CSCU9B3
Introduction to MySQL

Data Definition

Contents

- Introduction to MySQL
- Create a table (relation)
- Specify keys and relations
- Empty and drop tables

Introduction

- SQL is a declarative language for manipulating a relational database
- Use it to issue commands to the database for tasks such as:
  - Creating and managing tables
  - Inserting data into tables
  - Searching for and retrieving data from tables
  - Deleting data and tables
- MySQL is a particular version of SQL

Online Resources

http://dev.mysql.com/doc/refman/5.7/en/index.html

- Is Oracle's MySQL documentation site –
  - note that 5.7 is the version we will use in practicals here.
- You will mostly need:
  - Chapter 13: SQL Statement Syntax
  - Chapter 11: Data Types

- A better place to learn SQL is:

  http://www.w3schools.com/sql/default.asp

Database Engines

- MySQL supports a number of database engines, each designed for databases with different needs
- We will use the InnoDB engine as it supports inter-table constraints (Foreign keys)
- For more on Storage Engines, see:

  http://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

Making MySQL Statements

- We tend to use UPPER CASE for reserved words
- Strings are enclosed in forward single ‘quotes’ or double “quotes”
- Names of database elements such as tables and fields are enclosed in backwards `quotes`
- Statements are separated by semi-colons ;

SELECT `name` from `mytable` WHERE `name`=‘John’

- You can drop the use of quotes if it is safe to do so, for example for names with no spaces or special characters.

SELECT name from mytable WHERE name=‘John’

Creating a Table

- The simplest form of SQL **CREATE TABLE** looks like:

```
CREATE TABLE IF NOT EXISTS tablename
      (colname datatype,
       ... )

CREATE TABLE Staff
      (Sno    INT,
      Sname VARCHAR(20),
      Dept  VARCHAR(20),
      Grade VARCHAR(7))
```

- See http://dev.mysql.com/doc/refman/5.7/en/create-table.html for full details of CREATE TABLE

Data Types

- MySQL is strict about the use of data types.
- Data types include:
  - VARCHAR variable length text strings
  - INT integers
  - DECIMAL numbers with decimal places
  - DATE full date (yyyy-mm-dd)

  - And others that you can read about here

    http://dev.mysql.com/doc/refman/5.7/en/data-types.html

Table Constraints

- The specification of a column can include some extras:
  - ***default value*** (used if an insertion doesn't supply a value)
  - and a ***column constraint*** (next slide)

- We can add ***table constraint(s)*** before the closing bracket

- Both types of constraint are used to protect integrity (next slide)

Data Integrity

- Column constraints
  - enforcing entity and referential integrity

```
[NOT NULL | NULL]
[DEFAULT default_value]
[AUTO_INCREMENT]
[UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
PRIMARY KEY                (only one per table)
FOREIGN KEY REFERENCES  table (column)
```

e.g.

```
CREATE TABLE Staff
      (Sno INT PRIMARY KEY,
      Sname VARCHAR(20) NOT NULL,
```

Data Integrity: Foreign Keys

- The last of these declares a foreign key:

```
CREATE TABLE Staff
   ( ...
  Dept VARCHAR(20) FOREIGN KEY REFERENCES Depts(Dname),
  Grade VARCHAR(7) FOREIGN KEY REFERENCES Paytable
   )
```

  - The referenced column must be the key of its table
  - We shall not be allowed to insert a row into **`Staff`** unless **`Dept`** and **`Grade`** contain valid values (one found in the other table) or null
    - We can miss out the bracketed column-name if it is the same in both tables

Data Integrity: Cascaded Deletion

- We can add **`ON DELETE CASCADE`** to **`REFERENCES`**
- This means that if a row in the other table is deleted, all matching rows in this table should be deleted too

- For example, a **`Dependant`** table (for the dependants of employees) might declare the column:

```
Enum REFERENCES Employee ON DELETE CASCADE
```

- So if we delete employee 123 from the **`Employee`** table, then all his/her dependants are deleted from the **`Dependant`** table, thus protecting referential integrity
- We should only do this for weak entities!

Data Integrity: Deadly Embrace

- Notice that we cannot
  - declare the `Staff` table until we have declared `Depts` and `Paytable`
  - insert any data into `Staff` until we have matching data in `Depts` and `Paytable`
- This can form a deadly embrace
  - suppose the `Depts` table references the `Staff` table (for Head of Dept, for example)
  - then we cannot declare the `Depts` table until we have declared `Staff ;` but we cannot declare the `Staff` table until we have declared `Depts`
  - the solution to this problem comes later

Data Integrity: Table Constraints

- After the last field, we can add *table-constraints*
  - these look like column-constraints, but they can reference more than one column

```
CREATE TABLE HTR
    (Hour    char(6),
    Teacher char(3),
    Room     char(4),
    PRIMARY KEY (Hour, Teacher)) ;
```

  - this is how to declare composite primary keys
  - we can declare (possibly composite) foreign keys in the same sort of way, e.g. the `Staff` table could be rewritten to put the constraints at the end (next slide):

Table Constraints (continued)

```
CREATE TABLE Staff
     (Sno    INT PRIMARY KEY,
     Sname VARCHAR(20) NOT NULL,
     Dept  VARCHAR(20) FOREIGN KEY REFERENCES Depts (Dname),
     Grade VARCHAR(7) FOREIGN KEY REFERENCES Paytable) ;
```

– This could be written as:

```
CREATE TABLE Staff
     (Sno    INT,
     Sname VARCHAR(20) NOT NULL,
     Dept  VARCHAR(20),
     Grade VARCHAR(7),
     PRIMARY KEY(Sno),
     CONSTRAINT FOREIGN KEY (Dept) REFERENCES Depts (Dname),
     CONSTRAINT FOREIGN KEY (Grade) REFERENCES Paytable) ;
```

Altering an Existing Table

- We can change tables, using **`ALTER TABLE,`** even after they contain data
- Amongst other possibilities, we can add or modify columns

```
ALTER TABLE Staff ADD
    (StreetAddress VARCHAR(20),
    TownAddress    VARCHAR(20)) ;

ALTER TABLE Staff MODIFY
    (TownAddress   DEFAULT 'Stirling') ;
```

Altering Tables: Avoiding Deadly Embrace

- We can use **`ALTER TABLE`** to add a constraint
  - This gets us out of the "deadly embrace" mentioned earlier

```
CREATE TABLE Dept
       (Dname  VARCHAR(20) PRIMARY KEY,
       Head   INT) ;

CREATE TABLE Staff
       (Sno   INT PRIMARY KEY,
       Sname VARCHAR(20) NOT NULL
       Dept  VARCHAR(20)
       CONSTRAINT FOREIGN KEY (Dept)
       REFERENCES Dept (Dname) ;

ALTER TABLE Dept ADD
    FOREIGN KEY (Head) REFERENCES Staff(Sno) ;
```

Dropping and Deleting

- We can completely remove a table: both its data (if any) and its definition

```
DROP TABLE tablename ;
DROP TABLE tablename CASCADE CONSTRAINTS ;
```

  - the second form removes Foreign Key constraints in associated tables (which otherwise could not be updated)

- Removing the data alone (not the definition):

```
DELETE FROM tablename ;
DELETE FROM tablename WHERE condition ;
```

  - we shall deal with conditions later

Getting Data into Tables

- There are two ways of using SQL to get data into tables
- Firstly, with the values in the SQL statement

```
INSERT INTO Staff VALUES
    (123,'Lee','CompSci','II.7') ;
```

  - if we are not loading all the columns, use this form:

```
INSERT INTO Staff (Sno, Sname) VALUES
    (456, 'Waldenstein') ;
```

- Secondly, by extracting the data from existing tables

```
INSERT INTO Loan
    SELECT DISTINCT Sno, Bno, Date_out
    FROM Staff_Borrower ;
```

- We can also use a bulk-loader utility (phpMyAdmin has one)

Getting Even More Data In

```
INSERT INTO `books` (`Name` ,`Number`)
VALUES ('book1', '1'), ('book2', '2'), ('book3', `3');
```

- See http://dev.mysql.com/doc/refman/5.7/en/insert.html for more details of the INSERT syntax

Changing Data in a Table

- Use:

```
UPDATE table SET field=value, field=value WHERE condition
```

- See http://dev.mysql.com/doc/refman/5.7/en/update.html for full syntax

End of Lecture

- Next SQL lecture will look at manipulating table data
  - Querying a database