Politecnico di Torino

SQL language: basics

0

SQL language: basics

- SQL Language
- Language Instruction
- Sample notation and database
- SELECT Statement
- Aggregate Functions
- Operator GROUP BY

1

1

The SQL language

- A language for managing relational databases
  - Structured Query Language
- SQL provides commands to
  - define the schema of a relational database
  - read and write data
  - define the schema of derived tables
  - define user access privileges
  - manage transactions
- The SQL language may be used in two ways
  - interactive
  - compiled
    - a host language encapsulates the SQL commands
    - SQL commands can be distinguished from the host language commands by means of appropriate syntactic mechanisms

2

2

The SQL language

- SQL is a *set-level* language
  - operators are applied to relations (tables)
  - the result is always a relation (table)
- SQL is a *declarative* language
  - it describes *what to do* and not how to do it
  - it has a higher level of abstraction compared to traditional programming languages

3

3

SQL instructions

The SQL language

4

4

The SQL language

- Can be divided into
  - DML (Data Manipulation Language)
    - language for querying and updating the data
  - DDL (Data Definition Language)
    - language for defining the database structure

5

5

Data Manipulation Language

- To query a database in order to extract data of interest
  - SELECT
- To modify a database instance
  - INSERT: insertion of new information into a table
  - UPDATE: update of the information in the database
  - DELETE: cancellazione di dati obsoleti

Data Definition Language

- To define a database schema
  - creation, modification and deletion of tables: CREATE, ALTER, DROP TABLE
- To define derived tables
  - creation, modification and deletion of tables whose content is obtained from other database tables: CREATE, ALTER, DROP VIEW
- To define complementary data structures for efficiently retrieving the data
  - creation and deletion of indices: CREATE, DROP INDEX
- To define user access privileges
  - grant and revocation of privileges on resources: GRANT, REVOKE
- To define transactions
  - termination of a transaction: COMMIT, ROLLBACK

Notation and example database

The SQL Language

Syntax of SQL commands

- Notation
  - language keywords
    - upper case
  - variable terms
- Grammar
  - angle brackets < >
    - to isolate a syntactic term
  - square brackets [ ]
    - the enclosed term is optional
  - braces { }
    - the enclosed term may not appear or may be repeated an arbitrary number of items
  - vertical bar |
    - a term must be chosen among the options separated by the vertical bars

Example database: Supply-Product

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Foreign key

Foreign key

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200

Example database: Supply-Product

- Supplier and part DB
  - table P describes the available products
    - primary key: PId
  - table S describes the suppliers
    - primary key: SId
  - table SP describes supplies, by relating each product to the suppliers that provide it
    - primary key: (SId, PId)
    - PId: Foreign key (SP) REFERENCES PId(P)
    - Sid: Foreign key (SP) REFERENCES SId(S)

The SELECT statement: basics

The SQL language

12

SELECT

SELECT [DISTINCT] *ListOfAttributesToDisplay*
FROM *ListOfTablesToUse*
[WHERE *TupleConditions*]
[GROUP BY *ListOfGroupingAttributes*]
[HAVING *AggregateConditions*]
[ORDER BY *ListOfOrderingAttributes*]

13

Basic SELECT(n.1)

- Find the codes and the number of employees of the suppliers based in Paris

```
SELECT SId, #Employees
FROM   S
WHERE City='Paris';
```

R
||
$\pi_{\text{SId, \#Empolyees}}$
|
$\sigma_{\text{City='Paris'}}$
|
S

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

SId	#Employees
S2	10
S3	30

14

Basic SELECT(n.2)

- Find the codes of all products in the database

```
SELECT PId
FROM P;
```

R
||
π_{PId}
|
P

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

R

PId
P1
P2
P3
P4
P5
P6

15

Basic SELECT(n.3)

- Find the codes of the products supplied by at least one supplier

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT PId
FROM SP;
```

R

PId
P1
P2
P3
P4
P5
P6
P1
P2
P2
P3
P4
P5

16

Basic SELECT(n.3)

- Find the codes of the products supplied by at least one supplier

```
SELECT PId
FROM SP;
```

R
||
π_{PId}
|
SP

- It does not eliminate duplicates

17

Elimination of duplicates: DISTINCT

- DISTINCT keyword allows the elimination of duplicates
- Find the codes of the *distinct* products supplied by at least one supplier

```
SELECT DISTINCT PId
FROM SP;
```

SP

SId	PID	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

R

PId
P1
P2
P3
P4
P5
P6

18

18

Selection of all information

- Find all information related to products

```
SELECT PId, PName, Color, Size, Store
FROM P;
```

or

```
SELECT *
FROM P;
```

R

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

19

19

Selection with an expression

- Find the codes of the products and the sizes expressed with the US standard

```
SELECT PId, Size-14 [AS USSize]
FROM P;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

R

PId	USSize
P1	26
P2	34
P3	34
P4	30
P5	26
P6	38

- Definition of a new *temporary* column for the computed expression
  - the name of the temporary column may be defined by means of the AS keyword

20

20

The WHERE clause

- It allows expressing selection conditions applied to each tuple individually
- A Boolean expression composed by one or more predicates
- Simple predicates
  - comparison between attributes and constants
  - text search
  - NULL values

21

21

The WHERE clause (n.1)

- Find the codes of the suppliers based in Paris

```
SELECT SId
FROM S
WHERE City='Paris';
```

F

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

SId
S2
S3

22

The WHERE clause (no.2)

- Find the codes and the number of employees of the suppliers that are not based in Paris

```
SELECT SId, #Employees
FROM S
WHERE City<>'Paris';
```

F

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

SId	#Employees
S1	20
S4	20
S5	30

23

Boolean expressions (no.1)

- Find the codes of the suppliers based in Paris that have more than 20 employees

```
SELECT SId
FROM S
WHERE City='Paris' AND #Employees>20;
```

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

Sid
S3

DBG 24

24

Boolean expressions (no.2)

- Find the codes and the number of employees of the suppliers based in Paris or London

```
SELECT SId, #Employees
FROM S
WHERE City='Paris' OR City='London';
```

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

SId	#Employees
F1	20
F2	10
F3	30
F4	20

DBG 25

25

Boolean expressions (no.3)

- Find the codes and the number of employees of the suppliers based in Paris and in London
  - the query may not be satisfied
    - each supplier has only one city

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

DBG

26

Text search

- LIKE operator

  AttributeName LIKE *CharacterString*

  - the _ character represents a single arbitrary character (non-empty)
  - the % character represents an arbitrary sequence of characters (possibly empty)

27

27

Text search (no.1)

- Find the codes and the names of the products whose name begins with the letter B

```
SELECT PId, PName
FROM P
WHERE PName LIKE 'B%';
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

R

PId	PName
P3	Blouse
P4	Blouse

DBG 28

28

Text search (no.2)

- The Address attribute contains the string 'London'

  Address LIKE '%London%'

- The supplier identification number is 3 and
  - it is preceded by a single unknown character
  - it is exactly 2 characters long

  SId LIKE '_3'

- The Store attribute does not have an 'e' in the second position

  Store NOT LIKE '_e%'

DBG 29

29

Searching for NULL values

- IS special operator
  AttributeName IS [NOT] NULL
- With NULL values, any comparison predicate is false

30

30

Managing NULL values

- Find the codes and the names of products with a size greater than 44

```
SELECT PId, PName
FROM P
WHERE Size>44;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	NULL	Paris
P6	Shorts	Red	42	London

R

PId	PName
P2	Jeans
P3	Blouse

- The tuples with NULL size are not selected: the predicate Size>44 evaluates to false
- With NULL values, any comparison predicate is false

31

Searching for NULL values (no.1)

- Find the codes and the names of the products whose size is unknown

```
SELECT PId, PName
FROM P
WHERE Size IS NULL;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	NULL	Paris
P6	Shorts	Red	42	London

R

PId	PName
P5	Skirt

32

32

Searching for NULL values(n.2)

- Find the codes and the names of products with a size greater than 44, or that may have a size greater than 44

```
SELECT PId, PName
FROM P
WHERE Size>44 OR Size IS NULL;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	NULL	Paris
P6	Shorts	Red	42	London

R

PId	Pname
P2	Jeans
P3	Blouse
P5	Skirt

33

Result ordering

- ORDER BY clause
  ORDER BY *AttributeName* [ASC | DESC]
  {, *AttributeName* [ASC | DESC]}
  - the default ordering is ascending
    - if DESC is not specified
  - the ordering attributes must appear in the SELECT clause
    - even implicitly (as in SELECT *)

34

34

Result ordering (no.1)

- Find the codes of the products and their sizes, ordering the result by decreasing size

```
SELECT PId, Size
FROM P
ORDER BY Size DESC;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

R

PId	Size
P2	48
P3	48
P4	44
P6	42
P1	40
P5	40

35

Result ordering (no.2)

- Find all information related to the products, ordering the result by increasing name and decreasing size

```
SELECT PId, PName, Color, Size, Store
FROM P
ORDER BY PName, Size DESC;
```

```
SELECT *
FROM P
ORDER BY PName, Size DESC;
```

R

PId	PName	Color	Size	Store
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P2	Jeans	Green	48	Paris
P1	Jumper	Red	40	London
P6	Shorts	Red	42	London
P5	Skirt	Blue	40	Paris

36

36

Result ordering (no.3)

- Find the codes of the products and the sizes expressed with the US standard, ordering the result by increasing size

```
SELECT PId, Size-14 AS USSize
FROM P
ORDER BY USSize;
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

R

PId	USSize
P5	26
P1	28
P6	28
P4	30
P2	34
P3	34

37

37

Join

- Defined by the FROM and WHERE clauses
- The result and efficiency of the query
  - are independent of the order of the tables in the FROM clause
  - are independent of the predicate order in the WHERE clause
  - the optimal execution order is selected by the DBMS (optimizer module)
- FROM clause with N Tables
  - at least N-1 join conditions in the WHERE clause

38

38

Join (n.1)

- Find the names of the suppliers that provide product P2

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SId	PID	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200

39

39

Cartesian product

- Find the names of the suppliers that provide product P2

```
SELECT SName
FROM S, SP ;
```

40

Cartesian product

S.SId	S.SName	S.#Empl	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S1	Smith	20	London	S2	P1	300
...	...	...	...	...	...	...
S2	Jones	10	Paris	S1	P1	300
...	...	...	...	...	...	...
S2	Jones	10	Paris	S2	P1	300
...	...	...	...	...	...	...

41

41

Join (n.1)

=

S.SId	S.SName	S.#Empl	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S1	Smith	20	London	S2	P1	300
...	...	...	...	...	...	...
S2	Jones	10	Paris	S1	P1	300
...	...	...	...	...	...	...
S2	Jones	10	Paris	S2	P1	300
...	...	...	...	...	...	...

42

Join (n.1)

=

S.SId	S.SName	S.#Empl	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S2	Jones	10	Paris	S2	P1	300
S2	Jones	10	Paris	S2	P2	400
S3	Blake	30	Paris	S3	P2	200
S4	Clark	20	London	S4	P3	200
S4	Clark	20	London	S4	P4	300
S4	Clark	20	London	S4	P5	400

43

Join (n.1)

- Find the names of the suppliers that provide product P2

SELECT SName
FROM S, SP
WHERE S.SId=SP.SId;

Join condition

TableName.AttributeName

44

Join (n.1)

- Find the names of the suppliers that provide product P2

SELECT SName
FROM S, SP
WHERE S.SId=SP.SId AND PId='P2';

Join condition

TableName.AttributeName

45

Join (n.1)

SP.PId='P2'

=

S.SId	S.SName	S.#Empl	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S2	Jones	10	Paris	S2	P1	300
S2	Jones	10	Paris	S2	P2	400
S3	Blake	30	Paris	S3	P2	200
S4	Clark	20	London	S4	P3	200
S4	Clark	20	London	S4	P4	300
S4	Clark	20	London	S4	P5	400

46

Join (n.1)

S.Sid	S.SName	S.#Empl	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P2	200
S2	Jones	10	Paris	S2	P2	400
S3	Blake	30	Paris	S3	P2	200

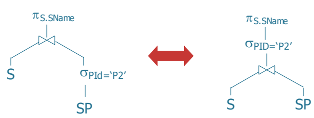
R

SName
Smith
Jones
Blake

47

Join (n.1)

- Find the names of the suppliers that provide product P2
  - in relational algebra



DBG

48

Join (n.1)

- Find the names of the suppliers that provide product P2
  - in relational algebra

```
SELECT SName
FROM S, SP
WHERE S.SId=SP.SId
      AND PId='P2';
```

⟺

```
SELECT SName
FROM S,SP
WHERE PId='P2' AND
      S.SId=SP.SId;
```

- The result and efficiency are independent
  - from the order of the predicates in the WHERE clause
  - from the order of the tables in the FROM clause

DBG

49

SQL Declarability

- In relational algebra (procedural language) we define the order in which the operators are applied
- In SQL (declarative language) the best order is chosen by the optimizer independently
  - from the order of the conditions in the WHERE clause
  - from the order of the tables in the FROM clause

DBG

50

Join (n.2)

- Find the name of suppliers who provide at least one red product

```
SELECT SName
FROM S, SP, P
WHERE  S.SId=SP.SId AND P.PId=SP.PId
       AND Color='Red';
```

- FROM Clause with N Tables
  - at least N-1 join conditions in the WHERE clause

DBG

51

Join (n.2)

- Find the pairs of supplier codes such that both suppliers are based in the same city

```
SELECT SX.SId, SY.SId
FROM S AS SX, S AS SY
WHERE  SX.City=SY.City;
```

S AS SX

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

S AS SY

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

DBG 52

52

Join (n.2)

- Find the pairs of supplier codes such that both suppliers are based in the same city

```
SELECT SX.SId, SY.SId
FROM S AS SX, S AS SY
WHERE  SX.City=SY.City;
```

R

SX.SId	SY.SId
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

- The result includes
  - pairs of identical values
  - permutations of the same pairs of values

DBG 53

53

Join (n.2)

- Find the pairs of supplier codes such that both suppliers are based in the same city

```
SELECT SX.SId, SY.SId
FROM S AS SX, S AS SY
WHERE  SX.City=SY.City AND
          SX.SId <> SY.SId;
```

R

SX.SId	SY.SId
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
S3	S2
~~S3~~	~~S3~~
S4	S1
~~S4~~	~~S4~~
~~S5~~	~~S5~~

- It removes pairs of identical values

54

Join (n.2)

- Find the pairs of supplier codes such that both suppliers are based in the same city

```
SELECT SX.SId, SY.SId
FROM S AS SX, S AS SY
WHERE  SX.City=SY.City AND
          SX.SId < SY.SId;
```

R

SX.SId	SY.SId
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
~~S3~~	~~S2~~
~~S3~~	~~S3~~
~~S4~~	~~S1~~
~~S4~~	~~S4~~
~~S5~~	~~S5~~

R

SX.Sid	SY.Sid
S1	S4
S2	S3

- It eliminates the permutations of the same pairs of values

55

Join: alternative syntax

- Different types of join may be specified
  - outer join
- It allows differentiating between
  - join conditions and
  - tuple selection conditions

SELECT [DISTINCT] *Attributes*
FROM *Table JoinType* JOIN *Table* ON
JoinCondition
[WHERE *TupleConditions*];

JoinType = < INNER | [FULL | LEFT | RIGHT] OUTER >

56

INNER join

- Find the names of the suppliers that supply at least one red product

```
SELECT SName
FROM P INNER JOIN SP ON P.PId=SP.PId
       INNER JOIN S ON S.SId=SP.SId
WHERE P.Color='Red';
```

57

OUTER join

- Find the codes and the names of the suppliers together with the codes of the products they provide, also including the suppliers that are not supplying any product

```
SELECT S.SId, SName, PId
FROM S LEFT OUTER JOIN SP ON
       S.SId=SP.SId;
```

S.Sid	S.SName	SP.Sid
S1	Smith	P1
S1	Smith	P2
S1	Smith	P3
S1	Smith	P4
S1	Smith	P5
S1	Smith	P6
S2	Jones	P1
S2	Jones	P2
S3	Blake	P2
S4	Clark	P3
S4	Clark	P4
S4	Clark	P5
S5	*Adams*	*NULL*

58

Aggregate Functions

Introduction to SQL

59

Aggregate function

- It operates on a set of values
- It produces a single (aggregate) value as a result
- It is specified in the SELECT clause
  - non-aggregate attributes may not be specified at the same time
  - multiple aggregate functions may be specified simultaneously
- Aggregate functions are only evaluated once all predicates in the WHERE clause have been applied

DBMG 60

60

Aggregate functions

- COUNT: count of elements in a given attribute
- SUM: sum of values for a given attribute
- AVG: average of values for a given attribute
- MAX: maximum value of a given attribute
- MIN: minimum value of a given attribute

DBMG 61

61

COUNT

- Counts the number of elements in a set
  - rows in a table
  - (possibly distinct) values for one or more attributes

COUNT (<*| [DISTINCT | ALL] ListOfAttributes >)}

- If the function argument is preceded by DISTINCT, it counts the number of distinct values of the argument

62

62

The COUNT function (n.1)

- Find the number of suppliers

```
SELECT COUNT(*)
FROM S;
```

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

→

R

5

DBMG 63

63

The COUNT function (n.2)

- Find the number of suppliers that supply at least one product

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT COUNT(*)
FROM SP;
```

→

R

12

- It counts the number of supplied products, not the suppliers

DBMG 64

64

The COUNT function (n.2)

- Find the number of suppliers that supply at least one product

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT COUNT(SId)
FROM SP;
```

→

R

12

- It still counts the number of supplied products, not the suppliers

DBMG 65

65

The COUNT function (n.2)

- Find the number of suppliers that supply at least one product

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT COUNT(DISTINCT SId)
FROM SP;
```

R

4

- It counts the number of distinct suppliers

66

66

Aggregate functions and WHERE

- Aggregate functions are only evaluated once all predicates in the WHERE clause have been applied

67

67

Aggregate functions and WHERE

- Find the number of suppliers providing product P2

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT COUNT(*)
FROM SP
WHERE PId='P2';
```

SId	PId	Qty
S1	P2	200
S2	P2	400
S3	P2	200

R

3

- Aggregate functions are only evaluated once all predicates in the WHERE clause have been applied

68

68

SUM, MAX, MIN, AVG

- SUM, MAX, MIN and AVG
  - they allow an attribute or an expression as argument
- SUM and AVG
  - they only allow numeric type or time interval attributes
- MAX and MIN
  - they require an expression that can be ordered
    - may also be applied to character strings and time instants

69

69

The SUM function

- Find the overall quantity of supplied pieces for product P2

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT SUM(Qty)
FROM SP
WHERE PId='P2';
```

SId	PId	Qty
S1	P2	200
S2	P2	400
S3	P2	200

R

800

70

70

The GROUP BY operator

Introduction to SQL

71

71

GROUP BY

- Grouping clause

  GROUP BY ListOfGroupingAttributes

  - The order of grouping attributes is irrelevant
- In the SELECT statement *only*
  - attributes specified in the GROUP BY clause
  - aggregate functions

  are allowed to appear
- Attributes that are unambiguously determined by other attributes already present in the GROUP BY clause may be added *without altering the result*

72

72

Grouping

- *For each product*, find the overall quantity of supplied pieces

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

SP

SId	PId	Qty
S1	P1	300
S2	P1	300
S1	P2	200
S2	P2	400
S3	P2	200
S1	P3	400
S4	P3	200
S1	P4	200
S4	P4	300
S1	P5	100
S4	P5	400
S1	P6	100

R

PId	
P1	600
P2	800
P3	600
P4	500
P5	500
P6	100

73

73

Grouping

- *For each product*, find the overall quantity of supplied pieces

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

SP

SId	PId	Qty
S1	P1	300
S2	P1	300
S1	P2	200
S2	P2	400
S3	P2	200
S1	P3	400
S4	P3	200
S1	P4	200
S4	P4	300
S1	P5	100
S4	P5	400
S1	P6	100

R

PId	
P1	600
P2	800
P3	600
P4	500
P5	500
P6	100

```
SELECT PId, SUM(Qty)
FROM SP
GROUP BY PId;
```

74

74

GROUP BY and WHERE

- For each product, find the overall quantity of pieces supplied by suppliers based in Paris

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

75

75

GROUP BY and WHERE

- For each product, find the overall quantity of pieces supplied by suppliers based in Paris

```
SELECT   …
FROM SP, S
WHERE SP.SId=S.SId AND City='Paris'
…
```

76

76

GROUP BY and WHERE

- For each product, find the overall quantity of pieces supplied by suppliers based in Paris

S.SId	S.SName	S.#Employees	S.City	SP.SId	SP.PId	SP.Qty
S1	Smith	20	London	S1	P1	300
S1	Smith	20	London	S1	P2	200
S1	Smith	20	London	S1	P3	400
S1	Smith	20	London	S1	P4	200
S1	Smith	20	London	S1	P5	100
S1	Smith	20	London	S1	P6	100
S2	*Jones*	*10*	*Paris*	*S2*	*P1*	*300*
S2	*Jones*	*10*	*Paris*	*S2*	*P2*	*400*
S3	*Blake*	*30*	*Paris*	*S3*	*P2*	*200*
S4	Clark	20	London	S4	P3	200
S4	Clark	20	London	S4	P4	300
S4	Clark	20	London	S4	P5	400

77

77

GROUP BY and WHERE

- For each product, find the overall quantity of pieces supplied by suppliers based in Paris

```
SELECT PId, SUM(Qty)
FROM SP, F
WHERE SP.SId=S.SId AND City='Paris'
GROUP BY PId;
```

- Products that are not supplied by any supplier are not included in the result

DBG 78

78

GROUP BY and WHERE

- For each product, find the overall quantity of pieces supplied by suppliers based in Paris

SP.PId	SP.Qty
P1	300
P2	400
P2	200

R

SP.PId	
P1	300
P2	600

DBG 79

79

GROUP BY and SELECT

- For each product, find the code, the name and the overall supplied quantity

```
SELECT P.PId, PName, SUM(Qty)
FROM P, SP
WHERE S.PId=SP.PId
GROUP BY P.PId, PName
```

- attributes that are unambiguously determined by other attributes already present in the GROUP BY clause may be added *without altering the result*

DBG 80

80

Group selection condition: HAVING

- You cannot use the WHERE clause to define selection conditions on groups
- Selection condition on groups expressed in HAVING clause:

  HAVING Group Conditions

  - it is possible to specify conditions only on aggregated functions

81

81

Group selection condition(n.1)

- Find the overall quantity of supplied pieces for the products for which at least 600 pieces are supplied *overall*

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

SP

SId	PId	Qty
S1	P1	300
S2	P1	300
S1	P2	200
S2	P2	400
S3	P2	200
S1	P3	400
S4	P3	200
S1	P4	200
S4	P4	300
S1	P5	100
S4	P5	400
S1	P6	100

R

PId	
P1	600
P2	800
P3	600

DBG 82

82

Group selection condition (n.1)

- Find the overall quantity of supplied pieces for the products for which at least 600 pieces are supplied *overall*

```
SELECT PId, SUM(Qty)
FROM SP
GROUP BY PId
HAVING SUM(Qty)>=600;
```

- The HAVING clause allows the specification of conditions on the aggregate functions

DBG 83

83

Group selection condition (n.2)

- Find the codes of the red products supplied by more than one supplier

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

DBMG 84

84

Group selection condition(n.2)

- Find the codes of the red products supplied by more than one supplier

```
SELECT SP.PId
FROM SP, P
WHERE SP.PId=P.PId AND Color='Red'
GROUP BY SP.PId
HAVING COUNT(*)>1;
```

DBMG 85

85

Group selection condition (n.2)

- Find the codes of the red products supplied by more than one supplier

S.Sid	S.PId	S.Qty	P.PId	P.Pname	P.Colore	P.Size	P.Store
S1	P1	300	P1	Jumper	Red	40	London
S2	P1	300	P1	Jumper	Red	40	London
S1	P6	100	P6	Shorts	Red	42	London

R

PId
P1

DBMG 86

86

Politecnico di Torino

DBMG

Nested queries

SQL language: basics

1

Nested queries

- Introduction
- The IN operator
- The NOT IN operator
- The tuple constructor
- The EXISTS operator
- The NOT EXISTS operator
- Correlation among queries
- The division operation

DBMG

2

Politecnico di Torino

DBMG

Introduction

Nested queries

3

Introduction

- A nested query is a SELECT statement contained within another query
  - query nesting allows decomposing a complex problem into simpler subproblems
- SELECT statements may be introduced
  - within a predicate in the WHERE clause
  - within a predicate in the HAVING clause
  - in the FROM clause

DBMG

4

Example database: Supply-Product

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Foreign key

Foreign key

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

DBMG

5

5

Nested queries (no.1)

- Find the codes of the suppliers that are based in the same city as S1
- By using a formulation with nested queries, the problem may be decomposed into two subproblems
  - city of supplier S1
  - codes of the suppliers based in the same city

DBMG

6

Nested queries (no.1)

- Find the codes of the suppliers that are based in the same city as S1

IDs of the suppliers based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```

City of supplier S1

- The '=' operator may be used only if it is known in advance that the inner SELECT statement always returns a single value
- An equivalent formulation may be defined using a join operation

7

Equivalent formulation

- The equivalent formulation with join is characterized by
  - a FROM clause including all the tables referenced by the FROM clauses of each SELECT statement
  - appropriate join conditions in the WHERE clause
  - if needed selection predicates added in the WHERE clause

8

FROM clause (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM S                      SX
WHERE City = (SELECT City   SY
              FROM S
              WHERE SId='S1');
```

9

FROM clause (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT ...
FROM S AS SX, S AS SY
...
```

10

Join condition (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```

11

Join condition (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT ...
FROM S AS SX, S AS SY
WHERE SX.City=SY.City
...
```

12

Selection predicate (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SId
FROM S
WHERE City = (SELECT City
              FROM S
              WHERE SId='S1');
```

13

SELECT clause (no.1)

- Find the codes of the suppliers that are based in the same city as S1

```
SELECT SY.SId
FROM S AS SX, S AS SY
WHERE SX.City=SY.City AND
      SX.SId='S1';
```

14

Equivalent formulation (no.2)

- Find the codes of the suppliers whose number of employees is smaller than the maximum number of employees

```
SELECT SId
FROM S
WHERE #Employees < (SELECT MAX(#Employees)
                    FROM S);
```

- An equivalent formulation with join is not possible

15

IN OPERATOR

- It expresses the concept of membership to a set of values

AttributeName IN *(NestedQuery)*

- It allows to write a query by
  - breaking down the problem into subproblems
  - following a "bottom-up" process
- The nested query can be replaced with a list of values
- The equivalent formulation with the join is characterized by
  - FROM clause containing the tables referenced in the FROM of all SELECTs
  - appropriate join conditions in the WHERE clause
  - any selection predicates added in the WHERE clause

16

16

The IN operator (no.1)

- Find the names of the suppliers who supply product P2
- Decomposition of the problem into two subproblems
  - codes of the suppliers of product P2
  - names of the suppliers with such codes

17

The IN operator (no.1)

- Find the names of the suppliers who supply product P2

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

SId
S1
S2
S3

```
(SELECT SId
 FROM SP
 WHERE PId='P2')
```

Codes of the suppliers of P2

18

The IN operator (no.1)

- Find the names of the suppliers who supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

```
SELECT SName
FROM S
WHERE SId IN (SELECT SId
              FROM SP
              WHERE PId='P2');
```

Set membership

Codes of the suppliers of P2

SId
S1
S2
S3

19

19

Example 1: Equivalent Formulation with Join

- Find the names of the suppliers who supply product P2

IN	JOIN
SELECT SName FROM S WHERE SId IN (SELECT SId FROM SP WHERE PId='P2');	SELECT SName FROM S, SP WHERE S.SId=SP.SId AND PId='P2';

20

20

Example 2: IN Operator

- Find the name of suppliers who provide at least one red product

```
SELECT SName
FROM S
WHERE SId IN (SELECT SId
              FROM SP
              WHERE PId IN (SELECT PId
                            FROM P
                            WHERE Color='Red'));
```

Red Product Codes

Red Product Supplier Codes

Supplier names with those codes

21

21

Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

IN	JOIN
SELECT SName FROM S 1 WHERE SId IN (SELECT SId FROM SP WHERE PId IN (SELECT PId FROM P WHERE Color='Red'));	SELECT SName FROM *S,SP* 1 WHERE *S.SId=SP.SId*

22

22

Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

IN	JOIN
SELECT SName FROM S WHERE SId IN (SELECT SId FROM SP 2 WHERE PId IN (SELECT PId FROM P WHERE Color='Red'));	SELECT SName FROM *S, SP, P* 2 WHERE *S.SId=SP.SId AND* *P.SId= SP.SId*

23

23

Example 2: Equivalent formulation

- Find the name of suppliers who provide at least one red product

IN	JOIN
SELECT SName FROM S WHERE SId IN (SELECT SId FROM SP WHERE PId IN (SELECT PId FROM P 3 WHERE Color='Red'));	SELECT SName FROM *S, SP, P* WHERE *S.SId=SP.SId AND* *P.SId= SP.Sid AND* 3 *Color = 'Red'*

24

24

NOT IN OPERATOR

- It expresses the concept of exclusion from a set of values

AttributeName NOT IN *(NestedQuery)*

- It requires the identification of an appropriate *set to be excluded* defined by
  - a nested query
  - a list of values
- There is no equivalent formulation with join

48

48

Example 1: Concept of exclusion

- Find the names of the suppliers who *do not* supply product P2
  - it is not possible to express the query with a join operation

~~SELECT SName~~
~~FROM S, SP~~
~~WHERE S.SId = SP.SId~~
~~AND PId <>'P2';~~

Wrong solution

- The query matches the request:
  - Find the name of suppliers who provide at least one product other than P2

DBMG 49

49

Wrong solution (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

R

SName
Smith
Jones
Clark

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

DBMG

50

The NOT IN operator (no.1)

- Find the names of the suppliers who *do not* supply product P2
- Set to be excluded
  - suppliers of product P2

```
SELECT SName
FROM S
WHERE SId NOT IN (SELECT SId
                  FROM SP
                  WHERE PId='P2');
```

does not belong to

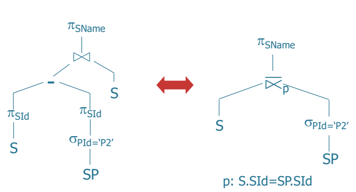
Codes of the suppliers who supply P2

DBMG

51

NOT IN and relational algebra (no.1)

- Find the names of the suppliers who *do not* supply product P2



DBMG

52

The NOT IN operator (no.2)

- Find the names of the suppliers who only supply product P2

Find the names of the suppliers of P2 who have never supplied products other than P2

- Set to be excluded
  - suppliers of products other than P2

DBMG

53

The NOT IN operator (no.2)

- Find the names of the suppliers who only supply product P2

```
SELECT SName
FROM S, SP
WHERE S.SId NOT IN (SELECT SId
                    FROM SP
                    WHERE PId<>'P2')
      AND S.SId=SP.SId;
```

Codes of the suppliers who supply at least one product other than P2

54

Alternative solution (no.2)

- Find the names of the suppliers who only supply product P2

```
SELECT SName
FROM S
WHERE S.SId NOT IN (SELECT SId
                    FROM SP
                    WHERE PId<>'P2')
      AND S.SId IN (SELECT SId
                    FROM SP);
```

Codes of the suppliers who supply at least one product other than P2

55

The NOT IN operator (no.3)

- Find the names of the suppliers who *do not* supply any red products
- Set to be excluded:
  - suppliers of red products, identified by their codes

```
SELECT SName
FROM S
WHERE SId NOT IN
       (SELECT SId
        FROM SP
        WHERE PId IN  (SELECT PId
                       FROM P
                       WHERE Color='Red'));
```

Codes of the suppliers of at least one red product

56

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

```
SELECT SName
FROM S
WHERE SId IN
       (SELECT SId
        FROM SP
        WHERE PId NOT IN (SELECT PId
                          FROM P
                          WHERE Color='Red'));
```

Codes of the suppliers of non-red products

- The set of elements to be excluded is incorrect

57

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

```
SELECT SName
FROM S
WHERE SId IN
       (SELECT SId
        FROM SP
        WHERE PId NOT IN (SELECT PId
                          FROM P
                          WHERE Color='Red'));
```

Codes of the suppliers of non-red products

- The set of elements to be excluded is incorrect

58

Wrong alternative (no.3)

- Find the names of the suppliers who *do not* supply any red products

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

59

TUPLE CONSTRUCTOR

- It allows defining a temporary structure for a tuple
  - the attributes belonging to it must be listed within ()

(AttributeName$_1$, AttributeName$_2$, ...)

- It enhances the expressive power of the IN and NOT IN operators

60

Example (no.1)

TRIP (TId, StartingPlace, Destination, DepartureTime, ArrivalTime)

- Find the pairs of starting places and destinations for which none of the trips lasts more than 2 hours

61

Example (no.1)

TRIP (TId, StartingPlace, Destination, DepartureTime, ArrivalTime)

- Find the pairs of starting places and destinations for which none of the trips lasts more than 2 hours

```
SELECT StartingPlace, Destination
FROM TRIP
WHERE (StartingPlace, Destination) NOT IN
      (SELECT StartingPlace, Destination
       FROM  TRIP
       WHERE ArrivalTime-DepartureTime>2);
```

Tuple constructor

62

EXISTS OPERATOR

- The EXISTS operator admits a nested query as a parameter and returns
  - true if the nested query returns a non-empty set (that is, it returns at least one tuple)
  - false if the internal query returns the empty set (i.e., no tuple)
- In the internal query of EXISTS, the SELECT clause is mandatory, but irrelevant, because the attributes are not displayed
- The correlation condition ties the execution of the internal query to the values of the attributes of the current tuple in the external query

63

The EXISTS operator (no.1)

- Find the names of the suppliers of product P2

⬇

Find the names of the suppliers for which there exists a product supply for P2

64

Correlation condition (no.1)

- Find the names of the suppliers of product P2

```
SELECT SName
FROM S
WHERE EXISTS (SELECT *
              FROM SP
              WHERE PId='P2'
                    AND SP.SId=S.SId);
```

Correlation condition

- The correlation condition ties the execution of the internal query to the values of the attributes of the current tuple in the external query

65

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

```
SELECT *
FROM SP
WHERE PId='P2'
      AND SP.SId='S1'
```

Value of SId in the current line of table S

66

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including EXISTS is true for S1 since there exists a supply for P2 by S1
  - S1 belongs to the result of the query

67

How EXISTS works (no.1)

- Find the names of the suppliers of product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

- The predicate including EXISTS is false for S4 since there does not exist a supply for P2 by S4
  - S4 does not belong to the result of the query

68

Result of the query (no.1)

- Find the names of the suppliers of product P2

R

SName
Smith
Jones
Blake

69

Scope of attributes

- A nested query may reference attributes defined within outer queries
- A query may not reference attributes defined
  - within a nested query at an inner level
  - within a different query at the same level

70

NOT EXISTS OPERATOR

- The EXISTS operator admits a nested query as a parameter and returns
  - true if the nested query returns an empty set (i.e., no tuple)
  - false if the nested query returns a non-empty set (that is, it returns at least one tuple)
- In the internal query of EXISTS, the SELECT clause is mandatory, but irrelevant, because the attributes are not displayed
- The correlation condition ties the execution of the internal query to the values of the attributes of the current tuple in the external query

71

71

The NOT EXISTS operator (no.1)

- Find the names of the suppliers that *do not* supply product P2

Find the names of the suppliers for which there does not exist a product supply for P2

The NOT EXISTS operator (no.1)

- Find the names of the suppliers who *do not* supply product P2

```
SELECT SName
FROM S
WHERE NOT EXISTS (SELECT *
                  FROM SP
                  WHERE PId='P2'
                  AND SP.SId=S.SId );
```

Correlation condition

- The correlation condition ties the execution of the internal query to the values of the attributes of the current tuple in the external query

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

```
SELECT *
FROM SP
WHERE PId='P2'
      AND SP.SId='S1'
```

Value of SId in the current line of table S

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

- The predicate including NOT EXISTS is false for S1 since there exists a supply for P2 by S1
  - S1 does not belong to the result of the query

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

- The predicate including NOT EXISTS is true for S4 since there does not exist a supply for P2 by S4
  - S4 does belong to the result of the query

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

How NOT EXISTS works (no.1)

- Find the names of the suppliers who *do not* supply product P2

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

- The predicate including NOT EXISTS is true for S5 since there exists a supply for P2 by S5
  - S5 does belong to the result of the query

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Result of the query (no.1)

- Find the names of the suppliers who *do not* supply product P2

R

SName
Clark
Adams

78

CORRELATION BETWEEN QUERIES

- It may be required to bind the computation of a nested query to the value(s) of one or more attributes in an outer query
  - the binding is expressed by one or more correlation conditions
- A correlation condition
  - must be specified in the WHERE clause of the nested query that requires it
  - is a predicate that binds some attributes of tables appearing in the nested query's FROM clause to attributes of tables appearing in the FROM clause of outer queries
- Correlation conditions may not be expressed
  - within queries at the same nesting level
  - with references to attributes of a table appearing in the FROM clause of a nested query

79

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId
FROM SP AS SPX
WHERE Qty = (...
                          )
```

Maximum quantity for the current product

80

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId
FROM SP AS SPX
WHERE Qty = (SELECT MAX(Qty)
             FROM SP AS SPY
             ...           )
```

Maximum quantity

81

Correlation among queries (no.1)

- For each product, find the code of the supplier who supplies the highest quantity

```
SELECT PId, SId
FROM SP AS SPX
WHERE Qty = (SELECT MAX(Qty)
             FROM SP AS SPY
             WHERE SPY.PId=SPX.PId);
```

Correlation condition

Maximum quantity for the current product

82

Correlation among queries (no.2)

TRIP (TId, StartingPlace, Destination, DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime < (...
                                   )
```

Average duration of trips on the current route

85

Correlation among queries (no.2)

TRIP (<u>TId</u>, StartingPlace, Destination, DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime <
      (SELECT AVG(ArrivalTime-DepartureTime)
       FROM  TRIP AS TB
       ...    )
```

Average duration of trips

86

Correlation among queries (no.2)

TRIP (<u>TId</u>, StartingPlace, Destination, DepartureTime, ArrivalTime)

- Find the codes of the trips whose duration is lower than the average duration of the trips on the same route (i.e., same starting place and destination)

```
SELECT TId
FROM TRIP AS TA
WHERE ArrivalTime-DepartureTime <
      (SELECT AVG(ArrivalTime-DepartureTime)
       FROM  TRIP AS TB
       WHERE TB.StartingPlace=TA.StartingPlace
             AND TB.Destination=TA.Destination);
```

Correlation conditions

87

DIVISION OPERATOR

- In SQL, the division operation can be performed using the COUNT operator, to verify that all the elements of interest belong to the reference set

88

88

The division operation (no.1)

- Find the codes of the suppliers who supply *all* products
- In relational algebra we must use the division operator

$$R = \pi_{SId,PId}(SP) \,/\, \pi_{PId}(P)$$

89

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products
- Remark
  - all products that may be supplied are listed in table P
  - a supplier is supplying all products if he or she is supplying a number of distinct products equal to the cardinality of P

90

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

```
SELECT COUNT(*)
FROM P
```

Total number of products

91

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

```
SELECT SId
FROM SP
GROUP BY SId
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM P)
```

For each supplier, total number of products supplied

Total number of products

Division in SQL (no.1)

- Find the codes of the suppliers who supply all products

SP(<u>SID</u>, <u>PID</u>, Qty)

```
SELECT SId
FROM SP
GROUP BY SId
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM P)
```

For each supplier, total number of products supplied

Total number of products

Division in SQL (no.1) – Different SP TABLE

- Find the codes of the suppliers who supply all products

SP(<u>SID</u>, <u>PID</u>, <u>Date</u>, Qty)

```
SELECT SId
FROM SP
GROUP BY SId
HAVING COUNT(DISTINCT PID)= (SELECT COUNT(*)
                             FROM P);
```

For each supplier, total number of products supplied

Total number of products

Division in SQL: procedure (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2
- We must count
  - the number of products supplied by S2
  - the number of products supplied both by an arbitrary supplier and by S2
- The two counts must be equal

Division in SQL (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2

```
SELECT COUNT(*)
FROM SP
WHERE SId='S2'
```

Number of products supplied by S2

Division in SQL (no.2)

- Find the codes of the suppliers who supply at least all of the products supplied by supplier S2

```
SELECT SId
FROM SP
WHERE PId IN (SELECT PId
              FROM SP
              WHERE SId='S2')
GROUP BY SId
HAVING COUNT(*)=(SELECT COUNT(*)
                 FROM SP
                 WHERE SId='S2');
```

For each supplier, the total number of products supplied, including only the products supplied by S2

Products supplied by S2

Number of products supplied by S2

Politecnico di Torino

Set Operators

0

SQL Language: Set Operators

- The UNION Operator
- The INTERSECT Operator
- The EXCEPT Operator

1

UNION

- Set union operator

A UNION B

- It performs the union of the two relational expressions A and B
  - relational expressions A and B may be generated by SELECT statements
  - it requires schema compatibility between A and B
  - removal of duplicates
    - UNION removes duplicates
    - UNION ALL does not remove duplicates

2

UNION: example

- Find the codes of products that are either red or supplied by supplier S2 (or both)

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400
S1	P1	300

3

UNION: example

- Find *the codes of products that are* either *red* or supplied by supplier S2 (or both)

```
SELECT PId
FROM P
WHERE Color = 'Red'
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris

PId
P1
P4

4

UNION: example

- Find the *codes of the products that are* either red or *supplied by supplier S2* (or both)

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S2	P6	100
S2	P1	300
S3	P2	400
S4	P2	200
S4	P3	200
S4	P4	300
S1	P5	400

```
SELECT PId
FROM SP
WHERE Sid = 'S2'
```

PId
P6
P1

5

UNION: example

- Find the codes of products that are either red or supplied by supplier S2 (or both)

```
SELECT PId
FROM P
WHERE Color = 'Red'

UNION

SELECT PId
FROM SP
WHERE Sid = 'S2'
```

Schema Compatibility

R

PId
P1
P4
P6

Removing the duplicate

6

6

UNION ALL: example

- Find the codes of products that are either red or supplied by supplier S2 (or both)

```
SELECT PId
FROM P
WHERE Color = 'Red'

UNION ALL

SELECT PId
FROM SP
WHERE Sid = 'S2'
```

Schema Compatibility

PId
P1
P4

PId
P1
P6

R

PId
P1
P1
P4
P6

Duplicates are not removed

7

7

INTERSECT

- Set intersection operator

A INTERSECT B

- It performs the intersection of the two relational expressions A and B
  - relational expressions A and B may be generated by SELECT statements
  - it requires schema compatibility between A and B

8

8

INTERSECT: example

- Find the cities where both one or more suppliers and one or more stores are based

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Blue	44	London
P5	Skirt	Blue	40	Paris

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

9

9

INTERSECT: example

- Find *the cities where* both *one or more suppliers* and one or more stores *are based*

```
SELECT City
FROM S
```

S

SId	NameS	#Employees	City
F1	Smith	2	London
F2	Jones	1	Paris
F3	Blake	3	Paris
F4	Clark	2	London
F5	Adams	3	Athens

City
London
Paris
Paris
London
Athens

10

10

INTERSECT: example

- Find *the cities where* both one or more suppliers and *one or more stores are based*

```
SELECT Store
FROM P
```

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

Store
London
Paris
Rome
London
Paris
London

11

11

INTERSECT: example

- Find the cities where both one or more suppliers and one or more stores are based

```
SELECT City
FROM S

INTERSECT

SELECT Store
FROM P;
```

City
London
Paris
Paris
London
Athens

Store
London
Paris
Rome
London
Paris
London

R

London
Paris

12

12

Equivalence with other operators

- The intersection operation may also be performed by means of JOIN and IN

JOIN

- The FROM clause contains the relations involved in the intersection
- The WHERE clause contains join conditions between the attributes listed in the SELECT clauses of relational expressions A and B

IN

- One of the two relational expressions is turned into a nested query using operator IN
- The attributes in the outer SELECT clause, grouped by a tuple constructor, make up the left-hand side of the IN operator

13

13

Example: equivalence with join

- Find the cities where both one or more suppliers and one or more stores are based

```
SELECT Store
FROM S, P
WHERE S.City =P.Store;
```

14

14

Example: equivalence with IN

- Find the cities where both one or more suppliers and one or more stores are based

```
SELECT Store
FROM P
WHERE Store IN (SELECT City
                FROM S);
```

15

15

EXCEPT

- Set difference operator

A EXCEPT B

- It subtracts relational expression B from relational expression A
  - it requires schema compatibility between A and B

16

16

EXCEPT: example

- Find the cities where one or more suppliers, but no stores are based

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

17

17

EXCEPT: example

- Find *the cities where one or more suppliers,* but no stores *are based*

SELECT City
FROM S

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

City
London
Paris
Paris
London
Athens

18

18

EXCEPT: example

- Find *the cities where* one or more suppliers, but no *stores are based*

SELECT Store
FROM P

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

Store
London
Paris
Rome
London
Paris
London

19

19

EXCEPT: example

- Find the cities where one or more suppliers, but no stores are based

SELECT City
FROM S

EXCEPT

SELECT Store
FROM P;

City
London
Paris
Paris
London
Athens

Store
London
Paris
Rome
London
Paris
London

R

Athens

20

20

Equivalence with the NOT IN operator

- The EXCEPT operation may also be performed by means of the NOT IN operator
  - relational expression B is nested within the NOT IN operator
  - the attributes in the SELECT clause of relational expression A, together by a tuple constructor, make up the left-hand side of the NOT IN operator

21

21

Equivalence with the NOT IN operator: example

- Find the cities where one or more suppliers, but no stores are based

```
SELECT City
FROM S
WHERE City NOT IN (SELECT Store
                   FROM P);
```

22

22

Advanced queries

SQL Language

0

SQL language: advanced queries

- Derived tables
- CTE
- Spatial queries
- JSON queries

1

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

2

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
```


3

Computing two-level aggregates (no.1)

- Find the maximum average (achieved by a student)

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

4

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

5

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
```

6

6

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

7

7

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;
```

8

8

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;
```

9

9

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)

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

10

10

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
```

11

11

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 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
Join condition
Correlation condition

12

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

13

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

14

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

15

Computing two-level aggregations (no.1)

- Find the maximum average (achieved by a student)

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

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

16

Computing two-level aggregations (no.1)

- Find the maximum average (achieved by a student)

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

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

17

Calculating aggregations with different granularities

- 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

18

18

Calculating aggregations with different granularities

- Step 1: find the average salary for each airline

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

19

19

Calculating aggregations with different granularities

- 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 )
```

20

20

Calculating aggregations with different granularities

- 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 > AvgSalary. AvgSal
```

21

21

Referenced CTE

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

CITY (<u>CodeC</u>, CName, Region)
DRIVER (<u>CodeD</u>, DName, Surname, 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

22

22

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, CName, Region
        FROM DAILY_RUN DR, DRIVER D, CITY C
        WHERE DR.CodeD =  D.CodeD AND D.CodeC = C.CodeC
        GROUP BY DR.CodeD, DR.CodeC, CName, Region )
```

23

23

Referenced CTE

- Step 2: calculate the average distance for each city

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

24

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
```

25

Recursive CTE syntax

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

26

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

27

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;
```

28

Recursive CTEs

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

EMPLOYEES

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

hierarchy

EID	Name	Surname	BossId*	Level
4	Roxanna	Fairlie	NULL	0

29

Recursive CTEs

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

EMPLOYEES

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

hierarchy

EID	Name	Surname	BossId*	Level
4	Roxanna	Fairlie	NULL	0
5	Hermie	Comsty	4	1
1	Domenic	Leaver	5	2
7	Faulkner	Challiss	5	2

DBMG 30

30

Recursive CTEs

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

EMPLOYEES

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

hierarchy

EID	Name	Surname	BossId*	Level
4	Roxanna	Fairlie	NULL	0
5	Hermie	Comsty	4	1
1	Domenic	Leaver	5	2
7	Faulkner	Challiss	5	2
3	Kakalina	Atherton	7	3
6	Pooh	Goss	7	3
2	Cleveland	Hewins	1	3

DBMG 31

31

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

32

32

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));
```

DBMG 33

33

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

DBMG 34

34

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

DBMG 35

35

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 36

36

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

37

37

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

Value

Embedded JSON

Array

DBMG 38

38

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"}

DBMG 39

39

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. $.a)
    - [i]: to access the i-th element of an array
    - wildcard * or ** ($.*)

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

RESULT
1

DBMG 40

40

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 41

41

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]"}

42

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"]

43

Politecnico di Torino

Update commands

SQL language: basics

0

SQL language: update commands

- ➢Introduction
- ➢The INSERT command
- ➢The DELETE command
- ➢The UPDATE command

1

1

Update instructions

- Update operations alter the state of the database
  - integrity constraints must be checked to ensure that they are still verified
- Each instruction may update the contents of a single table

- INSERT
  - inserting new tuples into a table
- DELETE
  - deleting tuples from a table
- UPDATE
  - modifying the content of tuples in a table

2

2

INSERT

- Inserting a single tuple
  - assignment of a constant value to each attribute

INSERT INTO *TableName*
[(ColumnList)]
VALUES *(CostantList);*

- Inserting multiple tuples
  - read from other tables by means of a SELECT command
  - it must not include an ORDER BY clause

INSERT INTO *TableName*
[(ColumnList)]
Query;

3

3

Example 1: Inserting a tuple

- Insert product P7 with Name: Jumper, Color: Purple, Size: 40, Store: Helsinki

```
INSERT INTO P (PId, PName, Color, Size, City)
        VALUES ('P7', 'Jumper', 'Purple',40,'Helsinki');
```

- A new tuple is inserted into table P with the specified values
- Omitting the field list is equivalent to specifying all fields, according to the column order specified upon table creation
  - If the table schema changes, the INSERT command is no longer valid

4

4

Example 2: Inserting a tuple

- Insert product P8 with Store: Istanbul, Size: 42

```
INSERT INTO P (PId, Store, Size)
        VALUES ('P8', 'Instanbul', 42);
```

- A new tuple is inserted into table P with the specified values
  - PName and Color are assigned the NULL value
- For all attributes whose values are not specified, the domain of the attribute must allow the NULL value

5

5

Example 3: Referential integrity constraints

- Insert a new supply for supplier S20, product P20 and quantity 1000

```
INSERT INTO SP (SId, PId, Qty)
    VALUES ('S20', 'P20', 1000);
```

- Referential integrity constraint
  - P20 and S20 must already be present in the P and S tables, respectively
  - if the constraint is not satisfied, the insertion should not be executed

DBMG 6

6

Example 4: Inserting multiple records

TOTAL-SUPPLIES (PId, TotalQty)

- For each product, insert the overall supplied quantity into table TOTAL-SUPPLIES
  - aggregate data extracted from table SP

```
INSERT INTO TOTAL-SUPPLIES (PId, TotalQty)
    (SELECT PId, SUM(Qty)
     FROM SP
     GROUP BY PId);
```

DBMG 7

7

DELETE

```
DELETE FROM TableName
[ WHERE predicate];
```

- Deletion of all tuples satisfying the predicate from table *TableName*
- It must be ensured that the deletion does not cause the violation of referential integrity constraints

8

8

Example 1: Clearing Table Contents

- Delete all supplies

```
DELETE FROM SP ;
```

- If no WHERE clause is specified, all tuples satisfy the selection predicate
  - the contents of table SP are deleted
  - the table itself is *not* deleted

DBMG 10

9

Example 2: Referential integrity contraints

- Delete the tuple corresponding to the supplier with code S1

```
DELETE FROM S
WHERE SId='S1';
```

- If SP includes supplies related to the deleted suppliers, the database loses its integrity
  - a violation of the referential integrity constraint between SP and S occurs
  - the deletion must be propagated

DBMG 10

10

Example 2: Referential Integrity constraints

- Delete the tuple corresponding to the supplier with code S1

```
DELETE FROM S
WHERE SId='S1';

DELETE FROM SP
WHERE SId='S1';
```

- To maintain integrity, the deletion operations must be completed on both tables

DBMG 11

11

Example 3: Referential integrity constraints

- Delete the suppliers based in Paris
- If SP includes supplies referring to the deleted suppliers, the referential integrity constraint between SP and S is violated
  - such supplies must also be deleted from SP

```
DELETE FROM SP
WHERE SId IN (SELECT SId
              FROM S
              WHERE City='Paris');

DELETE FROM S
WHERE City='Paris';
```

DBG 12

12

UPDATE

UPDATE *TableName*
SET *column = expression*
{, column=expression}
[WHERE *predicate];*

- All records in table *TableName* satisfying the predicate are modified according to the assignment *column=expression* in the SET clause

13

13

Example 1: Updating a tuple

- Update the features of product P1: assign Yellow to Color, increase the size by 2 and assign NULL to Store

```
UPDATE P
SET Color = 'Yellow',
    Size=Size+2,
    Store = NULL
WHERE PId='P1';
```

- The tuple identified by code P1 is updated

DBG 14

14

Example 2: Multiple updates

- Update all suppliers based in Paris by doubling the number of employees

```
UPDATE S
SET #Employees=2*#Employees
WHERE City='Paris';
```

- All tuples selected by the predicate in the WHERE clause are updated

DBG 15

15

Example 3: Update with nested query

- Update to 10 the quantity of supplied products for all suppliers based in Paris

```
UPDATE SP
SET Qty = 10
WHERE SId IN (SELECT SId
              FROM S
              WHERE City='Paris');
```

DBG 16

16

Example 4: Updating multiple tables

- Change the code of supplier S2 to S9

```
UPDATE S
SET SId='S9'
WHERE SId='S2';
```

- If SP includes supplies related to the updated suppliers, the referential integrity constraint is violated
  - such supplies must also be updated in SP

DBG 17

17

Example 4: Updating multiple tables

- Change the code of supplier S2 to S9

```
UPDATE S
SET SId='S9'
WHERE SId='S2';

UPDATE SP
SET SId='S9'
WHERE SId='S2';
```

- To maintain integrity, the update must be completed on both tables (integrity constraints checking must be temporarily disabled)

DBMG 18

18

Politecnico di Torino

Table management

The SQL Language

0

The SQL Language: Table management

- Creating a table
- Altering a table
- Deleting a table
- The data dictionary
- Data integrity

1

Creating tables

Table management

2

2

CREATE

```
CREATE TABLE TableName
(AttributeName Domain [DefaultValue ] [Constraints]
{ , AttributeName Domain [DefaultValue ] [Constraints]}
OtherConstraints
);
```

- It allows
  - defining all attributes (i.e., columns) of the table
  - defining integrity constraints on the table data
- Domain
  - it defines the data type of an attribute
  - predefined domains of the SQL language (elementary domains)
  - user-defined domains (starting from the predefined domains)
- Constraints
  - integrity constraints for the specific attribute
- OtherConstraints
  - general integrity constraints on the table

3

3

Domain definitions

- *Default*
  - it allows specifying a default value for the attribute
- *GenericValue*
  - a value compatible with the attribute domain
- *USER
  - user identifier
- NULL
  - standard default value

DEFAULT < *GenericValue* | USER | CURRENT_USER | SESSION_USER | SYSTEM_USER | NULL>

4

4

Elementary domains

Data type	SQL
Text	CHARACTER [VARYING] [(*Length*)] [CHARACTER SET *CharacterFamilyName*] VARCHAR (Length) TEXT
Binary	BIT [VARYING] [(*Length*)] BLOB BINARY
Boolean	BOOLEAN
Integer numbers	INTEGER SMALLINT BIGINT
Real numbers	NUMERIC [(*Precision, Scale*)] DECIMAL [(*Precision, Scale*)] FLOAT [(*n*)] REAL DOUBLE PRECISION

5

5

Elementary domains: real numbers

- Exact representations
  - NUMERIC and DECIMAL are base-ten numbers
  - Precision: total number of digits
  - Scale: number of decimal places
  - Example: for number 123.45 precision is 5, scale is 2
- Approximate numeric domains
  - FLOAT (n): n specifies precision
  - it is the number of bits used to store the mantissa of a floating point number represented in scientific notation
  - it is a value ranging from 1 to 53 (the default value is 53)

6

Domini elementari

Tipologia di dato	SQL
Time	TIMESTAMP *[(Precision)]* [WITH TIME ZONE] DATE DATETIME
JSON	JSON
Spatial	SDO_GEOMETRY GEOMETRY POINT LINESTRING POLYGON

The definition of data types in SQL differs depending on the DBMS used

7

7

Domini elementari

Tipologia di dato	SQL
Time	TIMESTAMP *[(Precision)]* [WITH TIME ZONE] DATE DATETIME
JSON	JSON
Spatial	...D_GEOMETRY ...OMETRY ...INT ...ESTRING ...LYGON

- Stores the year, the month, the day, the hour, the minutes, the seconds and possibly the fraction of second
- it uses 19 characters, plus the characters needed to represent the precision

YYYY-MM-DD hh:mm:ss:p

The definition of data types in SQL differs depending on the DBMS used

8

8

Defining a domain (1/2)

CREATE DOMAIN *DomainName* AS *DataType*

[*DefaultValue*] [*Constraint*]

- It defines a new domain that may be used in attribute definitions
  - *DataType* is an elementary domain
- Example

```
CREATE DOMAIN Grade AS SMALLINT
    DEFAULT NULL
    CHECK (Grade >= 18 and Grade <=30)
```

9

Definition of supplier-product database

- Creating the Supplier Table

```
CREATE TABLE S (
    SId         CHAR(5),
    SName       CHAR(20),
    NEmployees  SMALLINT,
    City        CHAR(15));
```

- Creating the Supply Table

```
CREATE TABLE SP (
    SId         CHAR(5),
    PId         CHAR(6),
    Qty         INTEGER);
```

- Creating the Product Table

```
CREATE TABLE P (
    PId         CHAR(6),
    PName       CHAR(20),
    Color       CHAR(6),
    Size        SMALLINT,
    Store       CHAR(15));
```

The definition of integrity constraints is missing

10

10

Altering tables

Table management

11

11

ALTER TABLE

- The following “alterations” are possible
  - adding a new column
  - defining a new default value for an existing column (attribute)
    - for example, replacing a previous default value
  - deleting an existing column (attribute)
  - defining a new integrity constraint
  - deleting an existing integrity constraint

```
ALTER TABLE TableName
< ADD COLUMN <Attribute-Definition> |
  ALTER COLUMN AttributeName
    < SET <Default-Value-Definition> | DROP DEFAULT>|
  DROP COLUMN AttributeName
    < CASCADE | RESTRICT > |
  ADD  CONSTRAINT [ConstraintName]
    < Unique-Constraint-Definition > |
    < Integrity-Constraint-Definition > |
    < Check-Constraint-Definition > |
  DROP CONSTRAINT [ConstraintName]
    < CASCADE | RESTRICT >
```

- RESTRICT (default option)
  - the element (column or constraint) is not removed if it appears in the definition of some other element
- CASCADE
  - all elements with a dependency on a deleted element will be removed, until there are no unresolved dependencies

12

12

Deleting tables

Table management

DBMG 13

13

DROP TABLE

```
DROP TABLE TableName
  [ RESTRICT | CASCADE];
```

- All rows in the table are deleted along with the table
- RESTRICT
  - the table is not deleted if it appears in the definition of some table, constraint or view
  - default option
- CASCADE
  - if the table appears in the definition of some view, the latter is also deleted

14

14

Data Dictionary

DBMG 15

15

The data dictionary

- Metadata are information (data) about data
  - they may be stored in database tables
- The data dictionary contains the metadata of a relational database
  - it contains information about the database objects
  - it is managed directly by the relational DBMS
  - it may be queried by means of SQL commands
- It contains various information
  - descriptions of all database structures (tables, indices, views)
  - SQL stored procedures
  - user privileges
  - statistics
    - on the database tables
    - on the database indices
    - on the database views
    - on the evolution of the database

DBMG

16

Information about tables

- For each database table, the data dictionary contains
  - table name and physical structure of the file storing the table
  - name and data type for each attribute
  - name of all indices created on the table
  - integrity constraints

DBMG

17

Data dictionary tables

- Data dictionary information is stored in several tables
  - each DBMS uses different names for different tables
- The data dictionary may be queried by means of SQL commands

18

The Oracle data dictionary

- In Oracle 3 collections of information are defined for the data dictionary
  - USER_*: metadata related to the current user's data
  - ALL_*: metadata related to all users' data
  - DBA_*: metadata about system tables
- USER_* contains different tables and views, including:
  - USER_TABLES contains metadata to the user tables
  - USER_TAB_STATISTICS contains statistics computed on the user tables
  - USER_TAB_COL_STATISTICS contains statistics computed on user table columns

19

Querying the data dictionary no. 1

- Show the name of user-defined tables and the number of tuples stored in each table

```
SELECT Table_Name, Num_Rows
FROM USER_TABLES;
```

R

Table_Name	Num_Rows
S	5
P	6
SP	12

20

Querying the data dictionary no.2 (1/2)

- For each attribute in the supplier-product table, show the attribute name, the number of distinct values and the number of tuples with a NULL value

```
SELECT Column_Name, Num_Distinct, Num_Nulls
FROM USER_TAB_COL_STATISTICS
WHERE Table_Name = 'SP'
ORDER BY Column_Name;
```

R

Column_Name	Num_Distinct	Num_Nulls
SId	4	0
PId	6	0
Qty	4	0

21

Data Integrity

Table management

22

22

Integrity constraints

- Data in a database are correct if they satisfy a set of correctness rules
  - rules are called *integrity constraints*
  - example: Qty >=0
- Data update operations define a new state for the database, which may not necessarily be correct
- Checking the correctness of a database state may be done
  - by *application procedures*, performing all required checks
  - through the definition of *integrity constraints* on the tables
  - through the definition of *triggers*

23

Application procedures

Each application includes all required correctness checks

Pros	Cons
• "flexible" approach	• checks may be "circumvented" by interacting directly with the DBMS • a coding error may have significant effects on the database • the knowledge about integrity constraints is typically "hidden" inside applications

24

24

Table integrity constraints

- Integrity constraints are
  - defined in the CREATE or ALTER TABLE statements
  - stored in the system data dictionary
- Each time data are updated, the DBMS automatically verifies that the constraints are satisfied

25

Table integrity constraints

Pros	Cons

• *declarative* definition of constraints, whose verification is delegated to the system
  ◦ the data dictionary describes all of the constraints in in the system
• unique centralized check point
  ◦ constraint verification may not be circumvented
	• they may slow down application execution
	• it is not possible to define constraints of an arbitrary type
	  ◦ example: constraints on aggregated data

26

26

Triggers

- Triggers are procedures executed automatically when specific data updates are performed
  - defined through the CREATE TRIGGER command
  - stored in the system data dictionary
- When a modification event occurs on data under the trigger's control, the procedure is automatically executed

27

Trigger

Pros	Cons

• they allow defining complex constraints
  ◦ normally used in combination with constraint definition on the tables
• unique centralized check point
  ◦ constraint verification may not be circumvented
	• complex
	• they may slow down application execution

28

28

Fixing violations

- If an application tries to execute an operation that causes a constraint violation, the system may
  - block the operation, causing an error in the application execution
  - execute a compensating action so that a new correct state is reached
    - example: when a supplier is deleted, its supplies are also deleted

29

Integrity constraints in SQL-92

- The SQL-92 standard introduced the possibility to specify integrity constraints in a declarative way, delegating to the system the verification of their consistency
  - table constraints
    - restrictions on the data allowed in table columns
  - referential integrity constraints
    - manage references among different tables
      - based on the concept of foreign key

30

Table Constraints

- They are defined on one or more columns of a table
- They are defined in the creation instructions of:
  - Tables
  - Domains
- Type of constraints:
  - Primary key
  - Admissibility of NULL values
  - Uniqueness
  - General tuple constraints
- They are checked after each SQL statement that operates on the table subject to the constraint
  - Entering new data
  - Changing the value of constrained columns
- If a constraint is violated, the SQL statement that caused the violation results in an execution error

31

31

Primary Key

- A primary key is a set of attributes that uniquely identifies rows in a tables
- Only one primary key may be specified for a given table
- Primary key definition
  - composed of a single attribute

    AttributeName Domain PRIMARY KEY

  - composed of one or more attributes

    PRIMARY KEY (*ListOfAttributes*)

32

32

Primary Key examples

a single attribute

```
CREATE TABLE S (  SId          CHAR(5) PRIMARY KEY,
                  SName        CHAR(20),
                  NEmployees   SMALLINT,
                  City         CHAR(15))
```

one or more attributes

```
CREATE TABLE SP (
                  SId          CHAR(5),
                  PId          CHAR(6),
                  Qty          INTEGER,
                  PRIMARY KEY (SId, PId));
```

33

33

Admissibility of the NULL value

- The NULL value indicates absence of information
- When a value must always be specified for a given attribute

  AttributeName Domain NOT NULL

  - NULL value is not allowed

34

34

NOT NULL: example

```
CREATE TABLE S (SId          CHAR(5),
                SName        CHAR(20) NOT NULL,
                NoEmployees  SMALLINT,
                City         CHAR(15));
```

35

UNIQUE

- An attribute or a set of attributes may not take the same value in different rows of the table
  - for a single attribute

    AttributeName Domain UNIQUE

  - for one or more attributes

    UNIQUE (*ListOfAttributes*)

- It is possible to repeat the NULL value (it is seen as a different value in each row)

36

36

Candidate key

- A candidate key is a set of attributes that may serve as a primary key
  - it is unique
  - it does not allow the NULL value
- The combination UNIQUE NOT NULL defines a candidate key that does not allow null values

AttributeName Domain UNIQUE NOT NULL

DBMG

37

Unique constraint: example

```
CREATE TABLE P (    PId     CHAR(6),
                    PName   CHAR(20) NOT NULL UNIQUE,
                    Color   CHAR(6),
                    Size    SMALLINT,
                    Store   CHAR(15));
```

DBMG

38

General Tuple Constraints

- They allow expressing general conditions on each tuple
  - tuple or domain constraints

    AttributeName Domain CHECK (*Condition*)

- Predicates that can be specified in the WHERE clause can be specified as a condition
- The database is correct if the condition is true

39

39

General tuple constraints: example

```
CREATE TABLE S (SId           CHAR(5) PRIMARY KEY,
                SName         CHAR(20) NOT NULL,
                NoEmployees   SMALLINT
                                  CHECK (NoEmployees>0),
                City          CHAR(15));
```

DBMG

40

Referential Integrity Constraints

- They manage the link between tables by means of the value of attributes
- The foreign key is defined in the CREATE TABLE statement of the referencing table

  FOREIGN KEY (*ListReferencingAttributes*)
  REFERENCES *TableName* [(*ListReferencedAttributes*)]

- If the referenced attributes have the same name as the referenced attributes, they are not required

41

41

Example: Defining a Foreign Key

```
CREATE TABLE SP (
    SId        CHAR(5),
    PId        CHAR(6),
    Qty        INTEGER,
    PRIMARY KEY (SId, PId),
    FOREIGN KEY (SId)
        REFERENCES S(SId),
    FOREIGN KEY (PId)
        REFERENCES P(PId));
```

42

42

Politiche di gestione dei vincoli

- Integrity constraints are checked after each SQL command that may cause their violation
- Insert or update operations on the referencing table that violate the constraints are not allowed
- In the CREATE TABLE statement of the referencing table

    FOREIGN KEY (*ListReferencingAttributes*)
    REFERENCES
    TableName [(*ListReferencedAttributes*)]
    [ON UPDATE
    <CASCADE | SET DEFAULT | SET NULL | NO ACTION>]
    [ON DELETE
    <CASCADE | SET DEFAULT | SET NULL | NO ACTION>]

- Update or delete operations on the referenced table have the following outcome on the referencing table:
    - CASCADE: the update or delete operation is propagated
    - SET NULL/DEFAULT: a null or default value is set in the columns for the tuples whose values are no longer present in the referenced table
    - NO ACTION: the offending action is not executed

43

43

Example: Product-Supply DB

- table P: describes the available products
    - Primary key: PId
    - Product name cannot have NULL or duplicate values
    - The size is always greater than zero
- table S: describes suppliers
    - Primary key: Sid
    - Supplier name cannot have NULL or duplicate values
    - The number of employees is always greater than zero
- table SP: describes supplies, relating products to the suppliers who supply them
    - Primary key: (SId, PId)
    - Quantity cannot be null and is greater than zero
    - Referential integrity constraints

44

44

Constraint Management: Example 1

- SP (referencing table)
    - insert (new tuple PId, SId) -> No
    - update (SId) -> No
    - delete (tuple) -> Ok
- S (referenced table)
    - insert (new tuple) -> Ok
    - update (SId) -> cascaded update (cascade)
    - delete (tuple) -> cascaded update (cascade)
      prevent action (no action)

45

SQL Example: Product-Supply DB

```
CREATE TABLE P ( PId        CHAR(6) PRIMARY KEY,
                 PName      CHAR(20) NOT NULL UNIQUE,
                 Color      CHAR(6),
                 Size       SMALLINT
                              CHECK (Size > 0),
                 Store      CHAR(15));

CREATE TABLE S (SId           CHAR(5) PRIMARY KEY,
                SName         CHAR(20) NOT NULL UNIQUE,
                NoEmployees   SMALLINT
                                CHECK (NoEmployees>0),
                City          CHAR(15));

CREATE TABLE SP (SId        CHAR(5),
                 PId        CHAR(6),
                 Qty        INTEGER
                   CHECK (Qty IS NOT NULL and Qty>0),
                   PRIMARY KEY (SId, PId),
                   FOREIGN KEY (SId)
                              REFERENCES S(SId)
                   ON DELETE NO ACTION
                   ON UPDATE CASCADE,
                   FOREIGN KEY (PId)
                              REFERENCES P(PId)
                   ON DELETE NO ACTION
                   ON UPDATE CASCADE);
```

46

46

Constraint Management: Example 2

- Employees (<u>EId</u>, EName, City, DId)
- Departments (<u>DId</u>, DName, City)

- Employees (referencing table)
    - insert (new tuple) -> No
    - update (DId) -> No
    - delete (tuple) -> Ok
- Departments (referenced table)
    - insert (new tuple) -> Ok
    - update (DId) -> cascaded update (cascade)
    - delete (tuple) -> cascaded update (cascade)
      prevent action (no action)
      set to unknown value (set null)
      set to default value(set default)

47

Politecnico di Torino

Advanced topics

SQL language

0

SQL Language: Advanced Topics

- Views
- Transactions
- Access control
- Index management
- Physical design

1

Views

Advanced Topics

2

The concept of view

- A view is a *"virtual"* table
  - the content (tuples) is defined by means of an SQL query on the database
    - the content of the view depends on the content of the other tables present in the database
  - the content is *not* memorized physically in the database
    - it is recalculated every time the view is used by executing the query that defines it
- A view is an object of the database
  - it can be used in queries as if it were a table
- If the query refers to a view, it has to be reformulated by the DBMS before execution
- This operation is carried out automatically
  - the references to the view are substituted by its definition

3

Example n.1: definition of the view

- Definition of the view *small suppliers*
  - the suppliers that have fewer than 3 employees are considered "small suppliers"
- The view "small suppliers"
  - contains the code, name, number of employees and city of the suppliers that have fewer than 3 employees.

Name of the views

```
CREATE VIEW SMALL_SUPPLIERS AS
SELECT SId, SName, #Employees, City
FROM S
WHERE #Employees<3;
```

Query associated with the view

4

Example n.1: query

- View the code, name, employee number and city of "small suppliers" in London
- The query can be answered without using views

```
SELECT *
FROM S
WHERE #Employees<3 AND City='London';
```

- The query can be answered using the view defined previously

```
SELECT *
FROM SMALL_SUPPLIERS
WHERE City='London';
```

- The view SMALL_SUPPLIERS is used like a table

5

Example n.2: definition of the view

- Definition of the view *number of suppliers per product*
  - The view contains the product code and the number of different suppliers providing it

Attributes of the view

Name of the view

```
CREATE VIEW NUMSUPPLIERS_PER_PRODUCT
(PId, #Suppliers) AS
SELECT SId, COUNT(*)
FROM SP
GROUP BY PId;
```

Query associated with the view

6

6

Advantages of views

- Simplification of queries
  - by breaking down a complex query into subqueries associated with the views
- Security management
  - it is possible to introduce different privacy protection mechanisms for each user or group
    - access authorization is associated with the view
    - each user, or group, accesses the database only via views that are appropriate for the operation they are authorized to carry out
- Database maintenance and evolution
  - If a database is restructured, it is possible to change the views
    - it is not necessary to re-formulate the queries written before the restructuring and present in the applications that have already been developed

7

7

Creation of views

CREATE VIEW ViewName [(AttributList)]
AS SQLquery;

- If the names of the attributes of a view are not specified
  - use those present in the SQL query SELECT
- Attribute names have to be specified if
  - they represent the result of an internal function
  - they represent the result of an expression
  - they are constant
  - two columns (from different tables) have the same name

8

8

Cancelling views

DROP VIEW ViewName;

- Cancelling a table that a view refers to can have various effects
  - automatic elimination of the associated views
  - automatic invalidation of the associated views
  - prohibition to execute the operation of cancelling the table
- the effect depends on the specific DBMS

9

9

Updating views

- It is possible to update the data in a view *only* for some types of views
- Only views in which a single row of each table corresponds to a single row of the view can be updated (Standard SQL-92)
  - univocal correspondence between the tuple of the view and the tuple of the table on which it is defined
  - it is possibile to propagate without ambiguity the changes made to the view to each table on which it is defined
- *It is not possible to update* a view which in the outermost block of the query that defines it:
  - does not contain the primary key of the table on which it is defined
  - contains joins that represent one-to-many or many-to-many matches
  - contains aggregated functions
  - contains the DISTINCT keyword
- Some non-updatable views can become updatable by modifying the SQL expression associated with the view
  - it may be necessary to reduce the information content of the view

10

10

Example n.1

- View SUPPLIER_CITY

```
CREATE VIEW SUPPLIER_CITY AS
SELECT SId, City
FROM S;
```

11

11

Example n.1: insertion

- Insertion in SUPPLIER_CITY of

('S10', 'Rome')

- corresponds to the insertion in S of

('S10',NULL,NULL,'Rome')

  - the attributes SName, #Employees have to admit the value NULL

12

Example n.1: deletion

- Deletion from SUPPLIER_CITY of

('S1', 'London')

- corresponds to the deletion from S of

('S1', 'Smith',20,'London')

  - identification of the tuple to delete is enabled by the primary key

13

Example n.1: update

- update in SUPPLIER_CITY of

('S1', 'London') to ('S1', 'Milan')

- update in S of

('S1', 'Smith',20,'London') to ('S1', 'Smith',20,'Milan')

  - identification of the tuple to be updated is enabled by the primary key

14

Example n.1: updating

- The view SUPPLIER_CITY *can be updated*
  - each tuple of the view corresponds to a single tuple of table S
  - the changes carried out on the view can be propagated to the table on which it is defined

15

Example n.2

- View NUMEMPLOYEE_CITY

```
CREATE VIEW NUMEMPLOYEE_CITY AS
SELECT DISTINCT #Employees, City
FROM   S;
```


16

Esempio n.2: insertion

- Insertion in NUMEMPLOYEE_CITY of

(40, 'Rome')

- it is impossible to insert in S

(NULL, NULL, 40, 'Rome')

  - the value of the primary key is missing

17

Example n.2: deletion

- Deletion from NUMEMPLOYEE_CITY of

(20, 'London')

  - several tuples are associated with the pair (20, 'London')
    - Which tuple has to be deleted from S?

DBG 18

18

Example n.2: update

- Update in NUMEMPLOYEE_CITY of

(20, 'London') to (30, 'Rome')

  - Several tuples are associated with the pair (20, 'London')
    - Which tuple has to be updated in S?

DBG 19

19

Example n.2: updating

- The view NUMEMPLOYEE_CITY *cannot be updated*
  - the primary key of table S is not present in the view
    - the insertion of new tuples in the view cannot be propagated to S
  - some tuples of the view correspond to several tuples in the table S
    - the association between the tuples in the view and the tuples in the table is ambiguous
    - it is not possible to propagate the changes carried out on the tuples of the view to the tuples of the table on which it is defined

DBG 20

20

Example n.3: non-updatable view

```
CREATE VIEW SUPPLIER_LONDON AS
SELECT *
FROM S
WHERE City='London';
```

- The view is non-updatable
  - it does not explicitly select the primary key of table S
- It is sufficient to replace the symbol "*" with the name of the attributes

DBG 21

21

Example 4: non-updatable view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT DISTINCT SId, SName
FROM S, SP
WHERE S.SId = SP.SId AND Qty >100;
```

- The view is not updatable
  - there is a join
  - the DISTINCT keyword is present

DBG 22

22

Example n.4: changed view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT SId, SName
FROM S
WHERE SId IN (SELECT SId
   FROM SP
   WHERE Qty>100);
```

- The view is updatable
  - the join was removed using the IN operator
  - the keyword DISTINCT is no longer necessary

! It is not always possible to rewrite the query to make the view updatable

DBG 23

23

Transaction

Advanced Topics

Transaction

- A transaction is necessary when several users can simultaneous access the same data
- It provides efficient mechanisms to
  - manage competing access to data
  - recovery after a malfunction
- It is a logical unit of work, which cannot be further broken down
  - a sequence of data modification operations (SQL statements) that takes the database from one consistent state to another consistent state
  - there is no need to maintain consistency in intermediate states
- A system that makes a mechanism available for definining and executing transactions is called a transactional system
- The DBMS contains architecture blocks that offer transaction management services

Beginning a transaction

- To define the beginning of a transaction, the SQL language uses the instruction
  - START TRANSACTION
- Usually the instruction to begin a transaction is omitted
  - the beginning is implicit for
    - the first SQL instruction of the programme that accesses the database
    - the first SQL instruction following the instruction ending the previous transaction

Ending a transaction

- The SQL language has instructions for defining the end of a transaction
  - Transaction successful
    - COMMIT [WORK]
    - the action associated with the instruction is called *commit*
  - Transaction failed
    - ROLLBACK [WORK]
    - the action associated with the instruction is called *abort*

Commit

- Action executed when a transaction ends with success
- The database is in a new (final) correct state
- The changes to the data executed by the transaction become
  - permanent
  - visibile to other users

Rollback

- Action executed when a transaction ends because of an error
  - for example, an error in application
- All the operations modifying the data executed during the transaction are “cancelled”
- The database returns to the state prior to the beginning of the transaction
  - the data is visible again to the other users

Example

- Transfer the sum of 100
  - from current account number IT92X0108201004300000322229
  - to current account number IT32L0201601002410000278976

```
START TRANSACTION;

UPDATE Account
  SET Balance= Balance - 100
  WHERE IBAN='IT92X0108201004300000322229';

UPDATE Account
  SET Balance = Balance + 100
  WHERE IBAN= 'IT32L0201601002410000278976';

COMMIT;
```

30

30

Properties of transactions

- The principal properties of transactions are
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- They are summarized by the English acronym *ACID*

31

31

Atomicity

- A transaction is an indivisible unit (atom) of work
  - all the operations contained in the transaction have to be executed
  - or none of the operations contained in the transaction have to be executed
    - the transaction has no effect on the database
- The database cannot remain in an intermediate state during the processing of a transaction

32

32

Consistency

- The execution of a transaction has to take the database
  - from an initial state of consistency (correct)
  - to a final state of consistency
- Correctness is verified by integrity constraints defined on the database
- When there is a violation of the integrity constraint the system intervenes
  - to abort the transaction
  - or to modify the state of the database by eliminating the violation of the constraint

33

33

Isolation

- The execution of a transaction is independent from the simultaneous execution of other transactions
- The effects of a transaction are not visible by other transactions until the transaction is terminated
  - the visibility of unstable intermediate states is avoided
    - an intermediate state can be aborted by a subsequent rollback
    - in case of a rollkback, it would be necessary to rollback other transactions that have observed the intermediate state (domino effect)

34

34

Durability

- The effect of a transaction that has executed a commit is memorized permanently
  - the changes to the data carried out by a transaction ending successfully are permanent after a commit
- It guarantees the reliability of the operations of data modification
  - the DBMS provides mechanisms for recovery to the correct state of the database after a malfunction has occurred

35

35

Access control

Advanced topics

36

Data security

- Protection of data from
  - unauthorized readers
  - alteration or destruction
- The DBMS provides protection tools which are defined by the database administrator (DBA)
- Security control verifies that users are authorized to execute the operations they request
- Security is guaranteed through a set of constraints
  - specificied by the DBA in an appropriate language
  - memorized in the data dictionary system

37

Resources

- Any component of the database scheme is a resource
  - table
  - view
  - attribute in a table or view
  - domain
  - procedure
  - ...
- Resources are protected by the definition of *access privileges*

38

Access privileges

- Describe access rights to system resources
- SQL provides very flexible access control mechanisms for specifying
  - the resources users can access
  - the resources that have to remain private

39

Privileges: characteristics

- Each privilege is characterized by the following information
  - the resource it refers to
  - the type of privilege
    - describes the action allowed on the resource
  - the user granting the privilege
  - the user receiving the privilege
  - the faculty to transmit the privilege to other users

40

Types of privilege

- INSERT
  - enables the insertion of a new object in the resource
  - valid for tables and views
- UPDATE
  - enables updating the value of an object
  - valid for tables, views and attributes
- DELETE
  - enables removal of objects from the resource
  - valid for tables and views
- SELECT
  - enables using the resource in a query
  - valid for tables and views
- REFERENCES
  - enables referring to a resource in the definition of a table scheme
  - can be associated with tables and attribu tes
- USAGE
  - enables use of the resource (e.g. a new type of data) in the definition of new schemes

41

Resource creator privileges

Resource creator

- When a resource is created, the system grants all privileges over that resource to the user who created it
- Only the resource creator has the privilege to eliminate a resource (DROP) and modify a scheme (ALTER)
  - the privilege to eliminate and modify a resource cannot be g ranted to any other user

System administrator

- The system administrator (user system) possesses all privileges over all the resources

42

42

Management of privileges in SQL

- Privileges are granted or revoked using SQL instructions
  - GRANT
    - grants privileges over a resource to one or more users
  - REVOKE
    - revokes privileges granted to one or more users

43

43

GRANT

GRANT *PrivilegeList* ON *ResourceName*
TO *UserList*
[WITH GRANT OPTION]

- *PrivilegeList*
  - specifies the list of privileges
  - ALL PRIVILEGES
    - Keyword for identifying all privileges
- *ResourceName*
  - specifies the resource for which the privilege is granted
- *UserList*
  - Specifies the users who are granted the privilege
- WITH GRANT OPTION
  - faculty to transfer the privilege to other users

44

44

GRANT

GRANT *PrivilegeList* ON *ResourceName*
TO *UserList*
[WITH GRANT OPTION]

- *PrivilegeList*
  - specifies the list of privileges
  - ALL PRIVILEGES
    - Keyword for identifying all privileges
- *ResourceName*
  - specifies the resource for which the privilege is granted
- *UserList*
  - Specifies the users who are granted the privilege
- WITH GRANT OPTION
  - faculty to transfer the privilege to other users

45

45

Examples

GRANT ALL PRIVILEGES
ON P TO Smith, Singh

- Users Smith and Singh are granted all privileges for table P

GRANT SELECT ON S TO Smith
WITH GRANT OPTION

- User Smith is granted the privilege to SELECT in table S
- User Smith has the faculty to grant the privilege to other users

46

46

REVOKE

REVOKE *PrivilegeList* ON *ResourceName*
FROM *UserList*
[RESTRICT|CASCADE]

- Can remove
  - all the privileges that have been granted
  - a subset of privileges granted
- RESTRICT
  - the command must not be executed if revoking the user's privileges entails revoking other privileges
    - Example: the user has received the privileges with the GRANT OPTION and has propagated the privileges to other users
  - default value
- CASCADE
  - revokes also all the privileges which have been propagated
    - generates a chain reaction
  - for each privilege revoked
    - all granted privileges are revoked in a cascade
    - all database elements which have been created exploiting these privileges are removed

47

47

Examples

REVOKE UPDATE ON P FROM White

- User White's privilege to UPDATE table P is revoked
  - the command is not executed if it entails revoking the privilege of other users

REVOKE SELECT
ON S FROM Red CASCADE

- User Red's privilege to SELECT table S is revoked
- User Red had received the privilege through GRANT OPTION
  - if Red has propagated the privilege to other users, the privilege is revoked in cascade
  - if Red has created a view using the SELECT privilege, the view is removed

48

48

Concept of role

- The role is an access profile
  - Defined by its set of privileges
- Each user has a defined role
  - it enjoys the privileges associated with that role
- Advantages
  - access control is more flexible
    - a user can have different roles at different times
  - it simplifies administration
    - an access profile need not be defined at the moment of its activation
    - it is easy to define new user profiles

49

49

CREATE ROLE

CREATE ROLE *RoleName*

- Definition of role privileges and user roles
  - instruction GRANT
- A user can have different roles at different times
  - dynamic association of a role with a user

    SET ROLE *RoleName*

50

50

Index management

Advanced Topics

51

51

Physical data organization

- In a relational DBMS the data are represented as collections of records memorized in one or more files
  - the physical organization of the data in a file influences the time required to access the information
  - each physical data organization makes some operations efficient and others cumbersome
- There is no physical data organization that is efficient for any type of data reading and writing

52

52

Indexes

- *Indexes* are the accessory physical structures provided by the relational DBMS to improve the efficiency of data access operations
  - indexes are realized using different types of physical structures
    - trees
    - hash tables
- The instructions for managing the indexes are not part of the standard SQL

53

53

Index definition in SQL

- SQL language provides the following instructions for defining indexes
  - to create an index
    - CREATE INDEX
  - to cancel an index
    - DROP INDEX
- The instructions for the management of indexes are not part of the standard SQL

DBG 54

54

CREATE INDEX

CREATE INDEX *NomeIndice*
ON *NomeTabella (ElencoAttributi)*

- The order in which the attributes appear in *AttributeList is important*
- the order of the index keys is
  - first on the basis of the first attribute in *AttributeList*
  - equal in value to the first attribute on the values of the second attribute
  - and so on, in order, until the last attribute

Use the minimum number of attributes, usually one

55

55

Example

- Creation of an index on the attribute Residence of the table EMPLOYEE

CREATE INDEX ResidenceIndex
ON EMPLOYEE (Residence)

- The index is jointly defined on the two attributes
- The index keys are ordered
  - first on the basis of the value of the attribute Surname
  - of equal value to the attribute Surname, on the value of the attribute Name

DBG 56

56

DROP INDEX

DROP INDEX *NomeIndice*

- Eliminate the index with the name *IndexName*
- This command is used when
  - the index is no longer utilized
  - the improvement in performance is insufficient
    - reduced reduction in response time for the queries
    - slowing down of updates due to index maintenance

57

57

Physical design

Advanced Topics

DBG 58

58

Physical design: input data

- Logical scheme of the database
- Characteristics of the chosen DBMS
  - physically available options
    - physical memory structures
    - indexes
- Data volumes
  - cardinality of tables
  - cardinality and distribution of the attribute values domain
- Estimate of application load
  - most important queries and their frequency
  - most important updating operations and their frequency
  - response time requirements for important queries/updates

DBG 59

59

Physical design: result

- Physical scheme of the database
  - physical organization of tables
  - indexes
- Memorization and operating parameters
  - Initial file sizes, expansion possibilities, free space at outset, ...

DBMG 60

60

Procedure

- Physical design is carried out empirically, using a trial and error approach
  - there are no reference methodologies
- Characterization of the application load
  - for each important query it is necessary to define
    - access relationships
    - attributes to be viewed
    - attributes involved in selections/joins
    - degree of selectivity of selection conditions
  - for each important update it is necessary to define
    - type of update (Insertion, cancellation, modification)
    - relation to any attributes involved
    - degree of selectivity of selection conditions

DBMG 61

61

Procedure: choices to be made

- Choices to be made
  - physical structuring of the files containing the tables
- choice of attributes to index
  - driven by estimating applicative load and data volume
  - definition of type for each index
  - e.g. hash or B-tree
- any variations of the scheme
  - horizontal partitioning in the secondary memory
  - denormalization of tables
    - used in data warehouses

DBMG 62

62

Tuning

- If the result is not satisfactory
  - *Tuning*, adding and removing indexes
- This is a procedure guided by the availability of tools that enable
  - verification of the execution plan adopted by the chosen DBMS
    - the execution plan defines the sequence of activities carried out by the DBMS to execute a query
      - data access methods
      - join methods
  - assess the execution cost of various alternatives

DBMG 63

63