9.SQL- Data Manipulation Language

SQL stands for Structured Query Language.. It is the standard language for relational databases. Basically every relational DBMS (RDBMS) implements SQL, although with slight variations.

SQL can be divided into three subsets:

- The Data Definition Language (DDL), which includes commands for creating schema objects (tables, views etc) and manipulating them. It deals only with the metadata.
- The Data Manipulation Language (DML), which includes the commands for changing or retrieving the actual data.
- The Data Control Language (DCL), which includes the commands for database administration; that is, creating users, assigning permissions etc

SQL commands are English based. SQL is NOT case sensitive, except for data inside quotes; however there is a strong stylistic convention that requires writing keywords in all uppercase. Character constants are enclosed in SINGLE QUOTES, while double quotes are reserved for spelling names which contain special characters.

Although SQL statements do NOT need to end in a semicolon, most SQL interpreters use this character (;) to represent the end of a statement. That way you can use line breaks to make your statement more readable. SQL ignores extra white spaces and line breaks.

9.1. The CREATE TABLE statement

The command to create a new table is CREATE TABLE; after the command we include the table name, and then a list of fields and constraints, separated by commas. For each field we include the field name, the data type, and any **column constraints**, that is constraints that affect only one column or field.

For example, a simple table with three fields would look like this:

```
CREATE TABLE Book (
      ISBN  CHAR(9)      PRIMARY KEY,
      Title VARCHAR(20)  UNIQUE NOT NULL,
      Pages Integer
);
```

Example 1: Simple table

In this example, ISBN is a character field, with exactly 9 characters, and is the primary key. Title is a character field with up to 20 characters, which is unique (no two rows have the same value) and cannot be null, and pages is an integer. Notice ISBN has a column constraint, specifying that it is the primary key, and title has two constraints, UNIQUE and NOT NULL.

9.1.1. Common Datatypes

Although the full set of datatypes available varies with each DBMS, the following datatypes are among the most useful and commonly available:

- **CHAR(n)** This is a fixed-length character string. If you insert a value with less than the full number of characters, it gets 'padded' with spaces. Character constants are denoted by their value, enclosed in single quotes.

- **VARCHAR(n)** This is a variable-length character string, with a maximum size specified.
- **NUMERIC(prec,scale)** This is a fixed-precision number. The first number (the precision) specifies the TOTAL number of digits available, while the second (the scale) specifies how many of those are after the decimal period, such that, for example, a field of type NUMERIC(3,2) could range from -9.99 to 9.99.
- **DATE, TIMESTAMP** These are datatypes for representing moments in time, with varying precision or granularity. Depending on the DBMS there may be several variation on the date datatype.
- **FLOAT(p)** This are floating point values, with at least p binary digits of precision.
- **INTEGER** Most DBMSs will map this type to either a traditional binary integer (like an int in C or Java) or to some fixed-precision value.

9.1.2. Constraints

Many times the basic datatypes are too permissive for our purposes, since we have more information about each field; in that case, we want to constrain the allowable values. SQL provides a rich set of constraints (like PRIMARY KEY, UNIQUE and NOT NULL, introduced in Example 1).

Constraints in SQL are usually created as part of the CREATE TABLE statement (although many DBMSs support a separate CREATE CONSTRAINT statement); within the CREATE TABLE statement, the syntax we use for the constraint can be per field (in what we call column constraints) or for the whole table (table constraints). The difference is purely syntactical, the two kinds of constraints are equivalent.

The table constraints are added after all fields in the CREATE TABLE statement. Example 2 illustrate the different syntax; it creates a table equivalent to the one on Example 1, but defining them as table constraints instead of field constraints.

```
CREATE TABLE Book (
      ISBN  CHAR(9)     ,
      Title VARCHAR(20) NOT NULL,
      Pages Integer,
      CONSTRAINT Book_PK PRIMARY KEY(Isbn),
      CONSTRAINT Book_title UNIQUE (Title)
);
```

Example 2: Simple table with table constraints rather than column constraints

Notice that this notation allows us to give a name to each constraint (Book_PK and Book_title in our case), and also that NOT NULL has to be expressed as a column constraint.

SQL constraints include the following:

- **NOT NULL**
- **PRIMARY KEY**
- **UNIQUE**
- **FOREIGN KEY ... REFERENCES** this constraint allows us to express a foreign key.
- **CHECK** This constraint allows us to add an arbitrary predicate to be tested

We can also assign a default value to a column, by adding the keyword DEFAULT and then the value after the column, as if it was a constraint.

A more complete example for a SQL create table follows:

```
CREATE TABLE Student (
      Id     INTEGER     PRIMARY KEY,
      SSN    CHAR(9)     UNIQUE,
      Name   VARCHAR(20) NOT NULL,
      Age    INTEGER     DEFAULT 18
                         CHECK (Age BETWEEN 10 AND 100),
      Gender CHAR(1) CHECK(Gender='F' OR Gender='M')
      Major  CHAR(3) REFERENCES Major(Id)
);
```

Example 3: More constraint examples

If we want to do composite primary keys, and composite foreign key references, we need to express them as table constraints, since they constrain more than one column; table constraints go at the end of a table declaration, separated by commas; their basic syntax involves the keyword CONSTRAINT, an optional name given to the constraint, then the type of constraint, and then specific parameters depending on the constraint; when we need to specify a list of fields, we use parenthesis to delimit the list, and commas to separate each field on the list.

The following example illustrates several kinds of table constraints, at the end of the CREATE TABLE statement, the first one, named Standing_PK, specifies the composite primary key; the next one, Standing_Unique_Designation specifies that the combination of deg_code and designation needs to be unique , and the last one, Standing_Min_Max that the min_cr field needs to be less than the max_cr field.

```
CREATE TABLE Standing (
  deg_code      char(2) REFERENCES Degree(deg_code),
  min_cr        INTEGER DEFAULT 0 NOT NULL,
  max_cr        INTEGER NOT NULL,
  num           INTEGER NOT NULL,
  designation   VARCHAR(20) NOT NULL,
  CONSTRAINT Standing_PK
        PRIMARY KEY (deg_code, num),
  CONSTRAINT Standing_Unique_Designation
        UNIQUE (deg_code, designation),
  CONSTRAINT Standing_min_max
        CHECK (min_cr <= max_cr)
);
```

Example 4: Composite primary keys and foreign key references

9.1.2.1. More about Foreign Keys

When we specify a foreign key reference, the constraint may be violated not just by changes to that table, but also by changes to the referenced table; for example, the Major field of the Student table in Example 3 references the Id field of the major table; if the major table contains two rows, say, with ids of 'CS', and 'BW', the rows of the student table could contain any of those values in their major column, and this constraint would be enforced by the DBMS (so if we try to insert a row with different values, we would get an error).

Now imagine the row with id 'BW, is deleted from the major table; what should happen to the rows of the student table that have that major ? Intuitively, we have three possibilities; we could signal an error and disallow the deletion from the major table; we could delete the corresponding rows from the student table (probably not a good idea for this case) or we could set their major to some other value that makes sense.

SQL allows us to specify what happens to the referencing rows when a referenced row is deleted or its primary key changed (updated); after the REFERENCES constraint we can write ON DELETE (or ON UPDATE) and then one of the four following options: RESTRICT (which doesn't allow the delete to happen) CASCADE (which propagates the deletion or change to corresponding columns of the referencing table), SET NULL (which would set the referencing field on the referencing table to null) and SET DEFAULT (which would set the referencing field to its default value).

9.1.3. Further Examples

Consider the relational diagram of Figure 1, and assume that the CD number is a simple integer and that title, performer and song are strings that can vary widely in length but will never exceed 50 characters, and that those fields are always required (so NULL is not allowed). The corresponding SQL schema would look like Example 5.

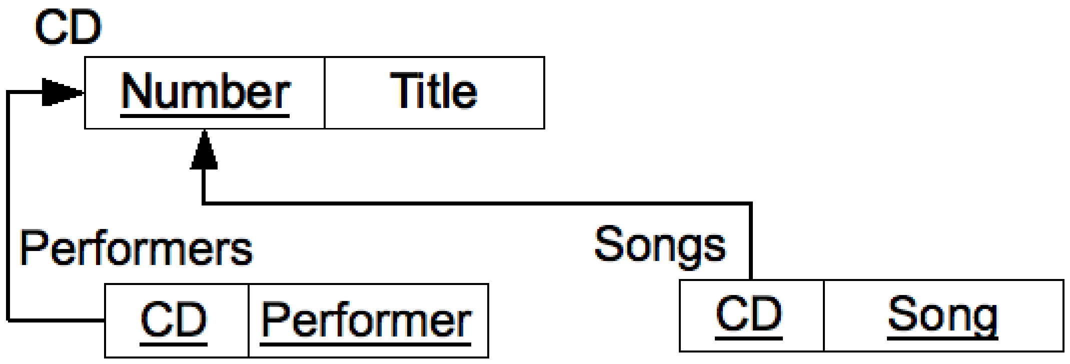


Figure 1: Sample schema for CDs

```
CREATE  TABLE Cd (
     Number INTEGER PRIMARY KEY,
     Title VARCHAR(50) NOT NULL
);

CREATE TABLE Performers (
     Cd INTEGER REFERENCES Cd(Number),
     Performer VARCHAR(50),
     CONSTRAINT Performers_PK PRIMARY KEY(Cd,Performer)
);

CREATE TABLE Songs (
     Cd INTEGER REFERENCES Cd(Number),
     Song VARCHAR(50),
     CONSTRAINT Songs_PK PRIMARY KEY (Cd, Performer)
);
```

Example 5: SQL Schema for the CD example

9.1.3.1. Products and categories, 1:m relationships

Now let's look at the ER model of figure 2, which represents products and categories, where a product belongs to at most one category; the corresponding relational schema is shown in Figure 3, and the SQL schema (with some assumptions made about the appropriate data types) is shown in Example 6.

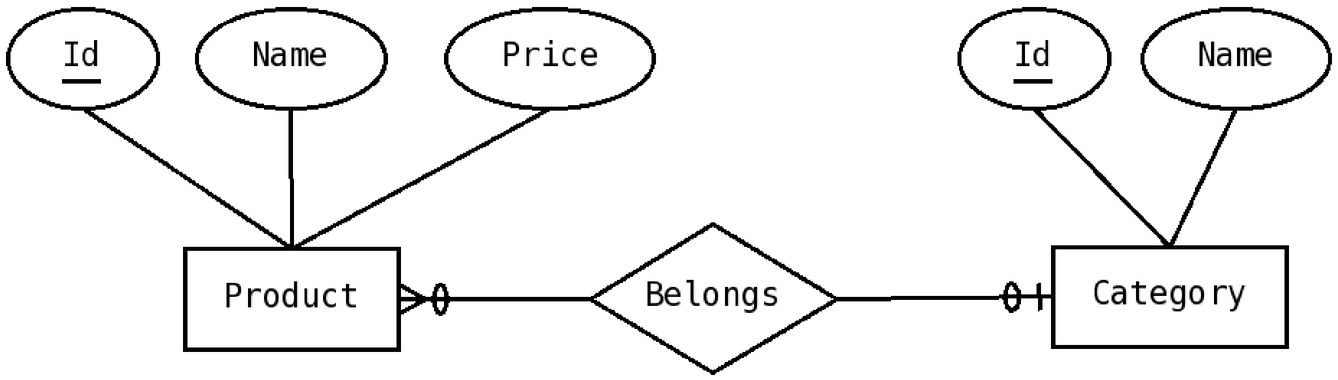


Figure 2: ER diagram for products where the product belongs to at most one category

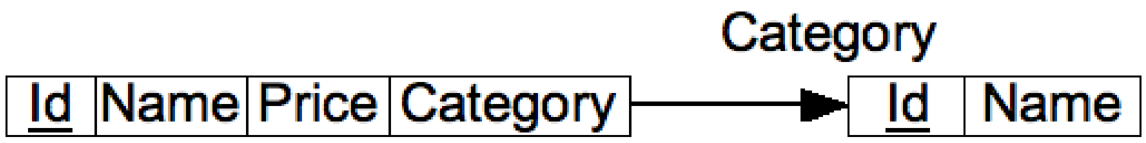


Figure 3: Relational schema for products and categories, where the product belongs to at most one category (corresponds to the ER diagram in Figure 2)

```
CREATE TABLE Category (
     Id INTEGER PRIMARY KEY,
     Name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE Product (
     Id INTEGER PRIMARY KEY,
     Name VARCHAR(50) UNIQUE NOT NULL,
     Price NUMERIC(7,2) NOT NULL,
     Category INTEGER REFERENCES Category(Id) ON DELETE SET NULL
);
```

Example 6: SQL schema for products and categories, one category per product

9.2. Views

Views allow us to create database objects that appear to be tables but whose values are calculated by the DBMS instead of being explicitly entered by the users; in a way, they are can be viewed as queries which are given a name.

Views are created with the CREATE VIEW statement; we write CREATE VIEW, then the name of the view, then the keyword AS and then a SELECT statement that specifies the values that will go in the view.

So, assume we have a Student table like in Example 1, with a column for Gender; we could define a view, called female_students that only includes those rows with a gender of 'F' (so only female students), with the SQL statement of Example 7.

```
CREATE VIEW FemaleStudents
AS
SELECT *
FROM Student
WHERE Gender='F';
```

Example 7: SQL schema for products and categories, one category per product

And after doing so we can use FemaleStudents in a SELECT statement as if it was a table, and the values returned will be calculated based on the rows of the Student table.

9.2.1. View Updates

One problem with views is how to deal with updates; for example, we could try to insert a row into the femaleStudents table; an intuitive solution would be to just insert a row into the student table; but what if we insert a row with a gender of 'M' ?

In general, there's no good solution for how to deal with view updates, especially since views can be defined with very complicated queries, for which no base table corresponds, and various DBMSs have slightly different policies on which views can be updated. We recommend to avoid updating any views if at all possible.

9.3. Other DDL Statements

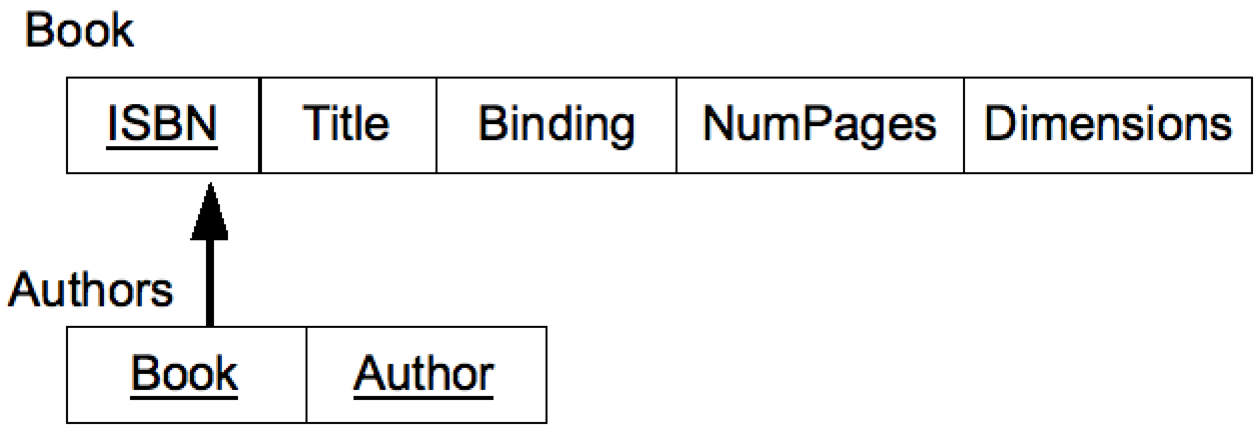
Other DDL statements include:

- **DROP TABLE** that allows us to completely eliminate a table (not just the data, but also the schema)
- **ALTER TABLE** that allows us to modify the schema of a table

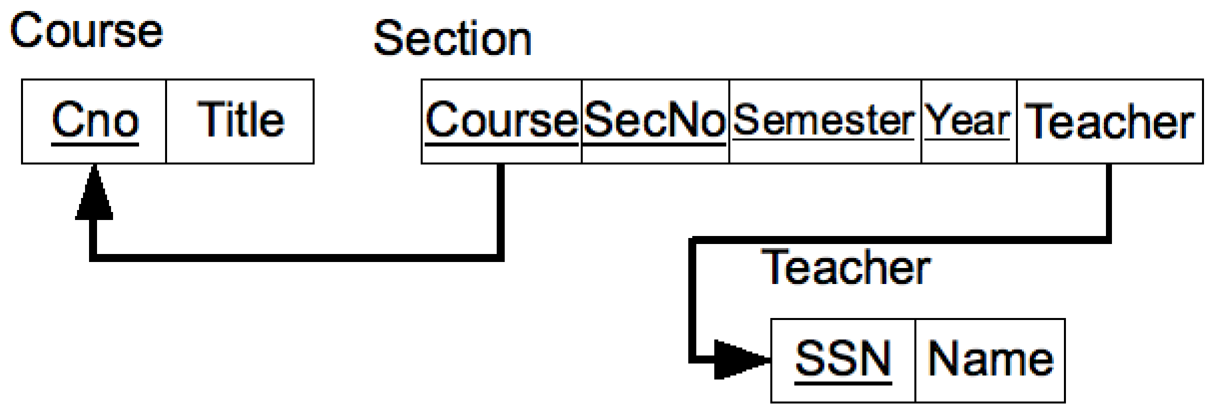
We also can create other kinds of schema objects, including views and indexes. In general, to create a schema object of a given kind, we will use CREATE and then the kind of object (TABLE,VIEW, INDEX etc), the name, and then syntax specific for that kind of object. To eliminate that object we use DROP, then the type (TABLE, VIEW, INDEX ...) and the name of the object.

9.4. Exercises

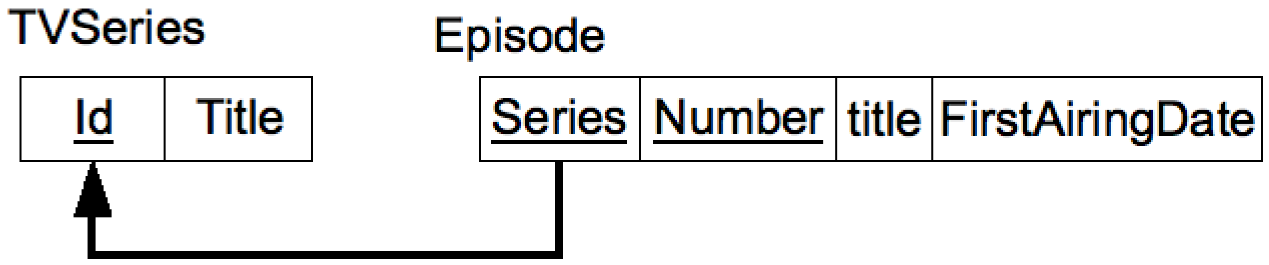
1. Given the following relational schema, create a corresponding SQL schema, making reasonable assumptions as to data types.



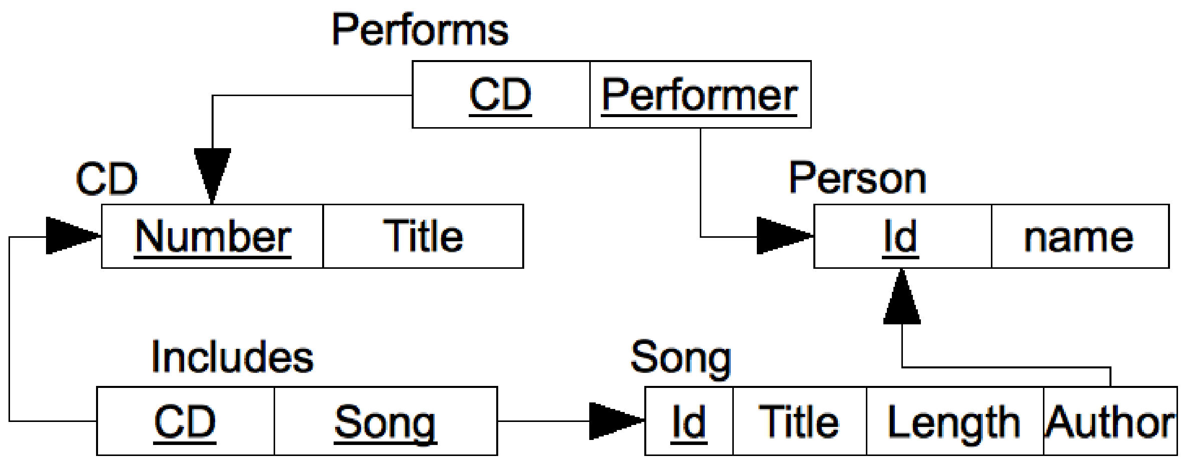
2. Given the following relational schema, create a corresponding SQL schema, making reasonable assumptions as to data types.



3. Given the following relational schema, create a corresponding SQL schema, making reasonable assumptions as to data types.



4. Given the following relational schema, create a corresponding SQL schema, making reasonable assumptions as to data types.


Performs
CD
Performer
CD
Number
Title
Person
Id
name
Includes
CD
Song
Song
Id
Title
Length
Author