big data Lecture 2: relational databases

wbg231

January 2023

1 data base management systems

- DBMS provide
  1. data integrity and consistency (that is they make sure that data can not be accidentally changed and is correct)
  2. concurrent access (that is two people can look at the same file at the same time)
  3. efficient storage and access
  4. standardized format and methods of administration
  5. standardized query interface (sql)
- there are many types of data base management systems (relational, semi-structured, object oriented, object relational etc)

the relational model

- data is organized into tables called relations
- each column represents a set of possible values
- a relation over sets $A_1 \cdots A_n$ is a subset of there cartesian product
- the row of a table are known as tuples
- any columns could take finite or infinite number of values
- a relation does not need to have all elements of the cartesian product it is just a subset
- relations are the abstract model of the data
- tables are explicably constructed relations
- a view is a relation defined implicitly and constructed dynamically at run time

- temporary tables are the output of queries
- tuples are un ordered
- tuples must be UNIQUE
- there is usually a key for which tuples can not have all of the same values
- **Slido question** if A has 5 elements and B has 3 and $A \cap B = \emptyset$ how many possible relations exist using A
- let $R \subset A \times B$ be a relation we know that $|A \times B| = 15$ and and R is teh power set of that so 2^{15} could also be $(2^{15} + 2^{15} - 1)$ if we say the order of product matters
- a relation is defined by a schema
- it can be tough to consider all edge cases when making a schema
- **slido** what constraints could/should one consider when adding customer name to a field to ensure data integrity
- this is a really hard question and there is not really a good solution, could hurt difference groups

relational data base

- structured data can be encoded by joining relations on a shared attributes
- the collection of schema defines your data model
- keys determine the identity of a row
- simple keys (one column), compound keys (multiple columns)
- can also have primary and alternate keys

foreign keys

- a key form one relation can be used as a column in another relation called a foreign key
- this makes sure that things are consistency between tables
- this is not automatic and must be ensured in the schema definition

normalization

- a normal database does not have any redundant information stored.
- this makes it easy to update data, as you only need to update a record in one place
- but it can also be tough since it requires a lot joints to get complex queries
- schema provide a degree of safety and validation

sql

- sql is the language we use for databases it is declarative ie we state what we want not how we compute it
- that is not the same as C or python which are procedural language
- think of sql as a protocol not a language
- typically iterate over rows in your host language using select statement
- we combine relations by joining data
- always run queries with parameters not variables to avoid SQL injection

types of joints

- cross join all combinations of rows (ie the cartesian product)
- left outer join, right outer join, full outer join: all rows are retained from one or both relations even if no match is found missing data is stored as null
- inner join: only matching rows are retained (like an outer join with out nulls)
- natural join (rows must match on all shared columns this is a a spacial case of inner join)

aggregation

- aggregation lets us summarize multiple tables into a single result
- these are group by statement ex Select zip, AVG(height) from people group by zip
- list of aggregation

1. avg, sum min, max
2. count(distinct X) number of unique values in col x
3. count(*) number of rows
4. count(x) number of non now rows in col x
5. $group_concat(X)$ concatenate string

indexing

- relational databases store data as a list of tuples but this may not be the best way to store things
- if we know how the data will be used we could tell it if the data should be stored in a certain way or what data type to use for storage
- an index is a data structure over one or more columns that can accelerate queries
- an example could be if you know you are going to be grouping by col value, in a lot of searches then indexing by that col value could be faster than just running without the index
- there are some drawbacks of indexing though
  1. they take time and space to construct
  2. updated becomes slower since you also need to update the index
  3. there is no guarantee they will actually speed things up
- when is it good to index
  1. when data is read more often than written
  2. when queries are predictable
  3. when queries rely on a small number of columns
  4. indexes can be added or deleted as needed

integrity

- file systems can not preform actions simultaneously which can make them have inconsistent results
- file systems can not handle concurrent file updates, ie two people could be reading or writing to one file at the same time which can lead to bad results
- WE more or less want our DBMS to be ACID compliant

ACID

- Atomicity ie operations are all or nothing (there are no partial updates operations are bundled in transactions (ie groups of updates))
- consistency updates must move from one valid state to another
- isolation concurrent operations do not depend on order of execution
- durability compiler transactions are permanent once a transaction is completed push it to disk

Atomicity in practice

- basically have a try except statement that if a transaction is successful we commit it, otherwise we roll back (so if something fails we just go back to a valid state)

transaction example

id	balance (≥ 0)
1	\$10
2	\$20

- imagine we have the above table
- we can not transfer 20 from account 1 to account 2, since that would make account 1 negative

consistency

- consistency is maintained by a schema in practice (schemas specificity values for there columns)

isolation in practice

- usually achieved by lacking the database during modification
- this is really important for distributed systems
- git is ACID compliant

- ### slido question

  what if we want to add a new edge and node to a graph that must always be connected which acid property would most easily fix this? Atomicity