lub **UNIQUE**
- **DEFAULT** – oznacza domyślną wartość

DDL

Tworzenie tabel - ograniczenia

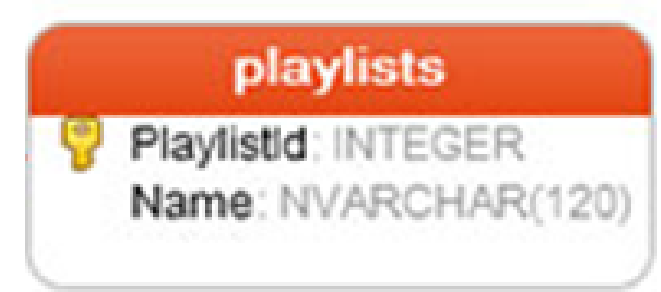
Niektóre z ograniczeń można nazwać. To przydaje się potem do manipulowania nimi np. usuwania:

- **PRIMARY KEY**
- **FOREIGN KEY** lub **REFERENCES**
- Więzy **CHECK** – nie myśleć o nich

DDL

Tworzenie tabel – pełny przykład 1 – wersja 1

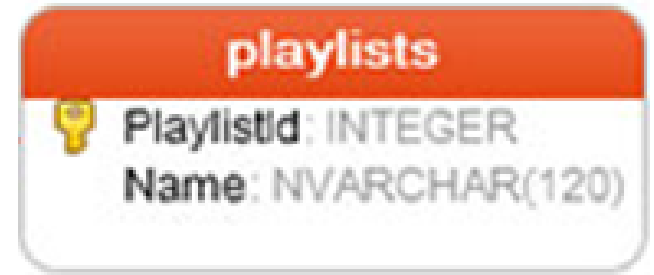
```
CREATE TABLE playlists(  
    PlaylistId INTEGER PRIMARY KEY,  
    Name NVARCHAR(120) NOT NULL  
  
);
```



DDL

Tworzenie tabel – pełny przykład 1 – wersja 2

```
CREATE TABLE playlists(  
    PlaylistId INTEGER,  
    Name NVARCHAR(120) NOT NULL,  
  
    PRIMARY KEY (PlaylistId)  
  
);
```



DDL

Tworzenie tabel – pełny przykład 2 – wersja 1

```
CREATE TABLE tracks(  
    TrackId INTEGER PRIMARY KEY,  
    Name NVARCHAR(200) NOT NULL,  
    AlbumId INTEGER NOT NULL REFERENCES albums(AlbumId),  
    MediaTypeId INTEGER NOT NULL REFERENCES media_types(MediaTypeId),  
    GenreId INTEGER NOT NULL REFERENCES genres(GenreId),  
    Composer NVARCHAR(220) NULL,  
    Milliseconds INTEGER NOT NULL,  
    Bytes INTEGER NOT NULL,  
    UnitPrice NUMERIC NOT NULL  
);
```


DDL

Tworzenie tabel – pełny przykład 2 – wersja 2

```
CREATE TABLE tracks(  
    TrackId INTEGER,  
    Name NVARCHAR(200) NOT NULL,  
  
    AlbumId INTEGER NOT NULL,  
  
    MediaTypeId INTEGER NOT NULL,  
  
    GenreId INTEGER NOT NULL,  
  
    Composer NVARCHAR(220) NULL,  
  
    Milliseconds INTEGER NOT NULL,  
  
    Bytes INTEGER NOT NULL,  
  
    UnitPrice NUMERIC NOT NULL,  
  
    PRIMARY KEY (TrackId),  
  
    FOREIGN KEY (AlbumId) REFERENCES albums(AlbumId),  
  
    FOREIGN KEY (MediaTypeId) REFERENCES media_types(MediaTypeId),  
  
    FOREIGN KEY (GenreId) REFERENCES genres(GenreId)  
  
);
```

DDL

Zmiana obiektów

Czasem zachodzi potrzeba zmiany istniejącej tabeli (lub obiektu).

Na ogół w bazach danych służy do tego komenda:

ALTER [obiekt]

Niestety ta komenda nie wchodzi w skład standardu SQL (chyba)

DDL

Zmiana obiektów

W przypadku zmian tabel trzeba zachować ciąg przyczynowo-skutkowy. Wyobrażamy sobie polecenie zmiany tabeli jako dwa powiązane wyrażenia:

ALTER TABLE (zmień tabelę

1. ALTER COLUMN (zmień kolumnę)
2. DROP COLUMN (usuń kolumnę)

DDL

Zmiana tabel

Przykład 1: zmienia rozmiar/typ kolumny Title

ALTER TABLE albums

ALTER COLUMN Title **NVARCHAR(500);**

Przykład 2: wprowadza ograniczenia NULL do kolumny Title

ALTER TABLE albums

ALTER COLUMN Title **NVARCHAR(500) NULL;**

DDL

Zmiana tabel

Przykład 3: usuwa kolumnę Title

```
ALTER TABLE albums  
DROP COLUMN Title;
```

Przykład 4: dodaje nazwane ograniczenie

```
ALTER TABLE albums  
ADD CONSTRAINT [nazwa] [ograniczenie];
```

Przykład 5: usuwa dowolne ograniczenie

```
ALTER TABLE albums  
DROP CONSTRAINT [nazwa];
```

DDL

Usunięcie obiektu/tabeli

Szczególnym rodzajem modyfikacji jest usunięcie (porzucenie ang. DROP) obiektu. Składania komendy to:

DROP [obiekt] [nazwa obiektu];

Przykład: Usuwa tabelę `albums`

DROP TABLE albums;

DML

Wstawienie danych

Do wstawienia danych służy komenda: **INSERT INTO**

Do dyspozycji jest kilka wariantów. Najpopularniejszym jest wariant gdy programista podaje wszystkie kolumny jawnie.

Przykład wstawiania nowego gatunku:

```
INSERT INTO genres(GenreId, Name)
```

```
VALUES(-1, 'Vaporwave');
```

DML

Wstawienie danych – więcej wyrafinowania

W komendzie INSERT INTO można wykorzystać SQL przy wyliczaniu wartości.

Przykład wstawiania nowego gatunku z kolejną liczbą porządkową:

```
INSERT INTO genres(GenreId, Name)  
VALUES((SELECT MAX(GenreId) + 1 FROM genres), 'Rapcore');
```


DML

Wstawienie danych – czynności administracyjne

W komendzie **INSERT INTO** można wykorzystać **SQL** przy definiowaniu zbioru wejściowego (zamiast **VALUES**). Często wykorzystywane przy przerzucaniu tymczasowo danych.

Przykład wstawiania nowego gatunku z kolejną liczbą porządkową:

```
INSERT INTO genres(GenreId, Name)  
SELECT TrackId * -1, Name FROM tracks  
WHERE TrackId BETWEEN 50 AND 55;
```

Ta komenda na górze nie ma sensu



DML

Wstawienie danych – wyjątek 1

Kolumny oznaczone jako:

- **NULL**
- Z więzami **DEFAULT**

Można pominąć przy wypisywaniu ich w klauzuli **INSERT INTO**.

Musimy się wtedy liczyć że zostaną uzupełnionymi wartościami pustymi/domyślnymi. Załóżmy że nazwa gatunku byłaby oznaczona jako NULL. Komendę INSERT INTO moglibyśmy zapisać:

INSERT INTO genres(GenreId) **VALUES**(-2);

Co spowodowałoby dodanie wiersza: (-2, NULL)

DML

Wstawienie danych – wyjątek 2

Jeśli nie chcemy pominąć żadnej kolumny to możemy nie wypisywać listu kolumn. Czyli zamiast pisanie:

```
INSERT INTO genres(GenreId, Name)
```

```
VALUES(-1, 'Vaporwave');
```

Możemy napisać:

```
INSERT INTO genres
```

```
VALUES(-1, 'Vaporwave');
```

DML

Aktualizacja danych

Do aktualizowania danych służy komenda: **UPDATE**

Do dyspozycji są tylko dwie opcje:

- Z **WHERE** – modyfikacja rekordów spełniających kryteria
- Bez **WHERE** – modyfikacja wszystkich rekordów

Składnia:

UPDATE [tabela]

SET [kolumna] = [wartość], ..., [kolumna] = [wartość]

WHERE [warunek]

DML

Aktualizacja danych – opcja 1

Przykład: aktualizacja gatunku o identyfikatorze -2

Składnia:

UPDATE genres

SET Name = ,Industrial'

WHERE GenreId = -2;

DML

Aktualizacja danych – opcja 2

Przykład: aktualizacja wszystkich gatunków – wszystkie nazwy zmienia na duże litery

Składnia:

UPDATE genres

SET Name = UPPER(Name);

DML

Usunięcie danych

Do usuwania danych służy komenda: **DELETE**

Do dyspozycji są tylko dwie opcje:

- Z **WHERE** – usunięcie rekordów spełniających kryteria
- Bez **WHERE** – usunięcie wszystkich rekordów

Składnia:

DELETE FROM [tabela]

WHERE [warunek]

DML

Usunięcie danych – przykłady

Przykład 1: usunięcie gatunku o identyfikatorze -2

DELETE FROM genres **WHERE** GenreId = -2;

Przykład 2: usunięcie danych z tabeli genres

DELETE FROM genres;



Nie wygłupiać się z tym

Transakcje

Każde pojedyncze zapytanie / komenda są **transakcją**, czyli operacją która jest:

- **A** – Atomowa, czyli niepodzielna
- **C** – Spójna, czyli gwarantuje spełnienie ograniczeń
- **I** – Izolowana, czyli inne transakcje wykonywane w tym samym czasie nie wpływają na wynik rozważanej transakcji
- **D** – Trwała, czyli efekt transakcji jest zachowany nawet po awarii

Transakcje

Wszystkie 4 cechy transakcji tworzą skrót:

A – Atomicity (pl. Atomowa)

C – Consistency (pl. Spójna)

I – Isolation (pl. Izolowana)

D – Durability (pl. Trwała)

Transakcje

Transakcję można utworzyć w SQL'u jawnie za pomocą wyrażenia **BEGIN TRANSACTION** które rozpoczyna transakcję oraz wyrażień **COMMIT** do zatwierdzenia efektów lub **ROLLBACK** do odrzucenia efektów transakcji

BEGIN TRANSACTION

INSERT INTO genres **VALUES**(-1, ,Happy Hardcore');

INSERT INTO genres **VALUES**(-2, ,House');

COMMIT

Administracja

Access

- System zarządzania bazą danych
- Silnik Access'a to **`Jet DB`**
 - Duża część funkcjonalności nie jest dostępna z poziomu Access'a
- Jednoplikowa baza danych
- Jednoużytkownikowa baza danych
- Cztery główne obiekty z którymi pracujemy:
 - Tabele
 - Kwerendy
 - Formularze
 - Raporty

Administracja

Access - funkcjonalność



Plik:

- Zapis/wczytanie bazy
- Kompaktowanie bazy
- Szyfrowanie bazy

Narzędzia główne:

- Widok danych

Tworzenie:

- Kreator tabel
- Kreator kwerend
- Kreator formularzy
- Kreator raportów

Dane zewnętrzne:

- Import z plików
- Łączenie z innymi bazami
- Łączenie z Sharepointem

Narzędzia:

- Kompaktowanie
- Niestandardowy kod (makra)
- Widok związków (Relacji)
- Analiza wydajności

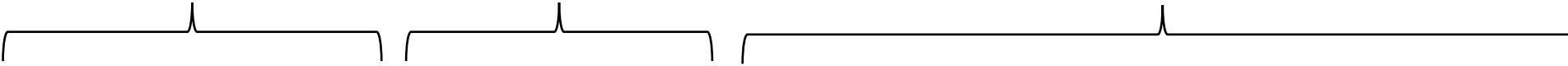
Administracja

Access – tworzenie tabel

Nazwy kolumn

Rodzaje danych

Niewidoczny opis



Utwory	Nazwa pola	Typ danych	Opis (opcjonalny)
	Identyfikator	Autonumerowanie	
	Tytuł	Krótki tekst	
	Cena	Waluta	
	Kompozytor	Krótki tekst	

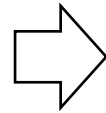
Zakładka: Tworzenie -> Tabela

Administracja

Access – tworzenie tabel

Po wybraniu kolumny pokazują się właściwości pola w której możemy wprowadzić ograniczenia (w zależności od typu danych)

Utwory	
Nazwa pola	Typ danych
Identyfikator	Autonumerowanie
Tytuł	Krótki tekst
Cena	Waluta
Kompozytor	Krótki tekst



Ogólne	Odnosnik
Rozmiar pola	100
Format	
Maska wprowadzania	
Tytuł	Tytuł
Wartość domyślna	
Reguła poprawności	
Tekst reguły spr. poprawności	
Wymagane	Tak
Zerowa dł. dozwolona	Tak
Indeksowane	Tak (Bez duplikatów)
Kompresja Unicode	Tak
Tryb IME	Bez kontrolki
Tryb zdania edytora IME	Brak
Wyrównanie tekstu	Ogólne

Zakładka: Tworzenie -> Tabela

Administracja

Access – tworzenie tabel

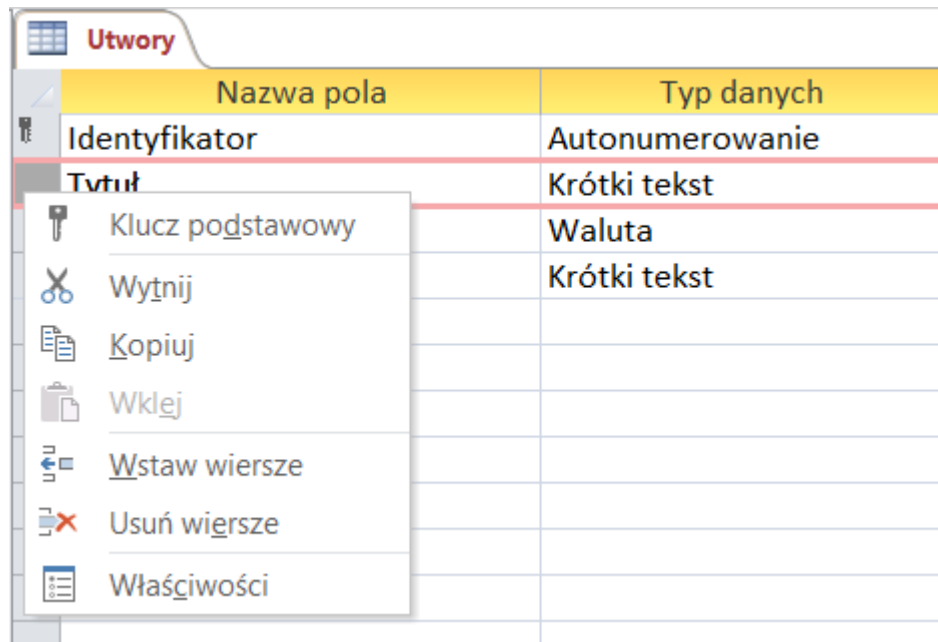
Kilka istotnych ograniczeń:

- Wymagane
- Wartość domyślna
- Reguła poprawności
- Format
- Maska wprowadzania
- Indeksowane – umożliwia wymuszenie unikatowych wartości

Administracja

Access – tworzenie tabel

Wybór klucza podstawowego odbywa się przez wybranie kolumn i kliknięcie **PPM->Klucz Podstawowy**



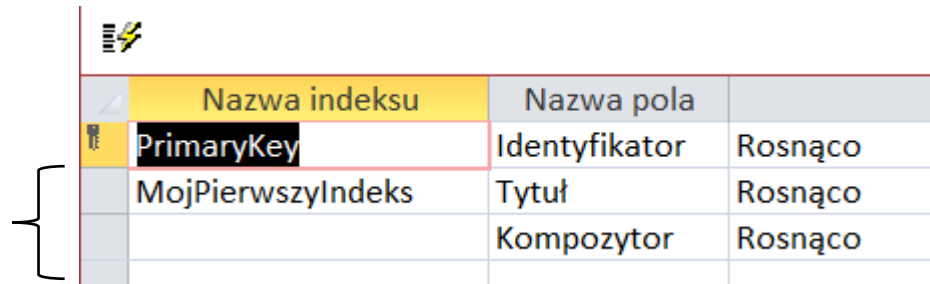
Zakładka: Tworzenie -> Tabela

Administracja

Access – indeksowanie

Umożliwia wybranie kolumn przeznaczonych do szybkiego dostępu oraz do narzucenia unikatowości w ramach kilku kolumn.

Indeks na kolumnach
Tytuł i Kompozytor



Nazwa indeksu	Nazwa pola	Rosnąco
PrimaryKey	Identyfikator	Rosnąco
MojPierwszyIndeks	Tytuł	Rosnąco
	Kompozytor	Rosnąco

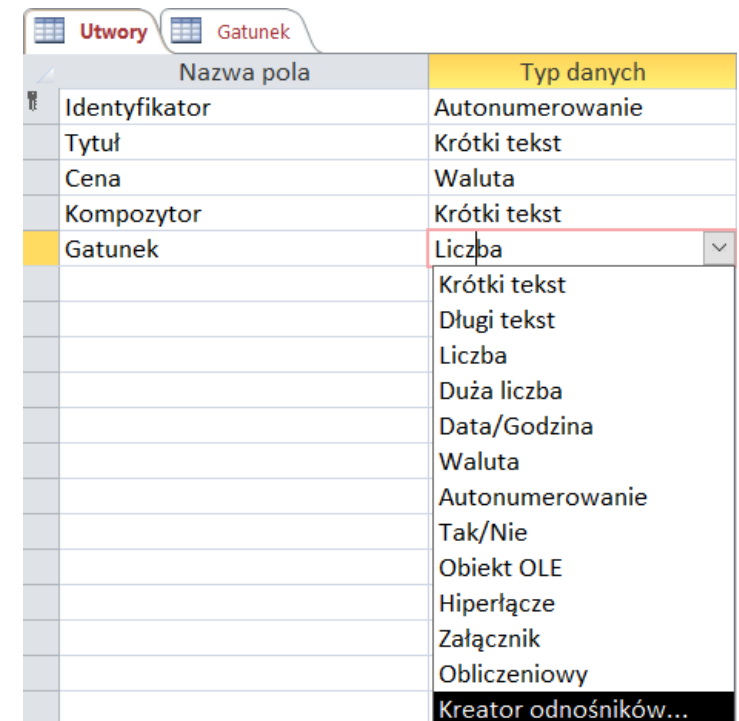
Administracja

Access – klucze obce

Założmy że mamy tabelę Gatunek(Id, Nazwa).

Zmodyfikujemy tabelę Utwory żeby posiadała wskazanie na gatunek:

- Nazywamy kolumnę Gatunek
- Jak typ wybieramy Kreator Odnośników



Nazwa pola	Typ danych
Identyfikator	Autonumerowanie
Tytuł	Krótki tekst
Cena	Waluta
Kompozytor	Krótki tekst
Gatunek	Liczba
	Krótki tekst
	Długi tekst
	Liczba
	Duża liczba
	Data/Godzina
	Waluta
	Autonumerowanie
	Tak/Nie
	Obiekt OLE
	Hiperłącze
	Załącznik
	Obliczeniowy
	Kreator odnośników...

Zakładka: Tworzenie -> Tabela

Administracja

Access – klucze obce

Podczas 5 etapowego kreatora zostaniemy poproszeni o:

1. Wybranie czy pole ma być pobrane z tabeli oraz z jakiej tabeli
2. Wybrać odpowiedni pole
3. Ustalić sortowanie
4. Wybrać czy chcemy pokazywać kolumnę klucza głównego (zalecane)
5. Ustalić czy chcemy włączyć by baza danych pilnowała **integralności danych (więcej zaraz)**

Administracja

Access – klucze obce

Uzyskujemy czytelne dla użytkownika powiązanie pomiędzy dwoma tabelami:

Gatunek		Utwory				
Identyfikator	Tytuł	Cena utworu	Kompozytor	Gatunek	Kliknij, aby dodać	
1	Kto tam	12,30 zł	Mój kompozytor	MetaLove		
4	Kto tam	0,00 zł	Ktoś	Rock		
* (Nowy)		0,00 zł				

Zakładka: Narzędzia główne -> widok arkusza danych

Administracja

Integralność danych

Taki przypadek:

Utwory wskazują na gatunki

Usuwa gatunek **Pop**

Co się dzieje z wskazaniem na gatunek w tabeli utworów?

Administracja

Integralność danych

W SQL'u przy tworzeniu ograniczenia FOREIGN KEY / REFERENCES możemy doprecyzować co ma się dziać gdy wiersze na które wskazuje tabela są usuwane. Znamy trzy techniki:

- Zabronić usunięcia rekordów (ang. RESTRICT)

FOREIGN KEY x REFERENCES gatunek(Id) ON DELETE RESTRICT

- Ustawić wartości kolumny na NULL (ang. SET NULL)

FOREIGN KEY x REFERENCES gatunek(Id) ON DELETE SET NULL

- Usunąć rekordy powiązane... kaskadowo (ang. CASCADE)


FOREIGN KEY x REFERENCES gatunek(Id) ON DELETE CASCADE

Administracja

Access – klucze obce

Zaznaczenie opcji włącz integralność danych umożliwia doprecyzowanie które zachowanie chcemy osiągnąć. Sprawdźmy jak działa kaskadowe usuwanie.

Kreator odnośników



Jaka etykieta ma być przypisana do obiektu: pole odnośnika?

Nazwa

Czy chcesz włączyć integralność danych między tymi tabelami?

☒ Włącz integralność danych

☐ Usuwanie kaskadowe

☒ Ograniczenie usuwania

Czy chcesz przechowywać wiele wartości dla tego odnośnika?

☐ Zezwalaj na wiele wartości

To już wszystkie odpowiedzi, których kreator potrzebował do utworzenia kontrolki: pole odnośnika.

[Bez tytułu]

Anuluj < Wstecz Dalej > Zakończ

Administracja

Access – klucze obce

Przed:

Gatunek		Utwory			
Identyfikator	Tytuł	Cena utworu	Kompozytor	Gatunek	Kliknij, aby dodać
1	Kto tam	12,30 zł	Mój kompozytor	MetaLove	
4	Kto tam	0,00 zł	Ktoś	Pop	
6	Ttoo	12,00 zł	K	MetaLove	
*	(Nowy)	0,00 zł			

Gatunek		Utwory	
Identyfikator	Nazwa	Kliknij, aby dodać	
+	1 Rock		
+	2 MetaLove		
+	3 Pop		

Po:

Gatunek		Utwory			
Identyfikator	Tytuł	Cena utworu	Kompozytor	Gatunek	Kliknij, aby dodać
1	Kto tam	12,30 zł	Mój kompozytor	MetaLove	
#Usunięty	#Usunięty	#Usunięty	#Usunięty		
6	Ttoo	12,00 zł	K	MetaLove	
*	(Nowy)	0,00 zł			

Gatunek		Utwory	
Identyfikator	Nazwa	Kliknij, aby dodać	
+	1 Rock		
+	2 MetaLove		
*	(Nowy)		

Administracja

Access - Kompaktowanie

Na poprzednim slajdzie widać że baza danych nie odzyskuje pamięci



Rekord o id = 4 stworzył sztuczny pusty wiersz. Jeśli nas to bodzi, to możemy wykonać proces kompaktowania bazy danych, czyli defragmentacji + sprawdzenia czy ograniczenia nałożone na dane są spełnione

Żeby to zrobić: Narzędzia -> Kompaktuj i napraw bazę danych

Gatunek		Utwory				
Identyfikator	Tytuł	Cena utworu	Kompozytor	Gatunek	Kliknij, aby dodać	
1	Kto tam	12,30 zł	Mój kompozytor	MetaLove		
6	Ttoo	12,00 zł	K	MetaLove		

Administracja

Kompaktowanie

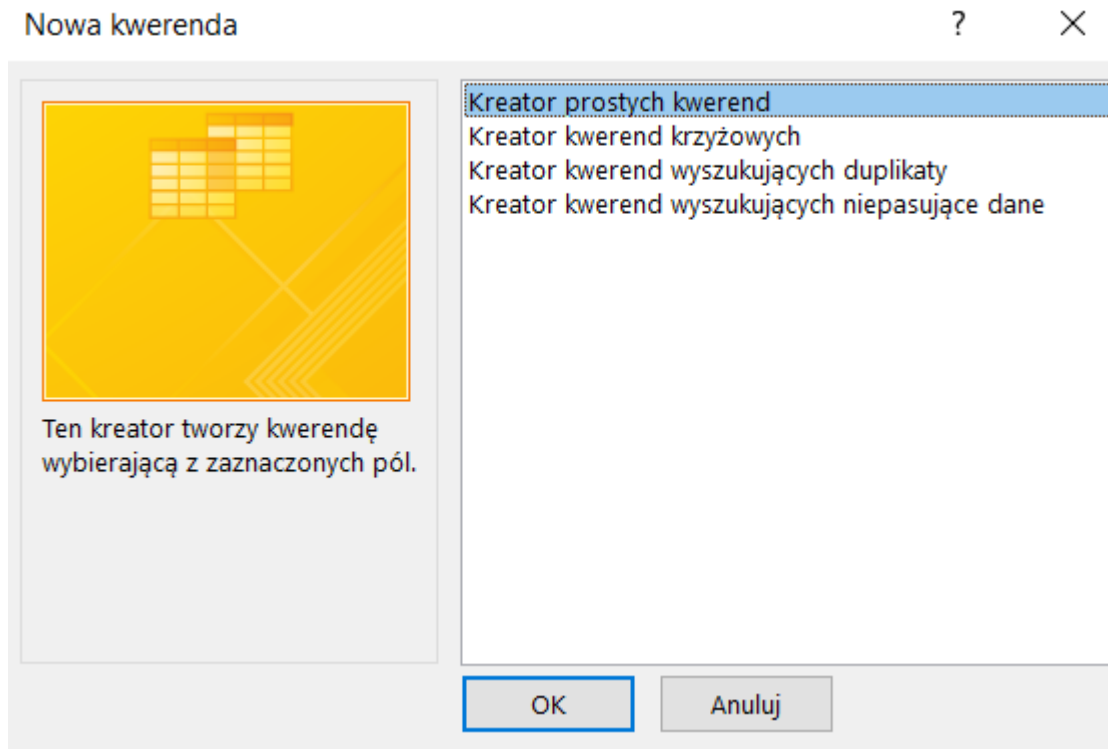
Kompaktowanie jest podobne do **naprawiania** tabel w innej bazie danych: **MySQL**

W MySQL/MariaDB naprawianie tabel odbywa się z użyciem polecenia: **REPAIR TABLE [nazwa tabeli]**

Administracja

Access - Zapytania

Access umożliwia tworzenie zapytań za pomocą kreatora:



Zakładka: Tworzenie -> Kreator Kwerend

Administracja

Access - Zapytania

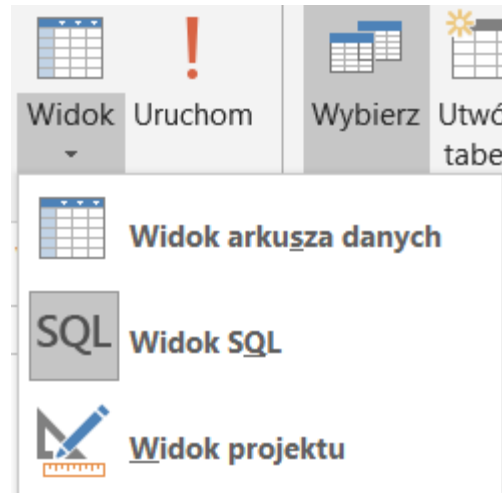
Mamy cztery szablony do dyspozycji:

1. Kwerendy proste
2. Kwerendy krzyżowe (PIVOT)
3. Kwerendy wyszukiujące duplikaty (HAVING)
4. Kwerendy wyszukiujące niepasujące dane (SEMI/LEFT JOIN)

Administracja

Access - Zapytania

Po przejściu przez kreator możemy przejść do widoku SQL wybierając na wstążce opcje Widok.



Administracja

Access - Zapytania

Po przejściu przez kreator możemy przejść do widoku SQL wybierając na wstążce opcje Widok i obejrzeć co zaproponował Kreator. Przykład kwerendy szukającej duplikaty:

SELECT

First(Utwory.[Tytuł]) **AS** [Tytuł Pole],
Count(Utwory.[Tytuł]) **AS** LiczbaPowtórzeń

FROM Utwory

GROUP BY Utwory.[Tytuł]

HAVING (((Count(Utwory.[Tytuł]))>1));

Administracja

Access - Zapytania

Po przejściu przez kreator możemy przejść do widoku SQL wybierając na wstążce opcje Widok i obejrzeć co zaproponował Kreator. Przykład kwerendy szukającej duplikaty:

SELECT

First(Utwory.[Tytuł]) **AS** [Tytuł Pole],
Count(Utwory.[Tytuł]) **AS** LiczbaPowtórzeń

FROM Utwory

GROUP BY Utwory.[Tytuł]

HAVING (((Count(Utwory.[Tytuł]))>1));

Administracja

Access - Formularze i Raporty

Wykorzystywane do prezentacji danych i jako forma dialogu pomiędzy bazą danych, a użytkownikiem:

- Formularze na ogół służą do wprowadzania danych
- Raporty na ogół prezentują podsumowanie danych

Administracja

Nadawanie uprawnień

Standard SQL'a opisuje w jaki sposób nadawać uprawnienia użytkownikom Bazy Danych. Ogólna składnia wygląda następująco:

GRANT [uprawnienia]

ON [obiekt bazy danych]

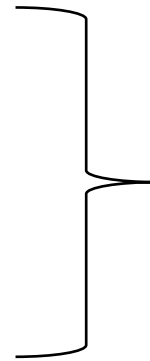
TO [użytkownik]

Co ciekawe w standard SQL'a nie precyzuje jak tworzyć użytkowników

Administracja

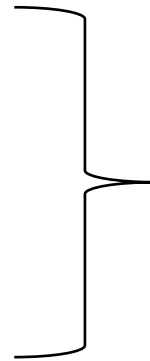
Nadawanie uprawnień - przykłady

GRANT SELECT, UPDATE
ON mojaBaza.wspanialaTabela
TO test@localhost
WITH GRANT OPTION;



Nadaje uprawnienia **SELECT** i **UPDATE** użytkownikowi test. Dodatkowo użytkownik może nadawać te uprawnienia innym osobom

GRANT ALL PRIVILEGES
ON mojaBaza.wspanialaTabela
TO test@localhost;



Nadaje wszystkie uprawnienia użytkownikowi test

Administracja

Odbieranie uprawnień

Standard SQL'a opisuje w jaki sposób odbierać uprawnienia użytkownikom Bazy Danych. Ogólna składnia wygląda następująco:

REVOKE [uprawnienia]

ON [obiekt bazy danych]

FROM [użytkownik]

Administracja

XAMPP

Kombajn kilku aplikacji/komponentów które połączone umożliwiają tworzenie oraz zarządzanie aplikacjami internetowymi. Do komponentów które należą do tego mixu należą:

- Apache HTTP Server (server HTTP + silnik PHP)
- MySQL / MariaDB (serwer BD)
- Apache Tomcat (serwer Java)
- Filezilla (serwer FTP)

Administracja

PHPMyAdmin + Tipy na egzamin

PHPMyAdmin to potwór działający na lokalnie na porcie 80 (zazwyczaj):

<http://localhost/phpmyadmin/index.php>

Jeśli port jest inny to wpisujemy go za nazwą hosta (localhost) np. phpMyAdmin działający na porcie 8111:

<http://localhost:8111/phpmyadmin/index.php>

Potwór umożliwia zarządzanie Bazą Danych MySQL / MariaDB

Administracja

MySQL: Pro-tip

W rzeczywistym życiu:

Istnieje lepsze środowisko dla MySQL

MySQL Workbench

<https://www.mysql.com/products/workbench/>

Administracja

PHPMyAdmin + Tipy na egzamin

The screenshot displays the PHPMyAdmin interface for a MySQL server at 127.0.0.1. The top navigation bar includes tabs for 'Bazy danych', 'SQL', 'Status', 'Konta użytkowników', 'Eksport', 'Import', 'Ustawienia', 'Replikacja', 'Zmienne', 'Kodowania znaków', and 'Więcej'. The left sidebar shows a tree view of databases, including 'information_schema', 'mojabd', 'wspaniala', 'Indeksy', 'Kolumny', 'Nowy', 'Druga', 'PK', 'Test', 'Trzecia', 'mysql', 'performance_schema', 'phpmyadmin', and 'test'.

The main content area is divided into three panels:

- Ustawienia ogólne**: Shows the 'Sortowanie połączenie z serwerem' dropdown set to 'utf8mb4_unicode_ci'.
- Ustawienia wyglądu**: Shows the 'Język - Language' dropdown set to 'Polski - Polish', the 'Motyw' set to 'pmahomme', and the 'Rozmiar czcionki' set to '82%'. A link 'Więcej ustawień' is also present.
- Serwer bazy danych**: Lists server details:
 - Serwer: 127.0.0.1 via TCP/IP
 - Typ serwera: MariaDB
 - Połączenie z serwerem: SSL is not being used
 - Wersja serwera: 10.1.38-MariaDB - mariadb.org binary distribution
 - Wersja protokołu: 10
 - Użytkownik: root@localhost
 - Kodowanie znaków serwera: UTF-8 Unicode (utf8)
- Serwer WWW**: Lists web server details:
 - Apache/2.4.38 (Win64) OpenSSL/1.1.1b PHP/7.3.3
 - Wersja klienta bazy danych: libmysql - mysqlnd 5.0.12-dev - 20150407 - \$Id: 7cc7cc96e675f6d72e5cf0f267f48e167c2abb23 \$
 - Rozszerzenie PHP: mysqli, curl, mbstring
 - Wersja PHP: 7.3.3
- phpMyAdmin**: Lists version and documentation links:
 - Informacja o wersji: 4.8.5
 - Dokumentacja
 - Oficjalna strona phpMyAdmina
 - Współpraca
 - Pomoc techniczna
 - Lista zmian
 - Licencja

The bottom panel shows the 'Konsola' (Console) with the following SQL command:

```
>CREATE TABLE `mojabd`.`wspaniala` ( `PK` INT NOT NULL AUTO_INCREMENT , `Test` VARCHAR(100) NOT NULL , `Druga` INT NOT NULL , `Trzecia` INT NOT NULL , PRIMARY KEY (`PK`))...
```


Administracja

PHPMyAdmin + Tipy na egzamin

Z poziomu phpMyAdmin możemy:

1. Nadać uprawnienia użytkownikom
2. Utworzyć bazę danych
3. Utworzyć tabele
4. Wykonywać zapytania

Administracja

PHPMyAdmin + **Tipy na egzamin**

Na egzaminie E14 zazwyczaj użytkownik łączy się z bazą z poziomu języka PHP. Będzie to wymagało nazwy użytkownika oraz podania hasła.

Domyślnie skonfigurowana instancja Apache XAMPP + MySQL ma użytkownika **root bez ustawionego hasła**

!!!

Administracja

PHPMyAdmin + **Tipy** na egzamin

Ogólny schemat tworzenia aplikacji na egzamin E14:

1. Upewniamy się jaki użytkownik jest wymagany w bazie danych
 - Być może **root** wystarczy
2. Piszemy kod php który otwiera połączenie z Bazą Danych
3. Piszemy kod php który wykonuje zapytanie z użyciem połączenia z pkt 2)
4. Piszemy kod php który zamyka połączenie z Bazą Danych

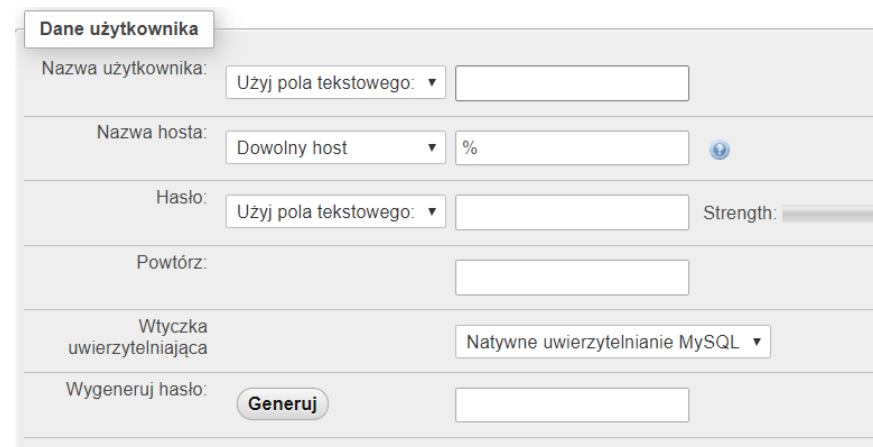
Administracja

PHPMyAdmin – użytkownicy – etap 1

W PHPMyAdmin użytkownika możemy utworzyć na jeden z dwóch sposobów:

1. Wybrać Serwer -> Konta użytkowników
2. Wybrać Bazę Danych -> Uprawnienia

Po czym wybieramy opcję „Dodaj konto użytkownika”



The screenshot shows the 'Dane użytkownika' (User Data) form in PHPMyAdmin. It contains the following fields and options:

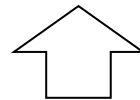
- Nazwa użytkownika:** A dropdown menu with 'Użyj pola tekstowego:' and an adjacent text input field.
- Nazwa hosta:** A dropdown menu with 'Dowolny host' and an adjacent text input field containing '%'. A small blue icon is visible to the right.
- Hasło:** A dropdown menu with 'Użyj pola tekstowego:' and an adjacent text input field. To the right is a 'Strength:' indicator with a progress bar.
- Powtórz:** A text input field for repeating the password.
- Wtyczka uwierzytelniająca:** A dropdown menu with 'Natywne uwierzytelnianie MySQL' selected.
- Wygeneruj hasło:** A button labeled 'Generuj' and an adjacent text input field.

Administracja

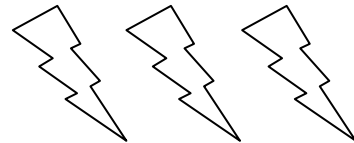
PHPMyAdmin – użytkownicy – etap 1

Na potrzeby testów warto ustawić:

- Hosta jako **localhost**
- Wybrać opcję: Globalne uprawnienia – zaznacz wszystko



W poważnych zastosowaniach to na ogół bardzo zła praktyka



Administracja

PHPMyAdmin – kod aplikacji – etap 2

Na początku naszego skryptu PHP warto utworzyć 4 zmienne w które wpisujemy: host, nazwę użytkownika, hasło oraz nazwę bazy danych

```
<?php
    $server = "localhost";
    $uzytkownik = "root";
    $haslo = "";
    $bazadanych = "mojabd";
```

Administracja

PHPMyAdmin – kod aplikacji – etap 2

W drugiej kolejności otwieramy połączenie, podając cztery parametry z naszych zmiennych. Odbywa się to poprzez użycie słowa kluczowego **new**.

Powołujemy obiekt klasy **mysqli**:

```
$polaczenie = new mysqli($server, $uzytkownik, $haslo, $bazadanych);
```

W pierwszym parametrze podajemy host, w drugim użytkownika, w trzecim hasło, a w czwartym nazwę bazy danych

Administracja

PHPMyAdmin – kod aplikacji – etap 2

Warto dodać fragment kodu który poinformuje o ewentualnym problemie z połączeniem. Służy do tego właściwość: **connect_error**

```
if ($polaczenie->connect_error) {  
    die("Błąd połączenia: " . $polaczenie->connect_error);  
}
```


Administracja

PHPMyAdmin – kod aplikacji – etap 3

Pobranie danych odbywa się przez metodę połączenia **query(SQL)**.

Funkcja ta zwraca obiekt wyniku który jest kolekcją wierszy. Na samym początku wynik wskazuje na pusty wiersz...

```
$sql = "SELECT PierwszaKolumna, DrugaKolumna AS Alias2 FROM mojaTabela";
$wynik = $polaczenie->query($sql);

if ($wynik->num_rows > 0) {
    printf("<table border=\"1\">");
    printf("<tr><th>PK</th><th>Druga</th></tr>");
    while($row = $wynik->fetch_assoc()) {
        printf( "<tr><td>%s</td><td>%s</td></tr></br>", $row["PierwszaKolumna"], $row["Alias2"]);
    }
    printf("</table>");
} else {
    echo "Brak wyników :(";
}
```

Administracja

PHPMyAdmin – kod aplikacji – etap 3

... ale wywołania metody **fetch_assoc()** powodują przesunięcie się na kolejny wiersz wynikowy oraz zwrócenie pojedynczego wiersza (tutaj \$row). \$row jest tablicą asocjacyjną gdzie kluczami są aliasy z **SELECT**.

```
$sql = "SELECT PierwszaKolumna, DrugaKolumna AS Alias2 FROM mojaTabela";
$wynik = $polaczenie->query($sql);

if ($wynik->num_rows > 0) {
    printf("<table border=\"1\">");
    printf("<tr><th>PK</th><th>Druga</th></tr>");
    while($row = $wynik->fetch_assoc()) {
        printf( "<tr><td>%s</td><td>%s</td></tr></br>", $row["PierwszaKolumna"], $row["Alias2"]);
    }
    printf("</table>");
} else {
    echo "Brak wyników :(";
}
```

Administracja

PHPMyAdmin – kod aplikacji – etap 3

\$row jest tablicą asocjacyjną gdzie kluczami są aliasy z **SELECT**.

```
$sql = "SELECT PierwszaKolumna, DrugaKolumna AS Alias2 FROM mojaTabela";
```

A dostęp przez:

```
$row["PierwszaKolumna"]
```

Oraz gdy korzystamy z aliasów:

```
$row["Alias2"]
```

Administracja

PHPMyAdmin – kod aplikacji – etap 4

Na koniec zamykamy połączenie:

```
$polaczenie->close();
```

To jest bardzo ważne!

Baza danych ma skończoną liczbę połączeń. Przy 150 minutowym teście można je wyczerpać i Baza Danych będzie odmawiać połączeń.

Pozostałe informacje

Preprocessor jako ograniczenie

PHP z nazwy jest pre-procesorem czyli przed-przetwarzającym hipertekst.

Wszystko co robi PHP jest wykonywane po stronie serwera... i tylko tam. To co widzimy na przeglądarce jest wynikiem działania kodu PHP.

Pozostałe informacje

Preprocessor jako ograniczenie

Wynika z tego takie „śmieszne” ograniczenie, że nie można napisać kodu w PHP który wykonuje jakąś formę **dialogu** z użytkownikiem.

Przynajmniej nie łatwo.

Ogólnie rzecz biorąc PHP stosowany jest do tzw. statycznych stron internetowych.

Pozostałe informacje

Wstrzykiwanie SQL'a

Bardzo złym pomysłem jest pozwalanie użytkownikowi na manipulowanie zapytaniami.

Na pierwszy rzut oka takie rozwiązanie jest ok:

```
$sql = "SELECT PierwszaKolumna FROM mojaTabela WHERE Imie = '" . $imie . "'";
```

Ale jeśli użytkownik wpisze w \$imie ciąg:

cokolwiek' OR 1=1; --

To wyciekną wszystkie dane z tabeli **mojaTabela**

Pozostałe informacje

Wstrzykiwanie SQL'a

Ten atak nazywa się:

wstrzykiwaniem SQL'a
(ang. SQL Injection)

Przeciwdziała się mu za pomocą przygotowanych zapytań (ang. Prepared Statement) – wszędzie tam gdzie użytkownik podaje wartość korzystamy z **przygotowanych zapytań**.

```
$stmt = $conn->prepare("SELECT PierwszaKolumna FROM mojaTabela WHERE Imie = ?");  
$stmt->bind_param("s", $imie);
```


That's all for now