Persistent Storage using Databases

Devdatta kulkarni

What is Persistent Storage?

- Storage medium that supports storage of data independent of whether an application program is running or not
  - Examples of persistent storage
    - Files written to disk
    - Records written to a database (topic of today's lecture)
- Opposite of persistent storage
  - In-memory storage (Main memory, random access memory, memory)
  - Examples of Non-persistent storage
    - Objects in our Java applications
      - The object "exists" only while the program is running

What is a database?

- Persistent storage in which data is in one or more *tables*
- Data is managed by database server
- A table has one or more *columns*

Retrieving data from a database

- How to retrieve data from a database table?
  - Using Structured Query Language (SQL)

- What is SQL?
  - SQL is a *declarative language* that supports writing queries over a set of tables to retrieve records that match the specified query criteria

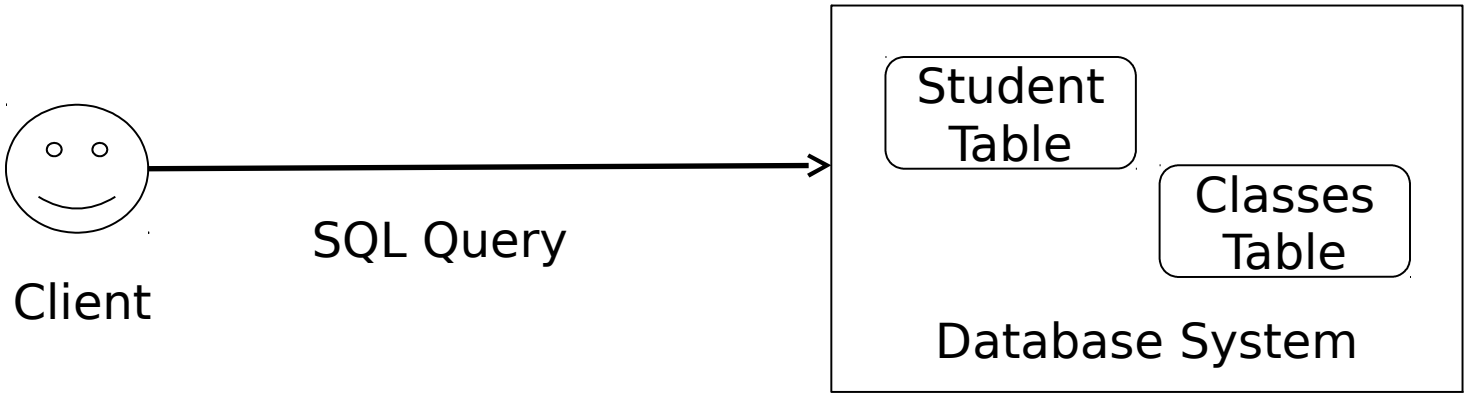
SQL

- What is a declarative language?
  - It is a language in which you write code identifying “what” the answer should look like instead of writing code identifying “how” to produce the answer

- Suppose we wanted to retrieve records of all the students living in Austin from the Student table:
  - In the declarative model, we will write a query of the following form and send it to the database system which manages all the database tables

  “Retrieve all records from the Student table for which City column is equal to Austin”

Querying a Database system



SQL Statements and CRUD Operations

- SQL Insert == Create (REST POST)
- SQL Update == Update (REST PUT)
- SQL Delete == Delete (REST DELETE)
- SQL Select == Read (REST GET)

SQL Examples

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

- Create
  - http://www.w3schools.com/sql/sql_insert.asp
- Read
  - http://www.w3schools.com/sql/sql_select.asp
- Update
  - http://www.w3schools.com/sql/sql_update.asp
- Delete
  - http://www.w3schools.com/sql/sql_delete.asp

SQL

- Some of the most important SQL commands
  - http://www.w3schools.com/sql/sql_syntax.asp
    - Create database
    - Create table
    - Select
    - Update
    - Delete
    - Insert Into
    - Alter database
    - Alter table
    - Drop table

Installing MySQL

- MySQL Community Server
  - http://dev.mysql.com/downloads/
    - On Mac, download the .dmg file
- Starting MySQL Server (on Mac)
  - In Spotlight search for MySQL
  - MySQL window should open up with “Start MySQL Server” button on it
  - MySQL Server Instance should be in “stopped” status
  - Hit the “Start MySQL Server” button; you may be prompted for a password
  - Enter the password; MySQL server should be up
- MySQL Clients
  - Command line
    - /usr/local/mysql/bin/mysql – if you install the .dmg from above link
  - MySQL Workbench
  - SQuirrel SQL
    - http://squirrel-sql.sourceforge.net/
- Local MySQL Server
  - 127.0.0.1
  - Port number 3306
  - Usename: root

Setting up the Database

- Run mysql client
  - /usr/local/mysql/bin/mysql -u root
- Create Database
  - create database student_courses;
- Create User
  - Example: create user 'devdatta'@'localhost’;
- Grant Privileges
  - GRANT ALL ON student_courses.* TO ‘devdatta’@’localhost’;
- Logout
  - On mysql prompt: “quit”
- Login as user;
  - /usr/local/mysql/bin/mysql –u devdatta –h localhost
- Create Table
  - use student_courses;

Problem

- Design a database schema to represent the following:
  - Courses in the CS department
  - Students in the CS department
  - A student can take at most one course
  - A course can be taken by zero or more students
- We want to support following queries:
  - Find all the students who are taking a particular course
  - Find all students and the courses that they are taking

Designing DB

- A *course* table to represent courses
- A *students* table to represent students
- A *relationship* from the *students* table to *courses* table to indicate that a student is taking a particular course

Courses table

A row in the table represents one course

Table columns:

- Name of the course
- Course number
- A unique id

Constraints:

- Name: Should not be NULL
- Course number: Should not be NULL
- Unique id: This will be the PRIMARY KEY of the table

Primary key

- What is a primary key?
  - Any attribute that can uniquely identify a row in the table
- If the resource has a unique attribute, can we use that as the primary key (natural key)?
  - Examples:
    - SSN number to represent a person
    - UTEID to represent a student
- Or, should we generate unique ids and use those (surrogate key)?

Pros/Cons Analysis

- Using Natural keys (resource attributes)
  - Pro
    - Easier to write queries
      - http://stackoverflow.com/questions/63090/surrogate-vs-natural-business-keys
  - Con
    - Problematic if the attribute name is changed in the future
- Using surrogate keys
  - Pro
    - Not affected by changes to a resource’s attributes
  - Con
    - Does not have meaning outside of the service

Courses table

create table courses(name varchar(255) NOT NULL, course_num varchar(20) NOT NULL, course_id int NOT NULL AUTO_INCREMENT, PRIMARY KEY(course_id));

course_id is the *surrogate key*

AUTO_INCREMENT:

- Let the database generate the values for course_id

Student table

A row in the table represents one student

A student can take at most one course

Table columns:

- Name of the student
- A unique id
- A column to capture the relationship between a student and a course

Constraints:

- name: Should not be NULL
- Unique id: This will be the PRIMARY KEY of the table
- course_id: A FOREIGN KEY to courses table

Foreign Key

- What is a Foreign key?
  - A column in a table that refers to primary_key column in another table

Student table

create table students(name varchar(255) NOT NULL, student_id int AUTO_INCREMENT, course_id int, PRIMARY KEY(student_id), FOREIGN KEY(course_id) REFERENCES courses(course_id));

student_id is the *surrogate key*

AUTO_INCREMENT:

- Let the database generate the values for student_id

Populate the tables

- insert into courses(name, course_num) values("Data Management", "CS347");
- insert into students(name) values("Student 2");
- insert into students(name, course_id) values("Student 4", (select course_id from courses where course_num="CS378"));

Queries

- Queries:
  - Find all the students who are taking a particular course
  - Find all students and the courses that they are taking

Join

- A *join* is a mechanism that allows writing of queries over several tables

- A join of two tables selects all the rows from both the tables as long as there is a match between the specified columns of the two tables

Types of Joins

- Inner Join
  - If we imagine tables being sets, then inner join can be thought of as *intersection* of the two sets with the intersection criteria being the matching column values on the join column
- Left outer join
  - Inner join + rows from the “left” table which may not have matched on the join column
- Right outer join
  - Inner join + rows from the “right” table which may not have matched on the join column
- Full outer Join
  - Left outer join + Right outer join – duplicates
    - MySQL does not support it

Join

- Find all students who are taking course CS378

```
select * from students join courses on students.course_id = courses.course_id where courses.course_num="CS378";
```

Join

- Find all students and the courses that they are taking
- Find the students who are taking at least one course and find the information of the course that they are taking

select * from students join courses on students.course_id = courses.course_id;

Left outer join

- Find all the students irrespective of whether they are taking any course; for those who are taking at least one course, find the information about the course

```
select * from students left outer join courses on students.course_id = courses.course_id;
```

Right outer join

- Find all the courses irrespective of whether a course is being taken by any student. For the course that is being taken by at least one student, find the information about all those students

```
select * from (students right outer join courses on students.course_id = courses.course_id);
```

Revisiting Problem

- We want to add support for the following:
  - A student can take more than one course
- And we want to support following query:
  - Find all courses that a student is taking

How to store data in tables?

- How would we represent a Student who is taking several classes?
  - Issues
    - We would need to repeat student information in each such record
    - If we have to change any attribute of a student then we will have to update all such records
- Ideally we would like to:
  - Avoid repetition
  - Store data in one place
- The concept of *Normalization* is defined for this purpose

Normalization

- Store data in one place
  - Single row in a single table
- Student – course example
  - 3 tables
    - student
    - courses
    - student_courses_xref
- Flip-side to normalization
  - In order to retrieve data, we need to query it using ``joins''

Schema change

- Our current schema cannot support the new requirement
  - Why?
    - Because it is not possible to represent a student with multiple courses
      - Adding multiple rows per student in the *students* table is not an option because each row in that table represents one and only one student
- Solution:
  - 3 tables
    - students
    - courses
    - student_courses_xref

The cross reference table

- Each row represents <student_id, course_id> pairs
  - E.g.:

<student_1, course_1>

<student_1, course_2>

<student_2, course_1>

Join

- Find all students and the courses that they are taking

```
select s.name, c.name from student s join student_courses_xref scx on s.uteid=scx.uteid join courses c on c.course_id=scx.course_id;
```

```
select s.name "Student", c.name "Course" from student s, course c, student_courses_xref scx where s.student_id=scx.student_id and c.course_id=scx.course_id;
```

Left outer join

- select s.name as 'student_name', c.name as 'course_name' from (students s left join student_courses_xref scx on s.student_id = scx.student_id left join courses c on c.course_id = scx.course_id) order by course_name;

Right outer join

- select s.name as 'student_name', c.name as 'course_name' from (students s right join student_courses_xref scx on s.student_id = scx.student_id right join courses c on c.course_id = scx.course_id) order by course_name;

References

- SQL Tutorial
  - http://www.w3schools.com/sql/
- Primary key vs. surrogate key
  - http://stackoverflow.com/questions/2186260/when-to-use-an-auto-incremented-primary-key-and-when-not-to
  - http://stackoverflow.com/questions/63090/surrogate-vs-natural-business-keys