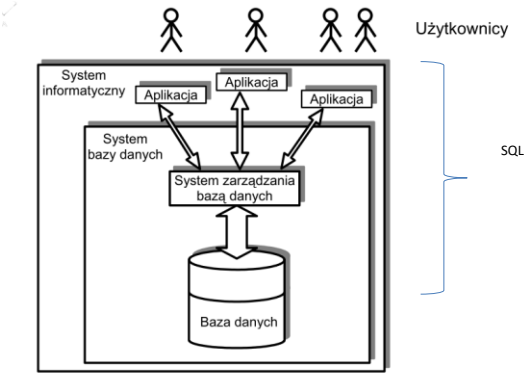



## Czym jest SQL?

SQL to Strukturalny Język Zapytań (z ang. Stuctured Query Language) służący do wykonywania wszelkich operacji **w systemach relacyjnych baz danych** RDBMS (z ang. Relational Database Management System).
Jest to język uniwersalny, stosowany zarówno w systemach komercyjnych, jak: DB2, MS SQL czy Oracle, jak i rozwijanych na zasadach OpenSource, jak PostgreSQL czy darmowe wersje MySQL.
Początki SQL sięgają wczesnych lat 70. ubiegłego wieku, kiedy to w firmie IBM został rozpoczęty projekt bazodanowy o nazwie System/R, na którego potrzeby powstał język SEQUEL (z ang. Structured English Query Language). Druga wersja SEQUEL-a, SEQUEL/2, w roku 1997 została przemianowana na SQL.
SQL pozwala na wykonywanie następujących operacji:
**Uzyskiwanie (pobieranie), modyfikowanie, wstawianie, usuwanie, sterowanie danych**

### Podgrupy SQLa

**DDL** (**Data Definition Language**) można operować na strukturach, w których dane są przechowywane – czyli np. dodawać, zmieniać i kasować tabele lub bazy. Najważniejsze polecenia tej grupy to:

**CREATE** (np. CREATE TABLE, CREATE DATABASE, ...) – utworzenie struktury (bazy, tabeli, indeksu itp.),
**DROP** (np. DROP TABLE, DROP DATABASE, ...) – usunięcie struktury,
**ALTER** (np. ALTER TABLE ADD COLUMN ...) – zmiana struktury (dodanie kolumny do tabeli, zmiana typu danych w kolumnie tabeli).

**DML** (**Data Manipulation Language**) służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania oraz dokonywania zmian. Najważniejsze polecenia z tego zbioru to:
**INSERT** – umieszczenie danych w bazie,
**UPDATE** – zmiana danych,
**DELETE** – usunięcie danych z bazy.
Dane tekstowe muszą być zawsze ujęte w znaki pojedynczego cudzysłowu (').

### Podgrupy SQLa

**DC**L (**Data Control Language**) ma zastosowanie do nadawania uprawnień do obiektów bazodanowych. Najważniejsze polecenia w tej grupie to:

**GRANT** - służące do nadawania uprawnień do pojedynczych obiektów lub globalnie konkretnemu użytkownikowi
**REVOKE** – służące do odbierania wskazanych uprawnień konkretnemu
**DENY** - służy głównie do odbierania uprawnień dostępu do obiektów bazodanowych

**DQL** (**Data Query Language**) to język formułowania zapytań do bazy danych. W zakres tego języka wchodzi jedno polecenie - **SELECT**.

Często SELECT traktuje się jako część języka DML, ale to podejście nie wydaje się właściwe, ponieważ DML z definicji służy do manipulowania danymi - ich tworzenia, usuwania i uaktualniania.
Na pograniczu obu podgrup znajduje się polecenie SELECT INTO, które dodatkowo modyfikuje (przepisuje, tworzy) dane.

**TCL Transaction Control Language** – język sterowania przepływem danych (kontrola transakcji) . Najważniejsze polecenia w tej grupie to:

**COMMIT** – zatwierdzenie transakcji
**ROLLBACK** – wycofanie transakcji
**SAVEPOINT** – punkt przywracania transakcji

Podczas pracy z bazą danych zachodzi konieczność wydobycia określonych informacji, zwłaszcza takich, które spełniają pożądany przez nas warunek. Ponieważ zmienna relacyjna (tabela) ma dane pogrupowane dzięki atrybutom (kolumnom), możemy na takich zbiorach danych wykonywać operacje.
Na początek trochę teorii. Operacje, o których mówimy, przedstawia się za pomocą operatorów algebry relacyjnej. Operator taki na wejściu pobiera argumenty będące relacjami, natomiast zwraca relację wynikową. Operatory algebry relacyjnej przedstawia tabela:

| Operator | Symbol |
|---|---|
| Selekcja | б |
| Projekcja | П |
| Złączenie | ⋈ |
| Suma | U |

SELEKCJA

Wyobraźmy sobie selekcjonera drużyny piłki nożnej. Jego zadaniem jest wybór zawodników spełniających określone kryteria. Przyjmijmy, że kryterium selekcji będzie wiek powyżej 18 lat.
Matematycznie operacja selekcji będzie miała postać:

ZAWODNICY

| Nr | Imię | Wiek (lat) |
|---|---|---|
| 1 | Janek | 12 |
| 2 | Marek | 15 |
| 3 | Heniek | 22 |
| 4 | Kornel | 34 |
| 5 | Wojtek | 25 |
| 6 | Tadeusz | 43 |

$\sigma_{\text{Wiek} > 18\ (\text{Zawodnicy})}$

Gdzie:
$\sigma_{\text{Wiek} > 18\ (\text{Zawodnicy})}$

Warunek selekcji

Atrybut
(kolumna tabeli
– zbiór określonych cech)

Selekcja

W strukturalnym języku zapytań taka selekcja będzie miała postać:

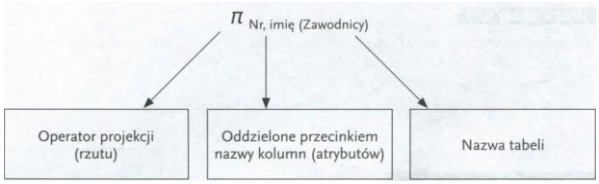
***SELECT * FROM zawodnicy WHERE wiek > 18;***

W efekcie otrzymamy wynik:

```
mojabaza=# SELECT * FROM zawodnicy WHERE wiek > 18;
 nr |  imie   | wiek
----+---------+------
  3 | Heniek  |   22
  4 | Kornel  |   34
  5 | Wojtek  |   25
  6 | Tadeusz |   43
(4 wiersze)
```

Należy pamiętać, że zapytanie kończymy, wpisując znak średnika ;

**Projekcja** nazywana bywa również **rzutem** i możemy ją zdefiniować **jako wybór kolumn**. Operator projekcji na wejściu przyjmuje nazwy kolumn, a na wyjściu zwraca ich zawartość. Możemy prześledzić to na wcześniejszym przykładzie z selekcjonerem piłkarskim. Wyobraźmy sobie sytuację, że selekcjoner będzie potrzebował listy wszystkich zawodników wraz z kandydatami (zawodnikami poniżej 18 roku życia). Lista będzie musiała składać się z imion i numerów zawodników. Aby otrzymać taki zbiór za pomocą algebry relacyjnej, należy posłużyć się wyrażeniem:




Wyrażenie to możemy również zapisać za pomocą języka SQL w postaci:

***SELECT nr, imie FROM zawodnicy***;

```
mojabaza=# SELECT nr, imie FROM zawodnicy;
 nr |  imie
----+--------
  1 | Janek
  2 | Marek
  3 | Heniek
  4 | Kornel
  5 | Wojtek
  6 | Tadeusz
(6 wierszy)
```

**Złączenie (*JOIN*)** służy do pobierania danych z dwóch lub większej liczby tabel w celu porównania lub zestawienia. W tabelach biorących udział w złączeniu muszą występować kolumny, które są zgodne i spełniają warunki pozwalające na dokonanie złączenia. **Zaleca się, aby kolumny te łączyły dwie relacje (tabele) na zasadzie: klucz podstawowy, klucz obcy**, chociaż warunek ten nie jest niezbędny do wykonania złączenia.
Dla przykładu wykorzystamy dwie tabele: zawodnicy i pokoje.

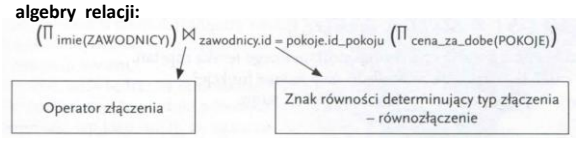
| nr [PK] integer | imie character varying(50) | wiek integer |
|---|---|---|
| 1 | Janek | 12 |
| 2 | Marek | 15 |
| 3 | Heniek | 22 |
| 4 | Kornel | 34 |
| 5 | Wojtek | 25 |
| 6 | Tadeusz | 43 |

| id_pokoju [PK] integer | cena_za_dobe integer | id_goscia integer |
|---|---|---|
| 1 | 50 | 1 |
| 2 | 60 | 2 |
| 3 | 70 | 3 |
| 4 | 20 | 4 |
| 5 | 10 | 5 |
| 6 | 23 | 6 |

Gdy założymy, że numer gościa (kolumna **id_goscia**) odpowiada numerowi zawodnika (kolumnie **nr**), wówczas, chcąc otrzymać zestawienie **imienia** zawodnika oraz **ceny**, którą ma zapłacić za pokój, wykonujemy złączenie:
***SELECT imię,cena_za_dobe FROM zawodnicy JOIN pokoje ON nr=id_goscia;***

```
mojabaza=# SELECT imie, cena_za_dobe FROM zawodnicy JOIN pokoje ON nr=id_goscia;
  imie   | cena_za_dobe
---------+--------------
 Marek   |           60
 Janek   |           50
 Heniek  |           70
 Kornel  |           20
 Wojtek  |           10
 Tadeusz |           23
(6 wierszy)
```

**A oto przykład operacji równozłączenia zapisany językiem algebry relacji:**



**Suma (Union)**

Teoriomnogościowa suma dotyczy dwóch relacji o tym samym schemacie - językiem algebry relacji może być zapisana jako $A \cup B$.

Dla przykładu wykorzystaliśmy dwie tabele: **prac_p_pomoc** i **pracownicy_bhp** Jak zostało to przedstawione na poniższej ilustracji, tabele mają tę samą strukturę.

| id_pracownika [PK] integer | imie character varying(50) | nazwisko character varying(50) | wiek integer |
|---|---|---|---|
| 1 | Jan | Nowak | 24 |
| 2 | Wojciech | Kowalski | 30 |
| 3 | Michal | Niemczak | 23 |
| 4 | Renata | Zawadzka | 25 |
| 5 | Irena | Szalowska | 55 |
| 6 | Wiktoria | Kowalska | 23 |
| 7 | Bozena | Przykoscinska | 40 |

| id_pracownik [PK] integer | imie character varying(50) | nazwisko character varying(50) | wiek integer |
|---|---|---|---|
| 1 | Jan | Nowak | 24 |
| 2 | Wojciech | Kowalski | 30 |
| 3 | Marian | Nowak | 33 |
| 4 | Marcin | Tracz | 26 |
| 5 | Ewa | Werner | 23 |

$$\Pi_{id\_pracownika,imie,nazwisko,wiek}(prac\_p\_pomoc) \cup \Pi_{id\_pracownika,imie,nazwisko,wiek}(pracownicy\_bhp)$$

```
mojabaza=# SELECT * FROM prac_p_pomoc
mojabaza-# UNION
mojabaza-# SELECT * FROM pracownicy_bhp;
 id_pracownika |   imie   |    nazwisko     | wiek
---------------+----------+-----------------+------
             3 | Marian   | Nowak           |   33
             5 | Ewa      | Werner          |   23
             3 | Michal   | Niemczak        |   23
             7 | Bozena   | Przykoscinska   |   40
             2 | Wojciech | Kowalski        |   30
             4 | Marcin   | Tracz           |   26
             5 | Irena    | Szalowska       |   55
             4 | Renata   | Zawadzka        |   25
             6 | Wiktoria | Kowalska        |   23
             1 | Jan      | Nowak           |   24
(10 wierszy)
```

Tabela 14.1. Pięć głównych kategorii składniowych SQL

| Lp. | Identyfikatory | Nazwy obiektów | Przykłady | | |
|---|---|---|---|---|---|
| 1 | Literały | Stałe | | | |
| 2 | Operatory | Spójniki | Arytmetyczne – dla tych wyrażeń można używać nawiasów ( ) | iloczyn | * |
| | | | | iloraz | / |
| | | | | modulo | % |
| | | | | suma | + |
| | | | | różnica | - |
| | | | Znakowe | konkatencja | + lub \|\| |
| | | | | symbol wieloznaczny (zastępujący dowolną liczbę znaków) | % |
| | | | | symbol wieloznaczny (zastępujący jeden znak) | - |
| | | | Logiczne | koniunkcja | AND |
| | | | | alternatywa | OR |
| | | | | negacja | NOT |
| | | | Porównania | równy | = |
| | | | | negacja | NOT |
| | | | Porównania | równy | = |
| | | | | mniejszy | < |
| | | | | większy | > |
| | | | | mniejszy lub równy | <= |
| | | | | większy lub równy | >= |
| | | | | różny | != lub <> |
| | | | Typowe dla SQL operatory relacji | operator przynależności do zbioru przedziału domkniętego | BETWEN *n* AND *n* |
| | | | | operator przynależności do zbioru | IN (...) |
| | | | | operator dopasowania do wzorca | LIKE |
| | | | | operator dopasowania do wyrażenia regularnego | REGEXP, RLIKE |
| | | | | operator porównania sprawdzający występowanie znacznika braku wartości NULL | IS NULL |

| Lp. | Identyfikatory | Nazwy obiektów | Przykłady | |
|---|---|---|---|---|
| 3 | Słowa kluczowe | Wyrazy interpretowane przez serwer w określony sposób | Słowa kluczowe to zarezerwowane – podobnie jak w językach programowania – ciągi znaków. Możemy do nich zaliczyć: | |
| | | | Instrukcje | CREATE, SELECT |
| | | | Klauzule | WHERE, JOIN |
| | | | Nazwy typów danych | INTEGER, CHAR |
| | | | Nazwy funkcji systemowych | ISNULL, ABS |
| | | | Nazwy zarezerwowane do użycia w przyszłości | zależnie od bazy danych |
| 4 | Komentarze | Ignorowane | Podwójny znak myślnika (-- –)to komentarz, którego używamy, aby określony wiersz nie był interpretowany. | |
| | | | Jeśli chcemy, aby został zignorowany więcej niż jeden wiersz, warto posłużyć się komentarzem, który odnosi się do bloku instrukcji:<br>/* tu<br>znajduje się<br>blok instrukcji<br>*/ | |
| 5 | Operator konkatencji | | Operator ten pozwala łączyć atrybut z:<br>• innym atrybutem,<br>• literałem,<br>• wyrażeniem arytmetycznym,<br>• wartością stałą.<br>Na podstawie tej operacji tworzona jest jedna wynikowa kolumna. | |

### Słowa kluczowe

SK to sformułowanie „zastrzeżone" dla BD, np. nie możemy kolumny nazwać np. **SELECT**. W związku z tym należy zapoznać się nie tylko ze słowami kluczowymi standardu SQL. lecz także implementacjami tego języka funkcjonującymi w SZBD: MySQL. PostgreSQL, Access itp.
Słowo kluczowe **DISTINCT**. Podczas pracy z bazą danych możemy obsługiwać tabele zawierające powtarzające się wartości tego samego atrybutu w kolejnych krotkach. DISTINCT umożliwia eliminowanie duplikatów, wartości kolejnych wierszy oraz kolumn- w zależności od użycia.
Klauzula **ORDER BY**. To klauzula służąca uporządkowaniu zwracanego wyniku. Pamiętać należy, iż stosujemy ją jako ostatnią klauzulę polecenia SELECT. Dane segregowane są domyślnie rosnąco lub według ustawień bazy danych.
Operator **BETWEEN...AND**. Używany jest do sprawdzenia, czy wartość znajduje się w podanym przedziale.
Operator **IN**. Służy do sprawdzenia, czy wartość znajduje się na zdefiniowanej liście. Operator **LIKE**. Służy do wybierania wartości, które odpowiadają wcześniej ustalonemu wzorcowi. Wzorzec ustalamy za pomocą symboli:
• % - odpowiada dowolnemu ciągowi znaków,
• .-odpowiada jednemu dowolnemu znakowi.
Operator ISNULL służy do wyszukiwania wartości NULL.

### Typy danych

Typ danych określa, jakiego rodzaju informacje mogą być przechowywane w poszczególnych kolumnach tabel lub w zmiennych oraz jakiego typu dane mogą być przekazywane jako parametry wywołania procedury lub funkcji. Listę typów danych zdefiniowanych w standardzie SQL3 zawiera poniższa tabela.

| Kategoria | Przykładowe typy danych | Opis |
|---|---|---|
| Typy liczbowe | INTEGER (INT), SMALLINT | Reprezentują liczby całkowite. |
| | NUMERIC (DECIMAL) | Reprezentuje liczby o określonej skali i precyzji. |
| | REAL | Reprezentuje liczby o zmiennej precyzji. |
| Typy daty i czasu | DATE | Reprezentuje datę. |
| | TIME | Reprezentuje czas. |
| Typy znakowe | CHAR | Reprezentuje ciąg znaków o określonej długości. |
| | VARCHAR | Reprezentuje ciąg znaków o zmiennej długości. |
| | NCHAR, NVARCHAR | Reprezentują ciąg znaków o stałej lub zmiennej długości zakodowanych w UNICODE. |
| Typy binarne | BINARY | Reprezentuje ciąg bitów o określonej długości. |
| | VARBINARY | Reprezentuje ciąg bitów o zmiennej długości. |
| | BLOB | Reprezentuje duże obiekty binarne. |
| Dokumenty XML | XML | Reprezentuje całe dokumenty XML. |

INTEGER (INT) – 4B
SMALLINT – 2B
BIGINT – 8B
REAL – 1E-37 – 1E+37
DOUBLE – 1E-307 – 1E+308
NUMERIC (8 - PRECYZJA,2 - SKALA) np.cena

Literały – ciąg znaków użyty pomiędzy cudzysłowami

**Wartość NULL**

Zgodnie z jednym z postulatów dra Codda serwery bazodanowe powinny spójnie przetwarzać wartość specjalną **NULL**.

Wartość NULL reprezentuje brakujące, nieznane lub nieistotne dane i jest różna od 0 oraz od pustego ciągu znaków. Na przykład brak ceny produktu nie oznacza, że jest on darmowy, a tylko że jego cena nie została jeszcze ustalona. Z powodu występowania wartości NULL w serwerach bazodanowych obowiązuje logika trójwartościowa, a nie dwuwartościowa. Porównanie wartości NULL z dowolną inną wartością daje więc w wyniku wartość nieznaną (ang.Unknown), a nie prawdę lub fałsz.

Reguły przetwarzania wartości NULL są następujące:

- Dwie wartości NULL nie są ani sobie równe, ani różne od siebie. Wartość NULL nie jest też równa, mniejsza czy większa od jakiejkolwiek innej wartości. Sensowne wyniki daje jedynie sprawdzanie (za pomocą operatora IS), czy dana wartość jest nieznana .
- Wynikiem wszystkich operacji zawierających NULL jest wartość NULL, co pokazuje poniższy przykład:

```
SELECT NULL/0, 'Ala' + NULL, 5 + NULL, 10 * NULL, NULL + NULL;
-------------------------------------------------------------
NULL          NULL          NULL          NULL          NULL
```

- Wartość NULL jest ignorowana przez wszystkie funkcje grupujące z wyjątkiem funkcji COUNT(*).