Databases

Topics

- History
  - RDBMS
  - SQL
- Architecture
  - SQL
  - NoSQL
- MongoDB, Mongoose

Persistent Data Storage

- What features do we want in a persistent data storage system?

- We have been using text files to store our data for example in class.   What is wrong with them?

- Access methods?
  - sequential, indexed, random
- Sharing granularity?
- Data consistency?
- Scalability
- Security?

Hierarchical and Network Data Models (1960s)

- Hierarchical:
  - Upside down tree with one-to-many relationship between parent and child records
  - Efficient searching, data independence, security, integrity
  - Complex, lack of standards
- Network
  - Directed Acyclic graph
  - Each record can have multiple parents (one-to-many)
  - Main problem: complexity

Relational Databases

Relational Database Management System (RDBMS)

- First published by Edgar F. Codd in 1970 (Turing Award in 1981)
- Jim Gray known for his work on IBM's **System R**, a pre-cursor to SQL (Turing Award 1998)
- Donald D. Chamberlin and Raymond F. Boyce created **SEQUEL** (Structures English Query Language). Later changed to **SQL**.

Relational Database

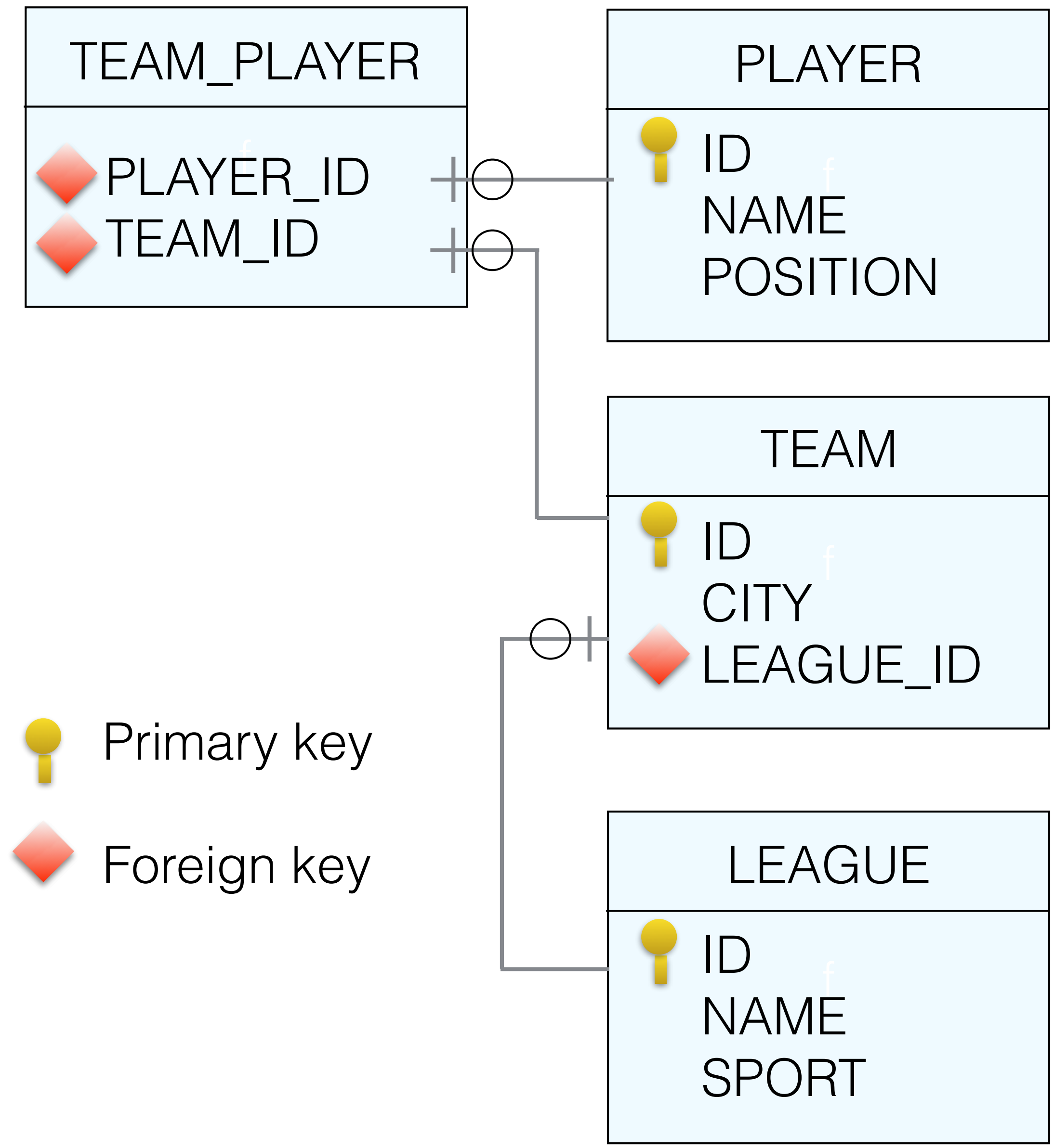
- A collection of tables with rows and columns
- Each row represents a record
- Each column represents an attribute of the records contained in the table

A recreational sports system has the following relationships:

- a player can be on many teams
- a team can have many players
- a team is in exactly one league
- a league has many leagues

The relationships between the Player, Team, and League entities:

- Many-to-many relationship between Player and Team
- Many-to-one relationship between Team and League



Why use an RDBMS?

- **Data safety:** Data is immune to program crashes.
- **Concurrent access:** Atomic access/updates.
- **Fault tolerance:** Replicated databases for instant failover on machine/disk.
- **Data integrity:** Aids to keep data meaningful.
- **Scalability:** Can handle small/large quantities of data in a uniform manner.
- **Reporting:** Easy to write SQL programs to generate arbitrary reports.

ACID

- Atomic
  - All operations in a transaction complete, or none do.
- Consistent
  - On completion of transaction, database is sound
  - Constraints of the database are applied and database is in a valid state
- Isolated
  - transactions don’t interfere with each other
- Durability
  - results of a transaction are permanent even in the case of failures

Most RDBMs are accessed/managed using SQL, a special purpose programming language.

- Data definition, manipulation, and controls.
    - Insert, query, update, delete.
- Schema creation and modification.

Examples

- SQLite
- MySQL
- PostgreSQL
- Access
- PointBase

Structured Query Language

- Standard for accessing relational database
    - Very cool theory behind it: relational algebra
- Queries
    - Lots of ways to extract information
    - You specify what you want
    - The database system figures out how to get it efficiently
    - Refer to data by contents, not just name

SQL Example Commands

```
CREATE TABLE Users (
  id INT AUTO_INCREMENT,
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  location VARCHAR(30));

INSERT INTO Users (
  first_name,
  last_name,
  location)
  VALUES
  ( ‘Emma’,
    ‘Swan’,
    ‘Storybrook’);
```

```
DELETE FROM Users WHERE
    last_name=‘Swan’;

UPDATE Users
    SET location = ‘New York’
    WHERE id = 2;

SELECT * FROM Users;

SELECT * FROM Users
    WHERE id = 2;
```

Keys and Indexes

- Consider a model fetch:

  SELECT * FROM Users WHERE id = 2;

- Database could implement this by:
  - Scan the Users table and return all rows with id = 2
  - Have built an index that maps id numbers to table rows.

- Index: Selected columns of a database optimized for searching

- Uses keys to tell database that building an index would be a good idea

Keys

- Primary key: Organize data around accesses.
  - Uniquely identifies a row in the table.

- Secondary key: Other columns that are have unique values

- Foreign key: The primary key defined in another table.

Schemas

- Schemas define the structure of the database
  - tables, columns indexes
- Schema needs to be defined before you can add data
- Not a great match with agile development approaches, because each new feature may require a schema change.
  - Database migrations allow incremental changes
- NoSQL databases are built to allow the insertion of data without a predefined schema.

Object Relational Mapping

- Writing lots of raw SQL queries isn't much fun
- Still need to map data from the database to objects in the application
- 2nd generation frameworks (Rails, Django) handled mapping of objects to SQL db
- Rails' Active Record
  - Objects map to database records
  - One class for each table in the database (Models)
  - Objects of the class correspond to rows in the table
  - Attributes of an object correspond to columns from the row
- Handle all the schema creation and SQL commands behind the object interface

NoSQL

- NoSQL databases do not use the relational model or SQL
  - Flexible schema
  - Quicker/cheaper to set up
  - Scalability
  - No declarative query language -> more programming
  - Relaxed consistency -> higher performance and availability
  - Relaxed consistency -> fewer guarantees

- Document types apply a key to a “document” with a complex data structure
  - Documents can contain many different key-value pairs, or key-array pairs, or even nested documents
  - Use standard data format, e.g. JSON, XML
  - Can be organized by collection, tag, etc.

Graph Databases

- Graph types store information about networks (nodes with connections), such as social connection and road maps
  - Connections are NOT ad hoc; this means performance improvements
  - Semantic queries
- Examples: neo4j, MarkLogic, OpenCog

Key-value Databases

- Key-value types are simplest NoSQL databases.
- Every item in the database is stored as an attribute name (“key”) together with its value
- Examples: cassandra, Couchbase

NoSQL

- Using SQL databases provided reliable storage for early web applications

- Led to NoSQL databases that matched web application object model

- MongoDB - Most prominent
  - Data model: Stores **collections** containing **documents**
  - **Dynamic schemas -**rather than predefined structure
  - Has expressive query language
  - Can use **indexes** for fast lookups
  - Tries to handle scalability, reliability, etc.
  - Caching: mostly in RAM

NoSQL vs Relational

- Relational databases usually **scale vertically**: a single system server hosts the entire database.
  - If eventual consistency is okay then more aggressive replication (and scaling) is possible
- NoSQL databases usually support **auto-sharding**: natively and automatically storing data across an arbitrary number of servers.

NoSQL vs Relational

- NoSQL databases also support **automatic replication**: high availability and disaster recovery without involving separate applications to manage these tasks.

- Many NoSQL database technologies have excellent integrated **caching** capabilities, keeping frequently-used data in system memory and removing the need for a separate caching layer.

BASE

- Basic Availability
  - Most of the database is available most of the time
- Soft-state
  - Stores don't have to be *write-consistent**, nor do different replicas have to be mutually consistent all the time
- Eventual consistency
  - At some later point, data stores will be consistent

* A write to data X is completed before any successive writes to X by the same process.

MongoDB

- MongoDB is an open-source, NoSQL database that uses a JSON-like (BSON) document-oriented model.

- Data is stored in collections (rather than tables).
  - Uses dynamic schemas rather than a predefined structure. Fields can be added/removed at will.
  - Works with many programming languages.
  - Caching: Most recent kept in RAM.
  - No transactions, but allows atomic operations.

MongoDB Document

```
{ name: 'Kate Monster’,
    ssn: '123-456-7890',
    addresses : [
      { street: '123 Sesame St',
        city: 'Anytown', cc: 'USA' },
      { street: '123 Avenue Q',
        city: 'New York', cc: 'USA' }
     ]
}
```

CRUD

- Create, Read, Update, Delete
- Simpler access model than SQL

Schema Enforcement?

- JSON blobs provide flexibility but not what is always wanted
- Consider: `<h1>Hello {{person.informalName}}</h1>`
  - Good: typeof person.informalName == 'string' and length < something
  - Bad: Type is 1GB object, or undefined, or null, or ...
- Would like to enforce a schema on the data
  - Can be implemented as validators on mutating operations
- Mongoose - Object Definition Language (ODL)
  - Take familiar usage from ORMs and map it onto Mongoose
  - Effectively masks the lower level interface to MongoDB with something that is friendlier

Next…

- MongoDB and Mongoose examples
- Web application architectures.