MySQL

A practical introduction to database design

Dr. Chris Tomlinson
Bioinformatics Data Science Group,
Room 126, Sir Alexander Fleming Building
chris.tomlinson@imperial.ac.uk
Database Classes

• 24/09/18 : AM Software Carpentry Database Introduction
  • SQL-lite

• 24/09/18 : PM Database Design
  • MySQL
Database Server

• The data in a database is accessed externally through a piece of software called a database server. A database server simply waits for requests for data and returns them via the network to the requester.

• When using SQLite the server is located on the local machine

• Other database systems can use an external server

• This afternoon we will be using MySQL
MySQL : Database Server
Typical Database Architecture

- A client machine sends and receives messages from the database server
- Requests are in SQL
- Structured Query Language
Database Client

• A piece of software that arbitrates communication between database and remote user.
• We will use the MySQL client HeidiSQL
Connecting to a database server

- The server is a remote resource
- We send SQL commands to the server
- The server returns the results of SQL commands to the client
- The client formats the results
- We read and understand the results?
- Ensembl
A Relational Database

• A relational database organizes data in tables (or relations).

• A table is made up of rows and columns.

• A row is also called a record (or tuple).

• A column is also called a field (or attribute).

• A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store a huge amount of data, and effectively retrieve selected data.

• A language called SQL (Structured Query Language) was developed to work with relational databases.
A Database Table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>alpha_2</th>
<th>alpha_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Afghanistan</td>
<td>af</td>
<td>afg</td>
</tr>
<tr>
<td>2</td>
<td>Åland Islands</td>
<td>ax</td>
<td>ala</td>
</tr>
<tr>
<td>3</td>
<td>Albania</td>
<td>al</td>
<td>alb</td>
</tr>
<tr>
<td>4</td>
<td>Algeria</td>
<td>dz</td>
<td>dza</td>
</tr>
<tr>
<td>5</td>
<td>American Samoa</td>
<td>ac</td>
<td>asm</td>
</tr>
<tr>
<td>6</td>
<td>Andorra</td>
<td>ad</td>
<td>and</td>
</tr>
<tr>
<td>7</td>
<td>Angola</td>
<td>ao</td>
<td>ago</td>
</tr>
<tr>
<td>8</td>
<td>Anguilla</td>
<td>ai</td>
<td>ala</td>
</tr>
<tr>
<td>9</td>
<td>Antarctica</td>
<td>aiq</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Antigua and Barbuda</td>
<td>ag</td>
<td>ate</td>
</tr>
<tr>
<td>11</td>
<td>Argentina</td>
<td>ar</td>
<td>arg</td>
</tr>
<tr>
<td>12</td>
<td>Armenia</td>
<td>am</td>
<td>arm</td>
</tr>
<tr>
<td>13</td>
<td>Aruba</td>
<td>aw</td>
<td>abw</td>
</tr>
<tr>
<td>14</td>
<td>Australia</td>
<td>au</td>
<td>aus</td>
</tr>
<tr>
<td>15</td>
<td>Austria</td>
<td>at</td>
<td>aut</td>
</tr>
<tr>
<td>16</td>
<td>Azerbaijan</td>
<td>az</td>
<td>aze</td>
</tr>
<tr>
<td>17</td>
<td>Bahamas</td>
<td>bs</td>
<td>bhs</td>
</tr>
<tr>
<td>18</td>
<td>Bahrain</td>
<td>bh</td>
<td>bhr</td>
</tr>
<tr>
<td>19</td>
<td>Bangladesh</td>
<td>bd</td>
<td>bgd</td>
</tr>
<tr>
<td>20</td>
<td>Barbados</td>
<td>bb</td>
<td>bb</td>
</tr>
<tr>
<td>21</td>
<td>Belarus</td>
<td>by</td>
<td>blr</td>
</tr>
<tr>
<td>22</td>
<td>Belgium</td>
<td>be</td>
<td>bel</td>
</tr>
<tr>
<td>23</td>
<td>Belize</td>
<td>bz</td>
<td>blz</td>
</tr>
<tr>
<td>24</td>
<td>Benin</td>
<td>bj</td>
<td>ben</td>
</tr>
<tr>
<td>25</td>
<td>Bermuda</td>
<td>bm</td>
<td>bmu</td>
</tr>
<tr>
<td>26</td>
<td>Bhutan</td>
<td>bt</td>
<td>bth</td>
</tr>
<tr>
<td>27</td>
<td>Bolivia, Plurinational State of</td>
<td>bo</td>
<td>bol</td>
</tr>
<tr>
<td>28</td>
<td>Bonaire, Sint Eustatius and Saba</td>
<td>bq</td>
<td>bes</td>
</tr>
<tr>
<td>29</td>
<td>Bosnia and Herzegovina</td>
<td>bo</td>
<td>bih</td>
</tr>
<tr>
<td>30</td>
<td>Botswana</td>
<td>bv</td>
<td>bwa</td>
</tr>
</tbody>
</table>
Real Example

• Start the database client HeidiSQL. Connect using the following parameters;
  • Host: bss-db01.bioinformatics.ic.ac.uk
  • Username: msc_user
  • Password: 8grombas
  • Port: 3307

• Look at the msc_countries database by double clicking on it
• There is one table in the database called country – double click on it
• Click on the data tab to view the data
• Country has 3 attributes and 249 records
  • i.e. There are 249 countries in the table and we have three pieces of information about each one
• Use the Query tab to try out some SQL queries;
  • Select count(*) from country
  • Select * from country where name like ‘B%’
Real Example (2)

- Look at the CREATE CODE tab for the table country
- You will see;
  - `CREATE TABLE `country` ( `id` INT(11) NOT NULL AUTO_INCREMENT, `name` VARCHAR(50) NOT NULL DEFAULT '', `alpha_2` VARCHAR(2) NOT NULL DEFAULT '', `alpha_3` VARCHAR(3) NOT NULL DEFAULT '', PRIMARY KEY (`id`))`

- This is the SQL required to create the table
Recap: Typical Database Architecture

- A client machine sends and receives messages from the database server.
- Requests are in SQL.
CREATE TABLE `country`
(`id` INT(11) NOT NULL AUTO_INCREMENT,
`name` VARCHAR(100) NULL DEFAULT NULL,
`code` CHAR(2) NOT NULL,
PRIMARY KEY (`id`))

The primary key of a relational table uniquely identifies each record in the table.

It can either be a normal attribute that is guaranteed to be unique (such as Social Security Number in a table with no more than one record per person) or it can be generated by the DBMS.

Primary keys may consist of a single attribute or multiple attributes in combination.
SQL naming conventions

• Naming conventions speed up development time by keeping things **consistent and predictable**. Groups working together may use an agreed naming convention.

• Whenever I create a new database table I adhere to the following naming conventions;

  • The name of the table is a noun descriptive of the entity being stored and is in lower case singular

  • Attributes are descriptive nouns (when possible) and also in lower case

  • If two words are needed spaces are represented by the _ character

  • EVERY TABLE has an integer primary key as its first field called id

  • If the table needs a name attribute (i.e. the entity being modelled has a name) then this is the second field in the table and is called name
### SQL data types

All SQL table attributes must have a data type.

The data type you use for an attribute depends on the type of data that will be stored in that database column.

<table>
<thead>
<tr>
<th>DATA TYPES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>String, length 0 - 255</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String, length 0 - 255</td>
</tr>
<tr>
<td>TINYTEXT</td>
<td>String, length 0 - 255</td>
</tr>
<tr>
<td>TEXT</td>
<td>String, length 0 - 65535</td>
</tr>
<tr>
<td>BLOB</td>
<td>String, length 0 - 65535</td>
</tr>
<tr>
<td>MEDIUMTEXT</td>
<td>String, length 0 - 16777215</td>
</tr>
<tr>
<td>MEDIUMBLOB</td>
<td>String, length 0 - 16777215</td>
</tr>
<tr>
<td>LONGBLOB</td>
<td>String, length 0 - 4294967295</td>
</tr>
<tr>
<td>* TINYINT</td>
<td>Integer, -128 to 127</td>
</tr>
<tr>
<td>* SMALLINT</td>
<td>Integer, -32768 to 32767</td>
</tr>
<tr>
<td>* MEDIUMINT</td>
<td>Integer, -8388608 to 8388607</td>
</tr>
<tr>
<td>* INT</td>
<td>Integer, -2147483648 to 2147483647</td>
</tr>
<tr>
<td>* BIGINT</td>
<td>Int, -9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Decimal (precise to 23 digits)</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Decimal (24 to 53 digits)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>&quot;DOUBLE&quot; stored as string</td>
</tr>
<tr>
<td>DATE</td>
<td>YYYY-MM-DD</td>
</tr>
<tr>
<td>DATETIME</td>
<td>YYYY-MM-DD HH:MM:SS</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>YYYYMMDDHHMMSS</td>
</tr>
<tr>
<td>TIME</td>
<td>HH:MM:SS</td>
</tr>
<tr>
<td>ENUM</td>
<td>One of preset options</td>
</tr>
<tr>
<td>SET</td>
<td>Selection of preset options</td>
</tr>
</tbody>
</table>

* Note: "UNSIGNED" TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT have the same range of values but start at 0, e.g. TINYINT UNSIGNED is between 0 and 255.
Data types of Table attributes

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Andorra</td>
<td>AD</td>
</tr>
<tr>
<td>2</td>
<td>United Arab Emirates</td>
<td>AE</td>
</tr>
<tr>
<td>3</td>
<td>Afghanistan</td>
<td>AF</td>
</tr>
<tr>
<td>4</td>
<td>Antigua and Barbuda</td>
<td>AG</td>
</tr>
<tr>
<td>5</td>
<td>Anguilla</td>
<td>AI</td>
</tr>
<tr>
<td>6</td>
<td>Albania</td>
<td>AL</td>
</tr>
<tr>
<td>7</td>
<td>Armenia</td>
<td>AM</td>
</tr>
<tr>
<td>8</td>
<td>Angola</td>
<td>AO</td>
</tr>
<tr>
<td>9</td>
<td>Antarctica</td>
<td>AQ</td>
</tr>
<tr>
<td>10</td>
<td>Argentina</td>
<td>AR</td>
</tr>
<tr>
<td>11</td>
<td>American Samoa</td>
<td>AS</td>
</tr>
<tr>
<td>12</td>
<td>Austria</td>
<td>AT</td>
</tr>
<tr>
<td>13</td>
<td>Australia</td>
<td>AU</td>
</tr>
<tr>
<td>14</td>
<td>Aruba</td>
<td>AW</td>
</tr>
<tr>
<td>15</td>
<td>Åland Islands</td>
<td>AX</td>
</tr>
<tr>
<td>16</td>
<td>Azerbaijan</td>
<td>AZ</td>
</tr>
<tr>
<td>17</td>
<td>Bosnia and Herzegovina</td>
<td>BA</td>
</tr>
<tr>
<td>18</td>
<td>Barbados</td>
<td>BB</td>
</tr>
<tr>
<td>19</td>
<td>Bangladesh</td>
<td>BD</td>
</tr>
<tr>
<td>20</td>
<td>Belgium</td>
<td>BE</td>
</tr>
<tr>
<td>21</td>
<td>Burkina Faso</td>
<td>BF</td>
</tr>
<tr>
<td>22</td>
<td>Belize</td>
<td>BG</td>
</tr>
<tr>
<td>23</td>
<td>Bahrain</td>
<td>BH</td>
</tr>
<tr>
<td>24</td>
<td>Burundi</td>
<td>BI</td>
</tr>
<tr>
<td>25</td>
<td>Benin</td>
<td>BJ</td>
</tr>
<tr>
<td>26</td>
<td>Saint Barthélemy</td>
<td>BL</td>
</tr>
<tr>
<td>27</td>
<td>Bermuda</td>
<td>BM</td>
</tr>
</tbody>
</table>

- **char (2)**: Examples include `Andorra`, `United Arab Emirates`, `Afghanistan`, `Antigua and Barbuda`, `Anguilla`, `Albania`, `Armenia`, `Angola`, `Argentina`, `American Samoa`, `Austria`, `Australia`, `Aruba`, `Åland Islands`, `Azerbaijan`, `Bosnia and Herzegovina`, `Barbados`, `Bangladesh`, `Belgium`, `Burkina Faso`, `Belize`, `Bahrain`, `Burundi`, `Benin`, and `Saint Barthélemy`.

- **varchar (50)**: Examples include country names and codes.

- **integer**: Not explicitly shown in the image, but implied by the table structure.
Exercise 2 – Use HeidiSQL to create a database and table

- Create a database – name it after your last_name. Call it msc_<Last name>
  - Right click on the host icon -> create new -> database

- Select the database and create a new table (right click on database -> create new) – call it student
  - Create 3 fields in student;
    - id (integer)
    - first_name (varchar 50)
    - last_name (varchar 50)
  - Make id into the primary key
    - In the table builder interface;
      - Make it not null, auto increment.
      - Right click on the id attribute and set as Primary key
  - Save the table
  - Look at the table creation SQL in the interface – this is the command that you have just run to create your table

- Populate the table with the names of yourself and your fellow students.
  - Use the data tab and add records using the + button
  - If you have set up the primary key properly you need only to add the first_name and last_name. The id will be set automatically
  - Use the tick button to commit each row

- Use SQL queries to select subsets of the data;
  - SELECT * from student where first_name like ‘C%’
  - SELECT last_name from student
Data relationships

• So far we have looked at a database that contains single data table. This is very similar to a spread sheet.

• A useful database will contain many tables representing different entities

• With a Relational Database you are able to create relationships between the different tables in the database

• This feature gives RDMS their power
Types of Relationships Between Database Tables

• One to One

• One to Many

• Many to Many
Visualising the Relationships: Entity Relationship Diagrams

- **STUDENT**
  - student_id
  - student_name
  - student_address

- **COURSE**
  - course_name
  - course_number

- **INSTRUCTOR**
  - instructor_no
  - instructor_name
  - instructor_faculty

- **SEAT**
  - seat_no
  - seat_position

- **CLASS**
  - course_name
  - section_number
  - num_registered
  - class_date_time

- **SECTION**
  - section_number

- **PROFESSOR**
  - professor_id
  - professor_name
  - professor_faculty

- **RELATIONSHIPS**
  - STUDENT→SEAT (fills)
  - STUDENT→TAKES→COURSE (takes)
  - COURSE→HAS→CLASS (has)
  - INSTRUCTOR→TEACHES→CLASS (teaches)
  - SECTION→HAS→CLASS (has)
  - INSTRUCTOR→TEACHES→SECTION (teaches)
One to One Relationship

• May be used to describe a relationship between two different entities
• But each row in table 1 has a corresponding row in table 2
• If a one to one relationship exists it may be sometimes be better to combine the entities in a single table
Foreign Keys

• References from a record to a record in another tables need to be made to a unique feature of the record
  • Primary key

• A reference to another record’s primary key from a record is called a foreign key
SQL naming conventions(2)

• Naming conventions speed up development time by keeping things consistent and predictable. Groups working together may use an agreed naming convention.

• Whenever I create a foreign key I adhere to the following naming convention;
  • The name of the foreign key takes the form
    • <foreign table>_id
  • The foreign key references the id field of the foreign table
Implementing a One to One Relationship

• Go to the your student database
  • Select create new -> table – name the table identity_card
  • Create three attributes in the table
    • id (int) -> autoincrement, make into a primary key
    • card_number -> int
    • expiry_date -> date
  • Add your id card to the table as a record (and those of your neighbours if you can – add one identity card record for each of the students that you have input – make them up if you have to)
    • Dates are added in the SQL format YYYY-MM-DD
  • Go to the design panel of your student table and add a new attribute called identity_card_id, set the default value to 1
  • Go to the foreign key tab and make a new foreign key. Set the column as identity_card_id and the reference table is identity_card and the foreign column id . Leave the on Delete and on Update settings as they are (RESTRICT). Then save the table.
  • Change the foreign keys in your student table so that each references a unique record in identity_card
  • Try to delete a record that is referenced in identity_card. What happens?
Database Constraints

• We could not delete an identity_card that was referenced in another table because of the Foreign Key constraint that we set up
• The Foreign key ensures that the data in the database keeps its ‘referential integrity’
• You can set up many constraints on how data is added, updated and/or deleted so that the data your database keeps its integrity
One to Many Relationship

• This is a very common type of relationship in a database

• It is often used as a way of grouping data
One to Many : practical example

- In our example database each student has a nationality (a country of origin)
- But one country may have many students who come from it
- This is an example of a one to many relationship
ER Diagram – One to Many

MSc Student
Entity Relationship Diagram

- msc_student
  - id
  - first_name
  - last_name
  - identity_card_id
  - country_id

- identity_card
  - id
  - card_number
  - expiry_date

- country
  - id
  - name
  - alpha_2
  - alpha_3
One to many Relationship: Implementation

• Go to the countries database
  • Right click on the country table
  • Select create new -> table copy
  • Copy the table to your student database by selecting the database name from the list and call it country. Check that it has copied to your student database (The f5 key will refresh the interface)
  • Go to the student table design pane and add an extra int field called country_id. Set the default value of country_id to 235 (this is on the right) and save the modified table
  - Then go to the Foreign Keys tab above and add a new foreign key. Connect the column country_id to the reference table country, foreign column id. Leave the on Update and on Delete settings as ‘RESTRICT’
  - You have now created a relationship between every student record with one country record. Correct the countries of the student records in your table.
  - Add a new student – you should now see the countries as a pulldown menu along with their primary key. The HeidiSQL client interface has interpreted the relationship that you have made between the tables
  - Update your student records with their correct countries.
  - Now go to the country table. Try to delete a country that has a/some student(s) associated with it – What happens?
  - Write an SQL select query to return only the UK students based on the country_id (235)
  - Write a nested (two select statements in one using brackets) query to return UK students based on the country name ‘United Kingdom’
ER Diagram – So far

MSc Student
Entity Relationship Diagram

One to one

One to many
Many to many relationship

- Many to many relationships occur when one entity can have more than one link to another entity and vice versa.

- It is easier to think of this relationship in terms of examples.

- One example is authors and academic papers:
  - One author can have many papers
  - One paper can have many authors

- In this case we cannot simply make a link between the two tables – as there could be many links in either table so we potentially would need more than one foreign key in each table.
ER diagram: Many to many relationship

MSc Student
Entity Relationship Diagram

- msc_student
  - id
  - first_name
  - last_name
  - identity_card_id
  - country_id

- country
  - id
  - name
  - alpha_2
  - alpha_3

- identity_card
  - id
  - card_number
  - expiry_date

Many to many
Q: How can we make a many to many relationship in a database?

• Could be an interview question

• A: make a linking table that contains the foreign keys to both tables. The linking tables can contain multiple records for records of both tables.
SQL naming conventions (3)

- Naming conventions speed up development time by keeping things consistent and predictable. Groups working together may use an agreed naming convention.

- Whenever I create a linking table I adhere to the following naming convention:
  - The name of the linking table is a concatenation of the names of the two tables it links together
  - The linking table contains three fields
    - Id (primary key), foreign key 1 and foreign key 2
  - The naming convention previously outlined is used for foreign keys
Many to Many Relationship: practical implementation

• Create a table called ‘course’ in your student database. Give it two attributes id and name. Make id into the primary key and set it to auto increment. Save the new table.

• Add some records to course table making up the information.

• Make a linking table called student_course.
  • Give it three fields id, student_id and course_id (all ints). Make id into the primary key.
  • Set up the foreign key relationships between the linking table and the other two tables and save the table.
  • Add some records to the linking table.

• Write the SQL Select statement to select all the students who are on a particular course first by id;
  • Select first_name, last_name from student, student_course where course_id = 1 and student_id = student.id
  • Then by name
  • Select first_name, last_name from student, student_course, course where course.name = ‘<name>’ and course.id= course_id and student_id = student.id
Our student database

MSc Student
Entity Relationship Diagram

msc_student
- id
- first_name
- last_name
- identity_card_id
- country_id

country
- id
- name
- alpha_2
- alpha_3

course
- id
- name

identity_card
- id
- card_number
- expiry_date
Adding Indexes to Attributes

- When tables get larger retrieving information from them gets slower and slower.

- In order to improve performance you can pre-sort table information in the database by adding an index to it.

- This speeds up data retrieval time but does increase data insert time.

- You would do this to attributes that you know you are likely to use in select, update and delete statements.

- We have already added a primary key (index) to our table.
Adding an Index: FULLTEXT

- One of the types of index that you can add in MySQL is called FULLTEXT.

- A FULLTEXT index allows certain types of text search operations to be carried out efficiently.

- Add a FULLTEXT index to the surname of your msc_student table by right clicking on it in the table description. Select ‘Create New Index’ and select FULLTEXT.
Database Design

• Determine what the data you are going to store is about. => think about why you want to store the data.

• List the items you should be tracking.

• One of the main objectives is to avoid repetitions of data in tables. If you have to type the same text in different tables then your design is incorrect

• If you have repeated information in your database:
  • Need to type over & over again the same thing (tedious and prone to errors)
  • Database is larger than it needs to be (therefore slower and more expensive to store)
  • If modification needed, every copy needs to be modified
  • Makes data search slower and storage not optimized
Database design quiz
Database Design : Practical