



# Bazy danych SQL

Adam ŁAgoda



1. Teoria baz danych
2. MySQL - Instalacja i przygotowanie.
3. Instrukcje języka SQL.
4. Typy danych oraz postacie normalne.
5. Diagramy ER.
6. Złączenia i transakcje.
7. Procedury składowane i wyzwalacze.
8. Indeksy.
9. Procesy.
10. Ciekawostki.

## Baza danych

**Baza danych** – to uporządkowany zbiór danych z pewnej dziedziny tematycznej, zorganizowany w sposób umożliwiający ich wyszukiwanie według zadanych kryteriów.



# Serializacja

Java zapewnia mechanizm serializacji, pozwalający przechowywać obiekty w postaci np. plików XML.

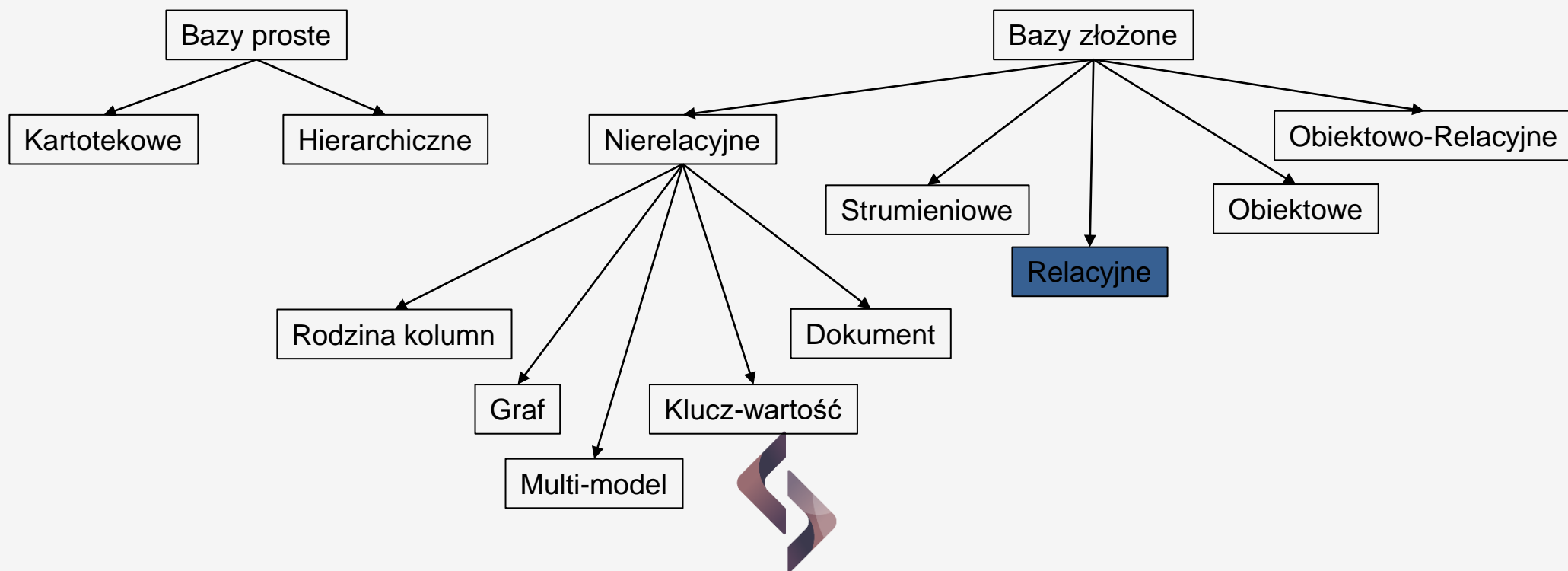
```
Object obj = Deserialize(xml)
String xml = Serialize(obj)
```

Przewagą baz danych nad taką formą przechowywania informacji jest m. in.:

- Mniejsza wydajność odczytu (deserializacja XML-a),
- Brak możliwości zmiany nazw pól,
- Brak języka zapytań,
- Pobieranie całych obiektów – brak prostego mechanizmu odwoływania się do pól.



# Rodzaje baz danych



# RDBMS

System zarządzania relacyjną bazą danych  
(ang. Relational Database Management System RDBMS).





# Wstęp do baz SQL

Pojęcia ogólne

## Co to jest SQL

**SQL** jest standardowym językiem do **przechowywania, manipulowania i pobierania** danych w bazach danych.





## Co to jest SQL

- SQL - Structured Query Language,
- Język specjalizowany (domain specific vs general-purpose language),
- SQL umożliwia uzyskiwanie dostępu i manipulowanie bazami danych,
- Jest to język deklaratywny z rozszerzeniami proceduralnymi,
- SQL jest standardem ANSI (American National Standards Institute) oraz ISO (International Organization for Standardization),
- Aktualna wersja standardu: SQL:2019.



## Co można zrobić przy pomocy SQL - I

- 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



## Co można zrobić przy pomocy SQL - II

- 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



## SQL jest standardem, ale...

Chociaż 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.



## Podział SQL

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



## SQL - uwagi

MySQL akceptuje polecenia pisane zarówno małymi, jak i dużymi literami. Na zajęciach będziemy jednak pisać wszystkie słowa kluczowe dużymi literami. Każde polecenie SQL kończymy średnikiem.





# Projektowanie baz danych

Wstęp

## Co oznacza RDBMS

**RDBMS** oznacza system zarządzania relacyjnymi bazami danych. RDBMS jest podstawą SQL i dla wszystkich nowoczesnych systemów baz danych, takich jak MS SQL Server, IBM DB2, Oracle, MySQL i Microsoft Access.





## Co oznacza RDBMS

Dane w RDBMS są przechowywane w obiektach bazy danych zwanych **tabelami**.

Tabela to zbiór powiązanych wpisów danych i składa się z **kolumn** i **wierszy**.



## Tabele i kolumny

Każda **tabela** jest podzielona na mniejsze jednostki nazywane **polami**.

Pole to **kolumna** w tabeli, która ma na celu przechowywanie szczegółowych informacji o każdym rekordzie w tabeli.



# Column vs row-oriented DBMS

Tabela jest formą abstrakcji. Fizycznie dane na dysku są zapisywane w blokach. Systemy relacyjnych baz danych stosują strategie zapisu w blokach danych w formie wierszy lub kolumn.

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	60000
002	12	Jones	Mary	80000
003	11	Johnson	Cathy	94000
004	22	Jones	Bob	55000

## Row-oriented

**001:**10, Smith, Joe, 60000;  
**002:**12, Jones, Mary, 80000;  
**003:**11, Johnson, Cathy, 94000;  
**004:**22, Jones, Bob, 55000;

## Column-oriented

10:**001**, 12:**002**, 11:**003**, 22:**004**;  
Smith:**001**, Jones:**002**, Johnson:**003**, Jones:**004**;  
Joe:**001**, Mary:**002**, Cathy:**003**, Bob:**004**;  
60000:**001**, 80000:**002**, 94000:**003**, 55000:**004**;

rowid

Kompresja obiektów

...; Smith:001; **Jones:002,004**; Johnson:003;...





Tytuł: MongoDB w akcji  
Autor: Kyle Banker, Peter Bakkum, Shaun Verch,  
Doug Garrett, Tim Hawkins  
Data wydania: 2016-12-23  
ISBN: 978-83-283-1918-9  
Kategoria: bazy danych  
Stron: 512  
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: 149.00

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	autor	data wydania	isbn	kategoria	stron	wydawnictwo	cena
MongoDB w akcji	Kyle Banker, Peter Bakkum, Shaun Verch, Doug Garrett, Tim Hawkins	2016-12-23	978-83-283-1918-9	bazy danych	512	Helion	89.00
MySQL. Vademecum profesjonalisty.	Paul DuBois	2014-03-28	978-83-246-8146-4	bazy danych	1216	Helion	149.00
<b>Spring w akcji. Wydanie IV</b>	Craig Walls	2015-08-13	978-83-283-0849-7	programowanie java	624	Helion	<b>89.00</b>

Dane zorganizowane w taki sposób tworzą jedną z podstawowych struktur baz danych: **tabelę**.

# Schemat baz danych

**Schemat baz danych** – określa jaka powinna być struktura danych oraz w jaki sposób dane są powiązane.



# Relacyjna baza danych

Baza danych, w której zbiór danych przechowywany jest w postaci **tabel** połączonych **relacjami**.





# Relacje w bazach danych

Wstęp



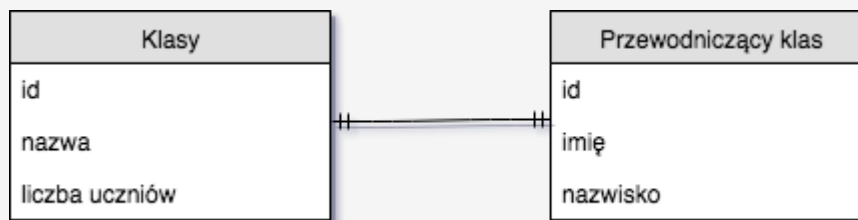
## Relacje

W relacyjnych bazach danych rozróżniamy 3 typy relacji ze względu na rodzaj powiązania istniejącego pomiędzy tabelami.



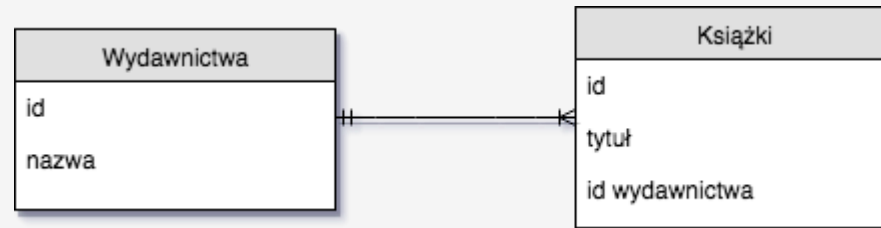
## Jeden do jednego [1-1]

Relacja *jeden-do-jeden* oznacza, iż jeden rekord w pierwszej tabeli odpowiada dokładnie jednemu rekordowi w tabeli drugiej. Klasa ma jednego przewodniczącego, a przewodniczący uczęszcza do jednej klasy.



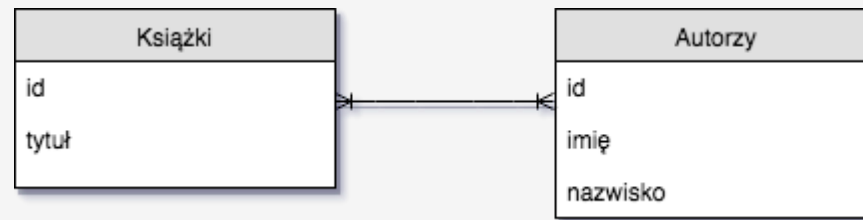
## Jeden do wielu [1-N]

Relacja *jeden-do-wielu* oznacza, iż jeden rekord w pierwszej tabeli odpowiada wielu rekordom w tabeli drugiej. Książka wydawana jest przez jedno wydawnictwo, ale wydawnictwo wydaje wiele książek.



## Wiele do wielu [N-N]

Relacja *wiele-do-wielu* oznacza, że kilka rekordów z pierwszej tabeli odpowiada wielu rekordom z tabeli drugiej. Książka może mieć wielu autorów, a autorzy mogą napisać wiele książek.





# MySQL

Instalacja i pierwsze kroki

## MySQL vs MariaDB

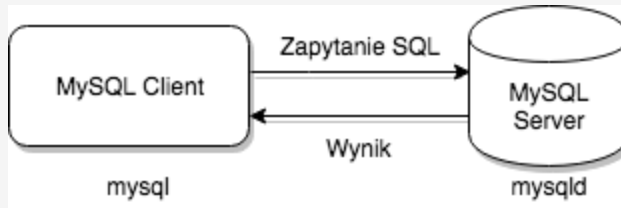
**MySQL** – baza danych rozwijany obecnie przez firmę Oracle, dostępna na licencji GPL i komercyjnej.

**MariaDB** – baza danych stworzona przez grupę (głównie) byłych pracowników MySQL AB, pod przewodnictwem Michaela Wideniusa, współtwórcy MySQL.



# mysql - działanie

Architektura klient - serwer



# MySQL - instalacja

1. Pobieramy paczkę (420.6M) ze strony:  
<https://dev.mysql.com/downloads/installer/>
2. Instalator zawiera pełen zestaw narzędzi deweloperskich. Do naszych potrzeb wykorzystamy:
  - MySQL Shell
  - MySql Server 8.0
  - MySQL Workbench 8.0
  - MySQL Notifier 1.1.8
  - Documentation
  - Samples and Examples





## MySQL Shell

Rozbudowany klient MySQL Server. Poza standardową funkcjonalnością `mysqlsh` udostępnia obsługę języków skryptowych (**Java Script i Python**) oraz między innymi API do administrowania bazą danych – AdminAPI

Po uruchomieniu wykonujemy następujące polecenia:

1. `\sql` – przejście do trybu SQL w konsoli
2. `\connect --mysqlx root@localhost:33060` – połączenie się z MySQL Server z wykorzystaniem X Protocol



## MySQL Shell - opcjonalnie

MySQL Shell umożliwia nie tylko wykonywanie poleceń SQL, ale również składni języka JavaScript. Dzięki temu możemy sterować np. atrybutami obiektu aktualnej sesji:

1. `\use mysql;`
2. `\select * from user;` - wyniki przedstawione w formie tabelarycznej
3. `\js`
4. `shell.options.set(,resultFormat','json')`
5. `session.sql(„select * from user”);` - wynik zapytania w formacie JSON
6. `shell.options.set(,resultFormat','json')`





# Typy danych

# Typy danych w MySQL

<b>Znakowe</b>		CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, SET, ...
<b>Liczbowe</b>	<i>Całkowite</i>	BIT, TINYINT, BOOL, BOOLEAN, SMALLINT, MEDIUMINT, INTEGER, BIGINT, ...
	<i>Zmiennoprzecinkowe</i>	FLOAT, DOUBLE, DECIMAL, ...
<b>Daty i czasu</b>		DATE, TIME, DATETIME, TIMESTAMP, YEAR
<b>Inne</b>		JSON

<https://dev.mysql.com/doc/refman/5.7/en/data-types.html>



# Typy danych w MySQL

Znakowe		CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, SET, ...
Liczbowe	<i>Całkowite</i>	TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, BIT(M)
	<i>Stałoprzecinkowe</i>	NUMERIC, DECIMAL
	<i>Zmiennoprzecinkowe</i>	FLOAT, DOUBLE
Daty i czasu		DATE, TIME, DATETIME, TIMESTAMP, YEAR
Inne		JSON

<https://dev.mysql.com/doc/refman/5.7/en/data-types.html>



# Typ stałoprzecinkowy



## Typy daty

Date – *YYYY-MM-DD* np. *2020-07-04*

Time – *hh:mm:ss* np. *09:00:00*

Datetime – *YYYY-MM-DD hh:mm:ss* np. *2020-07-04 09:00:00*

Timestamp – jak Datetime. Przechowuje dodatkowo informacje o strefie czasowej.



# Łańcuchy znaków

CHAR (N) – Blok pamięci o wielkości N znaków. Jeśli wstawimy krótszy łańcuch, uzupełniany jest spacjami.

Jeśli dłuższy, to jest ucinany do N znaków.

VARCHAR (N) – Zapis poprzedzony 1-2 bitowym prefixem. Brak uzupełniających spacji.

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
"	' '	4 bytes	"	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

<https://dev.mysql.com/doc/refman/5.7/en/char.html>





## BINARY VS CHAR

Zarówno typy CHAR i BINARY, jak i VARCHAR i VARBINARY są do siebie bardzo podobne. Różnica polega na sposobie zapisu.

BINARY has binary character set and collation.  
CHAR has non binary character set and collation.

COLLATION opisuje jak dane są sortowane i porównywane w bazie danych.  
Np. na podstawie polskiego alfabetu *Polish\_CI\_AS*



## BLOB

Typ danych do zapisu dużych plików binarnych.

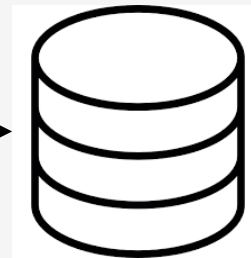
Zapis w postaci łańcucha bajtów, reprezentującego znaki oraz *collation*.

Porównywanie na podstawie wartości numerycznej bajtów.

Znalazł zastosowanie do przechowywania niewielkich plików graficznych.



`getBytes()`



## Podstawowe typy danych

Ważnym etapem projektowania schematu bazy danych jest ustalenie typów danych. Staramy się dobrać typ o minimalnej wielkości, który cechuje się oczekiwanym przez nas poziomem szczegółowości.

- Składowanie,
- Oszczędność pamięci RAM podczas joinów,
- Zmniejszenie ruchu sieciowego,
- Zwiększenie efektywności przechowywania wierszy na stronach – mniej operacji I/O,





# Podstawowe operacje

## Tworzenie bazy danych

Tabelę tworzy się za pomocą instrukcji

**CREATE DATABASE** nazwa\_db.

Przykład:

**CREATE DATABASE** *reading\_room*;



## 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 reading_room;
```



# Utworzenie użytkownika

Utworzenie użytkownika:

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

Przyznanie uprawnień do bazy:

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

Przeładowanie uprawnień – alternatywa dla restartu połączenia z bazą:

```
FLUSH PRIVILEGES;
```

**SHOW GRANTS** – wyświetla uprawnienia użytkowników.



# Tworzenie tabeli

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 tabeli

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  
);
```



## Auto-increment

Do kolumn całkowitoliczbowych możemy ustawić atrybut AUTO\_INCREMENT, który powoduje automatyczne ustawienie wartości pola z rosnącego ciągu.

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



## Tworzenie tabel c.d.

Ustawienie domyślnego zestawu znaków na UTF-8 i polskiego sortowania

```
CREATE TABLE `nazwa_tabeli` DEFAULT  
CHARACTER SET utf8 COLLATE utf8_polish_ci;
```





# Informacje o bazie

# Wyświetlenie listy baz danych

Polecenie:  
**SHOW DATABASES;**



## Wyświetlenie listy tabel

Po wyborze bazy (poleceniem USE):  
**SHOW TABLES;**



## Wyświetlenie schematu tabeli

Polecenie:

```
DESC nazwa_tabeli;
```





# Modyfikacja struktury



## 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;
```



# Modyfikacja tabeli

Przykład:

```
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 tabeli

Polecenie **DROP TABLE** służy do usuwania tabeli

**DROP TABLE** nazwa\_tabeli.

Przykład:

```
DROP TABLE books;
```



## Usuwanie tabeli - II

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

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



# TRUNCATE

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

```
TRUNCATE nazwa_tabeli;
```





# Manipulacja danymi

SQL DML

## 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

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

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

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');
```



## 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 books(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'); // BŁĄD!
```



## Ograniczenie CHECK

Instrukcja **CHECK** służy do ograniczenia wartości jakie mogą być wpisane do wybranej kolumny.

Przykład:

```
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'); // BŁĄD! Ale w MySQL ignorowane.
```



## 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  
);
```

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

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



## INSERT IGNORE

Zakładając, że mamy tabelę z wartością unikalnej i próbujemy dodać kolejny wpis, możemy uniknąć wywołania błędu do tabeli za pomocą słowa kluczowego **IGNORE**. Przykład:

```
CREATE TABLE przykladowa_tabela (  
    unikalna CHAR(10) UNIQUE,  
    kolumna INT  
);
```

```
INSERT INTO przykladowa_tabela(unikalna, kolumna) VALUES ('WARTOSC_1', 10); # OK
```

```
INSERT IGNORE INTO przykladowa_tabela(unikalna, kolumna) VALUES ('WARTOSC_1', 20); #  
ZIGNOROWANE
```





## ON DUPLICATE KEY UPDATE

Jeżeli dodamy rekord ze zduplikowaną wartością w kolumnie unikalnej, to możemy zaktualizować wartość tego pola, korzystając z instrukcji **ON DUPLICATE KEY UPDATE**:

```
CREATE TABLE przykladowa_tabela (  
    unikalna CHAR(10) UNIQUE,  
    kolumna INT  
);  
  
INSERT INTO przykladowa_tabela(unikalna, kolumna) VALUES ('WARTOSC_1', 10);  
INSERT INTO przykladowa_tabela(unikalna, kolumna) VALUES ('WARTOSC_1', 20)  
ON DUPLICATE KEY UPDATE kolumna = 20;
```



## 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
```



## DELETE

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

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



## DELETE

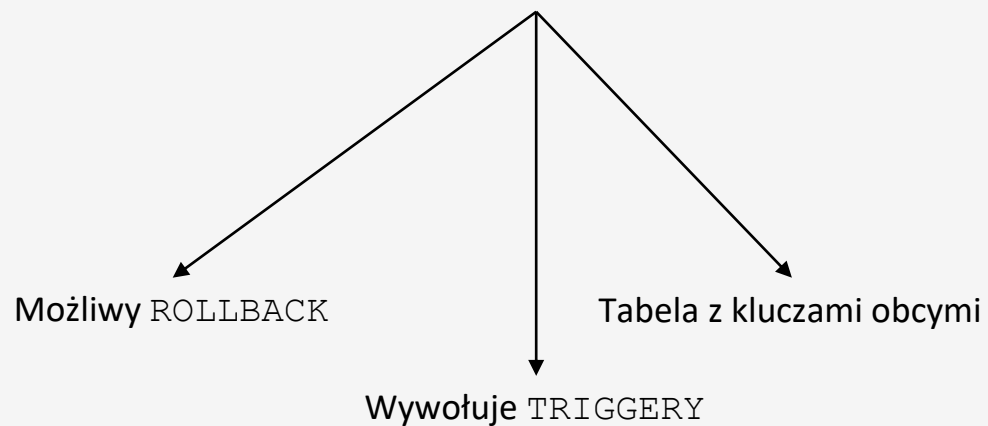
Przykład:

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

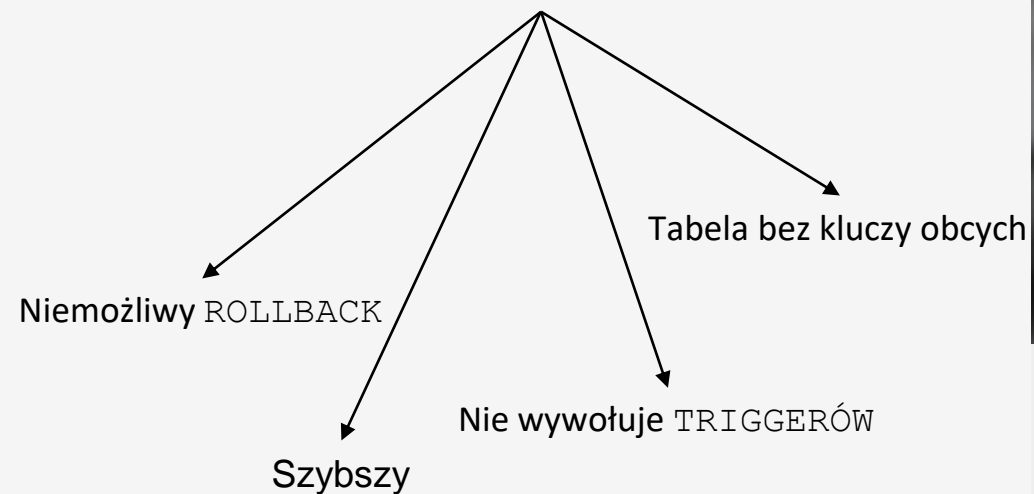


# DELETE vs TRUNCATE

DELETE FROM nazwa\_tabeli;



TRUNCATE nazwa\_tabeli;





# Pobieranie danych

SQL DQL

# 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 books;
```

```
SELECT * FROM books; //wyświetla wszystkie kolumny
```



## ALIAS

```
SELECT column_name AS alias_name  
FROM table_name;
```

```
SELECT column_name, ...  
FROM table_name AS alias_name;
```





## Instrukcja WHERE

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

Przykład:

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



## WHERE - operator

### Operatory algebraiczne

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

## WHERE - operator

Operatory algebraiczne – przykłady

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

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



# WHERE - operatory

## Operatory logiczne

Symbol	Opis
NOT	negacja
OR	suma logiczna
<b>AND</b>	<b>Iloczyn logiczny</b>

Priorytet operatorów logicznych:  
NOT > AND > OR

Operatory algebraiczne mają  
wyższy priorytet niż logiczne!



## WHERE - operator

Operatory logiczne – przykłady

```
SELECT * FROM books WHERE page_count > 1000 AND category  
= 'bazy danych';
```

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



## WHERE - operators

### 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	<b>podobny do wzorca</b>

# WHERE - operator

Operatory specjalne – przykłady

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

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

```
SELECT * FROM books WHERE title LIKE 'MongoD_ w akcji';
```

```
SELECT * FROM books WHERE title LIKE '%akcji%';
```



## 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





## 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

## Funkcje czasu

Przykład:

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

+	-----	+	-----	+	-----	+
	NOW ()		CURDATE ()		CURTIME ()	
+	-----	+	-----	+	-----	+
	2017-04-16 13:40:17		2017-04-16		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

  
[https://dev.mysql.com/doc/refman/5.7/en/string-functions.html#function\\_format](https://dev.mysql.com/doc/refman/5.7/en/string-functions.html#function_format)

---

## Funkcje czasu

Przykład:

```
SELECT title FROM books WHERE  
lower(author)='krzysztof barteczko';
```

```
+-----+  
| title |  
+-----+  
| Java Programowanie praktyczne od podstaw |  
| Java. Uniwersalne techniki programowania |  
+-----+
```



## Wybrane funkcje numeryczne

Symbol	Opis
ROUND()	zaokrąglanie liczby
RAND()	losowa liczba zmiennoprzecinkowa (od 0 do 1)
PI()	liczba pi
FLOOR()	zaokrąglanie w dół
CEIL()	zaokrąglanie w górę
TRUNCATE()	obcinanie liczby do określonej liczby miejsc po przecinku
+, -, *, /	standardowe operacje na liczbach

<https://dev.mysql.com/doc/refman/5.7/en/numeric-functions.html>

## Funkcje czasu

Przykład:

```
SELECT PI ( ) , ROUND ( PI ( ) ) , FLOOR ( PI ( ) ) , CEIL ( PI ( ) ) ;
```

PI ( )	ROUND ( PI ( ) )	FLOOR ( PI ( ) )	CEIL ( PI ( ) )
3.141593	3	3	4



## SELECT DISTINCT

Instrukcja **SELECT DISTINCT** jest używana do zwracania tylko 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;
```



## SELECT DISTINCT

Przykład:

```
SELECT DISTINCT publisher FROM books;
```





## LIMIT

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

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



## 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;
```



## FUNKCJE AGREGUJĄCE

Funkcje agregujące pozwalają na wykonywanie na wynikach zapytań funkcji matematycznych.

Symb ol	Opis
AVG	średnia wszystkich wartości
SUM	suma wszystkich wartości
MAX	maksymalna wartość
<b>MIN</b>	<b>minimalna wartość</b>

# FUNKCJE AGREGUJĄCE

Funkcje agregujące – przykłady

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



## 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];
```



## PODZAPYTANIA

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

Przykład:

```
SELECT * FROM books  
WHERE price =  
(SELECT min(price) FROM books);
```



## PODZAPYTANIA

Podzapytania dzielimy na zagnieżdżone – najpierw wykonywane jest zapytanie wewnętrzne, którego wartości wynikowe są podstawiane do zapytania zewnętrznego oraz skorelowane, w którym dla każdego wiersza z wyniku zapytania zewnętrznego jest wykonywane zapytanie wewnętrzne.

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary >
  (SELECT AVG(salary)
   FROM employees
   WHERE department_id = outer.department_id);
```

Podzapytanie skorelowane



```
SELECT last_name, salary, department_id
FROM employees
WHERE salary >
  (SELECT AVG(salary)
   FROM employees
   WHERE department_id = 1);
```

Podzapytanie  
zagnieżdżone

## HAVING

Klauzula **HAVING** została dodana do SQL, ponieważ nie można było użyć słowa kluczowego WHERE z funkcjami agregującymi.

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





## 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);
```



## Operatory ALL oraz ANY

Operator **ANY** i **ALL** jest używany z klauzulą WHERE lub HAVING.

Każdy operator zwraca wartość true, jeśli którykolwiek z podzapytań spełnia warunek.

Operator ALL zwraca wartość true, jeśli wszystkie podzapytania spełniają warunek.



## Operatory ALL oraz ANY

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

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



## INSERT INTO SELECT

Instrukcja **INSERT INTO SELECT** kopiuje dane z jednej tabeli i wstawia ją do innej tabeli.

**INSERT INTO SELECT** wymaga zgodności typów danych w tabelach źródłowych i docelowych.

Istniejące rekordy w tabeli docelowej nie są naruszone.



## INSERT INTO SELECT

Polecenie SQL wstawiające wszystkie kolumny do nowej tabeli:

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



## INSERT INTO SELECT

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

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



## UNION

Operator **UNION** służy do łączenia zestawu wyników dwóch lub więcej instrukcji SELECT.

- Każda instrukcja SELECT w ramach UNION musi mieć taką samą liczbę kolumn
- Kolumny muszą również zawierać podobne typy danych
- Kolumny w każdej instrukcji SELECT muszą być również w tej samej kolejności



## UNION

```
SELECT column_name, ... FROM table1  
UNION  
SELECT column_name, ... FROM table2;
```





## UNION ALL

Operator UNION domyślnie wybierze tylko różne wartości. Aby umożliwić powielanie wartości, użyj **UNION ALL**:

```
SELECT column_name, ... FROM table1  
UNION ALL  
SELECT column_name, ... FROM table2;
```





# Relacje w bazach danych

Więzy integralności

# Więzy integralności

**Primary Key (klucz główny)** - zapewnia unikalność wartości w kolumnie. Najczęściej zakładany jest na kolumnę która przechowuje dane jednoznacznie określające pojedynczy wiersz. W tabeli może być tylko jeden klucz główny. Zapewnia nie występowanie wartości NULL.

**Unique (unikalność)** - Zapewnia unikalność wartości w kolumnie, jednak w przeciwieństwie do PRIMARY KEY takich kluczy może być więcej niż jeden, oraz umożliwia występowanie wartości NULL.

**NOT NULL** - Zapobiega wstawianiu wartości NULL do kolumny.

**Foreign Key (klucz obcy)** - Służy do definiowania relacji pomiędzy tabelami. Zapewnia że rekord w tabeli podrzędnej zawsze będzie miał swojego odpowiednika w tabeli nadrzędnej. Klucz obcy musi się odwoływać do kolumny (kolumn) w tabeli nadrzędnej, na których założony jest **UNIQUE** lub klucz główny.

**CHECK** – wartość musi spełniać warunek logiczny



## PRIMARY KEY

Ograniczenie **PRIMARY KEY** jednoznacznie identyfikuje każdy rekord w tabeli bazy danych.

Klucze podstawowe muszą zawierać wartości UNIQUE i nie mogą zawierać wartości NULL.


Tabela może zawierać tylko jeden klucz podstawowy, który może składać się z pojedynczych lub wielu pól.



## PRIMARY KEY

Następujący SQL tworzy klucz główny w kolumnie "id", gdy tworzona jest osoba:

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



## PRIMARY KEY

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

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



## FOREIGN KEY

Klucz obcy jest kluczem służącym do łączenia dwóch tabel.


Klucz obcy w tabeli wskazuje na PRIMARY KEY w innej tabeli.



## FOREIGN KEY

Następujący SQL tworzy klucz obcy w kolumnie "author\_id", gdy tworzona jest tabela "books":

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





# Więzy integralności

**ON DELETE** – akcja wykonywana przy usuwaniu danego wiersza

**ON UPDATE** – akcja wykonywana przy modyfikacji danego wiersza

**NO ACTION** - wyłącza mechanizm klucza obcego dla danej operacji (w przypadku MySQL to samo co RESTRICT)

**RESTRICT** - nie pozwala na dokonanie zmian naruszających powiązanie

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

**SET NULL** - ustawia odpowiednie atrybuty powiązanych tabel, dotąd wskazujące na usuwany/modyfikowany element klucza obcego, na wartość NULL, jeśli definicja tabeli to dopuszcza

**SET DEFAULT** - ustawia odpowiednie atrybuty powiązanych tabel, dotąd wskazujące na usuwany/modyfikowany element klucza obcego, na wartość DEFAULT, jeśli definicja tabeli to dopuszcza



## FOREIGN KEY

Przykład z wykorzystaniem usuwania i aktualizacji kaskadowej:

```
CREATE TABLE books (  
  id INT NOT NULL,  
  title VARCHAR(128) NOT NULL,  
  author_id INT,  
  ...  
  PRIMARY KEY (id),  
  FOREIGN KEY (author_id) REFERENCES authors(id)  
  ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=INNODB;
```





# Projektowanie

# Projektowanie

**Proces projektowania bazy danych możemy podzielić na kilka etapów:**

1. Określenie celu, któremu ma służyć baza danych (Cel bieżącego zadania: 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



## ERD

Rodzaj graficznego przedstawienia związków pomiędzy encjami używanymi w projektowaniu systemów informacyjnych do przedstawienia conceptualnych modeli danych używanych w systemie.

Istnieją różne modele przedstawiania schematów. Wykorzystujemy notację **Martina**.



## ERD

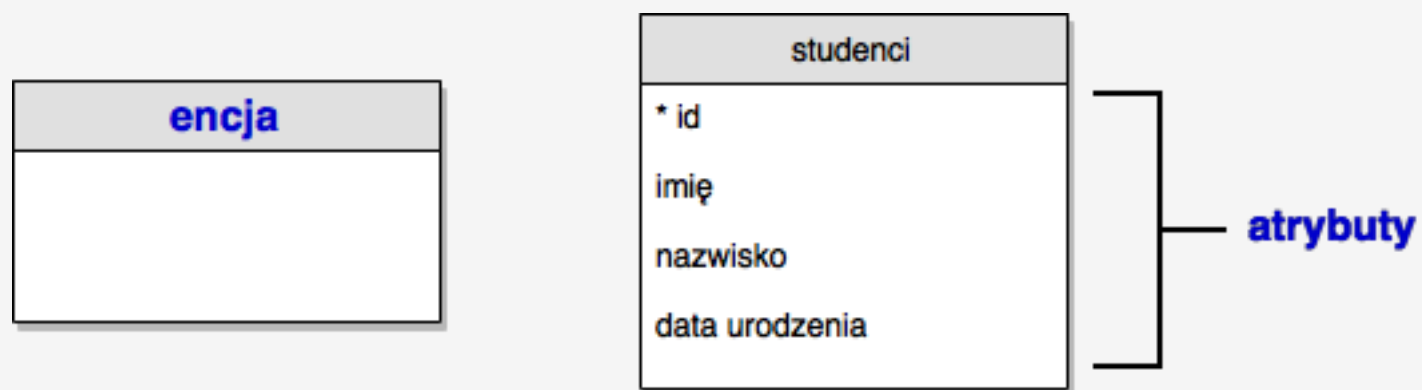
### Definicje

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



# ERD

## Elementy diagramów ERD



# ERD

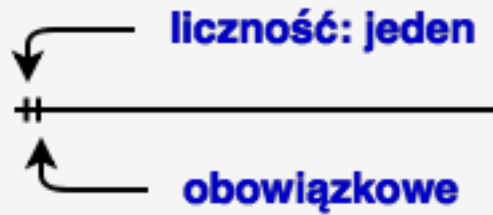
## Liczność w ERD



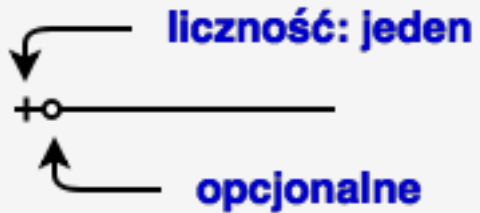


# ERD

## Reprezentacja związków w ERD



powiązanie jednokrotne, obowiązkowe (tylko jeden)

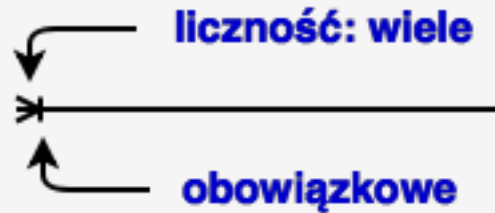


powiązanie jednokrotne, opcjonalne (jeden lub zero)

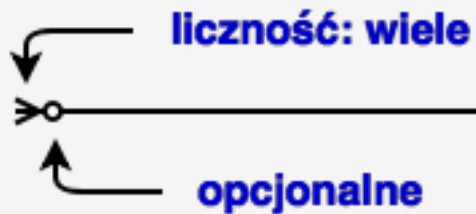


# ERD

## Reprezentacja związków w ERD



powiązanie wielokrotne, obowiązkowe (co najmniej jeden)

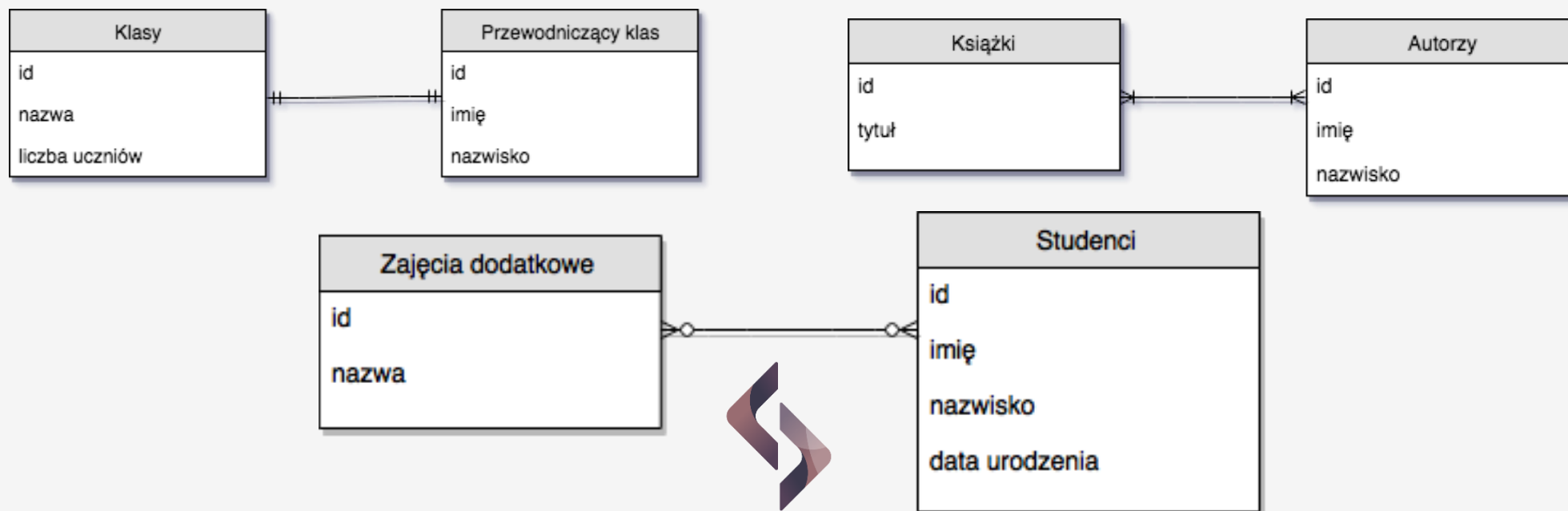


powiązanie wielokrotne, opcjonalne (zero lub więcej)



# ERD

## Przykłady relacji:





# Projektowanie

Postacie normalne

## I Postać normalna

1 NF mówi o atomowości danych. Tabela jest w pierwszej postaci normalnej, jeśli wiersz przechowuje informację o pojedynczym obiekcie, nie zawiera kolekcji, posiada klucz główny, a dane są atomowe.



# I Postać normalna

Tabela przed przeprowadzeniem procesu normalizacji

nr_zamowienia	nazwa_klienta	adres_klienta	data_zamowienia	szczegoly_zamowienia
200	Jan Kowalski	Łódź, 90-273 ul. Rewolucji 1905r 11 m 33, woj. łódzkie	2016-06-01	Buty 250 PLN; Koszulka 100 PLN
203	Jan Kowalski	Łódź, 90-273 ul. Rewolucji 1905r 11 m 33, woj. łódzkie	2016-06-05	Piłka 70 PLN
204	Jan Kowalski	Łódź, 90-273 ul. Rewolucji 1905r 11 m 33, woj. łódzkie	2016-06-08	Skarpety 5 PLN; Hulajnoga 100 PLN
202	Kamil Wąs	Warszawa, 02-290 ul. Hoża 1 m 56; woj. mazowieckie	2016-06-02	Narty 1200 PLN
201	Piotr Nowak	Kraków, 22-178 ul. Prosta 12 m 1; woj. małopolskie	2016-06-02	Narty 1200 PLN



# I Postać normalna

Tabela po przeprowadzeniu procesu normalizacji

nr_pozycji	numer_zamowienia	nazwa_klienta	adres	kod_poczty	województwo	data_zamowienia	element_zamowienia	ilosc	cena_jedn	wartosc_zamowienia	vat	miasto
1	200	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	2016-06-01	Buty	1	250	250	23	Łódź
2	200	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	2016-06-01	Koszulka	2	50	100	23	Łódź
3	203	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	2016-06-05	Piłka	1	70	70	23	Łódź
4	204	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	2016-06-08	Skarpetki	3	5	15	23	Łódź
5	204	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	2016-06-08	Hulajnoga	1	100	100	23	Łódź
6	202	Kamil Wąs	ul. Hoża 1 m 56	02-290	mazowiecki	2016-06-02	Narty	1	1200	1200	23	Warszawa
7	201	Piotr Nowak	ul. Prosta 12 m 1	22-178	małopolska	2016-06-02	Narty	1	1200	1200	23	Kraków

## II Postać normalna

Relacja jest w drugiej postaci normalnej wtedy i tylko wtedy, gdy jest w pierwszej postaci normalnej oraz każdy atrybut odnosi się do opisu konkretnej klasy.





## II Postać normalna

Tabela przed przeprowadzeniem procesu normalizacji

nr_pozycji	numer_zamowienia	nazwa_klienta	adres	miasto	województwo	data_zamowienia	element_zamowienia	ilosc	cena_jednosc	wartosc_zamowienia	vat
1	200	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	Łódź	łódzkie	2016-06-01	Buty	1	250	250	23
2	200	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	Łódź	łódzkie	2016-06-01	Koszulka	2	50	100	23
3	203	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	Łódź	łódzkie	2016-06-05	Piłka	1	70	70	23
4	204	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	Łódź	łódzkie	2016-06-08	Skarpetki	3	5	15	23
5	204	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	Łódź	łódzkie	2016-06-08	Hulajnoga	1	100	100	23
6	202	Kamil Wąs	ul. Hoża 1 m 56	Warszawa	mazowieckie	2016-06-02	Narty	1	1200	1200	23
7	201	Piotr Nowak	ul. Prosta 12 m 1	Kraków	małopolska	2016-06-02	Narty	1	1200	1200	23

## II Postać normalna

Tabela po przeprowadzeniu procesu normalizacji

ZAMOWIENIA				
numer_zamowienia	id_klienta	data_zamowienia	wartosc_zamowienia	vat
200	1	2016-06-01	350	23
201	3	2016-06-02	1200	23
202	2	2016-06-02	1200	23
203	1	2016-06-05	70	23
204	1	2016-06-08	115	23

KLIENCI					
id_klienta	nazwa_klienta	adres	kod_pocztowy	województwo	miasto
1	Jan Kowalski	ul. Rewolucji 1905r 11 m 33	90-273	łódzkie	Łódź
2	Kamil Wąs	ul. Hoża 1 m 56	02-290	mazowieckie	Warszawa
3	Piotr Nowak	ul. Prosta 12 m 1	22-178	małopolska	Kraków



## II Postać normalna

Tabela po przeprowadzeniu procesu normalizacji

PRODUKTY		
kod_produktu	nazwa	cena_jednostkowa
1	Buty	250
2	Koszulka	50
3	Piłka	70
4	Skarpetki	5
5	Hulajnoga	100
6	Narty	1200

SZCZEGOLY_ZAMOWIEN		
numer_zamowienia	kod_produktu	ilosc
200	1	1
200	2	2
201	6	1
202	6	1
203	3	1
204	4	3
204	5	1



### III Postać normalna

Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w drugiej postaci normalnej oraz gdy każda kolumna informacyjna nie należąca do klucza nie zależy od innej kolumny informacyjnej.



## III Postać normalna

ZAMOWIENIA					
numer_zamowienia	id_klienta	data_zamowienia	wartosc_zamowienia	vat	wartosc_brutto
200	1	2016-06-01	350	23	430,5
201	3	2016-06-02	1200	23	1476
202	2	2016-06-02	1200	23	1476
203	1	2016-06-05	70	23	86,1
204	1	2016-06-08	115	23	141,45



### III Postać normalna

ZAMOWIENIA					
numer_zamowienia	id_klienta	data_zamowienia	wartosc_zamowienia	vat	wartosc_brutto
200	1	2016-06-01	350	23	430,5
201	3	2016-06-02	1200	23	1476
202	2	2016-06-02	1200	23	1476
203	1	2016-06-05	70	23	86,1
204	1	2016-06-08	115	23	141,45

ZAMOWIENIA					
numer_zamowienia	id_klienta	data_zamowienia	wartosc_zamowienia	vat	
200	1	2016-06-01	350	23	
201	3	2016-06-02	1200	23	
202	2	2016-06-02	1200	23	
203	1	2016-06-05	70	23	
204	1	2016-06-08	115	23	





# Pobieranie danych

## złączenia tabel

# Złączenia

Złączenia wykorzystywane są do łączenia danych z dwóch lub więcej tabel.

- CROSS JOIN
- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN







## Visual Explanation of SQL Join Clauses

### Common Join Clauses

#### Inner Join

Return rows where key values intersect



```
SELECT A.col, B.col
FROM A
INNER JOIN B
ON A.Key = B.Key
```

#### Left Outer Join

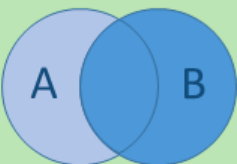
Return all rows from A, match B, when no match found, return NULLS for columns in B.



```
SELECT c1, c2
FROM A
LEFT OUTER JOIN B
ON JOIN A.Key = B.Key
```

#### Right Outer Join

Return all rows from B, match A, when no match found, return NULLS for columns in A.

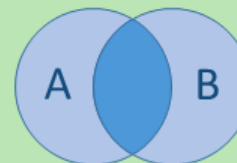


```
SELECT c1, c2
FROM A
RIGHT OUTER JOIN B
ON JOIN A.Key = B.Key
```

### Exotic Join Clauses

#### Full Join

Left and Right Join Combined – Return all rows that Match A and B, when no match, found, return nulls for that table



```
SELECT c1, c2
FROM A
FULL OUTER JOIN B
ON JOIN A.Key = B.Key
```

#### Minus (A - B)

Return row from A only if key value is exclusive to A.



```
SELECT c1, c2
FROM A
LEFT OUTER JOIN B
ON A.Key = B.Key
WHERE B.Key is NULL
```

#### Minus (B - A)

Return row from A only if key value is exclusive to B.



```
SELECT c1, c2
FROM A
RIGHT OUTER JOIN B
ON JOIN A.Key = B.Key
WHERE A.Key is NULL
```



# Indeksy

## Wyszukiwanie danych bez indeksów

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 – zalety i wady

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

Zysk szybkości wymaga pewnego kompromisu – jest nim utrata miejsca na dysku. Dlatego warto indeksować tyle te kolumny, które będziesz często przeszukiwał.



## Indeksy - wskazówki

- Przeanalizuj kolumny najczęściej występujące w zapytaniach SQL w poleceniach `WHERE`, `ORDER BY`, `GROUP BY`.
- Staraj się tworzyć wąskie indeksy (zawierające minimalną wymaganą ilość kolumn). Im bardziej złożony indeks, tym więcej miejsca zajmuje na dysku.
- Usuвай nieużywane indeksy.



## Indeksy - wskazówki

- W wielu przypadkach należy sobie odpowiedzieć na pytanie czy korzyści płynące z przyspieszenia odczytu danych po stworzeniu indeksu równoważą koszty związane ze spowolnieniem operacji modyfikujących.
- Zdecydowanie lepiej jest indeksować kolumny numeryczne w odróżnieniu do tekstowych, ponieważ pola numeryczne zajmują mniej pamięci.
- Nie ma sensu tworzyć indeksu na kolumnie która ma małe zróżnicowanie wartości (np. pola logiczne, płeć, stan cywilny, itp.).



## Indeksy - wskazówki

Kolejność indeksowania kolumn indeksu kompozytowego (składającego się z wielu kolumn) ma kluczowe znaczenie dla wykorzystania indeksu. W pierwszej kolejności z lewej strony w indeksie powinny występować kolumny o największym zróżnicowaniu wartości (np. indeks złożony z 3 kolumn o definicji: id\_klienta - miasto - płeć jest zdecydowanie lepiej zdefiniowany niż: płeć - id\_klienta - miasto ).



# Indeksy

## Przykład:

```
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);
```





# Indeksy

Usuwanie indeksów:

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



## Polecenie EXPLAIN

Pytanie, czy zapytanie jest optymalne?  
Polecenie EXPLAIN udostępnia informacje na temat wykorzystania indeksów, liczby przeszukanych dokumentów, itp.  
Poprzedzamy po prostu zapytanie słowem EXPLAIN.  
**EXPLAIN SELECT ...**



## Sugerowanie indeksów

Wykorzystanie/odrzućenie konkretnych indeksów w zapytaniu:

```
SELECT * FROM table1 USE INDEX  
(col1_index,col2_index)  
WHERE col1=1 AND col2=2 AND col3=3;
```

```
SELECT * FROM table1 IGNORE INDEX (col3_index)  
WHERE col1=1 AND col2=2 AND col3=3;
```





# Transakcije

# Transakcje

Sekwencja (uporządkowany zbiór) logicznie powiązanych operacji na bazie danych, która przeprowadza bazę danych z jednego stanu spójnego w inny stan spójny.



# Transakcje

SAVEPOINT

Rozpoczęcie

BEGIN

Wykonanie

Zamknięcie

COMMIT,  
ROLLBACK



## ACID

**ACID** jest skrótem od angielskich słów atomicity, consistency, isolation, durability, czyli niepodzielność, spójność, izolacja, trwałość.

Jest to zbiór właściwości gwarantujących poprawne przetwarzanie transakcji w bazach danych.



## ATOMICITY

Niepodzielność transakcji oznacza jej wykonanie jako całości. Gdy transakcja powoduje wiele zmian w bazie danych możliwe są jedynie dwie sytuacje:

- wszystkie zmiany zostaną wykonane gdy transakcja zostanie zatwierdzona (instrukcja COMMIT)
- wszystkie zmiany zostaną wycofane (instrukcja ROLLBACK)





## CONSISTENCY

Spójności bazy jest zachowana przez cały czas. Zmiana stanu bazy może dokonać się jedynie za pomocą instrukcji COMMIT lub ROLLBACK. Jeśli w danej chwili jakaś transakcja zmienia dane w jednej lub kilku tabelach inne działające w tym samym czasie transakcje zobaczą wszystkie nie zaktualizowane dane lub wszystkie nowe dane a nie zaś mieszankę nowych i starych danych.



## ISOLATION

Transakcje są zabezpieczone (odizolowane od siebie). Oznacza to, że podczas dziania nie mogą kolidować ze sobą lub zobaczyć nawzajem

niezatwierdzonych danych. Każda transakcja widzi bazę tak jakby była jedyną wykonywaną w danym czasie.

Izolacja jest osiągana za pomocą blokad. Zaawansowani użytkownicy bazy danych mogą dostosować poziom izolacji, zmniejszając je izolacji na rzecz zwiększenia wydajności i współbieżności.



## DURABILITY

Rezultat wykonania transakcji musi być trwały. Jeśli operacja COMMIT powiodła się, jej rezultaty nie mogą ulec zmianie w przypadku awarii systemu baz danych.





# silniki bazy danych MySQL

# MyISAM

Prostszy silnik, przez co dla początkujących łatwiejszy przy projektowaniu, ale:

- Brak obsługi transakcji
- Brak kluczy obcych i kaskadowego usuwania/aktualizacji wierszy
- Brak integralności (zgodności z ACID)
- Brak rollbacków (cofania operacji w transakcji)
- Ograniczenie liczby wierszy do 4 284 867 296 ( $2^{32}$ )
- Maksymalnie 64 indeksy na wiersz
- Blokada na poziomie tabeli

**Szybki i mało zasobożerny.**



## InnoDB

Nowsze rozwiązanie, większe możliwości, przez co nieco bardziej skomplikowane:

- Obsługa transakcji (ACID)
- Blokada przy zapisie na poziomie wiersza (co pozwala na lepsze zrównoleglenie)
- Obsługa kluczy obcych
- Bardziej odporny na uszkodzenie danych

Wolniejszy i bardziej zasobożerny. Nowszy i w przeciwieństwie do MyISAM wciąż rozwijany.  
Od 5.6 wsparcie dla wyszukiwania pełnotekstowe.



# Memory

Prosty silnik, mało funkcji, ale bardzo szybki. Dane przechowywane w pamięci RAM.

Porównanie silników w postaci tabelki:

<http://www.thegeekstuff.com/2014/02/myisam-innodb-memory/>



## Tworzenie tabel z wyborem silnika

```
CREATE TABLE nazwa_tabeli(...) ENGINE=typ;  
Gdzie typ, to InnoDB, MyISAM lub Memory.
```







Widoki

## Co to jest i po co

Gdy często korzystamy z jakichś danych na podstawie zapytania, możemy utworzyć tymczasową tabelę zwaną widokiem.



## Tworzenie widoków

```
CREATE VIEW sql_books AS  
SELECT title, page_count FROM books  
WHERE category = "bazy sql";
```



## Wykorzystanie

```
SELECT COUNT(*) AS book_count, SUM(pages) AS  
total_pages FROM books;
```



## Usunięcie widoków

Polecenie:

```
DROP VIEW sql_books;
```





# Procedury składowane

## Co to jest ?

Procedury składowane dodają wiele elementów programowania proceduralnego takich jak pętle, wyrażenia warunkowe itp., które rozszerzają standard SQL. Procedury składowane pozwalają na wykonanie złożonych obliczeń, które nie byłyby możliwe do wykonania przy pomocy zapytań SQL.



## Procedury składowane - zalety

- Redukcja ilości zapytań wykonanych przez aplikację do bazy danych ze względu na przeniesienie części logiki aplikacji do serwera bazy danych.
- Poprawa wydajności aplikacji ze względu na przeniesienie pewnych funkcjonalności w postaci prekompilowanego kodu do serwera bazy danych.
- Możliwość ponownego użycia funkcji w innych aplikacjach.





## Procedury składowane - wady

- Trudne w rozwijaniu ze względu na to, że wymagają od programisty znajomości dodatkowego języka.
- Trudne w debugowaniu.
- Może być nie przenaszalny na inny serwer bazy danych.



## Procedura - szablon

```
delimiter //  
CREATE PROCEDURE nazwa (IN param1 typ, IN param2 typ)  
BEGIN  
    # kod procedury  
END//  
delimiter ;
```



## Funkcja - szablon

```
delimiter //  
CREATE FUNCTION nazwa (param1 typ, param2 typ)  
RETURNS typ  
BEGIN  
    # kod funkcji  
    RETURN wynik;  
END//  
delimiter ;
```

```
CREATE FUNCTION nazwa (param1 typ, param2 typ)  
RETURNS typ  
RETURN wynik wykonania funkcji;
```



Usunięcie funkcji/procedury:

```
DROP PROCEDURE nazwa_procedury;  
DROP FUNCTION nazwa_funkcji;
```





# Triggery

# Triggery

Są to instrukcje, które pozwalają nam na wykonywanie pewnych operacji w trakcie wywołania pewnych instrukcji. Wyzwalacze pozwalają zmianę wartości kolumn w odpowiedzi na instrukcję DML, czyli INSERT, UPDATE, DELETE wykonaną na tabeli lub widoku.

```
delimiter //  
CREATE TRIGGER nazwa_triggera [AFTER/BEFORE] [DELETE/INSERT/UPDATE] ON nazwa_tabeli FOR EACH ROW  
BEGIN  
    # operacje  
END//
```

```
delimiter ;
```

```
CREATE TRIGGER nazwa_triggera [AFTER/BEFORE] [DELETE/INSERT/UPDATE] ON nazwa_tabeli FOR EACH  
# operacja
```





# Zarządzanie procesami w MySQL

## Zarządzanie procesami

Czasem, szczególnie przy wykonywaniu długotrwałych zapytań, istnieje potrzeba sprawdzenia jakie zapytania wykonują się aktualnie na serwerze i ewentualnie zakończenia ich działania.





# Wyświetlenie listy procesów

Służy do tego polecenie:  
SHOW PROCESSLIST;

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
6	user	localhost:62244	mojadb	Sleep	6		NULL
41234	root	localhost	mojadb	Query	0	executing	select * from tabela where nazwaKolumny = 'jakas_nazwa'



## Przerwanie działania procesu

Służy do tego polecenie:  
`KILL id_procesu;`





**Kopie zapasowe**

# Tworzenie kopii zapasowej danych

## Polecenie mysqldump

```
mysqldump -u user -p[hasło] [nazwa_bazy] >  
plik_zrzutu.sql
```



## Odzyskiwanie danych z kopii zapasowej

W terminalu wykonujemy:

```
mysql -u user -p[hasło] [nazwa_bazy] < plik_zrzutu.sql
```





# Ciekawostki

## Normalizacja cd.

Poza 3 poziomami normalizacji wg kryteriów Codd'a, istnieją także mniej popularne, wyższe poziomy:

- Postać normalna Boyda-Codd'a, tzw. 3,5NF
- 4 postać normalna 4NF
- 5 postać normalna 5NF



## Model fizyczny baz danych

Omawiane przez nas pojęcia tabel, wierszy i kolumn odwołują się do logicznego modelu bazy danych. Jest to abstrakcja, fizycznego sposobu przechowywania danych na nośniku (najczęściej dysku). Podstawowy model dyskowy zakłada przechowywanie *tabel* przez *plik*, który składa się ze *stron*, a te z *rekordów*, posiadających *pola*.

- Mechanizm miejscem na dysku
- Mechanizm buforowania danych w RAM





## Bazy danych in-memory

Dostępne są implementacje SZBD przechowywane całkowicie w pamięci RAM (np. H2). Jedną z zalet takich rozwiązań jest brak konieczności stosowania indeksów, ponieważ dostęp do każdego z wierszy jest realizowany w stałym czasie.

