Basi di Dati e Sistemi Informativi I

SQL

Data definition in SQL

SQL

- The name is an acronym for *Structured Query Language*
- SQL is not merely a query language. It contains the dual features of a
  - *Data Definition Language*, DDL (with commands for the definition of a relation database schema)
  - *Data manipulation Language*, DML (with commands for the modification and querying of a database instance)

SQL

- **History**:
  - First proposal: SEQUEL (IBM Research, 1974)
  - First implementation in SQL/DS (IBM, 1981)
- Standardization crucial for its diffusion
  - Since 1983, *standard de facto*
  - First standard, 1986, revised in 1989 (SQL-89)
  - Second standard, 1992 (SQL-2 or SQL-92)
  - Third standard, 1999 (SQL-3 or SQL-99)
- Most relational systems support the base functionality of the standard and offer proprietary extensions

SQL: syntax

- Angular brackets < and > are used to enclose terms;
- Square brackets [and] indicate that the enclosed term is optional, that is, it may not appear or appear only once;
- Curly brackets { and } indicate that the enclosed term may not appear or may be repeated an arbitarary number of times;
- Vertical bars indicate that one among the terms separated by the bars must appear;
- Curved brackets (and) must always be taken as SQL keywords and not as grammar definition

Domains

- Domains specify the content of attributes
- Two categories
  - Elementary (predefined by the standard)
    - Six families of elementary domains:
      - character, bit, exact numeric doamins, approximate numeric domains, date and time, temporal intervals
  - User-defined

Character

- Single characters or strings
- Strings may be of variable length
- A Character set different from the default one can be used (e.g., Latin, Greek, Cyrillic, etc.)
- Syntax:

`character` [`varying`] [(*Length*)] [`character set` *CharSetName*]

- Ex.s
  - 'string of 20 characters' : `character (20)`
  - 'string of Greek letters of variable length, maximum lenght 10000': `character varying (1000) character set Greek`
- It is possible to use `char` and `varchar`, respectively for `character` and `character varying`

Bit

- Single boolean values or strings of boolean values (may be variable in length)
- It is typically used to represent attributes, known as *flags*, which specify whether an object has or not a certain proprty.
- 'string of bits' for which the length is specified as a parameter. It is used for the concise representation of groups of properties.
- Syntax:
  `bit [ varying ] [ (`*Length*`) ]`
- Ex.s
  - 'string of 5 bits': `bit(5)`
  - 'string of bits of variable length and maximum length of 100':
    `bit varying(100)` or `varbit(100)`

Exact numeric domains

- Exact values, integer or with a fractional part
- Four alternatives:

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

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

  `integer`

  `smallint`

- `Numeric` and `decimal` represent number with a *decimal base*. *Precision* specifies the number of significant degits (e.g., `decimal(4)` values between –9,999 and +9,999). *Scale* indicates how many digits should appear after the decimal point (e.g., `numeric(6,3)` values between –999.999 and +999.999)
- The difference between `Numeric` and `decimal` lies in the fact that the numeric domain has exactly the precision as indicated, while the precision of the decimal domain should be taken as minimum requirement.

Approximate numeric domains

- Approximate real values
- Based on a floating point representation

  `float[(`*Precision*`)]`

  `double precision`

  `real`
- The approximate value of the real number is obtained by multiplying the mantissa by the power of 10 specified by the exponent (e.g., 0.17E16 represents the value 1.7×10^{15}, and 0.4E-6 represents 4×10^{-7}).
- A given precision can be specified for the domain `float`, which represents the number of digits dedicated to the representation of the mantissa, while the precision of the exponents depends on the implementation.
- The domain `double precision` represents the numbers with a greater precision than domain `real`.

Temporal instants

`date`

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

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

- They can be structured in fields.
  - `Date` allows the field *year*, *month* and *day*, `time` allows the fields *hour*, *minute* and *seconds*, and `timestamp` allows all the fields, from *year* to *second*.
  - For both `time` and `timestamp` we can specify the precision, which represents the number of decimal places that must be used in the representation of fractions of a second.

Temporal intervals

`interval` *FirstUnitOfTime* [`to` *LastUnitOfTime*]

- Units of time are divided into two groups:
  - year, month
  - day, hour, minute, second
- The first unit that appears in the definition can be characterized by the precision, which represents the number of decimal digits used in the representation. When the precision is not specified, it assumes the default value 2.
- Ex.s
  - `Interval year(5) to month`: the intervals up to 99,999 years and 11 months.
  - `Interval day(4) to second(6)`: the intervals up to 9,999 days, 23 hours 59 minutes and 59.999999 seconds

Schema definition

- A schema is a collection of objects:
  - domains, tables, indexes, assertions, views, privileges
- A schema has a name and an owner (the authorization)
- Syntax:

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

- After the create schema command, the user can define the schema components. It is not necessary for all the components to be defined at the same time as the schema is created: this can take place in several successive phases.

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
- Syntax:

`create table` *TableName*
(
AttributeName Domain [*DefaultValue*] [*Constraints*]
{, *AttributeName Domain* [*DefaultValue*] [*Constraints*] }
[*OtherConstraints*]
)

An example of `create table`

```
create table Department
(
  Name    char(20) primary key,
  Address char(50)
  City    char(20)
)
```

Table with three attributes of character string domain; the attribute Name constitutes the primary key of the table.

Another 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    numeric(9) default 0,
      City      character(15),
      unique(Surname,FirstName)
  )
```


User defined domains

- Comparable to the definition of variable types in programming languages
- A domain is characterized by
  - name
  - elementary domain
  - default value
  - set of constraints
- Syntax:
  `create domain` *DomainName* `as` *ElementaryDomain*
  [*DefaultValue*] [*Constraints*]
- Example:
  `create domain Mark as smallint default null`

Default domain values

- Define the value that the attribute must assume when a value is not specified during row insertion
- When a default value is not specified, the value *null* is assumed as default.
- Syntax:
  `default` < *GenericValue* | `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
- Ex. An attribute `NumberOfChildren`, which allows an integer as a value and which has the default value zero is defined by:

```
NumberOfChildren smallint default 0
```


Intra-relational constraints

- Constraints are conditions that must be verified by every database instance
- Intra-relational constraints involve a single relation
  - `not null` (on single attributes)
  - `unique`: permits the definition of keys; syntax:
    - 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`
  - `check`: described later

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
```

- The pair of attributes `FirstName` and `Surname` constitute the primary key:

```
FirstName character(20),
Surname  character(20),
Dept     character(15),
Salary   numeric(9) default 0
Primary key(FirstName,Surname)
```


Inter-relational constraints

- This contraint creates a link between the values of the attribute(s) of a table and the value of the attribute(s) of another table.
- Constraints may take into account several relations
  - `check`: described later
  - `references` and `foreign key` permit the definition of referential integrity constraints; syntax:
    - for single attributes
      `references` after the domain
    - for multiple attributes
      `foreign key` (*Attribute* {, *Attribute* })
      `references` …
  - It is possible to associate reaction policies to violations of referential integrity

Examples of inter-relational constraint

```
create table Employee
(
      RegNo char(6) primary key,
      FirstName char(20) not null,
      Surname char(20) not null,
      Dept char(15) reference Departmaent(DeptName),
      Salary numeric(9) default 0,
      City char(15),
      unique(FirstName,Surname)
)


create table Employee
(
      RegNo char(6) primary key,
      FirstName char(20) not null,
      Surname char(20) not null,
      Dept char(15) reference Departmaent(DeptName),
      Salary numeric(9) default 0,
      City char(15),
      primary key(RegNo),
      unique(FirstName,Surname),
      foreign key(FirstName,Surname) reference
                        PersonalRecord(FirstName,Surname)
)
```


Reaction policies for referential integrity constraints

- Reactions operate on the internal table, after changes to the external table
- Violations may be introduced (1) by updates on the referred attribute or (2) 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 char(6),
    FirstName char(20) not null,
    Surname char(20) not null,
    Dept char(15),
    Salary numeric(9) default 0,
    City char(15),
    primary key(RegNo),
    foreign key(Dept)
           references Department(DeptName)
           on delete set null
           on update cascade,
    unique(FirstName,Surname)
)
```


Schema updates

- SQL provides primitives for the manipulation of database schems, which enable the modification of previously introduced table definition.
- Two SQL statements: `alter` and `drop`
- The `alter` command allows the modification of domains and schemas of tables.

```
alter domain DomainName<set default DefaulValue|
           drop default |
           add contraint ConstraintDef |
           drop constraint ConstraintName >

alter table TableName
           alter column AttributeName
                  <set default DefaultValue |drop default>
           add consstraint ConstraintDef|
           drop constraint Constraint|
           add column AttributeDef |
           drop column Attribute-name >
```

- Ex:

```
alter table Department
        add column NoOfOffices numeric(4)
```

Schema updates

- The `drop` command allows the removal of components, whether they be schemas, domains, tables, views or assertions (assertions are constraints that are not associated with any particualr table, see later)

  `drop` < `schema` | `domain` | `table` | `view` | `assertion` >
  ComponentName [`restrict` | `cascade`]

- The `restrict` option specifies that the command must not be carried out if the component being deleted is not empty. A schema is not removed if it contains tables or other elements; a domain is not removed if it appears in a table definition; a table is not removed if it possesses rows or if it is present in a definition of a table or view; and, finally, a view is not removed if it is used in the definitionof other views.
- With the cascade option, the component is removed together with the components depending on it.
- Ex:

```
drop table TempTable cascade
```


Relational catalogues

- The catalog contains the data dictionary, the description of the data contained in the data base
- It is based on a relational structure (reflexive)
- The SQL-2 standard describes a
  - Definition_Schema (composed of tables that contain the descriptions of all the structures in the database
  - and an Information_Schema (composed of views on the Definition_Schema)