SQL 1: CREATE and DROP Tables

Databases and Interfaces

Matthew Pike & Yuan Yao

University of Nottingham Ningbo China (UNNC)

Overview

This Lecture

- What is a DBMS?
- What is SQL?
- How can we use SQL to:
    - `CREATE` a table in a database
    - Create a relationship between tables
    - `DROP` (delete) a table from a database

SQL and Database Management Systems (DBMS)

Database Management Systems (DBMS)

- A DBMS is a collection of programs that enables users to create and maintain a database.
    - SQL is (often) the language used to communicate with the DBMS.
- A DBMS will also provide an interface for programming languages to interact with the database
- A DBMS provides additional functions like concurrency, transactions, etc
- Examples of DBMSs, include:
    - SQLite
    - MariaDB
    - MySQL

SQL

- SQL is a standard language for accessing and manipulating databases.
- SQL is a declarative language, meaning that you specify what you want, not how to get it.
- Not all DBMS implementations are equal - some may support more features than others.
- SQL is an international standard (ANSI & ISO)

> Caution
>
> Although SQL is a standard, it is not supported *exactly* the same way by all DBMSs. In practice you will need to update your SQL queries to work with different DBMSs.

SQL Sublanguages

- SQL provides many types of operations for creating, selecting, updating and removing data in the database.
- There are sublanguages of SQL for performing types of operations:
    - Data Definition Language (DDL)
        - Syntax for creating and modifying database objects, such as tables and indices
    - Data Manipulation Language (DML)
        - Insert, retrieve and manipulate data in a database
    - Data Control Language (DCL)
        - Control security and concurrent access in a database

Creating Tables with CREATE in SQL

Terminology

- We have already looked at Relational and ER representations of data.
- Now, we will look at how to realize these designs in a real (relational) database, using SQL.
- Table 1 provides a mapping between the terminology used between different representations.

Relations	E/R Diagrams	Relational Databases
Relation	Entity	Table
Tuple	Instance	Row
Attribute	Attribute	Column/Field
Foreign Key	M:1 Relationship	Foreign Key
Primary Key	Attribute	Primary Key

Table 1: Terminology Mapping between Relational, E/R and Relational Databases

Going From E/R to SQL

Primary Keys are underlined in the E/R diagram.

- Write the SQL **CREATE** statement to create a table for the E/R diagram in Figure 1.
- To do this, we need to:
  1. Translate Entities into Tables
  2. Translate Attributes into Columns
  3. Approximate attribute domains by assigning data types to Columns
  4. Translate relationships into Foreign Keys

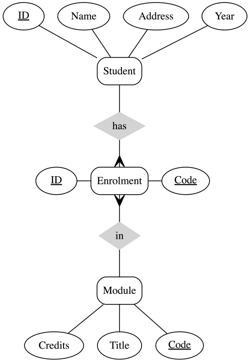

Figure 1: E/R Diagram for Student Module Enrolment

Example: Student Table

Goal

Create a table in SQL to represent the `Student` entity in Figure 2. Student IDs are unique and cannot be `NULL`. Addresses are optional and can be `NULL`. If not specified, the `Year` of study defaults to 1.

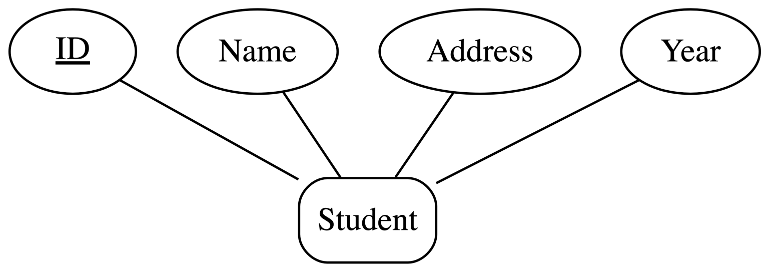

Figure 2: ER Diagram for Student Table

Step 1: Translate Entities to Tables

```
CREATE TABLE Student(
    ...
);
```

Step 2: Attributes of an Entity become Columns

```
CREATE TABLE Student (
    sID ,
    sName,
    sAddress,
    sYear
);
```

Step 3: Assign Types to Columns

```sql
CREATE TABLE Student (
    sID INTEGER,
    sName VARCHAR(50),      -- Reasonable?
    sAddress VARCHAR(255),  -- Reasonable?
    sYear INTEGER
);
```

Step 4: Add constraints

Note

Both commands below are equivalent

```
CREATE TABLE Student (
    sID INTEGER
        NOT NULL PRIMARY KEY,
    sName VARCHAR(50) NOT NULL,
    sAddress VARCHAR(255),
    sYear INTEGER DEFAULT 1
);
```

```
CREATE TABLE Student (
    sID INTEGER NOT NULL,
    sName VARCHAR(50) NOT NULL,
    sAddress VARCHAR(255),
    sYear INTEGER DEFAULT 1,
    CONSTRAINT pk_student
        PRIMARY KEY (sID)
);
```

Constraints

- Constraints are used to specify rules for the data that can be stored in a table.
- Constraints can be used to specify that a column cannot contain `NULL` values, or that all values must be `UNIQUE`.
- Each constraint is given a name. If you don’t specify a name, one will be generated for you.

Primary Key and Unique Constraints

- `PRIMARY KEY` constraints uniquely identify each row in a table
  - Primary keys cannot be `NULL`
  - Primary keys must be unique
  - Primary keys will typically add `NOT NULL` and `UNIQUE` constraints
- `UNIQUE` constraints ensure that all values in a column are different
  - `UNIQUE` constraints can be `NULL`
  - `UNIQUE` constraints must be unique
  - This has the same effect as a primary key constraint, except that the column(s) can contain `NULL` values
  - This effectively creates a candidate key for the table

Data Types

- SQL provides a number of data types for representing data in a database
- These include:
    - Numeric types: `INTEGER`, `REAL`, `NUMERIC`
    - Character types: `CHAR`, `VARCHAR(M)`
    - String types: `VARCHAR`, `TEXT`
    - Date and time types: `DATE`, `TIME`, `TIMESTAMP`

> **Caution**
>
> Not all data types are supported by all DBMSs, and some data types may be implemented differently by different DBMSs.

Examples of Data Types

Data Type	Description	Example
`INTEGER`	Integer value	`1`, `2`, `3`
`REAL`	Floating point value	`1.0`, `2.0`, `3.0`
`CHAR`	Fixed length string	`'a'`, `'b'`, `'c'`
`VARCHAR` or `TEXT`	Variable length string	`'a'`, `'ab'`, `'abc'`
`DATE`	Date value	`'2018-10-01'`

Table 2: Examples of data types in SQL

Types in SQLite

- Most SQL DBMSs uses *static*, rigid typing
    - With static typing a value's datatype is determined by the column in which the value is stored.
- SQLite uses a more general *dynamic* type system.
    - The datatype of a value is associated with the value itself, not with its column's datatype.
- SQLite 3 defines 5 affinity types, to which a column's datatype will be assigned:
    - `TEXT`, `NUMERIC`, `INTEGER`, `REAL`, `BLOB`

SQLite Types

More information on SQLite types can be found: https://www.sqlite.org/datatype3.html

Another Example: Module Table (1/2)

Module Table

The **`Module`** table stores information about modules offered by the university. Each module has a unique 8 character module code, a title and a credit value.

```
CREATE TABLE Module (
    ...
);
```

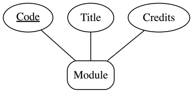

Figure 3: ER Diagram for the Module Table

Another Example: Module Table (2/2)

The `DEFAULT` clause

The `DEFAULT` clause can be used to specify a default value for a column. If no value is specified for a column when a new row is inserted into the table, the default value will be used instead.

```sql
CREATE TABLE Module (
    mCode CHAR(8) NOT NULL PRIMARY KEY,
    mTitle VARCHAR(100) NOT NULL,
    mCredits INTEGER NOT NULL DEFAULT 10
);
```

Relationships

Example: Student Module Enrolment

Linking Tables

How do we link the `Student` and `Module` tables to the `Enrolment` table?

- Currently, we have two tables: `Student` and `Module`
- We need to add a table, `Enrolment` to represent the relationship between students and modules
- This table will have two columns: `sID` and `mCode` to the `Enrolment` table

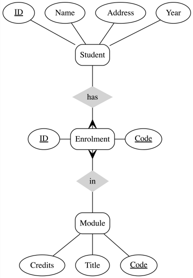

Figure 4: ER Diagram for the Student Module Enrolment example

Foreign Keys

- Foreign keys are used to create *relationships* between tables
- M:1 relationship: Represented by a foreign key in the *many* table
- M:M relationship: are split into two 1:M relationships
  - A table is used to represent the relationship between the two tables
- The `Enrolment` table:
  - Represents the relationship between `Student` and `Module`
  - Has two foreign keys: `sID` and `mCode`
  - The `sID` column is a foreign key to the `Student` table
  - The `mCode` column is a foreign key to the `Module` table

Example: Add Columns to Enrolment Table

```
CREATE TABLE Enrolment (
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL
    ...
);
```

Example: Add Foreign Keys

```
CREATE TABLE Enrolment (
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL,
    PRIMARY KEY (sID, mCode),
    FOREIGN KEY (sID)
        REFERENCES Student(sID),
    FOREIGN KEY (mCode)
        REFERENCES Module(mCode)
);
```

- Foreign Keys are also defined as *constraints*
- You need to specify the name of the table and the column that the foreign key references

Example: Add Referential Integrity Constraints

```
CREATE TABLE Enrolment (
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL,
    PRIMARY KEY (sID, mCode),
    CONSTRAINT en_fk1
        FOREIGN KEY (sID) REFERENCES Student(sID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT en_fk2
        FOREIGN KEY (mCode) REFERENCES Module(mCode)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Referential Integrity Constraints

- Referential integrity constraints can be specified for each foreign key
- When relations are updated or deleted, constraints are checked
- There are three options:
    - `RESTRICT`: The database will not allow the update or delete to proceed if it would break referential integrity
    - `CASCADE`: The database will update/delete related rows in the other table
    - `SET NULL`: The database will set the foreign key to `NULL` in the related row in the other table

⚠ SQLite Foreign Key Constraints

By default, SQLite does not enforce foreign key constraints. You need to enable them using the `PRAGMA` statement:

```
PRAGMA foreign_keys = ON;
```

Deleting Tables using `DROP`

Deleting Tables

- You can delete tables with the **DROP** keyword:
  - `DROP TABLE [IF EXISTS] table-name;`
- For example:
  - `DROP TABLE IF EXISTS Student;`
- Foreign Key constraints will prevent you from deleting a table if it is referenced by another table.
  - You can delete the referencing table first, then the referenced table

❗ Caution

Be **very careful** with this command. It will delete the table and all its data. There is no ‘undo’.

Exercise

Pilot Qualification Database

Problem Description

A pilot can be qualified to fly multiple aircraft, and an aircraft can be flown by multiple pilots. All pilots must have a name and age. All pilots begin with 1 year of experience (from training). All aircraft must have all attributes.

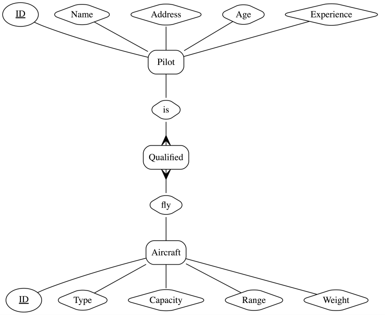

Figure 5: ER Diagram for the Pilot Qualification example

Reference Section

CREATE Table Definition

```
CREATE TABLE table-name (
    col-name-1 col-def-1,
    col-name-2 col-def-2,
    ...
    col-name-n col-def-n,
    constraint-1,
    ...
    constraint-k
);
```

- `table-name` is the name of the table to be created
- `col-name-n` is the name of the n-th column
- `col-def-n` is the definition of the n-th column
- `constraint-k` is the k-th constraint on the table

CREATE Column Definition

```
col-name col-def
    [NULL | NOT NULL]
    [DEFAULT default_value]
    [NOT NULL | NULL]
    [AUTO_INCREMENT]
    [UNIQUE]
    [PRIMARY KEY]
```

- `col-name` is the name of the column
- `col-def` is the definition of the column
- `NULL` or `NOT NULL`: whether the column can contain `NULL` values
- `DEFAULT default_value`: specifies a default value for the column
- `AUTO_INCREMENT`: column is an auto-incrementing integer
- `UNIQUE`: must contain unique values
- `PRIMARY KEY`: column is a primary key

Non-Exhaustive List of Column Constraints

More information here: https://www.sqlite.org/lang_createtable.html

Foreign Key Constraints

```
CONSTRAINT name
    FOREIGN KEY
        (col1, col2, ...)
    REFERENCES
        table-name
        (col1, col2, ...)
    ON UPDATE ref_opt
    ON DELETE ref_opt
```

- You need to provide:
    - A name for the constraint
    - The name of the column(s) in the referencing table
    - The name of the table being referenced
    - The name of the column(s) in the referenced table
    - The action to take when the referenced row is updated
    - The action to take when the referenced row is deleted
- `ref_opt` can be : `RESTRICT` | `CASCADE` | `SET NULL` | `SET DEFAULT`

SQLite Dot Commands

- The SQLite Command Line Interface (CLI) has special commands dot commands `.`
- `.` commands control the behaviour of the CLI
- The most useful commands are:
  - `.help` - Display a list of commands
  - `.tables` - Display a list of tables
  - `.import` - Import data from a file into a table
  - `.read` - Execute commands from a file
  - `.schema` - Display the schema of a table
  - `.quit` - Exit the command line tool

SQLite dot commands

More information here: https://www.sqlite.org/cli.html