Introduction to Relational Databases

- Bachelor Computer Science, Lille 1 University
- Sept 21st, 2011 (lecture 4/12)
- Today's lecturer: C. Kuttler
- Prologue:
  - example for table definition with constraints
- Topic: Introduction to SQL as a query language
  - basic queries
  - inserting data into tables

Example: contract management

Customer

Cus_ID	ADDRESS	TAX_ID

Contract

Con_ID	Cus_ID	DATE	AMOUNT

Detail

Con_ID	Prod_ID	Qt

Product

Prod_ID	NAME	PRICE

Example of contracts

Contract

Con_ID	Cus_ID	DATE	AMOUNT
1	3	1-6-97	50.000.000
2	4	3-8-97	8.000.000
3	3	1-9-97	5.500.000
4	1	1-7-97	12.000.000
5	1	1-8-97	1.500.000
6	3	3-9-97	27.000.000

Definition of the table Product

```
create table Product
 ( Prod_ID   char(6)
             primary key,
   Name      char(20),
   Price     smallint )
```

Definition of the table Customer

```
create table Customer
 ( Cus_ID     char(6)
              primary key,
   Address  char(50),
   Tax_ID       char(12) unique )
```


Definition of the table Contract

```
create table Contract
(  Con_ID    char(6)   primary key,
   Cus_ID    char(6)   not null
                       default='999999',
   Date      date,
   Amount    integer,
   foreign key Cus_ID
      references Customer
         on delete set default
         on update set default)
```


Definition of the table Detail

```
create table Detail
(  Con_ID  char(6),
   Prod_ID char(6),
   Qt      smallint,
    primary key(Con_ID,Prod_ID)
    foreign key Con_ID
      references Contract
        on delete cascade
        on update cascade
    foreign key Prod_ID
      references Product
        on delete no action
        on update no action)
```


Basic Queries in SQL

SQL as a query language

- SQL queries are declarative
  - The user specifies which information he wants, but not how to extract it from the data
- The DBMS's query optimizer translates the queries into an internal procedural representation
- The programmer concentrates on legibility, not on efficiency
- This is a key point in relational databases

SQL queries

- Its three parts are called clauses:
  - `select` clause (target list)
  - `from` clause
  - `where` clause
- Syntax:
  `select` *AttrEspr* {, *AttrEspr*}
  `from` *Table* {, *Table*}
  [`where` *Condition*]
- Meaning:
  - Make the Cartesian product of the tables in the `from` clause
  - Only consider those lines satisfying the `where` clause
  - For each line, evaluate the expression in the target list

Algebraic interpretation of SQL queries

- Generic query :

  `select` *Table1.Attribute1* , … , TableN.AttributeN
  `from` *Table1*, …, *TableM*
  `where` *Condition*

- Corresponds to the relational algebra query:

$$\pi_{Table1.Attribute1,\ldots,TableN.AttributeN}\left(\sigma_{Condition}(Table1 \times \ldots \times TableM)\right)$$

Example: managing university exams

Student

SID	NAME	CITY	MAJOR
123	Pierre	Lyon	Inf
415	Celine	Lille	Inf
702	Estelle	Paris	Log

EXAM

SID	CLASS	DATE	GRADE
123	1	7-9-03	10
123	2	8-1-03	8
702	2	7-9-03	5

CLASS

CID	TITLE	TEACHER
1	maths	Leguichet
2	CS	Duchat

Basic queries

```
select  *
from Student
```

SID	NAME	CITY	MAJOR
123	Pierre	Lyon	Inf
415	Celine	Lille	Inf
702	Estelle	Paris	Log

Basic queries

Student

Sid	Name	City	Major

```
select Name
from Student
where Major = 'Log'
```

Algebraic interpretation (without duplicates)

$\Pi_{\text{Name}}\ \sigma_{\text{Major='Log'}}$ **Student**

Syntax of `select` clause

```
select  *
select  Name, City
select distinct  City
select  City as HomeTown
select Grade * 0.05
         as Bonus
select  sum(Income)
```


Syntax of `from` clause

```
from  Student
from  Student as X
from  Student, Exam
from  Student join Exam
     on Student.Sid=Exam.Sid
```

Syntax of `where` clause

- Boolean expressions with simple predicates (as in algebra)
- A few extra predicates:
  - **between: containment within range**
    Date between 1-1-90 and 31-12-99
  - **like: pattern matching on strings**
    Major like 'Lo%'
    Targa like 'MI_777_8%'

Conjunction of predicates

- Extract computer science students from Lyon:

```
select *
from Student
where Major = 'Inf' and
      City = 'Lyon'
```

- Result:

Sid	Name	City	Major
123	Pierre	Lyon	Inf

Disjunction of predicates

- Extract students from Lyon or from Lille:

```
select *
from Student
where City = 'Lyon' or
      City = 'Lille'
```

- Result:

Sid	Name	City	Major
123	Pierre	Lyon	Inf
415	Celine	Lille	Inf

Boolean expressions

- Extract students from Paris, that study computer science or logistics:

```
select *
from Student
where City = 'Paris' and
      (Major = 'Inf' or
       Major = 'Log')
```

- Result:

Sid	Name	City	Major
702	Estelle	Paris	Log

Like operator

- Extract students with a name having an ‘e’ at its second position, and as last two positions ‘ot’:

```
select *
from Student
where Name like '_e%ot'
```

- Result:

Sid	Name	City	Major
123	Pierrot	Lyon	Inf

Duplicates

- In relational algebra, the result of queries do not contrain any duplicates
- In SQL however, the tables returned by queries may contain identical lines
- Duplicates can be eliminated by the keyword `distinct`

Duplicates

```
select
distinct Major
from Student
```

Major
Inf
Log

```
select Major
from Student
```

Major
Inf
Inf
Log

Dealing with null values

- Null values represent three distinct situations:
  - a value does not apply
  - a value applies, but is unknown
  - Unknown whether value applies or not
- SQL-89 uses a two-valued logic
  - A comparison with *null* returns FALSE
- SQL-2 uses a three-valued logic: True, False, Unknown
  - A comparison with *null* returns UNKNOWN
  - Provides a predicates for null values nulli:

    Attribute **`is`** [**`not`**] **`null`**

Queries with null values

```
select *
from Student
where City  is  [not]  null
```

if `City` has the value *null* then
`(City = 'Milano')` has value Unknown

Predicates and NULL values

- 3 valued logics (T,F,U)

  T and U = U
  T or U = T

  F and U = F
  F or U = U

  U and U = U
  U or U = U
  not U = U

- P =

```
(City is not null) and
(Major like 'CS%')
```

City	Major
Lyon	CS
Lyon	NULL
NULL	CS
Lyon	Log

P	TUPLE SELECTED
T	yes
U	no
F	no
F	no

Queries and NULL values

```
select *
from Student
where Major = 'CS' or
      Major <> 'CS'
```

Is equivalent to:

```
select *
from Student
where Major is not null
```


Basic queries with two tables

Extract names of students in logistics that have passed with at least 15

```
select Name
from Student, Exam
where Student.Sid = Exam.Sid
  and Major like 'Lo%' and Grade >= 15
```

Name
Pierre

Simple query with 3 tables

- Extract names of students in "Maths" that have had at least one 18

```
select Name
from Student, Exam, Class
where  Student.Sid = Exam.Sid
  and Class.Cid = Exam.Cid
  and Title like 'Mat%' and Grade = 18
```

$\pi_{Name}\ \sigma_{(Title\ like\ 'Mat\%')\ and\ (Grade\ =18)}$ (Student ⋈ Exam ⋈ Class)

Join in SQL-2

- SQL-2 has introduced an alternative Syntax for joins, explicitly representing in the `from clause`:

`select` *AttrEspr* {, *AttrEspr*}
`from`
Table { [*JoinType*] **`join`** *Table* `on` *Conditions*}

[`where` *OtherConditions*]

Join between two tables in SQL-2

```
select Name
from Student, Exam
where  Student.Sid = Exam.Sid
     and Major like 'Mat%' and Grade= 18
```

```
select Name
from Student join Exam
      on Student.Sid = Exam.Sid
where Major like 'Mat%'and Grade=
18
```


Example database: drivers and cars

DRIVER

FirstName	Surname	DriverID
Mary	Brown	VR 2030020Y
Charles	White	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

CarRegNo	Make	Model	DriverID
ABC 123	BMW	323	VR 2030020Y
DEF 456	BMW	Z3	VR 2030020Y
GHI 789	Lancia	Delta	PZ 1012436B
BBB 421	BMW	316	MI 2020030U

- Note that Marco Neri DOES NOT drive, and that the BMW 316 NON is not driven

Inner Join (mostly used)

- *JoinType* can be
  - `inner:` the usually used join, by default omitted
  - `right, left, full:` is the external join
- Extract drivers and their cars

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta

Left join

- Extract drivers with their cars, including those drivers that don't have a car:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver left join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Right join

- Extract drivers and their cars, including also the cars without drivers:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver right join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
NULL	NULL	NULL	BBB 421	BMW	316

Full join

- Extract all drivers and all cars, including both drivers without cars, and cars without drivers:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver full join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

Variables in SQL

Full syntax for variables

`select` *AttrEspr* [[`as`] *Alias*] {, *AttrEspr* [[`as`] *Alias*]
`from` *Table* [[`as`] *Alias*] {, *Table* [[`as`] *Alias*] }
[`where` *Condition*]

Two purposes:

- renaming the result in the select clause
- variables for relation names in the from clause

Basic queries with variables for relations names

Who are Giorgio's employees?

Employee

Sid	Name	HiringDate	Income	MgrID
1	Piero	1-1-95	3 M	2
2	Giorgio	1-1-97	2,5 M	null
3	Giovanni	1-7-96	2 M	2

Who are Giorgio's employees?

```
select  X.Name, X.MgrID, Y.Sid, Y.Name
from
        Employee as X, Employee as Y
where
        X.MgrID = Y.Sid
        and Y.Name = 'Giorgio'
```

X.Name	X.MgrID	Y.Sid	Y.Name
Piero	2	2	Giorgio
Giovanni	2	2	Giorgio

Modification commands

Modification commands in SQL

- Operations
  - **`insert`** `new lines`
  - **`delete`** `existing lines`
  - Modifying values of attribute values (**`update`**)
- set-oriented: all operations can apply on a set of tuples
- The commands can contain a condition, that may access other tables

Insertion

- Syntax:

  `insert into` *TableName* [(*AttributeList*)]
  < **`values`** (*ValueList*) | *SelectSQL*>
- Using **`values`**:

```
insert into Student
values ('456878', 'Giorgio Rossi',
        'Lyon', 'Log')
```

- Using a query:

```
insert into Chtis
  (select *
   from Student
   where City = 'Lille')
```


Insertion

- The order of attributes and values matters: positional notation – the first value is affected to the first attribute, etc
- If the *AttributeList* is omitted, then all attributes of the relation are considered, in the order in which they appear in the table's definition.
- If the *AttributeList* does not contain all attributes of the relation, the remaining attributes are assigned the default value (if specified, otherwise NULL)

Insertion

- Using **values with** *AttributeList*:

```
insert into Student(Sid,Name,City,Major)
values ('456878', 'Antoine Bailleul',
        'Bergues', 'Log')
```

- Using a query with *AttributeList*:

```
insert into Chtis(Sid,Name,City,Major)
  (select Sid, Name, City, Major
   from Student
   where City = 'Lille')
```


Deletion

- Syntax:

delete from *TableName* [**where** *Condition*]

- Delete the student with identifier 678678:

```
delete from Student
  where Sid = '678678'
```

- Delete all students that haven't taken any exam:

```
delete from Student
  where Sid not in (select Sid
                    from Exam)
```


Deletion

- The **delete** command deletes all tuples satisfying the condition from the table
- The command can lead to deletions in other tables, if there is a referential integrity constraint with **cascade**
- When the **where** clause is ommited, the **delete** command deletes all tuples
- Example: deleting all tuples from STUDENT (maintaining the table's schema):

```
delete from Student
```

- The complete STUDENT table can be deleted (content and schema):

```
drop table Student cascade
```


Modifications

- Syntax:

update *TableName*
set *Attribute* = < *Expression* | *SelectSQL* | **null** | **default** >
{ , *Attribute* = < *Expression* | *SelectSQL* | **null** | **default** >}
[**where** *Condition*]

- Examples:

```
update Exam
set Grade = 20
where Date = 1-4-03

update Exam
set Grade = Grade + 1
where Sid = '787989'
```

Modifications

- Although the language is set-oriented, the order of command is very important

```
update Employee
  set Income = Income * 1.1
  where Income <= 30
update Employee
  set Income = Income * 1.15
  where Income > 30
```

- When the commands are written in this order, no employee can benefit from two increases!