G64DBS Exercise 1: Designing Databases

Introduction

During this exercise we will cover how to design and create a database from a problem description; how to identify entities, draw an E/R diagram and how to create the necessary tables in SQL. Designing a database before you create any tables is an important step; it ensures problems when querying are kept to a minimum.

A Problem Description

A database is to be made to store information about a catalogue of CDs. Information to be stored about each CD includes their title, price, genre, and a list of tracks. Each CD will also have an artist, and each artist may produce several CDs. Tracks will have a title and a running time (in seconds). Artists have names associated with them and it should be possible to search the database by artist names.

Database Design

There are several steps in database design. Following the steps below will lead you through one common approach:

1. Identify entities, attributes and relationships from the problem description.
2. Construct an E/R diagram to represent the results of step 1.
3. Look for any issues that are apparent in the E/R diagram, and resolve them. In particular, try to remove many-to-many relationships.
4. Create database tables to represent the information contained in your E/R diagram.

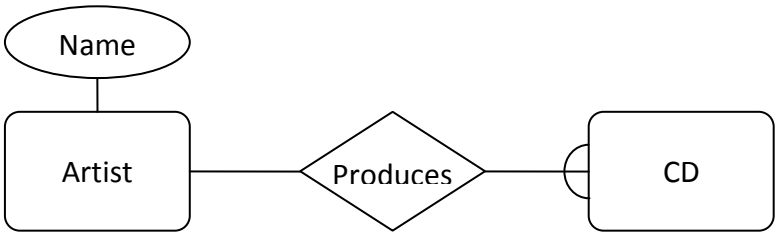
Identifying Entities, Attributes and Relationships

The first step in database design is to identify entities, attributes and relationships from the problem description. It is also necessary to identify cardinality ratios of the relationships found. For example, CD and Artist are two entities from the problem above. Artist has an attribute name. There is also a relationship between artists and CDs; artists produce CDs. This is a one-to-many relationship.

Exercise: Identify the other entities, attributes, relationships and cardinality ratios from the problem description above.

DRAWING AN E/R DIAGRAM

Once you have identified all the entities etc. from the description it is helpful to draw an E/R diagram. This can give you a useful picture of the layout of the database. Using the information above, we can produce a partial E/R diagram:



Exercise: Complete the E/R diagram with the additional information you have identified from the problem description.

PROBLEMS IN E/R DIAGRAMS

The E/R diagram provides and early tool for diagnosing some design issues. It makes the structure of your planned database explicit, and lets you think about it before you start writing SQL. There are several key points to look for:

- Many-to-many relationships are troublesome to represent in SQL, they should be split into two one-to-many relationships and an intermediate entity (which will become a linking table between the original entities).
- One-to-one relationships might represent an unnecessary division of tables. If you have a one-to-one relationship between entities A and B you might be able to remove the relationship by replacing A and B with a single entity.
- You can start to run through example queries using the E/R diagram. For example, using the partial diagram above we can see how to find all the CDs by a given artist. First we'd find the entry in the Artist table we're interested in, and we'd then follow the produces relationship to find any relevant CDs.

Exercise: If there are any many-to-many relationships in your E/R diagram, split them into two one-to-many relationships.

Exercise: If there are any one-to-one relationships in your E/R diagram, consider merging them into a single entity.

CREATING SQL TABLES

Once you are happy with your design, it is time to start writing some SQL. Before this you must ensure you have set up a MySQL account on the School's servers, following the procedure outlined in the Introduction Exercise.

To implement the database design in SQL you will need to create a table for each entity. A table representing an entity should have:

- A column for each attribute, with appropriate data type and NULL / NOT NULL specified. The default value is NULL if you don't specify anything here.
- A primary key and possibly some candidate keys. If you do not specify a primary key, MySQL may choose one for you if required. You should always specify one to avoid this problem. A primary key is always NOT NULL and UNIQUE, so you don't need to specify these values.

It is often useful to have:

- An ID column (INT) to act as a convenient primary key.
- A prefix for the table, which helps you relate columns to tables in queries.

For the artist table we might choose the prefix `art`, so an ID column would be called `artID`. A name column might be called `artName`. The ID would likely be an integer value, and the name would be a variable length string, `VARCHAR(255)`.

Each column should be set as being `NULL`, meaning it is not required, or `NOT NULL`, indicating a value must be given. A primary key must be unique, so `artID` should be `NOT NULL`. For `artName` it is a little less clear, however it doesn't make much sense to have an artist with no name, so this can be `NOT NULL` too.

There are also a number of issues surrounding keys. `artID` has already been selected as the primary key, but we could have used `artName`, since every artist should have a unique name. This makes `artName` a candidate key.

In summary, to represent the entity artist:

- We will create a table called Artist
- This will have columns called artID and artName.
- The artID column will be of type INT and NOT NULL.
- The artName column will be of type VARCHAR(255) and NOT NULL.
- artID is the primary key, so there will be a PRIMARY KEY constraint on it.
- artName is a candidate key, so there will be a UNIQUE constraint on it.

This means we can use the following SQL command to create the table:

```
CREATE TABLE Artist (
  artID INT NOT NULL,
  artName VARCHAR(255) NOT NULL,
  CONSTRAINT pk_artist PRIMARY KEY (artID),
  CONSTRAINT ck_artist UNIQUE (artName)
);
```

Note that there is a semicolon at the end of the statement. Pressing return in `mysql` doesn't actually execute the command – rather it starts a new line to edit. This lets you write long commands easily, one line at a time. The semicolon signals that you've finished the command and want to run it. Some `mysql` commands do not need a semicolon, such as `exit`. If you get a few lines in and realise you have made a mistake, enter `\c` and `mysql` will discard the current command, allowing you to start again.

Next we should create a CD table. This will follow more or less the same procedure as the Artist table. We must include a foreign key however, since there is a one-to-many relationship between Artist and CD. Creating a foreign key is usually done as follows:

- A foreign key column is put into the CD table.
- If you are using prefixes, it is often helpful to prefix your column with that of the referenced table columns. This means it has exactly the same name as the column it references.
- A `FOREIGN KEY` constraint is used to tell the database about the relationship.
- You should also consider adding an `ON DELETE` clause. `ON DELETE RESTRICT` is used by default however.

Part of a table definition for the CD entity is shown below. The foreign key represents the one-to-many relationship between Artist and CD. Given a CD entry, we can use the CD.artID to find the corresponding Artist.

```
CREATE TABLE CD (
  cdID INT NOT NULL,
  artID INT NOT NULL,
  -- Probably want some other attributes in here.
  CONSTRAINT pk_cd PRIMARY KEY (cdID),
  CONSTRAINT fk_cd_art FOREIGN KEY (artID) REFERENCES Artist (artID)
);
```

Note that to create a foreign key, the artist table must already exist. Often when a database is being created, referencing tables will be created before the referenced table. In this case, it is simpler to simply split up the process of table creation, adding the foreign keys later. Like this:

```
CREATE TABLE CD (
  cdID INT NOT NULL,
  artID INT NOT NULL,
  -- Probably want some other attributes in here.
  CONSTRAINT pk_cd PRIMARY KEY (cdID),
);
```

Then, once Artist has been created:

```
ALTER TABLE CD ADD CONSTRAINT fk_cd_art FOREIGN KEY (artID)
REFERENCES Artist (artID);
```

You could also add or remove columns from a table like this:

```
ALTER TABLE tableName ADD COLUMN colName INT NOT NULL;

ALTER TABLE tableName DROP COLUMN colName;
```

Using `ALTER TABLE` you can do a variety of other things, such as adding or removing constraints.

Exercise: Create tables to represent your whole database design. You should have a table for every entity, and a foreign key for every relationship. Your tables should use the InnoDB storage engine, as discussed below.

MySQL Engines

Unlike many DBMSs, MySQL comes equipped with a variety of storage engines. These operate on tables, and instruct the system how to process the data being added and removed. The default engine, `MyISAM`, is extremely fast. However, it has no support for foreign keys. If you create a table using a foreign key, it will ignore the constraint. For this reason, during the entirety of the practicals and coursework, we will be using an alternative engine, called `InnoDB`. This engine is not as fast as `MyISAM` (not that we will notice with the small databases we are using), however it enforces foreign key constraints.

The engine choice will also affect the creation of a table, so you must ensure that InnoDB is the engine choice when the table is created too.

To create a table using the `InnoDB` storage engine, simply add `ENGINE=InnoDB` to the end of your `CREATE TABLE` command. Like this:

```
CREATE TABLE Artist (
  artID INT NOT NULL,
  artName VARCHAR(255) NOT NULL,
  CONSTRAINT pk_artist PRIMARY KEY (artID),
  CONSTRAINT ck_artist UNIQUE (artName)
) ENGINE=InnoDB;
```

In order to avoid issues with engines in the practicals and coursework, you should always follow these steps:

- Create tables with `ENGINE=InnoDB` specified.
- Avoid changing the storage engine of a table. Instead, recreate the table if the engine is incorrect. Alternatively, add foreign keys after converting to InnoDB.
- Ensure all foreign keys reference a table that is also using InnoDB.

From now onwards it will be assumed you are using the InnoDB engine for your tables.

FURTHER EXERCISES

At this point, you have completed everything required for Exercise 1. If you have more time, or would like more practice, try the following:

Exercise: Familiarise yourself with the "Useful Commands" from below and in the lectures. Knowing these will significantly speed the rate at which you can code SQL, in particular debugging issues.

Exercise: If you haven't already, move your SQL statements into a file named ex1.sql. Place this file in your home directory somewhere. You will then be able to execute this file at any time using `source ex1.sql` to create your tables. If you add DROP statements to the beginning of the file, this will prevent SQL failing to create your tables due to duplicate names.

Exercise: INSERT, UPDATE and DELETE will be introduced in the next Exercise. If you'd like some practice, try inserting some information into your table. Remember that you can use `SELECT * FROM <table-name>;` to see a list of all the rows in your table.

USEFUL COMMANDS

Here are some SQL/MySQL commands you may find useful:

```
mysql> describe tableName;
```

This command will provide an overview of your table, including columns, names, data types and keys.

```
mysql> show tables;
```

This command shows a list of names all of your tables. Other information can be seen by querying the `information_schema.tables` table.

```
mysql> SHOW CREATE TABLE tableName;
```

This command will show you the create table command for a given table. It will list all columns and constraints. This is useful if you can't remember what constraints you have on that table, or what they are called.

```
mysql> system <command>
```

The system command allows you to enter unix commands without leaving `mysql`. For example, `system clear` will clear the screen.

```
mysql> source <filename>
```

This will execute any SQL code you provide it in a text file. You might find this an easier way of working than entering line by line. You must execute mysql from the directory that the text file is stored in. The common extension for these files is .sql. If you use notepad++ (installed on the school servers) then it will do syntax highlighting.