Business Intelligence

Lecture 4

SQL – DML and DDL

SQL - Structured Query Language

- Query Language
- Table Oriented
- Fast and Easy access to our data
- CRUD - Create, Read, Update, Delete

SQL

DML - Data Manipulation Language

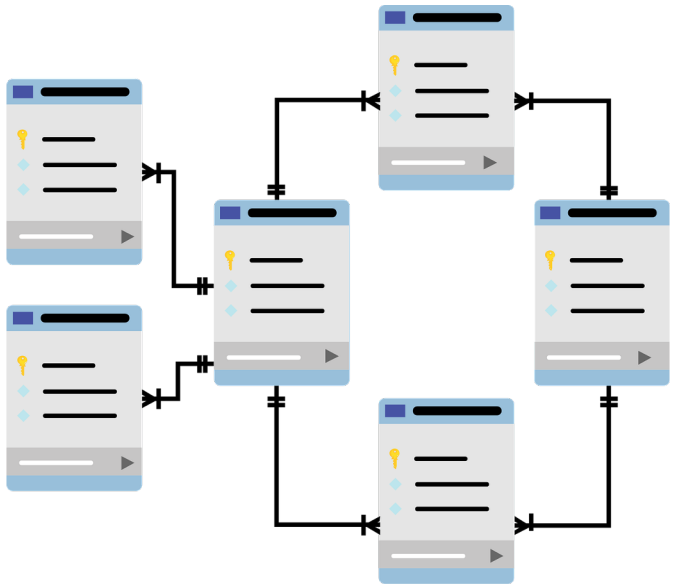
a. Select
b. Insert
c. Update
d. Delete

DDL - Data Definition Language

a. Create
b. Drop
c. Alter

Overview

What happens when we want to handle a few tables which relative one to each other ?



Foreign Key

Courses			
StudentID	CourseNumber	CourseName	Grade
111	289	DB	96
111	281	Algo	85
222	281	Algo	78

Students		
FirstName	LastName	StudentID
Avi	Cohen	111
Dan	Israeli	222
Ofer	Bar	333

Primary Key

How to merge two tables ? - Option 1

```
SELECT FirstName, CourseName, Grade

FROM Students as S, Courses as C

Where (S.StudentID = C.StudentID) and (C.CourseName = 'Algo')
```

How to merge two tables ? - Option 2

```
SELECT FirstName, CourseName, Grade

FROM Students as S JOIN Courses as C

ON S.StudentID = C.StudentID

Where (C.CourseName = ‘Algo’)
```

ON VS. USING

- When using JOIN statement we need to understand according to which column we want to join the tables.
- We will use ON when the columns name is not equal.
- We will use USING when the columns name is equal.

How to merge two tables ? - Option 3

```
SELECT FirstName, CourseName, Grade

FROM Students JOIN Courses

USING (StudentID)

Where (CourseName = 'Algo')
```

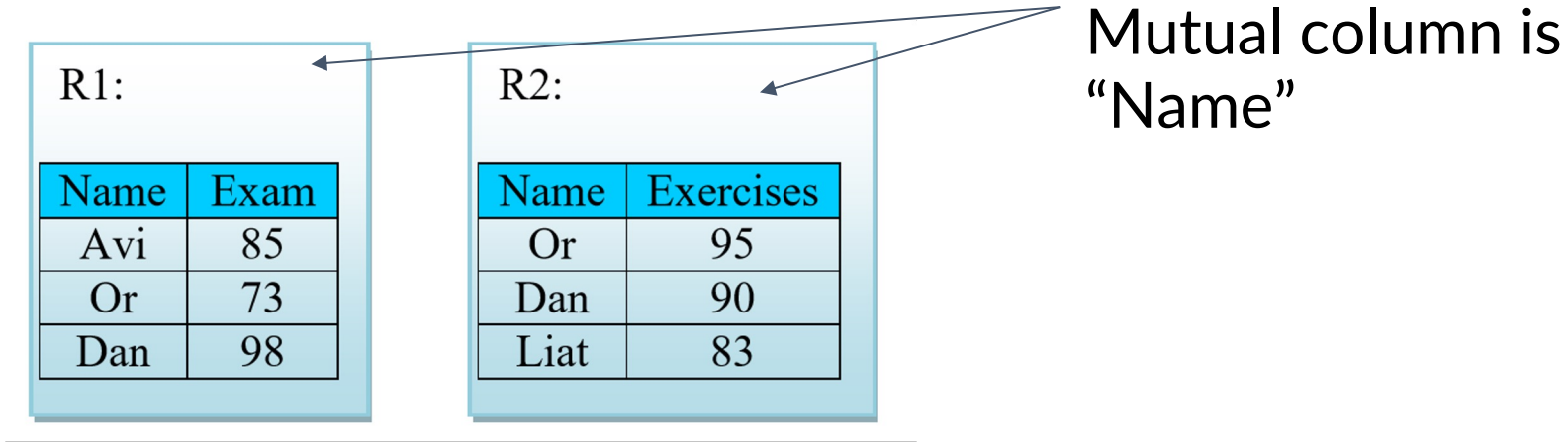
FirstName	CourseName	Grade
Avi	Algo	85
Dan	Algo	78

JOIN Types

- INNER JOIN
- OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

Let’s start with an example..

Given 2 Table called R1, R2:



LEFT OUTER JOIN

What happens here? Why Avi has “null” value?

What happens if we use only “JOIN”?

Name	Exam	Exercise
Avi	85	Null
Or	73	95
Dan	98	90

RIGHT OUTER JOIN

What happens here? Why Avi has “null” value?

What happens if we use only “JOIN”?

Name	Exam	Exercise
Or	73	95
Dan	98	90
Liat	Null	83

FULL OUTER JOIN

What happens here? Why Avi has "null" value?

What happens if we use only "JOIN"?

Name	Exam	Exercise
Avi	85	Null
Or	73	95
Dan	98	90
Liat	Null	83

Handling NULL values

- Sometimes part of our data is NULL, from many reasons:
  - Result of left/right/full outer join
  - Missing data while insert a new row
  - etc.
- What if we want to know who is the problematic data?

Example

Given 2 simple tables (Student and Courses, the favorite example):

Students		
FirstName	LastName	StudentID
Avi	Cohen	111
Dan	Israeli	222
Ofer	Bar	333

Courses			
StudentID	CourseNumber	CourseName	Grade
111	289	DB	96
111	281	Algo	85
222	281	Algo	78

Example

SELECT FirstName, LastName, CourseNumber

FROM Student LEFT OUTER JOIN Courses

USING (StudentID)

FirstName	LastName	CourseNumber
Avi	Cohen	289
Avi	Cohen	281
Dan	Israeli	281
Ofer	Bar	Null

Example

We store the previous query on new table called NewCourses.

```
SELECT *

FROM NewCourses

WHERE CourseNumber is NULL
```

FirstName	LastName	CourseNumber
Ofer	Bar	Null

Important Notes

- As we use “is NULL” , we can use “is NOT NULL”.
- Wrong Usage: FirstName = ‘NULL’

NULL is not a String!

CREATE Tables

```
CREATE TABLE table_name
(
        column_name1 data_type,
        column_name2 data_type,
        column_name3 data_type,
        ...
)
```

1 - Define your table schema + types

```
CREATE TABLE Students
(
        Id int NOT NULL,
        First_Name text,
        Last_Name text,
        Birth_Date date,
        Hour_Salary float
)
```

2 - Insert Values (with the same order!)

INSERT INTO Students

VALUES(123456789, ‘Israel’, ‘Israeli’, ‘1980-03-20’, 120.40)

3 - Check your Insertion

```
SELECT *

FROM Students
```

Students				
Id	First_Name	Last_name	Birth_date	Hour_Salary
0123456789	Israel	Israeli	20/03/1980	120.40

SubQuery with ALL

We want to find the city name where all amounts bigger (or equals) than all average amount of all cities.

```
SELECT City
FROM Students
GROUP BY City
HAVING AVG(Amount) >= ALL (SELECT AVG(Amount)
                           FROM Students
                           GROUP BY City)
```

2400
3000
2100
1000

City
Tel-Aviv

SubQuery with IN

We want to find all names which lives in one of the following cities: Eilat, Haifa, Jerusalem.

```
SELECT Name
FROM Students
WHERE City IN ('Eilat', 'Haifa', 'Jerusalem')
```

Name
Haim Itzhak

SubQuery with EXIST

What is the output of the following query?

```
SELECT SUM(Amount)
FROM Students
WHERE EXISTS (SELECT *
                FROM AgudaMembers as AM
                WHERE AM.Name = Students.Name)
```

SUM(Amount)
13,100

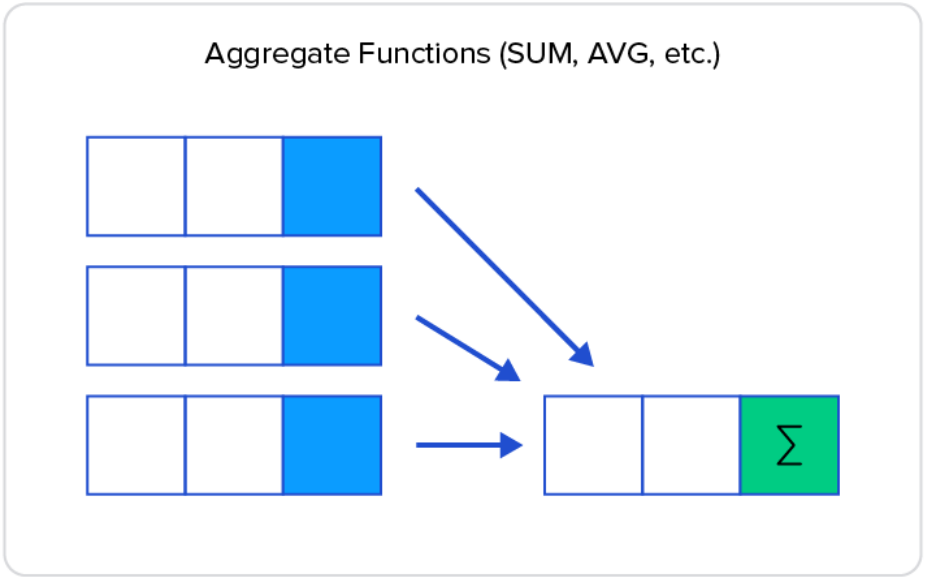
Window Functions

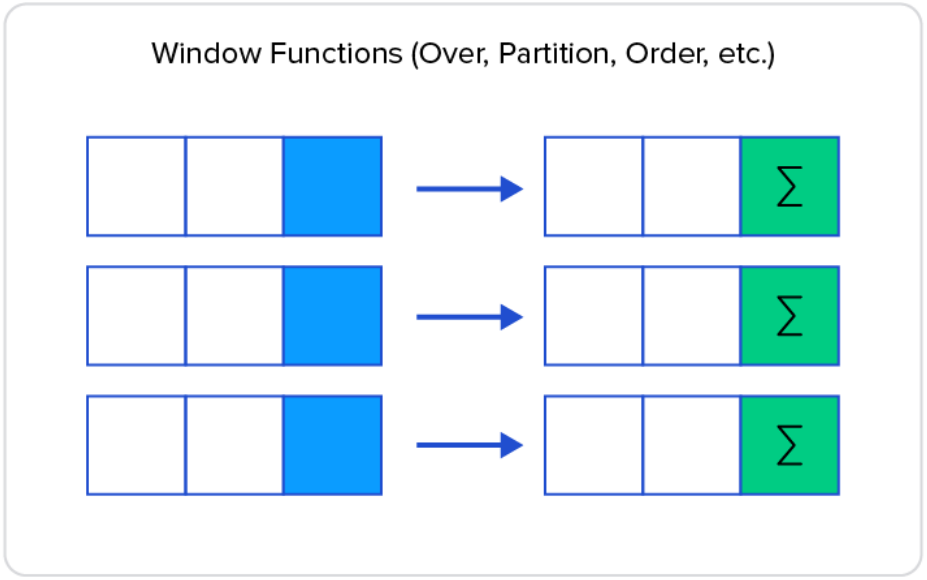
What is Window Functions?

Window functions are used to **perform a calculation on an aggregate value** based on a set of rows and return multiple rows for each group.

- The window word represents the group of rows on which the function will be operated.
- Performs a calculation in the same way that the aggregate functions would perform.
- Window functions work on a group of rows and return a total value for each row. As a result, each row retains its distinct identity.

What is Window Functions?





Syntax

```
window_function_name([ALL] expression)
OVER (
    [partition_defintion]
    [order_definition]
)
```

window_function: It indicates the name of your window function.

ALL (optional): count all values along with duplicates. We cannot use the DISTINCT keyword in window functions.

Expression: the column name for which we calculate an aggregated value.

Syntax

OVER: It specifies the window clauses for aggregate functions.

PARTITION BY: This clause divides the rows into partitions, and then a window function is operated on each partition.

ORDER BY: It is used to specify the order of rows within each partition.

When this clause is not defined, SQL Server will use the ORDER BY for the entire table.

Window Functions Types

Aggregate Window Functions

Operated on multiple rows. SUM(), MAX(), MIN(), AVG(), COUNT(), etc.

Ranking Window Functions

Ranks each row of a partition in a table.

RANK(), DENSE_RANK(), ROW_NUMBER(), NTILE(), etc.

Value Window Functions

Locally represented by a power series.

LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE(), etc.

Example – Sales Table

```
CREATE TABLE Sales(
    Employee VARCHAR(45) NOT NULL,
    Year INT NOT NULL,
    Country VARCHAR(45) NOT NULL,
    Product VARCHAR(45) NOT NULL,
    Amount DECIMAL(12,2) NOT NULL,
    PRIMARY KEY(Employee, Year)
);
```

Aggregration Window Functions

Example – Sum

```
SELECT Employee, Year, Country, Product, Amount, SUM(Amount)
OVER(PARTITION BY Country) as Total
FROM Sales
```

Result: aggregates the data **for each country** and displays the sum of **total sales** for each of them. It also inserts another column for the total sales as Total so that each row retains its identity.

Example – Sum

Employee	Year	Country	Product	Amount	Total
Or Peretz	2017	Israel	Computer	15000	45000
Or Peretz	2018	Israel	Computer	10000	45000
Or Peretz	2019	Israel	TV	20000	45000
Omer Doron	2018	USA	TV	20000	30000
Omer Doron	2019	USA	Mobile	10000	30000

Example – Avg

```
SELECT Employee, Year, Country, Product, Amount, AVG(Amount)
OVER(PARTITION BY Country, YEAR(Year)) as AvgSales
FROM Sales;
```

Result**:** produce the average sales for each country and each year. Here we have specified more than one average by specifying multiple fields in the partition list.

Example – Avg

Employee	Year	Country	Product	Amount	AvgSales
Or Peretz	2017	Israel	Computer	15000	15000
Or Peretz	2018	Israel	Computer	10000	15000
Or Peretz	2019	Israel	TV	20000	15000
Omer Doron	2018	USA	TV	20000	15000
Omer Doron	2019	USA	Mobile	10000	15000

Example – Count

The count function will return the total number of rows or records present in the table or group. The regular aggregate function uses the DISTINCT keyword not to count the duplicates rows. **But the window count function does not support this keyword**.

```
SELECT Employee, Year, Country, Product, Amount, COUNT(Product)
OVER(PARTITION BY Country) As TotalProduct
FROM Sales;
```

Example – Max

This function returns the maximum value for a specified group. When we have not defined the group, it will return the maximum value for the entire table.

```
SELECT Employee, Year, Country, Product, Amount, MAX(Product)
OVER(PARTITION BY Country) As TotalProduct
FROM Sales;
```

Ranking Window Functions

Ranking Window Functions

The RANKING function **ranks the values in a defined column** and categorizes them based on their rank. The following are the ranking functions supported in SQL Server:

RANK(), DENSE_RANK(), ROW_NUMBER(), and NTILE().

Data for examples:

FirstName	LastName	City
Luisa	Evans	Texas
Paul	Ward	Alaska
Peter	Bennett	California
Carlos	Patterson	New York
Rose	Huges	Florida
Marielia	Simmons	Texas
Antonio	Butler	New York
Diego	Cox	California

Example – Rank

It's used to **generate a unique rank** for each row in a table based on the specified value.

If this function gets the two records with the same value, it will assign the same rank to both records and skip the next ranking.

```
SELECT FirstName, LastName, City,
RANK () OVER (ORDER BY  City) AS RankNo
FROM table;
```

Example – Rank

FirstName	LastName	City	RankNo
Paul	Ward	Alaska	1
Peter	Bennett	California	2
Diego	Cox	California	2
Rose	Huges	Florida	4
Carlos	Patterson	New York	5
Antonio	Butler	New York	5
Luisa	Evans	Texas	7
Marielia	Simmons	Texas	7

Example – Dense Rank

It works the same as the RANK() function except that it does not skip any rank. It always assigns rank in consecutive order. It means that when two records are found equal, this function will assign the same rank to both records and the next rank being the next sequential number.

```
SELECT FirstName, LastName, City,
DENSE_RANK() OVER (ORDER BY city) AS RankNo
FROM table
```

Example – Dense Rank

FirstName	LastName	City	RankNo
Paul	Ward	Alaska	1
Peter	Bennett	California	2
Diego	Cox	California	2
Rose	Huges	Florida	3
Carlos	Patterson	New York	4
Antonio	Butler	New York	4
Luisa	Evans	Texas	5
Marielia	Simmons	Texas	5

Example – Ntile

This window function **distributes rows into a pre-defined number (N) of approximately equal groups**. It enables us to determine which percentile (or quartile, or other subdivision) a particular row belongs to.

```
SELECT FirstName, LastName, City,
NTILE(3) OVER ( ORDER BY city) AS RankNo
FROM table
```

Example – Ntile

FirstName	LastName	City	RankNo
Paul	Ward	Alaska	1
Peter	Bennett	California	1
Diego	Cox	California	1
Rose	Huges	Florida	2
Carlos	Patterson	New York	2
Antonio	Butler	New York	2
Luisa	Evans	Texas	3
Marielia	Simmons	Texas	3

Value Window Functions

Value Window Functions

The LEAD and LAG functions are used to **get the preceding and succeeding values** of specified rows from the current row within its partition.

```
CREATE TABLE Sales(
    Employee VARCHAR(45) NOT NULL,
    Year INT NOT NULL,
    Country VARCHAR(45) NOT NULL,
    Product VARCHAR(45) NOT NULL,
    Amount DECIMAL(12,2) NOT NULL,
    PRIMARY KEY(Employee, Year)
);
```

Example - Lead

```
SELECT Year, Product, Country, Amount,
LEAD(Amount,1) OVER (PARTITION BY Year ORDER BY Country) AS NextAmount
FROM Sales
```

Result: returns the amount and next amount detail of each employee.

It first split the result based on the **year** and then sorted each partition using the **country** column.

LEAD() - function on each partition to get the next sales detail.

Example - Lead

Year	Product	Country	Amount	NextAmount
2017	Computer	Canada	15000	10000
2017	Laptop	Israel	10000	20000
2017	TV	Israel	20000	NULL
2018	TV	Canada	20000	10000
2018	Mobile	USA	10000	NULL

Example - Lad

```
SELECT Year, Product, Country, Amount,
LAG(Amount, 1) OVER (PARTITION BY Year ORDER BY Country) AS PrevAmount
FROM Sales
```

Result: returns the amount and previous amount detail of each employee.

It first split the result based on the **year** and then sorted each partition using the **country** column.

Lad() - function on each partition to get the previous sales detail.

Example - Lad

Year	Product	Country	Amount	NextAmount
2017	Computer	Canada	15000	NULL
2017	Laptop	Israel	10000	15000
2017	TV	Israel	20000	10000
2018	TV	Canada	20000	NULL
2018	Mobile	USA	10000	20000

First and Last Values

These functions are used to find the first and last record in the table or a partition if the PARTITION BY clause is specified.

Note: these functions are mandatory to use the ORDER BY clause.

```
SELECT Year, Product, Country, Amount,
FIRST_VALUE(Amount) OVER(PARTITION BY Country ORDER BY Country) FirstAmount,
LAST_VALUE(Amount) OVER(PARTITION BY Country ORDER BY Country) LastAmount
FROM Sales
```

First and Last Values

Year	Product	Country	Amount	FirstAmount	LastAmount
2017	Computer	Canada	15000	15000	10000
2018	Laptop	Canada	10000	15000	10000
2017	TV	Israel	10000	10000	20000
2018	TV	Israel	15000	10000	20000
2019	Mobile	Israel	20000	10000	20000