DA summary

Chapter 3 Data Storage

1.collecting data

Big data’s 3V: ***Volume, Variety, Velocity***

The quantity of data collected: in petabytes, increased in terabytes

2.**relational database**: a set of relations

Relation(2 parts): Schema, Instance

Integrity constraints(ICS): condition that must be true for any instance of the database

a legal instance of a relation is one that satisfies all specified ICs

**Primary key**

Super key: a combination of columns that uniquely defines any row within a table

Candidate key

Foreign key: must correspond to a primary key in an another table

Normalization:

1NF: all elements atomic

2NF: every non-prime attribute depends on a candidate key or another non-prime attribute

3NF: some redundancy but dependency preserving

BCNF: no redundancy but not dependency preserving

SQL

DDL (Data Description Language): create, drop, alter   
 DML (Data Manipulation Language): insert, update, delete, select  
 DCL (Data Control Language): grant, revoke  
 TCL (Transaction Control Language): commit, rollback

**ACID transaction**: atomicity, consistency, isolation, durability

Atomicity（原子性）：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

Consistency（一致性）：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

Isolation（隔离性）：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。

Durability（持久性）：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

NoSQL

Not using the relational model; **schemaless**; running well on clusters, tend to be open-source

2 reasons that NoSQL databases are interesting:

application development productivity; large-scale data

Cons: difficult to draw boundaries; doesn’t support ACID transaction; some queries are hard

Pros: helps greatly with running on a cluster

Key points:

**An aggregate is a collection** of data that we interact with as a unit.

Key-value, document, and column-family databases can all be seen as forms of aggregate-oriented database

Aggregates make it easier for the database to manage data storage over clusters.

1) **Key-value database**simple hash table

When to use: storing session information; user profile, preferences; shopping cart data

When not to use: relationships among data; multi-operation transactions; query by data; operations by sets

2) **Document database**

schemaless; documents are stored in the value part of the key-value store

When to use: event logging; content management systems, blogging platforms; web analytics or real-time analytics; E-Commerce Applications

When not to use: complex transactions spanning different operations; queries against varying aggregate structure

3) **Column-family stores**

More efficient column: small amount of columns are computed; new values are supplied for a whole column

More efficient row: many columns of a row are needed; writing a new row

When to use: event logging; content management systems; blogging platforms; expiring usage; need aggregate using (sum or avg)

When not to use: frequent changes to the database (inserts and deletes maybe expensive)

4) **Graph database**

When to use: connected data; routing, dispatch, and location-based services; recommendation engines

When not to use: when you want to update all or a subset of entities; not “graph” data model

NoSQL Databases Goals: not using the relational model; schema less; running well on clusters; aggregates - nested data stored together

NoSQL

Advantages: easy to handle changes; easy to deal with non-uniform data

Disadvantages: database remains ignorant of the schema; implicit schema in the application code

**BASE properties**: basically available; soft state; eventually consistent

Basically available indicates that the system *does* guarantee availability, in terms of the CAP theorem.

Soft state indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.

Eventual consistency indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

NoSQL differs to RDBMS in the way entities get distributed and that no consistency is enforced across those entities

When to use RDBMS: table based; relations between distinct table entities and rows; referential integrity; ACID transactions; arbitrary queries and joins

When to use NoSQL:

If you just want to store your application entities in a persistent and consistent way   
 If you have hierarchical application objects and need some query capability into them   
 If you ever tried to store large trees or networks you will know that an RDBMS is not the best solution here   
 If you are running in the Cloud and need to run a distributed database for durability and availability.   
 You might already use a data warehouse for your analytics. If your data grows to large to be processed on a single machine, you might look into hadoop or any other solution that supports distributed Map/Reduce.