SQL W JEDEN DZIEŃ

PRAKTYCZNY KURS JĘZYKA SQL

Michael Robinson

SPIS TREŚCI

1. Dlaczego SQL jest ważny? 5

1.1. Rola SQL w nowoczesnym rozwoju oprogramowania 6

1.2. Porównanie SQL z innymi technologiami 9

1.3. Szybki start: Pierwsze zapytanie SELECT 12

2. Podstawy: Struktura bazy i typy danych 16

2.1. Tabele, kolumny i wiersze 17

2.2. Klucze podstawowe i obce ..... **Błąd! Nie zdefiniowano zakładki.**

2.3. Najważniejsze typy danych SQL i ich zastosowanie ..... **Błąd! Nie zdefiniowano zakładki.**

2.4. Praktyka: Projektowanie prostej schematu bazy danych ..... **Błąd! Nie zdefiniowano zakładki.**

3. Tworzenie i modyfikacja tabel ..... **Błąd! Nie zdefiniowano zakładki.**

3.1. Składnia CREATE TABLE z przykładami ..... **Błąd! Nie zdefiniowano zakładki.**

3.2. Dodawanie, modyfikacja i usuwanie kolumn za pomocą ALTER TABLE ..... **Błąd! Nie zdefiniowano zakładki.**

3.3. Usuwanie tabel z DROP TABLE..... **Błąd! Nie zdefiniowano zakładki.**

3.4. Ćwiczenie: Stwórz schemat bazy danych dla sklepu internetowego ..... **Błąd! Nie zdefiniowano zakładki.**

4. Wstawianie danych: INSERT INTO ..... **Błąd! Nie zdefiniowano zakładki.**

4.1. Podstawowa składnia INSERT INTO. **Błąd! Nie zdefiniowano zakładki.**

4.2. Wstawianie wielu wierszy jednocześnie..... **Błąd! Nie zdefiniowano zakładki.**

4.3. INSERT INTO SELECT: Kopiowanie danych między tabelami .. **Błąd! Nie zdefiniowano zakładki.**

4.4. Praktyka: Wypełnianie tabel sklepu internetowego danymi.. **Błąd! Nie zdefiniowano zakładki.**

5. Pobieranie danych: SELECT i klauzula WHERE ..... **Błąd! Nie zdefiniowano zakładki.**

5.1. Anatomia zapytania SELECT..... **Błąd! Nie zdefiniowano zakładki.**

5.2. Filtrowanie z WHERE: operatory porównania i logiczne ..... **Błąd! Nie zdefiniowano zakładki.**

5.3. Zaawansowane filtrowanie: LIKE, IN, BETWEEN..... **Błąd! Nie zdefiniowano zakładki.**

5.4. Ćwiczenie: Tworzenie raportów dla sklepu internetowego ... **Błąd! Nie zdefiniowano zakładki.**

6. Agregacja danych: GROUP BY i funkcje agregujące **Błąd! Nie zdefiniowano zakładki.**

6.1. Funkcje COUNT, SUM, AVG, MAX, MIN..... **Błąd! Nie zdefiniowano zakładki.**

6.2. Grupowanie wyników z GROUP BY.. **Błąd! Nie zdefiniowano zakładki.**

6.3. Filtrowanie grup z HAVING ..... **Błąd! Nie zdefiniowano zakładki.**

6.4. Ćwiczenie: Analiza sprzedaży w sklepie internetowym..... **Błąd! Nie zdefiniowano zakładki.**

7. Modyfikacja i usuwanie danych: UPDATE i DELETE **Błąd! Nie zdefiniowano zakładki.**

7.1. Składnia UPDATE z przykładami..... **Błąd! Nie zdefiniowano zakładki.**

7.2. Masowe aktualizacje z JOIN..... **Błąd! Nie zdefiniowano zakładki.**

7.3. Bezpieczne usuwanie z DELETE..... **Błąd! Nie zdefiniowano zakładki.**

7.4. Praktyka: Aktualizacja cen i usuwanie nieaktualnych produktów. **Błąd! Nie zdefiniowano zakładki.**

8. Indeksy: Optymalizacja wydajności zapytań..... **Błąd! Nie zdefiniowano zakładki.**

8.1. Tworzenie indeksów: składnia CREATE INDEX. **Błąd! Nie zdefiniowano zakładki.**

8.2. Typy indeksów i kiedy ich używać.... **Błąd! Nie zdefiniowano zakładki.**

8.3. Analiza planu wykonania zapytania. **Błąd! Nie zdefiniowano zakładki.**

8.4. Ćwiczenie: Optymalizacja zapytań w sklepie internetowym.. **Błąd! Nie zdefiniowano zakładki.**

9. Widoki i procedury składowane: Reużywalność kodu SQL..... **Błąd! Nie zdefiniowano zakładki.**

9.1. Tworzenie widoków dla często używanych zapytań ..... **Błąd! Nie zdefiniowano zakładki.**

9.2. Pisanie i wywoływanie procedur składowanych ..... **Błąd! Nie zdefiniowano zakładki.**

9.3. Funkcje zdefiniowane przez użytkownika ..... **Błąd! Nie zdefiniowano zakładki.**

9.4. Ćwiczenie: Tworzenie raportów i automatyzacja procesów .. **Błąd! Nie zdefiniowano zakładki.**

10. Najczęstsze pułapki i jak ich unikać ..... **Błąd! Nie zdefiniowano zakładki.**

10.1. Problemy z wydajnością: N+1 zapytania ..... **Błąd! Nie zdefiniowano zakładki.**

10.2. Nieprawidłowe używanie indeksów ..... **Błąd! Nie zdefiniowano zakładki.**

10.3. Błędy w złączeniach tabel ..... **Błąd! Nie zdefiniowano zakładki.**

10.4. Ćwiczenie: Debugowanie i optymalizacja problematycznych zapytań ..... **Błąd! Nie zdefiniowano zakładki.**

11. Narzędzia i zasoby do dalszej nauki..... **Błąd! Nie zdefiniowano zakładki.**

11.1. Popularne systemy zarządzania bazami danych: MySQL, PostgreSQL, SQLite..... **Błąd! Nie zdefiniowano zakładki.**

11.2. Narzędzia do wizualizacji i projektowania baz danych..... **Błąd! Nie zdefiniowano zakładki.**

11.3. Platformy online do ćwiczenia SQL **Błąd! Nie zdefiniowano zakładki.**

11.4. Rekomendowane książki i kursy online ..... **Błąd! Nie zdefiniowano zakładki.**

1. DLACZEGO SQL JEST WAŻNY?

1.1. ROLA SQL W NOWOCZESNYM ROZWOJU OPROGRAMOWANIA

1.1. Rola SQL w nowoczesnym rozwoju oprogramowania

SQL (Structured Query Language) to język programowania stworzony do zarządzania i manipulowania relacyjnymi bazami danych. Jego główne zastosowania obejmują:

- Tworzenie, modyfikowanie i usuwanie struktur baz danych
- Wstawianie, aktualizowanie i usuwanie danych
- Pobieranie informacji z baz danych
- Zarządzanie uprawnieniami i kontrolą dostępu do danych

Bazy danych są fundamentem większości nowoczesnych aplikacji. Znajdziesz je w:

- Aplikacjach internetowych
- Systemach zarządzania treścią
- Aplikacjach mobilnych
- Systemach analitycznych
- Platformach e-commerce
- Systemach zarządzania relacjami z klientami (CRM)

Używanie SQL w projektach programistycznych przynosi liczne korzyści:

- Efektywne przechowywanie i zarządzanie dużymi ilościami danych
- Szybkie i precyzyjne wyszukiwanie informacji
- Zapewnienie integralności i spójności danych
- Łatwe skalowanie aplikacji wraz ze wzrostem ilości danych
- Możliwość tworzenia zaawansowanych raportów i analiz
- Standaryzacja dostępu do danych w różnych systemach i językach programowania
- Optymalizacja wydajności aplikacji poprzez efektywne zapytania do bazy danych

SQL można łatwo zintegrować z większością popularnych języków programowania:

- Języki oferują biblioteki i sterowniki do komunikacji z bazami danych
- Zapytania SQL można osadzać bezpośrednio w kodzie aplikacji
- Istnieją ORM (Object-Relational Mapping) ułatwiające pracę z bazami danych
- API bazodanowe umożliwiają abstrakcję niskopoziomowych operacji SQL

Aplikacje oparte na SQL charakteryzują się wysoką skalowalnością i wydajnością:

- Indeksy i optymalizacja zapytań poprawiają szybkość dostępu do danych
- Mechanizmy buforowania zmniejszają obciążenie bazy danych
- Partycjonowanie danych umożliwia efektywne zarządzanie dużymi zbiorami
- Replikacja i klastrowanie zapewniają wysoką dostępność i odporność na awarie
- Transakcje gwarantują spójność danych przy równoczesnym dostępie

SQL oferuje wysoki poziom standaryzacji i przenośności:

- Podstawowa składnia SQL jest wspólna dla większości systemów bazodanowych
- Zapytania można łatwo przenosić między różnymi bazami danych
- Standard ANSI SQL zapewnia kompatybilność między systemami
- Migracja danych między różnymi bazami jest stosunkowo prosta
- Umiejętności SQL są uniwersalne i przydatne w różnych środowiskach

SQL odgrywa kluczową rolę w analizie danych i Business Intelligence:

- Umożliwia tworzenie złożonych zapytań analitycznych
- Wspiera agregację danych i funkcje statystyczne

- Pozwala na łączenie danych z wielu źródeł poprzez operacje JOIN
- Ułatwia generowanie raportów i dashboardów
- Wspiera tworzenie widoków i procedur do cyklicznych analiz
- Umożliwia eksplorację danych i odkrywanie ukrytych wzorców

W kontekście Big Data i przetwarzania w czasie rzeczywistym:

- SQL adaptuje się do pracy z ogromnymi zbiorami danych
- Wspiera przetwarzanie strumieniowe dla analiz w czasie rzeczywistym
- Integruje się z narzędziami do przetwarzania rozproszonych danych
- Pozwala na tworzenie zapytań na danych strukturalnych i niestrukturalnych
- Umożliwia optymalizację wydajności dla bardzo dużych wolumenów danych
- Wspiera paralelizację zapytań dla szybszego przetwarzania

Przyszłość SQL w erze chmury i mikrousług:

- SQL adaptuje się do modeli chmurowych, oferując elastyczność i skalowalność
- Bazy SQL as a Service stają się coraz popularniejsze
- SQL integruje się z architekturami mikrousługowymi
- Rozwój SQL wspiera przetwarzanie danych na krawędzi sieci
- Ewolucja SQL w kierunku obsługi danych nierelacyjnych i semi-strukturalnych
- Automatyzacja zarządzania bazami SQL w środowiskach chmurowych
- Rozwój narzędzi do monitorowania i optymalizacji wydajności SQL w chmurze

1.2. PORÓWNANIE SQL Z INNYMI TECHNOLOGIAMI

Główne cechy SQL jako języka zapytań relacyjnych:

- Deklaratywność - opisujesz co chcesz osiągnąć, nie jak to zrobić
- Standaryzacja - wspólna podstawowa składnia dla różnych systemów
- Wsparcie dla złożonych zapytań i operacji na danych
- Możliwość łączenia danych z wielu tabel
- Wbudowane funkcje agregujące i analityczne
- Obsługa transakcji i zapewnienie integralności danych
- Kontrola dostępu i zarządzanie uprawnieniami
- Optymalizacja zapytań przez system zarządzania bazą danych

Struktura danych w bazach SQL:

- Tabele jako podstawowe jednostki przechowywania danych
- Wiersze reprezentujące pojedyncze rekordy lub encje
- Kolumny definiujące atrybuty lub właściwości danych
- Klucze główne do unikalnej identyfikacji rekordów
- Klucze obce do tworzenia relacji między tabelami
- Indeksy dla optymalizacji wyszukiwania
- Widoki jako wirtualne tabele dla często używanych zapytań
- Schematy do organizacji i grupowania obiektów bazodanowych

Typy operacji CRUD w SQL:

- Create (tworzenie) - dodawanie nowych rekordów do bazy danych
- Read (odczyt) - pobieranie danych z bazy za pomocą zapytań SELECT
- Update (aktualizacja) - modyfikowanie istniejących rekordów
- Delete (usuwanie) - usuwanie rekordów z bazy danych
- Operacje INSERT do dodawania nowych danych
- Zapytania SELECT do pobierania i filtrowania informacji
- Polecenia UPDATE do zmiany wartości w istniejących rekordach

- Instrukcje DELETE do usuwania niepotrzebnych danych

Bazy NoSQL:

- Dokumentowe: przechowują dane w elastycznych strukturach podobnych do JSON
- Klucz-wartość: proste magazyny danych oparte na unikalnych kluczach
- Kolumnowe: optymalizowane do przechowywania i analizy dużych ilości danych
- Grafowe: specjalizują się w reprezentacji i analizie relacji między danymi

Główne różnice między SQL a NoSQL:

- Struktura danych: SQL - sztywna, tabelaryczna; NoSQL - elastyczna, zależna od typu
- Schema: SQL - zdefiniowana z góry; NoSQL - często schema-less lub elastyczna
- Skalowalność: SQL - głównie wertykalna; NoSQL - często horyzontalna
- Spójność danych: SQL - silna; NoSQL - często eventual consistency
- Złożoność zapytań: SQL - zaawansowane zapytania; NoSQL - prostsze operacje
- Transakcje: SQL - pełne wsparcie ACID; NoSQL - różny poziom wsparcia
- Normalizacja: SQL - wysoki poziom; NoSQL - często denormalizacja
- Relacje: SQL - wbudowane; NoSQL - często symulowane lub w przypadku baz grafowych

Scenariusze użycia SQL vs NoSQL:

- SQL: • Aplikacje wymagające złożonych transakcji i integralności danych • Systemy raportowania i analizy biznesowej • Aplikacje z jasno zdefiniowaną strukturą danych • Systemy finansowe i bankowe • Tradycyjne systemy zarządzania treścią

- NoSQL: • Aplikacje z dużą ilością danych i wysokim obciążeniem • Systemy przetwarzające dane w czasie rzeczywistym • Aplikacje wymagające elastycznej struktury danych • Platformy społecznościowe i aplikacje mobilne • Systemy IoT i analiza logów • Magazyny danych dla mikrousług

Hybrydowe podejścia: łączenie SQL i NoSQL w projektach:

- Wykorzystanie SQL do krytycznych danych wymagających integralności
- Użycie NoSQL dla danych o zmiennej strukturze lub wymagających szybkiego dostępu
- Synchronizacja danych między systemami SQL i NoSQL
- Wykorzystanie SQL jako głównego magazynu z NoSQL jako cache
- Implementacja mikrousług z różnymi bazami danych dla różnych funkcji
- Użycie SQL do raportowania i analiz, NoSQL do gromadzenia surowych danych
- Integracja danych z różnych źródeł za pomocą narzędzi ETL

Ewolucja SQL w kierunku obsługi danych nierelacyjnych:

- Wprowadzenie typów danych JSON i XML w bazach relacyjnych
- Rozwój funkcji do przetwarzania i analizy danych semi-strukturalnych
- Implementacja indeksów dla danych nierelacyjnych w bazach SQL
- Wsparcie dla przechowywania i zapytywania danych geograficznych
- Rozszerzenia SQL do obsługi grafów i analiz sieciowych
- Rozwój funkcji analitycznych do przetwarzania dużych zbiorów danych
- Integracja z zewnętrznymi systemami przetwarzania danych

Trendy w rozwoju technologii bazodanowych:

- Wzrost popularności rozwiązań chmurowych i DBaaS (Database as a Service)

- Rozwój baz danych zaprojektowanych dla środowisk kontenerowych
- Zwiększenie nacisku na automatyzację zarządzania i optymalizacji baz danych
- Integracja machine learning w zarządzaniu i optymalizacji zapytań
- Rozwój systemów multi-model łączących różne paradygmaty bazodanowe
- Zwiększenie wsparcia dla przetwarzania strumieniowego i analiz w czasie rzeczywistym
- Rozwój technologii zapewniających większą skalowalność i odporność na awarie
- Nacisk na bezpieczeństwo i zgodność z regulacjami dotyczącymi ochrony danych
- Ewolucja w kierunku baz danych edge computing dla IoT i aplikacji rozproszonych

1.3. SZYBKI START: PIERWSZE ZAPYTANIE SELECT

Struktura podstawowego zapytania SELECT:

- Słowo kluczowe SELECT rozpoczynające zapytanie
- Lista kolumn, które chcemy wybrać
- Słowo kluczowe FROM wskazujące tabelę źródłową
- Opcjonalna klauzula WHERE do filtrowania wyników
- Średnik (;) na końcu zapytania w większości systemów bazodanowych

Wybieranie kolumn z tabeli:

- Nazwy kolumn oddzielone przecinkami po słowie SELECT
- Możliwość wyboru dowolnej liczby kolumn

- Kolejność kolumn w zapytaniu określa kolejność w wynikach
- Możliwość używania aliasów dla kolumn za pomocą słowa AS
- Nazwy kolumn są zazwyczaj case-insensitive, ale najlepiej zachować konsystencję
- Możliwość używania funkcji na kolumnach, np. UPPER() dla tekstu

Użycie gwiazdki (*) do wyboru wszystkich kolumn:

- Symbol * zastępuje listę wszystkich kolumn w tabeli
- Pozwala na szybkie pobranie całej zawartości tabeli
- Użyteczne podczas eksploracji danych lub debugowania
- Może wpływać na wydajność przy dużych tabelach
- Nie zalecane w kodzie produkcyjnym ze względu na potencjalne zmiany struktury tabeli
- Lepiej jawnie wymieniać potrzebne kolumny w zapytaniach aplikacyjnych

Filtrowanie wyników za pomocą WHERE:

- Klauzula WHERE umieszczana po FROM w zapytaniu
- Określa warunki, które muszą spełniać zwracane wiersze
- Można łączyć wiele warunków za pomocą operatorów AND i OR
- Pozwala na precyzyjne zawężanie wyników zapytania
- Wspiera porównania z wartościami stałymi lub innymi kolumnami
- Możliwość używania funkcji w warunkach WHERE
- Warunki mogą odnosić się do dowolnych kolumn, nie tylko tych w SELECT

Proste operatory porównania:

- \= równe
- < mniejsze niż
- > większe niż

- <= mniejsze lub równe

- > \= większe lub równe
- <> lub != różne od
- Operatory działają na liczbach, tekście i datach
- Dla tekstu porównanie jest zazwyczaj case-insensitive
- Przy porównywaniu dat należy zwrócić uwagę na format
- Można używać IS NULL do sprawdzania wartości null

Sortowanie wyników za pomocą ORDER BY:

- Klauzula ORDER BY umieszczana na końcu zapytania
- Pozwala określić kolejność zwracanych wierszy
- Można sortować po jednej lub wielu kolumnach
- Domyślne sortowanie jest rosnące (ASC)
- Użyj DESC dla sortowania malejącego
- Możliwość sortowania po kolumnach nieuwzględnionych w SELECT
- Sortowanie może wpływać na wydajność przy dużych zbiorach danych
- Można sortować po wyrażeniach, np. ORDER BY LENGTH(nazwa)

Praktyczne przykłady prostych zapytań SELECT:

- SELECT imie, nazwisko FROM pracownicy;
- SELECT * FROM produkty WHERE cena < 100;
- SELECT nazwa, data_zamowienia FROM zamowienia ORDER BY data_zamowienia DESC;
- SELECT DISTINCT kategoria FROM produkty;
- SELECT imie, nazwisko FROM klienci WHERE miasto = 'Warszawa';
- SELECT tytul, autor FROM ksiazki WHERE rok_wydania >= 2000 AND rok_wydania <= 2020;
- SELECT nazwa_produktu, cena * 1.23 AS cena_z_vat FROM produkty;
- SELECT imie, nazwisko FROM pracownicy WHERE data_zatrudnienia IS NOT NULL;

Typowe błędy początkujących i jak ich unikać:

- Zapominanie o średniku na końcu zapytania - zawsze kończyć średnikiem
- Używanie pojedynczego cudzysłowu dla nazw kolumn - używać podwójnych cudzysłowów lub backticks
- Mieszanie AND i OR bez nawiasów - używać nawiasów dla jasności priorytetów
- Zapominanie o apostrofach dla wartości tekstowych - zawsze otaczać teksty apostrofami
- Próba używania aliasów kolumn w WHERE - aliasy działają tylko w ORDER BY i później
- Używanie * w produkcyjnych zapytaniach - zawsze wymieniać potrzebne kolumny
- Nieprawidłowa kolejność klauzul - pamiętać o kolejności SELECT, FROM, WHERE, ORDER BY
- Literówki w nazwach tabel lub kolumn - dokładnie sprawdzać nazwy

Narzędzia do wykonywania zapytań SQL:

- Konsole terminalowe specyficzne dla danego systemu bazodanowego
- Graficzne interfejsy dostarczane przez producentów baz danych
- Uniwersalne narzędzia jak DBeaver, HeidiSQL czy MySQL Workbench
- Zintegrowane środowiska programistyczne (IDE) z wtyczkami do baz danych
- Webowe interfejsy administracyjne jak phpMyAdmin dla MySQL
- Narzędzia do wizualizacji danych z możliwością wykonywania zapytań SQL
- Platformy chmurowe oferujące interfejsy do zarządzania bazami danych
- Specjalistyczne narzędzia do monitorowania i optymalizacji wydajności baz danych

2. PODSTAWY: STRUKTURA BAZY I TYPY DANYCH

2.1. TABELE, KOLUMNY I WIERSZE

Tworzenie tabeli z określonymi kolumnami:

Aby utworzyć nową tabelę w bazie danych SQL, użyj polecenia CREATE TABLE. Składnia jest następująca:

```
CREATE TABLE nazwa_tabeli (
nazwa_kolumny1 typ_danych1 [ograniczenia],
nazwa_kolumny2 typ_danych2 [ograniczenia],
...
);
```

Przykład:

```
CREATE TABLE pracownicy (
id INT PRIMARY KEY,
imie VARCHAR(50) NOT NULL,
nazwisko VARCHAR(50) NOT NULL,
data_urodzenia DATE,
stanowisko VARCHAR(100),
pensja DECIMAL(10, 2)
);
```

W tym przykładzie:

- INT to typ całkowity dla identyfikatora
- VARCHAR to typ tekstowy o zmiennej długości
- DATE to typ daty
- DECIMAL to typ liczbowy z precyzją dziesiętną

Ograniczenia, takie jak PRIMARY KEY czy NOT NULL, definiują reguły dla danych w kolumnach.

Dodawanie wierszy do tabeli:

Aby dodać nowe wiersze do istniejącej tabeli, użyj polecenia INSERT INTO. Oto podstawowa składnia:

```
INSERT INTO nazwa_tabeli (kolumna1, kolumna2, ...)
VALUES (wartość1, wartość2, ...);
```

Przykład dodania pracownika do tabeli:

```
INSERT INTO pracownicy (id, imie, nazwisko, data_urodzenia, stanowisko, pensja)
VALUES (1, 'Jan', 'Kowalski', '1990-05-15', 'Programista', 5000.00);
```

Możesz też dodać wiele wierszy jednocześnie:

```
INSERT INTO pracownicy (id, imie, nazwisko, data_urodzenia, stanowisko, pensja)
VALUES
(2, 'Anna', 'Nowak', '1985-10-20', 'Manager', 7000.00),
(3, 'Piotr', 'Wiśniewski', '1992-03-08', 'Analityk', 4500.00);
```

Modyfikowanie istniejących kolumn w tabeli:

Do modyfikacji struktury istniejącej tabeli służy polecenie ALTER TABLE. Oto kilka przykładów:

Dodawanie nowej kolumny:

```
ALTER TABLE pracownicy
ADD COLUMN email VARCHAR(100);
```

Zmiana typu danych kolumny:

```
ALTER TABLE pracownicy
ALTER COLUMN pensja DECIMAL(12, 2);
```

Usuwanie kolumny:

```
ALTER TABLE pracownicy
DROP COLUMN stanowisko;
```

Zmiana nazwy kolumny (składnia może się różnić w zależności od systemu zarządzania bazą danych):

```
ALTER TABLE pracownicy
RENAME COLUMN imie TO pierwsze_imie;
```

Dodawanie ograniczenia:

```
ALTER TABLE pracownicy
ADD CONSTRAINT uq_email UNIQUE (email);
```

Nazywanie tabel i kolumn zgodnie z najlepszymi praktykami:

- Używaj rzeczowników w liczbie mnogiej dla nazw tabel: customers, orders, products.
- Stosuj konwencję snake_case dla nazw tabel i kolumn: user_accounts, first_name.
- Unikaj słów kluczowych SQL jako nazw: zamiast "order" użyj "customer_order".
- Bądź konsekwentny w nazewnictwie: jeśli używasz prefiksu "customer_" w jednej tabeli, stosuj go we wszystkich powiązanych.
- Używaj pełnych, opisowych nazw: zamiast "addr", użyj "address".
- Unikaj skrótów, chyba że są powszechnie zrozumiałe: "id" jest akceptowalne, ale "qty" lepiej zapisać jako "quantity".
- Dla kolumn z kluczami obcymi, używaj nazwy tabeli w liczbie pojedynczej z sufiksem "_id": user_id w tabeli orders.

Określanie odpowiednich typów danych dla kolumn:

- Dla liczb całkowitych:
- TINYINT dla małych liczb (0 do 255)
- INT dla większości przypadków
- BIGINT dla bardzo dużych liczb
- Dla liczb zmiennoprzecinkowych:
- DECIMAL lub NUMERIC dla precyzyjnych obliczeń (np. kwoty pieniężne)
- FLOAT lub DOUBLE dla przybliżonych wartości naukowych

- Dla tekstu:
- CHAR(n) dla stałej długości tekstu
- VARCHAR(n) dla zmiennej długości tekstu
- TEXT dla długich fragmentów tekstu
- Dla dat i czasu:
- DATE dla samej daty
- TIME dla samego czasu
- DATETIME lub TIMESTAMP dla daty i czasu
- Dla wartości logicznych:
- BOOLEAN lub TINYINT(1)
- Dla danych binarnych:
- BLOB dla dużych obiektów binarnych
- Wybieraj najmniejszy możliwy typ danych, który pomieści wszystkie potencjalne wartości.

Usuwanie tabel i wierszy:

Usuwanie całej tabeli:

```
DROP TABLE nazwa_tabeli;
```

Przykład:

```
DROP TABLE pracownicy;
```

Usuwanie wszystkich wierszy z tabeli bez usuwania samej tabeli:

```
DELETE FROM nazwa_tabeli;
```

Przykład:

```
DELETE FROM pracownicy;
```

Usuwanie określonych wierszy z tabeli:

```
DELETE FROM nazwa_tabeli
```

```
WHERE warunek;
```

Przykład:

```
DELETE FROM pracownicy
WHERE data_urodzenia < '1980-01-01';
```

- Przed usunięciem danych zawsze twórz kopię zapasową.
- Używaj klauzuli WHERE ostrożnie, aby uniknąć przypadkowego usunięcia zbyt wielu danych.
- Rozważ użycie transakcji dla bezpieczniejszego usuwania:

```
BEGIN TRANSACTION;
DELETE FROM pracownicy WHERE id = 5;
-- Sprawdź, czy usunięto właściwe dane
COMMIT; -- lub ROLLBACK, jeśli coś poszło nie tak
```

- W przypadku dużych tabel, usuwanie partiami może być wydajniejsze:

```
DELETE TOP(1000) FROM duza_tabela
WHERE data < '2020-01-01';
```

Przeglądanie struktury istniejącej tabeli:

W większości systemów zarządzania bazami danych SQL możesz użyć polecenia DESCRIBE lub podobnego:

```
DESCRIBE nazwa_tabeli;
```

Alternatywnie, w standardzie SQL:

```
SELECT column_name, data_type, character_maximum_length, is_nullable
FROM information_schema.columns
WHERE table_name = 'nazwa_tabeli';
```

Dla bardziej szczegółowych informacji, w tym o ograniczeniach:

```
SELECT *
FROM information_schema.table_constraints
WHERE table_name = 'nazwa_tabeli';
```

W niektórych systemach możesz użyć narzędzi GUI do przeglądania struktury tabeli.

Kopiowanie struktury tabeli:

Aby skopiować strukturę tabeli bez danych:

```
CREATE TABLE nowa_tabela AS
SELECT *
FROM stara_tabela
WHERE 1 = 0;
```

Alternatywnie:

```
CREATE TABLE nowa_tabela LIKE stara_tabela;
```

Aby skopiować strukturę i dane:

```
CREATE TABLE nowa_tabela AS
SELECT * FROM stara_tabela;
```

Dla bardziej zaawansowanej kontroli:

```
CREATE TABLE nowa_tabela (
SELECT *
FROM stara_tabela
) WITH NO DATA;
```

Tworzenie tymczasowych tabel:

Tymczasowe tabele istnieją tylko w ramach bieżącej sesji:

```
CREATE TEMPORARY TABLE nazwa_temp_tabeli (
kolumna1 typ_danych1,
kolumna2 typ_danych2,
```

```
...
);
```

Przykład:

```
CREATE TEMPORARY TABLE temp_pracownicy (
id INT,
imie VARCHAR(50),
nazwisko VARCHAR(50)
);
```

Możesz też utworzyć tymczasową tabelę na podstawie wyników zapytania:

```
CREATE TEMPORARY TABLE temp_wyniki AS
SELECT id, imie, nazwisko
FROM pracownicy
WHERE data_zatrudnienia > '2023-01-01';
```

Tymczasowe tabele są automatycznie usuwane po zakończeniu sesji, ale możesz je usunąć wcześniej:

```
DROP TEMPORARY TABLE IF EXISTS temp_pracownicy;
```

- Tymczasowe tabele są widoczne tylko dla bieżącej sesji.
- Są przydatne do przechowywania pośrednich wyników złożonych obliczeń.
- Nie obciążają głównej bazy danych i są szybsze w operacjach.