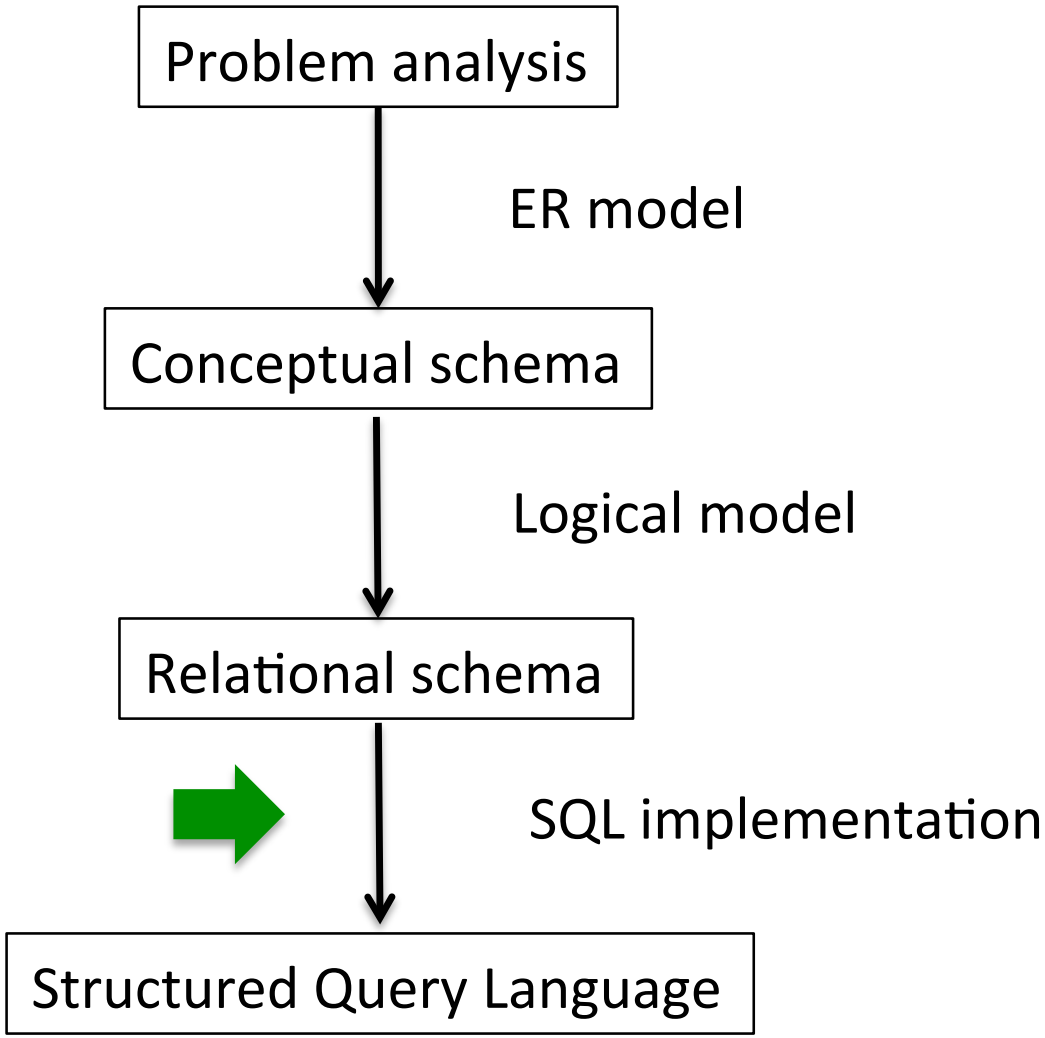
Database Management Systems

Lorraine Goeuriot
IUT 1, Université Grenoble Alpes
Auteur du cours : Marie-Christine Fauvet
Université Joseph Fourier, Grenoble – UFR IM^{2}AG

20 janvier 2016

Contenu

Structured Query Language - Introduction



Structured Query Language - Introduction

SQL is a language used to manage and query relational data :

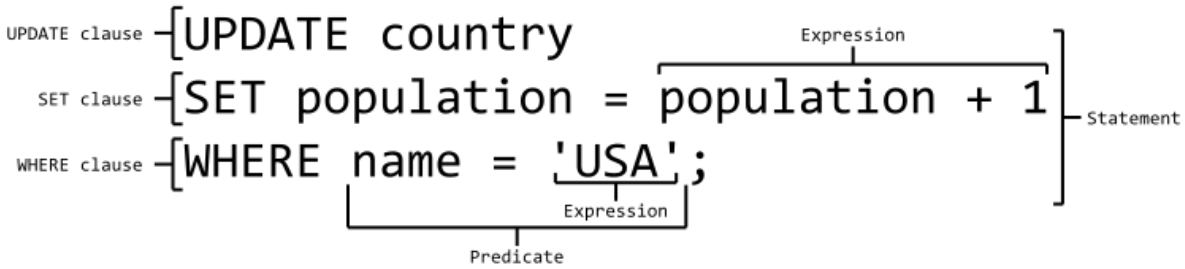
- developed at IBM in the 70s
- standardized in the 80s (ANSI/ISO)

It has been designed to be easily readable by humans.
SQL consists of :

- a data definition language (DDL) : manages table and index structure
- a data manipulation language (DML) : is the subset of SQL used to add, update and delete data
- a data control language (DCL) : authorizes users to access and manipulate data

Easy to write simple queries
VERY difficult to write complex queries !

Language Elements



- **Clauses** : constituent components of statements and queries
- **Expressions** : can produce either scalar values, or tables consisting of columns and rows of data
- **Predicates** : specify conditions that can be evaluated to boolean values (used to limit the effects of statements and queries)
- **Queries** : retrieve the data based on specific criteria
- **Statements** : may have a persistent effect on schemata and data, or may control transactions, program flow, connections, sessions, or diagnostics.

`https://en.wikipedia.org/wiki/SQL#Language_elements`

Data Definition Language

- Commands : CREATE, ALTER, DROP
- Can be applied to : databases, tables, views, indexes

Commands on DBs :

- `create database [if not exists] dbname [spec] ;`
  Create a DB on the DBMS
- `alter database dbname spec ;`
  Modification of a DB's characteristics
- `drop database dbname ;`
  Delete a DB

Commands on tables :

- `create table ... ;`
  Creation of a table with columns and constraints
- `alter table ... ;`
  Modification of the columns and constraints
- `drop table tablename ;`
  Deletion of a table
- `truncate table tablename ;`
  Deletion of the table's content

CREATE TABLE

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name
    (definition_col1,..., constraint_table, ...)
    [table_options]
    [partition_options]

Definition of the columns :

namecol type [NOT NULL | NULL]
    [DEFAULT default_value]
    [AUTO_INCREMENT]
    [PRIMARY] KEY]
    [COMMENT 'comments']
    [constraints_col...]

Column Types

- Numerical types :
  - Integer : `TINYINT (1)`, `SMALLINT (2)`, `MEDIUMINT (3)`, `INT (4)`, `BIGINT (8)`
  - Float : `FLOAT (4)`, `DOUBLE (8)`, `REAL (4)`
  - {`NUMERIC`, `DECIMAL`} `(precision, scale)`
- String : `CHAR`, `VARCHAR` ($<$ 255 car.), `TEXT`
- Boolean : `BOOLEAN`
- Time : `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`...
- String list : `ENUM` (fixed length), `SET` (variable length)

CREATE TABLE - Example

`Bar(name, address, license, openDate)`

- `name` is a string
- `address` is a string
- `license` is an integer
- `openDate` is a date

```
create table bar (
    name VARCHAR(20),
    address VARCHAR(100),
    license INT,
    openDate DATE);
```

Constraints

- There are two types of constraints :
  - Static integrity constraint (mark $\in [0, 20]$)
  - Relative integrity constraint (departure_time $<$ arrival_time)
- Two ways of defining constraints :
  - At the attribute level
  - At the table level (always the case when several attributes are involved)

CREATE TABLE - Not null integrity constraint

`bar(name, address, license, openDate)`
The license must be known at the creation

```
create table bar (
    name VARCHAR(20),
    address VARCHAR(100),
    license INT NOT NULL,
    openDate DATE);
```

CREATE TABLE - Default constraint

`bar(name, address, license, openDate)`
The opening date is the current day by default

```
create table bar (
    name VARCHAR(20),
    address VARCHAR(100),
    license INT NOT NULL,
    openDate DATE DEFAULT current_timestamp);
```

CREATE TABLE - Unicity constraint (1)

`bar(name, address, license, openDate)`
The address must be unique

```
create table bar (
    name VARCHAR(20),
    address VARCHAR(100) unique,
    license INT NOT NULL,
    openDate DATE DEFAULT current_timestamp);
```

CREATE TABLE - contrainte d'unicité (2)

`bar(name, address, license, openDate)`
The address must be unique

```
create table bar (
    name VARCHAR(20),
    address VARCHAR(100),
    license INT NOT NULL,
    openDate DATE DEFAULT current_timestamp);
    constraint [uniq_address] unique (address));
```

CREATE TABLE - Primary key constraint

- Defined at the attribute level :

  Create table bar(
  name VARCHAR(20) **PRIMARY KEY**,
  ...) ;

- Defined at the table level :

  Create table bar(
  name VARCHAR(20),
  constraint [pk_name] primary key (name),
  ...) ;

CREATE TABLE - Compound primary key

`sells(bar, beer, price)`
The constraint must be defined at the table level

```
create table sells (
    bar VARCHAR(20) NOT NULL,
    beer VARCHAR(20) NOT NULL,
    ...
    constraint [pk_sells] primary key (bar, beer));
```

CREATE TABLE - Primary key - Limits

- For an attribute to be a primary key, it must have a static length
- Hence an attribute of type `TEXT` cannot be chosen as a primary key
- We would use types `CHAR` or `VARCHAR` instead

CREATE TABLE - Validity constraint

`student(studentID, name, dob, gender)`
The gender is either female, or male

- Defined at the attribute level :

  create table student (
  idStudent VARCHAR(8) NOT NULL
  ...
  gender CHAR(1) NOT NULL **check (gender in ('F', 'M'))**);

- or at the table level :

  create table student (
  idStudent VARCHAR(8) NOT NULL,
  ...
  gender CHAR(1) NOT NULL,
  constraint [check_gender] check (gender in ('F', 'M')));

CREATE TABLE - Complex validity constraint

product(idProduct, initialPrice, selling Price)
The selling price must be higher than the initial price

```
create table product (
    idProduct INT NOT NULL PRIMARY KEY,
    initialPrice DOUBLE,
    sellingPrice DOUBLE
    constraint check_price check (initialPrice<sellingPrice));
```

CREATE TABLE - Foreign keys

`sells(bar#, beer#, price)`

Definition of foreign keys on this table :

```
CREATE  TABLE sells (
    bar VARCHAR(20),
    beer VARCHAR(20),
    price FLOAT,
    PRIMARY KEY (bar, beer),
    KEY c_sells_bar (bar),
    KEY c_sells_beer (beer),
    CONSTRAINT c_sells_bar FOREIGN KEY (bar) REFERENCES bars(name),
    CONSTRAINT c_sells_beer FOREIGN KEY (beer) REFERENCES beers(name) );
```

Columns definition

Creation of links from these attributes

Target table

Target column

Link back to the columns

CREATE TABLE - Strengthen foreign keys (1)

- Goal : manage the DB when modifying or deleting data with foreign keys
- Example :

bar(<u>name</u>, address, license, openDate)
sells(<u>bar#</u>, <u>beer#</u>, price)

What happens in table `sells` when a bar is deleted from the table `bars` ?

CREATE TABLE - Strengthen foreign keys (2)

bar(<u>name</u>, address, license, openDate)
sells(<u>bar#</u>, <u>beer#</u>, price)

What happens in table `sells` when a bar is deleted from the table `bars` ?

Default behaviour : **impossible if associated tuples are stored in** `sells`

CREATE TABLE - Strengthen foreign keys (3)

bar(<u>name</u>, address, license, openDate)
sells(<u>bar#</u>, <u>beer#</u>, price)

What happens in table `sells` when a bar is deleted from the table `bars` ?

How can we define another behaviour ?

foreign key (column_name[,...])
references reftable [(refcolumn [,...])]
[on delete action] [**on update** action]

CREATE TABLE - Strengthen foreign keys (4)

Actions :

- **no action** : default mode, blocks deletion if a tuple contains the element in related table
- **restrict** : same
- **cascade** : deletes linked elements in related tables
- **set null** : sets as null linked elements in related tables
- **set default** : same with the default value as default value

ALTER TABLE

Modification of a table schema :

- Add an attribute
- Delete an attribute
- Modify an attribute
- Add a constraint
- Delete a constraint
- Rename the table

ALTER TABLE - Syntax

alter table nomtable *modification*

With *modification*

- **add [column]** definitioncol
- **alter [column]** colname **type** type
- **alter [column]** colname **set default** expression
- **drop [column]** colname
- **rename to** tablename
- **add [constraint** constraintname constraintdef
- **drop constraint** constraintname

ALTER TABLE - exemples

- **alter table** bar **add** website **varchar(20)** ;
- **alter table** bar **alter column** address **varchar(150) not null**
- **alter table** bar **add constraint** uniq_license **unique** (license)
- **alter table** bar **drop constraint** uniq_license

Contenu

Data Management

- Insert data into tables :
  - `insert into ... values ...`
  - `insert into ... select ...`
- Data update
  - `update ... set ...`
- Data deletion
  - delete from ...

Insert Data (1)

- `insert into table [(col1, col2, ..., coln)] values (val1, val2, ..., valn)`
- Definition of columns optional (values given in the same order as the attributes)
- Values that are not defined will have the default value, or NULL if not set

Example : `bars(name, address, license, openDate)`

- `insert into bar values ("Australia Hotel", "The Rocks", 123456, '1940-12-01') ;`
- `insert into bar (name, license) values ("Regent Hotel", 987654) ;`

Insert Data (2)

- `insert into table [(col1, col2, ..., coln)] select ...`
- Insert data from another table
- The number of columns et their type have to be compatible with the target table

Data Update

- With a list of values :
  - **update** table **set** col1 = expression, [, col2 = expression] [**where** criteria]
  - Example : **update** drinker **set** address="The Rocks" **where** name="John" ;
- Using a query :
  - The expression can be a query returning a row and a column :

```
update sells
set price=(select price
           from sells
           where beer="Pale Ale"
           and bar="Regent Hotel);
```

Delete Data

`delete from table [where criteria]`

If no criteria, to delete all entries from a table :

`truncate table my_table`

Special case : Foreign keys

- Create/insert :
  - Parent table
  - Child table
- Delete :
  - Child table
  - Parent table

Observation : Foreign Keys cases (1)

```
SQL> insert into Bars values ('Le bon coin', 'Grenoble', 'ertoifn',
to_date ('29/02/2011','DD/MM/YYYY'));
*
ERROR at line 1:
ORA-01839: date not valid for month specified

SQL> insert into drinkers values ('Pierre', 'Grenoble');
*
ERROR at line 1:
ORA-00947: not enough values

SQL> insert into drinkers values ('Pierre', 'Grenoble', 'rrrrr');
1 row created.

SQL> insert into drinkers values ('Pierre', 'Voiron', 'aaaaa');
*
ERROR at line 1:
ORA-00001: unique constraint (BEERS.D_C0) violated
```

Observation : Foreign Keys cases (2)

```
SQL> insert into Frequents values ('Pierre', 'Bar les amis');
insert into Frequents values ('Pierre', 'Bar les amis')
*
ERROR at line 1:
ORA-02291: integrity constraint (BEERS.F_C1) violated
- parent key not found

SQL> insert into Frequents values ('Paul', 'Coogee Bay Hotel');
insert into Frequents values ('Paul', 'Coogee Bay Hotel')
*
ERROR at line 1:
ORA-02291: integrity constraint (BEERS.F_C2) violated
- parent key not found
```

Observation : Foreign Keys cases (3)

```
SQL> delete from Drinkers where name='John';
delete from Drinkers where name='John'
*
ERROR at line 1:
ORA-02292: integrity constraint (BEERS.F_C2) violated
- child record found

SQL> update Drinkers set name='Peter' where name='Adam';
update Drinkers set name='Peter' where name='Adam'
*
ERROR at line 1:
ORA-02292: integrity constraint (BEERS.F_C2) violated
- child record found
```