WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Bazy danych
Wykład w ramach przedmiotu „Informatyka” (EE-DI)

Dawid Warchoł

Literatura

Książki polecane przez prowadzącego:

1. Vinicius M. Grippa, Sergey Kuzmichev, „MySQL. Jak zaprojektować i wdrożyć wydajną bazę danych. Wydanie II”, Helion, 2022;
2. Hernandez Michael J., „Projektowanie baz danych dla każdego. Przewodnik krok po kroku”, Helion, 2021.

Szczególnie polecana jest pierwsza pozycja, która została wykorzystana przy tworzeniu tego wykładu.

Czym jest baza danych?

- Jest to zbiór powiązanych ze sobą informacji, które są przechowywane i zarządzane w sposób umożliwiający ich łatwe wyszukiwanie, modyfikowanie i organizowanie.

- Bazę danych można również zdefiniować jako abstrakcyjne, informatyczne odzwierciedlenie fragmentu rzeczywistości.

Historia baz danych
Część I – przed erą komputerów (do lat 50. włącznie):

Ręczne systemy kartotekowe:

- Dane były przechowywane w formie kartotek papierowych.
- Informacje były organizowane w segregatorach, kartotekach lub rejestrach (np. księgi rachunkowe, rejestry pacjentów w szpitalach).
- Problemem był długi czas dostępu do danych oraz trudność w ich aktualizacji.

Historia baz danych
Część II – lata 60. – 70.

Bazy hierarchiczne i sieciowe:

- Wraz z rozwojem komputerów powstały pierwsze elektroniczne systemy baz danych, np. IBM Information Management System (1966).
- Struktura hierarchiczna: dane były przechowywane w postaci drzewa (podobnie jak system plików i folderów w systemach operacyjnych).
- Model sieciowy (np. IDS – Integrated Data Store, CODASYL) umożliwiał bardziej skomplikowane powiązania między danymi.
- Problemem tych baz była trudność w modyfikacji struktury i skomplikowane zapytania.

Historia baz danych
Część II – lata 70. – 80.

Relacyjne bazy danych i język SQL:

- 1970 – Edgar F. Codd z IBM publikuje przełomowy artykuł o relacyjnym modelu danych (RDBMS).
- Dane są przechowywane w tabelach, które wykazują wzajemne relacje, co upraszcza ich organizację i przetwarzanie.
- 1973/74 – Powstaje SQL (Structured Query Language) – język do zarządzania bazą danych.
- 1986/87 – Język SQL staje się oficjalnym standardowym językiem relacyjnych baz danych (standardy ANSI i ISO).
- Powstają pierwsze komercyjne systemy RDBMS, np. MySQL, PostgreSQL, Microsoft SQL Server, Oracle, IBM DB2.

Historia baz danych
Część II – lata 90. – 00.

Epoka baz obiektowych i NoSQL:

- Obiektowe bazy danych (np. OODMS) łączą dane z mechanizmami programowania obiektowego.

- Pojawiają się systemy NoSQL – bazy danych nierelacyjne:
  - bazy dokumentowe (np. MongoDB),
  - kolumnowe (np. Cassandra),
  - grafowe (np. Neo4j).

- Bazy NoSQL pozwalają na szybsze przeszukiwanie i modyfikowanie danych nieustrukturyzowanych o dużej skali.

Historia baz danych
Część II – lata 2010 - obecnie

Nowoczesne bazy danych:

- Rozproszone i chmurowe bazy danych (np. Google BigQuery, Amazon DynamoDB).
- NewSQL – nowoczesne systemy RDBMS (np. CockroachDB, Google Spanner).
- Bazy blockchain – np. Ethereum, Hyperledger – zapewniają niezmienność danych.
- AI i automatyzacja, np. autonomiczna baza danych Oracle Autonomous DB.
  - Jest to baza, która dzięki wsparciu AI jest: samozarządzająca się, samonaprawiająca się i samooptymalizująca się (działa w chmurze).

Podsumowanie – rodzaje baz danych

- **Relacyjne** – do dziś najbardziej popularne i powszechnie używane. **Będą głównym tematem naszych wykładów.**
- Nierelacyjne (NoSQL) – w ostatnich latach rośnie ich popularność, jednak są nadal wyraźnie mniej popularne od baz danych relacyjnych.
- Obiektowe – rzadko używane. Stosuje się je w niektórych zastosowaniach przemysłowych, np. projektowanie CAD/CAM (w przemyśle lotniczym, motoryzacyjnym i budowlanym).
- Hierarchiczne – całkowicie przestarzałe (mogą być w użyciu w bardzo starych systemach przemysłowych, które nie zostały przez wiele lat zaktualizowane).

Zastosowania baz danych

1. Biznes i handel:

- **Systemy zarządzania relacjami z klientami (CRM):**
  - Przechowywanie danych klientów, historii zakupów, preferencji.
  - Ułatwianie personalizacji ofert i lepszej obsługi klienta.
- **Systemy zarządzania zasobami przedsiębiorstwa (ERP):**
  - Integracja procesów finansowych, logistycznych, produkcyjnych.
  - Usprawnienie zarządzania magazynem, łańcuchem dostaw.
- **E-commerce (handel internetowy):**
  - Przechowywanie informacji o produktach, zamówieniach, płatnościach.
  - Umożliwianie efektywnego zarządzania sklepem internetowym.

Zastosowania baz danych

2. Bankowość i finanse:

- Obsługa kont bankowych.
- Systemy giełdowe.

3. Administracja publiczna:

- Rejestry państwowe.
- Systemy e-administracji.

4. Opieka zdrowotna:

- Elektroniczna dokumentacja medyczna (EDM).
- Systemy zarządzania szpitalami.

Zastosowania baz danych

5. Media społecznościowe i aplikacje internetowe:

- Przechowywanie profili użytkowników, treści.
- Aplikacje mobilne.

6. Nauka i badania:

- Przechowywanie danych naukowych.
- Systemy informacji geograficznej (GIS).

Zastosowania baz danych - podsumowanie

- Bazy danych są stosowane we wszystkich stronach/serwisach/aplikacjach internetowych, w których można założyć konto i zalogować się.

- Zaleca się również stosować bazy danych wtedy, gdy mamy do czynienia z dużą ilością danych i potrzebujemy mieć możliwość szybkiego wyszukiwania/modyfikowania/dodawania/usuwania danych spełniających określone kryteria.

Jak nie projektować relacyjnej bazy danych

- Jest to baza studentów i ich ocen końcowych z przedmiotów.
- Jakie problemy występują z tą bazą danych?
- Jan Nowak i Tomasz Kowalski mają po dwie oceny z tych samych przedmiotów.
- Co może być przyczyną?

Imie	Nazwisko	Przedmiot	Ocena
Jan	Nowak	Informatyka	5.0
Jan	Nowak	Matematyka	2.0
John	Doe	Matematyka	4.0
Tomasz	Kowalski	Informatyka	3.0
Tomasz	Kowalski	Informatyka	2.0
Jan	Nowak	Teoria obwodów	3.5
Jan	Nowak	Teoria obwodów	4.5
John	Doe	Teoria obwodów	4.5

Jak nie projektować relacyjnej bazy danych

- Dodaliśmy kolumnę *Numer_indeksu*, dzięki czemu możemy stwierdzić, że jest dwóch studentów o imieniu i nazwisku Jan Nowak. Numer indeksu jest unikalny (unikatowy) dla każdego studenta. Pierwszy problem został więc rozwiązany.
- Widzimy jednak, że Tomasz Kowalski jest tylko jeden. Dlaczego więc ma dwie oceny z tego samego przedmiotu?

Numer_indeksu	Imie	Nazwisko	Przedmiot	Ocena
123456	Jan	Nowak	Informatyka	5.0
123456	Jan	Nowak	Matematyka	2.0
123459	John	Doe	Matematyka	4.0
123463	Tomasz	Kowalski	Informatyka	3.0
123463	Tomasz	Kowalski	Informatyka	2.0
123456	Jan	Nowak	Teoria obwodów	3.5
123457	Jan	Nowak	Teoria obwodów	4.5
123459	John	Doe	Teoria obwodów	4.0

Jak nie projektować relacyjnej bazy danych

- Dodaliśmy kolumny *Rok* i *Numer*, dzięki czemu możemy stwierdzić, że Tomasz Kowalski nie zaliczył Informatyki w 2024 r., więc zaliczał ją ponownie w 2025 r.
- Wystarczyłoby do tego dodanie kolumny *Rok*, jednak przy okazji dodaliśmy *Semestr*, aby móc stwierdzić, na którym semestrze odbywa się dany przedmiot.

Numer_indeksu	Imie	Nazwisko	Przedmiot	Rok	Semestr	Ocena
123456	Jan	Nowak	Informatyka	2025	1	5.0
123456	Jan	Nowak	Matematyka	2025	2	2.0
123459	John	Doe	Matematyka	2025	2	4.0
123463	Tomasz	Kowalski	Informatyka	2025	1	3.0
123463	Tomasz	Kowalski	Informatyka	2024	1	2.0
123456	Jan	Nowak	Teoria obwodów	2025	1	3.5
123457	Jan	Nowak	Teoria obwodów	2025	1	4.5
123459	John	Doe	Teoria obwodów	2025	1	4.0

Jak nie projektować relacyjnej bazy danych

- Teraz tabela wygląda w porządku. Ma jednak sporo powtarzających się informacji.
- Zauważmy, że numery indeksu są wprost powiązane z konkretnymi imionami i nazwiskami, które za każdym razem musimy podawać wpisując nową ocenę.
- Gdyby któryś ze studentów zmienił nazwisko, to musielibyśmy zmieniać go w bazie tyle razy, ile ocen ma ten student. Jest to mocno problematyczne. Co z tym zrobić?

Numer_indeksu	Imie	Nazwisko	Przedmiot	Rok	Semestr	Ocena
123456	Jan	Nowak	Informatyka	2025	1	5.0
123456	Jan	Nowak	Matematyka	2025	2	2.0
123459	John	Doe	Matematyka	2025	2	4.0
123463	Tomasz	Kowalski	Informatyka	2025	1	3.0
123463	Tomasz	Kowalski	Informatyka	2024	1	2.0
123456	Jan	Nowak	Teoria obwodów	2025	1	3.5
123457	Jan	Nowak	Teoria obwodów	2025	1	4.5
123459	John	Doe	Teoria obwodów	2025	1	4.0

Jak nie projektować relacyjnej bazy danych

- Aby poradzić sobie z tym problemem utworzyliśmy dwie tabele: *Student* i *Ocena*.
- Tabela *Ocena* przechowuje informacje o ocenach i zawiera jedynie numer indeksu studenta. W tabeli *Student* znajdują się dodatkowe dane studentów.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
123456	Informatyka	2025	1	5.0
123456	Matematyka	2025	2	2.0
123459	Matematyka	2025	2	4.0
123463	Informatyka	2025	1	3.0
123463	Informatyka	2024	1	2.0
123456	Teoria obwodów	2025	1	3.5
123457	Teoria obwodów	2025	1	4.5
123459	Teoria obwodów	2025	1	4.0

Jak nie projektować relacyjnej bazy danych

- Co nam to dało? Po pierwsze, nie musimy powtarzać tych samych danych za każdym razem, kiedy dodajemy nową ocenę studenta.
- Po drugie, jeśli chcielibyśmy zmienić nazwisko studenta, to wystarczy zrobić to tylko raz, a nie przy każdym wpisie dotyczącym oceny.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
123456	Informatyka	2025	1	5.0
123456	Matematyka	2025	2	2.0
123459	Matematyka	2025	2	4.0
123463	Informatyka	2025	1	3.0
123463	Informatyka	2024	1	2.0
123456	Teoria obwodów	2025	1	3.5
123457	Teoria obwodów	2025	1	4.5
123459	Teoria obwodów	2025	1	4.0

Jak nie projektować relacyjnej bazy danych

- Te tabele są już sensowne. Jednak w relacyjnej bazie danych zakłada się, że każda tabela powinna mieć kolumnę z unikalnymi wartościami, stanowiącą identyfikator. Taką kolumnę nazywa się **kluczem głównym** (podstawowym, ang. primary key).
- W tabeli *Student* kluczem głównym może być *Numer_indeksu*. Tabela *Ocena* nie posiada klucza głównego. Dodajmy go więc.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
123456	Informatyka	2025	1	5.0
123456	Matematyka	2025	2	2.0
123459	Matematyka	2025	2	4.0
123463	Informatyka	2025	1	3.0
123463	Informatyka	2024	1	2.0
123456	Teoria obwodów	2025	1	3.5
123457	Teoria obwodów	2025	1	4.5
123459	Teoria obwodów	2025	1	4.0

~~Jak nie projektować relacyjnej bazy danych~~
Jak zaprojektować relacyjną bazę danych

- Jak widać, kolumna *ID_oceny* (klucz główny) ma wartości niepowtarzające się, podobnie jak kolumna *Numer_indeksu* tabeli *Student*.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

~~Jak nie projektować relacyjnej bazy danych~~
Jak zaprojektować relacyjną bazę danych

- Czy kolumna *Numer_indeksu* występująca w tabeli *Ocena* nie mogłaby być kluczem głównym?
- Nie, ponieważ jej wartości nie są unikalne wewnątrz tabeli *Ocena*. Może być wiele ocen przypisanych do jednego studenta.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

~~Jak nie projektować relacyjnej bazy danych~~
Jak zaprojektować relacyjną bazę danych

- Czym więc jest kolumna *Numer_indeksu* w tabeli *Ocena*?
- Jest to tzw. **klucz obcy** (ang. foreign key).
- Klucz obcy jest kolumną, która odnosi się do klucza głównego z innej tabeli.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

~~Jak nie projektować relacyjnej bazy danych~~
Jak zaprojektować relacyjną bazę danych

- Klucz obcy może nazywać się inaczej niż klucz główny, do którego się odnosi.
- Tzn. atrybut *Numer_indeksu* w tabeli *Ocena* mógłby nazywać się np. *Student* albo *ID_studenta*.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

~~Jak nie projektować relacyjnej bazy danych~~
Jak zaprojektować relacyjną bazę danych

- Jest jeszcze jeden drobny problem z nasza baza danych.
- Tabela Ocena ma kolumnę o nazwie Ocena. Czy może tak być?
- Tak, ale nie jest to zalecane, ponieważ zmniejsza czytelność kodu zapytań i może prowadzić do problemów z korzystaniem z bazy.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Ocena
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

Jak zaprojektować relacyjną bazę danych

- Zmieniliśmy nazwę kolumny Ocena na Wartosc_oceny.
- Ostateczna wersja naszej bazy danych wygląda następująco.

Student

Numer_indeksu	Imie	Nazwisko
123456	Jan	Nowak
123459	John	Doe
123463	Tomasz	Kowalski
123457	Jan	Nowak

Ocena

ID_oceny	Numer_indeksu	Przedmiot	Rok	Semestr	Wartosc_oceny
2	123456	Informatyka	2025	1	5.0
3	123456	Matematyka	2025	2	2.0
4	123459	Matematyka	2025	2	4.0
5	123463	Informatyka	2025	1	3.0
1	123463	Informatyka	2024	1	2.0
6	123456	Teoria obwodów	2025	1	3.5
7	123457	Teoria obwodów	2025	1	4.5
8	123459	Teoria obwodów	2025	1	4.0

Diagram ERD

- Strukturę bazy danych można przedstawić w formie diagramu związków encji (ang. entity relationship diagram, ERD), zwanym czasem diagramem bazy danych.
- ERD składa się z tzw. encji, które odpowiadają tabelom bazy.
- Encje składają się z atrybutów, które odpowiadają kolumnom bazy.

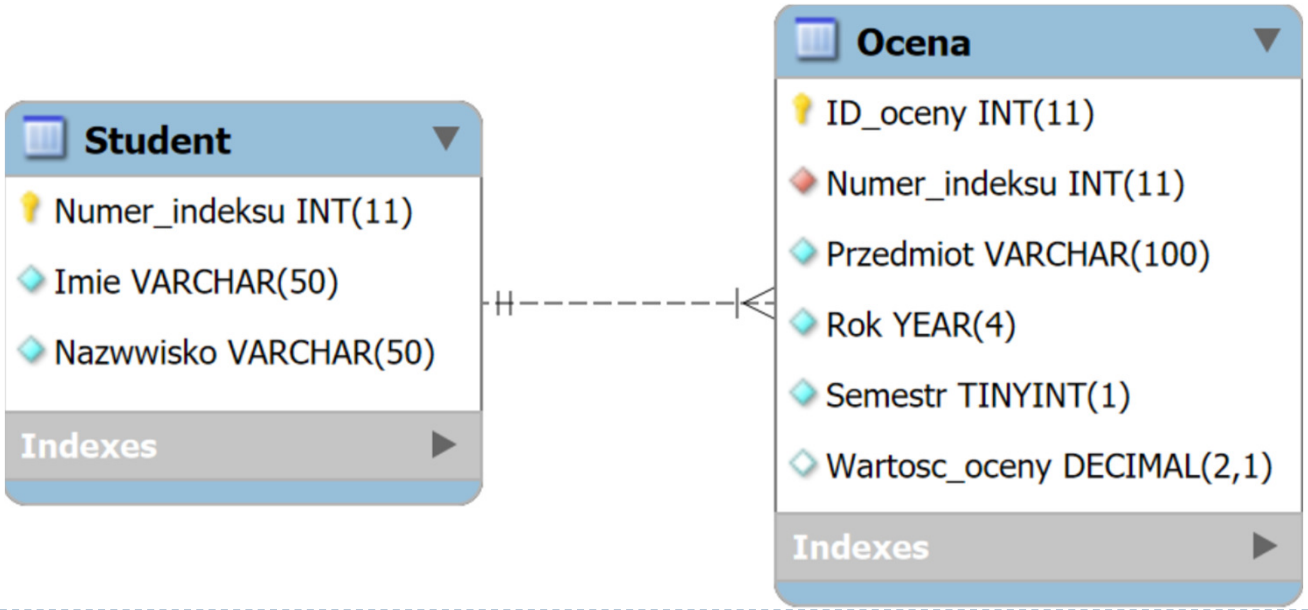


Diagram ERD

- W diagramach utworzonych przy użyciu narzędzia MySQL Workbench:
  - atrybut oznaczony złotym kluczem oznacza klucz główny,
  - atrybut oznaczony czerwonym kwadratem oznacza klucz obcy,
  - atrybut oznaczony wypełnionym kwadratem jest obowiązkowy,
  - atrybut oznaczony pustym kwadratem jest opcjonalny.

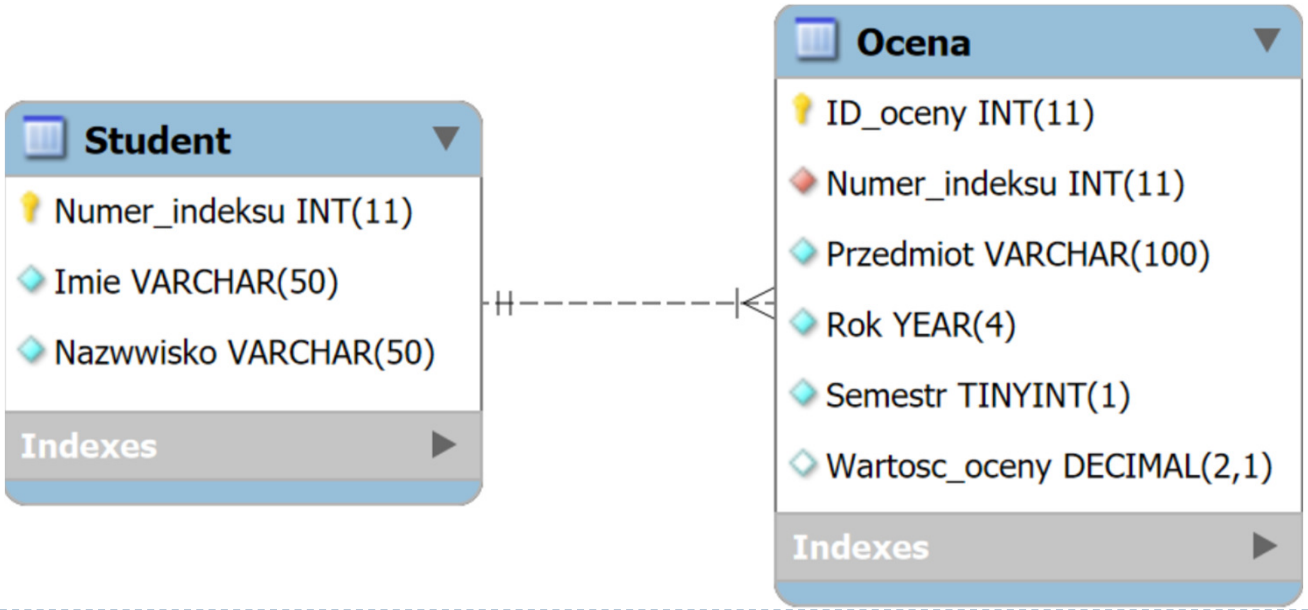


Diagram ERD

- W diagramach utworzonych przy użyciu narzędzia MySQL Workbench:
  - atrybut oznaczony złotym kluczem oznacza klucz główny,
  - atrybut oznaczony czerwonym kluczem oznacza klucz obcy,
  - atrybut oznaczony wypełnionym kwadratem jest obowiązkowy,
  - atrybut oznaczony pustym kwadratem jest opcjonalny.

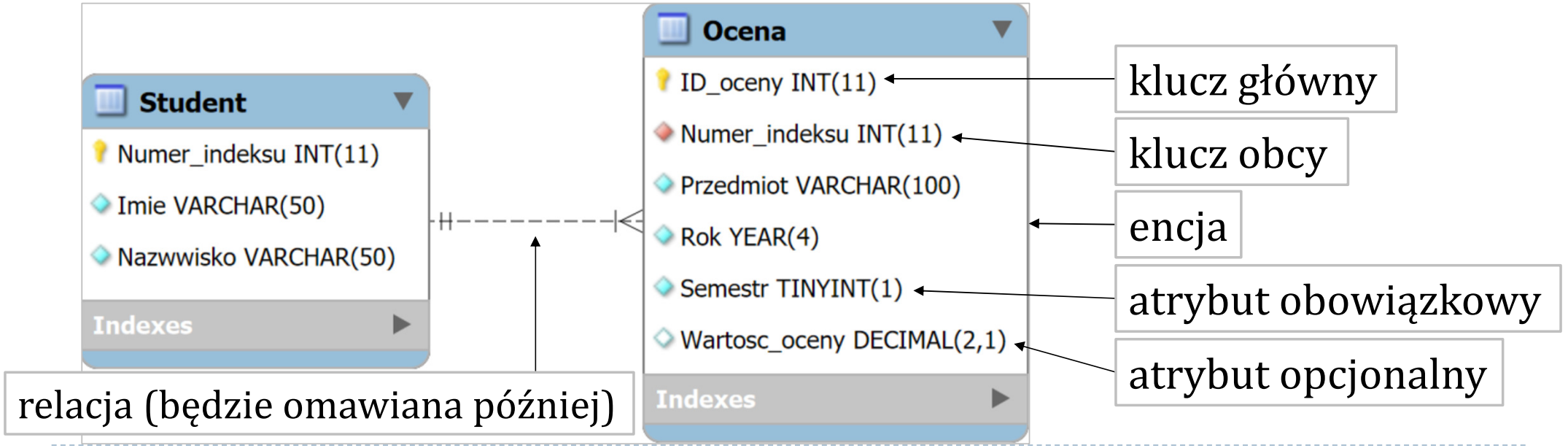


Diagram ERD

- Atrybut obowiązkowy oznacza, że tworząc nowy wpis (wiersz) w bazie, musimy podać jego wartość.
- Atrybut opcjonalny oznacza, że możemy pominąć jego wartość. W bazach danych brak wartości określa się słowem NULL.

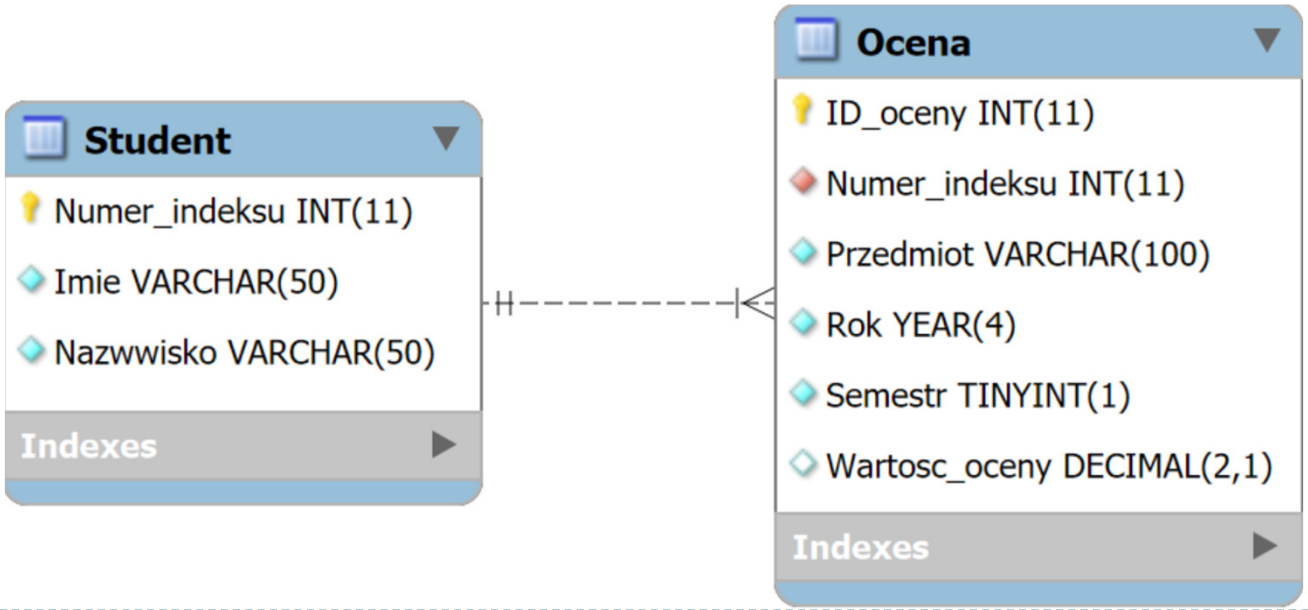


Diagram ERD

- Dlaczego Wartosc_oceny jest atrybutem opcjonalnym?
- Można to skojarzyć z pustą komórką w indeksie/dzienniku. Taka komórka nie zawiera jeszcze konkretnej oceny, ale od razu wiadomo jakiego przedmiotu, roku, semestru i studenta dotyczy.

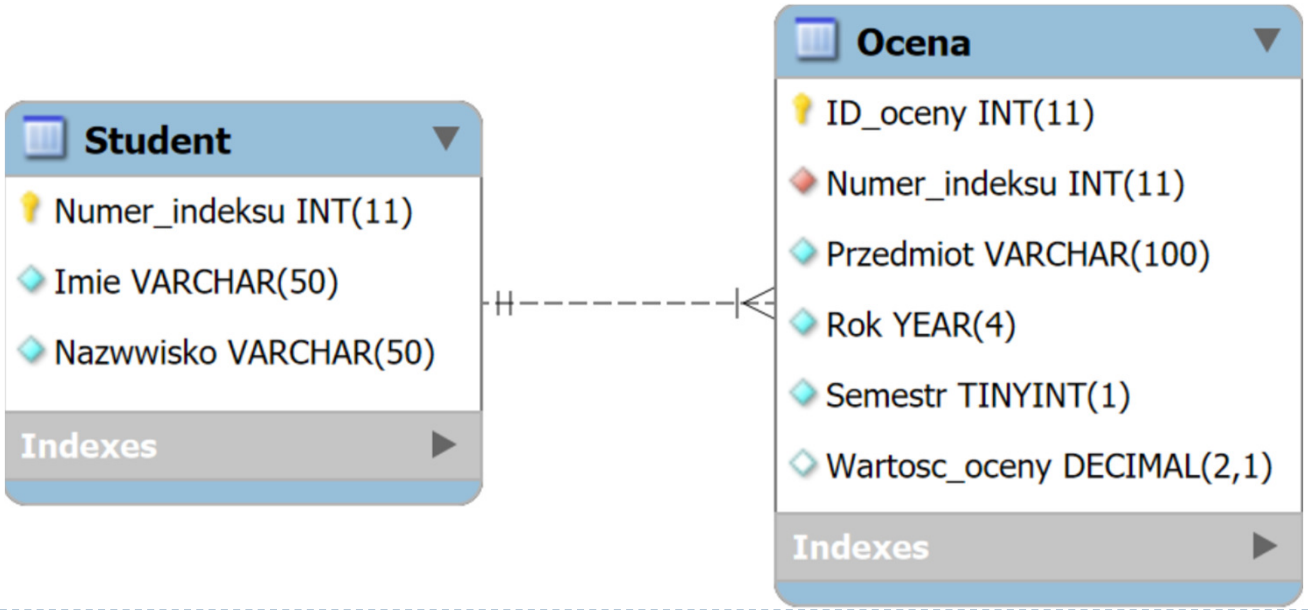


Diagram ERD

- W popularnym programie do obsługi baz danych phpMyAdmin diagramy ERD są tworzone automatycznie po utworzeniu bazy.
- Takie diagramy mają trochę inny wygląd. Jakie są różnice?
  - W nagłówkach encji, obok ich nazw widnieje nazwa bazy danych, w naszym przypadku „indeks_studencki”.
  - Klucze obce nie są oznaczone symbolem klucza. Opcjonalność również nie jest oznaczona
  - Pozostałe atrybuty mają symbole zależne od ich typu (np. ikona kalendarza w typach związanych z datą).

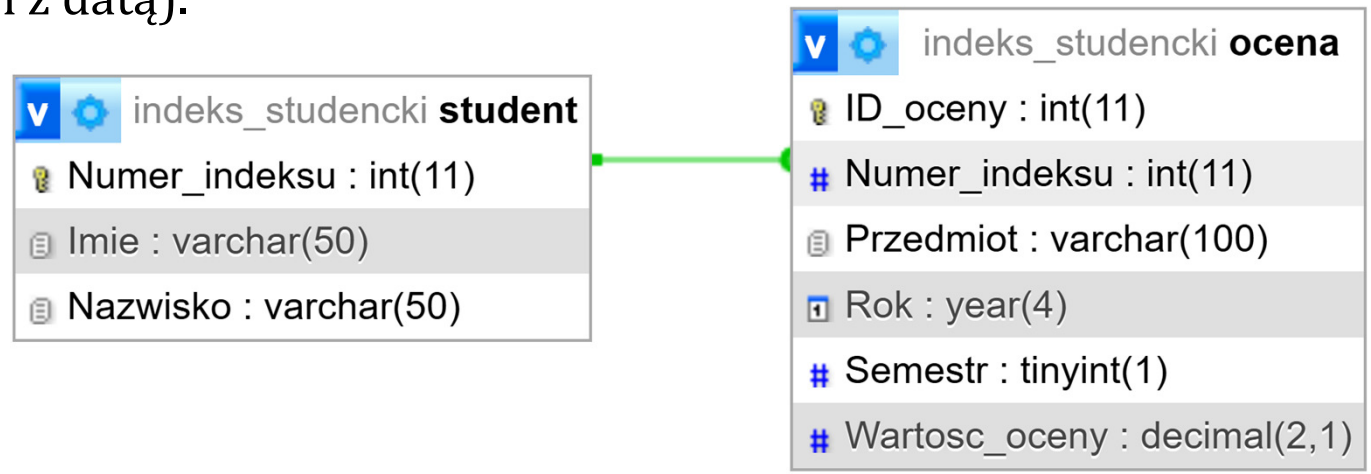
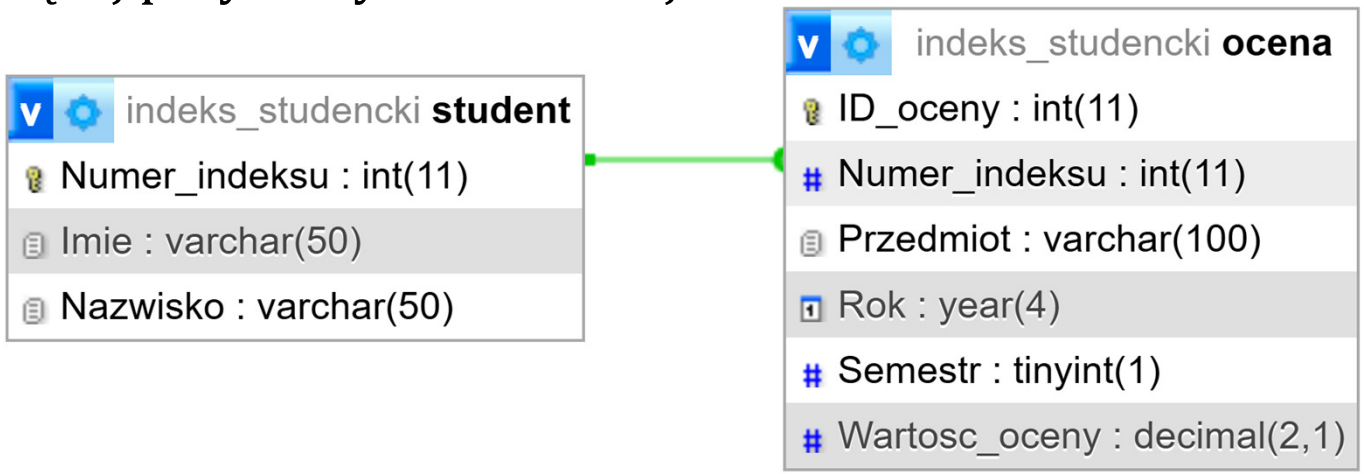


Diagram ERD

- phpMyAdmin jest jednym z najbardziej popularnych programów do zarządzania bazami danych i jest/będzie wykorzystywany na zajęciach laboratoryjnych.
- Z tego powodu scharakteryzowane zostały diagramy ERD automatycznie generowane przez phpMyAdmin.
- Przykłady w dalszej części wykładu będą jednak prezentowane głównie za pomocą diagramów utworzonych w MySQL Workbench, ponieważ są one czytelniejsze i zawierają więcej przydatnych informacji.



Ważniejsze typy danych w bazach MySQL – dane tekstowe

- VARCHAR(*n*) – napis o maksymalnej długości *n* (0 – 65535).
  - Stosuje się go do napisów takich jak: imiona, nazwiska, adresy e-mail, numery telefonów, kody produktów itp.

- TEXT – długi tekst o maksymalnej długości 65535 znaków.
  - Stosuje się je do: opisów, komentarzy, logów, danych o zmiennej i nieprzewidywalnej długości.
  - W pozostałych przypadkach lepiej użyć VARCHAR dla oszczędności miejsca w pamięci i lepszej wydajności zapytań w bazie danych.

Ważniejsze typy danych w bazach MySQL – liczby całkowite

- TINYINT(n) – liczba całkowita 1-bajtowa.
  Zakres liczbowy: [-128 – 128].
- SMALLINT(n) – liczba całkowita 2-bajtowa.
  Zakres liczbowy: [-32 768 – 32 767].
- MEDIUMINT(n) – liczba całkowita 3-bajtowa.
  Zakres liczbowy: [-8 388 608 – 8 388 608].
- INT(n) – liczba całkowita 4-bajtowa.
  Zakres liczbowy: [-2 147 483 648 – 2 147 483 648].
- BIGINT(n) – liczba całkowita 8-bajtowa.
  Zakres liczbowy: $[-2^{63} – 2^{63}]$.

- Parametr n oznacza minimalną długość wyświetlania (z ewentualnymi zerami z przodu).

Ważniejsze typy danych w bazach MySQL – liczby rzeczywiste

- FLOAT(*n*,*d*), DOUBLE(*n*,*d*) – liczba zmiennoprzecinkowa odpowiednio pojedynczej i podwójnej precyzji.
- DECIMAL(*n*,*d*) – liczba stałoprzecinkowa.
- Parametr *n* oznacza całkowitą liczbę cyfr, a *d* liczbę cyfr po przecinku.
- Kiedy stosować FLOAT/DOUBLE, a kiedy DECIMAL? Wyjaśnienie na następnym slajdzie.

Liczby zmiennoprzecinkowe a stałoprzecinkowe

- Czym różnią się liczby zmiennoprzecinkowe od stałoprzecinkowych?
- Oba typy nadają się do przechowywania liczb, które mogą być ułamkami.
- Liczby stałoprzecinkowe (DECIMAL) powinno się używać do reprezentowania kwot, ponieważ są w 100% precyzyjne.
- Przykładowo, w systemach bankowych nie można używać typów zmiennoprzecinkowych do reprezentowania kwot przelewów i stanu konta, ponieważ mają ograniczoną precyzję.

- Dlaczego zatem nie używa się liczb stałoprzecinkowych zawsze?
- Ponieważ zajmują więcej pamięci i są znacznie wolniejsze w przetwarzaniu.
- Liczby zmiennoprzecinkowe są przetwarzane w komputerze przez koprocesor arytmetyczny (FPU – Floating Point Unit), który bardzo przyspiesza obliczenia. Liczby stałoprzecinkowe są przetwarzane przez jednostkę arytmetyczno-logiczną procesora (ALU – Arithmetic Logic Unit), tak samo jak liczby całkowite.

Ważniejsze typy danych w bazach MySQL

Typy związane z datą i czasem:

- DATE – data w formacie YYYY-MM-DD
- DATETIME – data i czas (zależny od strefy czasowej) w formacie YYYY-MM-DD hh:mm:ss
- TIMESTAMP – data i czas (niezależny od strefy czasowej) w formacie YYYY-MM-DD hh:mm:ss
- TIME – czas w formacie hh:mm:ss
- YEAR – rok w formacie YYYY

Y – rok, M – miesiąc, D – dzień, h – godzina, m – minuta, s – sekunda

Stałe (literały) określonych typów

- W MySQL stałe typów tekstowych oraz typów związanych z datą i czasem zapisujemy w apostrofach, np. `'Jan'`, `'Kowalski'`, `'1918-11-11'`.

- Stałe typów liczbowych zapisujemy bez cudzysłowów (i żadnych innych dodatkowych znaków).

- Ułamki zmiennoprzecinkowe oraz stałoprzecinkowe zapisujemy przy użyciu kropki jako separatora dziesiętnego, np. `2.4`, a nie `2,4`.

Specjalne typy danych w bazach MySQL

- ENUM(*val1*, *val2*, *val3*,...), SET(*val1*, *val2*, *val3*,...) – specjalne typy tekstowe. Atrybuty należące do tych typów mogą mogą przyjmować jedynie napisy z ustalonego zbioru.
- W przypadku ENUM atrybut może przyjąć tylko jedną wartość (spośród wartości określonych w nawiasie).

```
status ENUM('oczekuje','wysłane','dostarczone','anulowane')
```

- W przypadku SET atrybut może przyjąć wiele wartości (spośród wartości określonych w nawiasie).

```
uprawnienia SET('pobieranie','dodawanie','usuwanie','edycja')
```

Pełna lista typów danych MySQL wraz ze szczegółowymi opisami znajduje się na stronie: *https://www.w3schools.com/mysql/mysql_datatypes.asp*

Zadanie 1

- W programie phpMyAdmin utworzyć bazę danych „indeks_studencki” zaprezentowaną na poprzednich slajdach.

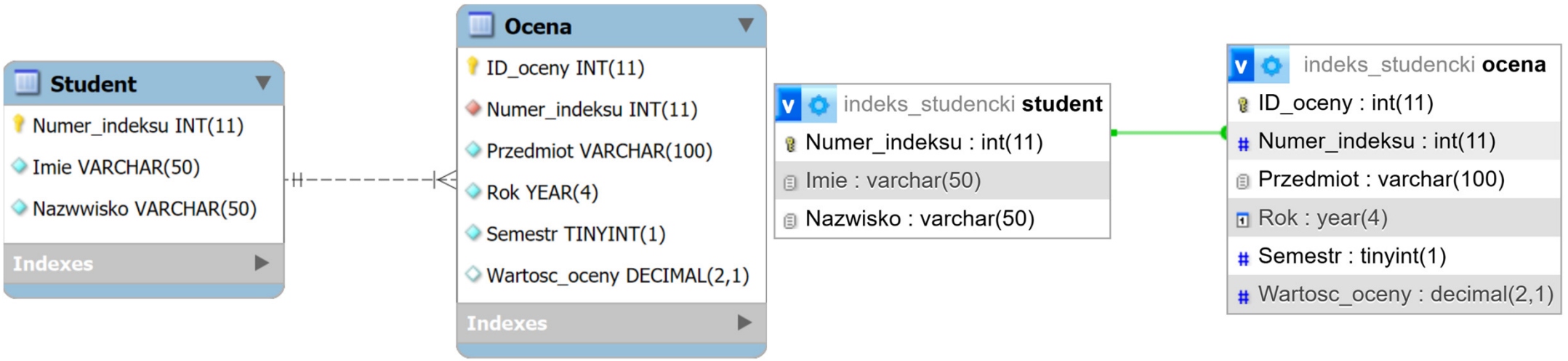
Od czego zacząć?

- Najpierw należy zainstalować program XAMPP (*https://www.apachefriends.org/pl/index.html*).
- Następnie za pomocą XAMPPa należy uruchomić serwery Apache i MySQL.
- Program phpMyAdmin można od teraz uruchomić za pomocą przeglądarki internetowej wpisując w polu adresu *http://localhost/phpmyadmin* lub *http://127.0.0.1/phpmyadmin*.

Serwer Apache umożliwia otwarcie strony (z aplikacją phpMyAdmin) postawionej na własnym komputerze, natomiast serwer MySQL umożliwia korzystanie z bazy danych.

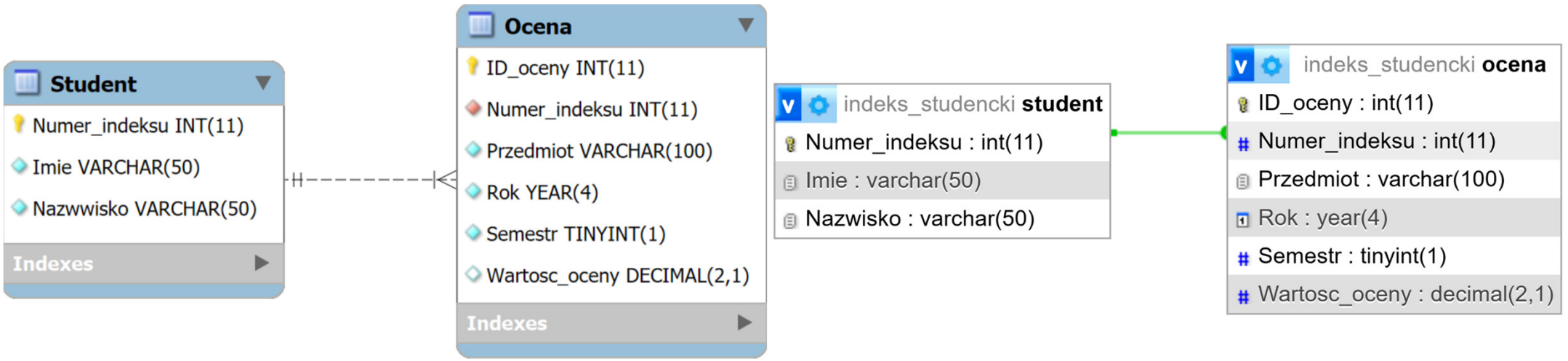
Rodzaje relacji w relacyjnych bazach danych

- Zwróćmy uwagę na połączenie między encjami Student i Ocena.
- Jest to linia, która z jednej strony ma rozgałęzienie (jest ono lepiej widoczne na diagramie MySQL Workbench).
- Oznacza to, że encje są połączone relacją **jeden-do-wielu**.
- **Jeden** student ma **wiele** ocen, ale konkretna ocena należy tylko do **jednego** studenta.



Rodzaje relacji w relacyjnych bazach danych

- W relacji jeden-do-wielu klucz obcy występuje zawsze po stronie tej encji, przy której jest rozgałęzienie.

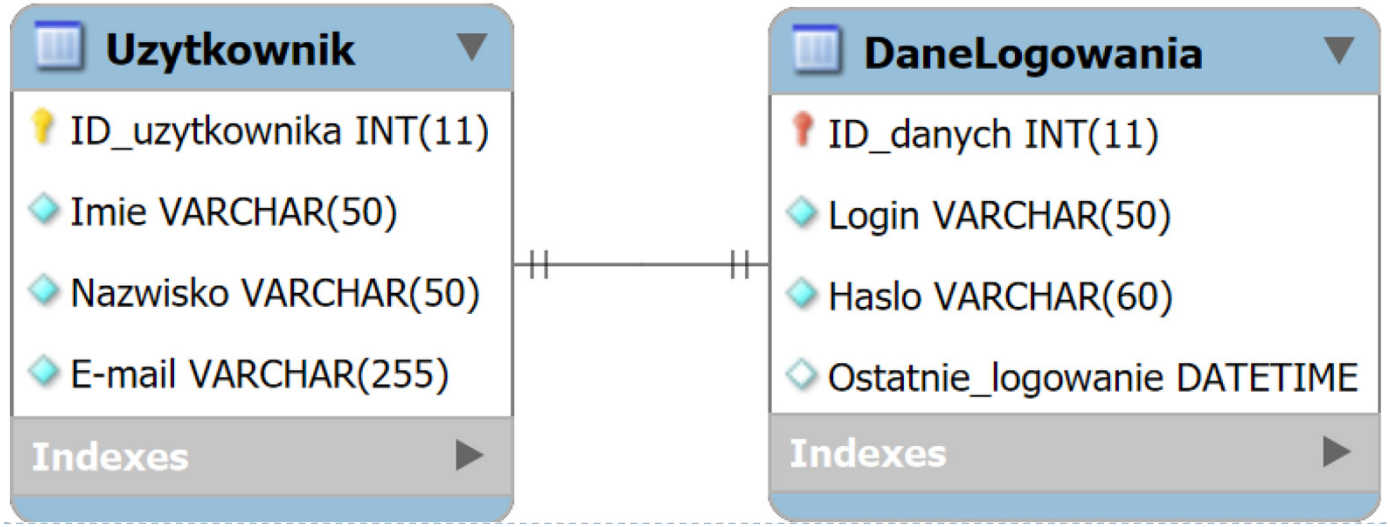


Rodzaje relacji w relacyjnych bazach danych

- Czy w relacyjnych bazach danych są możliwe relacje inne niż **jeden-do-wielu**?
- Relacje **jeden-do-jednego** są dopuszczalne, ale rzadko stosowane.
- Relacje **wiele-do-wielu** są niedopuszczalne. Mogą wystąpić na początkowym etapie projektowania bazy danych, ale trzeba je później wyeliminować.

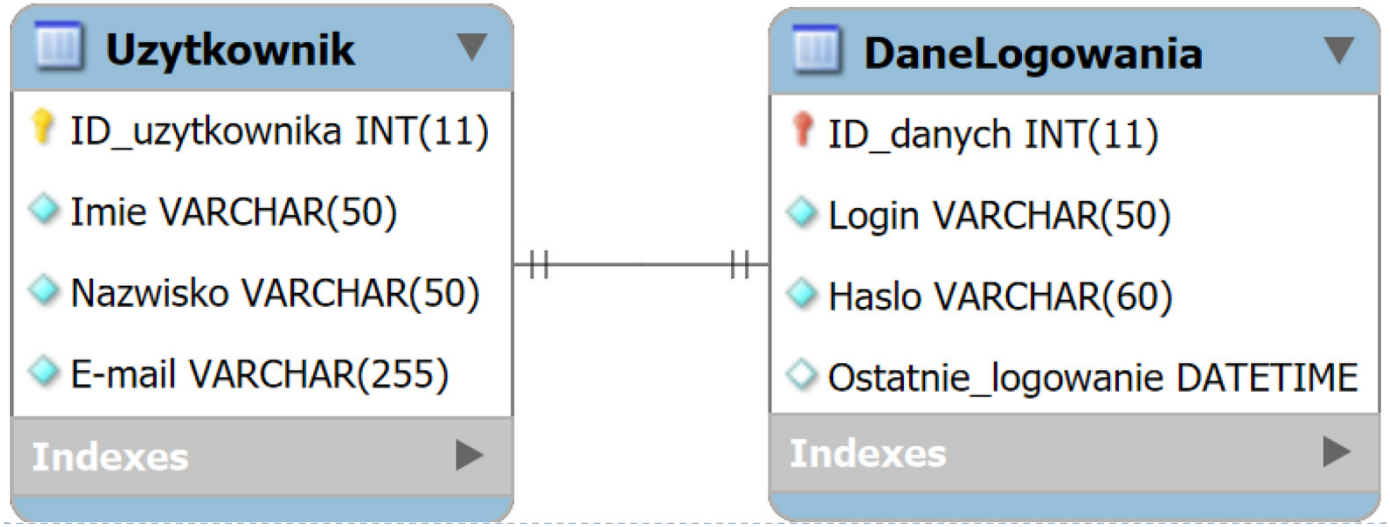
Relacja jeden-do-jednego

- Połączenie jeden-do-jednego nie ma rozgałęzienia. Co ono oznacza w praktyce?
- Użytkownik ma tylko **jeden** zestaw danych logowania (nie może logować się przy użyciu różnych haseł).
- Dane logowania dotyczą tylko **jednego** użytkownika.



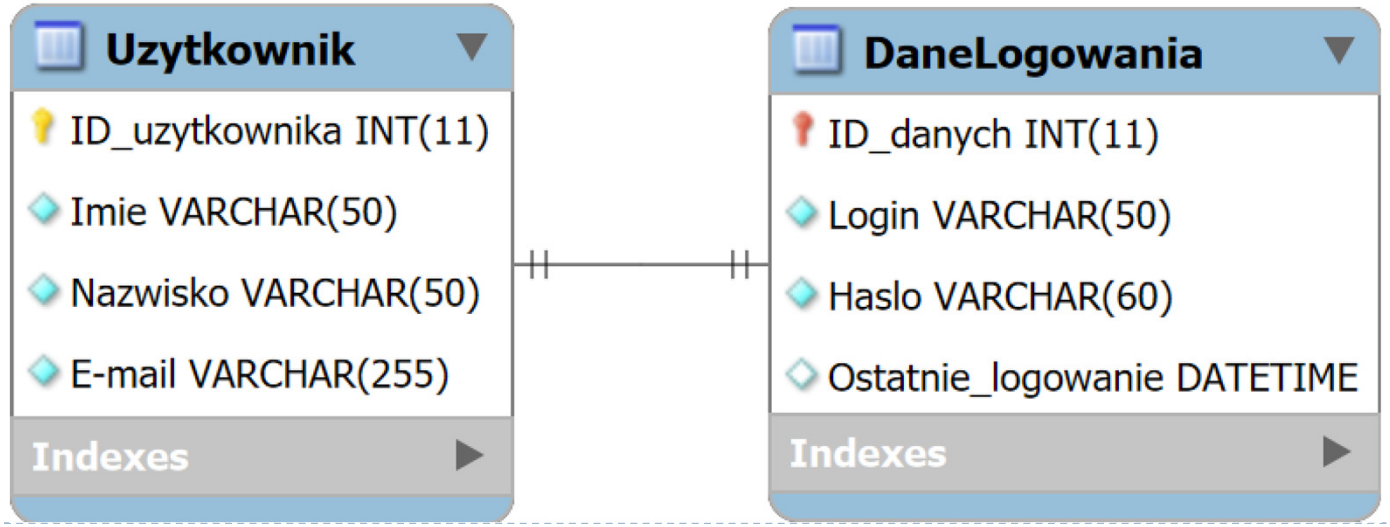
Relacja jeden-do-jednego

- W której encji wstawia się klucz obcy w relacji jeden-do-jednego?
- Z technicznego punktu widzenia można w dowolnej encji.
- W praktyce klucz najlepiej wstawić w tej tabeli, która jest logicznie zależna od drugiej. W tym przypadku dane logowania są zależne od użytkownika, a nie użytkownik od danych logowania (aby istniały dane logowania, musi istnieć użytkownik).



Relacja jeden-do-jednego

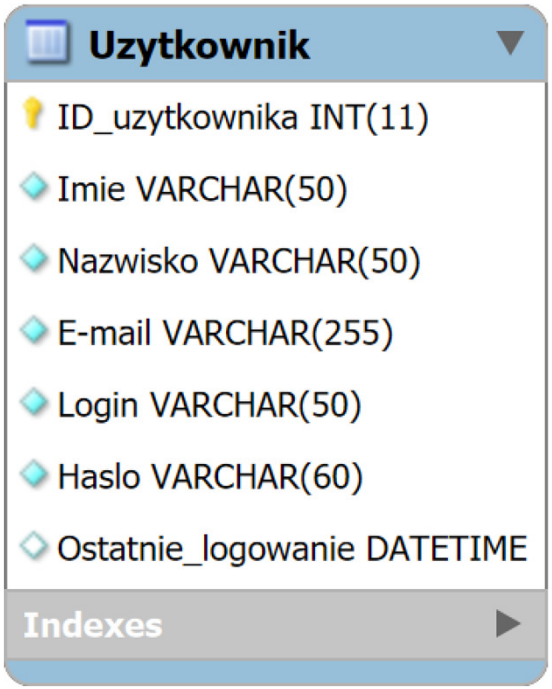
- Należy zwrócić uwagę, że w encji DaneLogowania klucz obcy ID_danych jest jednocześnie kluczem głównym (ikona czerwonego klucza).
- Jest to najlepszy sposób na realizację relacji jeden-do-jednego w praktyce.
- Wartość takiego klucza obcego nie może się powtarzać, ponieważ, jako klucz główny jest on unikalny. Oznacza to, że dane logowania zawsze dotyczą tylko jednego użytkownika.



Relacja jeden-do-jednego

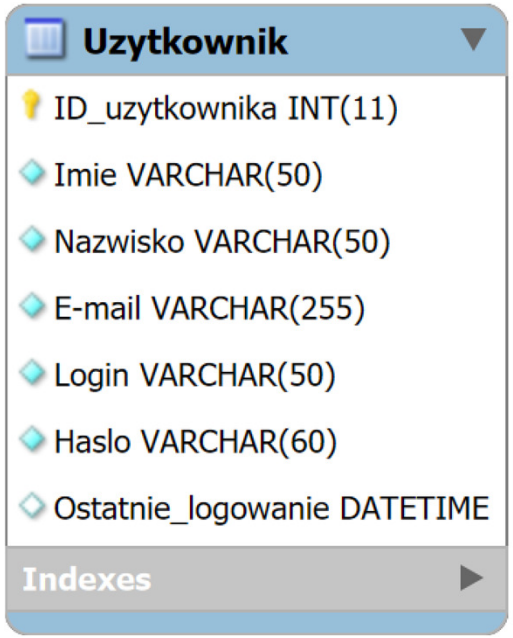
- Czy relacja jeden-do-jednego jest absolutnie konieczna? Czy można zamiast niej zrobić tylko jedną encję Uzytkownik i powstawiać do niej wszystkie atrybuty?

- Można. Oba podejścia mają sens, ale mają również wady i zalety.



Zalety jednej tabeli z umieszczonymi w niej wszystkimi atrybutami

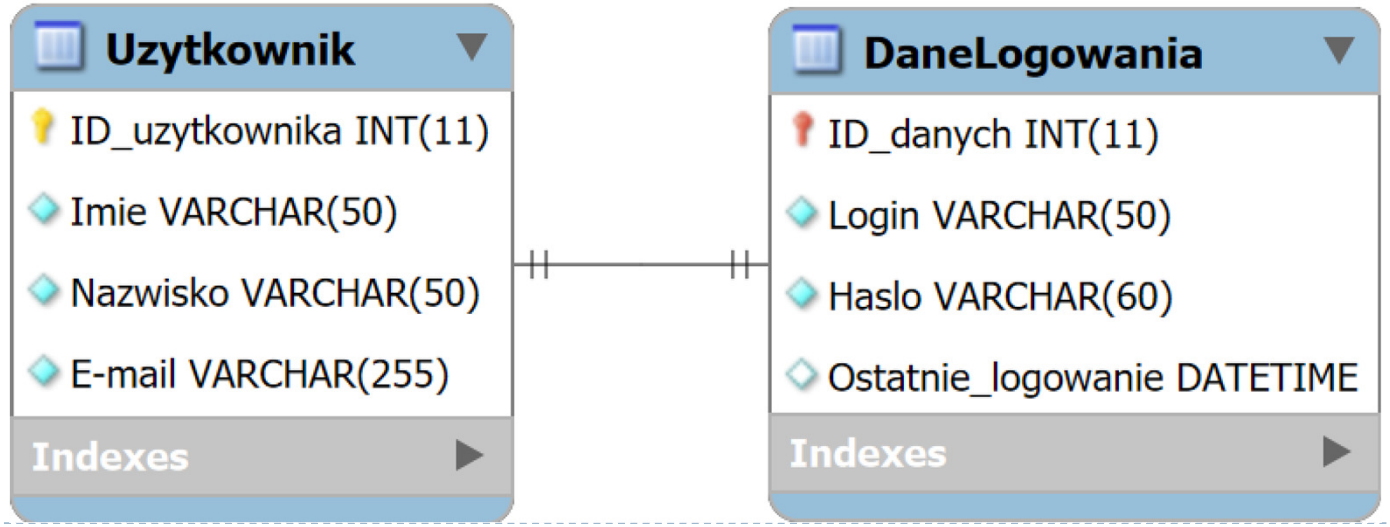
- Struktura bazy danych będzie uproszczona. Ułatwi to:
  - utworzenie takiej bazy danych,
  - wykonywanie zapytań i modyfikacje danych,
  - analizę takiej bazy danych, szczególnie w przypadku, gdy składa się ona z wielu tabel.



Zalety utworzenia dwóch tabel połączonych relacją jeden-do-jednego

Zaleta 1. Bezpieczeństwo.

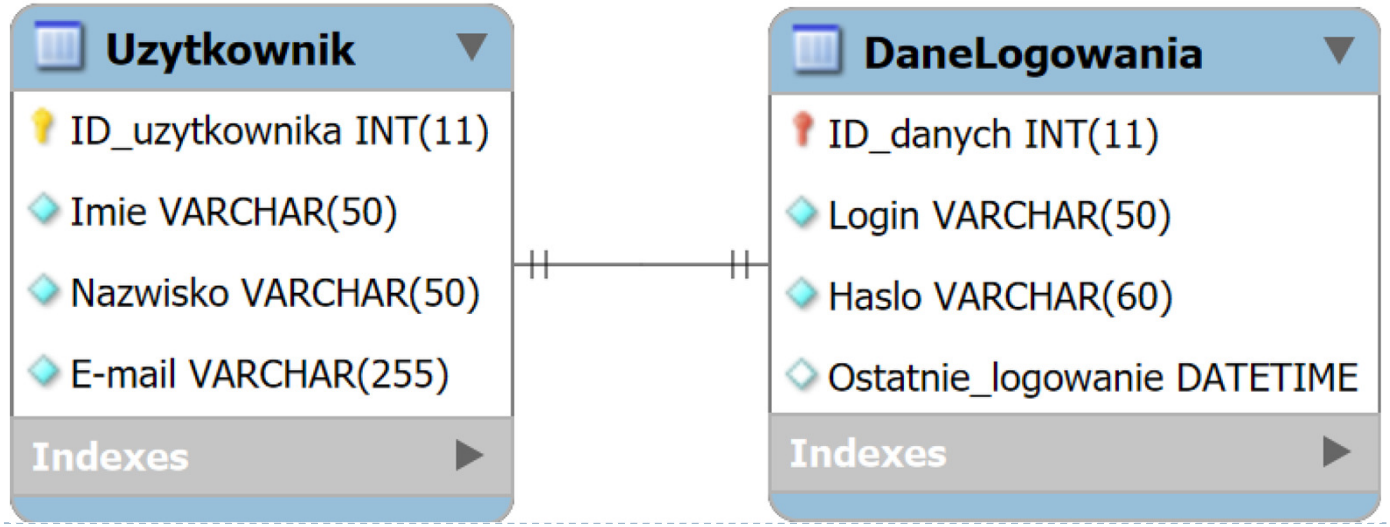
- Np. możemy nadać działowi HR uprawnienia z dostępem do tabeli Uzytkownik (dane osobowe), ale nie dawać mu dostępu do tabeli DaneLogowania (np. loginów, daty ostatniego logowania).



Zalety utworzenia dwóch tabel połączonych relacją jeden-do-jednego

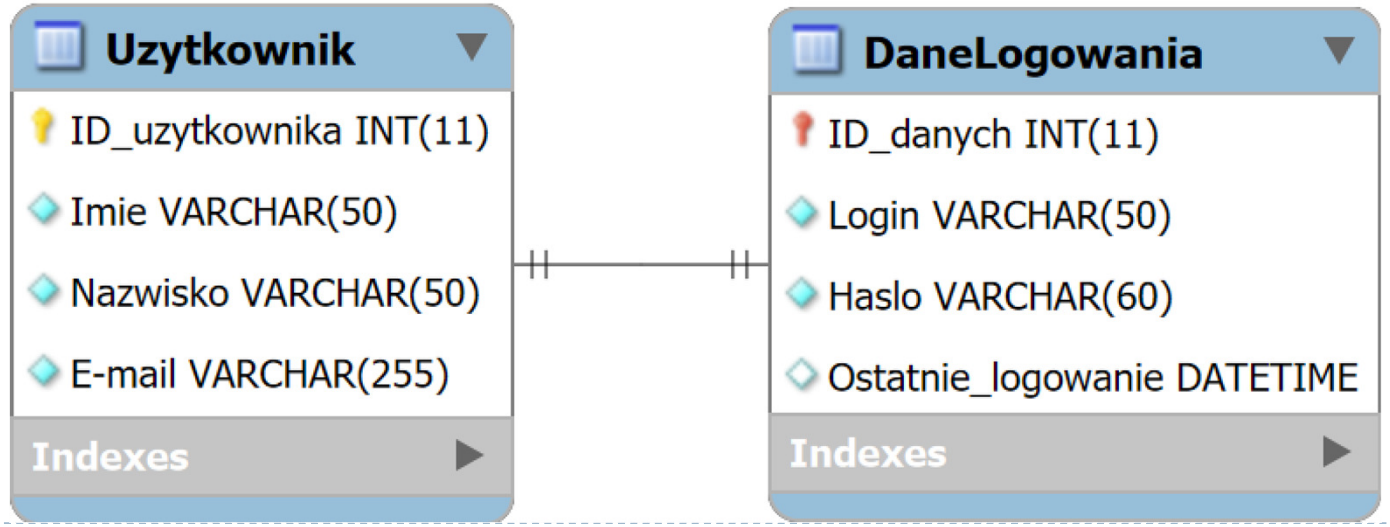
Zaleta 2. Częstotliwość dostępu (wydajność).

- Dane logowania są często aktualizowane, ale dane osobowe rzadko.
- Tabela Uzytkownik nie będzie obciążana częstymi zmianami danych logowania.
- Zapytania do tabeli Uzytkownik nie muszą ściągać niepotrzebnych danych.
- Przyczyni się to do zwiększenia szybkości operacji, co ma znacznie szczególnie w przypadku dużych baz danych.



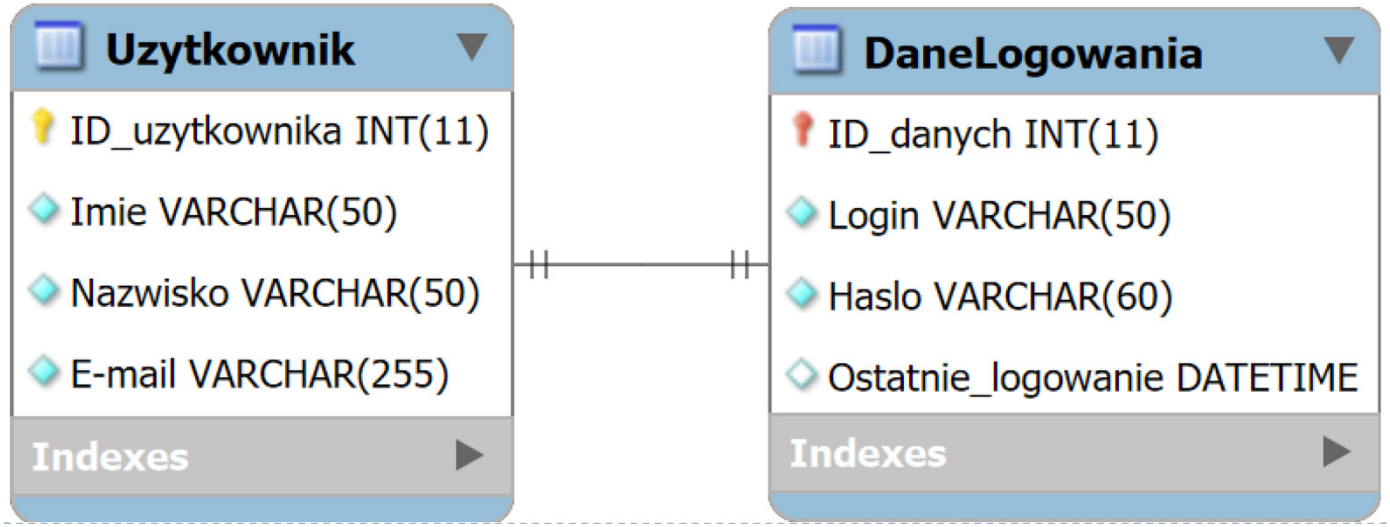
Dodatkowe spostrzeżenie – atrybut Ostatnie_logowanie

- Warto zwrócić uwagę, że w bazie danych użytkowników atrybut Ostatnie_logowanie:
  - jest typu DATETIME (data i czas w formacie YYYY-MM-DD HH:MI:SS);
  - jest atrybutem opcjonalnym; powodem jest fakt, że użytkownik, który założył konto, ale nie zdążył się jeszcze zalogować, nie będzie mieć wpisanej żadnej daty logowania.



Dodatkowe uwaga – przechowywanie haseł w bazie danych

- Jednym z atrybutów encji DaneLogowania jest hasło użytkownika.
- Haseł nie powinno się przechowywać w bazie danych w postaci jawnej (bezpośredniej) ze względów bezpieczeństwa.
- Nikt nie powinien mieć dostępu do takich haseł (ani haker, który uzyska dostęp do bazy danych, ani nawet administrator bazy danych).



Dodatkowe uwaga – przechowywanie haseł w bazie danych

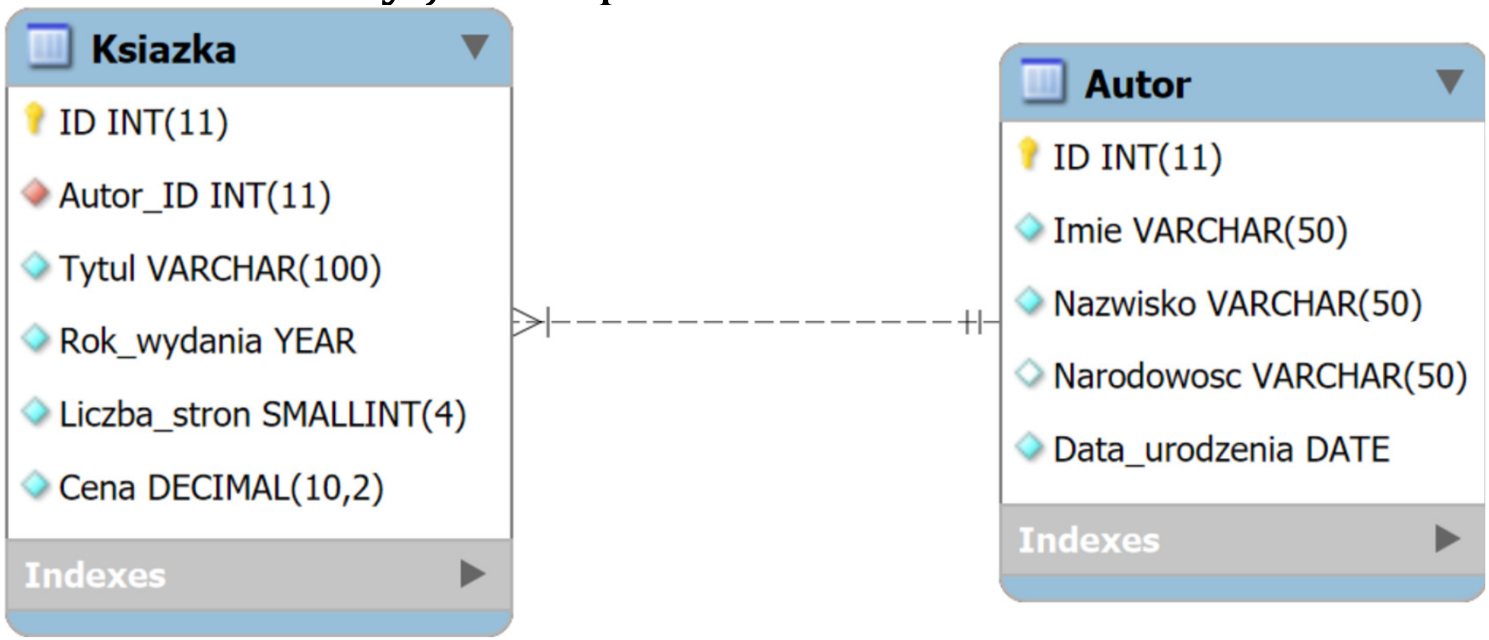
- Najczęściej hasła przechowuje się w formie zahaszowanej.
- Haszowanie polega na przekształceniu hasła na unikalny ciąg znaków.
- Taki ciąg jest nieodwracalny, co oznacza, że nie można z niego odzyskać oryginalnego hasła.
- Za każdym razem, gdy użytkownicy logują się do serwisów, wpisane przez nich hasła są haszowane i porównywane z zahaszowanym hasłem istniejącym w bazie danych.
- Popularne metody haszowania, to *bcrypt*, *argon2*, *PBKDF2*, *scrypt*.
- Metody te zwracają kod o stałej szerokości znaków.
  Przykładowo, w przypadku *bcrypt* jest to 60 znaków.

Ciekawostka – Dlaczego więc hasła czasami wyciekają?

- Pomimo reguły haszowania haseł zdarza się, że czasem hakerzy uzyskują do nich dostęp. Jakie mogą być powody?
  - W niektórych (szczególnie starych, tanich) systemach/stronach hasła mogą być przechowywane bez haszowania.
  - Używa się słabych lub przestarzałych metod haszowania, np. MD5, SHA-1.
  - Słabe hasła użytkowników mają znane kody, a więc hakerzy mając kod sprawdzają, czy nie pasuje on do jakiegoś znanego wzorca.
    - Dlatego nie warto używać haseł typu: 123456, qwerty, haslo123).
  - Hasła wyciekają nie z bazy danych, tylko innymi sposobami: phishing, keylogger itp.

Relacje wiele-do-wielu

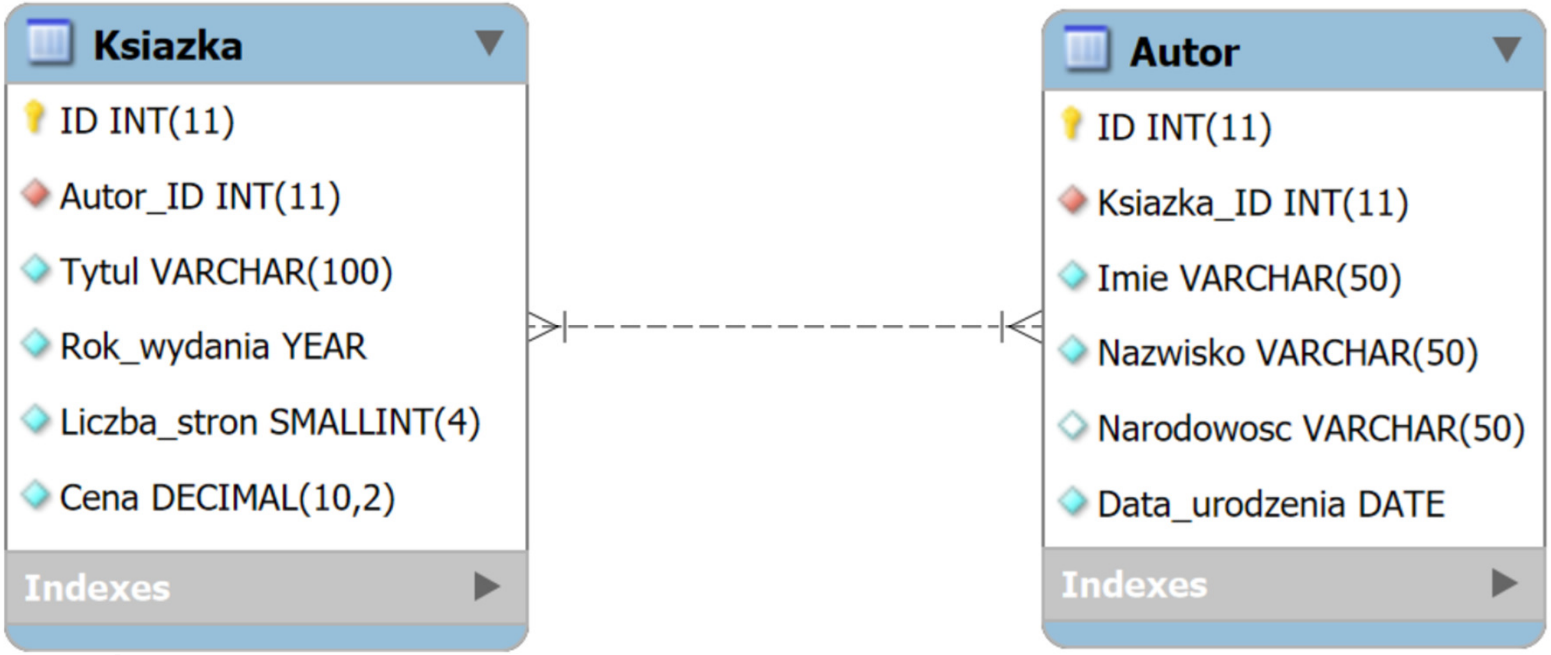
- Rozważmy przykład bazy danych księgarni.
- Czy taka struktura bazy jest odpowiednia?



- Odpowiedź: Nie jest odpowiednia, ponieważ jedna książka może mieć wielu autorów.

Relacje wiele-do-wielu

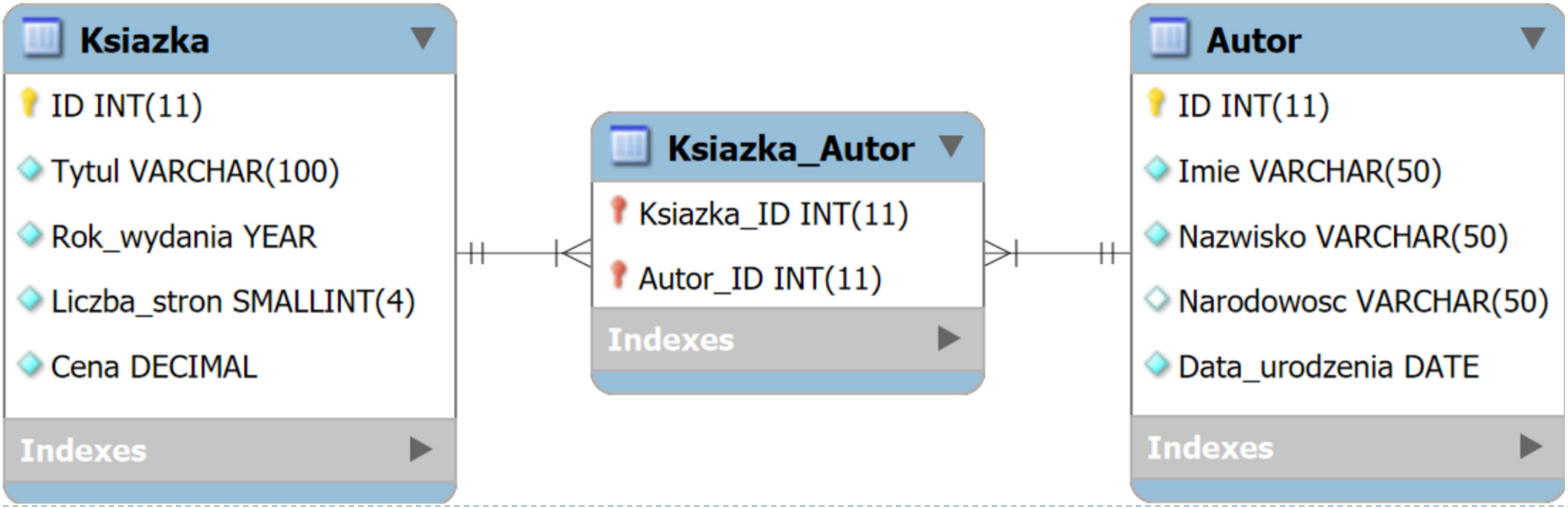
- Czy zatem baza danych o takiej strukturze jest poprawna?



- Odpowiedź: Jest poprawna na wstępnym etapie projektowania.
- Końcowa wersja diagramu musi zostać zmieniona, ponieważ relacyjne bazy danych nie dopuszczają relacji wiele-do-wielu.

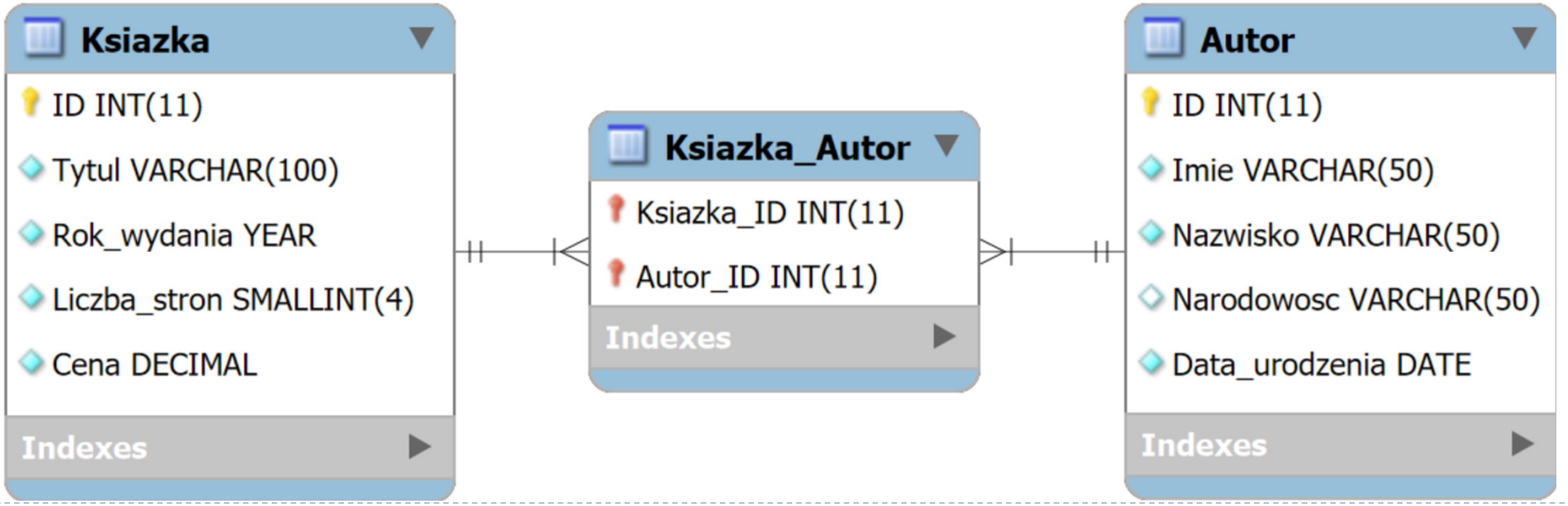
Relacje wiele-do-wielu – co zrobić, że je wyeliminować w końcowym projekcie bazy danych?

- Aby pozbyć się relacji wiele-do-wielu, należy utworzyć dodatkową encję, z którą pozostałe encje są połączone relacjami jeden-do-wielu.
- W ten sposób, w naszej bazie danych autor może mieć wiele książek, a książka wielu autorów.



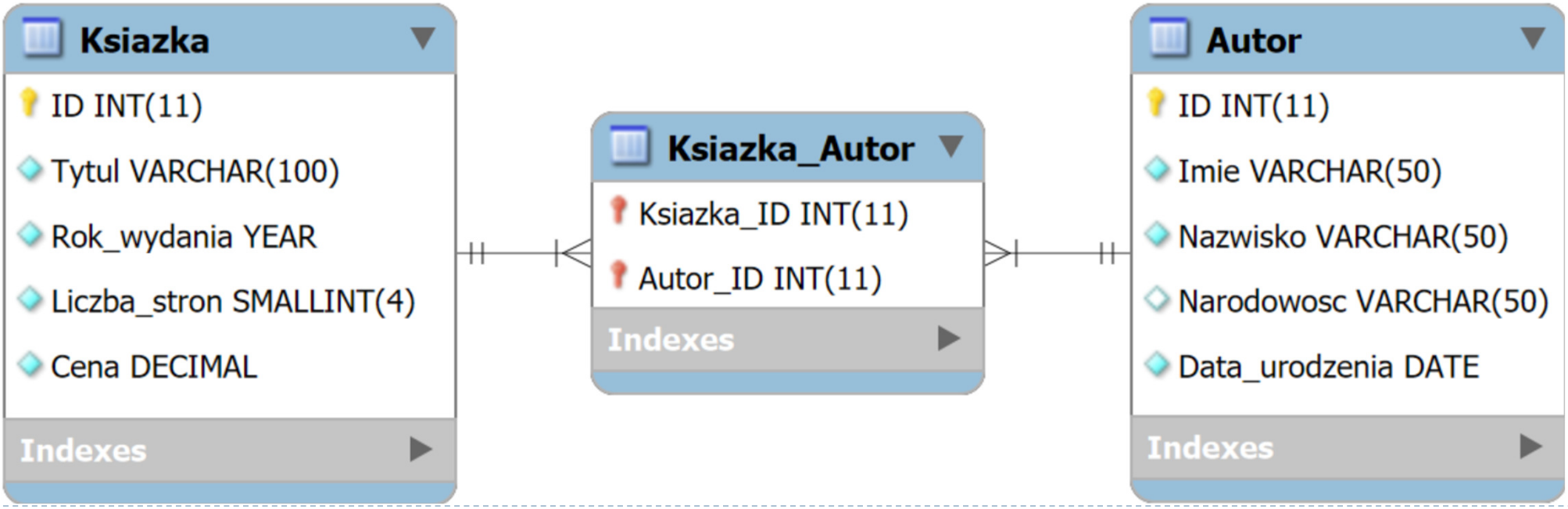
Relacje wiele-do-wielu – co zrobić, że je wyeliminować w końcowym projekcie bazy danych?

- Takie pozbywanie się relacji wiele-do-wielu nazywa się czasem normalizacją bazy danych.



Relacje wiele-do-wielu – co zrobić, że je wyeliminować w końcowym projekcie bazy danych?

- Należy zwrócić uwagę, że w encji Ksiazka_Autor nie ma klucza głównego.
- Tak naprawdę klucz główny występuje, ale nie jest on pojedynczym atrybutem, tylko złożeniem dwóch kluczy obcych.
- Takie złożenie jest unikalne, a więc może być kluczem głównym.



Baza danych sklepu

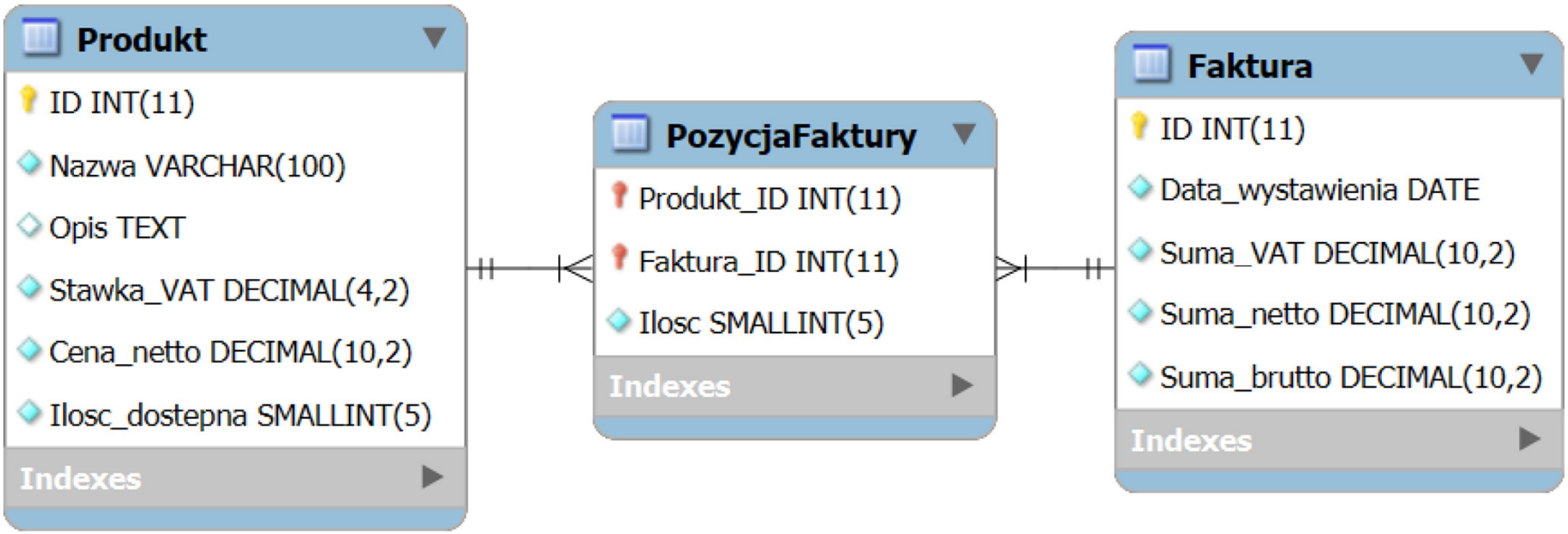
- Rozważmy bazę danych sklepu, w której przechowywane są informacje o produktach i wystawionych fakturach.
- Czy wystarczą w niej encje Produkt i Faktura? Jakiego rodzaju połączenie będzie między nimi?

- Na jeden produkt może być wystawione wiele faktur.
- Na jednej fakturze może być wiele produktów.

- Znów mamy więc do czynienia z relacją wiele-do-wielu.
- Spróbujmy od razu utworzyć znormalizowaną bazę danych z encją pomocniczą, z wyeliminowaną relacją wiele-do-wielu.

Baza danych sklepu

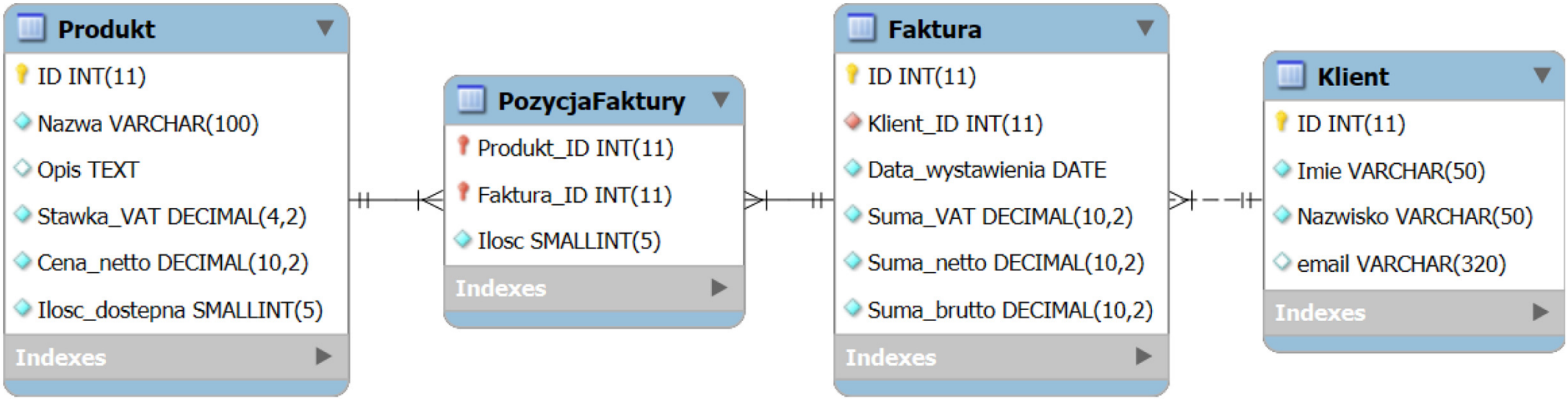
- Baza danych mogłaby wyglądać tak:



- Najprawdopodobniej posiadając sklep chcielibyśmy również przechowywać informacje o klientach. Dodajmy więc encję Klient.
- Z którą encją powinna być połączona encja Klient? Jakim typem relacji?

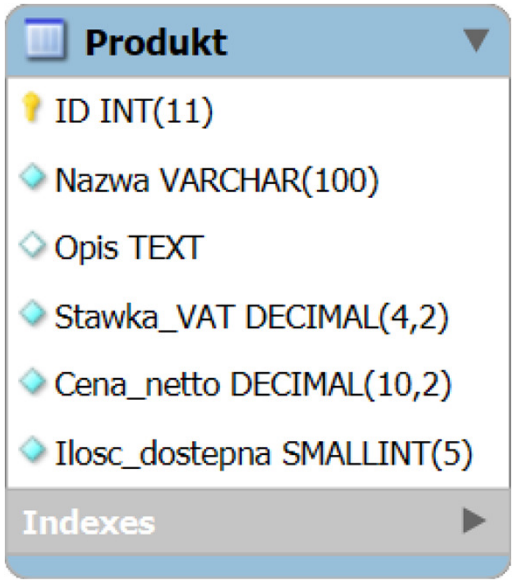
Baza danych sklepu

- Najrozsądniej byłoby połączyć klienta z fakturą (nie z pojedynczą pozycją, ale z całą fakturą) przy użyciu relacji jeden-do-wielu.
- Dlaczego?



Klucz unikatowy

- W tabeli Produkt mamy atrybut Nazwa, który nie jest kluczem (ani głównym ani obcym).
- Być może chcielibyśmy, żeby pomimo tego nazwy produktów były unikalne (unikatowe).
- Jest na to sposób. Taki atrybut można oznaczyć jako **klucz unikatowy** (klucz jednoznaczny, ang. unique key).
- Nie będzie on identyfikatorem tabeli, ale jego wartości nie będą mogły się powtarzać (tak jakby był kluczem głównym).



Relacje rekurencyjne

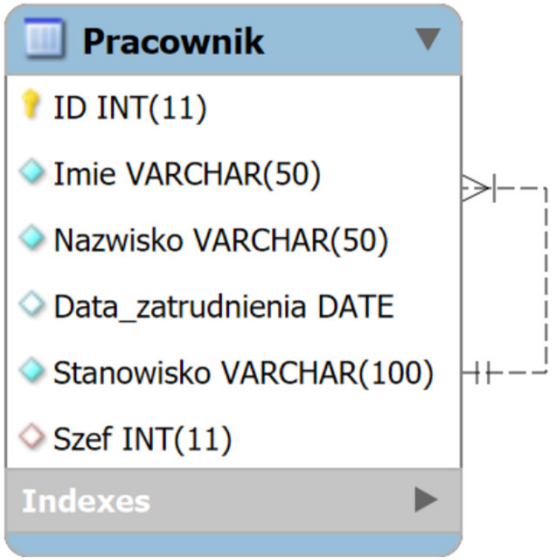
- W poniższej tabeli pracownicy mają kolumnę Szef, w której znajdują się liczby całkowite.
- Liczby te odnoszą się do ID innych pracowników.
- W ten sposób można oznaczyć, hierarchię w firmie, tzn. określić kto jest czyim szefem.
- Jan Nowak nie ma wpisanej żadnej wartości w kolumnie Szef, ponieważ jest dyrektorem i nie ma żadnego szefa.

Pracownik

ID	Imie	Nazwisko	Data_zatrudnienia	Stanowisko	Szef
131	Jan	Nowak	2023-10-01	dyrektor	NULL
132	Tomasz	Kowalski	2023-10-15	kierownik projektu	131
133	Alicja	Iksińska	2024-02-13	kierownik administracyjny	131
134	John	Doe	2024-04-03	wykonawca projektu	132
135	Jane	Doe	2025-05-19	sekretarka	133

Relacje rekurencyjne

- W diagramie ERD takiej bazy danych encja Pracownik jest połączona sama z sobą relacją jeden-do-wielu.
- Oznacza to, że jeden pracownik (szef) ma pod sobą **wielu** pracowników i sam jest pracownikiem, który ma nad sobą **jednego** szefa.
- Taka relacja nazywa się relacją rekurencyjną.

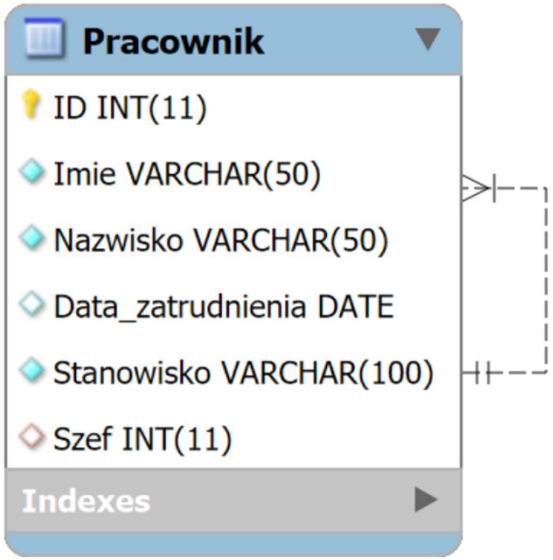


Pracownik

ID	Imie	Nazwisko	Data_zatrudnienia	Stanowisko	Szef
131	Jan	Nowak	2023-10-01	dyrektor	NULL
132	Tomasz	Kowalski	2023-10-15	kierownik projektu	131
133	Alicja	Iksińska	2024-02-13	kierownik administracyjny	131
134	John	Doe	2024-04-03	wykonawca projektu	132
135	Jane	Doe	2025-05-19	sekretarka	133

Relacje rekurencyjne

- Jak zrealizować relację rekurencyjną od strony technicznej?
- Szef jest kluczem obcym (opcjonalnym – niewypełniony kwadrat) odnoszącym się do klucza głównego występującego w tej samej tabeli (ID).
- Połączenie zaznaczone jest przerywaną linią ze względu na opcjonalność klucza obcego.



Pracownik

ID	Imie	Nazwisko	Data_zatrudnienia	Stanowisko	Szef
131	Jan	Nowak	2023-10-01	dyrektor	NULL
132	Tomasz	Kowalski	2023-10-15	kierownik projektu	131
133	Alicja	Iksińska	2024-02-13	kierownik administracyjny	131
134	John	Doe	2024-04-03	wykonawca projektu	132
135	Jane	Doe	2025-05-19	sekretarka	133

Język SQL (Structured Query Language)

- SQL jest językiem zarządzania bazą danych.
- Jest wspólny dla wszystkich systemów relacyjnych baz danych: MySQL, PostgreSQL, Microsoft Access, Microsoft SQL Server, Oracle, IBM DB2 itd.
- W każdym z tych systemów występują jednak różne dialekty, czyli mają pewne różnice w składni. Oznacza to, że w różnych systemach typy danych, funkcje, słowa kluczowe itp.:
  - mogą mieć inne nazwy i sposób użycia;
  - mogą być charakterystyczne dla danego systemu i nie występować w innych.
- Główny trzon języka SQL jest jednak wspólny dla wszystkich systemów.

Język SQL – główne zastosowania

- Tworzenie i usuwanie bazy danych.
- Tworzenie i modyfikacja struktury bazy danych,
  czyli tabel i relacji między nimi.
- Pobieranie danych z bazy (np. w celu ich wyświetlenia).
- Wstawianie, usuwanie i aktualizowanie danych.
- Zarządzanie uprawnieniami i rolami użytkowników bazy danych. Np.:
  - Zezwolenie grupie nauczycieli na wpisywanie, zmianę i wyświetlanie ocen;
  - Zezwolenie grupie uczniów na sprawdzanie ocen;
  - Zezwolenie dyrektorowi szkoły na dodawanie do bazy nowych nauczycieli.

Tworzenie i usuwanie bazy danych za pomocą języka SQL

- Tworzenie bazy danych za pomocą języka SQL jest bardzo proste:

```
CREATE DATABASE nazwa_bazy
```

- Bazę możemy usunąć w taki sposób:

```
DROP DATABASE nazwa_bazy
```

Tworzenie i usuwanie bazy danych za pomocą języka SQL

- Jeśli chcemy utworzyć bazę danych, ale tylko wtedy, jeśli nie istnieje baza o podanej nazwie, to można to zrobić w następujący sposób:

```
CREATE DATABASE IF NOT EXISTS nazwa_bazy
```

- Jeśli chcemy usunąć bazę danych, ale tylko wtedy, jeśli istnieje baza o podanej nazwie, to można to zrobić w następujący sposób:

```
DROP DATABASE IF EXISTS nazwa_bazy
```

- Polecenia z poprzednich slajdów zwrócą błąd jeśli baza danych już istnieje (CREATE) lub nie istnieje (DROP).
- Jeśli mamy cały skrypt z poleceniami SQL zaczynający się od utworzenia (i/lub usunięcia) bazy danych, to lepiej użyć wersji z tego slajdu, żeby uchronić się przed przerwaniem skryptu po pojawieniu się błędu już na początku.

Tworzenie tabel za pomocą języka SQL

```
CREATE TABLE Ocena (
  ID_oceny INT(11) NOT NULL AUTO_INCREMENT,
  Numer_indeksu INT(11) NOT NULL,
  Przedmiot VARCHAR(100) NOT NULL,
  Rok YEAR(4) NOT NULL,
  Semestr TINYINT(1) NOT NULL,
  Wartosc_oceny DECIMAL(2,1) DEFAULT NULL,
  PRIMARY KEY (ID_oceny),
  CONSTRAINT numer_indeksu_fk
      FOREIGN KEY (Numer_indeksu) REFERENCES Student (Numer_indeksu)
)
```

`AUTO_INCREMENT` wyjaśnienie slajdzie 114.

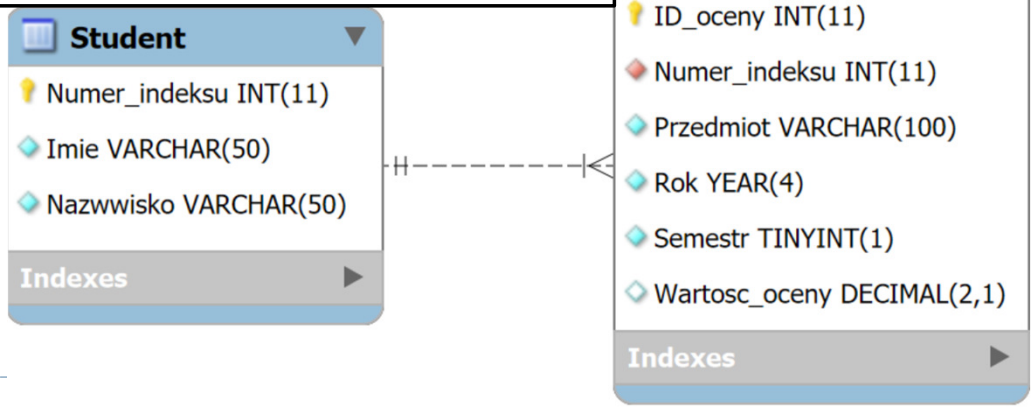
`NOT NULL` oznacza atrybut obowiązkowy.

`DEFAULT NULL` oznacza atrybut opcjonalny z domyślną wartością NULL.

`FOREIGN KEY` oznacza klucz obcy.

`PRIMARY KEY` oznacza klucz główny.

Nazwa ograniczenia może być dowolna.



Wprowadzanie zmian w strukturze tabeli

- Jeśli chcemy wprowadzić zmiany w strukturze już utworzonej tabeli, to możemy skorzystać z polecenia `ALTER TABLE`.

- Poniższe polecenie dodaje do tabeli Ocena atrybut (obowiązkowy) E_mail oraz zmienia atrybut Rok tak, aby był typu DATE i był opcjonalny:

```
ALTER TABLE Ocena
  ADD E_mail VARCHAR(255) NOT NULL,
  MODIFY Rok DATE DEFAULT NULL
```

Usuwanie tabel za pomocą języka SQL

- Usunąć tabelę możemy za pomocą instrukcji:

```
DROP TABLE Ocena
```

- Klauzule IF EXISTS oraz IF NOT EXISTS również działają z tabelami. Można więc utworzyć tabelę o ile już nie istnieje lub usunąć tabelę, jeśli już istnieje:

```
CREATE TABLE IF NOT EXISTS Ocena(
  …
)
```

```
DROP TABLE IF EXISTS Ocena
```

Pobieranie danych – SELECT

- Zapytanie SELECT służy do pobierania danych z bazy.
- Domyślnie pobrane dane są wyświetlane w oknie programu do zarządzania bazą danych.
- Składnia zapytania SELECT jest następująca:

```
SELECT <nazwy kolumn> FROM <nazwy tabel>
```

- Przykład. Wyświetl tytuły wszystkich książek.

```
SELECT Tytul FROM Ksiazka
```

- Jeśli chcemy wyświetlić wszystkie kolumny danej tabeli, to zamiast wypisywania ich nazw, można zastosować symbol gwiazdki:

```
SELECT * FROM Ksiazka
```

Pobieranie danych – łączenie kolumn w jedną

- Jeśli chcemy wyświetlić kilka kolumn w jednej (i np. oddzielić je spacją), to w bazach danych MySQL możemy użyć funkcji `CONCAT`.
- Zapytanie z poniższego przykładu zwróci imię i nazwisko każdego autora zapisane w pojedynczej kolumnie:

```
SELECT CONCAT(Imie, ' ', Nazwisko)
FROM Autor
```

- W bazach danych innych niż MySQL (np. PostgreSQL) do łączenia kolumn zamiast funkcji `CONCAT` stosuje się operator `||`:

```
SELECT Imie || ' ' || Nazwisko
FROM Autor
```

Pobieranie danych – `SELECT` z warunkiem

- Jeśli chcemy pobrać tylko te dane, które spełniają określony warunek, możemy skorzystać z klauzuli `WHERE`.
- Poniższe zapytanie powoduje pobranie danych o wszystkich książkach, których cena jest większa niż 50 zł.

```
SELECT * FROM Ksiazka
WHERE Cena > 50
```

- Warunki w języku SQL tworzymy podobnie jak w Pythonie.

Pobieranie danych bez duplikatów

- Jeśli chcemy pobrać wartości danej kolumny bez duplikatów (niepowtarzające się), to możemy skorzystać z funkcji `DISTINCT`.
- Przykładowo, poniższe zapytanie wyświetla wszystkie stawki VAT produktów. Wyświetlone jest tyle stawek, ile mamy produktów w bazie.

```
SELECT Stawka_VAT FROM Produkt
```

- Tak naprawdę stosujemy jedynie dwie stawki VAT dla produktów w naszej bazie: 5% i 23%.
- Gdybyśmy chcieli wyświetlić jedynie te dwie stawki jako wartości unikalne, niepowtarzające się dla każdego produktu, to zapytanie należy zmienić tak:

```
SELECT DISTINCT(Stawka_VAT) FROM Produkt
```

Podstawowe operatory języka SQL

Porównania:

- > większe
- < mniejsze
- >= większe lub równe
- <= mniejsze lub równe
- = równe
- != różne
- <> różne

Arytmetyczne:

- + suma
- - różnica
- * iloczyn
- / iloraz
- % reszta z dzielenia

Logiczne:

- AND – „i”
- OR – „lub”
- NOT – negacja („nieprawda, że”)

Dodatkowe operatory porównawcze języka SQL

- `BETWEEN` x `AND` y – sprawdzenie, czy kolumna ma wartość należącą do określonego przedziału (zakresu) (tylko dla przedziałów zamkniętych).
- `IS NULL` – sprawdzenie, czy kolumna nie ma wpisanej żadnej wartości.
- `IS NOT NULL` – sprawdzenie, czy kolumna ma wpisaną jakąkolwiek wartość (UWAGA: zapis `x != NULL` nie zadziała).
- `IN` – sprawdzenie, czy kolumna ma wartość z podanego zbioru wartości. Przykład: wyświetlenie wszystkich ocen z przedmiotów: Informatyka, Matematyka i Elektronika (z pominięciem pozostałych przedmiotów):

```
SELECT * FROM Ocena
WHERE Przedmiot IN ('Informatyka', 'Matematyka', 'Elektronika')
```

- `LIKE` – porównanie wartości z określonym wzorcem (operator `LIKE` zostanie omówiony na późniejszych slajdach).

Priorytety operatorów

- Im niżej występuje operator w tabeli, tym wyższy ma priorytet (wcześniej wykona się jego działanie).

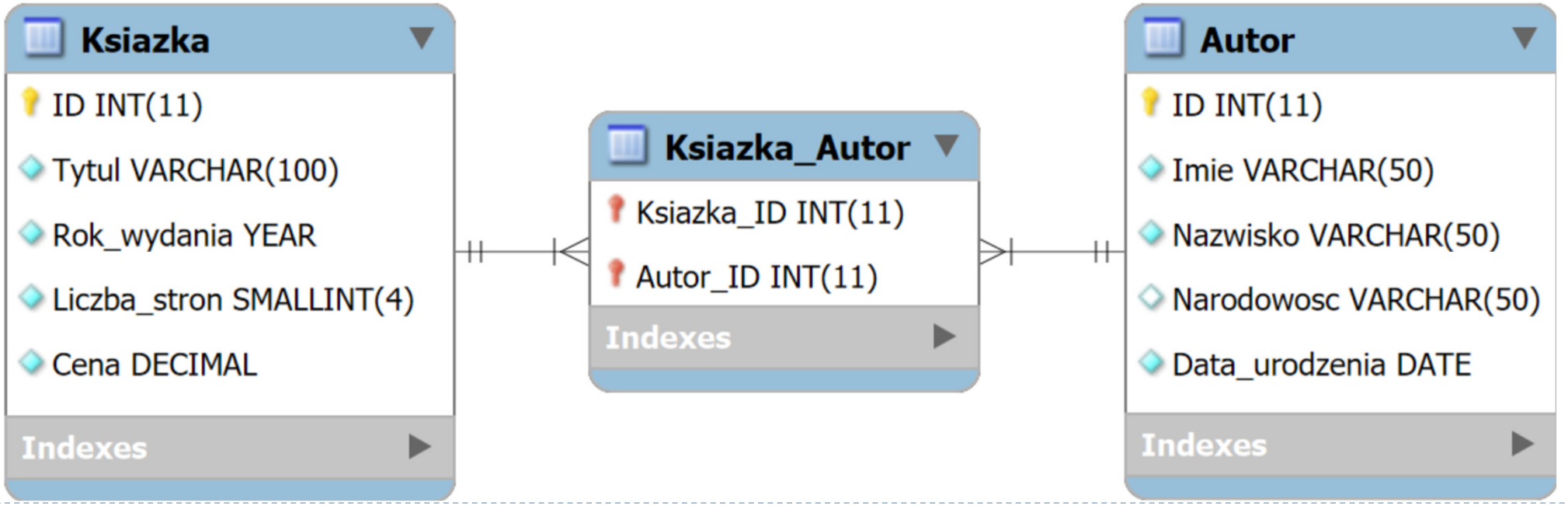
Operator	Opis
`OR`	logiczne „lub”
`AND`	logiczne „i”
`NOT`	logiczna negacja („nie prawda że”)
`=, !=, <, >, <=, >=, BETWEEN, IS NULL, IS NOT NULL, IN, LIKE`	operatory porównania
`+, -`	dodawanie, odejmowanie
`*, /`	mnożenie, dzielenie
`+x, -x`	plus i minus jednoargumentowe, np. -3
`()`	nawias

Czy SQL jest wrażliwy na wielkość liter (tak jak Python)?

- To zależy. W przypadku baz danych MySQL zasady są następujące:
- Słowa kluczowe są niewrażliwe na wielkość liter. Można więc pisać: `SELECT * FROM`, `select * from`, `Select * From` itp.
- Nazwy kolumn są również niewrażliwe na wielkość liter. Można więc pisać: `SELECT nazwisko`, `SELECT NAZWISKO` itp.
- Nazwy tabel i baz danych – zależy od systemu operacyjnego:
  - Windows – niewrażliwe;
  - Linux – wrażliwe;
  - macOS – domyślnie niewrażliwe.
- Dane tekstowe (np. typ VARCHAR) – zależy od ustawionych reguł kodowania i sortowania (ang. collation). Domyślne reguły kodowania i sortowania `utf8mb4_unicode_ci` są niewrażliwe na wielkość liter (ci = case insensitive).

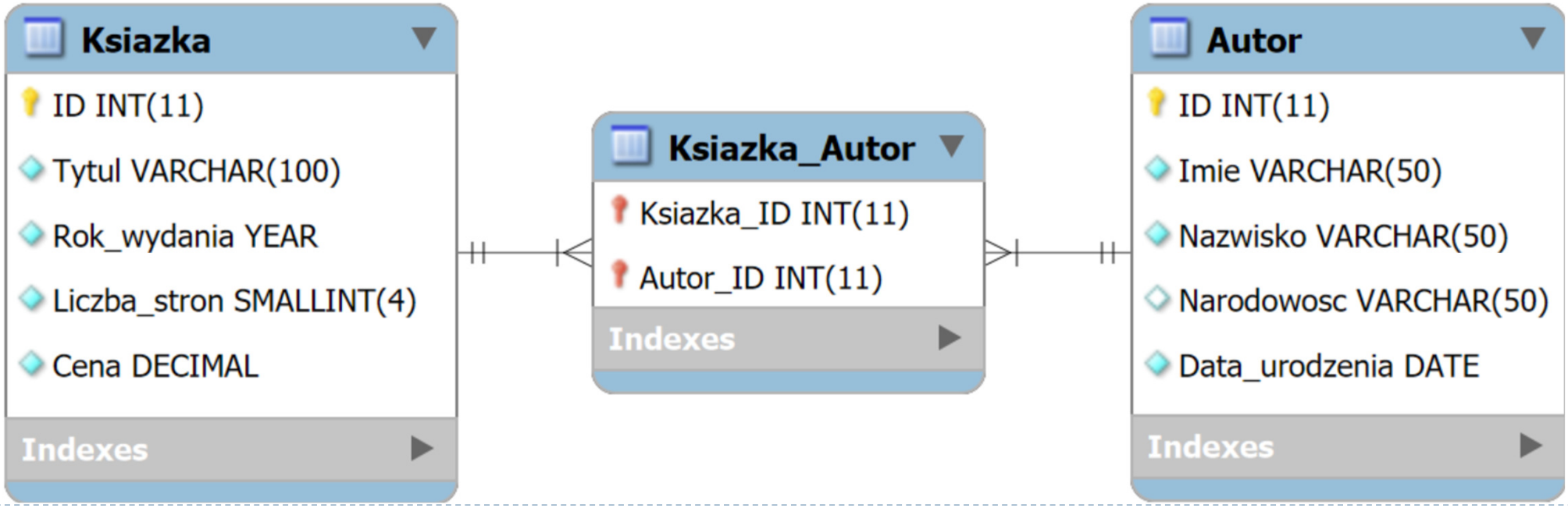
Zadanie 2

- Za pomocą języka SQL wyświetlić tytuł i rok wydania książek, które zostały wydane w latach 2019-2021 włącznie.



Zadanie 3

- Za pomocą języka SQL wyświetlić wszystkie dane autorów, którzy są Polakami lub urodzili się przed 1970 r.



Sortowanie pobranych danych

- Jeśli chcemy, żeby pobrane dane były posortowane, należy na końcu zapytania wstawić klauzulę `ORDER BY`.
- Poniższe zapytanie powoduje pobranie posortowanych danych o wszystkich książkach, których cena nie przekracza 75 zł.

```
SELECT * FROM Ksiazka
WHERE Cena <= 75
ORDER BY Cena ASC
```

- Słowo `ASC` oznacza, że ceny będą posortowane rosnąco. Sortowanie w kolejności malejącej można wykonać za pomocą słowa `DESC`.
- Domyślnie dane są sortowane rosnąco, a więc słowo `ASC` można pominąć, jeśli chcemy sortować w kolejności rosnącej.

Sortowanie pobranych danych

- Sortować możemy liczby (całkowite, zmiennoprzecinkowe, stałoprzecinkowe), napisy (alfabetycznie) oraz daty (np. od najwcześniejszej do najpóźniejszej).
- Można sortować po wielu kolumnach, których nazwy oddziela się przecinkiem.
- Poniższe zapytanie sortuje pobrane dane po cenie (rosnąco), a w przypadku, gdy książki będą miały te same ceny, posortuje je po tytule w kolejności alfabetycznej.

```
SELECT * FROM Ksiazka
WHERE Cena <= 75
ORDER BY Cena, Tytul
```

Ograniczenia liczby pobieranych wierszy

- Jeśli chcemy pobrać jedynie określoną liczbę wierszy spełniających podane kryteria, to możemy skorzystać z klauzuli `LIMIT`.
- Poniższe zapytanie pobiera jedynie trzy najdroższe książki w bazie (sortujemy od najdroższych i wyświetlamy pierwsze trzy):

```
SELECT * FROM Ksiazka
ORDER BY Cena DESC
LIMIT 3
```

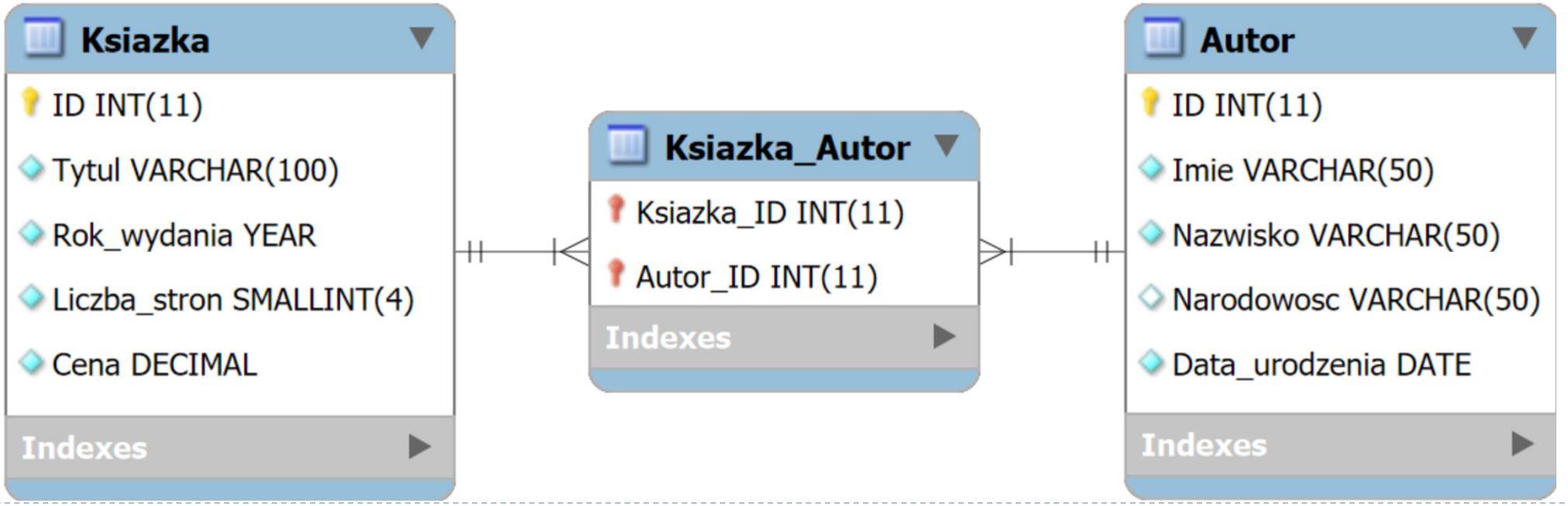
Pomijanie pierwszych wierszy podczas pobierania

- Jeśli chcemy pominąć kilka pierwszych wierszy, to można skorzystać z klauzuli `OFFSET`.
- Poniższe zapytanie pomija najdroższą książkę i wyświetla kolejne trzy (drugą, trzecią i czwartą z najdroższych):

```
SELECT * FROM Ksiazka
ORDER BY Cena DESC
LIMIT 3
OFFSET 1
```

Zadanie 4

- Za pomocą języka SQL wyświetlić dane zagranicznych autorów. Autorzy powinni być posortowani od najmłodszego do najstarszego.



Funkcje grupujące (agregujące)

- Jeśli chcemy obliczyć statystyki, takie jak wartość średnia, maksymalna, czy suma wszystkich wartości danej kolumny, to możemy skorzystać z funkcji grupujących.
- Poniższe zapytanie wylicza średnią cenę wszystkich książek nie krótszych niż 300 stron.

```
SELECT AVG(Cena) FROM Ksiazka
WHERE Liczba_stron >= 300
```

- Funkcja AVG oblicza średnią wszystkich wartości z podanej kolumny dla wierszy spełniających podany warunek dotyczący liczby stron.

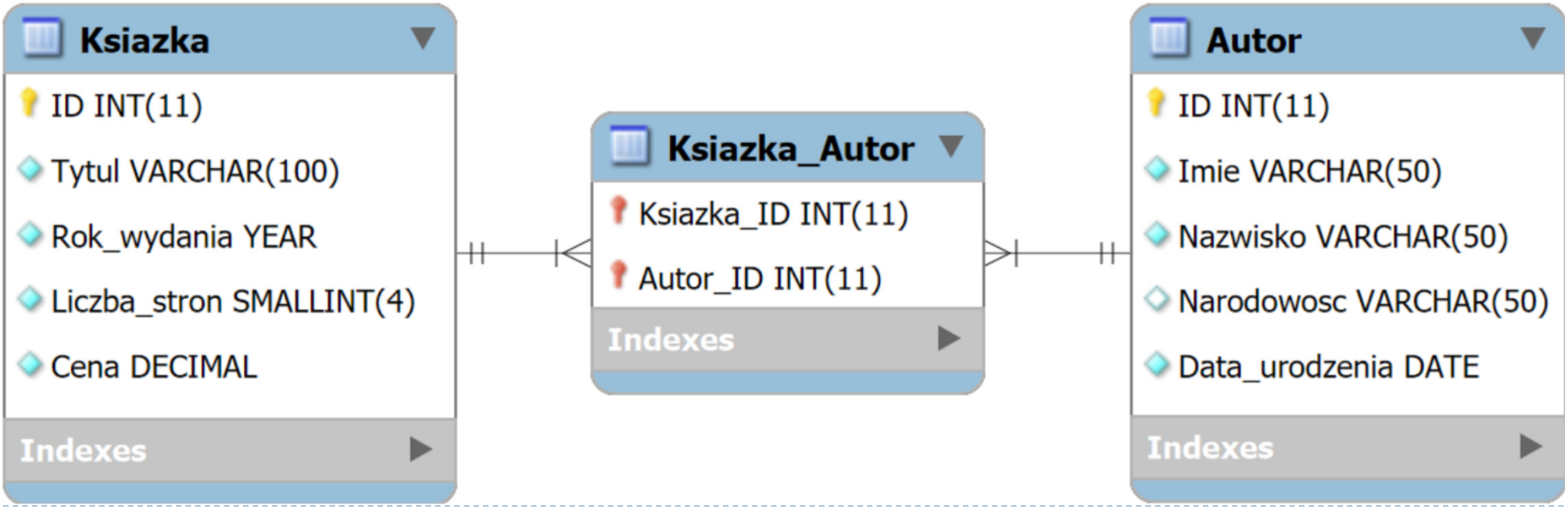
Funkcje grupujące (agregujące)

W bazach MySQL są dostępne następujące funkcje grupujące:

- `SUM`(kolumna) – zwraca sumę wartości z danej kolumny.
- `MAX`(kolumna) – zwraca maksymalną wartość z danej kolumny.
- `MIN`(kolumna) – zwraca minimalną wartość z danej kolumny.
- `AVG`(kolumna) – zwraca średnią wartość z danej kolumny.
- `VARIANCE`(kolumna) – zwraca wariancję wartości z danej kolumny.
- `STD`(kolumna) – zwraca odchylenie standardowe wartości z danej kolumny.
- `COUNT`(kolumna) – zwraca liczbę wierszy, które mają niepuste (inne niż NULL) wartości w kolumnie.
- `COUNT`(*) – zwraca liczbę wszystkich wierszy.
- `COUNT`(`DISTINCT` kolumna) – zwraca liczbę unikalnych wartości w kolumnie.

Zadanie 5

- Za pomocą języka SQL wyświetlić liczbę wszystkich autorów wprowadzonych do bazy danych.



Klauzula `GROUP BY`

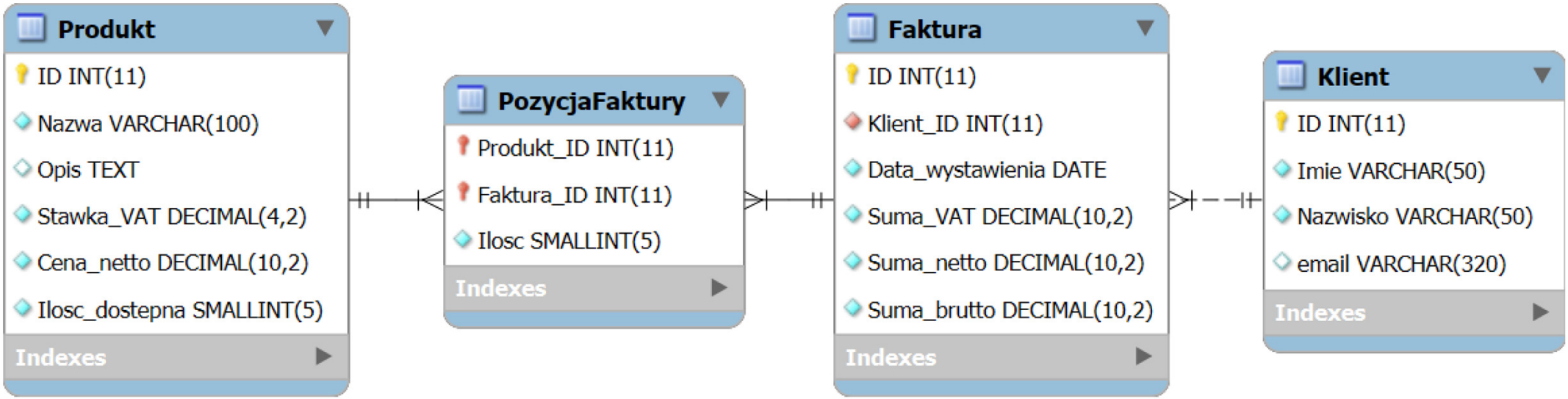
- Istnieje możliwość obliczenia statystyk dla określonych grup zamiast dla całych tabel.
- Przykładowo, za pomocą poniższego zapytania możemy obliczyć najniższą ocenę z każdego przedmiotu z osobna.

```
SELECT Przedmiot, MIN(Wartosc_oceny) FROM Ocena
GROUP BY Przedmiot
```

- Wybraliśmy kolumnę Przedmiot i średnią wszystkich ocen. Musimy jednak wskazać, że chcemy tą średnią pogrupować względem przedmiotu.
- Możemy to zrobić za pomocą klauzuli `GROUP BY`.
- Warto zapamiętać: Jeśli w jednym zapytaniu używamy funkcji grupującej i dodatkowo pobieramy dane z przynajmniej jednej kolumny (bez funkcji), to prawie zawsze trzeba będzie użyć klauzuli `GROUP BY`.

Zadanie 6

- Za pomocą języka SQL wyświetlić średnią cenę netto produktów o różnych stawkach VAT. Przy obliczaniu średniej należy pominąć produkty, których liczba sztuk w magazynie jest nie mniejsza niż 100.



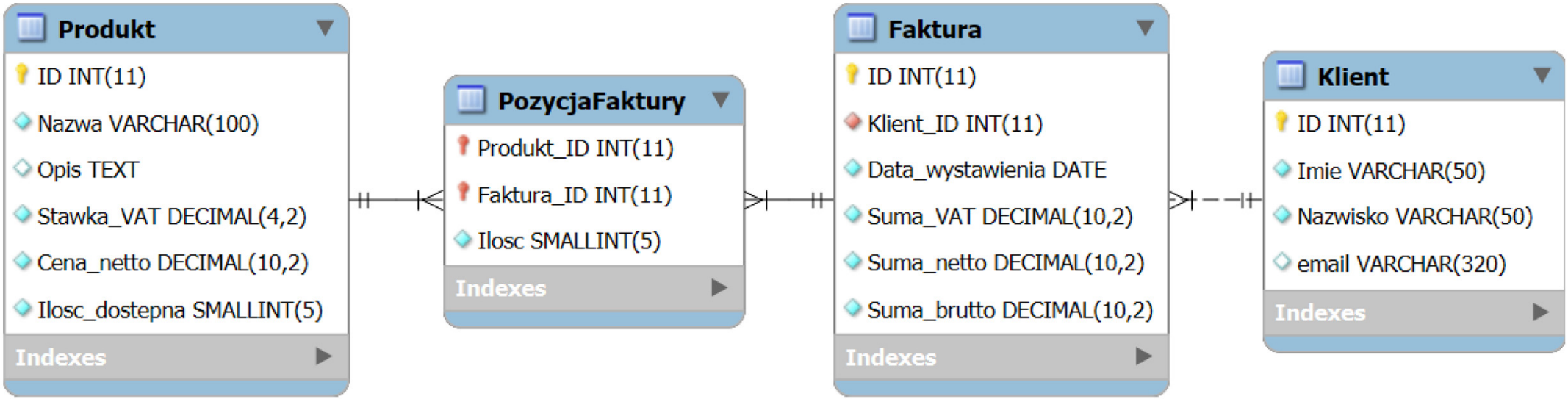
Określanie warunków dla grup utworzonych klauzulą `GROUP BY`

- Klauzula `HAVING` służy do określenia warunku dla grup po grupowaniu za pomocą `GROUP BY`.
- W przeciwieństwie do `WHERE` takie warunki są sprawdzane **po** grupowaniu, a **nie przed**.
- Poniższe zapytanie pobiera najniższą ocenę z każdego przedmiotu z osobna, ale tylko dla przedmiotów, z których ta najniższa ocena nie przekracza 3.0.

```
SELECT Przedmiot, MIN(Wartosc_oceny)
FROM Ocena
GROUP BY Przedmiot
HAVING MIN(Wartosc_oceny) <= 3.0
```

Pobieranie danych z wielu tabel

- Załóżmy, że chcemy wyświetlić daty wystawienia wszystkich faktur oraz imiona i nazwiska klientów, dla których zostały one wystawione.



Pobieranie danych z wielu tabel

- Załóżmy, że chcemy wyświetlić daty wystawienia wszystkich faktur oraz imiona i nazwiska klientów, dla których zostały one wystawione.
- Spróbujmy wymienić nazwy wszystkich atrybutów (po słowie `SELECT`) i nazwy wszystkich tabel, w których występują te atrybuty (po słowie `FROM`).

```
SELECT Imie, Nazwisko, Data_wystawienia FROM Klient, Faktura
```

- Powyższe zapytanie nie zwraca wyników takich, jakie byśmy oczekiwali. Wyświetlone zostają wszystkie możliwe kombinacje klientów i faktur, a nie faktury, które zostały wystawione dla konkretnych klientów.

Pobieranie danych z wielu tabel

- Aby wyświetlić dane z dwóch tabel zestawione ze sobą prawidłowo, należy wskazać w zapytaniu w jaki sposób tabele są ze sobą połączone.
- Można to zrobić poprzez utworzenie warunku, w którym przyrównujemy klucz obcy z jednej tabeli do odpowiadającego mu klucza głównego z drugiej tabeli:

```
SELECT Imie, Nazwisko, Data_wystawienia FROM Faktura, Klient
WHERE Faktura.Klient_ID = Klient.ID
```

- Innym sposobem jest skorzystanie z klauzuli `JOIN`:

```
SELECT Imie, Nazwisko, Data_wystawienia FROM Faktura
JOIN Klient ON Faktura.Klient_ID = Klient.ID
```

Pobieranie danych z wielu tabel – `RIGHT JOIN`

- Warto zwrócić uwagę, że po naszym zapytaniu wyświetleni zostaną jedynie klienci, którzy posiadają przynajmniej jedną fakturę (jeden z nich dwukrotnie, bo posiada dwie faktury).

Imie	Nazwisko	Data_wystawienia
Jan	Kowalski	2025-05-10
Anna	Nowak	2025-05-12
Piotr	Krakowski	2025-05-15
Jan	Kowalski	2025-05-16

- W bazie danych mamy klienta Adam Nowak, który prawdopodobnie nie dokonał jeszcze zakupu, a więc nie ma wystawionej faktury. Jego dane nie pojawiają się.

Pobieranie danych z wielu tabel – `RIGHT JOIN`

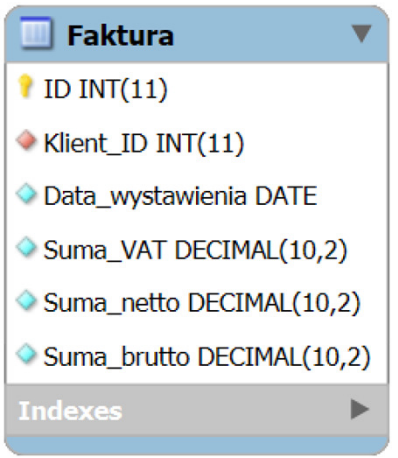
- Jeśli chcemy, aby Adam Nowak pojawił się pomimo braku faktury, możemy użyć klauzuli `RIGHT JOIN`.
- Powoduje ona, że dane z tabeli po prawej stronie od słowa `JOIN`, czyli w naszym przypadku tabeli Klient, wyświetlają się w całości, nawet jeśli dany wiersz nie ma odpowiadającego mu wiersza w drugiej tabeli (klient nie ma przypisanej faktury).

```
SELECT Imie, Nazwisko, Data_wystawienia FROM Faktura
RIGHT JOIN Klient ON Faktura.Klient_ID = Klient.ID
```

Imie	Nazwisko	Data_wystawienia
Jan	Kowalski	2025-05-10
Anna	Nowak	2025-05-12
Piotr	Krakowski	2025-05-15
Jan	Kowalski	2025-05-16
Adam	Nowak	*NULL*

Pobieranie danych z wielu tabel – `LEFT JOIN`

- Istnieją również inne rodzaje operacji łączenia.
- `LEFT JOIN` działa podobnie do `RIGHT JOIN`, ale wyświetla w pełni dane z tabeli po lewej stronie od słowa `JOIN`.
- Gdybyśmy mieli w naszej bazie danych faktury nie mające przypisanego klienta, to zostałyby one wyświetlone, jeśli użylibyśmy połączenia `LEFT JOIN`.
- Nie ma jednak możliwości, aby taka faktura istniała, ponieważ atrybut Klient_ID jest obowiązkowy (nie jest opcjonalny).
- Świadczy o tym wypełniony kwadrat przy nazwie atrybutu.

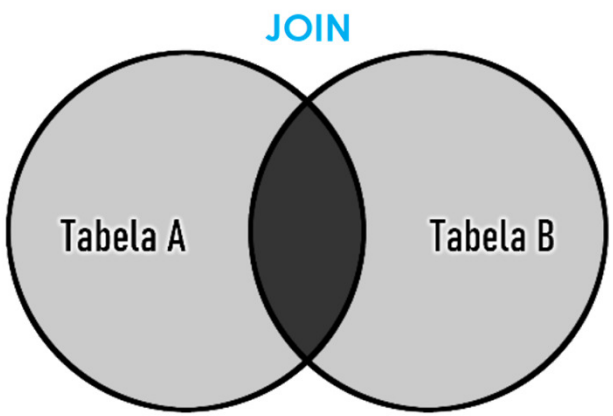


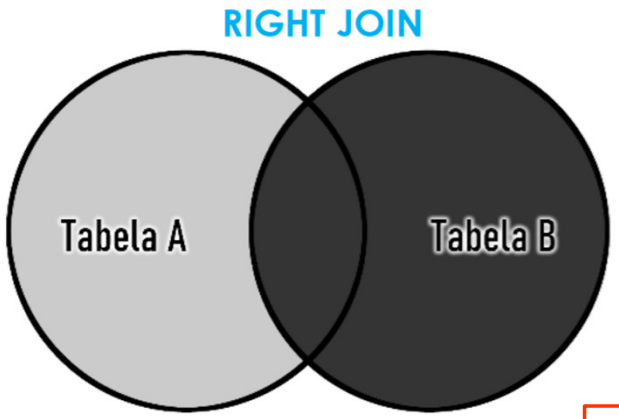
Pobieranie danych z wielu tabel – różne rodzaje operacji `JOIN`

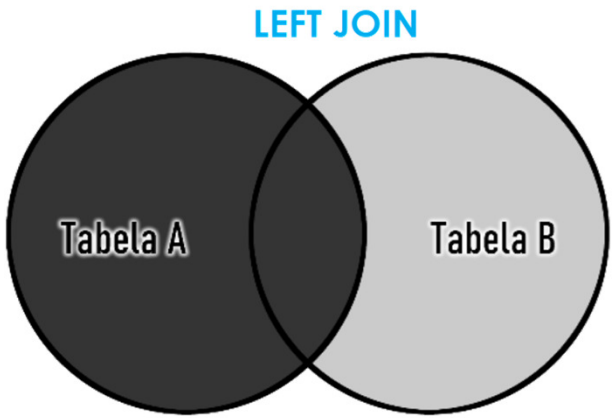
- Jest również dostępna operacja `FULL JOIN`.
- Wyświetla ona wszystkie dane z obu tabel, niezależnie od istnienia odpowiadających im wierszy.
- Niestety nie jest ona dostępna w bazach danych MySQL i SQLite. Występuje we wszystkich innych popularnych bazach danych.

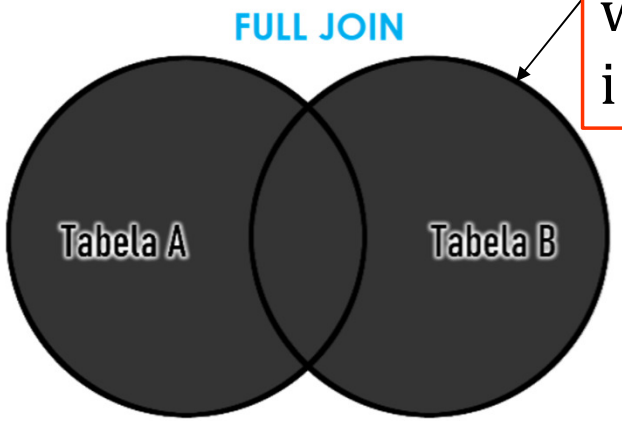
- Warto też wiedzieć, że podstawowy `JOIN` jest skrótową nazwą operacji `INNER JOIN`. Oznacza więc dokładnie to samo.
- `FULL JOIN` jest z kolei skrótową nazwą operacji `FULL OUTER JOIN`.

Pobieranie danych z wielu tabel – różne rodzaje operacji `JOIN` – wizualizacja









Niedostępna w MySQL i SQLite.

Pobieranie danych z wielu tabel + operator LIKE

- Załóżmy, że chcemy wyświetlić daty wystawienia wraz z imionami i nazwiskami klientów, ale tylko tych, których nazwiska zaczynają się na literę K.
- Możemy to wykonać dopisując do zapytania z poprzednich slajdów nowy warunek (połączony z poprzednim warunkiem operatorem logicznym AND), w którym zastosujemy operator LIKE.

```
SELECT Imie, Nazwisko, Data_wystawienia FROM Faktura
JOIN Klient ON Faktura.Klient_ID = Klient.ID
AND Nazwisko LIKE 'K%'
```

- Napis 'K%' oznacza wzorzec, w którym pierwszą literą jest K, a potem następuje dowolna ilość dowolnych znaków.

Pobieranie danych z wielu tabel + operator `LIKE`

- Oprócz symbolu **%**, istnieje jeszcze jeden symbol wieloznaczny. Jest nim **_**.
- W miejscu symbolu **_** może wystąpić dowolny znak, ale tylko jeden.
- Przykład 1: Do wzorca `'_os'` pasują słowa sos i kos, ale nie pasuje słowo kosmos.
- Przykład 2: Do wzorca `'%os'` pasują słowa sos, kos i kosmos.

Pobieranie danych z wielu tabel + aliasy

- Patrząc na tabelę uzyskaną za pomocą zapytania zwracającego dane z dwóch tabel możemy mieć wątpliwości czego dotyczą niektóre atrybuty.
- Przykładowo, osoba nie znająca struktury bazy danych widząc poniższą tabelę może nie wiedzieć czego dotyczy data wystawienia.

Imie	Nazwisko	Data_wystawienia
Jan	Kowalski	2025-05-10
Anna	Nowak	2025-05-12

- Możemy zmienić wyświetlaną nazwę atrybutu Data_wystawienia na nazwę, która jednoznacznie mówi, czego ten atrybut dotyczy, np. „data wyst. Faktury”.
- Można to zrobić za pomocą tzw. aliasów (słowo `AS` w języku SQL).

```
SELECT Imie, Nazwisko, Data_wystawienia AS 'Data wyst. faktury'
FROM Faktura
JOIN Klient ON Faktura.Klient_ID = Klient.ID
```

Pobieranie danych z wielu tabel + aliasy

```
SELECT Imie, Nazwisko, Data_wystawienia AS 'Data wyst. faktury'
FROM Faktura
JOIN Klient ON Faktura.Klient_ID = Klient.ID
```

- Po wykonaniu powyższego zapytania tabela będzie wyglądała następująco:

Imie	Nazwisko	Data wyst. faktury
Jan	Kowalski	2025-05-10
Anna	Nowak	2025-05-12

Pobieranie danych z wielu tabel – dwuznaczność

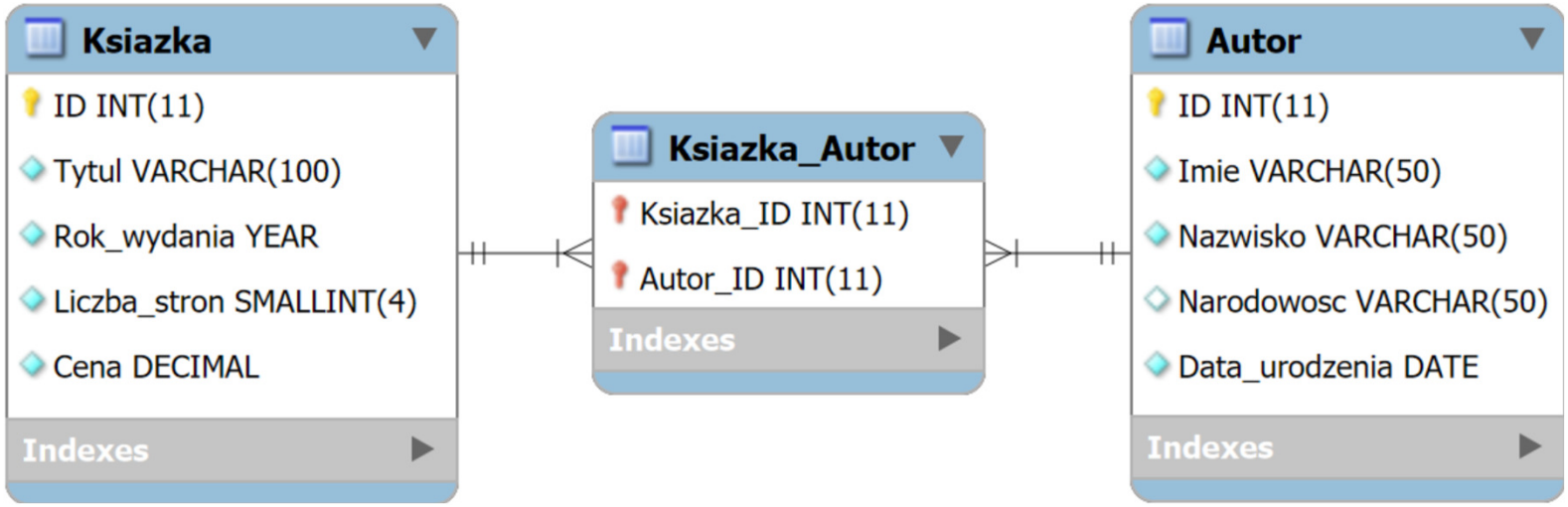
- Należy zwrócić uwagę, że może się zdarzyć sytuacja, w której występują atrybuty o tych samych nazwach w wielu tabelach.
- Jeśli chcielibyśmy wybrać któryś z tych atrybutów, to interpreter SQL nie wiedziałby, który dokładnie mamy na myśli. Pojawiłaby się dwuznaczność.
- Aby się upewnić, że takiej dwuznaczności nie będzie, należy nazwy atrybutów poprzedzać nazwami tabel, w których występują i znakiem kropki.

```
SELECT Klient.Imie, Klient.Nazwisko, Faktura.Data_wystawienia
FROM Faktura
JOIN Klient ON Faktura.Klient_ID = Klient.ID
```

Zadanie 7

- Za pomocą języka SQL wyświetlić tytuły wszystkich książek, których przynajmniej jeden autor jest Polakiem. Obok tytułów należy wyświetlić również imię i nazwisko każdego autora.

Podpowiedź: Należy wskazać dwa połączenia tabel, między pierwszą a drugą i między drugą a trzecią (dwa joiny).



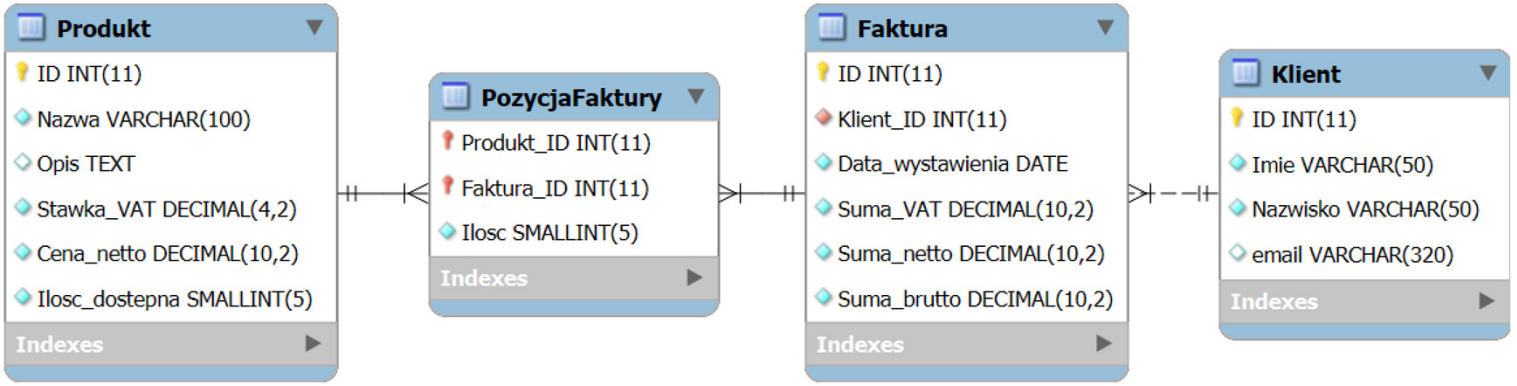
Podzapytania

- W języku SQL istnieje możliwość wykonania zapytania w zapytaniu. Nazywamy to podzapytaniem.

- Po co stosuje się podzapytania? Rozważmy następujący przykład (kolejny slajd).

Podzapytania

- Chcemy wyświetlić numer ID klientów, którzy kupili najdroższy produkt w bazie danych.
- Innymi słowy, chcemy pobrać numer klienta, który widnieje na fakturze, na której pozycji jest produkt, którego cena jest najwyższą ceną ze wszystkich produktów.



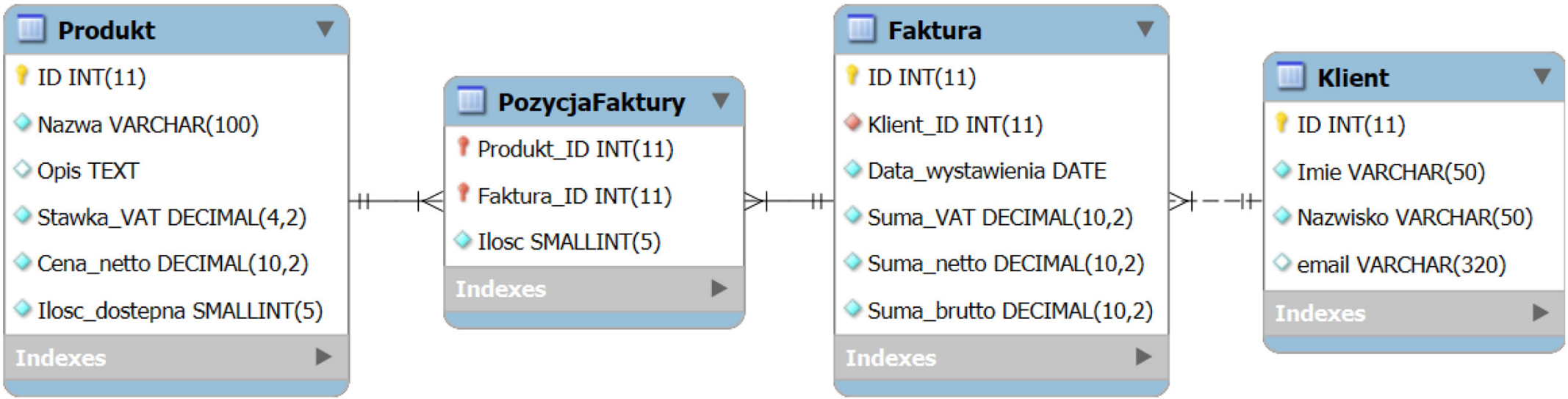

```
SELECT Klient_ID
FROM Faktura
JOIN Pozycjafaktury ON Pozycjafaktury.Faktura_ID = Faktura.ID
JOIN Produkt ON Pozycjafaktury.Produkt_ID = Produkt.ID
WHERE Produkt.Cena_netto = (SELECT MAX(Cena_netto) FROM Produkt)
```

podzapytanie wybierające najwyższą cenę spośród produktów

Zadanie 8

- Zmodyfikuj zapytanie z poprzedniego slajdu tak, aby wybierało imię i nazwisko klienta zamiast jego numeru. Czyli chcemy wyświetlić imiona i nazwiska klientów, którzy kupili najdroższy produkt w bazie danych.

Podpowiedź: Należy wykonać jeszcze jeden `JOIN`, tzn. dołączyć tabelę Klient.



Wstawianie danych – polecenie INSERT

- Polecenie INSERT służy do wstawiania nowych wierszy do bazy danych.
- Przykładowo, poniższe polecenie dodaje do bazy danych nowego autora książek, Adama Mickiewicza:

```
INSERT INTO Autor (ID, Imie, Nazwisko, Narodowosc, Data_urodzenia)
VALUES (8, 'Adam', 'Mickiewicz', NULL, '1798-12-24')
```

- Należy zauważyć, że dodając autora nie podano jego narodowości (podano wartość NULL).
- Jest to atrybut opcjonalny, więc można nie podać jego wartości i mimo to polecenie wykona się bezproblemowo.

Wstawianie danych – polecenie INSERT

- Jeśli nie zamierzamy podawać wartości pewnych atrybutów, to możemy polecenie uprościć.
- Zamiast wpisywać NULL można pominąć nazwę i wartość atrybutu:

```
INSERT INTO Autor (ID, Imie, Nazwisko, Data_urodzenia)
VALUES (8, 'Adam', 'Mickiewicz', '1798-12-24')
```

- Należy jednak pamiętać, że jeśli atrybut, który pominiemy będzie oznaczony jako obowiązkowy, to polecenie nie wykona się (zakończy się błędem).
- Warto również pamiętać, że wartość klucza głównego (w naszym przypadku atrybut ID) musi być unikalna. Do tej pory w bazie było siedmiu autorów (numery 1-7), a więc kolejny powinien mieć numer 8.

Wstawianie danych – polecenie INSERT – automatyczna inkrementacja kluczy głównych

- Zauważmy, że za każdym razem, kiedy dodajemy nowy wiersz do tabeli, musimy sprawdzić, jaka jest obecnie największa wartość klucza głównego, dodać do niej 1 i dopiero wtedy wykonać polecenie INSERT.
- Jest to dość czasochłonne. Czy nie da się tego procesu zautomatyzować?
- Da się. Rozwiązaniem jest oznaczenie atrybutu będącego kluczem obcym jako AUTO_INCREMENT.
- W phpMyAdmin służy do tego kolumna Extra i opcja AUTO_INCREMENT (A_I) w widoku struktury bazy danych.
- Tak oznaczony klucz główny będzie sam generował dla siebie wartość przy dodaniu nowego wiersza. Nie trzeba go więc podawać:

```
INSERT INTO Autor (Imie, Nazwisko, Data_urodzenia)
VALUES ('Adam', 'Mickiewicz', '1798-12-24')
```

Wstawianie danych – polecenie INSERT – funkcja NOW

- Funkcja NOW (bezparametrowa) służy do zwrócenia aktualnej daty i czasu.
- Jeśli podczas dodawania nowej faktury do bazy danych chcielibyśmy, żeby data jej wystawienia była wpisywana automatycznie jako data aktualna, to można to zrobić następująco:

```
INSERT INTO Faktura (ID, Klient_ID,
Data_wystawienia, Suma_VAT, Suma_netto, Suma_brutto)
VALUES (9, 2, NOW(), 23.00, 100.00, 123.00)
```

- Funkcję NOW można zastosować dla atrybutów o typach: DATE, DATETIME, TIMESTAMP oraz YEAR.
- Jeśli chcemy pobrać sam czas do kolumny typu TIME, to możemy użyć funkcji CURTIME lub TIME(NOW()).

Aktualizowanie danych – polecenie UPDATE

- Polecenie UPDATE służy do aktualizowania danych w tabelach bazy danych.
- Przykładowo, poniższe polecenie zmienia nazwisko studenta o numerze indeksu 123463 na Niewiadomski:

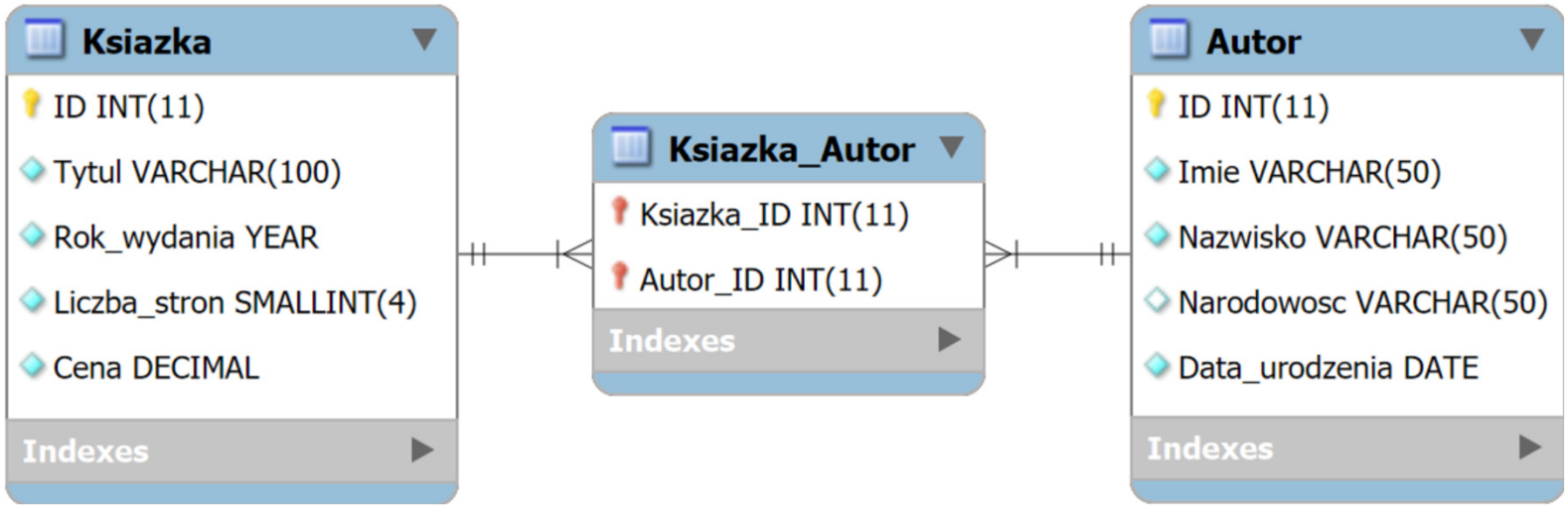
```
UPDATE Student
SET Nazwisko = 'Niewiadomski'
WHERE Numer_indeksu = 123463
```

- Następujące polecenie zwiększa o 50 dostępną liczbę sztuk kostek masła (produkt o ID równym 4).

```
UPDATE Produkt
SET Ilosc_dostepna = Ilosc_dostepna + 50
WHERE ID = 4
```

Zadanie 9

- Za pomocą języka SQL obniżyć o 5% ceny wszystkich książek wydanych przed rokiem 2022, których obecna cena jest nie mniejsza niż 50 zł.



Usuwanie danych – polecenie DELETE

- Polecenie DELETE służy do usuwania wierszy z bazy danych.
- Przykładowo, poniższe polecenie usuwa ocenę z przedmiotu Teoria obwodów studenta o numerze indeksu 123459:

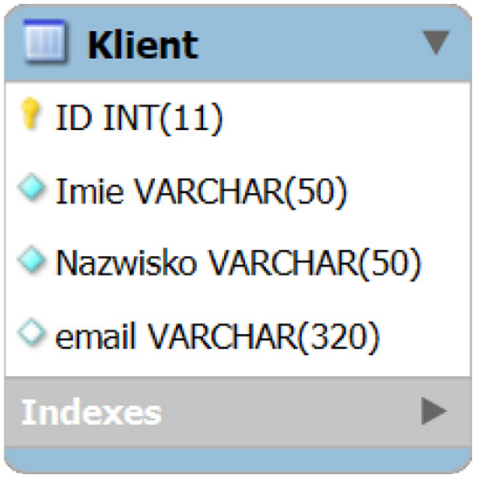
```
DELETE FROM Ocena
WHERE Przedmiot = 'Teoria obwodów' AND Numer_indeksu = 123459
```

- Student może mieć tylko jedną ocenę końcową z danego przedmiotu, więc mamy pewność, że usunięta zostanie jedna konkretna ocena.
- Jeśli jednak znamy wartość klucza głównego ID_oceny, to możemy go wpisać, aby uprościć warunek polecenia:

```
DELETE FROM Ocena
WHERE ID_oceny = 8
```

Zadanie 10

- Za pomocą języka SQL usunąć klientów sklepu, którzy nie maja wpisanego adresu e-mail lub ma on niepoprawny format.
- Za niewpisany adres e-mail uznajemy:
  - brak wartości (NULL);
  - pusty napis `''`.
- Za niepoprawny format adresu e-mail uznajemy:
  - adres, który nie zawiera znaków `'@'` oraz `'.'` wpisanych w odpowiedniej kolejności (należy użyć operatora `LIKE`);
  - adres, który ma mniej niż 6 znaków (do określenia ilości znaków tekstu należy użyć funkcji `CHAR_LENGTH`).
- Przed testowaniem zapytania należy wprowadzić do bazy danych czterech klientów, którzy nie mają wpisanego adresu e-mail lub ma on niepoprawny format (zgodnie z każdym podpunktem).



Usuwanie danych powiązanych z innymi tabelami

- Czy możemy usunąć z bazy danych studenta, który ma już wystawioną jakąś ocenę, czyli jest powiązany z inną tabelą (w tabeli Ocena jest wiersz z numerem indeksu tego studenta)?
- Domyślnie nie jest to możliwe. Przy próbie usunięcia pojawi się błąd.
- To zachowanie można jednak zmienić. Służy do tego reguła kluczy obcych `ON DELETE`, która może przyjąć następujące wartości:
  - `RESTRICT` lub `NO ACTION` (domyślnie) – nie można usunąć studenta.
  - `SET NULL` – student zostanie usunięty, a numer indeksu przy jego ocenach przyjmie wartość NULL (brak wartości).
  - `CASCADE` – student zostanie usunięty; zostaną również usunięte wszystkie jego oceny (wszystkie odpowiadające mu wiersze w innych tabelach).
- W phpMyAdmin reguły `ON DELETE` można ustawić w zakładce: Structure -> Widok relacyjny

Zmiana klucza głównego tabeli powiązanej z innymi tabelami

- Istnieje podobna reguła do `ON DELETE`, która dotyczy aktualizowania danych w tabeli. Nazywa się `ON UPDATE`.
- W przypadku bazy danych „indeks_studencki” dotyczy ona możliwości zmiany numeru indeksu (wartości klucza głównego) studenta, który ma już wystawioną jakąś ocenę.
- Reguła `ON UPDATE` może przyjąć analogiczne wartości do `ON DELETE`:
  - `RESTRICT` lub `NO ACTION` (domyślnie) – nie można zmienić klucza studenta.
  - `SET NULL` – klucz studenta zostanie zmieniony, ale numery indeksu przy jego ocenach (klucz obcy tabeli Ocena) przyjmą wartość NULL (brak wartości).
  - `CASCADE` – klucz zostanie zmieniony; zostaną również zmienione numery indeksu przy jego ocenach (wszystkie odpowiadające mu wiersze w innych tabelach).
- W phpMyAdmin reguły `ON UPDATE` można ustawić w zakładce: Structure -> Widok relacyjny

Zagrożenia związane z poleceniami `DELETE` i `UPDATE`

- UWAGA: Z poleceniami `DELETE` i `UPDATE` trzeba uważać.
- Jeśli nie wpiszemy warunku (lub jeśli wpiszemy warunek, który będzie zawsze spełniony), to usunięte/nadpisane zostaną wszystkie wiersze tabeli.
- Takiej operacji nie można cofnąć (chyba że wykonujemy polecenia w ramach tzw. transakcji).
- Nie wszystkie systemy bazodanowe pytają o potwierdzenie wykonania operacji usunięcia/nadpisania. Niektóre usuwają/nadpisują natychmiastowo.

Wykonywanie wielu poleceń na raz

- Jeśli chcemy wykonać wiele poleceń typu `INSERT`, `UPDATE`, `DELETE` za jednym razem, to możemy oddzielić je średnikami, tzn. wstawić średnik na końcu każdego polecenia z wyjątkiem ostatniego.
- Przykładowo, powyższy kod usuwa ocenę o ID = 8 oraz zmienia ocenę o ID = 7 na 5.0. Najpierw wykona się pierwsze polecenie, potem drugie.

```
DELETE FROM Ocena
WHERE ID_oceny = 8;

UPDATE Ocena
SET Wartosc_oceny = 5.0
WHERE ID_oceny = 7
```

- Jeśli chcemy wykonać tylko jedno polecenie, średnik nie jest konieczny (ale jest dopuszczalny).

Komentarze

- Komentarze jednolinijkowe możemy wstawiać w języku SQL przy użyciu podwójnego znaku myślnika `--`.
- Komentarze wielolinijkowe można wstawić za pomocą kombinacji `/*` na początku i `*/` na końcu komentarza.

```
-- Usunięcie oceny
DELETE FROM Ocena
WHERE ID_oceny = 8;

/* Aktualizacja oceny:
Zamiana oceny o ID 7 na 5.0 */
UPDATE Ocena
SET Wartosc_oceny = 5.0
WHERE ID_oceny = 7
```

„Zakomentowanie” poszczególnych instrukcji

- Komentarz wielolinijkowy, podobnie jak w przypadku języków programowania, możemy wykorzystać do „zakomentowania” poszczególnych instrukcji.
- Takie instrukcje będą traktowane jako komentarz, a więc nie będą wykonywane. Można je potem jednak łatwo przywrócić usuwając znaki początku i końca komentarza.

```
--Usunięcie oceny
DELETE FROM Ocena
WHERE ID_oceny = 8;

/*
UPDATE Ocena
SET Wartosc_oceny = 5.0
WHERE ID_oceny = 7
*/
```

Kopie bazy danych

- Pracując z bazami danych należy pamiętać o regularnym tworzeniu ich kopii, aby uniknąć sytuacji, w których duża ilość ważnych danych zostanie usunięta (np. w skutek awarii dysku lub pomyłkowego usunięcia danych poleceniem `DELETE`).

- W oprogramowaniu phpMyAdmin kopię można utworzyć klikając w zakładkę Export.

- Odtworzenie bazy danych z kopii można wykonać za pomocą zakładki Import.

Kopie baz danych z niniejszej prezentacji

- Kopie większości baz danych omawianych w ramach tej prezentacji można pobrać ze strony domowej prowadzącego, z materiałów z przedmiotu Informatyka (EE-DI-1).

- Są one w pliku archiwum o nazwie „KOPIE_BAZ_DANYCH.ZIP”.

Przechowywanie dużych ilości danych (np. plików)

- Jeśli chcemy w bazie danych przechowywać pliki, np. obrazy, wideo, audio, PDFy, to są na to dwa sposoby:
- Sposób I (zalecany): Przechowywać pliki poza bazą, np. na zewnętrznym serwerze, w chmurze. W takim wypadku w bazie danych przechowuje się ich adresy URL (jako napisy, np. VARCHAR).
- Sposób II: Można przechowywać pliki bezpośrednio w bazie danych. W MySQL istnieją specjalne typy do przechowywania dużych danych binarnych:
  - TINYBLOB – do 255 B,
  - BLOB – do 65 KB,
  - MEDIUMBLOB – do 16 MB,
  - LONGBLOB – do 4 GB.

Ładowanie plików do kolumn typu BLOB (i pokrewnych)

- Możemy załadować dane do kolumny typu BLOB, np. z dysku naszego komputera, przy użyciu funkcji `LOAD_FILE`:

```
INSERT INTO Obrazy (Nazwa_obrazu, dane)
VALUES ('zdjecie01', LOAD_FILE('C:/album/zdjecie01.jpg'));
```

Nadawanie uprawnień użytkownikom

- W bazach danych istnieje mechanizm kontroli dostępu, który określa jakie operacje dany użytkownik może wykonywać oraz na jakich obiektach (bazy, tabele, kolumny).
- Ogólna składania nadawania uprawnień w języku SQL jest następująca:

```
GRANT uprawnienia (kolumna1, kolumna2)
ON baza.tabela
TO 'użytkownik'@'host'
```

Nadawanie uprawnień użytkownikom

- Jeśli chcemy nadać uprawnienia do pobierania i wstawiania danych o produktach z bazy danych „Sklep” użytkownikowi o loginie „jan”, który łączy się z bazą lokalnie (z naszego komputera), to możemy to zrobić tak:

```
GRANT SELECT, INSERT
ON Sklep.Produkt
TO 'jan'@'localhost'
```

- Jeśli chcemy nadać uprawnienia jedynie do pobierania nazwy i opisu produktu (konkretnych kolumn), to robimy to następująco:

```
GRANT SELECT (Nazwa, Opis)
ON Sklep.Produkt
TO 'jan'@'localhost'
```

Nadawanie uprawnień użytkownikom

- Jeśli chcemy nadać uprawnienia pobierania i wstawiania danych do całej bazy sklepu, to robimy to tak:

```
GRANT SELECT, INSERT
ON Sklep.*
TO 'jan'@'localhost'
```

- Poniższe zapytanie pozwala nadać uprawnienia do wszystkich baz danych w naszym serwerze:

```
GRANT SELECT, INSERT
ON *.*
TO 'jan'@'localhost'
```

Nadawanie uprawnień użytkownikom z możliwością ich przekazania

- Jeśli chcemy nadać użytkownikowi uprawnienia oraz możliwość przekazania ich innym użytkownikom, to na końcu polecenia należy dodać `WITH GRANT OPTION`. Przykład:

```
GRANT SELECT, INSERT
ON Sklep.Produkt
TO 'jan'@'localhost'
WITH GRANT OPTION
```

Uprawnienia

Najczęściej przydzielanymi uprawnieniami są:

- `SELECT` – pobieranie danych,
- `INSERT` – wstawianie danych,
- `UPDATE` – aktualizacja danych,
- `DELETE` – usuwanie danych,
- `CREATE` – tworzenie bazy danych lub tabeli,
- `DROP` – usuwanie bazy danych lub tabeli,
- `ALTER` – modyfikacja struktury tabeli,
- `ALL PRIVELEGES` – pełne uprawnienia,
- `GRANT OPTION` – możliwość nadawania własnych uprawnień innym użytkownikom.

Format określania użytkowników

- Co dokładnie oznacza zapis `'użytkownik'@'host'`, czyli np. `'jan'@'localhost'`?
- Nie jest to adres e-mail pomimo symbolu @.
- Pierwszy człon oznacza login użytkownika, a drugi oznacza adres IP użytkownika, z którego łączy się on z bazą danych lub:
  - `'localhost'` – logowanie tylko z lokalnego komputera (tego, na którym działa serwer bazy danych;
  - `'%'` – dowolny host (mało bezpieczne);
  - `'%.firma.pl'` – domena (symbol % zastępuje dowolną liczbę dowolnych znaków, np. `pc1.firma.pl`, `db01.firma.pl`).

Odbieranie uprawnień

- Poniższe zapytanie odbiera użytkownikowi uprawnienia modyfikowania i usuwania danych z tabeli Ocena bazy Indeks_studencki:

```
REVOKE UPDATE, DELETE
ON Indeks_studencki.Ocena
FROM 'jan'@'localhost'
```

- Tak możemy odebrać użytkownikowi wszystkie uprawnienia dla bazy danych Indeks_studencki:

```
REVOKE ALL PRIVILEGES
ON Indeks_studencki.*
FROM 'jan'@'localhost'
```

Pokazywanie uprawnień

- Możemy również sprawdzić jakie uprawnienia ma przydzielone użytkownik:

```
SHOW GRANTS FOR 'jan'@'localhost'
```

Przydzielanie uprawnień rolom

- Uprawnienia możemy przydzielać pojedynczym użytkownikom, ale jest to często niewygodne i czasochłonne.
- Załóżmy, że mamy na uczelni 1000 nauczycieli akademickich i każdemu z nich musimy przydzielić uprawnienia.
- Później decyzją władz uczelni potrzebujemy zmienić uprawnienia wszystkich nauczycieli akademickich (np. nadać nowe, niektóre odebrać).
- Musielibyśmy zrobić wpisać 1000 nazw nauczycieli po przecinku w poleceniu `GRANT`.
- Można jednak utworzyć rolę „nauczyciel_akademicki”, dodać do niej wszystkich użytkowników pracujących jako nauczyciel, a potem przydzielić uprawnienia dla całej roli.
- W razie potrzeby zmiany uprawnień wystarczy zmienić je dla całej roli, a nie dla wszystkich użytkowników (nie trzeba podawać 1000 loginów).

Przydzielanie uprawnień rolom

- Rolę możemy utworzyć poleceniem:

```
CREATE ROLE 'nauczyciel_akademicki'
```

- Tak możemy przypisać rolę użytkownikom:

```
GRANT 'nauczyciel_akademicki'
TO 'jan'@'localhost', 'adam'@'localhost'
```

- Musimy ją później aktywować (użytkownik może być w kilku rolach, ale nie musi mieć wszystkich aktywnych):

```
SET DEFAULT ROLE 'nauczyciel_akademicki'
TO 'jan'@'localhost', 'adam'@'localhost'
```

- W ten sposób możemy nadać uprawnienia całej roli:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON Indeks_studencki.Ocena
TO 'nauczyciel_akademicki'
```