SQL 1: CREATE and DROP Tables

Databases and Interfaces

Matthew Pike & Yuan Yao

University of Nottingham Ningbo China (UNNC)

Overview

This Lecture

In this lecture, we will look at:

- Review - what is a DBMS?
- What is SQL? What does it allow us to do?
- How we can use SQL to:
    - `CREATE` tables in a database
    - Link tables together using `FOREIGN KEY` constraints
    - `DROP` (delete) a table from a database

DBMSs and SQL

Database Management Systems (DBMS)

A DBMS is a collection of programs that enables users to create, maintain, and interact with a database. Some key aspects of a DBMS include:

- A structured way to organise, store, and retrieve data.
- A language (often - SQL) to query and manipulate the data in the database.
- An administrative interface (often a CLI) to manage the DBMS.
- A programmatic interface (API) for applications to interact with the database.
- Critical functions like security, concurrency control, transaction management, crash recovery - to preserve data integrity.
- Examples of DBMSs, include:
    - SQLite
    - MariaDB
    - MySQL

SQL - Structured Query Language

SQL is a standard language for managing data in a relational database which builds upon Edgar F. Codd's relational model (Codd 1970). Some key aspects of SQL include:

- SQL is a declarative language, meaning that you specify what you want, not how to get it.
  - This is in contrast to imperative languages, which require you to specify the exact steps to achieve your desired outcome.
- Statements are not necessarily run/executed in the order they are written.
  - There are however rules about the order in which statements are declared.
- Example:
  - `SELECT * FROM Student;` - Retrieve all data from the `Student` table
  - `SELECT * FROM Student WHERE SID < 100;` - Retrieve all data from the `Student` table where the `SID` is less than 100.

- SQL became a standard of the:
  - American National Standards Institute (ANSI) in 1986;
  - International Organization for Standardization (ISO) in 1987.

SQL and DBMS Interoperability

Keep in mind that while SQL is a standard, it is not supported *exactly* the same way by all DBMSs. In practice, you may need to update your SQL queries to work with different DBMSs.

Consider:

> *"The folding of unquoted names to lower case in PostgreSQL is incompatible with the SQL standard, which says that unquoted names should be folded to upper case. Thus,* `Foo` *should be equivalent to* `FOO` *not* `foo` *according to the standard." (Wikipedia 2023)*

SQL Sublanguages

SQL consists of many types of operations for creating, selecting, updating and removing data in the database. Informally, we can divide SQL into three sublanguages:

1. **Data Definition Language (DDL)** - used for creating and modifying database objects, such as tables, indices, and other structural elements. DDL statements define the structure and organisation of the data in the database.
2. **Data Manipulation Language (DML)** - DML is used for inserting, retrieving, and manipulating data in a database.
3. **Data Control Language (DCL)** - DCL is used for controlling security and concurrent access to a database. It includes statements for granting and revoking privileges, creating and dropping user accounts, and managing transaction locking and isolation.

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 realise these designs in a real (relational) database, using SQL.
- Table 1 provides a mapping of the terminology used between different representations.

Relations	ER Diagrams (ERD)	Relational Databases
Relation	Entity	Table
Tuple	Instance	Row
Attribute	Attribute	Column/Field
Foreign Key	M:1 Relationship	Foreign Key
Primary Key	<u>Attribute</u>	Primary Key

Table 1: Terminology mapping between Relational, ERD and Relational Databases

- Goal:
    - Given an ERD (such as Figure 1), create a relational database using SQL to represent the structure of the data.
- 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.

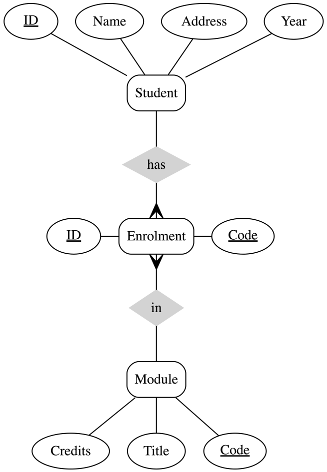

Figure 1: ERD for Student Module Enrolment

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.

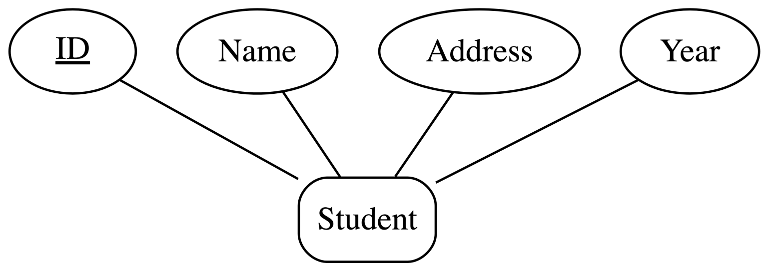

Figure 2: ER Diagram for Student Table

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
);
```

Comments in SQL

Just as with other programming languages, SQL supports comments. Comments are ignored by the DBMS and are used to document your code.

- Single line comments start with `--`
- Multi-line comments start with `/*` and end with `*/`

Step 4: Add constraints

> **Note**
>
> Both SQL statements below are equivalent to one another.

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

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

Constraints

- Constraints are an essential aspect of database design, as they enforce rules on the data stored in a table.
    - These rules ensure that the data is consistent, accurate, and reliable.
- For example:
    - Constraints can be used to specify that a column cannot contain `NULL` values, or that all values must be `UNIQUE`.
- You can specify a name for each constraint, which makes it easier to reference and manage them.
    - We saw in the previous example - a constraint named `pk_student`.
    - If you don't specify a name, one will be generated for you.
    - Naming constraints is good practice and should be done whenever possible.

Primary Key and Unique Constraints

In SQL, `PRIMARY KEY` and `UNIQUE` constraints are used to enforce uniqueness and non-nullness on columns or sets of columns in a table.

- A `PRIMARY KEY` constraint uniquely identifies each row in a table. It is a column or set of columns that cannot contain `NULL` values and must contain unique values for each row.
- A `UNIQUE` constraint ensures that all values in a column are different.
  - It is similar to a `PRIMARY KEY` constraint, but it can contain `NULL` values.

> SQLite allows `NULL` values in `PRIMARY KEY` columns!
>
> "According to the SQL standard, `PRIMARY KEY` should always imply `NOT NULL`. Unfortunately, due to a bug in some early versions, this is not the case in SQLite. Unless the column is an `INTEGER PRIMARY KEY` or the table is a `WITHOUT ROWID` table or a `STRICT` table or the column is declared `NOT NULL`, SQLite allows `NULL` values in a `PRIMARY KEY` column."
> - From https://www.sqlite.org/lang_createtable.html#primkeyconst.

Types in SQL

Data Types

Data Types are DBMS Dependent

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

- 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`

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

SQLite Types

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

- 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`

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 (
    ...
);
```

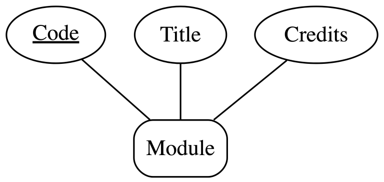

Figure 3: ER Diagram for the Module Table

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) PRIMARY KEY,
    mTitle VARCHAR(100) NOT NULL,
    mCredits INTEGER NOT NULL DEFAULT 10
);
```

Relationships

Example: Student-Module-Enrolment

- Currently, we have two tables:
  - `Student`
  - `Module`
- We need to add a table, `Enrolment` to represent the relationship between `Student` and `Module`.
- This table will have two columns:
  - `sID` – references the primary key in the `Student` table.
  - `mCode` – references the primary key in the `Module` table.

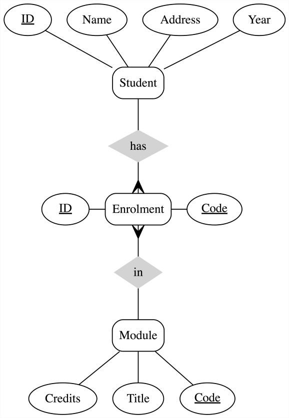

Figure 4: ERD for the Student, Module and Enrolment example

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.
    - This table is called a *link* or *junction* table.
- Why Foreign Keys are important:
    - Relationship building - Allow us to link data between tables.
    - Data Integrity - Relationships between tables are accurate and consistent.
    - Data Consistency - Data updates are propagated to all related tables.

> ⚠ Foreign Keys must reference a `UNIQUE` (typically `PRIMARY KEY`) column
>
> A foreign key must reference a `UNIQUE` column in the referenced table, otherwise foreign key constraints cannot be enforced.

Example: Add Columns to Enrolment Table

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

Module Table Definition

We haven't defined the `Module` table yet in the lecture slides. We will do this later in the lecture.

Example: Adding Foreign Keys

```
CREATE TABLE Enrolment (
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL,
    -- Composite Primary Key
    PRIMARY KEY (sID, mCode),
     -- Specify that sID is a foreign key
    FOREIGN KEY (sID)
        -- References the Student table
        REFERENCES Student(sID),
    FOREIGN KEY (mCode)
        REFERENCES Module(mCode)
);
```

- The `FOREIGN KEY` constraint specifies that the values in the column(s) must match values in the referenced column(s).
- The `REFERENCES` keyword specifies the table and column(s) that the foreign key references.
- The referenced column(s) must be a `PRIMARY KEY` or have the `UNIQUE` constraint.

Visualising Foreign Key Relationships

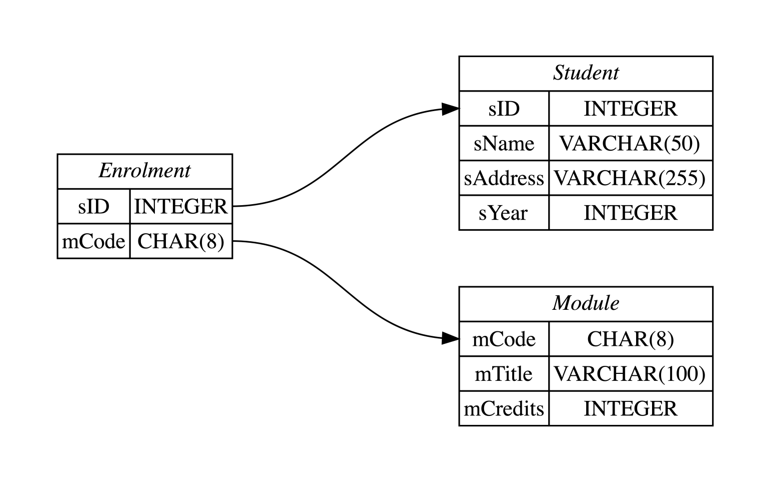

Figure 5: Visualisation of the foreign key relationships between the `Student`, `Module` and `Enrolment` tables.

Referential Integrity Constraints

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;
```

- 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

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
);
```

Deleting Tables using DROP

Deleting Tables

Practice Caution using DROP

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

- 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
    - Although, by default, SQLite does not enforce foreign key constraints. You need to enable them using the `PRAGMA` statement:
        - `PRAGMA foreign_keys = ON;`

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

Non-Exhaustive List of Column Constraints

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

```
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

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

SQLite dot commands

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

- 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

Problem Description

A pilot can be qualified to fly multiple aircraft, and an aircraft can be flown by many 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.

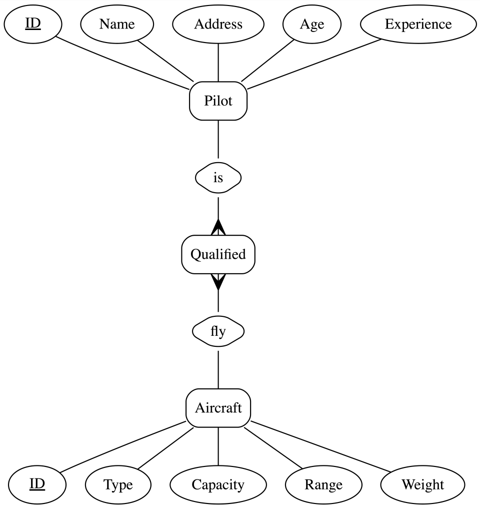

Figure 6: ERD for the Pilot Qualification example

Codd, Edgar F. 1970. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM* 13 (6): 377–87.

Wikipedia. 2023. "SQL — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/wiki/SQL.