Introduction to Database Systems

Data Definition and Manipulation in SQL

Werner Nutt

6. Data Definition and Manipulation in SQL

6.1 Data Definition

SQL

- Originally "**S**tructured **Q**uery **L**anguage", today a proper name
- A language with several functionalities
  - comprises both DDL and DML
- There exist several standards, and companies have added proprietary extensions
- We concentrate on the principles, not the details
- “History”:
  - First proposal of **SEQUEL** (IBM Research, 1974)
  - First implementation in SQL/DS (IBM) and Oracle (1981)
  - Since around 1983 there is a “de facto standard”
  - Standard definitions (ISO): 1986, then 1989, then **1992**,
    thereafter 1999 (e.g. triggers, oo features),
    2003, 2006 (XML)—so far, only partly realised

SQL-92

- A rich and complex language
- Three levels of adherence to the standard:
  - **Entry SQL**: similar to SQL-89
  - **Intermediate SQL**: comprises functionalities that are important for business applications; supported by commercial DBMSs
  - **Full SQL**: advanced functionalities
- Commercial systems offer features that are not part of the standard
  - Incompatibilities between systems
  - Incompatibilities with newer standards (e.g. triggers in SQL:1999)

Data Definitions in SQL

- Apart from the command **`create schema`** (which is used to create a schema), the most important command of the DDL in SQL is

  `create table`

  – Defines a relation schema (with attributes and integrity constraints)

  – Creates an empty instance of the schema

- Syntax:

`create table` *TableName* **(**

AttributeName Domain [*Constraint*]

....

AttributeName Domain [*Constraint*]

[*OtherConstraints*]

)

Create Table (Example)

```
create table Employee (
   EmpNo                character(6) primary key,
   FirstName            character(20) not null,
   LastName             character(20) not null,
   Dept                 character(15),
   Salary               numeric(9) default 0,
   City                 character(15),
   foreign key(Dept) references Department(DeptName),
   unique (LastName,FirstName)
)
```

SQL and the Relational Model

- **Difference**: a table instance in SQL is defined as a multiset (bag) of tuples.

- In particular, if a table does not have a primary key or a set of attributes that are defined as unique, it is possible that two identical tuples appear in an instance of that table.

  Thus, *in general an SQL table is not a relation*.

- If, however, a table has a primary key or a set of attributes that are defined as unique, there can never be two identical tuples in a relation.

  $\rightarrow$ It is advisable to define at least a primary key for a relation.

Domains

- **Elementary domains or types** (predefined)
  - **Character**: single characters or strings, both of fixed and variable length
  - **Bitstrings**: string elements are 0 and 1
  - **Numbers**: integers and reals
  - **Dates, timestamps, time intervals**
  - Introduced in SQL:1999
    - **Boolean**
    - **BLOB**, **CLOB** (binary/character large object): for large images or texts

  In some systems, enumeration types can be defined

- **User defined domains** (reusable)

Domain Definitions

- The instruction

  `create domain`

  defines a (simple) domain with integrity constraints and defaults, which can be reused in table definitions.

- Syntax

  `create domain` *DomainName*
  `as` *Type* [*Default*] [*IntegrityConstraint*]

- ***Example***:

```
create domain EmployeeAge
as smallint default null
check ( value >=18 and value <= 67 )
```

Constraints on a Relation

- **`not null`** (on single attributes)

- **`unique`**: allows one to define a (candidate) key:
  - single attribute:

    `unique` after the specification of the domain
  - several attributes (i.e., one or more):

    `unique (`*Attribute***`,`***...***`,`** *Attribute***`)`**

- **`primary key`**: definition of the primary key
  (only one, implies **`not null`**); syntax as for **`unique`**

- **`check`**, for more complex constraints (see below)

Constraints on a Relation (Example)

```
create table Employee (
   EmpNo              character(6) primary key,
   FirstName          character(20) not null,
   LastName           character(20) not null,
   Dept               character(15),
   Salary             numeric(9) default 0,
   City               character(15),
   foreign key (Dept) references Department(DeptName),
   unique (LastName,FirstName)
)
```

`primary key` (Alternate Definition)

```
create table Employee (
  EmpNo character(6) primary key,
  ...
)
```

or

```
create table Employee (
  EmpNo character(6),
  ...
  primary key (EmpNo)
)
```

Candidate Keys: Mind the Step!

```
create table Employee ( ...
  FirstName  character(20) not null,
  LastName   character(20) not null,
  unique (LastName,FirstName)
)
```

is different from:

```
create table Employee ( ...
  FirstName  character(20) not null unique,
  LastName   character(20) not null unique
)
```

Constraints Between Relations

- `check`, for complex constraints
- `references` and `foreign key` allow one to define referential integrity constraints.

  Syntax:

  – for single attributes:

  `references` after the specification of the domain

  – for several attributes:

  `foreign key(`*Attribute*`,`...`,`*Attribute*`)references`** ...

  The referenced attributes in the target table must form a key (**`primary key`** or **`unique`**). If they are missing, the foreign key refers to the primary key of the target table.

  Semantics: every combination (not involving NULL) of attribute values in the source table must appear in the target table.

- It is possible to add policies that specify how to react to constraint violations (which are caused by changes of the target table).

Foreign Keys (Example)

```
create table Student(
 StudNo   character(10) primary key,
 Name     character(20),
 Hons     character(3),
 Tutor    character(20) references Staff(Lecturer),
 Year     smallint)


create table Staff(
 Lecturer  character(20) primary key,
 RoomNo    character(4),
 Appraiser character(20),
 foreign key (Appraiser) references
             Staff(Lecturer)
             on delete set null
             on update cascade)
```

Policies

- Determine the effect of **`delete`** and **`update`** statements

- Syntax

  on **`delete`** *Action*

  where *Action* can be

`cascade`	(propagate the deletion)
`restrict`	(do nothing if the row is referenced)
`no action`	(as **`restrict`**, but return an error)
`set default`	
`set null`	

- The same actions exist for updates (**`on update`**)

Schema Updates

- **`alter domain`**: allows one to modify a domain definition
- **`alter table`**: allows one to modify a table
  - add or drop attributes
  - add or drop constraints
- **`drop domain`**: eliminates a domain
- **`drop table`**: eliminates a table

Catalogue or Data Dictionary

Every relational system offers predefined tables that collect data about:

- **tables**
- **attributes**
- **domains**
- …

For instance, the table **`Columns`** contains the attributes:

- **`Column_Name`**
- **`Table_name`**
- **`Ordinal_Position`**
- **`Column_Default`**
- …

6. Data Definition and Manipulation in SQL

6.2 Data Manipulaton

1. Data Definition
2. **Data Manipulation**

MotherChild

mother	child
Lisa	Mary
Lisa	Greg
Anne	Kim
Anne	Phil
Mary	Andy
Mary	Rob

FatherChild

father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Person

name	age	income
Andy	27	21
Rob	25	15
Mary	55	42
Anne	50	35
Phil	26	30
Greg	50	40
Frank	60	20
Kim	30	41
Mike	85	35
Lisa	75	87

Operations that Change the DB Instance

- Operations of
  - insertion: **`insert`**
  - elimination: **`delete`**
  - modification: **`update`**

- ... of *one or more* tuples of a relation ...

- ... using a *condition* that may also involve other relations

Insertion: Syntax

insert into *Table* [**(** *Attributes* **)**]
    values(*Values* **)**

(values are stated explicitly)

or

insert into *Table* [**(** *Attributes* **)**]
    select ...

(values are produced by a query)

Insertion: Examples

```
insert into person values('Peter',25,52)


insert into person(name, age, income)
   values('Paul',25,52)


insert into person(name, income)
   values('Mary',55)
```

(what about Mary’s age?)

```
insert into person (name)
   select father
   from fatherChild
   where  father not in (select name from person)
```

Insertion: Comments

- The ordering of attributes in the attribute list (if present) and of the values in the value list is crucial
- The list of attributes and the list of values must have the same number of elements
- If the list of attributes is missing, the list of all attributes is taken, with the ordering taken from the table definition
- If the list of attributes does not contain all attributes of the relation, the default value or the value null (if possible) is inserted for the missing attributes

Elimination of Tuples

Syntax:

delete from *Table* [**where** *Condition*]

Examples:

```
delete from person
where age < 35
```

(conditions are similar to query conditions)

```
delete from fatherChild
where child not in (select name from person)
```

Elimination: Comments

- *All tuples* that satisfy the *condition* are eliminated
- May cause *eliminations in other relations* if the repair policy **cascade** has been specified for those relations
- Note: if the **`where`** part is *missing*, it is understood as **`where true`**

Modification of Tuples

- **Syntax**:

`update` *TableName*
`set` *Attribute* **=** < *Expression* | **`select`** … | **`null`** | **`default`** >
[**`where`** *Condition*]

- **Semantics**: all tuples of the table are modified that satisfy the **`where`** condition

- *Examples*:

```
update person set income = 45
where  name = ‘Greg'

update person set income = income * 1.1
where  age < 30
```

References

In preparing the lectures I have used several sources.
The main ones are the following:

Books:

- A First Course in Database Systems, by J. Ullman and J. Widom
- Fundamentals of Database Systems, by R. Elmasri and S. Navathe

Slides:

- The slides of this chapter are mostly translations of material prepared by Maurizio Lenzerini (University of Rome, “La Sapienza”) and Diego Calvanese (Free University of Bozen-Bolzano) for their introductory course on databases at the University of Rome, “La Sapienza”