Unit 10 – Introduction to Databases

Roadmap – intro to databases, SQL, and NoSQL.

1. Slide 1 – Objectives
   1. ***“Schema” – Blueprint for the data***
   2. How to set up relationships between the data in our database.
2. Slide 2 – OG Database
   1. Disk Pack is what is represented by the little generic database icon.
   2. It used to be that the disk pack would be carried and then installed into a separate computer for reading and writing data.
3. Slide 3 – What is a database?
   1. RAM – a short-term, volatile memory
   2. File system – when we are pulling down repos, we deal with this each day; these are the main db for our computer.
   3. localStorage – stores data for longer than a session.
   4. Usually has some software – a DBMS (mySQL, PostgreSQL, oracle, mongoDB, etc).
4. Types of Databases
   1. Tables – SQL
      1. Kind of like excel or google sheets, with rows and columns.
      2. Relational data.
   2. Documents – NoSQL
      1. These are more like a JS object.
      2. It looks and feels like dealing with a JS object.
   3. Key-Value – NoSQL
5. We’ll be working with PostgreSQL - Table - and MongoDB – Documents.
6. Table Stores
   1. A relational database (SQL) is composed of tables.
   2. Columns are fixed, rows are added.
   3. Can’t have a table inside a table, but we can reference other tables through ids.
   4. “Data point” – where a column and row meet.
   5. A “record” is a full row containing each piece of data.
   6. In SQL databases, we can set up rules.
      1. Like saying “I want every data point to have data, no null values”
      2. Or “I want only numbers/strings/etc”
   7. We want an “id” for each entry, so that each record can be identified.
      1. Called a **primary key** **– uniquely identifies records.**
7. Relational DB
   1. Organization based on a **relational model**
   2. **Model** – one or more tables consisting of col/rows.
   3. **Column** referred to as “attributes” or “fields”
   4. **Row** referred to as either “records” or “tuples”
   5. **Model** describes the entire database, a “schema” is a blueprint.
   6. **PostgreSQL** – emphasizes extensibility
      1. One of the “cons” of SQL – rigid and difficult to modify
      2. PostgreSQL – makes it easier to extend our functionality.
      3. ACID Compliant
         1. ACID compliance is the highest standard of reliability for databases.
         2. **Atomicity** – transactions are “all or nothing.”
            1. If I work at a bank and a client wants to withdraw money, it should go from point A to B to C. If any part fails, it should ensure that the entire transaction fails.
         3. **Consistency** –
            1. Data written into a DB must be valid via implemented rules/constraints.
         4. **Isolation**
            1. Concurrency control – making sure that concurrent execution results in same result as running one after another.
         5. **Durability**
            1. Transactions remain committed even through loss of power.
8. Document Stores (NoSQL)
   1. Very close to JSON.
   2. Collection of objects (JS objects actually)
   3. Arbitrary nesting
   4. Built with emphasis on Speed.
   5. **NOT** ACID Compliant (by default).
      1. Some can be set up to be ACID compliant.
      2. And this is why document stores can function so quickly;
      3. There are layers of validation present in ACID compliant architecture that is not present here.
      4. Unlike databases, we don’t have to specify the kind of data we want to store.
   6. Document given to us looks exactly like a JSON object (strings in double quotes, nested in curly brackets).
   7. NoSQL used in big-data and real-time apps.
      1. Design simplicity;
      2. “horizontal scaling” of machine clusters;
      3. Fine control over availability
      4. Basically, fast and flexible as compared to SQL databases.
9. MongoDB
   1. Ad hoc queries – “on the fly” queries… These are “one-time use” queries, versus the usual stored, frequently used procedures.
   2. Indexing
      1. MongoDB will automatically assign an id field (automatic indexing)
   3. Replication
      1. Automatic replication of data of two or more copies.
   4. Load-balancing
      1. “sharding”
   5. Aggregation
      1. mapReduce -> used for batch processing of data and aggregation.
         1. “map” – batch processing
         2. “reduce” – aggregation
10. Key-Value Store
    1. Keys and values without nesting
    2. Often implemented for caching.
    3. Can hold a lot of data, but is not great for a primary database
    4. Better to go with a document store like MongoDB
    5. Or a relational DB.
    6. Redis, Riak, LocalStorage
       1. Redis, for example, holds all data in RAM and syncs back to disc every 1 second.
       2. This can result in data loss in the case that RAM happens to clear data in the 1 second that it has not synced.
11. Popularity:
    1. SQL is still ranked in the first 4 options
       1. Oracle/MySQL/PostgreSQL/etc
    2. MongoDB is in 5th
12. Schemas
    1. A map of our data.
    2. Here, we define our data’s attributes
    3. Comparison
       1. Key-value does not have a schema
       2. SQL – schema is strictly enforced.
          1. Makes sense, with ACID compliance.
       3. Document – doesn’t normally enforce a schema
          1. The object can have any keys and values, but you can decide on a schema.
13. Schema example (MongoDB)
    1. Uses JS
    2. Provides a blueprint for the data.
    3. MongoDB –
       1. When a multiple constraints need to be defined,
       2. We wrap those constraints in an object
       3. However, what is stored will still be defined by the “type” parameter.
14. Schema example (PostgreSQL)
    1. Absolutely required in a SQL database
    2. Here, we have a CREATE TABLE command “app”
    3. Columns defined with strings
       1. Constraints set after without any other operators.
       2. If we want to use an ID field, we **have to set it up.**
       3. When, in PostgreSQL had a record of “status” as:
          1. “status” varchar NOT NULL DEFAULT “applied”
       4. We have “NOT NULL” along with “DEFAULT” as a failsafe.
    4. At the bottom, we see our constraints and “foreign keys”
15. Foreign Keys
    1. First, Data Models..
16. Data Models
    1. Modeling data in a relational database
    2. You have a few different kind of relationships
       1. One to one/one to many/many to many
    3. Linking a model to a different model happens by **establishing relationships through the creation of foreign keys**.
    4. A foreign key links one model to another.
       1. In our “cats” example,
       2. Cats id and food id are the primary keys
       3. And the fav\_food id relates the cats to their foods. This is the foreign key.
    5. Many to Many relationships
       1. You cannot have many to many without something called a “join table.”
       2. One cat may have multiple favorite foods
       3. Join table allows multiple instances of the same cat to be linked to various records in another model
       4. And vice versa.
       5. So a join table might be linked to multiple.
       6. We need this **because we can’t use arrays**.
          1. (Recall: there is no nesting allowed in SQL databases).
17. Non-relational databases
    1. Document stores
    2. Nesting – fastest approach, but throws away referential integrity.
    3. Referencing – more robust but slower.
18. Security Vulnerabilities:
    1. **Databases are the most at risk.**
    2. Mass assignment
       1. An active record pattern is abused to modify data that should be private.
    3. Imagine a login userSchema that defines “isAdmin” as a Boolean property.
       1. A hacker using postman may guess the property and set it to be true, everything is now available.
    4. Mass assignment can be addressed via using:
       1. Denylist
       2. Allowlist
    5. In other words: **WE SHOULD NOT BE SENDING THE ENTIRE REQUEST BODY.**