Politecnico di Torino

DBMG

Advanced queries

SQL Language

SQL language: advanced queries

- Derived tables
- CTE
- Spatial queries
- JSON queries

Derived tables

- Define a temporary table that can be used for further computations
- A derived table
  - has the structure of a SELECT statement
  - is defined within a FROM clause
  - may be referenced as a normal table
- Derived tables allow
  - to calculate multiple levels of aggregation
  - an equivalent formulation of queries that require the use of correlation

Computing two-level aggregates (no.1)

- Find the maximum average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

Step 1: Find the average for each student

```
SELECT SId, AVG(Grade) AS StudentAVG
FROM  PASSED-EXAM
GROUP BY SId
```

DBMG

Computing two-level aggregates (no.1)

- Find the maximum average (achieved by a student)

STUDENT (<u>SId</u>, YearOfEnrolment)
PASSED-EXAM (<u>SId</u>, <u>CId</u>, Date, Grade)

Step 2: Find the maximum value of the average

```
SELECT MAX(StudentAVG)
FROM (SELECT SId, AVG(Grade) AS StudentAVG
      FROM  PASSED-EXAM
      GROUP BY SId) AS AVERAGES;
```

Derived table

Computing two-level aggregates (no.2)

- For each year of enrolment, find the highest average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

- 2-step solution
  - Find the average for each student
  - Group students by year of enrolment and calculate the maximum average

Computing two-level aggregates (no.2)

- For each year of enrolment, find the highest average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

- Step 1: Find the average for each student

```
SELECT SId, AVG(Grade) AS StudentAVG
FROM  PASSED-EXAM
GROUP BY SId
```

Computing two-level aggregates (no.2)

- For each year of enrolment, find the highest average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

- Step 2: Group students by year of enrollment and calculate the maximum average

```
SELECT ...
FROM STUDENT,
     (SELECT SId, AVG(Grade) AS StudentAVG
      FROM  PASSED-EXAM
      GROUP BY SId) AS AVERAGES
WHERE STUDENT.SId=AVERAGES.SId
....
```

Derived tables

Join condition

Computing two-level aggregates (no.2)

- For each year of enrolment, find the highest average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

- Step 2: Group students by year of enrollment and calculate the maximum average

```
SELECT .....
FROM STUDENT,
        (SELECT SId, AVG(Grade) AS StudentAVG
         FROM  PASSED-EXAM
         GROUP BY SId) AS AVERAGES
WHERE STUDENT.SId=AVERAGES.SId
GROUP BY YearOfEnrolment;
```

Computing two-level aggregates (no.2)

- For each year of enrolment, find the highest average (achieved by a student)

STUDENT (SId, YearOfEnrolment)
PASSED-EXAM (SId, CId, Date, Grade)

- Step 2: Group students by year of enrollment and calculate the maximum average

```
SELECT YearOfEnrolment, MAX(StudentAVG)
FROM STUDENT,
        (SELECT SId, AVG(Grade) AS StudentAVG
         FROM  PASSED-EXAM
         GROUP BY SId) AS AVERAGES
WHERE STUDENT.SId=AVERAGES.SId
GROUP BY YearOfEnrolment;
```

Correlation with derived tables

- For each product, find the ID of the supplier that provides the maximum quantity

P (<u>PId</u>, PName, Color, Size, Store)

S (<u>SId</u>, SName, #Employees, City)

SP (<u>SId</u>, <u>PId</u>, Qty)

- 2-step solution
  - Calculate the maximum quantity supplied for each product
  - Select suppliers that supply the maximum quantity, product by product

Correlation with derived tables

- For each product, find the ID of the supplier that provides the maximum quantity

P (PId, PName, Color, Size, Store)
S (SId, SName, #Employees, City)
SP (SId, PId, Qty)

- Step 1: Calculate the maximum quantity supplied for each product

```
SELECT PId, MAX(Qty) AS MQty
FROM SP
GROUP BY PId
```

DBMG

Correlation with derived tables

- For each product, find the ID of the supplier that provides the maximum quantity

P (<u>PId</u>, PName, Color, Size, Store)

S (<u>SId</u>, SName, #Employees, City)

SP (<u>SId</u>, <u>PId</u>, Qty)

- Step 2: Select suppliers that supply the maximum quantity, product by product

```
SELECT PId, SId
FROM SP,
      (SELECT PId, MAX(Qty) AS MQty
      FROM SP GROUP BY PId
      ) AS TMax
WHERE SP.PId = TMax.PId
AND   SP.Qty = TMax.MQty;
```

Derived table (the subquery aliased TMax)

Join condition (SP.PId = TMax.PId)

Correlation condition (SP.Qty = TMax.MQty)

Common Table Expression

- Defines a temporary table that can be used for further computation
- A CTE
  - has the structure of a SELECT
  - is defined by the WITH clause
  - can be referenced like a normal table
- A CTE can be used to
  - to calculate multiple levels of aggregation
  - provide an equivalent formulation of queries that require the use of correlation
- References
  - to CTE previously defined in the same WITH clause
  - recursive

CTE vs Derived tables

- CTE is preferred when
  - you must reference a derived table multiple times in a single query
  - you must perform the same calculation multiple times in multiple parts of the query
  - you want to increase the readability of complex queries

Syntax to define CTEs

```
WITH
cte_1 [(field_A, ...)] AS
(CTE query 1)
{, cte_X AS (CTE query X) }
SELECT field_A, field_B, ...
FROM cte_1
```

CTE Name

CTE query

Query

Computing two-level aggregates (no.1)

- Find the maximum average (achieved by a student)

STUDENT (<u>SId</u>, YearOfEnrolment)
PASSED-EXAM (<u>SId</u>, <u>CId</u>, Date, Grade)

- 2-step solution
  - find the average for each student
  - find the maximum value of the average

Computing two-level aggregates (no.1)

- Find the maximum average (achieved by a student)

STUDENT (<u>SId</u>, YearOfEnrolment)
PASSED-EXAM (<u>SId</u>, <u>CId</u>, Date, Grade)

```
WITH AVERAGES AS
        (SELECT SId, AVG(Grade) AS StudentAVG
         FROM  PASSED-EXAM
         GROUP BY SId)
SELECT MAX(StudentAVG)
FROM AVERAGES
```

Calculation of aggregates with different granularity

- Find all airlines where the average salary of all pilots of that airline is higher than the average of the salaries of all pilots in the database

PILOTS (<u>PID</u>, Name, Surname, Airline, Salary)

- 3-step solution:
  - find the average salary for each airline
  - find the average salary considering all pilots
  - find airlines with an average salary higher than the global average salary

Calculation of aggregates with different granularity

- Step 1: find the average salary for each airline

```
WITH AverageAirlineSalary AS
        (SELECT Airline, AVG(Salary) AS AvgAirlineSal
        FROM PILOTS
        GROUP BY Airline)
```

Calculation of aggregates with different granularity

- Step 2: find the average salary considering all pilots

```
WITH AverageAirlineSalary AS
        (SELECT Airline, AVG(Salary) AS AvgAirlineSal
        FROM PILOTS
        GROUP BY Airline),
AvgSalary AS
        (SELECT AVG(Salary) AS AvgSal
        FROM PILOTS )
```

Calculation of aggregates with different granularity

- Step 3: find airlines with an average salary higher than the global average salary

```
WITH AverageAirlineSalary AS
        (SELECT Airline, AVG(Salary) AS AvgAirlineSal
        FROM PILOTS
        GROUP BY Airline)
AvgSalary AS
        (SELECT AVG(SAlary) AS AvgSal
        FROM PILOTS )
SELECT Airline
FROM AverageAirlineSalary, AvgSalary
WHERE AverageAirlineSalary. AvgAirlineSal >
TotalSalary.TotSal
```

Referenced CTE

- Considering the average distances traveled for each city, calculate the maximum distance traveled for each region

CITY (<u>CodeC</u>, Name, Region)
DRIVER (<u>CodeD</u>, Name, Surname, CodeV, CodeC)
DAILY_RUN (<u>Date</u>, <u>CodeD</u>, Amount, Distance)

- 3-step solution:
  - calculate the distance traveled for each city by each driver
  - calculate the average distance for each city
  - calculate the maximum distance per region

Referenced CTE

- Step 1: calculate the distance traveled for each city by each driver

```
WITH totDistanceDrive AS
        ( SELECT SUM(Distance) AS TotalDistance, DR.CodeD, DR.CodeC, C.Name, Region
        FROM DAILY_RUN DR, CITY C
        WHERE DR.CodeD =  D.CodeD AND D.CodeC = C.CodeC
        GROUP BY DR.CodeA, DR.CodeC, C.Name, Region )
```

Referenced CTE

- Step 2: calculate the average distance for each city

```
WITH totDistanceDrive AS
        ( SELECT SUM(Distance) AS TotalDistance, DR.CodeD, DR.CodeC, C.Name, Region
        FROM DAILY_RUN DR, CITY C
        WHERE DR.CodeD =  D.CodeD AND D.CodeC = C.CodeC
        GROUP BY DR.CodeA, DR.CodeC, C.Name, Region )
averageDistance AS
        (SELECT AVG(TotalDistance) AS avgDist, CodeC, Region
        FROM totDistanceDrive
        GROUP BY CodeC, Region )
```

Referenced CTE

- Step 3: calculate the maximum average distance per region

```
WITH totDistanceDrive AS
        ( SELECT SUM(Distance) AS TotalDistance, DR.CodeA, DR.CodeC, Name, Region
        FROM DAILY_RUN DR, CITY C
        WHERE DR.CodeA, DR.CodeC,
        GROUP BY DR.CodeA, DR.CodeC, Name, Region ),
averageDistance AS
        ( SELECT AVG(TotalDistance) AS avgDist, CodC, Region
        FROM totDistanceDrive
        GROUP BY CodeC, Region )
SELECT MAX(avgDist), Region
FROM averageDistance
GROUP BY Region
```

Recursive CTE syntax

```
WITH RECURSIVE
cte_1 AS
(CTE query 1
UNION ALL
CTE query 2
)
SELECT *
FROM cte_1
```

- *Name of CTE* → cte_1 AS
- *Initial query* → (CTE query 1
- *Recursive query* → CTE query 2

Recursive CTEs

- For each employee, find the boss and level in the hierarchy

EMPLOYEES (EID, Name, Surname, BossID*)

EID	Name	Surname	BossId*
1	Domenic	Leaver	5
2	Cleveland	Hewins	1
3	Kakalina	Atherton	7
4	Roxanna	Fairlie	NULL
5	Hermie	Comsty	4
6	Pooh	Goss	7
7	Faulkner	Challiss	5

Recursive CTEs

```
WITH RECURSIVE hierarchy AS (
  SELECT   EID, Name, Surname, BossID, 0 AS level
  FROM EMPLOYEES
  WHERE  BossID IS NULL

  UNION ALL

  SELECT   E.EID, E.Nome, E.Cognome, E.BossID, level +1
  FROM EMPLOYEES E, hierarchy H
  WHERE E.BossID = H.EID
)

SELECT   G.Name, G.Surname, E. Name AS BossName, E. Surname AS BossSurname, level
FROM  hierarchy G LEFT JOIN EMPLOYEES E ON G.BossID= E.EID
ORDER BY level;
```

Spatial queries

- Spatial data can be represented by different geometries
  - Point
  - Polygon
  - Lines,
  - etc.
- MySQL provides functions to:
  - create geometries in various formats (WKT, WKB, internal)
  - convert geometries between different formats
  - access the qualitative or quantitative properties of a geometry
  - describe the relationships between two geometries
  - create new geometries from existing ones

Creating Geometry (MySQL)

- Point(x, y)
  - constructs a point using its coordinates
- LineString(pt [, pt] ...)
  - constructs a line using the points provided (at least 2)
- Polygon(ls [, ls] ...)
  - constructs a polygon from a series of lines

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

Geometry Properties (MySQL)

- ST_Dimension(g)
  - Returns the intrinsic dimension of the geometric value g
  - Size can be -1, 0, 1 or 2
- ST_Envelope(g)
  - Returns the minimum bounding rectangle (MBR) for the geometric value g
  - The result is returned as a polygon value defined by the corner points of the bounding rectangle
- ST_GeometryType(g)
  - Returns a string indicating the name of the geometry type of which geometry instance G is a member

Geometry Properties (MySQL)

- ST_X(p)
  - Returns the value of the X-coordinate of the Point p
- ST_Y(p)
  - Returns the Y-coordinate value of the Point p
- ST_Length(ls)
  - Returns the length of a line
- ST_Area(poly)
  - Returns the area of a polygon
- ST_Centroid(poly)
  - Returns the centroid of a polygon

Geometry Relationships (MySQL)

- ST_Difference(g1, g2)
  - Returns a geometry that represents the difference in the point set of geometries G1 and G2
- ST_Intersects(g1, g2)
  - Returns 1 or 0 to indicate whether G1 spatially intersects G2
- ST_Distance_Sphere(g1, g2 [, radius])
  - Returns the minimum spherical distance between two points and/or more points on a sphere, in meters
  - The optional **radius** argument must be indicated in meters. If omitted, the default radius is 6,370,986 meters

```
SELECT ST_Distance_Sphere(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('POINT(180 0)'));
```

RESULT
20015042.813723423

DBMG

JSON Query

- JSON, short for JavaScript Object Notation, is a format for exchanging data in client-server applications
- JSON data functions depend on the DBMS used
- JSON data functions used for
  - create data in JSON format
  - search within a JSON based on the path provided
  - edit JSON fields

JSON file example

```
{
 name: "Agritourism Mario Bros",
 address:{
       street: "Via Idraulici",
       number: 1
       city: "Funghetti",
 },
 Reviews:[
       {text: "Adventurous experience",
        timestamp: "2023-04-05T16:19:00",
        stars: 5}
 ],
 nReviews: 1,
 tags: ["agritourism", "nature"]
}
```

Key (name:)

Value ("Agritourism Mario Bros")

Embedded JSON (address)

Array (tags)

Create JSON (MySQL)

- JSON_ARRAY(target, candidate[, path])
  - evaluates a list of values (possibly empty) and returns a JSON array containing those values

```
SELECT JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME()) AS RESULT;
```

RESULT
[1, "abc", null, true, "11:30:24.000000"]

- JSON_OBJECT([key, val[, key, val] ...])
  - evaluates a (possibly empty) list of key-value pairs and returns a JSON object containing those pairs

```
SELECT JSON_OBJECT('id', 87, 'name', 'carrot') AS RESULT;
```

RESULT
{"id": 87, "name": "carrot"}

Search within JSON (MySQL)

- JSON_CONTAINS(target, candidate[, path])
  - returns 1 or 0
    - if a JSON ***candidate*** document is contained in the JSON ***target*** document
    - if the ***candidate*** is in a specific ***path*** within the ***target*** document
  - returns NULL
    - if any of the arguments is NULL
    - If the ***path*** does not identify a section of the ***target*** document
  - Path notation:
    - $: Document root
    - dot notation to specify the path (eg. $.a)
    - [i]: to access the i-th element of an array
    - wildcard * or ** ($.*)

```sql
SELECT JSON_CONTAINS('{"a": 1, "b": 2, "c": {"d": 4}}', '1', '$.a') AS RESULT;
```

RESULT
1

Search within JSON (MySQL)

- JSON_EXTRACT(json_doc, path[, path])
  - returns data from a JSON document in the paths provided as parameters
  - returns NULL if
    - any argument is NULL
    - no path locates a value in the document
- Alternative:
  - Use the operator ->

```
SELECT c, JSON_EXTRACT(c, "$.id")
FROM jemp
WHERE JSON_EXTRACT(c, "$.id") > 1
ORDER BY JSON_EXTRACT(c, "$.name");
```

```
SELECT c, c->"$.id"
FROM jemp
WHERE c->"$.id" > 1
ORDER BY c->"$.name";
```

c	c->"$.id"
{"id": "3", "name": "Barney"}	"3"
{"id": "4", "name": "Betty"}	"4"
{"id": "2", "name": "Wilma"}	"2"

DBMG

Edit JSON (MySQL)

- JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...)
  - appends the values to the end of the indicated arrays and returns the result

```
SELECT JSON_ARRAY_APPEND('["a", ["b", "c"], "d"]', '$[1]', 1) AS RESULT;
```

RESULT
["a", ["b", "c", 1], "d"]

- JSON_INSERT(json_doc, path, val[, path, val] ...)
  - inserts values into the JSON file and returns the result

```
SELECT JSON_INSERT('{ "a": 1, "b": [2, 3]}', '$.a', 10, '$.c', '[true, false]') AS RESULT;
```

RESULT
{"a": 1, "b": [2, 3], "c": "[true, false]"}

Edit JSON (MySQL)

- JSON_SET(json_doc, path, val[, path, val] ...)
  - inserts or updates JSON document values and returns the result

```
SELECT JSON_SET('{ "a": 1, "b": [2, 3]}' '$.a', 10, '$.c', '[true, false]') AS RESULT;
```

RESULT
{"a": 10, "b": [2, 3], "c": "[true, false]"}

- JSON_REMOVE(json_doc, path, [, path] ...)
  - removes the path in the JSON document and returns the result

```
SELECT JSON_REMOVE('["a", ["b", "c"], "d"]', '$[1]') AS RESULT;
```

RESULT
["a", "d"]