KPI: Key Performance Indicators

https://www.w3schools.com/sql/default.asp

https://www.codecademy.com/courses/sql-analyzing-business-metrics/lessons/advanced-aggregates/exercises/hello-advanced-aggregates?action=lesson_resume

one of the most influential companies in the world **MySQL**

- SELECT DISTINCT
- Count ()
- The WHERE clause is used to filter records that fulfill a specified condition
- ORDER BY ASC/DESC
- **INSERT INTO** Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
  VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
- where city is null / is not null;
- UPDATE Customers
  SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
  WHERE CustomerID = 1;
  Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. **If you omit the WHERE clause, all records in the table will be updated!**
- DELETE FROM *table_name* is used to delete existing **records** in a table
  WHERE *condition*;
- SQL SERVER: Select top 3 From … where…
  MYSQL: SELECT … LIMIT 3;
  Oracle: Select… Where ROWNUM<=3;
- MIN () returns the smallest value of selected column.
- Like %: zero, one, or multiple characters, _: a single character
- Wildcards: '[bsp]%' '[a-c]%' '[!bsp]%' Square Brackets[] quote marks''
- IN /NOT…IN…: WHERE Country IN ('Germany', 'France', 'UK');或者
  WHERE Country IN (SELECT Country FROM Suppliers);
- BETWEEN…AND / NOT BETWEEN…AND…operator selects values within a given range. The values can be numbers, text, or dates.
  JOIN
- SELECT…FROM…A INNER JOIN…B ON…
  LEFT JOIN RIGHT JOIN FULL OUTER JOIN SELF JOIN
  UNION : selects only **distinct** values by default

  Each SELECT statement within UNION must have the same number of columns

  The columns must also have similar data types

  The columns in each SELECT statement must also be in the same order

- UNION ALL: To allow **duplicate** values, use UNION ALL:

```
SELECT City FROM Customers
UNION
```

```
SELECT City FROM Suppliers
ORDER BY City;
```

- GROUP BY is often used with aggregate functions to group the result.
- Having the HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

- EXISTS: is used to test for the existence of any record in a subquery.
- ANY/ ALL

```
SELECT ProductName FROM Products
WHERE ProductID
= ANY (SELECT ProductID FROM OrderDetails WHERE Quantity > 99);
```

SQL statement returns TRUE and lists the productnames IF ANY satisfied.

- **SELECT INTO** copies data from one table into a new table.

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017 IN 'Backup.mdb' (another database)
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

- **INSERT INTO SELECT**

requires that **data types** in source and target tables **match**
The existing records in the target table are unaffected

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

- SQL Comments: --; or /*……*/

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

SQL Database

- Create database…;
- Drop database…;
-

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
```

```
FROM existing_table_name
WHERE ....;
```

- `DROP TABLE Shippers;`
- `TRUNCATE TABLE table_name;` is used to delete the data inside a table, but not the table itself.
- `ALTER TABLE table_name`
  `ADD column_name datatype;`
  `DROP COLUMN column_name;`
  `ALTER/MODIFY COLUMN column_name datatype;` to change the data type
- **SQL Constraints**
  NOT NULL - *Ensures that a column cannot have a NULL value*
  UNIQUE - *Ensures that all values in a column are different*
  PRIMARY KEY-*A combination of a **NOT NULL** and **UNIQUE.** Uniquely identifies each row in a table*
  FOREIGN KEY - *Uniquely identifies a row/record in another table*
  CHECK - *Ensures that all values in a column satisfies a specific condition*
  DEFAULT - *Sets a default value for a column when no value is specified*
  INDEX - *Use to create and retrieve data from the database very quickly*

- NOT NULL:
```
CREATE TABLE Persons (
          FirstName varchar(255) NOT NULL  PRIMARY KEY,
          Age int,
          UNIQUE(ID));
```

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
DROP CONSTRAINT UC_Person;
CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
```

Note: In the example above there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

FOREIGN KEY：is a key used to link two tables together. points to a PRIMARY KEY in another table.

- CHECK
- CREATE TABLE Persons

```
CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
```

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

- DEFAULT

```
CREATE TABLE Persons(
City varchar(255) DEFAULT 'Sandnes')
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
ALTER COLUMN City DROP DEFAULT;
```

- INDEX: Indexes are used to retrieve data from the database very fast

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, **only create indexes on columns** that will be frequently searched against.

CREATE UNIQUE INDEX

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
DROP INDEX index_name;
```

- Auto Increment

```
CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT);
```

- VIEW: a virtual table based on the result-set of an SQL statement

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName, Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName;
DROP VIEW view_name;
```

- INJECTION: one of the most common web hacking techniques

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

SQL Hosting

SQL Server: for database-driven web sites with high traffic， Microsoft
Oracle: a very powerful, robust and full featured SQL database system.
MySQL: MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions.

SQL Aggregate Functions:

AVG() COUNT() FIRST() LAST() MAX() MIN() ROUND() SUM()

SQL Date functions:

NOW() CURDATE() **DATEDIFF()** DATE_FORMAT() GETDATE()

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

NULL functions:

```
SELECT ProductName,UnitPrice*(UnitsInStock+IFNULL(UnitsOnOrder,0))
FROM Products
```

SQL DATA TYPES

Daily revenue:

```
select date(created_at),round(sum(price),2)
from purchases
group by 1
order by 1;
```

```
select
  date(g1.created_at) as dt,
  count(distinct g1.user_id)as total_users,
  count(distinct g2.user_id)as retained_users
```

```
from gameplays as g1
left join gameplays as g2 on
    g1.user_id = g2.user_id
    and date(g1.created_at)=date(datetime(g2.created_at,'-1 day'))
group by 1
order by 1
limit 100;
```

//

https://www.youtube.com/watch?v=QFj-hZi8MKk

- SQL Server interview question :- Explain RowNumber,Partition,Rank and DenseRank ?

select row_number() over (order by customername) as ordernumber,

- **row_number()** generate **unique** number
- **Rank()** for repeated data, rank function will generate the same number instead of unique (use the same number for repeated data)
- **Dense_rank()** based on rank() using continue numbers and do not skip a number
- **Partition by** other than group by , it do not change the output of the row

User-defined variables

Declare + initialize

Way1: SET @passing=66

Way2: Select @name :='lu lu'

//

找出同时在一月和二月都看过电影的 CostumerID

```
Select CustomerID from Movie m1, Movie m2
Where m1.CustomerID = m2.CustomerID
And Month(m1.DATE) = '1'
And Month(m2.DATE) = '2'
Order by CustomerID DESC
```

选出最受欢迎的电影

```
Select m1.Title from Movie m1
group by m1.Title
order by count(distinct(m1.customerID) desc
limit 1
```

#175 combine two tables

```
select FirstName,LastName,a.City, a.State
from Person AS p
left join Address as a
on p.PersonID=a.PersonId;
```

#176 Second Highest Salary

```
select Max(Salary) as SecondHighestSalary
from Employee
where Salary<
(select Max(Salary)
from Employee);
//////////////////////
select Salary
from employee
GROUP BY SALARY
ORDER BY SALARY
LIMIT 1,1;
```

177. Nth Highest Salary

181. Employees Earning More Than Their Managers

```
select a.Id,a.Name,a.Salary,a.ManagerId
From Employee as inner join Employee as b
on a.ManagerId=b.Id
where a.Salary>b.Salary;
```

182. Duplicate Emails (inline view)

```
select Email
from (
select count(Email) as count,Email
From Person as p
group by Email)
where Count>1;
```

183. Customers Who Never Order

```
select c.Name as Customers
from Customersa as c
left join
Ordersa as o
on c.Id=Customerid
where o.customerid is null
```

196. Delete Duplicate Emails

```
DELETE FROM Person
WHERE
    Person.Id NOT IN (SELECT
        minId
    FROM
        (SELECT
            MIN(Id) AS minId, Email
        FROM
            Person
        GROUP BY Email) AS tmp);
```

197. Rising Temperature

```
SELECT w1.Id FROM Weather w1, Weather w2
WHERE DATEDIFF(w1.Date, w2.Date)=1
AND w1.Temperature>w2.Temperature;
```

178. Rank Scores

```
select Score, DENSE_RANK() over (order by Score Desc) as Rank
from xx.dbo.Scores;
```

180. Consecutive Numbers

```
SELECT *
FROM Logs L1, Logs L2, Logs L3
WHERE (L1.Id = L2.Id + 1 AND L1.Num = L2.Num) AND
  (L1.Id = L3.Id + 2 AND L1.Num = L3.Num)
```

184. Department Highest Salary

```
select d.Name as Department, e.Name as Employee,m.Salary as Salary
from
(select d.Name as Department,max(Salary) as Salary
from xx.dbo.Employee as e
inner join xx.dbo.Department as d
on e.DepartmentId=d.Id
Group by d.Name) as m,
xx.dbo.Employee as e,
xx.dbo.Department as d
where e.DepartmentId=d.Id
and e.Salary=m.Salary
and m.Department=d.Name
;
```