Business Intelligence

Introduction to databases

Lecture outline

- Database fundamentals
- Relational model and other data models
- Incomplete information
- Keys and constraints
- Query languages
- Examples

Warning

- If you already took a database course, this lecture will be very easy for you

Database fundamentals

Data and information

- Data:
  - elementary information unit
- Information:
  - processed data matching specific corporate needs

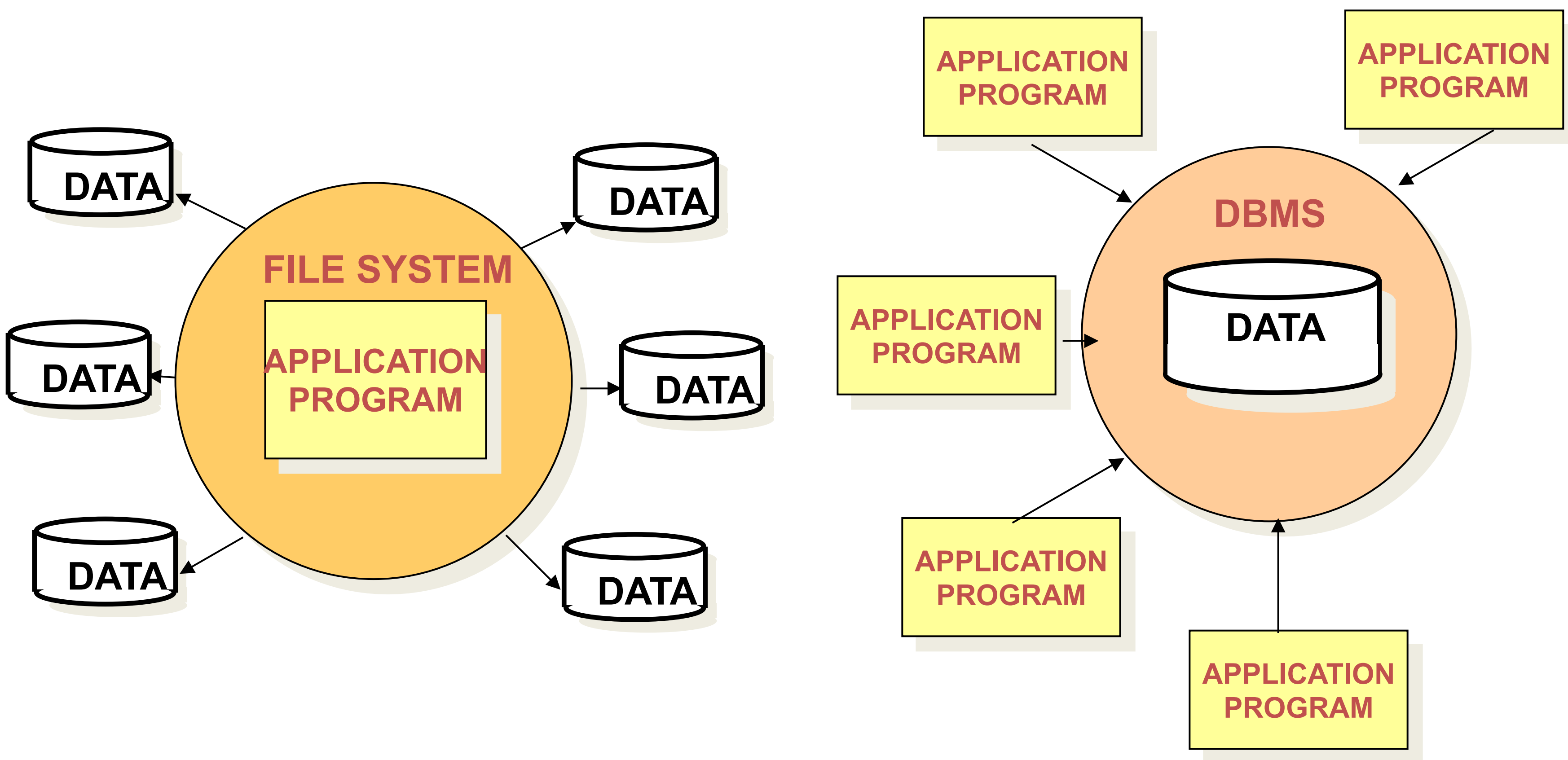
Data and information

- Data:

  < J. Smith,
  Business intelligence,
  spring semester >

- Information:
  - Who teaches BI? J. Smith
  - When does the course take place? In the spring semester

Comparing databases and file systems



Main features of DataBase Management Systems (DBMSs)

- Data sharing:
  - no replication in files
  - concurrency
- Data quality
  - integrity constraints
- Efficiency
  - loading, querying, sorting
- Access control
  - privacy
- Robustness

Data integration

- All data, independently of the applications that use them, appear only once
  - (useless redundancies are introduced by design)
  - memory waste is reduced
  - data integrity is improved (the same data cannot have two different values at the same time)

How to use a DBMS?

- Define the general structure of data
- Define the specific operations on data

schema

DATA STRUCTURE FOR BANK ACCOUNTS

John Smith’s bank account

Paul White’s bank account

……………………………………

instance

Example: university students

Student

STUDNO	NAME	CITY	DEPTCODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

The languages of a DBMS

- Data Definition Language (DDL)
  - examples: `CREATE`, `DROP`, `ALTER`
- Data Manipulation Language (DML)
  - examples: `SELECT`, `INSERT`, `UPDATE`, `DELETE`
- SQL (Structured Query Language) is a standardized language commonly used for both DDL and DML

DML: Query Language

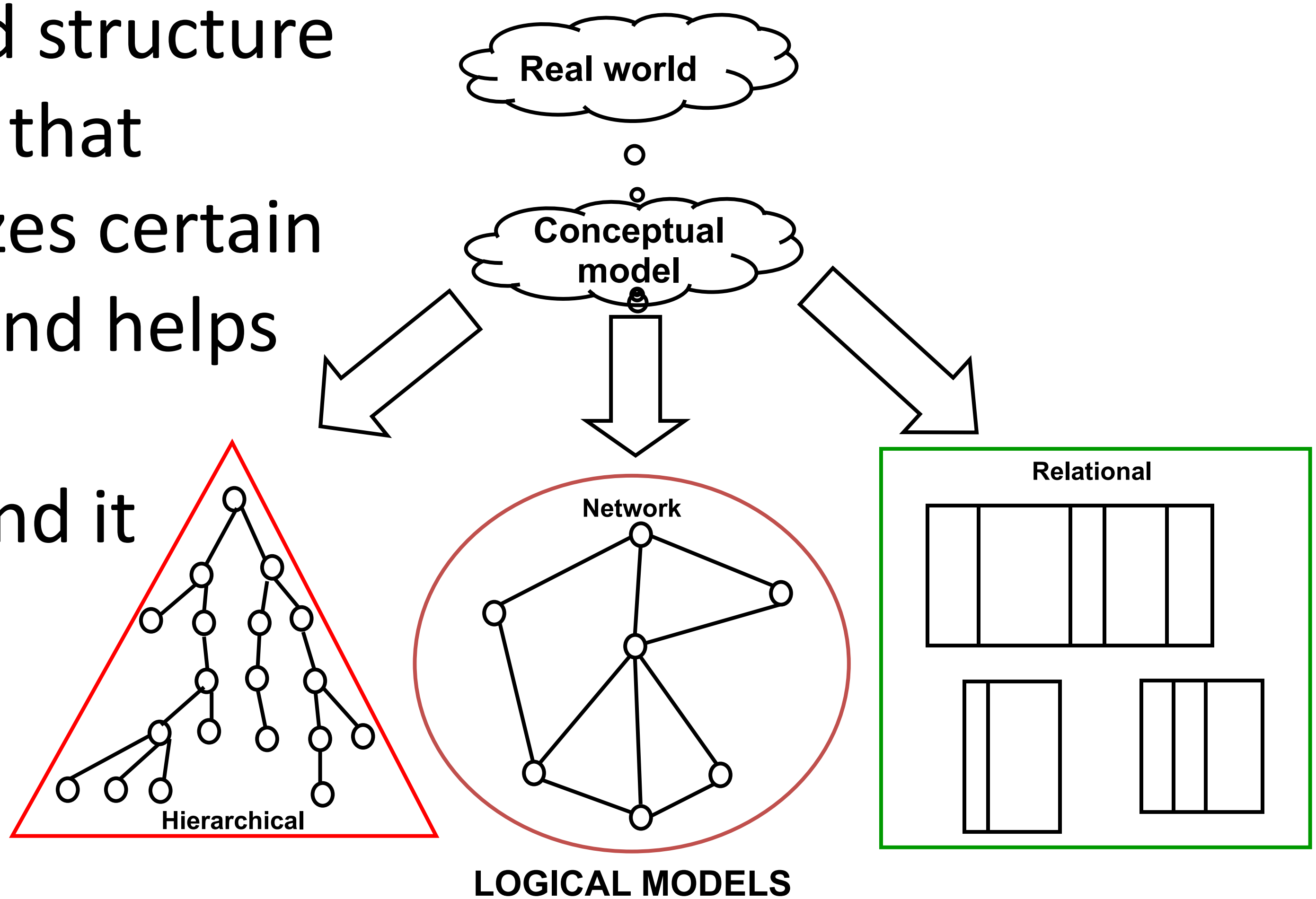
```
SELECT *
FROM Student
WHERE City = ‘Lugano’
```

STUDNO	NAME	CITY	DEPTCODE
123	Jack	Lugano	Inf

The relational model

Data models

- Models offer a simplified structure of reality that emphasizes certain aspects and helps to better understand it

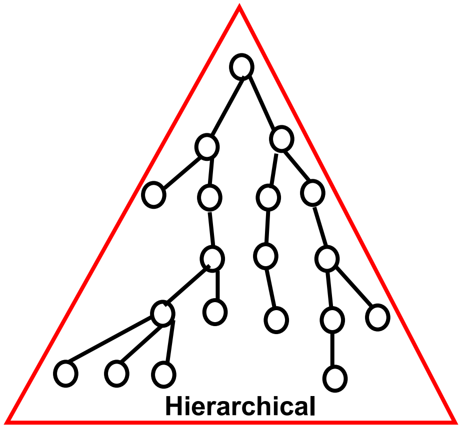


Timeline of models for data representation

- Hierarchical model (1960s)
- Network model (1970s)
- Relational model (1980s)
- Object-oriented model (1990s)
- XML model (2000s)
- NoSQL model (2010s)

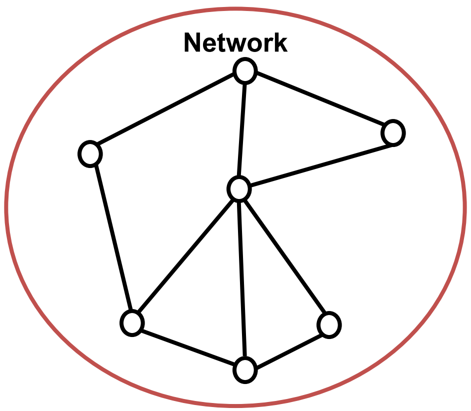
Logical data models

- Hierarchical
  - Data are represented as records
  - Relationships between data are represented with **pointers** in a tree structure



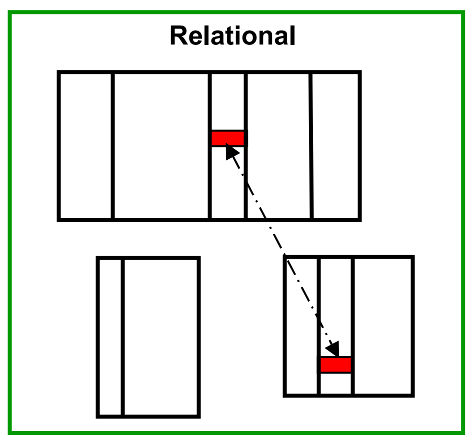
Logical data models

- Network (CODASYL consortium)
  - Data are represented as records
  - Relationships between data are represented with **pointers** in a complex graph structure



Logical data models

- Relational
  - Data are represented as tables
  - Relationships between data are obtained by associating attribute **values** in different tables



History of the relational model

- Invented by Edgar F. Codd in 1969, published 1970 (IBM Research)
  - E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6): 377-387 (1970)
- First projects: System R (IBM), Ingres (Berkeley University)
- Main technological findings: 1978-1980
- First commercial systems: start of 1980s (Oracle, IBM SQL/DS and DB2, Ingres, Informix, Sybase)
- Commercial success since 1985

Popularity of the models

Rank Sep 2023	Rank Aug 2023	Rank Sep 2022	DBMS	Database Model	Score Sep 2023	Score Aug 2023	Score Sep 2022
1.	1.	1.	Oracle	Relational, Multi-model	1240.88	-1.22	+2.62
2.	2.	2.	MySQL	Relational, Multi-model	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.75	+0.37	+0.29
5.	5.	5.	MongoDB	Document, Multi-model	439.42	+4.93	-50.21
6.	6.	6.	Redis	Key-value, Multi-model	163.68	+0.72	-17.79
7.	7.	7.	Elasticsearch	Search engine, Multi-model	138.98	-0.94	-12.46
8.	8.	8.	IBM Db2	Relational, Multi-model	136.72	-2.52	-14.67
9.	↑ 10.	↑ 10.	SQLite	Relational	129.20	-0.72	-9.62
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	128.56	-1.78	-11.47
11.	11.	↑ 13.	Snowflake	Relational	120.89	+0.27	+17.39
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model	110.06	+2.67	-9.06
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	100.45	+1.80	-9.70
14.	14.	14.	Splunk	Search engine	91.40	+2.42	-2.65
15.	↑ 16.	↑ 16.	Microsoft Azure SQL Database	Relational, Multi-model	82.73	+3.22	-1.69
16.	↓ 15.	↓ 15.	Amazon DynamoDB	Multi-model	80.91	-2.64	-6.51
17.	↑ 18.	↑ 20.	Databricks	Multi-model	75.18	+3.84	+19.56
18.	↓ 17.	↓ 17.	Hive	Relational	71.83	-1.52	-6.60
19.	19.	↓ 18.	Teradata	Relational, Multi-model	60.33	-0.98	-6.25
20.	20.	↑ 24.	Google BigQuery	Relational	56.46	+2.56	+6.34

Ranking from http://db-engines.com/en/ranking

Informal definition

column

schema

STUDNO	NAME	CITY	DEPTCODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco
234	Carl	Bern	Eco

instance

row

Relation and relationship

- In classical mathematics, a relation is a set-theoretic notion
- A relationship, a.k.a. association, indicates a correspondence between two entities in the model
  - We will see the Entity-Relationship model
- In the relational model, a relation has a slightly different meaning

Formal definition: domain and Cartesian product

- A domain D is any set of values
- Let $D_1, D_2, \ldots, D_n$ be *n* (possibly not distinct) domains
- The Cartesian product

  $$D_1 \times D_2 \times \ldots \times D_n$$

  is the set of all the ordered *n*-tuples

  $$<d_1, d_2, \ldots, d_n>$$

  such that $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$

Example

- $D_1 = \{a,b\}$
- $D_2 = \{1,2,3\}$
- $D_1 \times D_2 = \{<a,1>, <b,1>, <a,2>, <b,2>, <a,3>, <b,3>\}$

Formal definition: mathematical relation

- A mathematical relation over $D_1, D_2, \ldots, D_n$ is a subset of the Cartesian product

  $$D_1 \times D_2 \times \ldots \times D_n$$

- $D_1, D_2, \ldots, D_n$ are the domains of the relation
- A relation over *n* domains has degree *n*
- The number of *n*-tuples is called the cardinality of the relation
- In real applications, the cardinality is always finite

Example

- $D_1 = \{a,b\}$
- $D_2 = \{1,2,3\}$
- $D_1 \times D_2 = \{\langle a,1\rangle, \langle b,1\rangle, \langle a,2\rangle, \langle b,2\rangle, \langle a,3\rangle, \langle b,3\rangle\}$
- $R_1 = \{\langle a,1\rangle, \langle b,3\rangle\}$
- $R_2 = \{\langle a,1\rangle, \langle b,3\rangle, \langle a,2\rangle\}$
- $R_3 = \emptyset$ (the empty set)
- $R_4 = \{\langle a,1\rangle, \langle b,1\rangle, \langle a,2\rangle, \langle b,2\rangle, \langle a,3\rangle, \langle b,3\rangle\}$
- The degree of R_4 is 2; its cardinality is 6
  - How many relations are there over $D_1 \times D_2$?

Properties

- Degree of a relation:
  - number of domains (n)
- Cardinality of a relation:
  - number of tuples
- Attribute:
  - name given to a domain in a relation
  - in a relation, the attribute names cannot be repeated

Properties

- Based on the definitions, a mathematical relation is a set of ordered *n*-tuples
- Therefore, a relation is a set, and thus:
  - there is no ordering between the different *n*-tuples
  - the *n*-tuples of a relation are all distinct from one another
  - each single *n*-tuple is ordered: the *i*-th value of each tuple comes from the *i*-th domain, i.e., there is an ordering of domains

Example

$$Games \subseteq string \times string \times integer \times integer$$

- each domain has two distinct roles, depending on their position (it is a positional structure):
  - the first and the third position regard the name and goals of the home team
  - the second and fourth regard name and goals of the visiting team

FC Basel	FC Lugano	3	1
FC Lugano	Servette FC	2	0
FC Basel	AC Bellinzona	1	2
AC Bellinzona	Servette FC	0	1

Example under the relational model

- Each domain is associated with a name (attribute) describing its role in the relation
  - such a name is unique in the relation

Home	Visitor	HomeGoals	VisitorGoals
FC Basel	FC Lugano	3	1
FC Lugano	Servette FC	2	0
FC Basel	AC Bellinzona	1	2
AC Bellinzona	Servette FC	0	1

Example under the relational model

- Attributes are also column headers in a table
  - The order of attributes is immaterial (non-positional structure)
  - These are the same:

Home	HomeGoals	VisitorGoals	Visitor
FC Basel	3	1	FC Lugano
FC Lugano	2	0	Servette FC
FC Basel	1	2	AC Bellinzona
AC Bellinzona	0	1	Servette FC

Home	Visitor	HomeGoals	VisitorGoals
FC Basel	FC Lugano	3	1
FC Lugano	Servette FC	2	0
FC Basel	AC Bellinzona	1	2
AC Bellinzona	Servette FC	0	1

Formal definitions

- Let us associate each attribute with a domain, and let *dom*(A) indicate the domain associated with attribute A
- An *n*-tuple over a set X of *n* attributes is a function that, for each attribute A in X, maps A to a value of the domain *dom*(A)
- A relation over X is a set of *n*-tuples over X

Formal definition

- If t is an *n*-tuple over X and A∈X, then t[A] (or t.A) indicates the value of t over A
- In the example, if t is the first *n*-tuple of the table, then

  t[Visitor] = ‘FC Lugano’
- The same notation is also extended to sets (actually, sequences) of attributes.
  - It then denotes a tuple:

    t[Visitor,VisitorGoals] = <‘FC Lugano’, 1>
  - (We do not care to distinguish between a value and a tuple with just one value)

Tables and relations

- A table represents a relation if
  - The values of each column are from the same domain
  - Rows are different from one another
  - Column headers are different from one another
- Moreover, in a table representing a relation
  - the order of rows is irrelevant
  - the order of columns is irrelevant

Comparing the terminology

Formal definition	Informal definition
relation	table
attribute	column
tuple, n-tuple	row
domain	data type
cardinality	number of rows
degree	number of columns

- An important difference:
  - formal definition: no duplicates
  - informal definition: duplicates are possible

The relational model is value-based

- References between data in different relations are represented by means of values of the domains used in the *n*-tuples

Students

Studno	Last	First	DateOfBirth
1234	Black	Joe	12/12/1990
2345	White	John	11/11/1989
3456	Red	Paul	10/10/1991
4567	Green	Louise	08/08/1992

Exams

Student	Mark	Course
4567	A	01
4567	D	02
3456	B	04
1234	C	04

Courses

Code	Title	Teacher
01	Databases	Doe
02	Business Intelligence	Smith
04	Business Intelligence	Jones

Students

Studno	Last	First	DateOfBirth
1234	Black	Joe	12/12/1990
2345	White	John	11/11/1989
3456	Red	Paul	10/10/1991
4567	Green	Louise	08/08/1992

Exams

Student	Mark	Course
4567	A	01
4567	D	02
3456	B	04
1234	C	04

Courses

Code	Title	Teacher
01	Databases	Doe
02	Business Intelligence	Smith
04	Business Intelligence	Jones

Why values and not pointers?

- Independence of physical structures
- Only what is relevant from the user application viewpoint is represented
  - pointers are less understandable for the end user
- Data are more easily portable to different systems
- Pointers are directional
  - and may exist at the physical level

Cartesian product

Exam (E)

STUDNO	COURSECODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
702	2	7/9/13	C

Course (C)

COURSECODE	TITLE	TEACHER
1	BI	Doe
2	databases	Smith

E.STUDNO	E.COURSECODE	E.DATE	E.MARK	C.COURSECODE	C.TITLE	C.TEACHER
123	1	7/2/13	A	1	BI	Doe
123	2	8/1/13	B	1	BI	Doe
702	2	7/9/13	C	1	BI	Doe
123	1	7/2/13	A	2	databases	Smith
123	2	8/1/13	B	2	databases	Smith
702	2	7/9/13	C	2	databases	Smith

Cartesian product

Exam

STUDNO	COURSECODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
702	2	7/9/13	C

Course

COURSECODE	TITLE	TEACHER
1	BI	Doe
2	databases	Smith

SELECT * FROM Exam E, Course C

E.STUDNO	E.COURSECODE	E.DATE	E.MARK	C.COURSECODE	C.TITLE	C.TEACHER
123	1	7/2/13	A	1	BI	Doe
123	2	8/1/13	B	1	BI	Doe
702	2	7/9/13	C	1	BI	Doe
123	1	7/2/13	A	2	databases	Smith
123	2	8/1/13	B	2	databases	Smith
702	2	7/9/13	C	2	databases	Smith

Queries

- Which professors have examined Jack?

Student

STUDNO	NAME	CITY	DEPTCODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Exam

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
702	2	7/9/13	C

Course

COURSE CODE	TITLE	TEACHER
1	BI	Doe
2	databases	Smith

SQL queries

- SQL queries have the typical `select-from-where` structure
- Syntax:
  `select` *AttrExpr* {, *AttrExpr*}
  `from` *Table* {, *Table*}
  [`where` *Condition*]
- The three parts of the query are called:
  - `select` clause with the target list
  - `from` clause
  - `where` clause
- The query
  - makes the "cartesian product" of the tables in the `from` clause
  - considers only the rows satisfying the condition in the `where` clause
  - for each row, evaluates the expression in the `select` clause
- Complete syntax:
  `select` *AttrExpr* [[`as`] *Alias*] {, *AttrExpr* [[`as`] *Alias*] }
  `from` *Table* [[`as`] *Alias*] {, *Table* [[`as`] *Alias*] }
  [`where` *Condition*]

SQL query

- Which professors have examined Jack?

```
SELECT Teacher
FROM Course C, Student S, Exam E
WHERE Name = 'Jack'
AND S.StudNo = E.StudNo
AND E.CourseCode = C.CourseCode
```

SQL query

- Which professors have examined Jack?

Cartesian product

```
SELECT Teacher
FROM Course C, Student S, Exam E
WHERE Name = 'Jack'
AND S.StudNo = E.StudNo
AND E.CourseCode = C.CourseCode
```

Cartesian product

S.STUDNO	S.NAME	S.CITY	S.DEPTCODE	E.STUDNO	E.COURSECODE	E.DATE	E.MARK	C.COURSECODE	C.TITLE	C.TEACHER
123	Jack	Lugano	Inf	123	1	07/02/13	A	1	BI	Doe
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	1	BI	Doe
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	1	BI	Doe
123	Jack	Lugano	Inf	123	2	08/01/13	B	1	BI	Doe
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	1	BI	Doe
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	1	BI	Doe
123	Jack	Lugano	Inf	702	2	07/09/13	C	1	BI	Doe
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	1	BI	Doe
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	1	BI	Doe
123	Jack	Lugano	Inf	123	1	07/02/13	A	2	databases	Smith
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	2	databases	Smith
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	2	databases	Smith
123	Jack	Lugano	Inf	123	2	08/01/13	B	2	databases	Smith
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	2	databases	Smith
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	2	databases	Smith
123	Jack	Lugano	Inf	702	2	07/09/13	C	2	databases	Smith
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	2	databases	Smith
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	2	databases	Smith

SQL query

- Which professors have examined Jack?

```
SELECT Teacher
FROM Course C, Student S, Exam E
WHERE Name = 'Jack'
AND S.StudNo = E.StudNo
AND E.CourseCode = C.CourseCode
```

Cartesian product

+

equality of values across tables

SQL query

- Which professors have examined Jack?

```
SELECT Teacher
FROM Course C, Student S, Exam E
WHERE Name = 'Jack'
AND S.StudNo = E.StudNo
AND E.CourseCode = C.CourseCode
```

Cartesian product

+

equality of values across tables

The two things together give rise to what is called a **join** in DB terms
Joins have an alternative special syntax in SQL, which we won't see in this course

Applying value equalities

S.STUDNO	S.NAME	S.CITY	S.DEPTCODE	E.STUDNO	E.COURSECODE	E.DATE	E.MARK	C.COURSECODE	C.TITLE	C.TEACHER
123	Jack	Lugano	Inf	123	1	07/02/13	A	1	BI	Doe
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	1	BI	Doe
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	1	BI	Doe
123	Jack	Lugano	Inf	123	2	08/01/13	B	1	BI	Doe
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	1	BI	Doe
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	1	BI	Doe
123	Jack	Lugano	Inf	702	2	07/09/13	C	1	BI	Doe
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	1	BI	Doe
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	1	BI	Doe
123	Jack	Lugano	Inf	123	1	07/02/13	A	2	databases	Smith
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	2	databases	Smith
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	2	databases	Smith
123	Jack	Lugano	Inf	123	2	08/01/13	B	2	databases	Smith
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	2	databases	Smith
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	2	databases	Smith
123	Jack	Lugano	Inf	702	2	07/09/13	C	2	databases	Smith
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	2	databases	Smith
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	2	databases	Smith

Selecting columns

S.STUDNO	S.NAME	S.CITY	S.DEPTCODE	E.STUDNO	E.COURSECODE	E.DATE	E.MARK	C.COURSECODE	C.TITLE	C.TEACHER
123	Jack	Lugano	Inf	123	1	07/02/13	A	1	BI	Doe
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	1	BI	Doe
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	1	BI	Doe
123	Jack	Lugano	Inf	123	2	08/01/13	B	1	BI	Doe
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	1	BI	Doe
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	1	BI	Doe
123	Jack	Lugano	Inf	702	2	07/09/13	C	1	BI	Doe
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	1	BI	Doe
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	1	BI	Doe
123	Jack	Lugano	Inf	123	1	07/02/13	A	2	databases	Smith
456	Paula	Mendrisio	Inf	123	1	07/02/13	A	2	databases	Smith
789	Peter	Bellinzona	Eco	123	1	07/02/13	A	2	databases	Smith
123	Jack	Lugano	Inf	123	2	08/01/13	B	2	databases	Smith
456	Paula	Mendrisio	Inf	123	2	08/01/13	B	2	databases	Smith
789	Peter	Bellinzona	Eco	123	2	08/01/13	B	2	databases	Smith
123	Jack	Lugano	Inf	702	2	07/09/13	C	2	databases	Smith
456	Paula	Mendrisio	Inf	702	2	07/09/13	C	2	databases	Smith
789	Peter	Bellinzona	Eco	702	2	07/09/13	C	2	databases	Smith

Final result

C.TEACHER
Doe
Smith

Queries

- Which students got an A in BI?

Student

STUDNO	NAME	CITY	DEPTCODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Exam

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
702	2	7/9/13	C

Course

COURSECODE	TITLE	TEACHER
1	BI	Doe
2	databases	Smith

SQL query

- Which students got an A in BI?

```
SELECT Name
FROM Course C, Student S, Exam E
WHERE C.Title = 'BI'
AND E.Mark = 'A'
AND S.StudNo = E.StudNo
AND E.CourseCode = C.CourseCode
```

Tables occurring more than once

- Table aliases are useful
  - For readability
  - For those cases where the same table occurs more than once
- Aliases are like variables in programming languages

Who are Bo’s employees?

Employee

EMPNO	NAME	SALARY	MGRNO
1	Al	100 K$	2
2	Bo	200 K$	NULL
3	Carl	150 K$	2

```
SELECT X.Name
FROM Employee X, Employee Y
WHERE X.MgrNo = Y.EmpNo
AND Y.Name = ‘Bo’
```

X.Name
Al
Carl

Selection and projection

- The operation indicated by the `SELECT` clause is called a **projection**
  - It restricts the tuples on those attributes indicated after the `SELECT` keyword
  - It keeps them all if `SELECT` is followed by ‘*’
- The condition indicated in the `WHERE` clause performs a so-called **selection**:
  - Only the rows satisfying the condition are retained
- It’s a bit counterintuitive, but:
  - `SELECT` → projection
  - `WHERE` → selection
  - `FROM` → cartesian product

Observations

- Difference between schema and instance
- Quite different activities:
  - schema design
  - instance management
- Moving from data to information (query language)

Example

- Two instances of invoices

"Chez Philippe" Via Elvezia 23 9100 Somewhere		
Invoice no. 2369 12/5/2012		
3	cover charge	3.15
2	starters	6.22
3	entrée	12.60
2	steaks	19.00
	Total	41.98

"Chez Philippe" Via Elvezia 23 9100 Somewhere		
Invoice no. 2456 16/5/2012		
2	cover charge	2.10
1	starter	3.11
2	entrée	8.40
2	fish	25.5
2	coffee	1.60
	Total	<u>39.41</u>

Relational representation, 1

Invoice

Number	Date	Total
2369	12/5/2012	41.98
2456	16/5/2012	39.41

Detail

Number	Quantity	Description	Amount
2369	3	cover charge	3.15
2369	2	starters	6.22
2369	3	entrée	12.60
2369	2	steak	19.00
2456	2	cover charge	2.10
2456	1	starters	3.11
2456	2	entrée	8.40
2456	2	fish	25.50
2456	2	coffee	1.60

Relational representation, 2

Invoice

Number	Date	Total
2369	12/5/2012	41.98
2456	16/5/2012	39.41

Detail

Number	Row	Quantity	Description	Amount
2369	1	3	cover charge	3.15
2369	2	2	starters	6.22
2369	3	3	entrée	12.60
2369	4	2	steak	19.00
2456	1	2	cover charge	2.10
2456	2	1	starters	3.11
2456	3	2	entrée	8.40
2456	4	2	fish	25.50
2456	5	2	coffee	1.60

Incomplete information

Incomplete information

- The relational model imposes a rigid data structure:
  - information represented by n-tuples
  - only some n-tuples formats are allowed:
    - those that match the relation schema
- Available data may not always match the required format exactly, for several reasons

Incomplete information

Driver	License_no
Alice	A123456
Jim	
Bob	
Dave	

- Jim has a license but we do not know its number
- Bob does not have a license
- We do not know whether Dave has a license or not

Incomplete information

- Albeit a rather common practice, it is better not to use values of the domain (such as 0, “”, etc.) for representing incomplete information:
  - maybe there are no “unused” values in the domain
  - some “unused” values might become used later
- Such values require special care each time they are encountered

Incomplete information

- A rough but effective technique:
  - a null value (NULL) denotes the absence of a value of the domain (NULL is not part of the domain)
  - Formally, extending the notion of n-tuple is sufficient:

    t[A] is a value of dom(A) for every attribute A

    - or it is the null value NULL
  - Restrictions on the presence of null values are needed

Incomplete information

Student

STUDNO	NAME	CITY	DEPT-CODE
123	Jack	Lugano	NULL
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Exam

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
NULL	2	8/1/13	NULL
702	2	7/9/13	C

Course

COURSE CODE	TITLE	TEACHER
1	NULL	Doe
2	databases	Smith

NULL

- Three different cases of incomplete information:
  - value is unknown: a value of the domain applies, but we do not know which one
  - value is non-existent: no value of the domain applies
  - no information: it is not known whether a value of the domain applies or not
  - DBMSs do not distinguish between these kinds of null values, and therefore they implicitly adopt the “no information” semantics

Keys and constraints

Integrity constraints

- There are database instances that are syntactically correct but that do not represent feasible information for the application

Exams

Student	Grade_perc	Course
123	92	01
123	106	02
234	67	02
345	85	03

Integrity constraints

- Aim: excluding those instances that do not correspond to meaningful information in the application being represented
  - constraints on null values
  - key constraints
  - referential integrity
  - generic constraints

Integrity constraints

- Definition of integrity constraint
  – property that must always be kept satisfied by every instance of the relations
  – constraints can be regarded as logical formulas that map every database instance to either true or false

Integrity constraints

- Types of constraints:
  - intra-relation constraints.
    - domain constraints (i.e., constraints on values)
      - constraints on null values
    - tuple constraints
    - …
  - inter-relation constraints
    - referential integrity constraints
    - …

Integrity constraints

- Useful for describing the world of interest in a more accurate way than just by the schema structure
- Data quality
- Useful design tool
- May be used by the system for query optimization

Tuple constraints

- A condition on the values of each single n-tuple, independently of the other n-tuples
- Possible syntax: Boolean expression (with AND, OR, NOT) of atoms comparing attribute values or arithmetic expressions thereof
- Example (course 1 has a higher required grade):

    (Grade_perc >= 80) OR (course != 1)

- A tuple constraint is also a domain constraint if it only regards one attribute
    - Example:

      (Grade_perc >= 60) AND (Grade_perc <= 100)

Keys

- A key of a schema is a subset of the schema attributes that is unique and minimal
- Unique: no two tuples have the same key value
- Minimal: by removing any attribute from the key, uniqueness is lost
- A subset of attributes that is unique (but not necessarily minimal) is called a superkey

Keys in the example

(underlined in red)

Student

STUDNO	NAME	CITY	DEPT-CODE

Exam

STUDNO	COURSECODE	DATE	MARK

Course

COURSECODE	TITLE	TEACHER

Schemas with multiple keys

- One of them is called primary key
- The other ones are the alternate keys

CLIENT
(CLIENT_CODE, ADDRESS, SSN)

Primary key: CLIENT_CODE
Alternate key: SSN

Existence of keys

- Relations are sets, therefore each relation may contain the same tuple only once
  - the set of all attributes of a relation schema is always a superkey of the relation
- Since the set of attributes is finite, every relation schema always has (at least) one key

Importance of keys

- Existence of keys guarantees accessibility of all data in the database
- Every single value is univocally accessible through:
  - the name of the relation
  - the value of the key
  - the name of the attribute
- Keys are the main means to connect data in different relations
  - “the relational model is value-based”

Keys and null values

- In the presence of null values for the attributes forming the key
  - the identification of the corresponding n-tuple is not possible
  - references to other relations are also affected
- The presence of null values in keys must be limited
- Practical solution: for every relation we choose a primary key for which null values are not allowed

Foreign keys

- Pieces of information in different relations are connected by means of common values
  - In particular, (primary) key values
- A referential integrity constraint between relations R_1 and R_2 over attributes X imposes that the values over X of every n-tuple of the instance of R_1 also occur as values of the (primary) key of the instance of R_2

Foreign keys in the example

(green arrows)

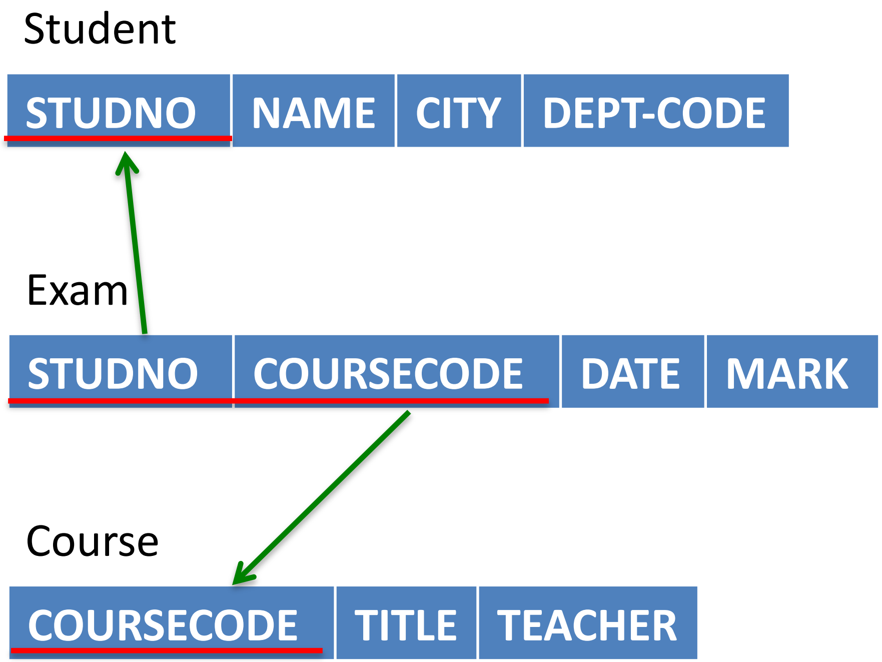


Table definition

- A table consists of
  - An ordered set of attributes
  - A (possibly empty) set of constraints
- SQL's `create table` command defines a relation's schema and creates an empty instance

```
create table Student
(  StudNo     char(6) primary key,
   Name       varchar(30) not null,
   City       varchar(20),
   Dept-code  char(3) )
```

Intra-relation constraints

- Constraints are conditions that must be satisfied by every instance of the database
- Intra-relation constraints regard a single relation (two cases: tuple level or table level)
  - `not null` (on a single attribute; tuple level)
  - `primary key`: defines the primary key (once per table; entails `not null`)
  - `unique`: allows the definition of alternate keys (table level)
  - `check`: can represent several kinds of constraint

Referential integrity

- It’s a hierarchical (parent-child) relationship between tables
- Some attributes of the child table are defined as a *foreign key*
- The values contained in the foreign key must always be present in the parent table

```
create table Exam
(   StudNo       char(6) not null,
    Coursecode   char(3) not null,
    Date         date,
    Mark         ENUM('A','B','C','D','E','F'),
    primary key (StudNo,Coursecode),
    foreign key (StudNo) references Student(StudNo),
    foreign key (Coursecode) references Course(Coursecode)
)
```

An incorrect instance

Student

STUDNO	NAME	CITY	DEPT-CODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Course

COURSECODE	TITLE	TEACHER
1	BI	Doe
2	databases	Smith

Exam

STUDNO	COURSE CODE	DATE	MARK	
123	1	7/2/13	A	
123	2	8/1/13	B	
789	2	7/9/13	C	
~~123~~	~~2~~	~~1/8/14~~	~~A~~	Violates `primary key`
~~702~~	~~NULL~~	~~1/2/15~~	~~NULL~~	Violates `not null`
~~555~~	~~1~~	~~3/4/16~~	~~B~~	Violates `foreign key`

Managing orphans

Student

STUDNO	NAME	CITY	DEPT-CODE
123	Jack	Lugano	Inf
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Exam

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
789	2	7/9/13	C

Managing orphans

Student

STUDNO	NAME	CITY	DEPT-CODE
~~123~~	~~Jack~~	~~Lugano~~	~~Inf~~
456	Paula	Mendrisio	Inf
789	Peter	Bellinzona	Eco

Exam

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
789	2	7/9/13	C

Orphan: tuple without parent because of deletions or updates in the parent table

Reacting to change

- Deleting/updating a parent's tuple may cause a violation of referential integrity
- Possible reactions:
  - `cascade`: propagates the change
  - `set null`: the foreign key is set to null
  - `set default`: the reference is assigned a default value
  - `no action`: disallows the operation
- Syntax:

```
on < delete|update >
   <cascade|set null|set default|no
action>
```

Reacting to change: deletion

What happens to exams if a student is deleted?

- `cascade`
  the exams of the deleted student are deleted, too
- `set null`
  the StudNo of the the exams of the deleted student are set to null
- `set default`
  the StudNo of the the exams of the deleted student are set to a default value
- `no action`
  student deletion is disallowed if there are exams for that StudNo

Reacting to change: update

What happens to exams if a student's StudNo is changed?

- `cascade`
  the StudNo of the exams of the updated student are updated, too
- `set null`
  the StudNo of the the exams of the updated student are set to null
- `set default`
  the StudNo of the the exams of the updated student are set a default value
- `no action`
  student update is disallowed if there are exams for that StudNo

Example syntax

```
create table Exam
( ....
    primary key(StudNo,CourseCode)
    foreign key (StudNo)
      references Student(StudNo)
        on delete cascade
        on update cascade
    foreign key (CourseCode)
      references Course(CourseCode)
        on delete no action
        on update no action )
```

Ordering

- It’s useful to sort results by relevance
- SQL has an `order by` clause
- Syntax:

  `order by` *Attr* [`asc` | `desc`] {, *Attr* [`asc` | `desc`] }

- The sorting conditions are evaluated one after the other
  – If there is a tie on the value of the first `order by` attribute, the second one is considered, and so on

Sorting the result

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
123	2	8/1/13	B
702	2	7/9/13	C
555	1	3/4/16	B

```
Select * from exam
order by mark, date desc
```

STUDNO	COURSE CODE	DATE	MARK
123	1	7/2/13	A
555	1	3/4/16	B
123	2	8/1/13	B
702	2	7/9/13	C

Aggregates

- Sometimes it's useful to extract statistical information from a set of values
- The easiest form is given by aggregate functions, which apply to a group of rows
  - `count` (cardinality)
  - `sum`
  - `max`
  - `min`
  - `avg` (average)

The `count` operator

- `count` gives the number of rows or of distinct values; syntax:

  `count`(< `*` | [`distinct` | `all`] *AttrList* >)

- Extract the number of students:

```
select count(*)
from Student
```

- Extract the number of distinct values of attribute `Name` among all rows of `Student`:

```
select count(distinct Name)
from Student
```

`sum`, `max`, `min`, `avg`

- Syntax:
  < `sum` | `max` | `min` | `avg` > ([`distinct` | `all`] *AttrExpr*)

- The `distinct` option considers each value only once
  - Useful only for `sum` and `avg`

- The `all` option is the default and considers all values different from *null*

A query with `sum`

```
select sum(amount) as tot
from Order
where ClientNo=2
```

Order

ORDNO	CLIENTNO	DATE	AMOUNT
1	1	7/2/12	50,000,000
3	2	8/1/13	12,000,000
5	2	7/9/13	1,500,000
4	3	8/8/16	8,000,000
6	3	9/9/17	1,500,000
2	4	1/1/18	5,500,000

Tot
13,500,000

Grouping

- Aggregates can be applied to a subset of the rows of a table thanks to the `group by` clause
- The groups themselves may be filtered via the `having` clause
- Example: extract the sum of amounts of orders placed after 8/8/12 for every client with at least 2 orders

```
select ClientNo, sum(Amount)
from Order
where Date > 8/8/12
group by ClientNo
having count(*) >= 2
```

Step 1: selection

- Evaluating the `where` clause

ORDNO	CLIENTNO	DATE	AMOUNT
~~1~~	~~1~~	~~7/2/12~~	~~50,000,000~~
3	2	8/1/13	12,000,000
5	2	7/9/13	1,500,000
4	3	8/8/16	8,000,000
6	3	9/9/17	1,500,000
2	4	1/1/18	5,500,000

Step 2: grouping

- Evaluating the `group by` clause

ORDNO	CLIENTNO	DATE	AMOUNT
3	2	8/1/13	12,000,000
5	2	7/9/13	1,500,000
4	3	8/8/16	8,000,000
6	3	9/9/17	1,500,000
2	4	1/1/18	5,500,000

Step 3: computing aggregates

- Computing `sum(amount)` and `count(*)` for each group

CLIENTNO	SUM(AMOUNT)	COUNT(*)
2	13,500,000	2
3	9,500,000	2
4	5,500,000	1

Step 4: group extraction

- Evaluating `having count(*) >= 2`

CLIENTNO	SUM(AMOUNT)	COUNT(*)
2	13,500,000	2
3	9,500,000	2
~~4~~	~~5,500,000~~	~~1~~

Step 5: generating result

- Evaluating `select` clause

CLIENTNO	SUM(AMOUNT)
2	13,500,000
3	9,500,000

Coherence between `group by` and target list

- Incorrect:

```
select Mark
from Exam
group by StudNo
```

Which mark? For which exam in the group?

- Incorrect:

```
select E.CourseCode, count(*), C.Teacher
from Exam E, Course C
where E.CourseCode= C.CourseCode
group by E.CourseCode
```

Here the teacher is univocally determined by the CourseCode (because of the foreign key), but this might not have been the case

- Correct:

```
select E.CourseCode, count(*), C.Teacher
from Exam E, Course C
where E.CourseCode= C.CourseCode
group by E.CourseCode, C.Teacher
```

Multiple grouping

- Extract the sum of quantities of details of orders placed by each client on each product, provided that the sum is above 50

Order	ORDNO	CLIENTNO	DATE	AMOUNT
Detail	ORDNO	PRODNO	QTY	

```
select ClientNo, ProdNo, sum(Qty)
from Order as O, Detail as D
Where O.OrdNo = D.OrdNo
group by ClientNo, ProdNo
having sum(Qty) > 50
```

A possible result after join and grouping

- Extract the sum of quantities of details of orders placed by each client, provided that the sum is above 50

O.ORDNO	CLIENTNO	D.ORDNO	PRODNO	QTY	
3	1	3	1	30	1,1 group
4	1	4	1	20	
3	1	3	2	30	1,2 group
5	1	5	2	10	
3	2	3	1	60	2,1 group
1	3	1	1	40	3,1 group
2	3	2	1	30	
6	3	6	1	25	

Final result

- Computing `sum(Qty)` for the groups and evaluating the `having` clause

CLIENTNO	PRODNO	SUM(QTY)
~~1~~	~~1~~	~~50~~
~~1~~	~~2~~	~~40~~
2	1	60
3	1	95