CSE 199: UB Seminar (Fall 2026) - Data management

Lecture 4: RDBMS and NoSQL

Zhuoyue Zhao
zzhao35 [at] buffalo [dot] edu

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Why Relational Database Management Systems?

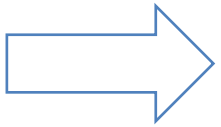
- **Flexible data modeling and query languages (last lecture)**

- **Can implement correct and performant query engines**
  - CSE 350, CSE 4/562, CSE 662
  - This lecture: an overview, and beyond RDBMS

Formal semantics of a basic SQL SELECT

- Selection: $\sigma_P R$
  - Selects the records in relation R that satisfy a predicate P
- Projection: $\pi_E R$
  - Evaluates and output expression(s) E on every row in R
- Sort is not a standard relational algebra operator
  - As its name suggest, it orders the output.

```
SELECT E
FROM R
WHERE P
ORDER BY S
```

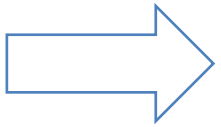


$Sort_S(\pi_E \sigma_P R)$

Formal semantics of a basic SQL SELECT with join

- Selection: $\sigma_P R$
  - Selects the records in relation R that satisfy a predicate P
- Projection: $\pi_E R$
  - Evaluates and output expression(s) E on every row in R
- Sort is not a standard relational algebra operator
  - As its name suggest, it orders the output.
- Cartesian product: $R1 \times R2$
  - Concatenates every pair of tuples $t_1 \in R_1, t_2 \in R_2$ into a tuple as output

```
SELECT E
FROM R1 JOIN R2
   ON R1.x = R2.x
WHERE P
ORDER BY S
```



$$Sort_S(\pi_E \sigma_P \sigma_{R1.x=R2.x} R_1 \times R_2)$$

Join can be expensive

- $R1$ has N rows, $R2$ has M rows
  - $R1 \times R2$ has NM rows

- Evaluating SQL strictly following the procedure described by its RA semantics
  - Very inefficient

- Query optimization
  finding alternative algorithms/execution plans to reduce cost

Naïve evaluation is expensive

```
SELECT * FROM STUDENT JOIN ENROLLMENT ON S.sid = E.sid
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7

$student \times enrollment$

sid	name	login	major	adm_year	sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
100	Alice	alicer34	CS	2021	102	s22	562	2.3
More results follows …… (9 tuples in total)								

$\sigma_{S.sid=E.sid} student \times enrollment$

Quadratic time

sid	name	login	major	adm_year	sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
102	Charlie	charlie7	CS	2021	102	s22	562	2.3
100	Alice	alicer34	CS	2021	100	f21	560	3.7

Optimized evaluation

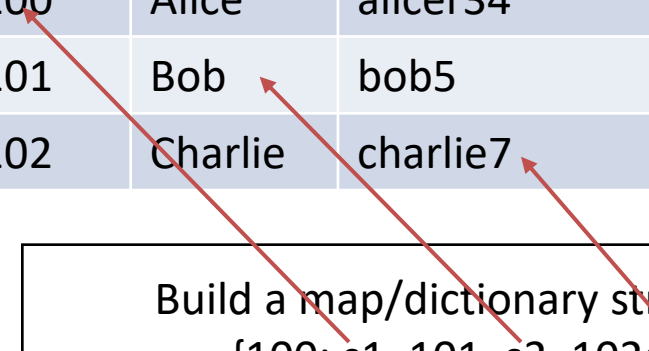
```
SELECT * FROM STUDENT JOIN ENROLLMENT ON S.sid = E.sid
```

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021

enrollment

sid	semester	cno	grade
100	s22	562	2.0
102	s22	562	2.3
100	f21	560	3.7



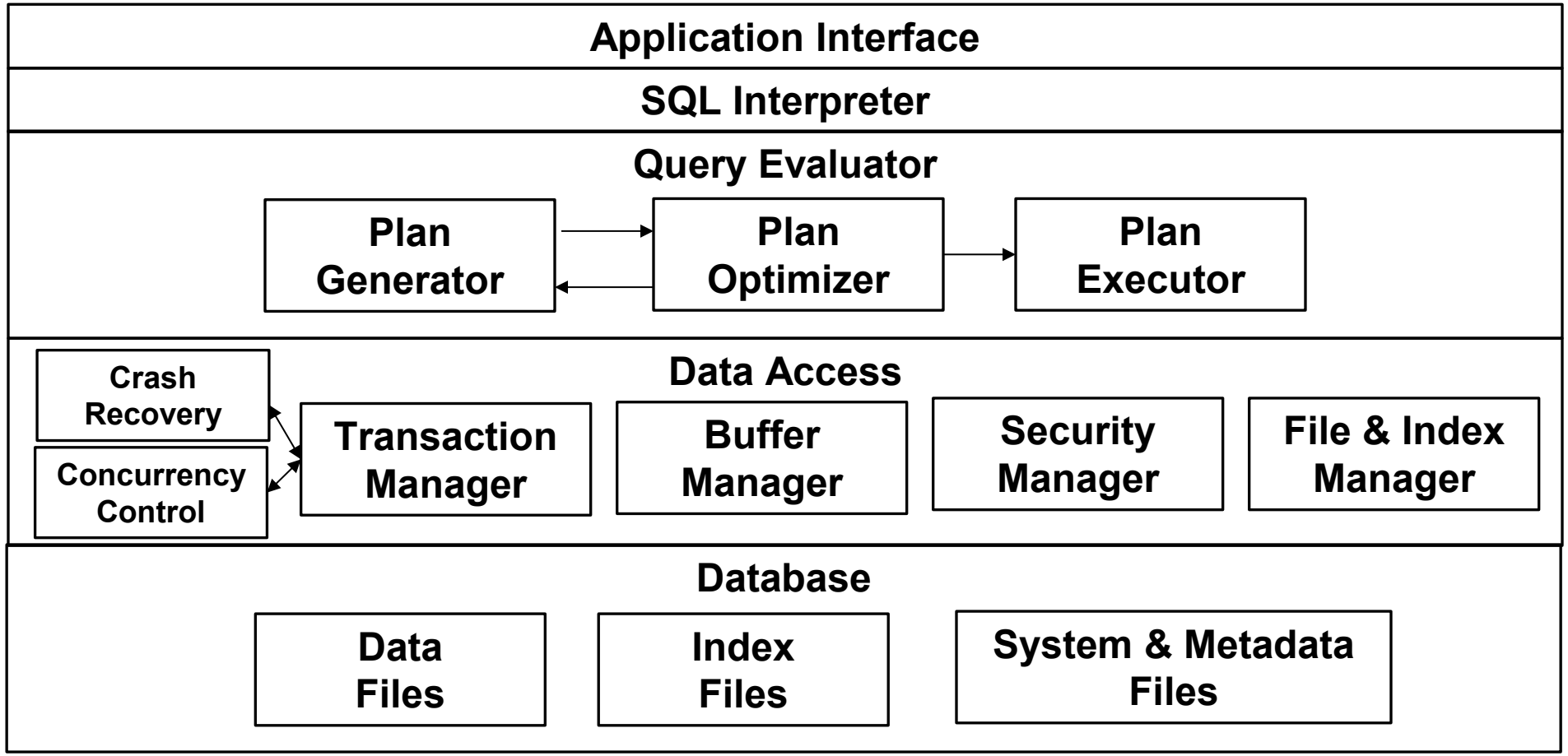
For each e in enrollment:
    lookup s in student using sid = 100
    output $e \circ s$

Linear time

sid	name	login	major	adm_year	sid	semester	cno	grade
100	Alice	alicer34	CS	2021	100	s22	562	2.0
102	Charlie	charlie7	CS	2021	102	s22	562	2.3
100	Alice	alicer34	CS	2021	100	f21	560	3.7

DBMS architecture

- DataBase Management System (DBMS) is a software system for convenient and efficient data access over databases.



Does RDBMS solve all the problems?

- Lots of missing values or inconsistent data fields available for data?
  - Needs more flexible schema
  - Document database (e.g., MongoDB)

- Larger than single machine capacity; need very fast retrieval by key?
  - Distributed key-value store (e.g., Canssandra)

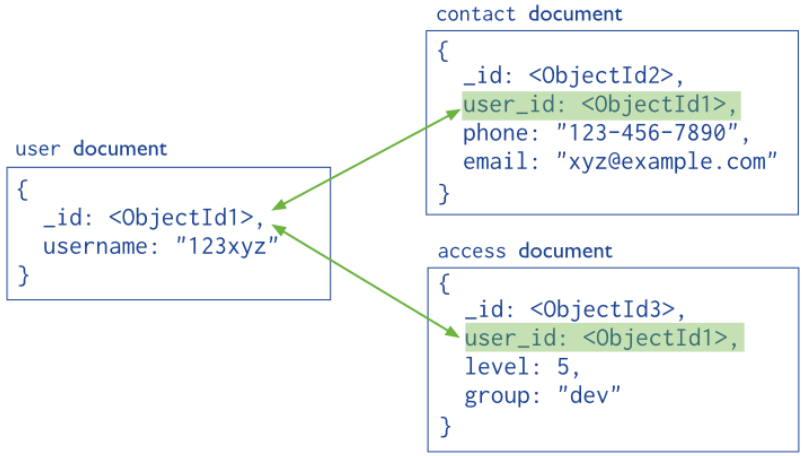
- And many more…

MongoDB: document data model

MongoDB has a *flexible schema*

References

store relationships
between data by
including links
or references



Embedded Data

denormalized data model

MongoDB Queries -- JSON data

```
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      {
         "type": "home",
         "number": "212 555-1234"
      },
      {
         "type": "fax",
         "number": "646 555-4567"
      }
   ],
   "gender":{
      "type":"male"
   }
}
```

JSON records describe the data they hold

New data elements in a records can be added or removed as needed.

MongoDB Queries -- simple data extraction

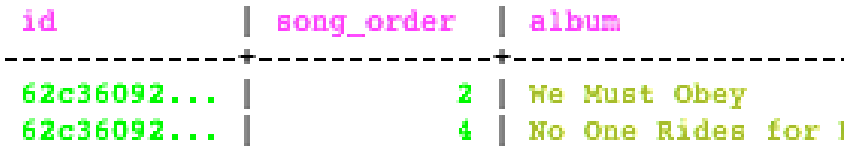
- An empty document query selects all documents in the collection:
  - `db.inventory.find( { } )`
- Equality conditions use { <field>: <value> } to select all documents that have that field with the specific value
  - `db.inventory.find( { type: "snacks" } )`
- Join support: done through $lookup operator
  - often less efficient than SQL

Apache Cassandra: Overview

- Distributed **database** management system
- Large amounts of data across many commodity servers
- Support for clusters spanning **multiple datacenters**
- Supports replication over multiple datacenters
- Fault-tolerant
- High performance and scalability
- Cassandra File System (HDFS compatible)
- Written in Java
- Cassandra Query Language (CQL)
- **NoSQL system**
- NO support for joins or subqueries => **denormalization**

Cassandra Data Modeling

Rows are organized into tables



```
id          | song_order | album
------------+------------+------------------
62c36092... |          2 | We Must Obey
62c36092... |          4 | No One Rides for
```

The first component of a table's primary key is the **partition key**

Within a partition, **rows are clustered** by the remaining columns of the key.

Other columns may be **indexed separately** from the primary key

Cassandra Data Modelling Music Example

```
CREATE TABLE songs (
  id uuid PRIMARY KEY,
  title text,
  album text,
  artist text,
  data blob
 );
```

```
CREATE TABLE playlists (
  id uuid,
  song_id uuid,
  PRIMARY KEY  (id, song_order )
);
```

SELECT * FROM playlists, songs, WHERE songs.id = playlists.song_id;

JOIN ON TWO TABLES

How to do join efficiently in a distributed fashion? (Online / Web Applications

Cassandra Data Modelling Music Example

```
CREATE TABLE songs (
  id uuid PRIMARY KEY,
  title text,
  album text,
  artist text,
  data blob
 );
```

```
CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid,
  title text,
  album text,
  artist text,
  PRIMARY KEY  (id, song_order )
);
```

id	song_order	album	artist	song_id	title
62c36092...	1	Tres Hombres	ZZ Top	a3e64f8f...	La Grange
62c36092...	2	We Must Obey	Fu Manchu	8a172618...	Moving in Stereo
62c36092...	3	Roll Away	Back Door Slam	2b09185b...	Outside Woman Blues

flatten the tables (denormalization)

```
SELECT * FROM playlists;
```

Quiz time

- Submit your answers in tophat