→ Flat files : Implementation

① How do we find a particular record?

↓

In the example, we used a for loop to parse each line and have a look if the data matched the condition. Here it worked fine. But what if there were a billion records?

↓

O(n) time complexity for data fetch is a lot.

② What if we now want to create a new application that uses the same database?

↓

for each language, the logic would have to duplicated for parsing the file.

↓

separate code for webapp and separate code for mobile app to access the same database.

③ What if two threads try to write to the same file at the same time?

↓

might lead to data consistency due to overwriting issues by one thread over the other.

→ Flat files : Durability

① what if m/c crashed while our program was updating a record ?

② Replicating the database on multiple m/cs for high availability ?

→ Database Management System : allows applications to store and analyze the information in a database. All the basic issues with the basic approach can be resolved using a DBMS. more like separation of concerns.

↓

reusable across applications.

↓

A general purpose DBMS is designed to allow the definition, creation, querying, update and administration of databases.

↓

assuming databases are stored in disks

↓

bunch of databases out there.

→ Early DBMS

Tight coupling b/w logical and physical layers : Example, defining the kind of datastructure used to store some data and then realising that the api doesn't serve the purpose that the data needed and is actually served by one on a different DS. The whole

data would have to be copied to the other data structure using the application code.

↓

Ted Codd : Relational Model

↓

Database abstraction to avoid this maintenance :

a) store database in simple data structures (tables)

b) access data through high level language

c) Physical storage left up to implementation.

↓

relation synonymous to table

clean separation b/w physical and logical layer ←

if the data is stored at one place, and due to some reason, it is moved to a different type of storage, then the user won't feel a thing.

↓

It's like calling an API. How that API or module handles queries is on the API and is of zero concern to the user.

→ Data Models

- Relational not the only one.
- Data model, high level concept of how we are going to describe the data in our database.

schema : description of a particular collection of data, using a given data model.

•) 1) Relational ← most DBMS ← This course

2) Key / Value
3) Graph
4) Document
5) Column-family
} NOSAL

6) Array / Matrix → Machine Learning → dataframes

7) Hierarchical
8) Network
} Obsolette / Rare

↓

original ones in time around 1960s.

↓

legacy applications

→ Relational model : 1) Structure : Schema → Definition of relations and their contents.

2) Integrity : database's contents satisfy constraints.

3) manipulation : Access and modify a database's content.

↓

Digital Musical Library example

↓

(table) Relation : (unordered set that contains the relationships of attributes), (that represents entities)

(record) tuple : set of attribute values in the relations.

- values are normally atomic (in Codd's paper)
- NULL value is member of each domain.

n-ary relation = Table with n columns

Primary Keys: attributes or set of attributes that uniquely identifies a tuple in a relation.

↓

Some DBMSs automatically create an internal primary key if we don't define one.
(not exposed to us)

↓

Synthetic (creating an id attribute)

foreign key: an attribute from one relation has to map to a tuple in another relation.

↓

for example, there are multiple artists working on an album, but since the fields are supposed to be atomic, this won't be possible. So we define a cross reference table with artist and album to show, whether an artist worked on an album. The artist_id becomes a foreign key for the artist relation, and album_id becomes a foreign key for the album relation.

↓

Protects us from inserting bad data. and we can store multiple artists for a given album

→ Data Manipulation Language (DML): How to store and retrieve information from a database? (relational algebra)

a) Procedural: high level strategy to find desired result

b) Non-procedural : specifies what data, but not how to find it

based on sets ↑ (relational calculus)

→ Relational algebra : 7 fundamental operations :

- σ : Select
- Π : project
- $\cup$: union
- $\cap$: intersection
- $-$: difference
- $\times$: Product
- $\bowtie$: Join

takes one or more relational i/p, and a new relation as o/p.

↓

same schema or attributes as input relation.

1) Select : subset of tuples from a relation that satisfy a given predicate

syntax : $\sigma_{predicate}(R)$

SQL : SELECT * FROM R
WHERE a_id = 'a2' AND
b_id > 102

2) Projection : Generate a new o/p relation that contains only the specified attributes.

syntax : $\Pi_{A_1, A_2, \ldots n}(R)$

example : $\Pi_{b\_id-100,\ a\_id}(\sigma_{a\_id='a2'}(R))$

SQL : SELECT b_id - 100, a_id FROM R
WHERE a_id = 'a2'

3) union : all tuples from either of two relations. (same attributes with same type)

Syntax: (R ∪ S)

SQL: (SELECT * FROM R)

UNION ALL

(SELECT * FROM S)

4) Intersect: all tuples that appear in both (same attributes, same types)

Syntax: (R ∩ S)

SQL: (SELECT * FROM R)

INTERSECT

(SELECT * FROM S)

5) Difference: tuples only appearing in the first.

Syntax: R − S

SQL: (SELECT * FROM R)

EXCEPT

(SELECT * FROM S) (used in testing)

6) Product: Cartesian product or cross joins.

↓

all possible combinations of tuples from i/p relations

Syntax: R * S

SQL: SELECT * FROM R CROSS JOIN S;

SELECT * FROM R, S

7) Join: Natural join → contains all tuples that are a combination of two tuples (one from each input relation) with common values for attributes.

Syntax : R ∞ S

SELECT * FROM R NATURAL JOIN S ;

Here the relations doesn't need to have attributes of same type, rather values are checked, and if they match, they are picked.