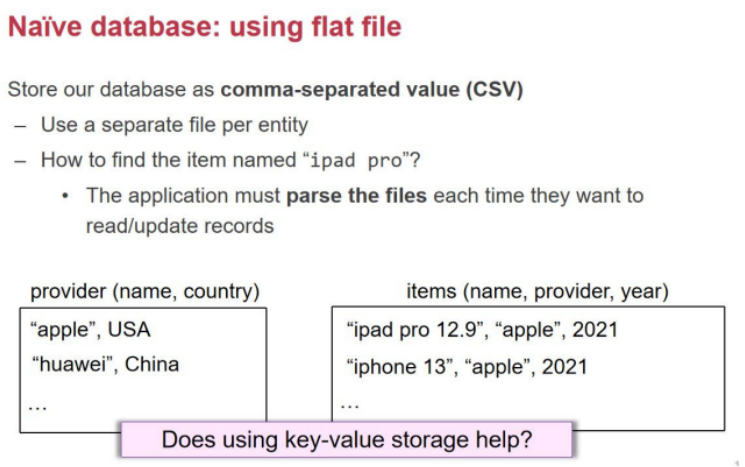
16 database+SQL

我们已经学过 kv-store 和filesystem 之后，我们为什么还需要一个 database，核心在于我们希望对真实世界的数据模式化——更规范地存储信息，更清晰地表示数据之间的关系。

- 如何实现？

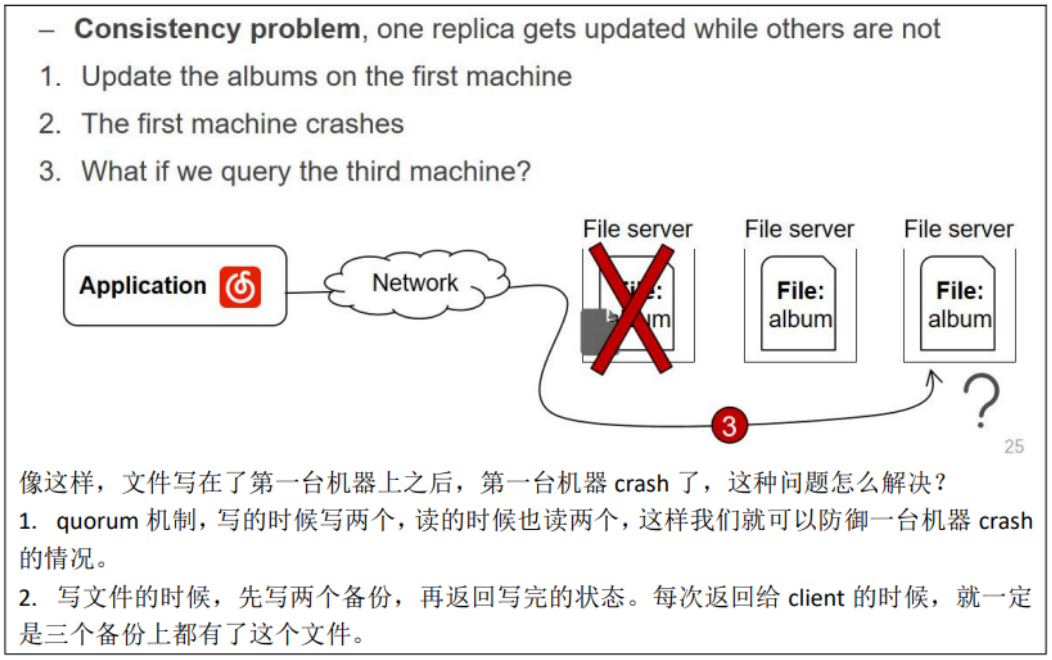


一种最简单的方式就是使用文件，比如 csv 文件，中间使用 comma 分割的一些 value（相当于数据库中的表）。

查找时可以将kv-store运用进来。

但是在查找稍复杂的内容、插入数据正确与否就不够用了，更何况建成大的数据中心。

- 提高availability→数据库副本→一致性问题→ACID



像这样，文件写在了第一台机器上之后，第一台机器 crash 了，这种问题怎么解决？

1. quorum 机制，写的时候写两个，读的时候也读两个，这样我们就可以防御一台机器 crash 的情况。
2. 写文件的时候，先写两个备份，再返回写完的状态。每次返回给 client 的时候，就一定是三个备份上都有了这个文件。

早期的数据库可以认为就是满足ACID、支持 transaction 的 key-value store

DBMS

- 对数据库进一步**解耦**，分离逻辑层和底层，通过分工提高生产力→**DBMS出现的初衷**

Database management system (DBMS)

A DBMS is software that allows applications to **store & analyze** information in a database

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

Application developer does not need to consider the above questions

- The database helps resolving these issues
- Developers only need to write the program using the DBMS data model & query language

- **DBMS最重要的两部分：**

  1、data model

Data models of DBMS

Data model: collection of concepts for describing the data in a database

Schema: a description of a particular collection of data, using a given data model

Example data model:

- Relational
- Key-value
- Graph
- Document
- ...

More expressiveness

Though some models are more nature in specific workloads

- E.g., the *graph* data model is more natural for graph workloads **aka. Polyglot**

Key-value store 也是一种 data model，但是它的表达能力是非常有限的。关系型模型可以作为主体存在，但是其他模型也会在特定任务上表现更好。

kv-store与SQL对比：

如果我们现在插入了包含错误时间的产品会怎么样呢？

(ipad pro, Apple, 2049)

- 如果使用 key-value store 的话，我们就要用如下的代码：

```
inserted_value = ...; // something from user
if inserted_value.year > 2021:
   report an error
kv_item.insert(key,inserted_value);
```

对于一个上层希望不写代码的人来说的话，这就太复杂了，如果我们使用 SQL 的话，我们就可以使用如下代码：

```
CREATE TABLE Item
(
Provider string NOT NULL ,
Year int NOT NULL,
...,
CONSTRAINT Year_Ck CHECK
(Year BETWEEN 1940 AND 2021)
)
```

That is the **declaration**

如果我们要建立数据之间的联系，比如一个 item 属于某个 provider，我们需要先检查是否是一个合法的 provider。

```
inserted_value = ...; // something from user
if kv_provider.get(inserted_value) == NULL:
   report an error
kv_item.insert(key,inserted_value);
```

在 SQL 中，我们就可以直接通过外键绑定的方式，指向 provider name，当我们插入一个错误的 provider name 的时候，SQL 就会报错“a foreign key fails”

Name	Provider	Year
ipad pro	Apple	2021
iphone	Apple	2021

Name	Country
Apple	USA
Huawei	China

```
CREATE TABLE Item
(
Provider string NOT NULL ,...,
FOREIGN KEY (Provider) REFERENCES Provider(Name)
)
```

外键可以解决手动写代码检查有没有 provider 的情况

如果我们要删除一个 provider 怎么办？就是把一个公司的所有物品删掉。在 key-value store 中实现这个功能，就是做一个遍历，把对应的 key 删掉

```
for k,v in kv_item:
    if v.provider == "Apple":
        kv_item.delete(k)
```

SQL 只需要加上这一行即可

```
CREATE TABLE Item
(
Provider string NOT NULL ,...,
FOREIGN KEY (Provider) REFERENCES Provider(Name)
ON DELETE CASCADE
)
```

而在数据库中的 best practice 是标记为“已删除”、“已离职”，因为数据太重要了，最好不要删掉。

kv-store 和 database 哪个更好取决于在实际运用中哪个更方便。

kv-store 是相对简单的，SQL 能表达的数据是它的超集。我们在数据库中可以简单定义一个 primary key，那么整张表就变成了一个 kv-store。

2、query language

查询语言是比较重要的，比如可以用自然语言，但是自然语言的表达有歧义、不够精确。我
们可以用专用的关系代数和集合论中的符号。有一套数学理论作为支撑。

我们先来看一个 query，如果用 key-value store 来做

Example #1: Using ~~SQL~~ to query the data

Example #1: find the item released in 2021

- How to query the data with the key-value store model ?
- What about data query efficiency?
  - Solution: use an **index (e.g., hashtable)**

```
result = []
for k,v in kv_item:
    if v.year == 2021:
        result.append(v)
print(result)
```

Opt.

Add an index offline

```
kv_item_year
for k,v in kv_item:
    kv_item_year[v.year].append(k)
```

```
result = []
for k in kv_item_year[2021]:
    result.append(kv_item[k])
print(result)
```

为了提高性能，最常见的方法就是加一个 offline 的 index。对于任何一年都有这个索引，当我们查询的时候就不需要遍历这个 kv_item 集合。index 建立的越多，就是空间换时间的过程。

如果我们用 SQL 去做，只需要如下来写，也就是在整张 table 中运用 Year = 2021 这个谓词来做筛选，

Example #1: Using SQL to query the data

Abstraction: `select`

- Choose a subset of the tuples from a relation that satisfies a selection **predicate**
- Can combine multiple predicates using conjunctions / disjunctions

Syntax: $\sigma_{predicates}(R)$

- E.g., $\sigma_{year=2021}(items)$

Example #1: find the item released in 2021

- How about adding an **index**?

```
select * from Item where Year=2021;
```

```
CREATE INDEX item_year_index ON
Item(Year);
```

```
mysql> select * from item where Year=2021;
+--------------+----------+------+
| Name         | Provider | Year |
+--------------+----------+------+
| ipad pro     | apple    | 2021 |
| mackbook 14  | apple    | 2021 |
+--------------+----------+------+
2 rows in set (0.00 sec)
```

SQL 毫无疑问也是可以加 INDEX 的。

SQL contains three components

Data Manipulation Language (DML)

– E.g., query language based on relational algebra + aggregators **and more!**

```
select * from Item where Year=2021;
```

Data Definition Language (DDL)

– E.g., how to create a table

```
CREATE TABLE Item ( Provider string NOT NULL , Year
int NOT NULL, ...,)
```

Data Control Language (DCL)

– E.g., whether a user has access permissions on a specific table

This course does not intend to teach the languages of SQL.
But is why it is designed this way.

6

有了这三部分之后，我们就定义了数据的动态、静态和权限控制。

SQL&transaction

它们的相似点：两者都实现了 ACID
它们的不同点：没有 SQL 的话，开发者需要手写事务机制

```
...
tx.begin();
...
tx.read();
...
tx.commit();
...
```

Transaction w/o SQL

```
START TRANSACTION
select * from ...;
... // other SQL
COMMIT;
...
```

Transaction w/ SQL

关系型模型是理论角度去阐述，SQL 是关系型模型的实现，在 SQL 实现中 row 就是 tuple， table 就是 relation。

Table schema design of relational mod

如果让我们来实现这么一个简历的页面，我们要怎么设计这个表呢？

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

我们可以用非常简单的方法记录。

ID	Name	Area
0	Bill Gates	greater seattle area

...

...

但是住在什么地方这个事情，如果有很多人坐在同一个地方，最后这个位置改名了，那么需要全部都改。我们可以把 area 用像指针一样的外键来处理。这就是 many to one 问题，本质就是 replication 问题。

Many-to-one: A matter of duplication, and normalizing

Store the string instead of ID is actually a matter of **duplication**

- The same value has appeared in many entities
- E.g., the “greater seattle area” appeared in all the people who lived there

Drawbacks of duplication:

- Consistent style and spelling
- Ambiguity (e.g., several cities with the same name)
- Hard for updating. Suppose the city has changed its name

Normalizing: remove the duplicated entities in various entities

- Useful for many-to-one relationship
- How to normalize is a big topic (out of the scope of this lecture)

另外一个是 one-to-many problem，一个人可以有多个教育背景。我们该怎么设计这个表呢？

1. 把表设计的大一点，高中、大学、master 都作为一列，上过就填入对应信息。但是这种方式是把所有的可能都变成了 column，浪费空间。
2. 我们应该把教育单独地变成一个表。

Solution: using another table

– And constraint the relationship with foreign key!

Table: user

ID	Name	Area
0	Bill Gates	us:91

Table: education

ID	User-ID	School
807	0	Harvard
806	0	Lakeside school

Document，比如 json 格式，解决了 one-to-many 问题，也就是 jsonarray。好处非常多，locality 非常好，所有信息都在一个文件中。Schema 也很灵活。但是我们要修改地名的话，many-to-one 还是有问题。

- 总结：

Summary: relational model vs. document model

Document model is a representative model in NoSQL databases

Benefits of **document model**

– Better locality
– Easy to represent one-to-many relation (but has duplication problem)
– Schema flexibility

Benefits of **relational model**

– Join supports
– Support one-to-many without duplication (e.g., with multiple tables)
– Better modeling many-to-one & many-to-many relationships