SQL Course - Day 1
Plan :
0. Wstęp - relacyjne bazy danych
1. SQL - pobieranie danych z pojedynczej tabeli
2. DML - Wstawianie, aktualizacja i usuwanie rekordów z tablicy
3. Złączenia - pobieranie danych z wielu tabel

Section : Wstęp - Relacyjne bazy danych
Historia SQL vs Sequel
Relacyjne bazy danych powstały wraz z publikacją E.F. Codda z 1970 r. "A Relational Model of Data for Large Shared Data Banks.". Pomysły Codda były odkrywcze w tamtym czasie. Na podstawie tych prac, w San Jose w Kalifornii dwaj koledzy, Donald D. Chamberlin i Raymond F. Boyce tworzyli SQUARE (określanie zapytań jako wyrażeń relacyjnych) - język zapytań. Do 1974 roku opublikowali język zapytań SEQUEL (Structured English Query Language) oparty na SQUARE. Niestety, z powodu naruszenia znaków towarowych w akronimie SEQUEL, który był już zarejestrowany przez firmę lotniczą Hawker Siddeley, nazwa została zmieniona na `Structured Query Language` i skrócona do `SQL`.

W Internecie toczy się wiele dyskusji dotyczących wymowy SQL. Niektórzy ludzie są za „sequelem", niektórzy za „S.Q.L.", a jeszcze inni tworzą własne wymowy.

Nalezy pamięta też, że konkretne implementacje baz danych mogą mieć własne preferencje.

Oficjalnym sposobem wymawiania `MySQL` jest `My Ess Que Ell` (nie "my sequel"), ale niektórym nie przeszkadza, że wymawia się je jako `my sequel` lub w inny lokalny sposób.

Poniewaz na naszym kursie posługujemy się implementacją MySQL - „My Ess Que Ell" - konsekwentnie będę uzywał wymowy "S.Q.L".

Bazy SQL vs NoSQL
SQL język manipulacji danymi - (SDA course)
- podstawowe zasady języka SQL
    - język SQL NIE jest case_sensitive - nie ma znaczenia czy piszemy małymi czy duzymi literami
    - przyjęta konwencja :
      - słowa kluczowe piszemy zazwyczaj dużymi literami,
      - zmienne małymi ze znakiem podkreślenia ale jest to konwencja nie twarde wymaganie
    - nawiasy wymuszają porządek wykonywania wyrazeń arytmetycznych
    - `--` - oznacza komentarz
    - napisy (łańcuchy znakowe) otaczamy pojedynczym cudzysłowem ` ' ' `
    - instrukcje lub klauzule kończy średnik `;`

Section 1 : SQL - Pobieranie danych z pojedynczej tabeli
Intro : zainstalowanie MySQL + workbanch + wgranie bazy danych
Szkolenie z SQL - instalacja MySQL + WorkBench
https://www.youtube.com/watch?v=UduVPtGGytY

Lesson 1 : Query SELECT
  - klazula SELECT ma ustalony porządek tylko słowa SELECT i FROM są obowiązkowe
  - mozemy uzyc gwiazdki `*` - wówczas wypiszemy wszystkie kolumny w tabeli lub jawnie wyspecyfikowac kolumny np. first_name, last_name, points
  - operatory porównania : <, >, <>, !=, =
  - słowo kluczowe AS
  - symbol gwiazdka "*"
  - wyrazenia arytmetyczne
  - pozostałe klauzule : FROM - WHERE - ORDER BY

- przykład :
```
    USE sql_store;

    SELECT last_name, first_name, points *10 + 100
```

```
    FROM customers          -- FROM sql_store.customers
    -- WHERE customr_id = 1
    -- ORDER BY first_name

#### Zadanie 1 -----------------
Z bazy danych sql_store, zwróc wszystkie produkty (z tabeli products) z kolumnami :
- name
- unit price
- new price (= unit price * 1.1)

Rozwiązanie :
    USE sql_store;

    SELECT
        name,
        unit_price AS 'unit price',
        unit_price * 1.1 AS 'new price'
    FROM Products;


### 2 : Klauzula WHERE
- klauzula WHERE umozliwia filtrowanie rekordów wyniku wg warunków
- Przykład :
    SELECT *
    FROM Customers
    WHERE points > 3000
- operatory porównania : <, >, <>, !=, =
- napisy otaczamy pojedynczym cudzysłowem  ` ' ' `
- non case sensitive np  state = 'VA' jest równowazne state = 'va'
- daty porównujemy takze stosując zapis w cudzysłowach
  - birth_date > '1990-01-01'

#### Zadanie 2 -----------------
Znajdz zamówienia zlozone w roku 2019.

Rozwiązanie :
    SELECT *
    FROM orders
    WHERE order_date >= '2019-01-01'

### 3 : Operatory OR, AND i NOT
- pierwszeństwo operatorów
- operator negacji

### 4 : Operator IN, NOT IN
- zastosowanie operatora IN /NOT IN do liczb i znaków

#### Zadanie 3 -----------------
Odszukaj Products spełniające warunek:
    -- quantity in stock =  { 49, 38, 72 }

    SELECT *
    FROM products
    WHERE quantity_in_stock IN (49, 38, 72)

### 5 : Operator BETWEEN … AND …
  - zastosowanie operatora zamiast porównania AND
  - porównanie inclusive - równoważne … <=X  AND … >= Y

#### Zadanie  4 -----------------
Odszukaj Klientów (Customers) urodzonych pomiędzy 1/1/1990 oraz 1/1/2000

### 6 : Operator LIKE oraz NOT LIKE
```

```
- zastosowanie do odszukiwania po polach String
- znak ‚%' zastępuje dowolny ciąg znaków - przykład LIKE '%b%'
- znak ‚_' zastępuje dokładnie jeden znak - przykład LIKE ‚b_____y'
- operator jak cały SQL `NIE jest` Case Sensitive

- Przykład : SELECT * FROM customers WHERE last_name LIKE '%b%'

#### Zadanie 5 ----------------
    -- Odszukaj Customers tych którzy spełniają warunek:
    --     1. address zawiera TRAIL lub AVENUE
    --     2. numer telefonu jest zakończony na 9
Rozwiązanie 1 :
    SELECT *
    FROM sql_store.customers
    WHERE   address LIKE '%TRAIL%' OR
            address LIKE '%AVENUE%'

lub z uzyciem wyrazenia regularnego REGEXP
    SELECT *
    FROM sql_store.customers
    WHERE address  REGEXP 'trail|avenue'
    -- address LIKE '%TRAIL%' OR
    -- address LIKE '%AVENUE%'

### 7 : Operator IS NULL oraz IS NOT NULL
#### Zadanie  6 ----------------
    -- Znajdź zamówienia (orders), które nie są wsyłane (shipped)
    SELECT *
    FROM orders
    WHERE shipper_id IS NULL

### 8 : Klauzule ORDER BY
  - domyślnie sortowane po kluczu głównym tabeli (pimary Key  column) w rozwiązaniach
produkcyjnych typowo - ID
  - domyślnie sortowanie jest w porządku naturalnym tzn 1, 2, 3 etc lub a,b, c, d etc
  - odwrócenie porządku sortowania - słowo kluczowe DESC np.
  - -- ORDER BY first_name DESC
  - sortowanie po kilku kolumnach, każda z nich może być oznaczona DESC
  - MsSQL Specific : kolumny po których sortujemy nie muszą być w nagłówku kwerendy

#### Zadanie 7 ----------------
    -- Odgadnij kwerendę patrząc na jej rezultat:
    SELECT *
    FROM order_items
    WHERE order_id = 2
    ORDER BY quantity * unit_price DESC

### 9 : Klauzula LIMIT
  - służy do ograniczenia ilości zwracanych rekordów
  - SELECT * FROM customers LIMIT 4
  - klauzula LIMIT musi być ostatnia w kwerendzie
  - dwa parametry oznaczają : skip = pomiń N początkowych wierszy, displ = wyświetl Y
następnych
  - … LIMIT 6, 3  -- wyświetl 7, 8 i 9 wiersz

#### Zadanie 8 ----------------
Wyświetl 3 najbardziej lojalnych klientów (podpowiedź im klient ma więcej punktów tym
jest bardziej lojalny)
    SELECT *
    FROM customers
    ORDER BY points DESC
    LIMIT 3
------------------------------------------------------------
```

```
## Section 2 : DML - Data Modification Language
### 1 : Atrybuty kolumn
- workbench - ustaw się na tablicy > kliknij na środkową ikonę
  - PK - PRIMARY KEY
  - NN - NOT NULL
  - AI - AUTO_INCREMENT
  - DEFAULT
  - INDEX
  - UQ - UNIQUE
  - G - GENERATED

### 2 : Wstawianie rekordów do tablicy
- wstawianie pojedynczego rekordu
    INSERT INTO customers
    VALUES (DEFAULT,'John','Smith',1990-01-01',NULL,'address 12','city 123','CA',DEFAULT
);

    - lub z pominięciem kolumn których nie musimy podawac z uwagi na autogenerację/
wartości domyślne (DEFAULT).
    INSERT INTO customers (
        last_name,
        first_name,
        birth_day,
        address,
        city,
        staye
    )
    VALUES (
        'Smith',
        'John',
        '1990-01-01',
        'address 12',
        'city 123','CA');

- wstawianie wielu rekordów
INSERT INTO shippers (name)
VALUES  ('Shipper 1'),
        ('Shipper 2'),
        ('Shipper 3');

#### Zadanie 1 ----------------
Wstaw trzy nowe wiersze opisujące produkty (products)

### 2 : Wstawianie rekordów do wielu tablic
- relacja 1 : n - orders -> order_items
    INSERT INTO orders (customer_id, order_date, status)
        VALUES (1,'2019-02-01', 1);
    INSERT INTO order_items
    VALUES
        (LAST_INSERTED_ID(), 1, 1, 2.95),
        (LAST_INSERTED_ID(), 2, 1, 4.50);

- LAST_INSERTED_ID() -
  - uzycie : SELECT LAST_INSERTED_RECORD()

### 3 : Tworzenie kopii tablicy existing >> new_one
- tworzenie kopii tablicy - wariant 1
    CREATE TABLE orders_archived AS
    SELECT * FROM orders
    -- nie ma przeniesionych atrybutów kolumn jak np PK

 - subquery mozna uzyc takze w poleceniu INSERT, mozna je wzbogacic o klauzule WHERE etc
```

```
#### Zadanie 2 -----------------
stwórz nową tablicę invoices_archive dla rekordów posiadających określoną (niepustą)
datę płatności

    USE  sql_invoicing;

    CREATE TABLE invoices_archived AS
        SELECT  *
        FROM invoices
        WHERE payment_date IS NOT NULL;

### 4 : Aktualizacja rekordów
- aktualizacja pojedynczego rekordu
    UPDATE invoices
    SET payment_total = 10, payment_date = CURRENT_TIMESTAMP -- biezaca data -
poprzednio GETDATE()
    WHERE invoice_id = 1

    - w wyraezniu SET mozemy uzyc wyrazeń arytmetycznych i odwołań do innych kolumn np
SET payment_date = due_date

- aktualizacja wielu rekordów - wymaga wyłączenia SAFE_MODE w MySQLWorkbench :
  - Main Menu > MySQLWorkbench > Preferences > SQL Wditor
  - odnznacz checkbox [Sfe Updates] (na dole ekranu)
  - reconnect instancję MySQLWorkbench
    UPDATE invoices
        SET
            payment_total = invoice_total * 0.8,
            payment_date = due_date
        WHERE client_id = 3

### 5 : Aktualizacja z uzyciem Subqueries
- proste wyszukiwanie : aktualizuj fakturę dla której nazwa klienta = 'Myworks'
        UPDATE invoices
        SET
            payment_total = invoice_total * 0.8,
            payment_date = due_date
        WHERE client_id = (
            SELECT client_id
            FROM clients
            WHERE name = 'Myworks'
        )

- aktualizacja wielu rekordów
    UPDATE invoices
        SET
            payment_total = invoice_total * 0.8,
            payment_date = due_date
        WHERE client_id IN (
            SELECT client_id
            FROM clients
            WHERE state IN ('CA', 'NY')
        )

### 6 : Kasowanie rekordów
    DELETE FROM invoices
    WHERE client_id = 2

- podobnie jak podczas aktualizacji mozna uzyc subqeeries aby skasowac więcej niz jeden
rekord

### 7 : Sprzątamy w bazach danych
```

```
- przed kolejnymi cwiczeniami odtwórmy początkowy stan tablic w DB
  - File Menu > Open Recent Scripts
  - wykonaj script create-databases.sql
-------------------------------------------------------------

## Section 3 : Pobieranie danych z wielu tabel
### 1 : Inner Joins
- w realnym projekcie rzadko kiedy pobieramy dane tylko z jednej tabeli - zwykle
korzystamy z dwóch lub więcej tablic
- przykład :
    SELECT *
    FROM orders     -- podstawowa tablica po której wykonujemy select
    JOIN customer   -- tablica dołączana INNER JOIN
        ON orders.customer_id = customer.customer_id

- do nazw tablic możemy użyć aliasów:
    SELECT *
    FROM orders o
    JOIN customer c
        ON o.customer_id = c.customer_id

#### Zadanie 1 ——————————
Dla tablicy order_items wykonaj złączenie z tablicą products, wyświetl wynik tak, żeby
po kolumnie : product_id, wyświetlała się nazwa produktu. Użyj aliasów.

    SELECT order_id, oi.product_id, p.name, quantity, oi.unit_price
    FROM order_items oi
    JOIN products p ON oi.product_id = p.product_id

### 2 : Złączenie wielu bazach danych
- Przykład: łączymy tabelę products w db sql_inventory z tablicą order_item w sql_store
  - prefixujemy nazwę tablicy nazwą bazy danych
    USE sql_story; -- w kontekscie tej bazy wykonujemy zapytanie

    SELECT *
    FROM order_items oi
    JOIN sql_inventory.products p
        ON oi.product_id = p.product_id

### 3 : Self Joins
- przykład złączenia tablicy z nią samą : w bazie sql_hr chcemy wyszukac pracowników i
ich managerów.

    USE sql_hr;

    SELECT *
    FROM employees e
    JOIN employees m  -- we need to other alias
        ON e.reports_to = m.employee_id
- uprośćmy zapytanie tak by było tylko id_pracownika, jego imię + imię przełozonego >
rezultat stanie się bardziej przejrzysty

    SELECT
        e.employee_id,
        e.first_name,
        m.first_name AS manager
    FROM employees e
    JOIN employees m  -- we need to other alias
        ON e.reports_to = m.employee_id

### 4 : Złączenie wielu tablic
- przykład złączenia tablic orders + customers + order_statuses
```

```
    USE sql_store;

    SELECT *
    FROM orders o
    JOIN customers c
        ON o.customer_id = c.customer_id
    JOIN order_statuses os
        ON o.status = os.order_status_id

#### Zadanie 2 : ------------------
Rezultat złączenia : payments[] + clients[] + payment_methds jest skomplikowany.
Uprość go tak aby wyświetlic jedynie :
        p.date,
        invoice_id,
        paymnts.amount,
        clients.name,
        payment_methods.status

 USE sql_invoicing;

 SELECT
        p.date,
        p.invoice_id,
        p.amount,
        c.name,
        pm.name
    FROM payments p
    JOIN clients c
        ON p.client_id = c.client_id
    JOIN payment_methods pm
        ON p.payment_method = pm.payment_method_id

### 5 : Compoud Join Conditions
- Przykład : klucz złozony : order_id + product_id w tablicy order_items[]
  - połącz z tablicą order_item_notes z uzyciem klucza złozonego

    SELECT *
    FROM order_items oi
    JOIN order_item_notes oin
        ON oi.order_id = oin.order_id      -- compoud join condition +
        AND oi.product_id = oin.product_id -- this

### 6 : Implicit Join Syntax
- jawne wskazanie warunku złączenia w klauzuli WHERE
    -- explicit join syntax
    SELECT *
    FROM orders o
    JOIN customers c
        ON o.customer_id = c.customer _id

    -- Implicit Join Syntax
    SELECT *
    FROM orders o, customers c
    WHERE o.customer_id = c.customer _id

- implicit join syntax nie jest polecane z dwóch powodów:
  - pominięcie warunku WHERE spowoduje, ze rezultate będzie iloczyn kartezjański
  - lepiej jawnie oddzielic warunki JOIN od filtru WHERE w skomplikowanych zapytaniach


### 7 : Outer Joins
- INNER JOIN (eq. JOIN) & OUTER JOIN
- Przykład inner join :
```

```
    USE sql_store

    SELECT
        c.customer_id,
        c.first_name,
        0.order_id
    FROM customers c
    JOIN orders o
        ON c.customer_id = o.customer_id
    ORDER BY c.customer_id  -- dla czytelności rezultatu

- aby zmienic INNER JOIN na > LEFT OUTER JOIN zmieniamy
    //...
    FROM customer c
    LEFT OUTER JOIN order o   -- OUTER is optional

- RIGHT OUTER JOIN wymaga takze zamiany kolejnosci tablic ( wowczas prawa strona warunku
 rozstrzyga co jest w selekcie)
    // ...
    FROM orders o
    RIGHT JOIN customers c

#### Zadanie 3  ------------------------------------
Zbuduj zapytanie Outer Join dające następujący rezultat :
  - mamy trzy kolumny: product_id, name (products []) + quantity (order_items [])
  - zapytanie musi znajdowac produkty nawet jeśli liczba zamówionych pruduktów wynosi 0
 (NULL)

Rozwiązanie :
    SELECT
        p.product_id,
        p.name,
        oi.quantity
    FROM products p
    LEFT JOIN order_items oi
        ON p.product_id = oi.product_id

### 8 : Outer Join pomiędzy wieloma tablicami
    SELECT
        c.customer_id,
        c.first_name,
        o.order_id
        sh.name AS shipper
    FROM customers c
    LEFT JOIN orders o
        ON c.customer_id = o.customer_id
    LEFT JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
    ORDER BY c.customer_id

- best practices : unikamy "prawych" outer joinów z powodu na komplikowanie całego
rozwiązania i trudności w zrozumieniu o co tak naprawdę chodzi.

#### Zadanie 4 ------------------------------------
Zbuduj zapytanie Outer Join zawierające : order_date, order_id (customer).first_name,
shipper oraz status (nazwa). Kazde zamówienie powinno posiadac swojego klienta.

    SELECT
        o.order_date,
        o.order_id,
        c.first_name AS customer,
        sh.name  AS shipper,
        os.name AS status
```

```
    FROM orders o
    JOIN customers c
        ON o.customer_id = c.customer_id
    LEFT JOIN shippers sh
        ON o.shipper_id = sh.shipper_id
    JOIN order_statuses os
        ON o.status = os.order_status_id

### 9 : Self outer join
- z bazy sql_hr znjadź wszystkich pracowników z managerami ORAZ samych managerow
    USE sql_hr;

    SELECT
        e.employee_id,
        e.first_name,
        m.first_name AS manager
    FROM employees e
    LEFT JOIN employees m
        ON e.reports_to = m.employee_id

### 10 : Klauzula USING
- uproszczenie złączenia w przypadku gdy łączymy tablice po tych samych nazwach kolumn
- rozszerzenie MySQL
    // ...
    FROM orders o
    JOIN customers c
        -- ON o.customer_id = c.customer_id
        USING (customer_id)
    // ...

- mozemy uzywac klauzuli USING takze do OUTER JOIN oraz do złączeń wielu tablic -
compound join conditions
  - zamiast :
    USE sql_store

    SELECT *
    FROM order_items oi
    JOIN order_item_notes oin
        ON oi.order_id = oin.order_id      -- compoud join condition +
        AND oi.product_id = oin.product_id -- this

  - mozna napisac :
    // ...
    JOIN order_item_notes oin
        USING(order_id, product_id)

#### Zadanie 5 ------------------------------------
Z bazy sql_invoicing wyszukaj : (payments).date, client_id, amount, oraz (
payment_methods).name
    USE sql_invoicing;

    SELECT
            p.date, c.client_id, amount, pm.name
        FROM payments p
        JOIN clients c USING (client_id)
        JOIN payment_methods pm
            ON p.payment_method = pm.payment_method_id

### 11 : Złączenie naturalne (natural joins)
- właściwosc MySQL - nie jest w standardzie SQL ANSI
- jest kolejnym uproszczeniem klauzuli USING
- moze prowadzic do nieprzewidzianych rezultatów, NIE REKOMENDOWAME do stosowania
    SELECT *  -- o.order_id, c.first_name
```

```
    FROM orders o
    NATURAL JOIN customers c

- powyzsze nie jest równowazne ani inner join
    SELECT *
    FROM orders o
    JOIN customers c USING ( customer_id)

- ani left outer loin
    SELECT *
    FROM orders o
    LEFT JOIN customers c USING ( customer_id)

### 12 : Cross Joins
- iloczyn kartezjański dla dwóch tablic : customer x product
- w praktyce bardzo ograniczone zastosowanie
    SELECT
        c.first_name AS customer,
        p.name AS product
    FROM customers c
    CROSS JOIN products p
    ORDER BY c.first_name

- przykład jw z implicit syntax
    // ...
    FROM customers c, products p
    ORDER BY c.first_name

### 13 : Unions
- złączenie wyników z róznych zapytań : składnia
  - pierwszy "SELECT"
  - UNION
  - drugi "SELECT"
- przykład : chcemy wszystkie zamówienia wcześniejsze ni z roku 2019 pokazac ze statusem
 `Archived` a z lat 2019, 2020 oraz 2021 `Active`
USE sql_store;

    SELECT
        order_id,
        order_date,
        'Active' AS status
    FROM orders
    WHERE order_date >= '2019-01-01' -- CURRENT_TIMESTAMP()
UNION
    SELECT
        order_id,
        order_date,
        'Archived' AS status
    FROM orders
    WHERE order_date < '2019-01-01' -- CURRENT_TIMESTAMP()

- złączyc mozna wyniki z dwóch zapytań z róznych tablic, jednak ilosc zwracanych kolumn
musi by taka sama

    SELECT first_name
    FROM customers
    UNION
    SELECT name
    FROM shippers
```