Create calculated field

|  |  |
| --- | --- |
| IF ()  THEN  END | IF ([Orange\_Cap\_Id]=[Player\_Id])  THEN [Player\_Name]  END |
| CELING (number) | zaokrąglenie do góry |
| FLOOR(number) | zaokrąglenie do dołu |
| ROUND(number, [declimals]) | zaokrągla, declimars precyzja |
| DIV(integer1,integer2) | zwraca część całkowitą z dzielenia |
| MAX(number,number) |  |
| SIGN(number) | zwraca znak dla liczb dodatnich (1), 0 i ujemnych (-1) |
| CONTAINS(‘Edureca’, ‘rek’) | sprawdza czy ciąg zawiera podane znaki  zwraca True/False |
| ENDSWITH(‘Edureca’, ‘eca’) | sprawdza czy ciąg kończą podane znaki  zwraca True/False |
| FIND(string, substring, [start]) | zwraca indeks pozycji ciągu w stringu  -start ustawia od kąd ma sprawdzać |
| FINDNTH(string, substring, occurrence) | sprawdza pozycje n tego wystąpienia ciągu w stringu |
| LEFT(string, number)  RIGHT(string, number) | zwraca część strunku na lewo od numeru |
| LEN(string) | zwraca długość stringu |
| LOWER(string)  UPPER(string) | przekształca string na małe litery |
| LTRIM(string)  RTRIM(string)  TRIM(string) | usuwa spacje z początku stringu  na końcu  bez spacji na początku i końcu |
| MAX(a,b)  MIN(a,b) | zwraca wartość najpóźniejszą po posortowaniu alfabetycznym |
| MIN(string, start, [length]) | zwraca ciąg od pozycji indeksu start |
| SPACE(number) | zwraca podaną liczbę spacji |
| SPLIT(string, delimiter, token number) | dzieli string wg znaku I zwraca podany element o numerze tokenu |
| DATE |  |
| DATEADD(date\_part, interval, date) | month, 3, #2004-04-15# przesuwa part(miesiąc) o iterwał i zwraca nową datę |
| DATEDIFF(date\_part, date 1, date2, [start of the week) | zwraca różnicę między datami  week, data, druga data, początek tygodnia – monday |
| DATENAME(date\_part, date)  DATEPART | zwraca część daty – string  zwraca integer |
| DATETRUNC | zwraca początek datego okresu z daty |
| DATE(date) | zwraca dzień z daty jako integer |
| ISDATE | zwraca True jeśli data jest poprawna |
| MAKEDATE(year, month, day)  MAKETIME  MAKEDAYTIME | przekształca dane w datę |
| MAX(a,b)  MIN(a,b) |  |
| MONTH(date) | zwraca miesiąc jako integer |
| NOW()  TODAY() | current date of time |
| YEAR(date) |  |
|  |  |
| Aggregate functions |  |
| ATTR (expression) |  |

<https://www.hackerrank.com/domains/sql>

<https://www.dolthub.com/discover>

<https://www.sql-practice.com/>

<https://www.youtube.com/watch?v=6X2-P9fNVvw&list=PLF2u7Zn-dIxbeais0AkBxUqdWM1hnSJDS&index=2>

<https://platform.stratascratch.com/coding?code_type=1&difficulties=3>

dATA SETS: <https://catalog.data.gov/dataset/?res_format=CSV>

**SQL**

Nazwa kolumny użyta w apostrowach `nazwa`

-- (komentarz)

**USE** sql\_store (nie trzeba później powtarzać z jakiej bazy korzystamy)

**SELECT \***

1, 2 (numery kolumn),

first name, second name, points \* 10 + 10 (można od razu dawać działania matematyczne),

cos + 1 **AS** ‘nazwa\_nowej\_kolumny’,ALIASING ustalanie nazwenictwa kolumn, AS można pominąć

‘wartość, np. old’ można stworzyć nową kolumnę z wartościami

**SELECT DISTINCT** Wartości unikalne

**FROM** sql\_store.orders;

**WHERE** customer\_id = ‘abc’, points > 100 (warunek) > >= < <= = != <> **AND** (first warunek) **OR**

**WHERE NOT** warunek (wartości przeciwne)

**WHERE** warunek != ‘Female” (różne)

**WHERE** person **IN** (‘aaa’, ‘bbb’, ‘cccc’) lub **NOT IN** (‘aaa’, ‘bbb’, ‘cccc’)

**WHERE** points **BETWEEN** 100 **AND** 300 >=100 i <= 300

**WHERE** last\_name **LIKE** ‘b%; **%** dowolny ciąg ‘%b%’ ‘\_\_ys’ **\_**pojedynczy znak

**WHERE** last\_name **REGEXP** ‘field’ (mamy dodatkowe opcje)

**^**początek stringu

**$**koniec stringu ‘filed$’

**I** (logical or - lub) dwa różne wartości ‘kowal|nowak’

**[**abc**]** [nzt]e w danej pozycji szuka znaków z nawiasu ‘[gim]e’ daje ge ie me

**[**a-d**]** wartości z przedziału

**WHERE** pole **IS NULL** (tylko wiersze gdzie nie ma wartości)

**WHERE** pole **IS NOT NULL** (tylko wiersze gdzie są wartości)

**ORDER BY** first\_name (sortowanie) **DESC** (malejąco)

**ORDER BY** kraj **DESC**, miasto **DESC**

**ORDER BY** 1, 2 (numery kolumn wybranych na górze)

**ORDER BY** quantity \* unit\_price **DESC** (można robić wyrażenia)

**ORDER BY**

(not province\_name = 'Ontario'), wyświetla wg pierwszego warunku, później drugiego

province\_name

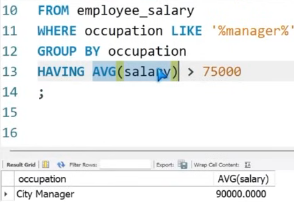
**ORDER BY**

province\_name = 'Ontario' desc,

province\_name

**GROUP BY** gender (zwija do pojedynczych wystąpień, do zastosowania funkcje, podobnie jak DISTINCT)

HAVING AVG(salary)>75000 pozwala na filtrowanie zagregowanych funkcji - warotości z group by



**LIMIT** 300 ogranicza liczbę wyświetlanych rekordów

**LIMIT** 6, 3 (pomiń, następnie weź kolejnych) w tym przykładzie 7-9

**Łączenie tabel**

**DWIE RÓŻNE, JOIN wyświetla zapisy które spełniają warunek**

ON

USING (taka sama nazwa kolumn) (order\_id, product\_id)

USE sql\_inventory;

**SELECT \***

**SELECT** orders.customer\_name, customer.address

**SELECT** o.customer\_name, c.address

**FROM** orders o lub stworzyć skrót (alias)

**JOIN** lub **INNER JOIN** customers c

**ON** orders.customer\_id **=** customers.customer\_id (warunek)

**ON** o.customer\_id **=** c.customer\_id (warunek)

**DWIE TAKIE SAME**

**SELECT \***

**FROM** employees e

**JOIN** employees m

**ON** e.reports\_to **=** m.employee\_id

**WIĘCEJ NIŻ 2 TABELE**

**SELECT \***

**SELECT** o.customer\_name, c.address

**FROM** orders o

**JOIN** customers c

**ON** o.customer\_id **=** c.customer\_id (warunek)

**JOIN** statuses s

**ON** o.customer\_id **=** s.status\_id (warunek)

**POWIĄZANIE Z DWOMA KOLUMNAMI - COMPOUND JOIN CONDITIONS**

**SELECT** \*

**FROM** orders\_items oi

**JOIN** order\_item\_notes oin

**ON** oi.order\_id = oin.order\_id

**AND** oi.product\_id = oin.product\_id

**OUTER JOIN – WYŚWIETLA WSZYSTKIE ZAPISY Z JEDNEJ TABELI I DOPASOWUJE TYLKO DO TYCH CO SPEŁNIAJĄ WARUNEK, pozostałe pola pozostają puste**

**LEFT JOIN, RIGHT JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN (left – wszystkie zapisy z pierwszej tabeli FROM)**

****

**SELECT**

c.customer\_id,

c.firts\_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

**CROSS JOIN - ŁĄCZENIE WSZYSTKICH REKORDÓW Z DWÓCH TABLIC (nie ma przyrównania ON/USING), każdy z każdym**

|  |  |
| --- | --- |
| SELECT  c.first\_name AS customer,  p.name AS product  FROM customers c  CROSS JOIN products p  ORDER BY c.first\_name | SELECT  c.first\_name AS customer,  p.name AS product  FROM customers c, products p  ORDER BY c.first\_name |

**UNIONS ŁĄCZENIE DWÓCH TABEL (selectów), muszą mieć tyle samo kolumn i taki sam typ wartości (dokładanie wierszy)**

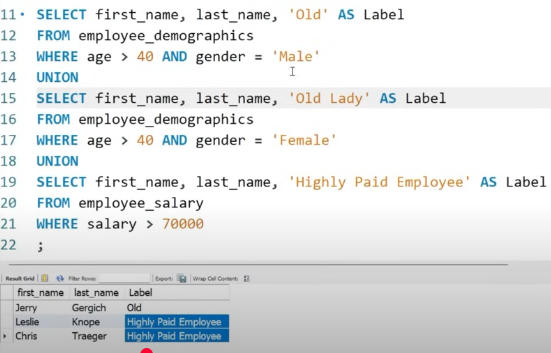
SELECT

order\_id, order\_date, ‘Archived’ AS status

FROM orders

WHERE order\_date < ‘2019-01-01’

**UNION** (pozwala łączyć ze sobą tabele z nieco innymi danymi, np. Old, New, Common)



**UNION DISTINCT** (lista z usuniętymi duplikatami)

**UNION ALL (wszytkie rezultaty)**

SELECT

order\_id, order\_date, ‘Active’ AS status

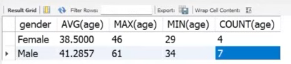
FROM orders

WHERE order\_date > ‘2019-01-01’

SELECT gender, count(AGE)

FROM persons

GROUP BY gender;



**SUBQUERIES** – zapytanie w zapytaniu

Wykorzystujemy dane, np indeksy z jednego SELECT i wykorzystujemy go w drugiem

SELECT \*

FROM demographics

**WHERE** employee\_id **IN**

(**SELECT** emeloyee\_id FROM employee\_salary WHERE dept\_id=1) musi zwrócić jedną koluimnę

**ZLICZENIE WSZYSTKICH WIERSZY W TABELI**

SELECT COUNT(\*) AS liczba\_wierszy FROM nazwa\_tabeli;

**ZLICZENIE WIERSZY SPEŁNIAJĄCYCH WARUNEK**

SELECT COUNT(\*) AS liczba\_wierszy FROM nazwa\_tabeli WHERE kolumna = 'wartość';

**ZLICZENIE UNIKALNYCH WARTOŚCI W KOLUMNIE**

SELECT COUNT(DISTINCT kolumna) AS liczba\_unikalnych FROM nazwa\_tabeli;

**ZLICZENIE WIERSZY W GRUPACH (GROUP BY)**

SELECT kolumna, COUNT(\*) AS liczba\_wierszy

FROM nazwa\_tabeli

GROUP BY kolumna;

**FUNKCJE NA KOLUMNACH**

liczbowe

**AVG**(kolumna) średnia

avg(gender = 'M') średnia przy warunku

**MIN**(kolumna)

**MAX**(kolumna)

**CAST**(distance\_cm AS DECIMAL(10, 2)) – zmienia typ danych

DECLIMAL

FLOAT(10,2)

przy dzieleniach aby nie było 0 można

1. \* male\_patients wymnożyć licznik – taki trik albo użyćCAST

CAST(male\_patients AS DECIMAL(10,2)) / total\_patients \* 100

**ROW\_NUMBER() OVER()** wyświetla nr kolejnego wiersza

**WYŚWIETLA WARTOŚĆ Z POPRZEDNIEGO WIERSZA**

LAG(count(admission\_date)) OVER(ORDER BY admission\_date)

lag(admission\_day, 1) over()

tekstowe

**COUNT**(kolumna) ilość wystąpień

**COUNT**(\*) suma wszytkie wiersze przy grupowaniu

**COUNT**(CASE WHEN gender = 'F' THEN 1 END) AS female\_patients\_number, wybrane wiersze, wg warunku przy grupowaniu - suma

**LENGTH**(kolumna) – ilość znaków

**LEN**(kolumna) – ilość znaków

**UPPER**(kolumna) – zamienia na DUŻE LITERY

**LOWER**(kolumna) – zamienia na male litery

**TRIM** – usuwa spacje z przodu i tyłu wartości kolumny

**LTRIM** – usuwa spacje z lewej

**RTRIM** – usuwa spacje z prawej

**LEFT**(kolumna, 4) – wyświetla pierwsze n znaków z lewej

**RIGHT**(kolumna, 4) – wyświetla pierwsze n znaków z prawek

**SUBSTRING**(kolumna, pozycja\_startowa, znaków) – wyświetla n znaków od podanej pozycji

(do wydobywania np. miesiąca)

**REPLACE** (kolumna, co, czym\_zastępujemy) – zamienia string innym

**LOCATE**(co\_szukamy, gdzie) – podaje pozycję, miejsce wystąpienia znaków w danym ciągu, wyświetla 0 jeśli nie ma danego stringu

**CONCAT**(kolumna1, ‘ ‘, kolumna2) – łączy kolumny, wartości itp. lub znak podwójny kol1 || ‘ ‘ || kol2

Tworzenie opisów wg danego warunku

**CASE** – tworzenie kolumny z wartościami wg warunku

**WHEN** age <= 30 THEN ‘Young’

**WHEN** age BETWEEN 31 and 50 THEN ‘Old’

**WHEN** age >= 50 **THEN** ‘On Death`s Door’

ELSE ‘aaa’

**END**

**END** AS nazwa\_nowej\_kolumny

**DODANIE WIERSZA/WIERSZY DO TABELI**

**INSERT INTO** customers (kolejne kolumny trzeba określić co mają zawierać)

**VALUES** (**DEFAULT**, ‘John’, ‘Smith’, **NULL**, **DEFAULT**)

**INSERT INTO** customers (first\_name, last\_name) określamy do których kolumn chcemy wstawiać, kolejność dowolna – kolumny nie wymienione przyjmą wartość DEFAULT

**VALUES** ( ‘John’, ‘Smith’),

( ‘Max’, ‘Kolonko’)

**SELECT LAST\_INSERT\_ID()** BUILD IN FUNCTION

Sprawdzenie jaki był index ostatnio dodanego wiersza, można go wykorzystać do dodania do innej tabeli

INSERT INTO orders (customer\_id, order\_date, status)

VALUES (1, ‘2022-02-23; 1)

INSERT INTO order\_items

VALUES

(LAST\_INSERT\_ID(), 1, 1, 2.95),

(LAST\_INSERT\_ID(), 2, 1, 3.95),

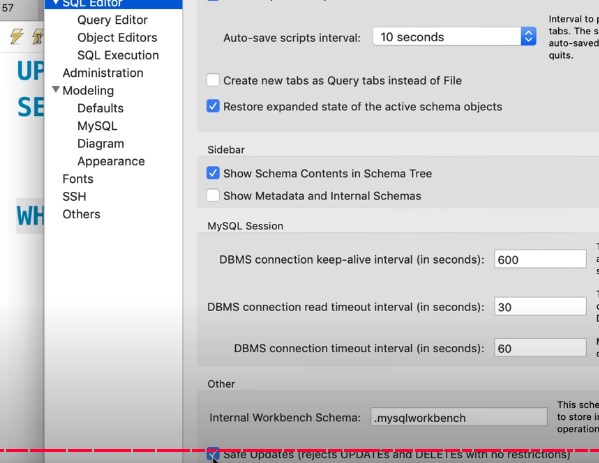
**AKTUALIZOWANIE / UPDATE REKORDU/ÓW**

**UPDATE** invoices

**SET** payment\_total = invoice\_total \*0.5, payment\_date= “2023-02-11”, status=NULL, order\_status=DEFAULT

**WHERE** invoice\_id = 1

Do aktualizowania wielu rekordów trzeba włączyć opcję w MYSQL



**UPDATE** invoices

**SET** payment\_total = invoice\_total \*0.5, payment\_date= “2023-02-11”, status=NULL, order\_status=DEFAULT

**WHERE** invoice\_id IN (3, 4, 5)

(lub kolejny drugi SELECT)

**WHERE client\_id =** (jeden wiersz – id wyszukane)

(SELECT cliend\_id from clients WHERE name = ‘Smith’)

**WHERE client\_id IN** (kilka wieszy id wyszukane)

(SELECT cliend\_id from clients WHERE name IN (‘CA’, “NY’))

**USUWANIE WIERSZY**

DELETE FROM invoices (wszystkie rekordy)

DELETE FROM invoices (wkazany rekord)

WHERE invoice\_id = 1

DELETE FROM invoices (wybrane rekordy)

WHERE invoice\_id = (

SELECT invoice\_id from clients WHERE name IN (‘CA’, “NY’))

**KOPIOWANIE DANYCH MIĘDZY TABELAMI**

Tracimy atrybuty pół I inteksowanie

CREATE TABKE orders\_archived AS

SELECT \* FROM orders

INSERT INTO orders\_archived

SELECT \*

FROM orders

WHERE order\_date <’2023-01-21’

**WINDOW FUNCTION**

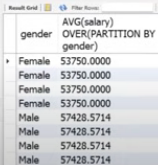
zamiast grupowania gdzie prezentowane są **zbiorcze wyniki grup z Partition by, funkcje WINDOW – poniżej średnia - do każdego wiersza** dają rezultat

SELECT gender, **AVG(salary) OVER(PARTITION BY gender)**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;



**Suma krocząca – rolling total – dodawanie wartości do poprzedniej np. skumulowanie – kolejność wyświetlana po orderby**

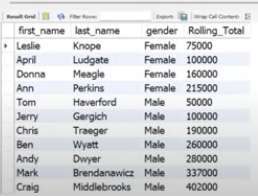
SELECT dem.first name, dem.last\_name, gender,

**SUM(salary) OVER(PARTITION BY gender ORDER BY dem.employee\_id) AS Rolling\_Total**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;

****

**NUMBER WIERSZA KOLEJNO DLA KAŻDEJ GRUPY**

SELECT dem.first name, dem.last\_name, gender, salary

**ROW\_NUMBER() OVER(PARTITION BY gender)**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;



**NUMBER WIERSZA KOLEJNO DLA KAŻDEJ GRUPY Z SORTOWANIEM WG WYBRANEJ KOLUMNY (bez powtórzeń, nawet dla tych samych wartości)**

SELECT dem.first name, dem.last\_name, gender, salary

**ROW\_NUMBER() OVER(PARTITION BY gender ORDER BY salary DESC)**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;



**RANK() NUMBER WIERSZA KOLEJNO DLA KAŻDEJ GRUPY Z SORTOWANIEM WG WYBRANEJ KOLUMNY Z POMINIĘCIEM (z powtórzeniem miejsca dla tych samych wartości) – nr pozycji, mogą być exekwo wówczas pomija kolejne miejsce**

SELECT dem.first name, dem.last\_name, gender, salary

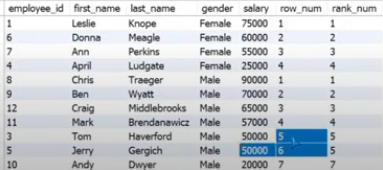
**ROW\_NUMBER() OVER(PARTITION BY gender ORDER BY salary DESC) AS row\_num,**

**RANK() OVER(PARTITION BY gender ORDER BY salary DESC) AS rank\_num**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;



**DENSE\_RANK() NUMBER WIERSZA KOLEJNO DLA KAŻDEJ GRUPY Z SORTOWANIEM WG WYBRANEJ KOLUMNY BEZ POMINIĘĆ NUMERÓW (z powtórzeniem miejsca dla tych samych wartości)**

SELECT dem.first name, dem.last\_name, gender, salary

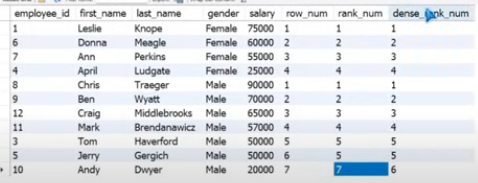
**ROW\_NUMBER() OVER(PARTITION BY gender ORDER BY salary DESC) AS row\_num,**

**DENSE\_RANK() OVER(PARTITION BY gender ORDER BY salary DESC) AS dense\_rank\_num**

FROM employee\_demographics dem

JOIN employee\_salary sal

ON dem.employee\_id = sal.employee.id;

****

ROW\_NUMBER() OVER (PARTITION BY gender order by salary DESC) numer\_porz,

RANK() OVER (PARTITION BY gender order by salary DESC) miejsce\_z\_przeskokiem,

DENSE\_RANK() OVER (PARTITION BY gender order by salary DESC) miejsce\_bez\_przeskoku

**WYŚWIETLA WARTOŚĆ Z POPRZEDNIEGO WIERSZA**

**LAG**(count(admission\_date)) OVER(ORDER BY admission\_date)

lag(admission\_day, 1) over()

**WITH nazwa AS (select) CTE, Common Table Expression – prawie jak TYMCZASOWE TABELE** (można później używać po nazwie) do prostych rzeczy, 1 level wyżej

do zaawansowanych obliczeń, bardziej czytelne obliczenia, od razu trzeba ich użyć bo później po ; jest błąd

definiowanie:

**WITH CTE\_Example (**można podać kolejno nazwy kolumn, później nie trzeba opisywać) **AS**

(select ….)

**TEMPORARY TABLES – TYMCZASOWE TABLICE WIDOCZNE WYŁĄCZNIE W DANEJ SESJI, RÓNIEŻ W NOWYCH OKNACH** do zaawansowanych rzeczy, ponownego wykorzystania danych

od nowa:

**CREATE TEMPORARY TABLE** temp\_table

(first\_name varchar(50),

last\_name varchar(50),

favorite\_movie varchar(100);

INSERT INTO temp\_tamle VALUES(….

z zapytania:

CREATE TEMPORARY TABLE subset\_data AS

SELECT column1, column2, column3

FROM original\_table

WHERE condition;

**STORED PROCEDURES – kod który można zapisać i wielokrotnie wykorzystywać**

**CREATE PROCEDURE** large\_salaries2(parametr który można przekazać do procedury, **employee\_id\_param**)

SELECT … **;**

lub z parametrem po nazwie musi być typ danych, konwencja jest nazwa\_param

CREATE PROCEDURE large\_salaries2(nazwa\_parametru INT)

Wywołanie

**CALL** large\_salaries()**;**

call large\_salaries2(25); z parametrem

DELIMITER $$ (określenie co kończy procedurę, dzięki temu może być bardzo złożona a nie tylko do średnika ; który rozdziela zapytania)

DELIMITER $$

**CREATE PROCEDURE** large\_salaries2()

**BEGIN**

SELECT \*

FROM employee\_salary

WHERE employee\_id = parametr;

**END $$**

DELIMITER ;

USE ‘nazwa\_bazy’;

DROP PROCEDURE IF EXISTS ‘nazwa\_procedury’;

….

**TRIGGER – AKCJA WYWOŁYWANA OKREŚLONYM WYDARZENIEM (pozwala oszczędzić czas, nie trzeba ręcznie aktualizować różnych tabel)**

DELIMITER $$

**CREATE TRIGGER** employee\_insert2

**AFTER INSERT ON** employee\_salary

**BEFORE INSERT ON** employee\_salary (lub)

**FOR EACH ROW**

BEGIN

**INSERT INTO** employee\_demographics (employee\_id, first\_name, last\_name)

**VALUES** (**NEW**.employee\_id,**NEW**.first\_name,**NEW**.last\_name); lub OLD (rows that were deleted or updated)

END $$

DELIMITER ;

zadziała jeśli

INSERT INTO employee\_salary (id, name, lastname,occupation,salary,dept, NULL)

VALUES(12, ‘sss’ …..

**EVENTS – AKCJA WYWOWYWANA W OKREŚLONYM CZASIE**

SHOW EVENTS;

-- we can drop or alter these events like this:

DROP EVENT IF EXISTS delete\_retirees;

DELIMITER $$

**CREATE EVENT** delete\_retirees

**ON SCHEDULE EVERY 30 SECOND**

DO BEGIN

DELETE

FROM parks\_and\_recreation.employee\_demographics

WHERE age >= 60;

END $$

now let's look at Events

-- Now I usually call these "Jobs" because I called them that for years in MSSQL, but in MySQL they're called Events

-- Events are task or block of code that gets executed according to a schedule. These are fantastic for so many reasons. Importing data on a schedule.

-- Scheduling reports to be exported to files and so many other things

-- you can schedule all of this to happen every day, every monday, every first of the month at 10am. Really whenever you want

-- This really helps with automation in MySQL

-- let's say Parks and Rec has a policy that anyone over the age of 60 is immediately retired with lifetime pay

-- All we have to do is delete them from the demographics table

--Data Cleaning—

1. Remove duplicates

2. Stanardize the Data

3. Null Values or blank values

4. Remove Any Columns or Rows

Copy data – leave Raw data, best practice – nie pracuje się na raw data

BEST PRACTICE TIP: do not work on Raw Data tables

**Tworzenie nowej tabeli na podstawie istniejącej**

**CREATE TABLE** nazwa\_tworzonej\_tabeli

**LIKE** nazwa\_tabeli\_na\_podstawie\_ktorej\_ma\_stworzyć\_strukturę\_nowej;

**Kopiowanie danych między tabelami**

**INSERT** layoffs\_staging

SELECT \*

FROM layoffs;

**REMOVE DUPLICATES**

Oznaczenie wierszy zduplikowanych za pomocą row\_num, wartość większa niż 1

Warto sprawdzić czy dobrze zadziałało i czym są duplikaty

Trzeba stworzyć kopię tabeli z dodatkową kolumną row\_num, gdzie w przypadku row\_num >1

wówczas kasujemy wiersze z tym indeksem

WITH duplicate\_cte AS (

SELECT \*,

ROW\_NUMBER() OVER(

PARTITION BY company, industry, total\_laid\_off, percentage\_laid\_off, `date`) AS row\_num

FROM layoffs\_staging

)

Sprawdzenie czy wystąpiły duplikaty

SELECT \*

FROM duplicate\_cte

WHERE row\_num > 1;

DELETE

FROM duplicate\_cte

WHERE row\_num >1;

**TWORZENIE NOWEJ TABELI**

CREATE TABLE `layoffs\_staging2` (

`company` text PRIMARY KEY,

`location` text,

`industry` text,

`total\_laid\_off` text,

`percentage\_laid\_off` text,

`date` text,

`stage` text,

`country` text,

`funds\_raised` double DEFAULT NULL,

`row\_num` INT

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

INSERT INTO layoffs\_staging2

SELECT \*,

ROW\_NUMBER() OVER(

PARTITION BY company, location, industry, total\_laid\_off, percentage\_laid\_off, `date`, stage, country, funds\_raised) AS row\_num

FROM layoffs\_staging;

DELETE

FROM layoffs\_staging2

WHERE row\_num > 1;

**-- STANDARDIZING DATA**

Sprawdzamy każdą kolumnę, na spacje, błędy, powtórzenia, puste wartości

Finding issues in data and fixing it

spaces

TRIM – usuwa spacje na samym początku i na samym końcu

**UPDATE** layoffs\_staging2

**SET** company = TRIM(company);

Usunięcie niepoprawnych kategorii



**UPDATE** layoffs\_staging2

**SET** industry = "Crypto"

**WHERE** industry like "Cryp%";

Usuwa podany znak

TRIM(**TRAILING** ‘.’ FROM country)

Zamianie wartości puste na NULL aby łatwiej było czyścić dane

UPDATE layoffs\_staging2

SET industry = NULL

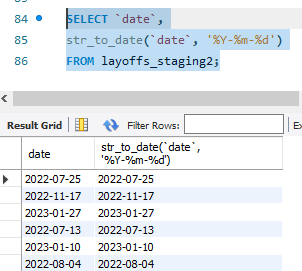
WHERE industry ='';

Zmiana daty na format data

SELECT `date`,

str\_to\_date(`date`, '%Y-%m-%d')

FROM layoffs\_staging2;

****

Zmiana typu kolumny na date

**ALTER TABLE** layoffs\_staging2

**MODIFY COLUMN** `date` **DATE**;

Możemy teraz stosować funkcje wyodrębniające np. rok

SELECT YEAR(`date`)

FROM layoffs\_staging2;

Pokazuje różne wartości z kolumny

SELECT distinct industry

FROM layoffs\_staging2;

3. Null Values or blank values

SELECT distinct total\_laid\_off

FROM layoffs\_staging2

WHERE total\_laid\_off **IS NULL** or total\_laid\_off = **'';**

**Kopiowanie ze znanych**

**Wyszukujmy przypadki gdzie w danej kolumnie są brakujące wartości, które już zostały zdefiniowane**

SELECT table1.industry, table2.industry

FROM layoffs\_staging2 table1

JOIN layoffs\_staging2 table2

ON table1.company = table2.company

AND table1.location = table2.location

WHERE (table1.industry IS NULL OR table1.industry = '')

AND (table2.industry IS NOT NULL and table1.industry <>'') ;

**Uzupełniamy tym co jest**

UPDATE layoffs\_staging2 table1

JOIN layoffs\_staging2 table2

ON table1.company = table2.company

AND table1.location = table2.location

SET table1.industry = table2.industry

WHERE (table1.industry IS NULL OR table1.industry = '')

AND (table2.industry IS NOT NULL and table1.industry <>'');

**Obliczenia na podstawie istniejących danych z innych kolumn**

**Pozbycie się wierszy bez danych**

SELECT \*

FROM layoffs\_staging2

WHERE (total\_laid\_off IS NULL or total\_laid\_off = '') and (percentage\_laid\_off = '' or percentage\_laid\_off IS NULL);

**DELETE**

FROM layoffs\_staging2

WHERE (total\_laid\_off IS NULL or total\_laid\_off = '') and (percentage\_laid\_off = '' or percentage\_laid\_off IS NULL);

**Usunięcie kolumny**

ALTER TABLE layoffs\_staging2

DROP COLUMN row\_num;

**SUMOWANIE WG MIESIĘCY**

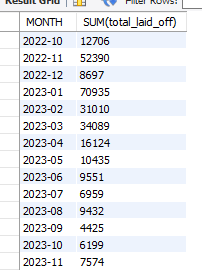
SELECT SUBSTRING(`date`, 1, 7) AS `MONTH`, SUM(total\_laid\_off)

FROM layoffs\_staging2

WHERE SUBSTRING(`date`, 1, 7) IS NOT NULL or SUBSTRING(`date`, 1, 7) <> ''

GROUP BY `MONTH`

ORDER BY 1 ASC;



**SUMOWANIE NARASTAJĄCO**

WITH Rolling\_Total AS

(

SELECT SUBSTRING(`date`, 1, 7) AS `MONTH`, SUM(total\_laid\_off) AS total\_off

FROM layoffs\_staging2

WHERE SUBSTRING(`date`, 1, 7) IS NOT NULL or SUBSTRING(`date`, 1, 7) <> ''

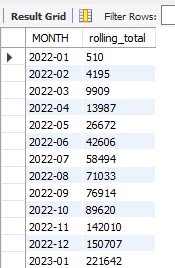
GROUP BY `MONTH`

ORDER BY 1 ASC

)

SELECT `MONTH`, SUM(total\_off) OVER(ORDER BY `MONTH`) AS rolling\_total

FROM Rolling\_Total;



RANKING FIRM WG OBRÓTÓW W KOLEJNYCH LATACH

CTE in CTE

WITH

Company\_Year (company, years, total\_laid\_off) AS (

SELECT company, YEAR(`date`), SUM(total\_laid\_off)

FROM layoffs\_staging2

GROUP BY company, YEAR(`date`)

),

Company\_Year\_Rank AS (

SELECT \*, DENSE\_RANK() OVER(PARTITION BY years ORDER BY total\_laid\_off DESC) as Ranking

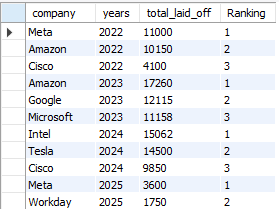
FROM Company\_Year

WHERE years IS NOT NULL)

SELECT \*

FROM Company\_Year\_Rank

WHERE Ranking <=3;



SELECT \*

FROM Company\_Year\_Rank

WHERE Ranking <=3;