Introduction to Relational Databases

- Licence Informatique, Université Lille 1
- Sept 14, 2011 (lecture 3/12)
- Today's lecturer: A. Bonifati
- Topic: Introduction to SQL
  - Data Definition Language
  - Data types
  - Table creation
  - Constraints

Introduction to SQL

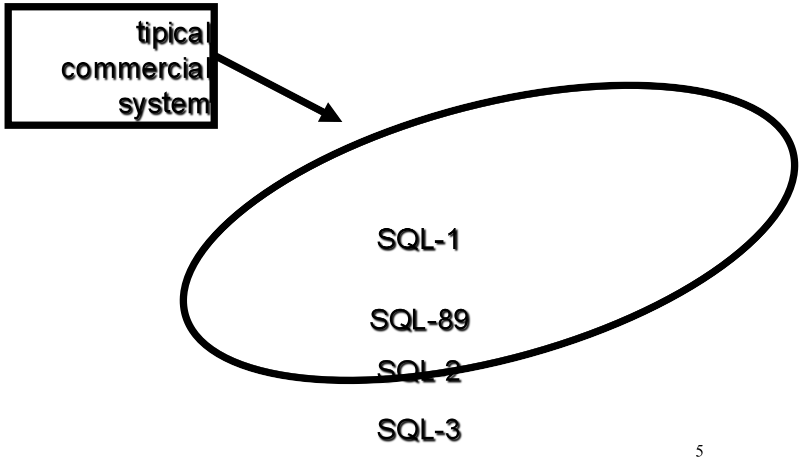
SQL

- Structured Query Language
- Consists of:
  - DDL: definition of domains, relations, indexes, authorizations, views, constraints, procedures, triggers
  - DML: query language, update language, transactional commands
- History:
  - First proposal: SEQUEL (IBM Research, 1974)
  - First commercial implementation in SQL/DS (IBM, 1981)
  - Standardization (1986-2003)

Standardization of SQL

- The standardization has been of utmost importance for the success of SQL (mainly within ANSI and ISO)
  - From 1983, it is a standard de facto
  - SQL-1: SQL-86 (basic constructs), SQL-89 (referential integrity constraints)
  - SQL-2: SQL-92 – most adopted version so far
  - SQL-3: SQL:1999 e SQL:2003 – most complete version, with triggers, objects, external functions, extensions for Java and XML
- In SQL-2 there are three levels:
  - Entry SQL (more or less equivalent to SQL-89)
  - Intermediate SQL
  - Full SQL
- Most of the systems are compliant to the Intermediate level and offer proprietary extensions for advanced functions

Expressiveness of commercial systems versus SQL standard



Definition of schemas in SQL

Definition of schemas

- A schema is a collection of objects:
  - domain, tables, indices, assertions, views, privilegdes
- Each schema has a name and an owner
- Typical systems do not implement the schema definition of SQL-2 (that however define these at the beginning of a DDL session)

- Syntax:

  `create schema` [SchemaName]
  [[`authorization`] Authorization]
  { SchemaElementDefinition }

Data types

- Data types specify the possible values, for each attribute
  - Similar to type definitions in programming languages
- Two categories
  - Built-in (predefined by the SQL standard)
    - SQL-2 distinguishes 6 families
  - user-defined

Built-in data types, 1

- Characters
  - Single charaters or strings
  - Strings can have variable lengths
  - Can use character sets that differ from the defaults (e.g., Latin, Greek, Cyrillic, etc.)
  - `character` [`varying`] [(Length)]
    [`character set` CharacterFamilyi]
  - Can use more compact alternatives as `char` and `varchar`, respectively for `character` and `character varying`
  - Examples:
    - `char(6)`
    - `varchar(50)`

Built-in data types, 2

- Bit
  - Boolean values (true/false), singole or in a sequence (the sequence may be of variable length)
  - Syntax:

    `bit` [`varying`] [(Length)]

    Examples: bit(100), varbit(680)
- Precise numeric types
  - Numeric values: integers or reals
  - 4 alternatives:

    `numeric` [(Precision [, Scale])]

    `decimal` [(Precision [, Scale])]

    `integer`

    `smallint`

Built-in data types, 3

- Approximate numeric types
  - Approximate real values
  - Based on a floating point representation: integer part + exponent

    `float` [(Precision)]

    `real`

    `double precision`

Built-in data types, 4

- Time points
  - Ammettono dei campi

    `date` (fields `month, day, year`)

    `time` [(Precision)] [`with time zone`] : (fields `hour, minute, second`)

    `timestamp` [(Precision)] [`with time zone`]

    with timezone, one has two additional fields `timezone_hour` and `timezone_minute`

    - Example: timestamp(4) with time zone 2-30-2004 3-13-42.0564 5-30
- Time intervals

    `interval` FirstTimeUnit [`to` LastTimeUnit]

  - We distinguish 2 groups of time units gruppi:
    - year, month
    - day, hour, minute, second
  - Examples:
    - `interval year to month`
    - `interval second`

Built-in data types, 5

- New built-in types in SQL-3
  - `Boolean`
  - `Bigint`
  - `BLOB` Binary Large OBject
  - `CLOB` Character Large Object
- SQL:1999 also introduces constructors (`REF`, `ARRAY`, `ROW`; they go beyond the relational model and we won't talk about them)

User defined data types

- Similar to type definitions in programming languages: for an object, define the values it may take
- A data type is specified by
  - name
  - elementary type
  - default value
  - constraints
- Syntax:

  `create domain` DomainName `as` ElementaryDomain
  [ValoreDefault] [Constraints]
- Example:

  `create domain Grade as smallint default null`
- Comparison to programming languages
- \+ constraints, default values, richer basic types
  \- approved constructors (only renaming of types)

Default values for types

- Define the value of an attribute, when no value is specified as a tuple is inserted
- Syntax:

  `default` < GenericValue | `user` | `null` >
- GenericValue represents a value compatible with the type, given by a constant or expression
- `user` is the login of the user that executes the command

"null" values

`Null`
is a polymorphic value (that is included in all types), and means that a value is unknown

- the value exists in reality, but is unknown to the database (ex.: birthday)
- The value doesn't apply (ex.: driver's licence number for children)
- It is unknown if the value is unknown, or used (ex.: driver's licence number for adults)

Definition of application domains

```
create domain DaylyPrice
as decimal(3)
     default 1,00
     not null
```


Table creation

- Each SQL table consists of:
  - an ordered set of attributes
  - a set of constraints (may be empty)
- `create table` command
  - Defines the schema of a relation, by creating an empty instance
- Syntax:

```
create table TableName
  (
  AttributeNameo Type [ DefaultValue ] [ Constraints ]
  {, AttributeNameo Type [ DefaultValue ] [ Constraints ] }
  [ OtherConstraints ]
  )
```


`create table` example (1)

```
create table Student
  (  Sid       character(6) primary key,
     Name       varchar(30) not null,
     City       varchar(20),
     Major      char(3) )
```


`create table` examples (2)

```
create table Exam
 (  Sid char(6),
    Cid   char(6),
    Date       date  not null,
    Grade       smallint not null,
    primary key(Sid,Cid) )

 create table Class
 (  Cid (6) primary key,
    Title   varchar(30) not null,
    Teacher  varchar(20) )
```

Referential integrity

- Integrity constraints: conditions that must be satisfied by all instances of the data base
- Contraints on a single relation
  - `not null` (for one attribute)
  - `primary key` (implies not null);
    - For a single attribute:
      `primary key`, after the type
    - For several attributes:
      `primary key`(Attribute{, Attribute })
  - `unique`: key candidates, syntax as for `unique`
  - `check`: will be explained later (can represent generic predicates in SQL)

Examples of integrity constraints

- Each pair of Name and FirstName uniquely identifies a tuple

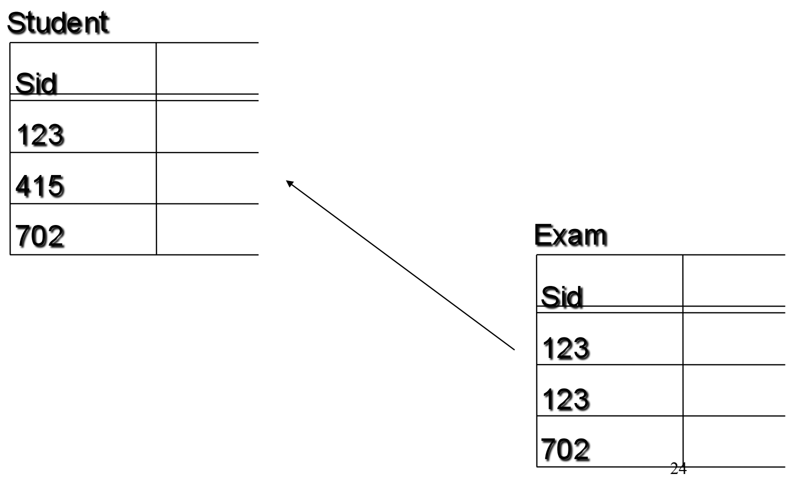
```
Name character(20) not null,
FirstName character(20) not null,
unique(Name,FirstName)
```

- Note the difference to the following definition (more restrictive):

```
Name character(20) not null unique,
FirstName character(20) not null unique,
```


Referential integrity

Example: Student - Exam



The orphan problems

Student

Sid	
~~123~~	
415	
702	

orphans:
Tuples that are without parent, after deletion of modification of the parent table

Exam

Sid	
~~123~~	
~~123~~	
702	

How to deal with orphans?

- After modification of the parent table, some operations are performed on the child table
- Violation can be introduced by:
  - (1) updates of the referred attribute
  - (2) deletion of tuples
- Possible reactions:
  - `cascade`: propagates the modification
  - `set null`: cancels the referring attribute
  - `set default`: assigns the default value to the tuple
  - `no action`: makes the mofification impossible
- The reaction can depend on the kind of event ; Syntax:

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


Dealing with orphans: deletion

If a tuple is deleted within Student, what happens to his/her exams?

- `cascade`
  the Student's exams are also deleted
- `set null`
  the Sid within Exam is set to null
- `set default`
- the Sid within Exam is set to the default value
- `no action`
  The deletion of tuples within Student is forbidden

Dealing with orphans: update

If an Sid is modified within Student, what happens to his/her exams?

- `cascade`
  the Sid of the students within Exam is also modified
- `set null`
  the Sid of the students within Exam is set to null
- `set default`
  the Sid of the students within Exam is set to the default value
- `no action`
  the modification of the Sid within Student is forbidden

Syntax for integrity constraints

- Attributes that are foreign keys inside the child relation must have values present as key values inside the father relation
- `references` and `foreign key` for referential integrity constraints;
- Syntax:
  - for one attribute
    `references` after Type
  - for one or more attributes
    `foreign key` (Attribute {, Attribute })
    `references` …

Definition: inside the child relation

```
create table Exam
   ( ....
     ....
     foreign key Sid
       references Student
         on delete cascade
         on update cascade )
```


Definition: inside the child relation

```
create table Exam
   ( Sid char(6) references Student
         on delete cascade
         on update cascade ,
     ......)
```


It is allowed to have multiple fathers!

```
create table Exam
( ....
    primary key(Sid,Cid)
    foreign key Sid
      references Student
        on delete cascade
        on update cascade
    foreign key Cid
      references Class
        on delete no action
        on update no action )
```

An incorrect instance

Sid	Name	City	Major
123			
415			
702			

Exam

Sid	Cid	Date	Grade
123	1	7-9-97	30
123	2	8-1-98	28
123	2	1-8-97	28
702	2	7-9-97	20
702	1	NULL	NULL
714	1	7-9-97	28

it violates the key

it violates the NULL

it violates the ref. integrity