Introduction to SQL

Alessandro Bozzon
TI1506: Web and Database Technology
ti1506-ewi@tudelft.nl

TU Delft

Strategy: Uniform

Announcements

- Midterm next week!
  - Be there on time
  - No material allowed

- Check quizzes
- Try “Exam Practice Lounge”

Course overview [DB]

1. Introduction to Database Systems
2. The Relational Model
3. **Introduction to SQL**
4. Advanced SQL Topics
5. Introduction to NoSQL database systems
6. Conceptual Data Modelling with ER Diagrams
7. Database Conceptual Design
8. Database Logical Design

At the end of this lecture, you should be able to …

- **Describe** and **design** SQL programs for the creation, altering, and manipulation of tables

- **Develop** logical database schema, with principled design that enforce data integrity

- **Prototype** and **deploy** database applications using open-source database systems (e.g., MySQL)

Join

Querying Multiple Tables

DB1

- All possible tuple combinations
- What if we want to retrieve

the name of all the **suppliers of product "P2"**

Products				
CodeP	**NameP**	**Color**	**Size**	**Storehouse**
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supply		
CodeS	**CodeP**	**Amount**
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

Supplier			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Cross Product

DB1

- All possible tuple combinations

Find the name of **all** the suppliers **of** product P2

```
SELECT NameS
FROM Supplier, Supply
```

Result

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

DB1

Simple `JOIN`

- `Supplier.CodeS = Supply.CodeS` is a **`JOIN` CONDITION**

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

Our Original Query

DB1

Find the name of **all** the suppliers **of** product P2

```
SELECT NameS
FROM Supplier, Supply
WHERE Supplier.CodeS = Supply.CodeS AND CodP = "P2"
```

Result

Supplier				Supply		
CodeS	**NameS**	**Shareholders**	**Office**	**CodeS**	**CodeP**	**Amount**
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

NameS
John
Victor
Anna

Another Query

DB1

Find the name of supplier of **at least one red product**

```
SELECT NameS
FROM Supplier, Supply, Products
WHERE Supplier.CodeS = Supply.CodeS AND Supply.CodeP = Product.CodeP
       AND Color = "Red"
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
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

NameS
John
Victor

If there are *N* tables in the **`FROM`** clause, at least *N* – *1* **`JOIN`** conditions in the **`WHERE`** clause

Q1

JOINs in SQL-2

- SQL-2 introduced an alternative syntax for the representation of JOINs, representing them explicitly in the from clause:

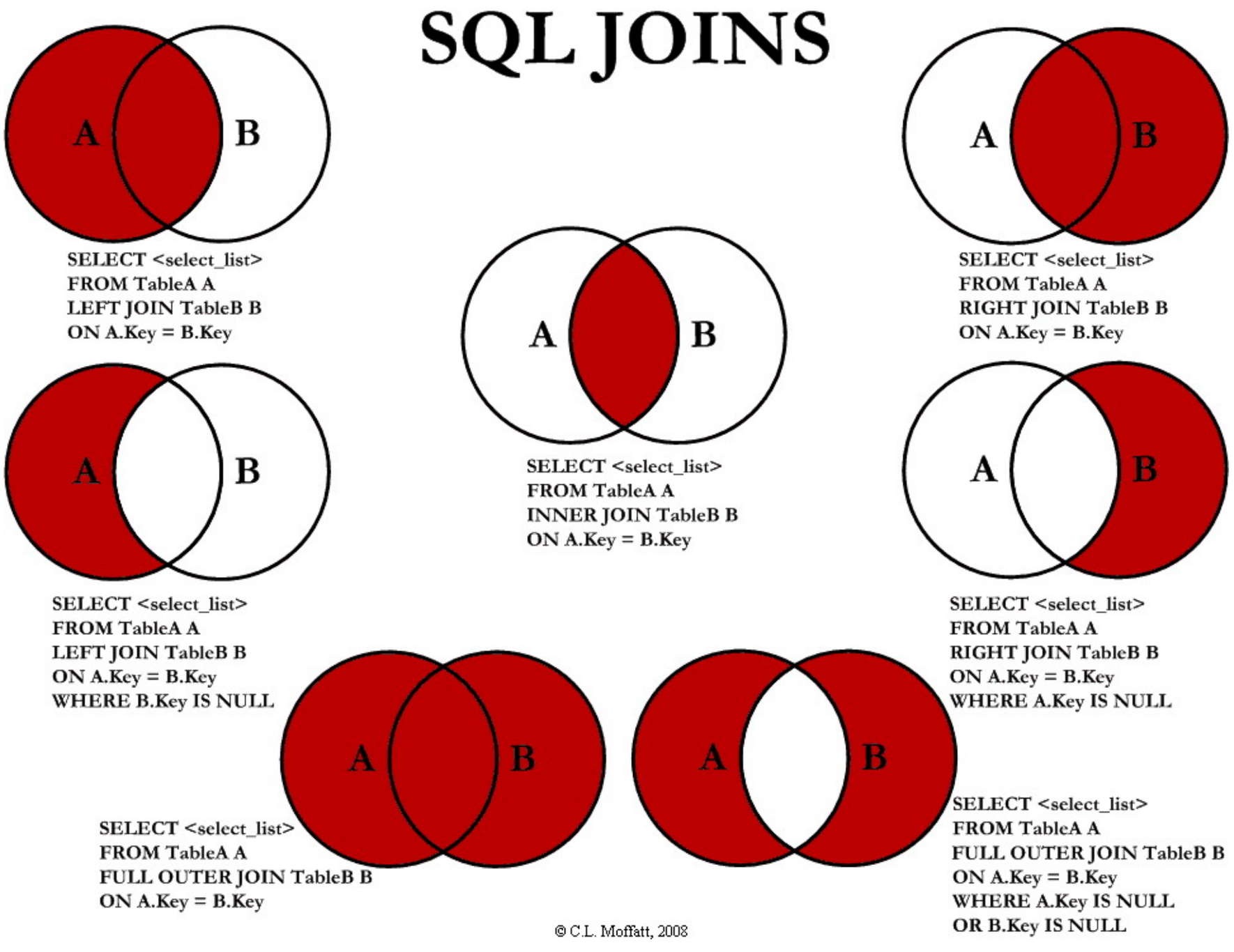
```
SELECT TargetList
FROM Table [[ AS ] Alias ]
    { [ JoinType] JOIN Table [[ AS ] Alias ] ON JoinConditions }
[ WHERE Condition ]
```

- JoinType can be any of **INNER**, **RIGHT [OUTER]**, **LEFT [OUTER]** or **FULL [OUTER]**, permitting the representation of outer joins
- The keyword **NATURAL** may precede JoinType (rarely implemented)

JOINs in SQL-2

- **Natural `JOIN`** on two relations *R* and *S*
  - No join condition specified
  - Implicit **`EQUIJOIN`** condition for each pair of attribute with same name from *R* and *S*
- **`INNER JOIN`**
  - Default type of join in a joined table (equivalent to **`JOIN`**)
  - Must specify **`JOIN`** attributes
  - Tuple is included in the results only if a matching tuple exists in the other relation
- **`LEFT OUTER JOIN`**
  - Every tuple in left table must appear in result
  - If no matching tuple: values for attributes in the right table set to **`NULL`**
- **`RIGHT OUTER JOIN`**
  - Every tuple in right table must appear in result
  - If no matching tuple: values for attributes in the left table set to **`NULL`**
- **`FULL OUTER JOIN`**
  - If no matching tuple: values for attributes in the left and/or right tables set to **`NULL`**

JOINs (Visually Explained)



https://www.codeproject.com/KB/database/Visual_SQL_Joins/Visual_SQL_JOINS_orig.jpg

DB1

INNER JOIN

Find the name of supplier of **at least one red product**

```
SELECT NameS
FROM Products INNER JOIN Supply ON Supply.CodeP = Products.CodeP
              INNER JOIN Supplier ON Supplier.CodeS = Supply.CodeS
WHERE Color = "Red"
```

Result

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
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

NameS
John
Victor

- Same results as in Slide 12

LEFT OUTER JOIN

DB1

Find 1) the *code* and *name* of *supplier*, and 2) the *code* of the supplied *products*, **showing also suppliers of no products**

```
SELECT Supply.CodeS, NameS, CodeP
FROM Supply LEFT OUTER JOIN Supplier ON Supplier.CodeS = Supply.CodeS
```

Result

CodeS	NameS	CodeP
S1	John	P1
S1	John	P2
S1	John	P3
S1	John	P4
S1	John	P5
S1	John	P6
S2	Victor	P1
S2	Victor	P2
S3	Anna	P2
S4	Angela	P3
S4	Angela	P4
S4	Angela	P5
S5	Paul	NULL

Q2

2:00

Which query return the same result set as the following query?

```
SELECT DISTINCT b.actor_id
FROM imdb.roles AS a,
       imdb.roles AS b,
       imdb.movies AS movies
WHERE a.movie_id = movies.id
  AND b.movie_id = movies.id
  AND a.actor_id <> b.actor_id
  AND a.actor_id = 393411
```

1

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id WHERE a.actor_id
<> b.actor_id AND a.actor_id = 393411
```

2

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id INNER JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id =
393411
```

3

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id INNER JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id <>
b.actor_id AND a.actor_id = 393411
```

4

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a LEFT JOIN
imdb.movies AS movies ON a.movie_id = movies.id RIGHT JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id <>
b.actor_id AND a.actor_id = 393411
```

2:00

Which query return the same result set as the following query?

```
SELECT DISTINCT b.actor_id
FROM imdb.roles AS a,
       imdb.roles AS b,
       imdb.movies AS movies
WHERE a.movie_id = movies.id
  AND b.movie_id = movies.id
  AND a.actor_id <> b.actor_id
  AND a.actor_id = 393411
```

1

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id WHERE a.actor_id
<> b.actor_id AND a.actor_id = 393411
```

2

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id INNER JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id =
393411
```

3

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a INNER JOIN
imdb.movies AS movies ON a.movie_id = movies.id INNER JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id <>
b.actor_id AND a.actor_id = 393411
```

4

```
SELECT DISTINCT b.actor_id FROM imdb.roles AS a LEFT JOIN
imdb.movies AS movies ON a.movie_id = movies.id RIGHT JOIN
imdb.roles AS b ON b.movie_id = movies.id WHERE a.actor_id <>
b.actor_id AND a.actor_id = 393411
```

Aggregate Queries

Aggregate Queries

- **Aggregate Query**: query in which the result depends on the consideration of **sets of rows**

- The result is a single (**aggregated**) value

- Expressed in the **`SELECT`** clause
  - aggregate operators are evaluated on the rows accepted by the **WHERE** conditions

- SQL-2 offers five aggregate operators
  - **`COUNT, SUM, MAX, MIN, AVG`**

Operator COUNT

- COUNT returns the number of rows or distinct values

```
COUNT (<*| [DISTINCT | ALL] AttributeList >)
```

- The DISTINCT keyword forces the count of distinct values in the attribute list

DB1

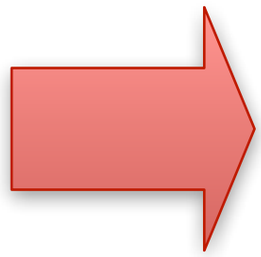
COUNT example / I

Find the number of suppliers in the database

```
SELECT COUNT(*)
FROM Supplier
```

Result

Supplier			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht



Result
5

COUNT example /2

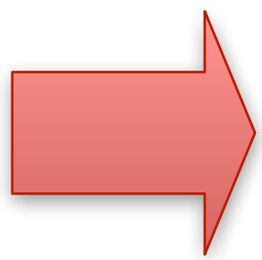
Find the number of suppliers with at least one supply

```
SELECT COUNT(*)
FROM Supply
```

Result

- Is it right?
- Equivalent to `SELECT COUNT(CodeP)` or `SELECT COUNT(CodeS)`

Supply		
CodeS	**CodeP**	**Amount**
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



Result
12

Q3

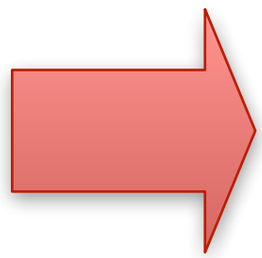
COUNT example /3

Find the number of suppliers with at least one supply

```
SELECT COUNT(DISTINCT CodeS)
FROM Supply
```

Result

Supply		
CodeS	**CodeP**	**Amount**
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



Result
4

DB1

COUNT example /4

Count the number of suppliers that supply the product P2

```
SELECT COUNT(*)
FROM Supply
WHERE CodeP = "P2"
```

Result

Supply		
CodeS	**CodeP**	**Amount**
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

Result
3

Operators `SUM,MAX,MIN,AVG`

- `SUM,MAX,MIN,AVG`
  - Allowed arguments are attributes or expressions

- `SUM,AVG`
  - Only numeric types

- `MAX,MIN`
  - Attribute must be sortable
  - Applied also on strings and timestamps

DB1

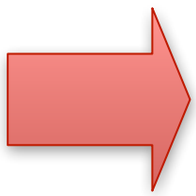
SUM example

Find the total number of supplied items for product P2

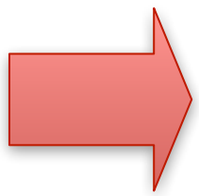
```
SELECT SUM(Amount)
FROM Supply
WHERE CodeP = "P2"
```

Result

Supply		
CodeS	**CodeP**	**Amount**
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



Supply		
CodeS	**CodeP**	**Amount**
S1	P2	200
S2	P2	400
S3	P2	200



Result
800

`NULL` Values and Aggregates

- All aggregate operations ignore tuples with **`NULL`** values on the aggregated attributes

Q4

Aggregate Query and Target List

DB2

```
SELECT  FirstName, Surname, MAX(Salary)
FROM Employee JOIN Department ON Dept = DeptName
WHERE Department.City = "London"
```

- This is an incorrect query, although syntactically admissible

- Whose name? The target list must be homogeneous

- The **`GROUP BY`** clause will help us

Grouping Rows

- Queries may apply aggregate operators to subsets of rows

For each product find the total amount of supplied items

```
SELECT CodeP, SUM(Amount)
FROM Supply
GROUP BY CodeP
```

Result

Supply		
CodeS	**CodeP**	**Amount**
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

Supply		
CodeS	**CodeP**	**Amount**
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

CodeP	**Amount**
P1	600
P2	800
P3	600
P4	500
P5	500
P6	100

GROUP BY clause / I

DB2

- The order of the grouping attributes does not matter
- The **SELECT** clause can contain
  - Attributes specified in the **GROUP BY** clause
  - Aggregated functions
  - Attributes **univocally** determined by attributes already specified in the **GROUP BY** clause

- Incorrect Query

```
SELECT Office
FROM Employee
GROUP BY Dept
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

DB2

GROUP BY clause /2

- The order of the grouping attributes does not matter
- The **SELECT** clause can contain
  - Attributes specified in the **GROUP BY** clause
  - Aggregated functions
  - Attributes **univocally** determined by attributes already specified in the **GROUP BY** clause

- Incorrect Query

```
SELECT DeptName, COUNT(*), D.City
FROM Employee E JOIN Department D ON
     E.Dept = D.DeptName
GROUP BY DeptName
```

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

DB2

GROUP BY clause /3

- The order of the grouping attributes does not matter
- The **SELECT** clause can contain
  - Attributes specified in the **GROUP BY** clause
  - Aggregated functions
  - Attributes **univocally** determined by attributes already specified in the GROUP BY clause

- **Correct Query**

```
SELECT DeptName, COUNT(*), D.City
FROM Employee E JOIN Department D ON
     E.Dept = D.DeptName
GROUP BY DeptName, D.City
```

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

Q5

GROUP BY example

- Queries may apply aggregate operators to subsets of rows

For each product find the total amount of supplied items

```
SELECT CodeP, SUM(Amount)
FROM Supply JOIN Supplier ON Supply.CodeS = Supplier.CodeS
WHERE Office = "Den Haag"
GROUP BY CodeP
```

Result

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

CodeP	Amount
P1	300
P1	300
P2	200
P2	400

CodeP	Amount
P1	600
P2	600

HAVING clause / I

DB2

- Conditions on the result of an aggregate operator require the **HAVING** clause
- Only predicates containing aggregate operators **should appear** in the argument of the **HAVING** clause

Find the departments in which the average salary of employees working in office number 20 is higher than 25

```
SELECT Dept
FROM Employee
WHERE Office = "20"
GROUP BY Dept
HAVING AVG(Salary) > 25
```

HAVING clause /2

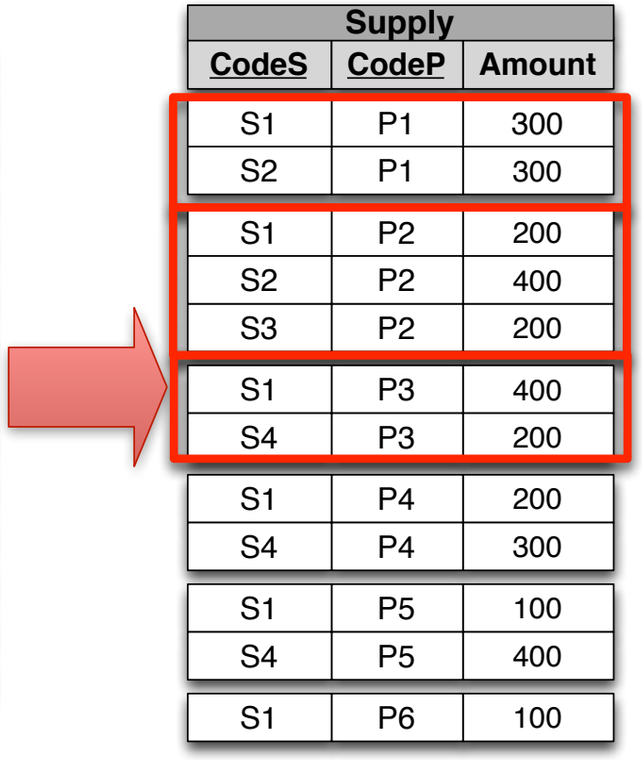
DB1

Find the total number of supplied items for products that count at least 600 total supplied items

```
SELECT CodeP, SUM(Amount)
FROM Supply
GROUP BY CodeP
HAVING SUM(Amount) >= 600
```

Result

Supply		
CodeS	**CodeP**	**Amount**
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



Supply		
CodeS	**CodeP**	**Amount**
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

CodeP	Amount
P1	600
P2	800
P3	600

HAVING clause /3

DB1

Find the code of red products supplied by more than one supplier

```
SELECT Supply.CodeP
FROM Supply JOIN Products ON Supply.CodeP = Product.CodeP
WHERE Color = "Red"
GROUP BY Supply.CodeP
HAVING COUNT(*) > 1
```

Result

Products					Supply		
CodeP	NameP	Color	Size	Storehouse	CodeS	CodeP	Amount
P1	Sweater	Red	40	Amsterdam	S1	P1	300
P1	Sweater	Red	40	Amsterdam	S2	P1	300
P2	Jeans	Green	48	Den Haag	S1	P2	200
P2	Jeans	Green	48	Den Haag	S2	P2	400
P2	Jeans	Green	48	Den Haag	S3	P2	200
P3	Shirt	Blu	48	Rotterdam	S1	P3	400
P3	Shirt	Blu	48	Rotterdam	S4	P3	200
P4	Shirt	Blu	44	Amsterdam	S1	P4	200
P4	Shirt	Blu	44	Amsterdam	S4	P4	300
P5	Skirt	Blu	40	Den Haag	S1	P5	100
P5	Skirt	Blu	40	Den Haag	S4	P5	400
P6	Coat	Red	42	Amsterdam	S1	P6	100

Result
P1

Q6

Schema Definition

Defining a Database Schema

- A database schema comprises:
  - declarations for the relations (“tables”) of the database
  - domains associated with each attribute
  - integrity constraints
- A schema has a *name* and an *owner* (the authorisation)
- Many other kinds of elements may also appear in the database schema, including:
  - privileges, views, indexes, triggers
- Syntax:

```
CREATE SCHEMA [ SchemaName ]
 [ [ authorization ] Authorization ]
 { SchemaElementDefinition }
```

Domains

- Specify the content of attributes

- Two categories
  - **Elementary** (predefined by the standard)
  - **User-defined** (not available in all RDBMS implementations)

```
CREATE DOMAIN Grade AS SMALLINT
       DEFAULT NULL
       CHECK (Grade >= 0 and Grade <= 10)
```

Elementary Domains

- **Bit**
  - Single boolean values or strings of boolean values (may be variable in length)
  - Syntax: `BIT [varying] [(Length)]`

- **Exact numeric domains**
  - Exact values, integer or with a fractional part
  - Four Alternatives
    - `NUMERIC [(Precision [, Scale ])]`: fixed point number, with user-specified precision of `Precision` digits, of which `Scale` digits to the right of decimal point.
    - `DECIMAL [(Precision [, Scale ])]`: functionally equivalent to `NUMERIC`
    - `INTEGER`: a finite subset of the integers that is machine-dependent
    - `SMALLINT`: a machine-dependent subset of the integer domain type

Elementary Domains

- **Approximate real values**
- Based on floating point representation
- **`FLOAT [(Precision)]`**: floating point number, with user-specified precision of at least **`n`** digits. By default **`n`** is 53, but it can be less
- **`REAL`**: floating point numbers, with machine-dependent precision
- **`Double Precision`**: double-precision floating point numbers, with machine-dependent precision

Elementary Domains

- Temporal Instants
  - **`DATE`**: format **`yyyy-mm-dd`**
  - **`TIME [(Precision)] [ with time zone ]`**: format **`hh:mm:ss:p`** with an optional decimal point and fractions of a second following.
  - **`TIMESTAMP [(Precision)] [ with time zone ]`**: format **`yyyy-mm-dd hh:mm:ss:p`**

- Temporal intervals **`INTERVAL FirstUnitOfTime [ TO LastUnitOfTime ]`**
  - Units of time are divided into two groups:
  - year, month
  - day, hour, minute, second

Table Definition

- An `SQL` table consists of
  - an ordered set of **attributes**
  - a (possibly empty) set of constraints

- **Statement `CREATE TABLE`**
  - defines a relation schema, creating an empty instance
- **Constraints**
  - integrity checks on attributes
- **OtherConstraints**
  - integrity constraints on the table

- Syntax:

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

Example of CREATE table

```
CREATE table Employee
 (
   RegNo      CHARACTER(6) PRIMARY KEY,
   FirstName CHARACTER(20) NOT NULL,
   Surname    CHARACTER(20) NOT NULL,
   Dept    CHARACTER (15)
           REFERENCES Department(DeptName)
           ON DELETE SET NULL
           ON UPDATE CASCADE,
   Salary DECIMAL (9) DEFAULT 0,
   City CHARACTER(15),
   UNIQUE(Surname,FirstName)
```

Q7

Default Domain Values

- Define the value that the attribute must assume *when a value is not specified during row insertion*
- Syntax:

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

- **`GenericValue`** represents a value compatible with the domain, in the form of a constant or an expression
- **`USER*`** is the login name of the user who issues the command

Constraints / 1

- Constraints are **conditions that must be verified by every database instance**
- Defined in the `CREATE` or `ALTER TABLE` operations
- Automatically verified by the DB after each operation

- Advantages
  - **declarative** specification of constraints
  - **unique centralised** verification
- Disadvantages
  - might slow down execution
  - pre-defined type of constraints
    - e.g. no constraint on aggregated data
    - but triggers can help

Constraints /2

- An operation that violates a constraints might cause two type of reactions:
  - the operation is **aborted**, causing an application error
  - a **compensation action** is taken, to reach a new consistent state

- Three type of constraints
  - Intra-relational constraints (or **table constraints**)
  - Inter-relational constraints (or **referential integrity constraints**)
  - *Generic* integrity constraints and **assertions**

Intra-relational Constraints

- Intra-relational constraints involve a single relation
  - **`NOT NULL`** (on single attributes)
    - upon tuple insertion, the attribute MUST be specified
  - **`UNIQUE`**: permits the definition of *candidate keys*
    - for single attributes: `UNIQUE`, after the domain
    - for multiple attributes: `UNIQUE( Attribute , Attribute )`
  - **`PRIMARY KEY`**: defines the primary key
    - once for each table
    - implies `NOT NULL`
    - syntax like `UNIQUE`

`PRIMARY KEY` vs. `UNIQUE`

- The SQL standard allows DBMS implementers to make their own distinctions between **`PRIMARY KEY`** and **`UNIQUE`**
  - Example: some DBMS might automatically create an index (data structure to speed search) in response to **`PRIMARY KEY`**, but not **`UNIQUE`**

- However, standard SQL requires these distinctions:
  - There can be **only one** **`PRIMARY KEY`** for a relation, but several **`UNIQUE`** attributes
  - No attribute of a **`PRIMARY KEY`** can ever be **`NULL`** in any tuple.
  - But attributes declared **`UNIQUE`** may have NULLs
    - and there may be several tuples with **`NULL`**!

Q8

Example of Intra-Relational Constraints

- Each pair of FirstName and Surname uniquely identifies each element

```
FirstName CHARACTER(20) NOT NULL,
Surname CHARACTER(20) NOT NULL
UNIQUE(FirstName,Surname)
```

- Note the difference with the following (stricter) definition

```
FirstName CHARACTER(20) NOT NULL UNIQUE,
Surname CHARACTER(20) NOT NULL UNIQUE
```

Inter-relational constraints

- Constraints may take into account several relations

- **`REFERENCES`** and **`FOREIGN KEY`** key permit the definition of referential integrity constraints. Syntax:
  - for single attributes: **`REFERENCES`**, after the domain
  - for multiple attributes: **`FOREIGN KEY (Attribute , Attribute)`** or **`REFERENCES`**

- **It is possible to associate reaction policies to violations of referential integrity**

Reaction Policies for Referential Integrity Constraints

- Reactions operate on the internal table, after changes to the external table
- Violations may be introduced
  - by **updates** on the referred attribute
  - by **row deletions**

- Reactions
  - **`CASCADE`**: propagate the change
  - **`SET NULL`**: nullify the referring attribute
  - **`SET DEFAULT`**: assign the default value to the referring attribute
  - **`NO ACTION`**: forbid the change on the external table

- Reactions may depend on the event; syntax:

```
ON < DELETE | UPDATE >
            < CASCADE | SET NULL | SET DEFAULT | NO ACTION >
```

Example of inter-relational constraint

```
CREATE table Employee
 (
   RegNo      CHARACTER(6) PRIMARY KEY,
   FirstName CHARACTER(20) NOT NULL,
   Surname    CHARACTER(20) NOT NULL,
   Dept     CHARACTER (15)
            REFERENCES Department(DeptName)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
   Salary NUMERIC (9) DEFAULT 0,
   City CHARACTER(15),
   UNIQUE(Surname,FirstName)
```

Schema Updates

- Two SQL statements:
  - **`ALTER`**: to modify a domain, the schema of a table, or a user
  - **`DROP`**: to remove schema, domain, table, etc.

```
ALTER TABLE Department ADD COLUMN NoOfOffices numeric(4)

DROP TABLE TempTable CASCADE
```

Relational Catalogues

- The catalog contains:
  - The data dictionary
  - The description of the data contained in the data base (tables, etc.)
  - Statistics about the data (distribution, access, growth)
- It is based on a relational structure (reflexive)

- It can be queried!
- The SQL-2 standard describes a **`Definition Schema`** (composed of tables) and an **`Information Schema`** (composed of views)

Q9

Data Manipulation

Data Modification in SQL

- Statements for:
  - insertion `INSERT`
  - deletion `DELETE`
  - change of attribute values `UPDATE`

- All the statements **can operate on a set of tuples** (set-oriented)
- In the condition it is possible to access other relations

Insertions / I

```
INSERT INTO TableName [ (AttributeList) ]
                  < values (ListOfValues) | SELECT SQL>
```

- Using Values

```
INSERT INTO Department(DeptName, City)
       VALUES ("Production","Toulouse")
```

- Using a subquery

```
INSERT INTO LondonProducts
      (SELECT Code, Description
       FROM Product
       WHERE ProdArea = "London")
```

Insertions /2

- The ordering of the attributes (if present) and of values is meaningful (first value with the first attribute, and so on)

- If *AttributeList* is omitted, all the relation attributes are considered, in the order in which they appear in the table definition

- If *AttributeList* does not contain all the relation attributes, to the remaining attributes it is assigned:
  - the default value (if defined)
  - the `NULL` value
  - `PRIMARY KEY`s might get special handling

Deletions / I

- The **DELETE** statement removes from the table all the tuples that satisfy the condition

```
DELETE FROM TableName
   [ WHERE Condition ]
```

- The removal may produce deletions from other tables if a referential integrity constraint with CASCADE policy has been defined
- If **WHERE** clause is omitted, **DELETE** removes all the tuples
- schema)

Deletions /2

- To remove all the tuples from **DEPARTMENT** keeping the table

```
DELETE FROM Department
```

- To remove table **DEPARTMENT** completely (content and schema):

```
DROP TABLE Department CASCADE
```

- Remove the Production department:

```
DELETE FROM Department
       WHERE DeptName = "Production"
```

- Remove the departments without employees

```
DELETE FROM Department
       WHERE DeptName NOT IN (SELECT Dept
                              FROM Employee)
```

Updates / I

```
UPDATE TableName
    SET Attribute = < Expression | SELECT SQL | NULL | default >
    {, Attribute = < Expression | SELECT SQL | NULL | default >}
    [ WHERE Condition ]
```

- Examples

```
UPDATE Employee
       SET Salary = Salary + 5
       WHERE RegNo = "M2047"
```

```
UPDATE Employee
       SET Salary = Salary * 1.1
       WHERE Dept = "Administration"
```

Updates /2

- Since the language is set oriented, the order of the statements is important

```
UPDATE Employee
   SET Salary = Salary * 1.1
   WHERE Salary <= 30

UPDATE Employee
   SET Salary = Salary * 1.15
   WHERE Salary > 30
```

- If the statements are issued in this order, some employees may get a double raise

Q10

Today we covered

- SQL as
  - a Retrieval Language
  - a Schema Creation and Modification Language
  - a Data Manipulation Language

Readings

- **Book**
  - *4.3: Basic Retrieval Queries in SQL*
  - *5.1.1: Comparison involving NULL and three-valued logic*
  - *5.1.5: Explicit Sets and Renaming of Attributes in SQL*
  - *5.1.6: Joined Tables in SQL and Outer Joins*
  - *5.1.7: Aggregated Functions in SQL*
  - *5.1.8: Grouping: the GROUP BY and HAVING Clauses*
  - *5.4: Schema Change Statements in SQL*

- **Suggested Readings on Blackboard**

End of Lecture