Bazy Danych SQL

na przykładzie MySQL

Przemysław Studziński

Plan prezentacji

1. Podstawowe pojęcia.
2. Relacje.
3. Język SQL.
4. MySQL.
5. Typy danych.
6. Podstawowe operacje.
7. Manipulowanie danymi.
8. Pobieranie danych.
9. Klucze podstawowe i obce.
10. Projektowanie bazy danych.
11. Złączenia tabel.
12. Indeksy.
13. Transakcje.

Podstawowe pojęcia

Baza danych

Baza danych jest zorganizowanym zbiorem informacji (danych) według określonych reguł. Przechowywana jest zazwyczaj w formie elektronicznej.

Tradycyjne bazy danych

Przykłady:

- Papierowa kartoteki: np. pacjentów w przychodni, w bibliotece,
- Dziennik lekcyjny,
- Książka telefoniczna.

Wady:

- Zgromadzone dane zajmują dużą powierzchnię,
- Odnalezienie danych może zajmować dużo czasu,
- Utrudniona analiza zgromadzonych danych,
- Brak możliwości łatwego tworzenia kopii zapasowych.

Tabelę w bazie danych można „*porównać*” także do arkusza Excel, który może posłużyć jako tabela. ☺

Komputerowe bazy danych

Właściwości (zalety):

- Szybkie wyszukiwanie informacji.
- Przechowywanie dużej ilości danych na małej powierzchni.
- Łatwe analiza zgromadzonych danych.
- Łatwy dostęp i możliwość dostępu zdalnego.
- Łatwe tworzenie kopii zapasowych.
- Zachowanie spójności danych.

Rodzaje baz danych

Bazy proste:

- kartotekowe;
- hierarchiczne.

Bazy złożone:

- relacyjne;
- obiektowe;
- relacyjno-obiektowe;
- nierelacyjne (NoSQL).

Nierelacyjne bazy danych (NoSQL)

Pod pojęciem bazy **nierelacyjnej** (NoSQL database) najczęściej rozumie się przechowywanie danych w formie listy par obiektów klucz-wartość, w których nie występują powiązania relacyjne między przechowywanymi obiektami.

Bazy NoSQL będą omawiane w dalszej części kursu.

DBMS

System zarządzania bazą danych (DBMS — Data Base Management System) - zarządza danymi w bazie, umożliwia ich przetwarzanie (wykonuje operacje na danych). Zwyczajowo nazywany jest silnikiem bazy danych .

RDBMS

System zarządzania relacyjną bazą danych (RDBMS - ang. Relational Database Management System). Poniżej przykłady relacyjnych systemów bazodanowych (wszystkie obsługują język SQL, z drobnymi różnicami nazwach).

Relacyjna baza danych

- Zbiór danych przechowywanych w tabelach połączonych **relacjami**.
- Schematy baz danych – zgrupowane relacje.

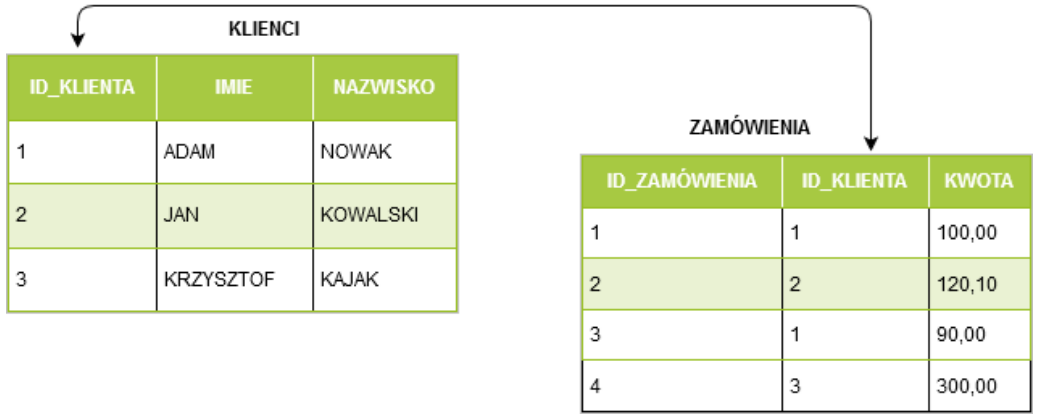
Tabela

Dwuwymiarowa struktura zbudowana z wierszy i kolumn zawierająca dane na określony temat, np. dane klientach, uczniach, lekcjach itp.

Relacje

Relacja

Logiczne połączenie pomiędzy między tabelami w relacyjnej bazie danych.

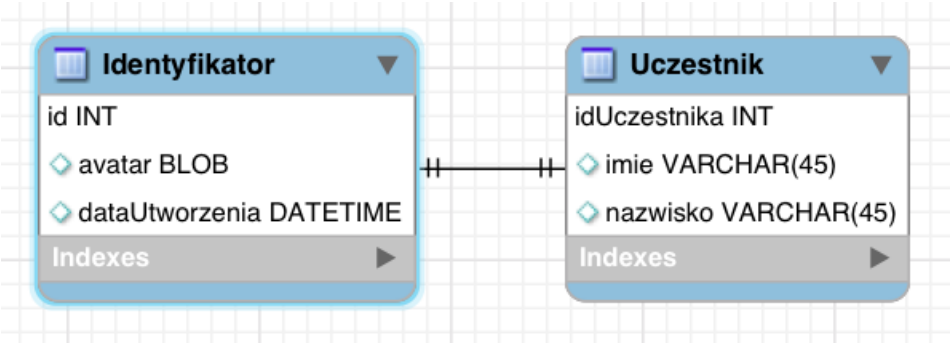


Typy relacji

- Relacja jeden do jednego
- Wiele do jednego
- Wiele do wielu

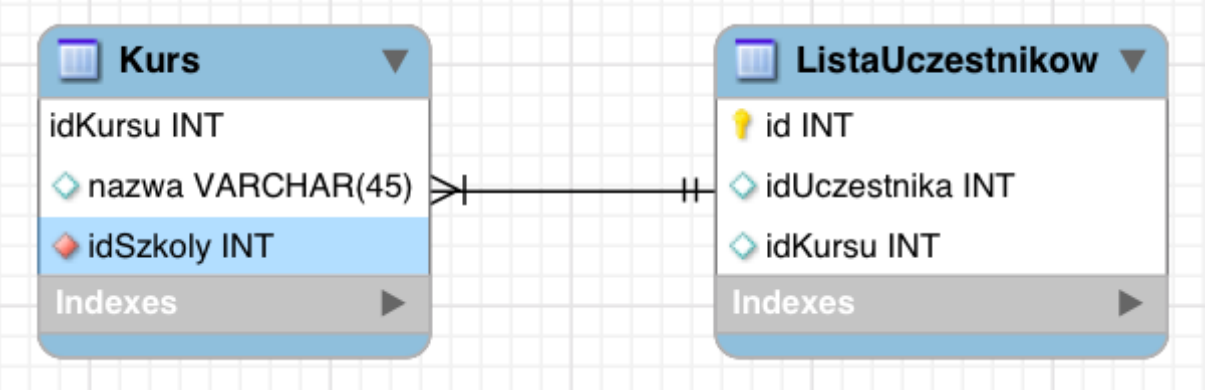
Relacja jeden do jednego

Relacja *jeden-do-jeden* oznacza, iż jeden rekord w pierwszej tabeli odpowiada dokładnie jednemu rekordowi w tabeli drugiej.



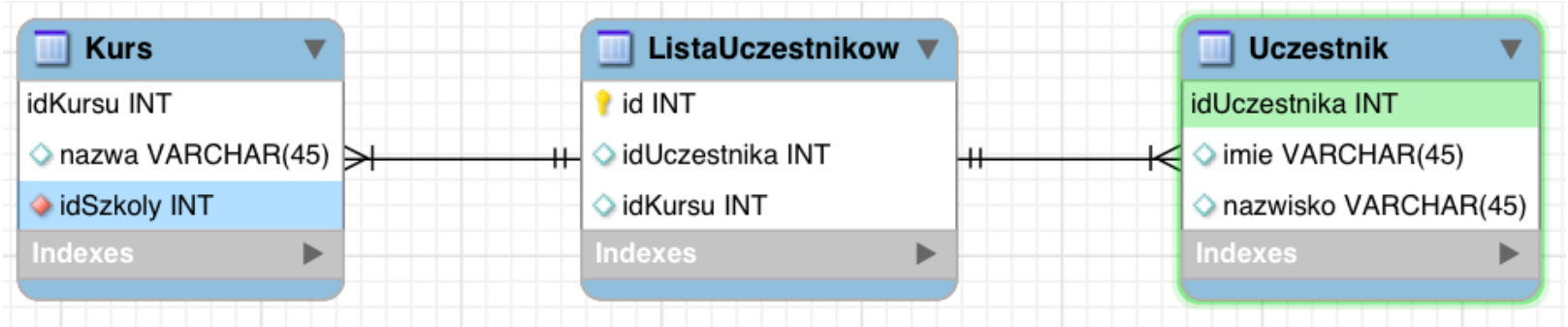
Relacja jeden do wielu

Relacja *jeden-do-wielu* oznacza, iż jeden rekord w pierwszej tabeli odpowiada wielu rekordom w tabeli drugiej.



Relacja wiele do wielu

Relacja *wiele-do-wielu* oznacza, że kilka rekordów z pierwszej tabeli odpowiada wielu rekordom z tabeli drugiej. Tworzona za pomocą nowej pomocniczej tabeli.



Język SQL

SQL

SQL – strukturalny język zapytań (ang. Structured Query Language). Jest to podstawowy język do komunikacji z relacyjnymi bazami danych. Pozwala na przechowywanie, manipulowanie i pobieranie danych.

Podzbiór języka SQL	Instrukcje
SQL DQL (ang. *Data Query Language*)	`SELECT`
SQL DML (*ang. Data Manipulation Language*)	`INSERT/UPDATE/DELETE`
SQL DDL (*ang. Data Definition Language*)	`CREATE/DROP/ALTER`
SQL DCL (*ang. Data Control Language*)	`GRANT/REVOKE/DENY`

Co możemy robić przy użyciu języka SQL

- wykonywać zapytania na bazie danych;
- pobierać dane z bazy danych;
- wstawiać rekordy do bazy danych;
- aktualizować rekordy w bazie danych;
- usuwać rekordy z bazy danych;
- tworzyć nowe bazy danych;
- tworzyć nowe tabele w bazie danych;
- tworzyć procedury przechowywane w bazie danych;
- tworzyć widoki w bazie danych;
- ustawiać uprawnienia do tabel, procedur, widoków.

Standard SQL

SQL jest standardem ANSI (*American National Standards Institute*), istnieją różne wersje języka SQL. Jednakże, aby być zgodnym ze standardem ANSI, wszystkie one wspierają co najmniej najważniejsze polecenia (takie jak SELECT, UPDATE, DELETE, INSERT, WHERE) w podobny sposób.

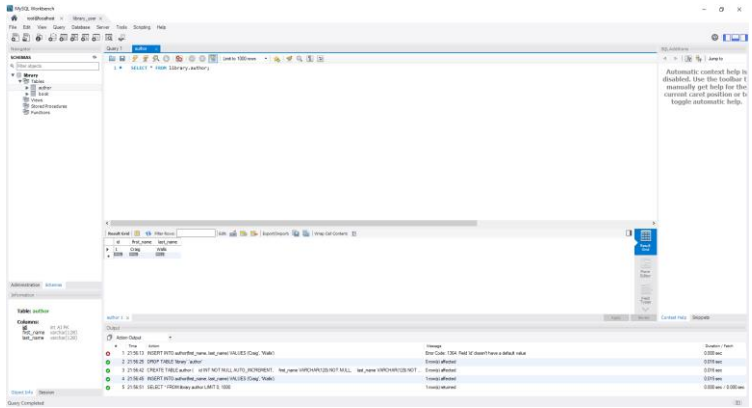
MySQL

MySQL

MySQL – baza danych rozwijany obecnie przez firmę *Oracle*, dostępna na licencji *GPL* i komercyjnej.

MySQL Workbench

MySQL Workbench - narzędzie graficzne do zarządzania i projektowania baz danych **MySQL**.
Alternatywą dla graficznego rozwiązania jest np. **MySQL Shell** – obsługujący polecenia z linii komend.



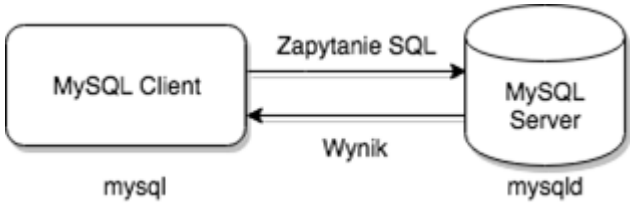
Instalacja

1. Przejdź na stronę: www.mysql.com.
2. Przejdź do działu *Downloads*.
3. *Wybierz opcję Community (GPL) Downloads.*
4. *Zainstaluj narzędzia:*
    - MySQL Community Server
    - MySQL Workbench

Szczegółowa instrukcja instalacji w dokumencie SDA: *Instrukcja instalacji oprogramowania*.

Architektura Kclient-Serwer

- Baza danych wraz z systemem zarządzania znajduje się na komputerze zwanym ***serwerem bazy danych***.
- Serwer baz danych odpowiada za zarządzanie danymi, operacjach na danych, a także za ich udostępnianie.
- Z serwerem komunikują się komputery użytkowników zwane ***klientami***.
- Na kursie klientem można nazwać oprogramowanie MySQL Workbench. W późniejszym etapie kursu będzie to aplikacja napisana przez Was w Javie.



Typy danych

Typy danych

- Numeryczne (np. TINYINT, INT, FLOAT, DECIMAL)
- Data i czas (np. DATE, DATETIME, TIMESTAMP)
- Łańcuchowy (np. CHAR, VARCHAR, TEXT, BINARY, BLOB)

MySQL-Data-Types.pdf

https://dev.mysql.com/doc/refman/8.0/en/data-types.html
https://www.w3schools.com/sql/sql_datatypes.asp

Podstawowe operacje

Operacje *CRUD*

Skrót *CRUD* oznacza cztery podstawowe operacje wykonywane na danych implementowane w aplikacjach bazodanowych.

- **C – create (tworzenie)**
- **R – read (odczyt)**
- **U – update (aktualizacja)**
- **D – delete (usuwanie)**

Tworzenie bazy danych

Tabelę tworzy się za pomocą instrukcji

```
CREATE DATABASE nazwa_db;
```

Przykład:

```
CREATE DATABASE library;
```

Wybór bazy danych

Otwierając program do zarządzania bazą danych (**mysql**), domyślnie nie łączymy się do żadnej bazy danych. Polecenia SQL muszą być wykonywane w kontekście docelowej bazy danych.

W celu wybrania bazy danych należy skorzystać z instrukcji wbudowanej w klienta bazy danych **mysql.**

```
USE nazwa_db;
```

Przykład:

```
USE library;
```

Tworzenie użytkownika

Utworzenie użytkownika:

```
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'password';
```

Przykład:

```
CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'password';
```

Przyznawanie uprawnień

Przyznanie uprawnień do bazy:

```
GRANT ALL PRIVILEGES ON nazwa_db.* TO 'myuser'@'localhost';
```

Przykład:

```
GRANT ALL PRIVILEGES ON library.* TO 'library_user'@'localhost';
```

Po nadaniu uprawnień należy przeładować uprawnia (unikamy w ten sposób restartu serwera MySQL).

Przeładowanie uprawnień:

```
FLUSH PRIVILEGES;
```

➡ *Przyznawanie uprawnień z poziomu MySQL Workbench.*

Odbieranie uprawnień

Odbieranie uprawnień – polecenie **REVOKE** w najprostszej postaci:

```
REVOKE rodzaj_uprawnienia ON typ_obiektu FROM uzytkownikowi;
```

Przykład:

```
REVOKE ALL PRIVILEGES ON library.* FROM 'library_user'@'localhost';
```

Uprawnienia

Rodzaje najczęściej przyznawanych uprawnień:

- **`SELECT`** - odczyt danych z bazy
- **`INSERT`** - dodawanie rekordów do bazy danych
- **`DELETE`** - usuwanie rekordów z bazy danych
- **`UPDATE`** - modyfikowanie istniejących rekordów
- **`CREATE`** - tworzenie baz danych i tabel
- **`DROP`** - usuwanie baz danych i tabel
- **`ALTER`** - zmiana struktury tabel
- **`GRANT`** - przyznawanie uprawnień
- **`ALL PRIVILEGES`** – wszystkie uprawniania

Ćwiczenie 1

1. Zaloguj się do narzędzia MySQL WorkBench za pomocą konta *root* utworzonego podczas instalacji.
2. Utwórz bazę danych o nazwie *test*.
3. Utwórz użytkownika składającego się z pierwszej litery Twojego imienia i 3 pierwszych liter Twojego nazwiska (bez polskich znaków, dla mnie będzie to np. *pstu*).
4. Przyznaj sobie wszystkie uprawnienia do bazy *test*.
5. Utwórz połączenie do bazy danych *test* za pomocą zdefiniowanego wcześniej użytkownika w narzędziu MySQL WorkBench.
6. Przetestuj czy możesz zalogować się do bazy *test „swoim" loginem*.
7. Użytkownikiem *root* odbierz sobie wszystkie uprawnienia do bazy danych *test*.
8. Przetestuj czy możesz się zalogować się do bazy *test „swoim" loginem*.
9. Usuń użytkownika, którego utworzyłeś (Polecenie *DROP USER nazwa_uzytkownika*).

Tworzenie tabel

Tabelę tworzy się za pomocą instrukcji **CREATE TABLE**.

Szablon:

```
CREATE TABLE nazwa_tabeli (
          nazwa_kolumny_1 typ_danych,
          nazwa_kolumny_2 typ_danych
);
```

Przykład:

```
CREATE TABLE author (
          first_name VARCHAR(128),
          lastname VARCHAR(128)
);
```

Tworzenie tabel c.d

Próba utworzenia już istniejącej tabeli skutkuje powstaniem błędu. Można tego uniknąć stosując konstrukcję **IF NOT EXISTS**:

```
CREATE TABLE IF NOT EXISTS nazwa_tabeli (
          nazwa_kolumny_1 typ_danych,
          nazwa_kolumny_2 typ_danych
);
```

Ćwiczenie 2

1. Utwórz bazę danych o nazwie `library`.
2. Utwórz tabelę *book* zawierającą książki. Pola:
   `title, author, published, isbn, category, page_count, publisher, price`
2. Jakiego typu powinno być każde z tych pól?
3. Odpowiednie polecenia zapisz w pliku *sql*.
4. Pliki *sql* również można commitować do gita – proszę utwórz repozytorium gdzie będziesz mógł/mogła dokumentować i zapisywać swoją pracę.

Informacje o obiektach w bazie danych

Wyświetlanie listy baz danych:
Polecenie:

```
SHOW DATABASES;
```

Wyświetlanie listy tabel:
Po wyborze bazy (poleceniem **USE**):

```
SHOW TABLES;
```

Wyświetlanie definicji tabeli:
Polecenie:

DESC *nazwa_tabeli;*

Modyfikacja tabeli

Polecenie **ALTER TABLE** służy do modyfikacji istniejącej tabeli w bazie danych.

```
ALTER TABLE nazwa_tabeli <AKCJA>
```

Przykład:

- `ALTER TABLE nazwa_tabeli ADD COLUMN nowa_kolumna tinyint;`
- `ALTER TABLE nazwa_tabeli CHANGE COLUMN kolumna1 nowa_nazwa tinyint;`
- `ALTER TABLE nazwa_tabeli DROP COLUMN kolumna2;`
- `ALTER TABLE nazwa_tabeli MODIFY COLUMN kolumna1 tinyint NOT NULL;`
- `ALTER TABLE nazwa_tabeli ADD CONSTRAINT kolumna_constraint PRIMARY KEY(kolumna1);`
- `ALTER TABLE nazwa_tabeli DROP PRIMARY KEY;`

Usuwanie tabel

Polecenie **DROP TABLE** służy do usuwania tabeli

```
DROP TABLE nazwa_tabeli.
```

Przykład:

```
DROP TABLE book;
```

Polecenie **DROP TABLE** można stosować wraz ze słowami kluczowymi **IF EXISTS**, aby uniknąć błędu usuwania nieistniejącej tabeli:

```
DROP TABLE IF EXISTS nazwa_tabeli;
```

Manipulacja danymi

Wstawianie danych

Polecenie **INSERT** służy do dodawania danych do wybranej tabeli.

Przykład:

```
INSERT INTO nazwa_tabeli(kolumna1, kolumna2, kolumna3)
VALUES (wartość_kolumna_1, wartość_kolumna_2, wartość_kolumna_3);
```

Wstawianie danych c.d.

Polecenie **INSERT** nie zawsze musi wstawiać wartości do wszystkich kolumn zdefiniowanych w wybranej tabeli.

Przykład:

```
INSERT INTO nazwa_tabeli (kolumna1, kolumna3)
VALUES (wartość_kolumna_1, wartość_kolumna_3);
```

Wstawianie danych c.d

W poleceniu **INSERT** kolumny mogą być wymieniane w dowolnej kolejności.

Polecenie

```
INSERT INTO nazwa_tabeli(kolumna1, kolumna2)
VALUES('wartość_kolumna_1', 'wartość_kolumna_2');
```

oraz

```
INSERT INTO nazwa_tabeli(kolumna2, kolumna1)
VALUES('wartość_kolumna_2', 'wartość_kolumna_1');
```

dają taki sam rezultat.

Wstawianie danych c.d.

Gdy przy dodawaniu wpisu podajemy wszystkie pola możemy pominąć definicję, do których pól wartości, pamiętając o zachowaniu kolejności zgodnie ze schematem tabeli.

```
CREATE TABLE nazwa_tabeli (
    kolumna1 VARCHAR(100),
    kolumna2 VARCHAR(100);
);
INSERT INTO nazwa_tabeli
VALUES ('wartość_kolumna_1', 'wartość_kolumna_2');
```

Ćwiczenie 3

1. Korzystając z polecenia INSERT, dodaj do tabeli *book* następujące dane:

Tytuł: Spring w akcji. Wydanie IV
Autor: Craig Walls
Data wydania: 2015-08-13
ISBN: 978-83-283-0849-7
Kategoria: Programowanie java
Stron: 624
Wydawnictwo: Helion
Cena: 89.00

Tytuł: MySQL. Vademecum profesjonalisty.
Autor: Paul DuBois
Data wydania: 2014-03-28
ISBN: 978-83-246-8146-4
Kategoria: Bazy danych
Stron: 1216
Wydawnictwo: Helion
Cena: 149.90

Modyfikacja danych

Polecenie **UPDATE** służy do aktualizowania danych w wybranej tabeli

Przykład:

```
UPDATE nazwa_tabeli
SET kolumna1 = wartosc_1
WHERE kolumna2 = wartosc_2;
```

Wartość NULL

Jeżeli używając instrukcji **INSERT** nie zostaną podane wartości dla wszystkich kolumn, to dla niewymienionych kolumn zostanie wstawiona wartość **NULL**.

Przykład:

```
INSERT INTO book (title, author) VALUES ('Nowa książka', 'Autor');
```

Ograniczenie NOT NULL

Instrukcja **NOT NULL** umieszczona w definicji kolumny skryptu tworzącego tabelę (**CREATE TABLE**) oznacza, że dane wstawiane do tej tabeli muszą być różne od **NULL** (muszą mieć jakąś wartość).

Przykład:

```
CREATE TABLE PRZYKLADOWA_TABELA (
          KOLUMNA_1 VARCHAR(10) NOT NULL,
          KOLUMNA_2 VARCHAR(10) NOT NULL
);

INSERT INTO PRZYKLADOWA_TABELA(KOLUMNA_1, KOLUMNA_2) VALUES('WARTOSC_1',
'WARTOSC_2'); # OK

INSERT INTO PRZYKLADOWA_TABELA(KOLUMNA_1) VALUES('WARTOSC_1'); # BLAD!
```

Ograniczenie CHECK

```
CREATE TABLE PRZYKLADOWA_TABELA (
          KOLUMNA   CHAR(1) CHECK ( KOLUMNA IN ( 'M', 'K' ) )
);
INSERT INTO PRZYKLADOWA_TABELA(KOLUMNA) VALUES('M'); # OK
INSERT INTO PRZYKLADOWA_TABELA(KOLUMNA) VALUES('Z'); # BLAD!
```

Ograniczenie UNIQUE

Instrukcja **UNIQUE** dodana do definicji kolumny powoduje, że w wybranej kolumnie mogą znajdować się tylko wartości unikatowe. Przykład:

```
CREATE TABLE przykladowa_tabela (
            unikalna    CHAR(10) UNIQUE
);

Lub

CREATE TABLE przykladowa_tabela (
            unikalna    CHAR(10),
            UNIQUE(unikalna)
);

INSERT INTO przykladowa_tabela(unikalna) VALUES('WARTOSC_1'); # OK

INSERT INTO przykladowa_tabela(unikalna) VALUES('WARTOSC_1'); # BLAD!
```

Wartość domyślna

Instrukcja **DEFAULT** wykorzystywana jest w sytuacji gdy kolumna powinna przyjmować wartość (domyślną) nawet jeśli nie została podana w instrukcji **INSERT**.

Przykład:

```
CREATE TABLE przykladowa_tabela (
            kolumna1 VARCHAR(10) NOT NULL,
            kolumna2 VARCHAR(10) DEFAULT 'WARTOSC_2'
);
INSERT INTO przykladowa_tabela(kolumna1) VALUES('WARTOSC_1'); # OK
SELECT * FROM przykladowa_tabela; # kolumna1=WARTOSC_1, kolumna2=WARTOSC_2
```

Ćwiczenie 4

1. Zmodyfikuj tabelę book (wykorzystaj ALTER TABLE) :
   - Kolumny `title, author, isbn` - nie akceptują wartości NULL.
   - Wartości w kolumnie `isbn` powinny być unikatowe.
   - Kolumna `publisher` powinna być wypełniane wartością 'nieznana' jeśli nie zostanie podana wartość w INSERT.
2. Dodaj kolumnę z liczbą sztuk książki `in_stock` z wartością domyślną 0.
3. Zaktualizuj skrypt *sql* tworzenia tabeli uwzględniając powyższe ograniczenia.

Polecenie TRUNCATE

Instrukcja **TRUNCATE** służy do usuwania wszystkich wierszy z tabeli.

```
TRUNCATE nazwa_tabeli;
```

Usuwanie danych

Instrukcja **DELETE** służy do usuwania wybranych wierszy z tabeli.

```
DELETE FROM nazwa_tabeli WHERE [warunek];
```

Przykład:

```
DELETE FROM book WHERE isbn = '978-83-283-0849-7';
```

Ostrzeżenie – DELETE, UPDATE

Zarówno przy **DELETE** jak i **UPDATE** należy pamiętać o dodaniu warunku **WHERE**. W przeciwnym razie usuniemy wszystkie wpisy w tabeli (a przy **UPDATE** wszystkie zaktualizujemy)!

Pobieranie danych

Polecenie SELECT

Polecenie **SELECT** służy do pobierania danych z baz. Zapytania **SELECT** nazywane są też zapytaniami wybierającymi

```
SELECT kolumna1, kolumna2 FROM nazwa_tabeli;
```

Przykład:

```
SELECT title, author FROM book;

SELECT * FROM book; #wyświetla wszystkie kolumny
```

ALIAS

```
SELECT column_name AS alias_name
FROM table_name;

SELECT column_name, ...
FROM table_name AS alias_name;

Przykłady:
SELECT title AS tytuł FROM book;

SELECT b.title FROM book AS b;
```

Instrukcja WHERE

Instrukcja **WHERE** służy do filtrowania danych w tabeli.

Przykład:

```
SELECT kolumna1, kolumna2 FROM nazwa_tabeli WHERE warunek;
```

WHERE – operatory

Operatory algebraiczne

Symbol	Opis
=	równe
<	mniejsze niż
<=	mniejsze niż lub równe
>	większe niż
>=	większe niż lub równe
<>	różne
!=	nie równe

WHERE – operatory c.d.

Operatory algebraiczne – przykłady

```
SELECT * FROM book WHERE title = 'wartość';

SELECT * FROM book WHERE page_count >= 1000;
```

WHERE – operatory c.d.

Operatory logiczne:

Symbol	Opis
NOT	negacja
OR	suma logiczna
AND	Iloczyn logiczny

WHERE – operatory c.d.

Operatory logiczne – przykłady:

```
SELECT * FROM book WHERE page_count > 1000 AND category = 'bazy
danych';
SELECT * FROM book WHERE page_count > 1000 OR NOT category = 'bazy
danych';
```

WHERE – operatory c.d.

Operatory specjalne

Symbol	Opis
BETWEEN	przedział dwustronnie domknięty
IN	lista, alternatywa dla wielu OR dla kolumny
ANY	prawda jeśli jedna pozycja na liście prawdziwa
SOME	prawda jeśli kilka pozycji na liście prawdziwych
ALL	prawda jeśli wszystkie pozycje na liście są prawdziwe
EXISTS	prawda jeśli zapytanie zwraca rekordy
LIKE	podobny do wzorca

Operator LIKE

Operator **LIKE** jest używany w klauzuli **WHERE** do wyszukiwania określonego wzoru w kolumnie.

W połączeniu z operatorem LIKE istnieją dwa **symbole wieloznaczne**:

% - znak procentowy oznacza zero, jeden lub wiele znaków
_ - podkreślenie oznacza pojedynczy znak

WHERE – operatory c.d.

Operatory specjalne – przykłady:

```
SELECT * FROM book WHERE price BETWEEN 50 AND 90;

SELECT * FROM book WHERE isbn IN ('978-83-283-0849-7', '978-83-246-8146-4');

SELECT * FROM book WHERE title LIKE 'Spring w akcji. Wydanie I_';

SELECT * FROM book WHERE title LIKE '%Spring%';
```

SELECT DISTINCT

Instrukcja **SELECT DISTINCT** jest używana do zwracania tylko różnych (różnych) wartości. Wewnątrz tabeli, kolumna często zawiera wiele podwójnych wartości, a czasami chcesz tylko wymienić różne (odrębne) wartości.

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

Przykład:
SELECT DISTINCT publisher **FROM** book;

Klauzula LIMIT

Klauzula **LIMIT** pozwala wybrać ograniczoną liczbę rekordów (począwszy od).

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT offset, number;
```

Przykład:

SELECT publisher **FROM** book **LIMIT** 1,2;

Instrukcja ORDER BY

Instrukcja `ORDER BY` pozwala na sortowanie wyników zapytań. Domyślnie (jeśli jawnie nie podamy sposobu sortowania) wartości w kolumnie sortowane są rosnąco.

```
SELECT kolumna1, kolumna2 FROM nazwa_tabeli ORDER BY kolumna1,
kolumna2, ... kolumnaN [ASC, DESC];
```

```
SELECT * FROM book ORDER BY author;
```

```
SELECT * FROM book ORDER BY author DESC;
```

Instrukcja GROUP BY

Operator **GROUP BY** pozwala na grupowanie wyników zapytań. Wykorzystywany najczęściej do obliczania funkcji agregujących dla grup.

```
SELECT kolumna1, kolumna2, SUM(kolumna3)
FROM nazwa_tabeli
GROUP BY kolumna1, kolumna2, ... kolumnaN;
```

Przykład:

```
SELECT  publisher, SUM(price)
FROM book
GROUP BY publisher;
```

Funkcje agregujące

Funkcje agregujące pozwalają na wykonywanie na wynikach zapytań funkcji matematycznych.

Symbol	Opis
AVG	średnia wszystkich wartości
COUNT	zwraca ilość wierszy
SUM	suma wszystkich wartości
MAX	maksymalna wartość
MIN	minimalna wartość

Funkcje agregujące c.d.

Funkcje agregujące, przykłady:

```
SELECT SUM(page_count) AS total_pages FROM book;
SELECT AVG(page_count) FROM book;
SELECT MIN(page_count) FROM book;
SELECT MAX(page_count) FROM book;
```

Klauzula HAVING

Klauzula **WHERE** wykonuje się przed grupowaniem, a zatem nie można w tej klauzuli sprecyzować warunku zawierającego funkcje grupowe. Aby taki warunek zawrzeć w zapytaniu należy zastosować dodatkową klauzulę **HAVING** wraz z odpowiednim warunkiem.

```
SELECT column_name, ...
FROM table_name
WHERE condition
GROUP BY column_name, ...
HAVING condition
ORDER BY column_name, ...;
```

Przykład:

```
SELECT count(*) as count, publisher
FROM book
GROUP BY publisher
HAVING count > 1;
```

Pracodawcy lubią o to pytać na rozmowach kwalifikacyjnych ☺

Podzapytania

W SQL możemy wykorzystywać podzapytania, których wynik może stanowić zbiór wykorzystywany jako zbiór dla operatora **WHERE**.

Przykład:

```
SELECT * FROM book

WHERE price =(SELECT min(price) FROM book);
```

Operator EXISTS

Operator **EXISTS** służy do sprawdzania istnienia jakiegokolwiek rekordu w podzapytaniu.
Operator **EXISTS** zwraca wartość **true**, jeśli podzapytanie zwraca jeden lub więcej rekordów.

```
SELECT column_name, ...
FROM table_name
WHERE EXISTS
 (SELECT column_name
  FROM table_name
  WHERE condition);
```

Przykład:

```
SELECT * FROM book
WHERE EXISTS(SELECT * FROM book WHERE price < 100);
```

Ćwiczenie 5

1. Wczytaj dane ze skryptu input_books.sql
2. Wyświetl książki z kategorii *Klasyka polska.*
3. *Policz ile jest książek w przedziale cenowym 10 - 30 zł.*
4. *Wyświetl największą liczbę stron oraz najmniejszą w książkach.*
5. *Wyświetl tytuł książki z największą ilością stron.*
6. Wyświetl książki wydane przed 2000 rokiem.
7. Policz ile książek wydało każde z wydawnictw po 2000 roku.
8. *Policz jaka jest suma stron książek wydanych przez wydawnictwo PWN.*
9. *Wyświetl kategorie książek dla których istnieją co najmniej 2 książki, których cena jest większa od 10.01 zł.*
10. *Wyświetl tytuły książek posortowane alfabetycznie w porządku malejącym, które nie są lekturą i których cena jest większa niż 20 zł.*
11. *Wyświetl wszystkie dane z tabeli book tłumacząc nazwy kolumn na język polski.*

Wybrane funkcje czasu

Symbol	Opis
NOW()	aktualna data i czas
CURDATE()	aktualna data
CURTIME()	aktualny czas
DATE()	wyciąganie datę
EXTRACT()	zwracanie elementy czasu/daty
DATE_ADD()	dodanie przedziału czasu do daty
DATE_SUB()	odjęcie przedziału czasu od daty
DATEDIFF()	liczba dni pomiędzy datami
DATE_FORMAT()	wyświetlenie daty/czasu w różnym formacie

Wybrane funkcje czasu c.d.

Przykład:

```
SELECT NOW(),CURDATE(),CURTIME();
```

```
+---------------------+------------+-----------+
| NOW()               | CURDATE()  | CURTIME() |
+---------------------+------------+-----------+
| 2020-02-16 13:40:17 | 2020-02-16 | 13:40:17  |
+---------------------+------------+-----------+
```

Wybrane funkcje łańcuchów

Symbol	Opis
CONCAT()	łączenie łańcuchów znaków
CONCAT_WS()	łączenie łańcuchów znaków z użyciem separatora
LOWER()	zamiana łańcucha na małe litery
UPPER()	zamiana łańcucha na duże litery
TRIM()	usuwanie “białych” znaków z początku i końca
SUBSTR()	wybieranie podciągu znaków z łańcucha
FORMAT()	wyświetla liczbę w określonym formacie

Wybrane funkcje łańcuchów c.d.

Przykład:

```
SELECT title FROM book
WHERE lower(author)='craig walls';
```

https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

Instrukcja INSERT INTO

Instrukcja **INSERT INTO** pozwala kopiować dane z jednej tabeli i wstawić je do innej tabeli. Wymaga zgodności typów danych w tabelach źródłowych i docelowych.

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Przykład:

```
INSERT INTO magazine(SELECT * FROM book);
```

```
CREATE TABLE magazine
title VARCHAR(256),
author VARCHAR(128),
published DATE,
isbn VARCHAR(32),
category VARCHAR(128),
page_count INT,
publisher VARCHAR(128),
price FLOAT,
in_stock INT);
```

Instrukcja INSERT INTO c.d.

Poniższe polecenie wstawia tylko niektóre kolumny z jednej tabeli do innej:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

Przykład:
INSERT INTO magazine (author) **SELECT** author **FROM** book;

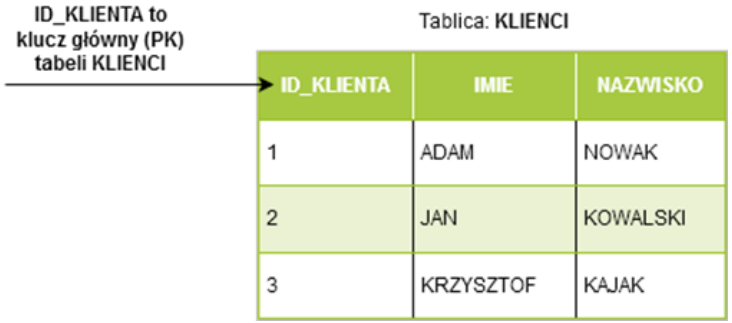
Ćwiczenie 6

1. Utwórz tabele *backup_book* i przekopiuj do niej wszystkie książki.
2. Utwórz tabele *selected_book* i przekopiuj do niej wszystkie książki z kategorii *Bazy danych* (pomijając kolumnę kategoria).

Klucze podstawowe i obce

Klucz podstawowy (ang. primary key)

- Jedno lub więcej pól tabeli, których wartość jednoznacznie identyfikuje wiersz w tabeli.
- Nie może zawierać powtarzających się danych i nie może mieć wartości pustej (NULL).
- Zapewnia unikalną wartość dla każdego wiersza danej kolumny.
- Każda tabela może mieć najwyżej jeden klucz główny.



PRIMARY KEY

Poniższa instrukcja SQL tworzy klucz główny w kolumnie *id*, gdy tworzona jest osoba:

```
CREATE TABLE author (
    id INT NOT NULL,
    first_name VARCHAR(128) NOT NULL,
    last_name VARCHAR(128) NOT NULL,
    PRIMARY KEY (id)
);
```

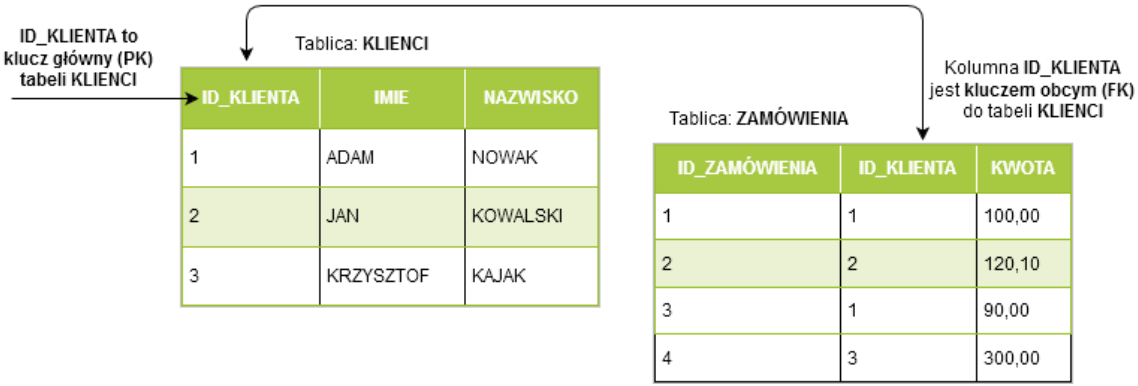
PRIMARY KEY c.d.

Aby stworzyć klucz główny oparty na wielu kolumnach, użyj następującej składni SQL:

```
CREATE TABLE author (
    id INT NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    CONSTRAINT PK_author PRIMARY KEY (id, last_name)
);
```

Klucz obcy (ang. foreign key)

- Pole (zbiór pól) odwołujący się do pola (zbioru pól) klucza podstawowego w innej tabeli.
- Umożliwia definiowanie relacji pomiędzy tabelami.



ID_KLIENTA	IMIE	NAZWISKO
1	ADAM	NOWAK
2	JAN	KOWALSKI
3	KRZYSZTOF	KAJAK

ID_ZAMÓWIENIA	ID_KLIENTA	KWOTA
1	1	100,00
2	2	120,10
3	1	90,00
4	3	300,00

FOREIGN KEY

Następująca instrukcja SQL tworzy klucz obcy przechowywany w kolumnie `author_id`.

```
CREATE TABLE book (
    id INT NOT NULL,
    title VARCHAR(128) NOT NULL,
    author_id INT,
    ...
    PRIMARY KEY (id),
    FOREIGN KEY (author_id) REFERENCES author(id)
);
```

Więzy integralności

- Więzy integralności (ang. constraint) zapobiegają powstawaniu osieroconych wierszy w tabelach.
- Wiersz z tabeli nadrzędnej (rodzica) nie może być usunięty jeśli istnieją wiersze zależne w tablicy podrzędnej (dziecka).
- Mechanizm mający na celu wymuszanie więzów integralności to połączenie klucza podstawowego tabeli rodzica oraz klucza obcego odwołującego się do tego klucza dla tabeli dziecka.

Przykłady:

RESTRICT - nie pozwala na dokonanie zmian naruszających powiązanie.

CASCADE - nakazuje zmianom na propagację kaskadową wzdłuż drzewa powiązanych tabel.

AUTOINCREMENT

Pozwala automatycznie wygenerować unikalny numer po wstawieniu nowego rekordu do tabeli.
Przykład:

```
CREATE TABLE author (
    id INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(128) NOT NULL,
    last_name VARCHAR(128) NOT NULL,
    PRIMARY KEY (id)
);
INSERT INTO author(first_name, last_name) VALUES ('Craig', 'Walls');
```

id	first_name	last_name
1	Craig	Walls
NULL	NULL	NULL

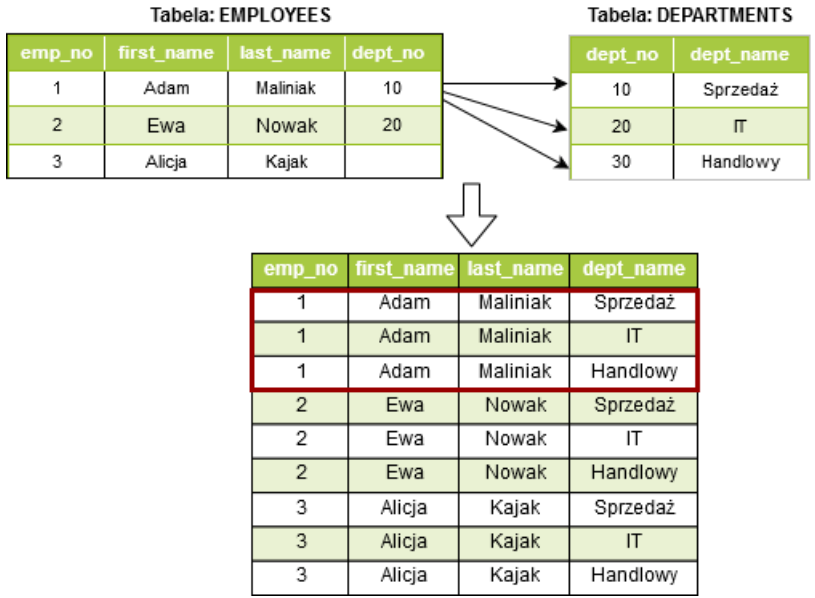
Złączenia tabel

CROSS JOIN (iloczyn kartezjański)

```
SELECT lista_kolumn
  FROM tabela_1
 CROSS JOIN table_2;
```

Przykład:

```
SELECT emp_no, first_name, last_name,
       dept_name
  FROM employees
 CROSS JOIN departments;
```

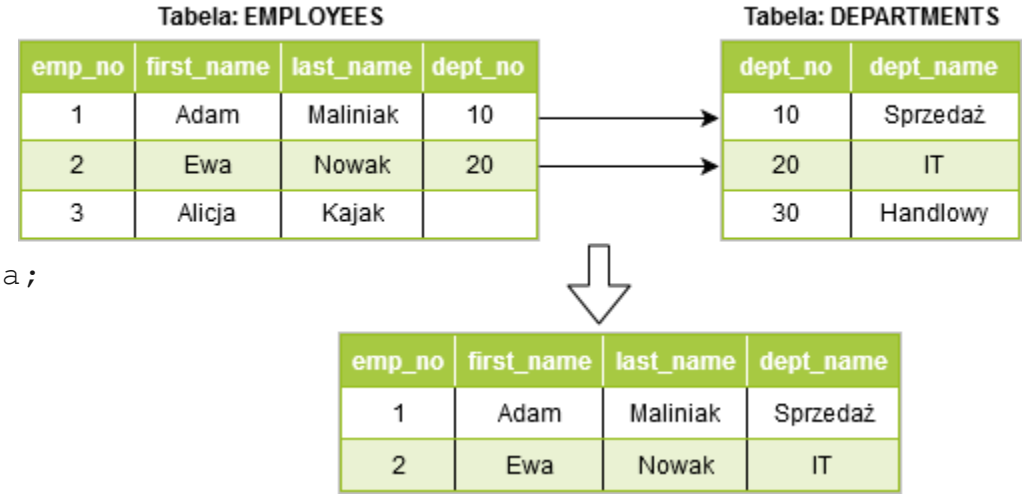


JOIN (INNER JOIN)

```
SELECT * FROM tabela_1
JOIN table_2
ON tabela_1.kolumna = tabela_2.kolumna;
```

Przykład:

```
SELECT e.emp_no, e.first_name,
       e.last_name, d.dept_name
  FROM employees e
  JOIN departments d
    ON e.dept_no = d.dept_no;
```

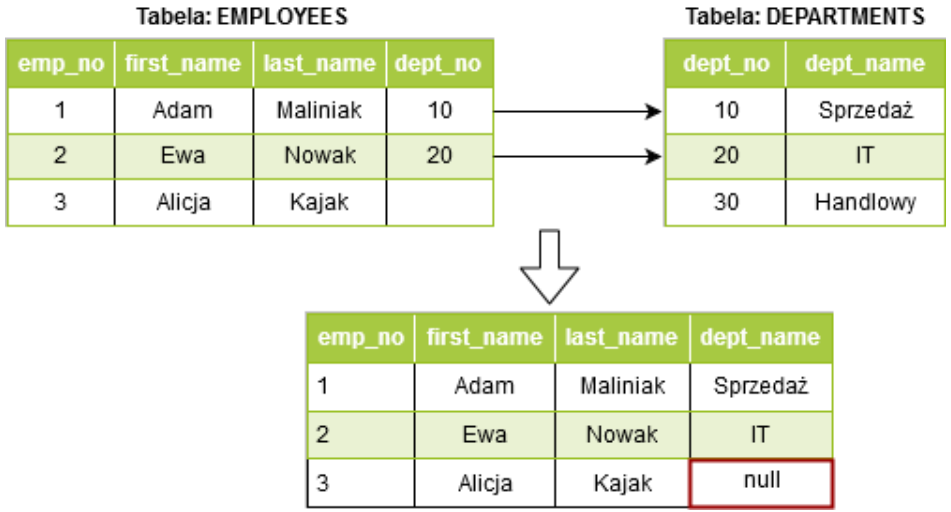


LEFT JOIN (LEFT OUTER JOIN)

```
SELECT * FROM tabela_1
LEFT JOIN table_2
ON tabela_1.kolumna = tabela_2.kolumna;
```

Przykład:

```
SELECT e.emp_no, e.first_name,
       e.last_name, d.dept_name
  FROM employees e
  LEFT JOIN departments d
    ON e.dept_no = d.dept_no;
```

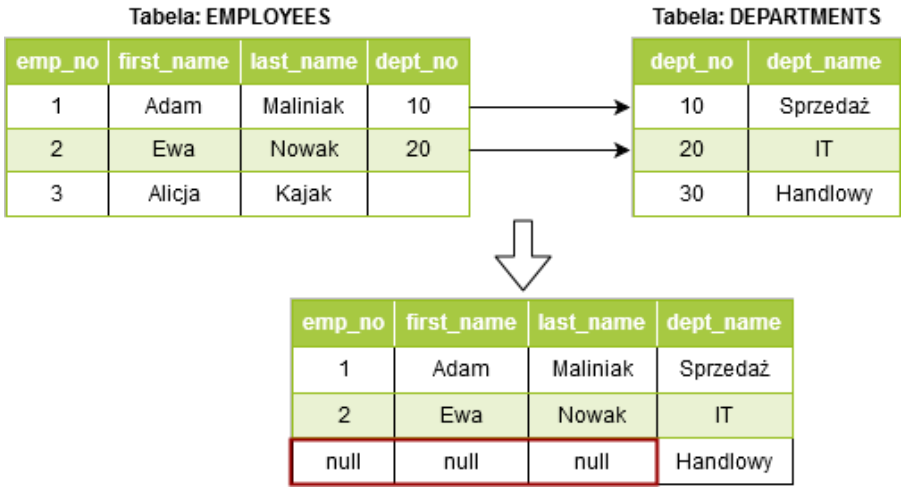


RIGHT JOIN (RIGHT OUTER JOIN)

```
SELECT * FROM tabela_1
RIGHT JOIN table_2
ON tabela_1.kolumna = tabela_2.kolumna;
```

Przykład:

```
SELECT e.emp_no, e.first_name,
       e.last_name, d.dept_name
  FROM employees e
 RIGHT JOIN departments d
    ON e.dept_no = d.dept_no;
```



UNION

```
SELECT kolumny
  FROM tabela_1
 UNION
SELECT kolumny
  FROM tabela_2;
```

Przykład:

```
SELECT first_name, last_name
  FROM employees
 UNION
SELECT first_name, last_name
  FROM external_employees;
```

Tabela: EMPLOYEES

emp_no	first_name	last_name	dept_no
1	Adam	Maliniak	Sprzedaż
2	Ewa	Nowak	IT
3	Alicja	Kajak	Handlowy

Tabela: EXTERNAL_EMPLOYEES

ext_no	first_name	last_name
1	Marcin	Kura
2	Ewa	Nowak
3	Zenon	Chleb

first_name	last_name
Adam	Maliniak
Ewa	Nowak
Alicja	Kajak
Marcin	Kura
Zenon	Chleb

UNION ALL

```
SELECT kolumny
  FROM tabela_1
 UNION ALL
SELECT kolumny
  FROM tabela_2;
```

Przykład:

```
SELECT first_name, last_name
  FROM employees
 UNION ALL
SELECT first_name, last_name
  FROM external_employees;
```

Tabela: EMPLOYEES

emp_no	first_name	last_name	dept_name
1	Adam	Maliniak	Sprzedaż
2	Ewa	Nowak	IT
3	Alicja	Kajak	Handlowy

Tabela: EXTERNAL_EMPLOYEES

ext_no	first_name	last_name
1	Marcin	Kura
2	Ewa	Nowak
3	Zenon	Chleb

first_name	last_name
Adam	Maliniak
Ewa	Nowak
Alicja	Kajak
Marcin	Kura
Ewa	Nowak
Zenon	Chleb

Ćwiczenie 7

1. Utwórz nową bazę danych *cinema*.
2. Utwórz tabele *actor* i *movie*. Im więcej pól opisujących aktora i film dodasz tym lepiej. Pamiętaj o utworzeniu relacji pomiędzy tabelami. Jaki rodzaj relacji zastosujesz?
3. Do tabeli *actor* dodaj swojego ulubionego aktora.
4. Do tabeli *movie* dodaj trzy filmy, w których zagrał Twój ulubiony aktor.
5. Za pomocą jednego polecania *select* wypisz informacje o filmach, w których zagrał Twój ulubiony aktor.
6. Dodaj kolejny dowolny film (w którym Twój ulubiony aktor nie grał) oraz aktora który zagrał w tym filmie.
7. Upewnij się, że Twoje polecenie z podpunktu 5) wciąż działa prawidłowo.
8. Zastanów się jaki inny aktor zagrał w filmach, które wypisałeś w podpunkcie 4). Do każdego filmu utworzonego w podpunkcie 4) dodaj co najmniej jednego nowego aktora który zagrał w tym filmie. Czy masz możliwość powiązania nowego aktora z istniejącym filmem?
9. Wypisz (instrukcja select) tytuły filmów w których zagrał więcej niż jeden aktor.

Projektowanie bazy danych

Proces projektowania bazy danych

Proces projektowania bazy danych możemy podzielić na kilka etapów:

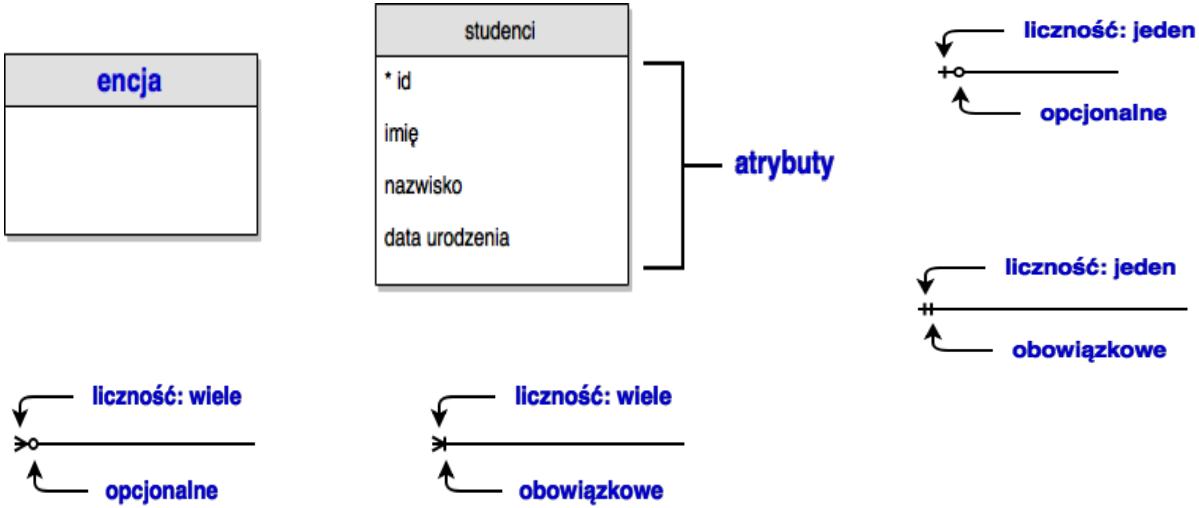
1. Określenie celu, któremu ma służyć baza danych (Cel bieżącego zadania: np. ulepszenie księgarni lub innego sklepu).
2. Określenie tabel do przechowywania danych.
3. Określenie pól w tabelach (nazwy, typy danych, formaty..).
4. Przypisanie polom jednoznacznych wartości w każdym rekordzie (ustalenie klucza podstawowego).
5. Określenie relacji między tabelami.
6. Udoskonalenie projektu – sprawdzenie poprawności działania.
7. Edycji danych i tworzenie innych obiektów.

Diagram ERD

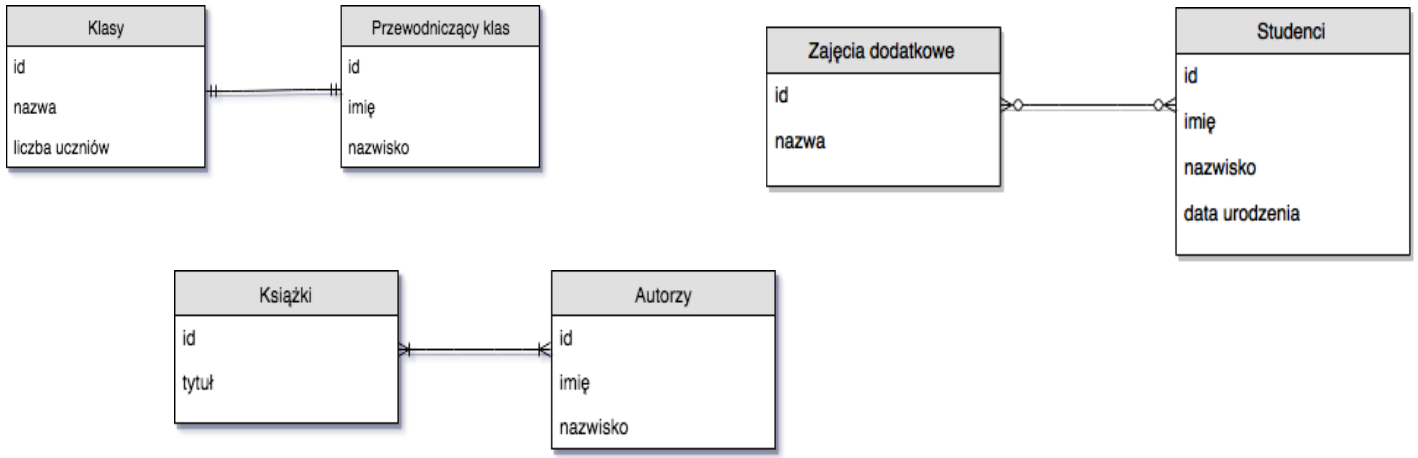
Entity Relationship Diagram - rodzaj graficznego przedstawienia związków pomiędzy encjami używanymi w projektowaniu systemów informacyjnych do przedstawienia konceptualnych modeli danych używanych w systemie.

- **Encja** – istniejący obiekt, rozróżnialny od innych bytów tego samego typu (np. student, wydział, katedra itp.).
- **Atrybut** – informacja charakteryzująca encję (np. wiek, wydział).

Elementy diagramu ERD



ERD - relacje


Klasy
id
nazwa
liczba uczniów
Przewodniczący klas
id
imię
nazwisko
Zajęcia dodatkowe
id
nazwa
Studenci
id
imię
nazwisko
data urodzenia
Książki
id
tytuł
Autorzy
id
imię
nazwisko

Ćwiczenie 8

1. Utwórz diagram ERD dla Księgarni. Na potrzeby ćwiczenia zakładamy, że potrzebujemy danych o Autorach, Książkach, Wydawnictwach, Czytelnikach/klientach .
2. Należy zaprojektować tabele oraz określić relację między nimi na diagramie ERD w narzędziu MySQL Workbench (*File -> New Model -> Add Diagram*).

https://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial-creating-a-model.html

Indeksy

Wyszukiwanie danych

Kiedy warunki z klauzuli **`WHERE`** operują na kolumnie, dla której nie utworzono indeksu, system zarządzania relacyjnymi bazami danych zaczyna od samego początku takiej kolumny, a następnie analizuje jej kolejne wiersze.

Jeśli Twoja tabela zawiera ogromną ilość danych, to wykonanie takiej operacji może potrwać zauważalną chwilę.

Indeksy cechy

- Tworzone na pojedynczych kolumnach lub zbiorze kolumn.
- Są strukturami fizycznym zajmującym dodatkowe miejsce na dysku.
- Przyspieszają wyszukiwanie danych.
- Przyspieszają sortowanie danych.
- Spowalniają aktualizację danych w kolumnie.

Indeksy wady i zalety

ZALETY:
W przypadku utworzenia indeksu na kolumnie system zarządzania relacyjnymi bazami danych przechowuje dodatkowe informacje o tej kolumnie, które są w stanie przyspieszyć wyszukiwanie.

WADY:
Zysk szybkości wymaga pewnego kompromisu – jest nim utrata miejsca na dysku oraz spowolnienie działania operacji modyfikujących dane. Dlatego warto indeksować tyle te kolumny, które będziesz często przeszukiwał.

Indeksy

Tworzenie indeksu:

```
CREATE INDEX index_name ON table_name (column_name);
CREATE INDEX index_name ON table_name (column1_name, column2_name);
CREATE UNIQUE INDEX index_name ON table_name (column_name);
```

Usuwanie indeksu:

```
DROP INDEX index_name ON table_name;
```

Polecana literatura:

Dokumentacja MySQL
Tutorial SQL w3schools.com
MySQL tutorial

software development academy

Kontakt:

przemyslawstudzinski1993@gmail.com

https://www.linkedin.com/in/przemyslawstudzinski/