

# Lecture 1: Course Introduction



# Welcome to CS 4423/6423!

- Joy Arulraj (School of Computer Science)
- This course is all about building database systems.
- Why do we even care about database systems?



# Importance of Database Systems

Banking

Healthcare

Airlines

E-Commerce

# Why take this course?

Curiosity

Scalability

Efficiency

Versatility

# Why take this course?

Storage Management

Query Optimization

Index Structures

SIMD Instructions



# Course Overview



# Course Objectives

- Learn about building a database system from scratch.
- Become proficient in systems programming.
- Understand the impact of hardware trends on software design.

# Course Topics

- This course focuses on the internals of a database system:
  - Logging and Recovery
  - Concurrency Control
  - Query Optimization
  - Potpourri of advanced topics

# Previous Course (4420/6422)

- This course builds upon a prior course that covered:
  - Relational Databases
  - Storage Management
  - Index Structures
  - Query Execution

# Expected Background

- Should have taken an introductory course on computer systems.
- All programming assignments will be in C++.
  - Programming assignment #1 will help get you caught up with C++.
  - If you have not encountered C++ before, need to put in extra effort.
  - Use a large language model like ChatGPT for assistance.
  - Relevant parts of C++ will be briefly covered in this course.

# Course Logistics

- Course Website (link on Canvas)
- Discussion Tool: Ed (link on Canvas)
- Grading Tool: Gradescope (link on Canvas)
- In-Class Quiz Tool: Point Solutions (link on Canvas)

# Course Rubric

- Exams (40%)
- Programming Assignments (20%)
- Exercise Sheets (15%)
- In-Class Quizzes (15%)
- Project based on [TokenSmith](#) (10%)

# TokenSmith: Local LLM-based Textbook Tutor

## DB Learning Assistant

Principles and Paradigms" by Paul D. Gray, which cover the topic in detail.

how does two-phase locking protocol ensures conflict serializability

The two-phase locking protocol ensures conflict serializability through the following steps:

- Growing Phase**: During this phase, transactions can acquire locks but cannot release any locks. This ensures that no transaction releases locks it does not hold, preventing conflicts.
- Shrinking Phase**: This phase is when transactions can release locks but cannot acquire new locks. This phase ensures that transactions do not attempt to acquire locks they do not hold, again preventing conflicts.

By alternating between the growing and shrinking phases, the protocol ensures that all transactions can read and write to data while avoiding conflicts. Once all transactions have completed their shrinking phase, conflicts are resolved, ensuring that the final state is serializable.

Ask a question about your textbook...

## Textbook

100%

Silberschatz-Korth-Sudarshan: Database System Concepts, Fourth Edition VII. Other Topics 21. Application Development and Administration © The McGraw-Hill Companies, 2001

Exercises 813

21.6 Suppose a system runs three types of transactions. Transactions of type A run at the rate of 50 per second, transactions of type B run at 100 per second, and transactions of type C run at 200 per second. Suppose the mix of transactions has 25 percent of type A, 25 percent of type B, and 50 percent of type C.

- What is the average transaction throughput of the system, assuming there is no interference between the transactions.
- What factors may result in interference between the transactions of different types, leading to the calculated throughput being incorrect?

21.7 Suppose the price of memory falls by half, and the speed of disk access (number of accesses per second) doubles, while all other factors remain the same. What would be the effect of this change on the 5 minute and 1 minute rule?

21.8 List some of the features of the TPC benchmarks that help make them realistic and dependable measures.

21.9 Why was the TPC-D benchmark replaced by the TPC-H and TPC-R benchmarks?

21.10 List some benefits and drawbacks of an anticipatory standard compared to a reactionary standard.

21.11 Suppose someone impersonates a company and gets a certificate from a certificate issuing authority. What is the effect on things (such as purchase orders or programs) certified by the impersonated company, and on things certified by other companies?

### Project Suggestions

Each of the following is a large project, which can be a semester-long project done by a group of students. The difficulty of the project can be adjusted easily by adding or deleting features.

**Project 21.1** Consider the E-R schema of Exercise 2.7 (Chapter 2), which represents information about teams in a league. Design and implement a Web-based system to enter, update, and view the data.

**Project 21.2** Design and implement a shopping cart system that lets shoppers collect items into a shopping cart (you can decide what information is to be supplied for each item) and purchased together. You can extend and use the E-R schema of Exercise 2.12 of Chapter 2. You should check for availability of the item and

< 1C of 916 >

# Course Policies

- Programming assignments & exercise sheets must be own work.
  - Not group assignments.
  - You may not copy source code from other people or the web.
  - Plagiarism will not be tolerated.
  - We will follow the late submission policy listed on Canvas.
- Academic Honesty
  - Refer to Georgia Tech Academic Honor Code.
  - If you are not sure, ask me.



# Textbooks for Reference

- Silberschatz, Korth, & Sudarshan:
  - Database System Concepts. McGraw Hill, 2020.
- Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom:
  - Database Systems: The Complete Book. Prentice-Hall, 2008.

# Intro Sheet

- Upload a one-page PDF with your details on Gradescope.
  - Picture (ideally 2x2 inches of face).
  - Name, interests, and other details mentioned on Gradescope.
- Purpose of this sheet
  - Help me know more about your background for tailoring the course.
  - Recognize you in class.



# In-Person Office Hours

- Sign up for a ten-minute slot in the sign-up sheet (link on Canvas)
- Teaching assistants will guide you with assignments & sheets.

# Auditing + Late Policy

- Course not tailored for auditing or P/F mode
- Late Policy: 25% reduction in grade for every late day
- 4 penalty-free late days for the entire semester

# Motivating Application



# Social Media Analytics Application

Social  
Media  
Analytics

Social Trends

Sentiments

Interactions

# Flat-File Database System

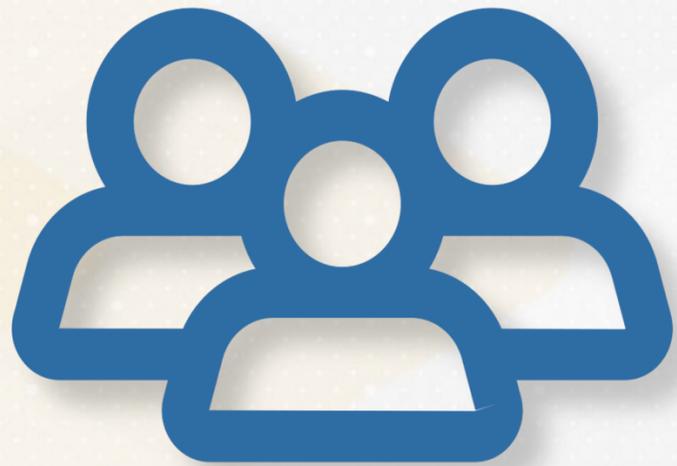
Data

Users.txt

Posts.txt

Interactions.txt

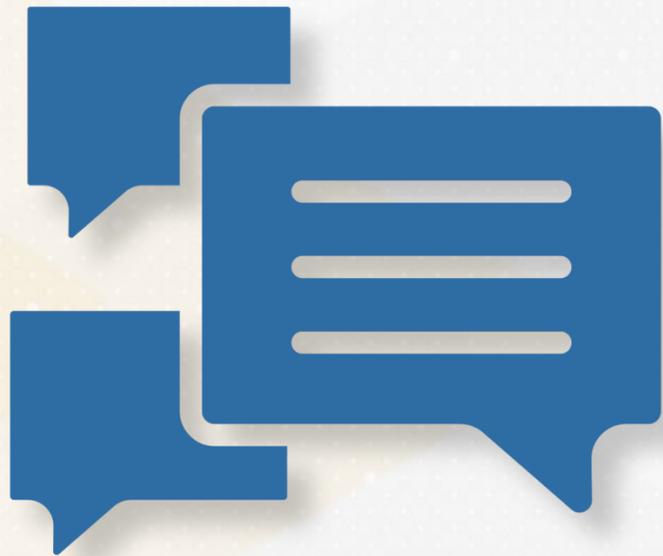
# Users Text File



Users.txt

UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

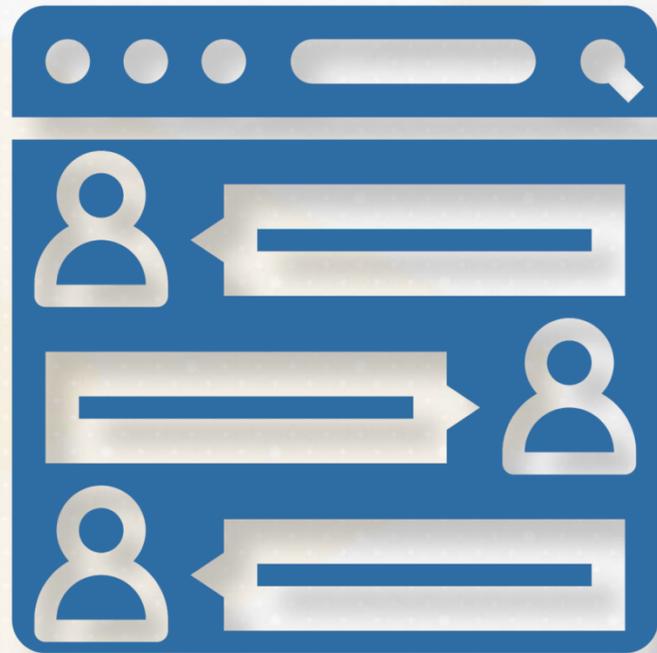
# Posts Text File



Posts.txt

PostID,	UserName,	Location
1001,	Timothée Chalamet,	Excited to start filming my new movie!
1002,	Lana Condor,	Had a great time at the beach today! 
1003,	Liu Yifei,	Enjoying the scenery in Beijing! 
1004,	Burna Boy,	Live performance tonight in Lagos! 
1005,	Kriti Sanon,	Loving the vibrant energy of Mumbai! 

# Interactions Text File



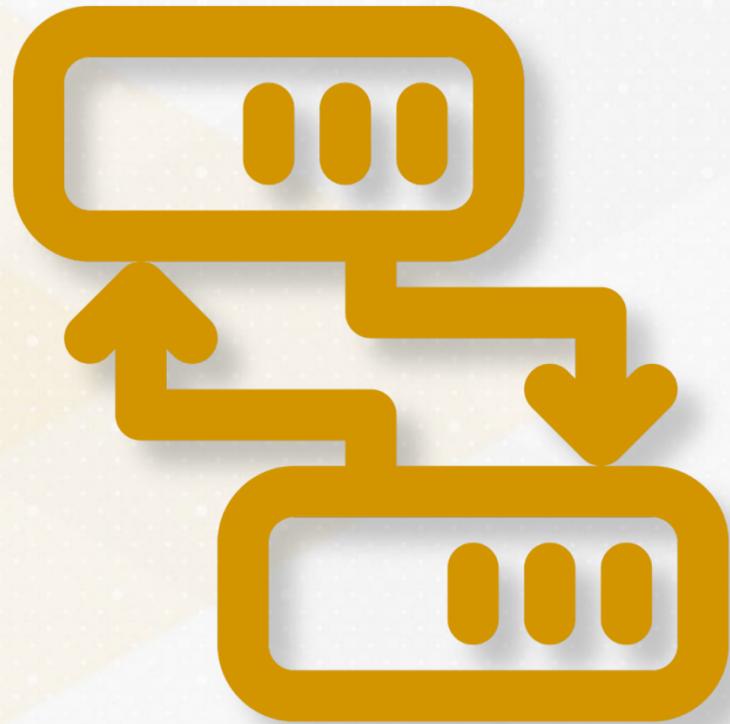
Interactions.txt

PostID,	UserName,	Reaction Type,	Comment
1001,	Lana Condor,	Comment,	Love it!
1002,	Liu Yifei,	Like,	-
1003,	Burna Boy	Like,	-
1004,	Kriti Sanon	Comment,	Wish I could be there!

# Limitations of Flat-File Database



# Limitation #1: Data Redundancy



PostID,	UserName,	PostContent
1001,	Timothée Chalamet,	Excited to start filming my new movie!
1006,	Timothée Chalamet,	Exploring the streets of Paris! 🇫🇷
1007,	Timothée Lamet,	Just wrapped up a day of filming 🎬
1008,	Timothée Chalamet,	Any book recommendations?

# Limitation #2: Slow Operations

<b>UserName</b>	<b>Location</b>
Timothée Chalamet,	Paris
<del>Lana Condor,</del>	<del>Los Angeles</del>
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai



# Limitation #3: Slow Queries

UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

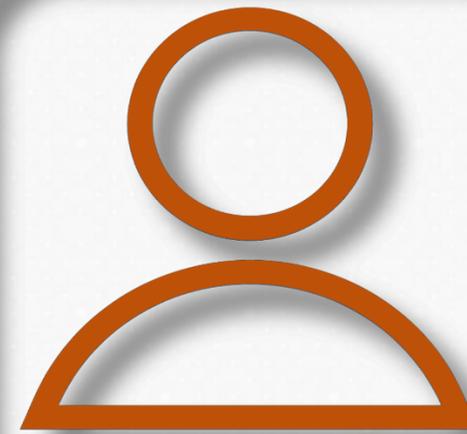
# Limitation #4: Concurrent Updates

USER 1



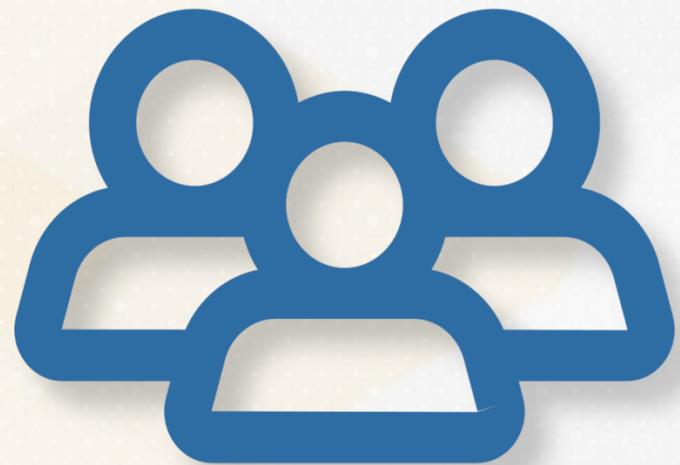
Xavier Laurent,  
Paris

USER 2



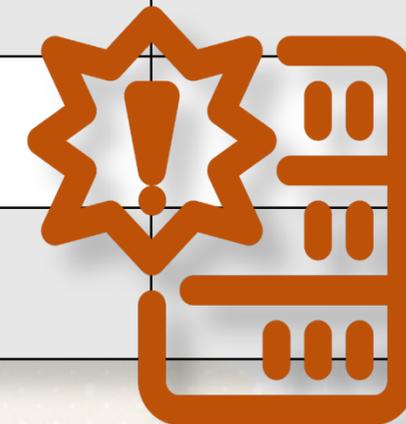
Xavier Laurent,  
New York

# Limitation #5: Handling Disk Failure



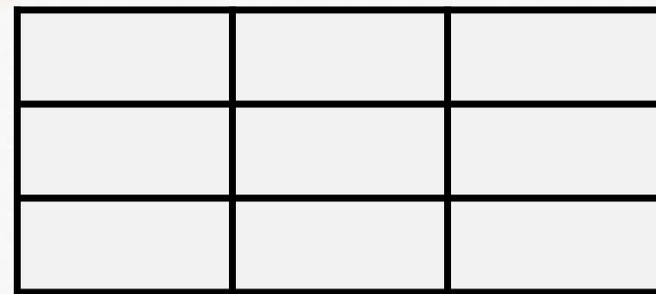
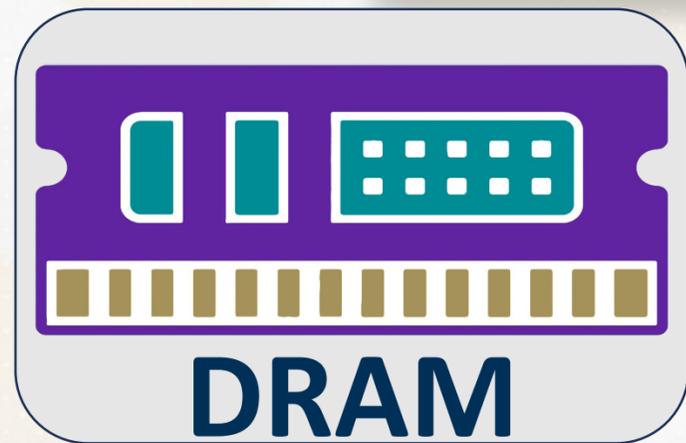
**Users.txt**

UserName,	Location,	Country
Timothée Chalamet,	Paris,	France
Lana Condor,	Los Angeles,	USA
Liu Yifei,	Beijing,	China
Burna Boy,	Lagos	
Kriti Sanon,	Mumbai	

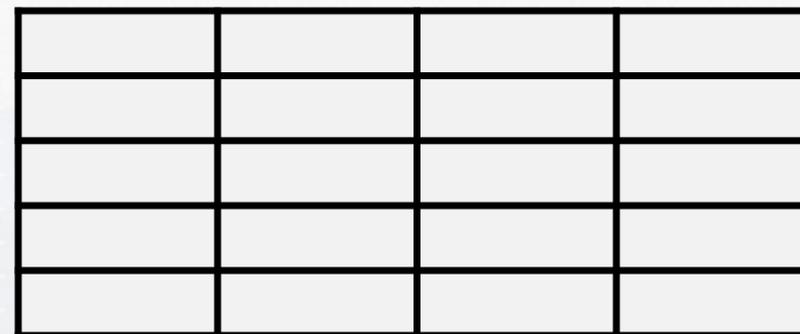


# Limitation #6: Memory Management

*Faster access - not durable*



*Cached Pages*



*Database*

*Slower access - but durable*

# Limitation #7: Usability

Custom Code

Comments Query Code

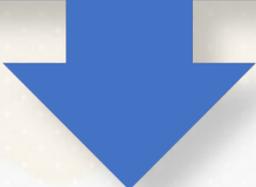
```
def get_comments_by_user(file_path, user_name):
    comments = []
    with open(file_path, 'r') as file:
        for line in file:
            post_id, user, reaction_type, comment_text = line.strip().split(', ')
            if user == user_name and reaction_type == "Comment":
                comments.append((post_id, comment_text))
    return comments
```

# Relational Database



# Relational Database

**Column / Attribute**



**Row/  
Tuple**



UserName	Location
Timothée Chalamet	Paris
Liu Yifei	Beijing
Burna Boy	Lagos
Kriti Sanon	Mumbai

# Relational Database

UserID	UserName	Location
1	Timothée Chalamet	Paris
2	Lana Condor	Los Angeles
3	Liu Yifei	Beijing
4	Burna Boy	Lagos
5	Kriti Sanon	Mumbai

Users

PostID	UserID	PostContent
1001	1	Excited to start filming my new movie!
1002	2	Had a great time at the beach today! 🌊☀️
1003	3	Enjoying the scenery in Beijing! 🏞️
1004	4	Live performance tonight in Lagos! 🎤🎵
1005	5	Loving the vibrant energy of Mumbai! 🌃

Posts

PostID	UserID	ReactionType	Content
1001	2	Comment	Love it!
1002	3	Like	-
1003	4	Like	-
1004	5	Comment	Wish I could be there!

Interactions

# Relational Database

List of Tables

Logical



Physical

Storage  
Formats

Indexing Data  
Structures

# Relational Database

Logical Database  
Design

Simple Query Language for  
Complex Data Manipulation

Physical Database  
Design

Optimize Indexing for Storage  
Hardware

# Benefits of Relational Database



# Benefit #1: No Data Redundancy

UserID	UserName	Location
1	Sir Timothée Chalamet	Paris

PostID	UserID	PostContent
1001	1	Excited to start filming my new movie!
1006	1	Exploring the streets of Paris!
1007	1	Just wrapped up a day of filming
1008	1	Any book recommendations?

# Benefit #2: Fast Operations



❖ Efficient Data Deletion

❖ User (Tuple) Removal

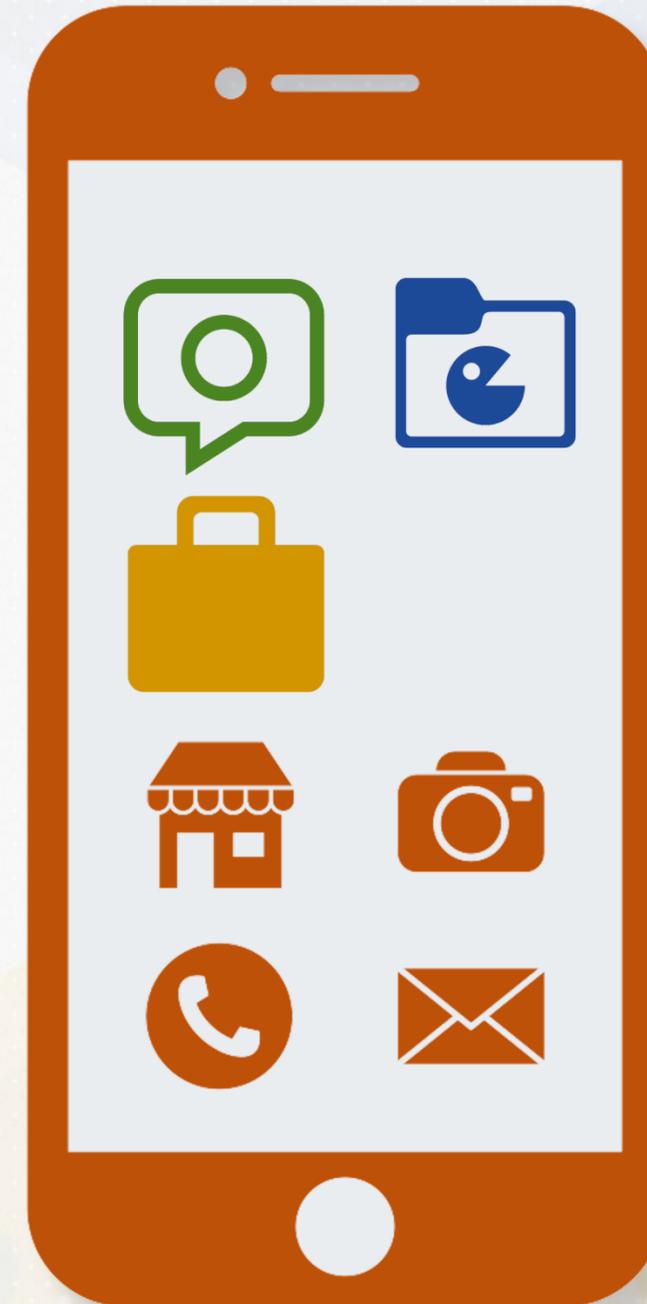
❖ Fast Deletion

# Benefit #3: Fast Queries

Index Database

Apps in labeled  
folders

Location-based  
index

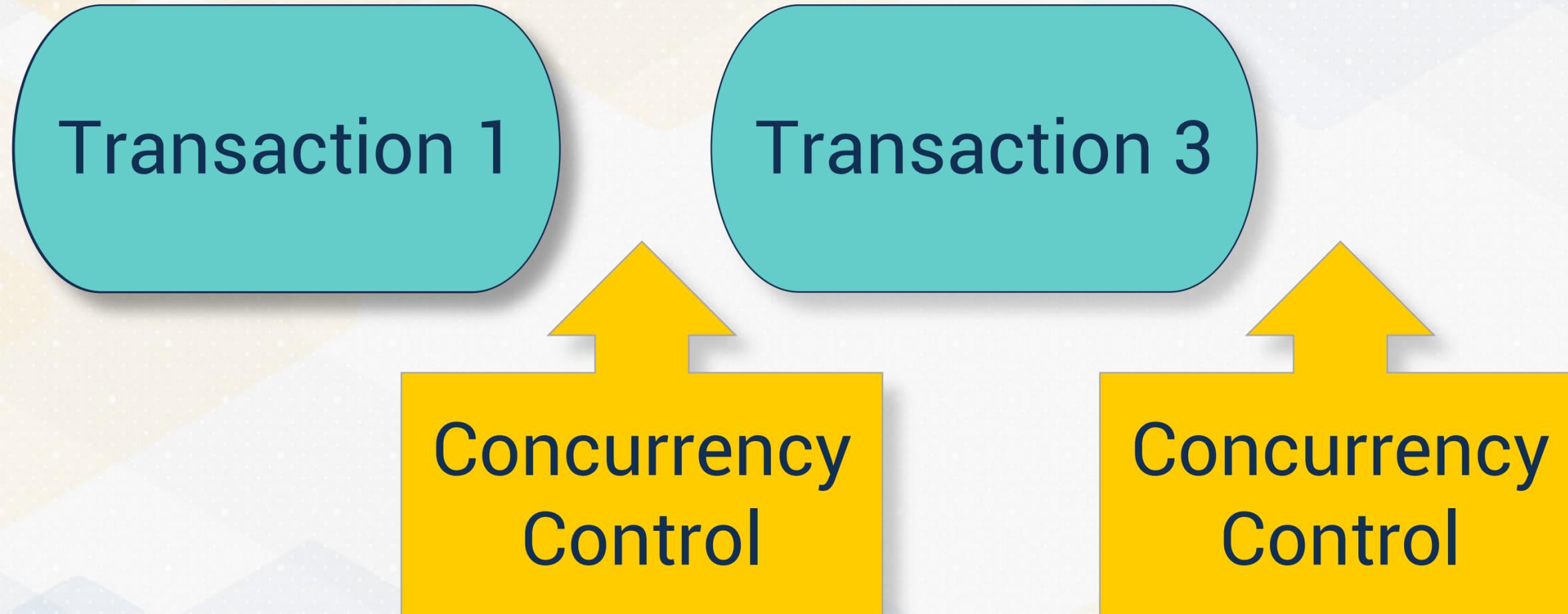


# Benefit #3: Fast Queries

```
SELECT *  
FROM Users  
WHERE LOCATION = 'Mumbai';
```

UserName,	Location
Timothée Chalamet,	Paris
Lana Condor,	Los Angeles
Liu Yifei,	Beijing
Burna Boy,	Lagos
Kriti Sanon,	Mumbai

# Benefit #4: Concurrent Updates



USER 1



Timothée Chalamet,  
Paris

USER 2



Xavier Laurent,  
New York

USER 2



Timothée Chalamet,  
Paris

# Benefit #5: Handling Failures

Atomic Transactions

“All or Nothing”

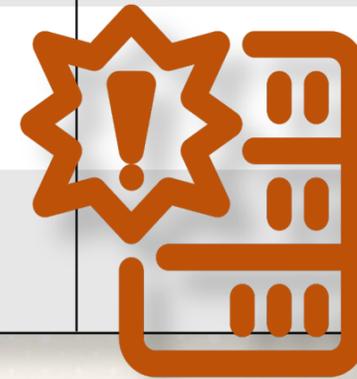
UserName	Location	Country
Timothée Chalamet	Paris	France
Lana Condor	Los Angeles	USA
Liu Lifei	Beijing	China
Burna Boy	Lagos	
Kriti Sanon	Mumbai	

Reversion



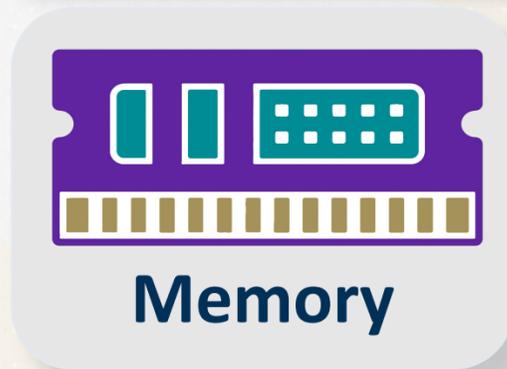
# Benefit #5: Handling Failures

UserName	Location	Country
Timothée Chalamet	Paris	France
Lana Condor	Los Angeles	USA
Liu Lifei	Beijing	China
Burna Boy	Lagos	
Kriti Sanon	Mumbai	



# Benefit #6: Memory Management

*Faster access  
- not durable*



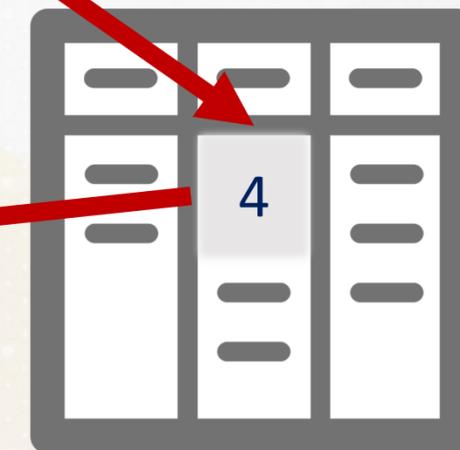
2		3

*Cached Pages*



1			
			5

*Database*



*Transaction Log*

*Slower access  
- but durable*

# Benefit #7: Usability

UserName	Location
Timothée Chalamet	Paris
Lana Condor	Los Angeles
Liu Yifei	Beijing
Burna Boy	Lagos
Kriti Sanon	Mumbai

SQL = Declarative

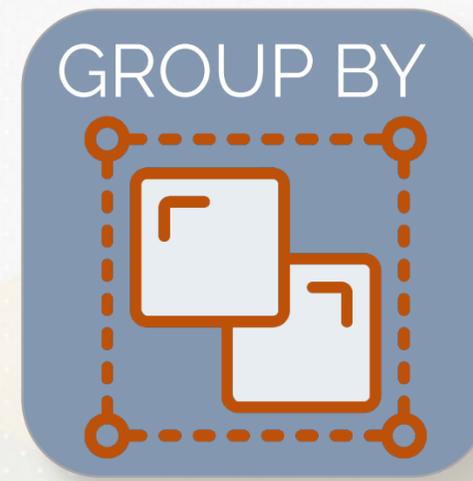
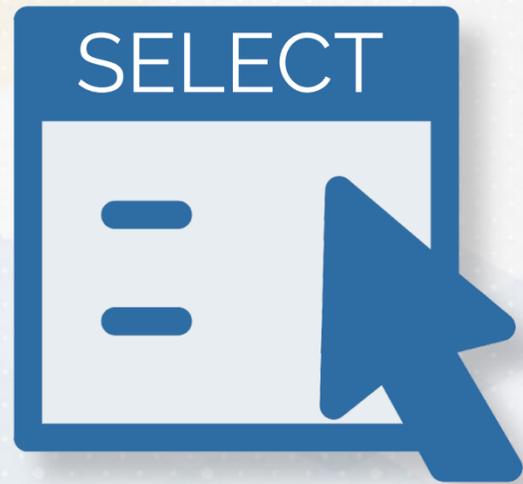
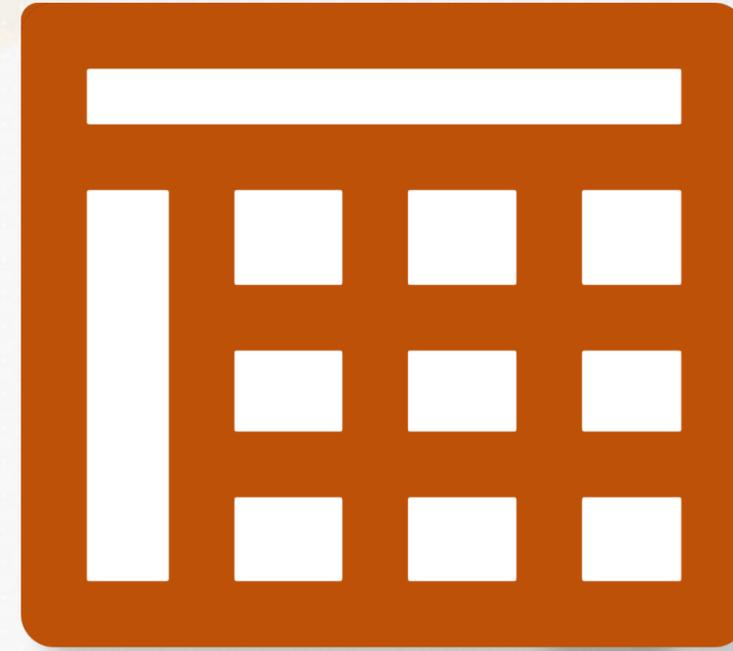
Python, C++ = Imperative

# Relational Operators

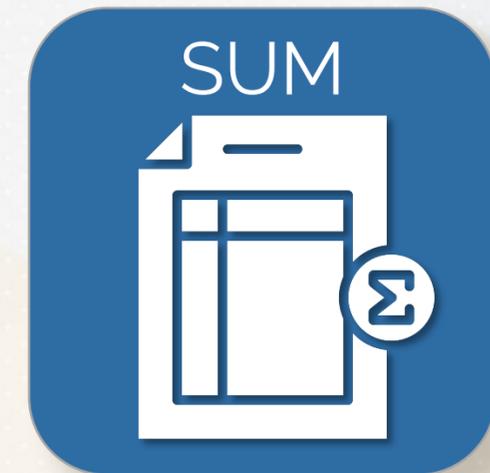
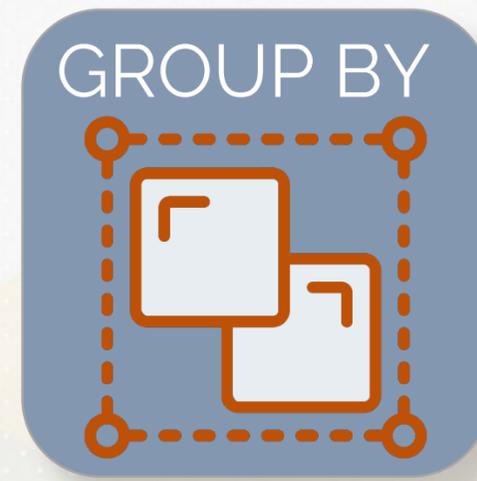
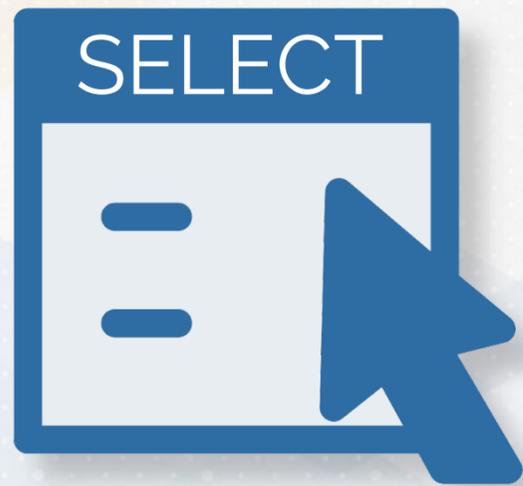


# Relational Operators

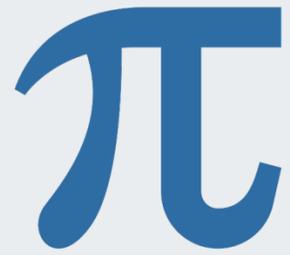
Relations



# Relational Operators



# SELECT (Projection Operator)



Select specific columns from a table

Example: Retrieve locations of all users.

```
SELECT Location
```

```
FROM Users;
```

# WHERE (Selection Operator)



Filters rows based on specified conditions

Example: Find all interactions that are "Like" reactions.

```
SELECT *
```

```
FROM Interactions
```

```
WHERE ReactionType = 'Like';
```

# GROUP BY (Grouping Operator)

Y

- Groups rows of same values
- Used with aggregate functions like SUM

Example: Count the number of reactions that each post received.

```
SELECT PostID,  
       COUNT(*) AS ReactionCount  
FROM Interactions  
GROUP BY PostID;
```

# SUM (Aggregation Operator)

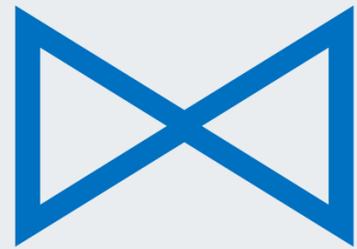


- Adds group values
- Defined by **GROUP BY** clause

Example: Total number of posts made by each user, grouping the results by UserID.

```
SELECT UserID,  
       COUNT(PostID) AS TotalPosts  
FROM Posts  
GROUP BY UserID;
```

# JOIN (Join Operator)



- Links rows from two different tables
- Combine information from both

Example: Total number of interactions each post receives.

```
SELECT Posts.PostID,  
       COUNT(Interactions.ReactionType) AS TotalInteractions  
FROM Posts  
JOIN Interactions ON Posts.PostID = Interactions.PostID  
GROUP BY Posts.PostID;
```

# Relational Algebra



# Relational Algebra

Theoretical  
Foundation

Creating  
a Query

Sequence of  
Operators

Query  
Data

# Relational Algebra

Combine Operators

Filter Interactions

Combine Tables

Group & Count  
Results

Project Fields

Sort Popular Posts

```
SELECT Interactions.PostID,  
       COUNT(*) AS Likes,  
       Users.UserID, Users.Username  
FROM Interactions  
JOIN Users ON Interactions.UserID =  
Users.UserID  
WHERE Interactions.ReactionType = 'Like'  
GROUP BY Interactions.PostID,  
         Users.UserID,  
         Users.Username  
ORDER BY Likes DESC;
```

# Relational Algebra

Filters Interactions

Combine Tables

Group & Aggregate

Project Output

Order Posts

```
T Likes DESC
(π PostID, Likes, UserID, Username
 (γ PostID, UserID, Username;
 COUNT(*) → Likes
 (σ ReactionType='Like'
 (Interactions) ⋈ Users)
 )
 )
```

# BuzzDB



# BuzzDB



# Insertion Operator in C++

UserName,	Location
Timothée Chalamet,	Paris
,	;

```
void BuzzDB::insert(int key, int value) {  
    Tuple newTuple = {key, value};  
    table.push_back(newTuple); // Add to main table  
vector  
    index[key].push_back(value); // Also, update the index  
map  
}
```

Key-Based Tuple Retrieval

# Populating the Database

## BuzzDB Insert Method Creation

```
int main() {
    BuzzDB db;
    // Populating the database
    db.insert(1, 100); db.insert(1, 200);
    db.insert(2, 50);
    db.insert(3, 200); db.insert(3, 200); db.insert(3, 100);
    db.insert(4, 500);
    // Executing aggregation query
    db.selectGroupBySum();
    return 0;
}
```

# Aggregation Query

selectGroupBySum  
method

Tally Summarization

Iterate Over Keys

Iterate Over Values

```
void BuzzDB::selectGroupBySum() {  
    // Iterate over each key  
    for (auto const &pair : index) {  
        int sum = 0;  
        // Sum values for this key  
        for (auto const &value : pair.second) {  
            sum += value;  
        }  
        std::cout << "key: " << pair.first << ", sum: " << sum  
<< '\n';  
    }  
}
```

# Aggregation Query

Database  
Initiation

Sum Key  
Values

```
int main() {  
    ...  
    // Executing aggregation query  
    db.selectGroupBySum();  
    return 0;  
}
```

// PROGRAM OUTPUT

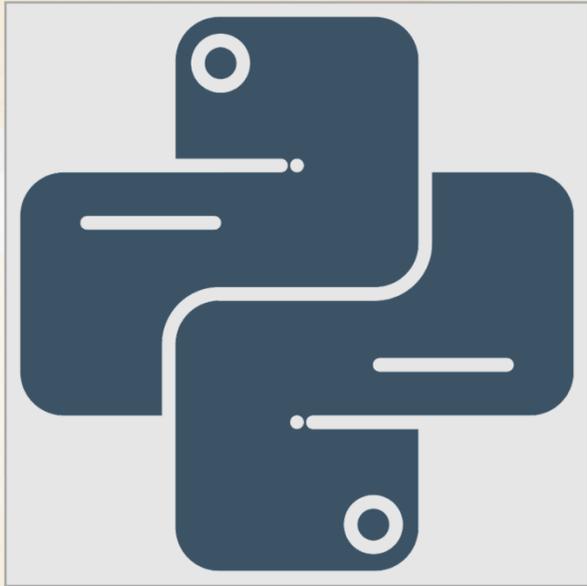
key: 1, sum: 300

key: 2, sum: 50

key: 3, sum: 500

key: 4, sum: 500

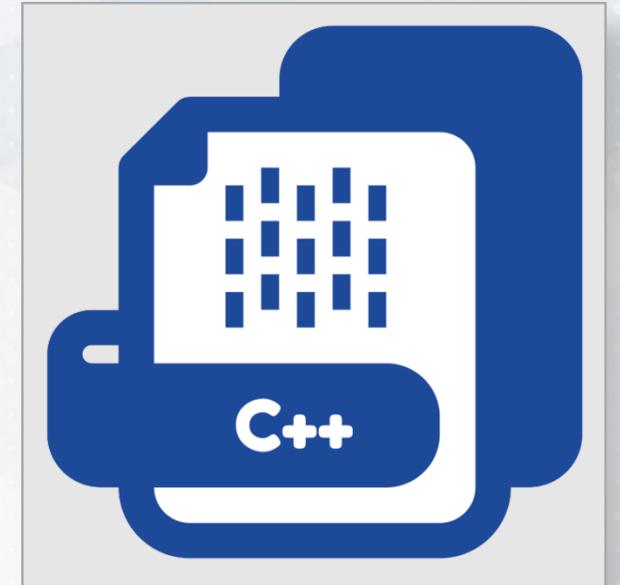
# C++ vs Python



Superior  
Performance

Executable Code

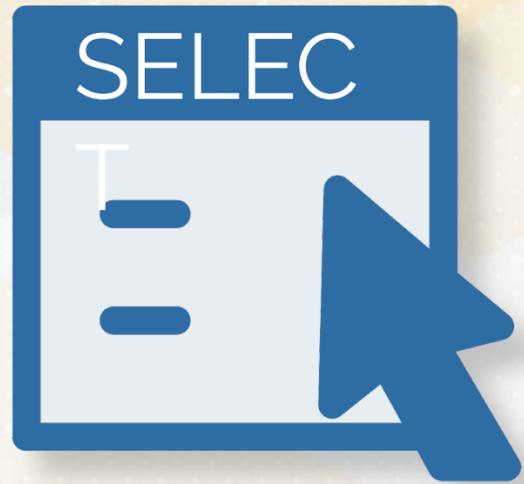
Fast & Efficient



# Transactions

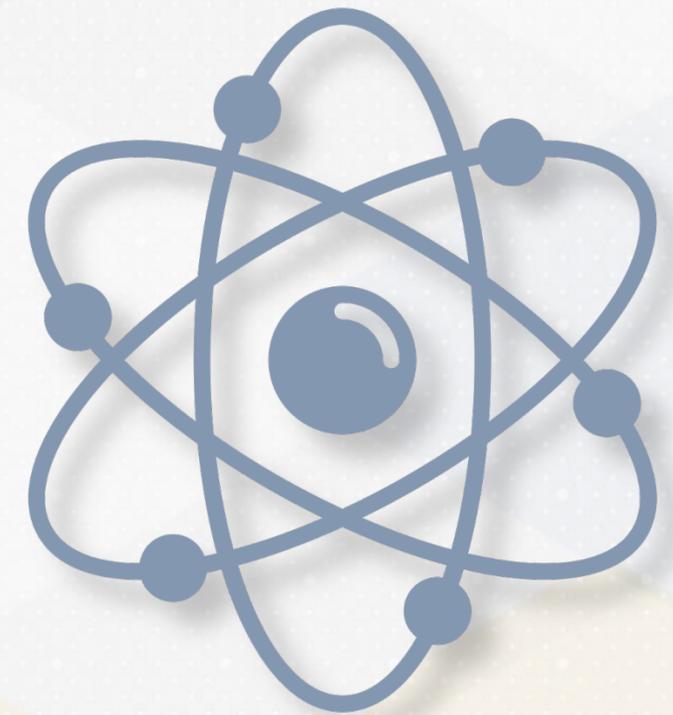


# Queries vs Transactions



✓ Multiple Database Changes

# Transaction: Atomicity Property



# Transaction: Atomicity Property



# Transaction: Consistency Property

\$100 from Account 1 to Account 2

Update Failure → No Partial Updates

```
BEGIN TRANSACTION;  
  
-- Withdraw $100 from account 1  
UPDATE ACCOUNT SET Balance = Balance - 100 WHERE AccountID = 1;  
  
-- Deposit $100 into account 2  
UPDATE ACCOUNT SET Balance = Balance + 100 WHERE AccountID = 2;  
  
COMMIT;
```

# Transaction: Isolation Property

Clerk 1 → \$500	\$1500 vs \$1800
--------------------	---------------------

```
-- Clerk 1: Deposits $500 into account 1
BEGIN TRANSACTION;
SELECT Balance FROM ACCOUNT WHERE AccountID = 1; -- Suppose it returns $1000
UPDATE ACCOUNT SET Balance = 1000 + 500 WHERE AccountID = 1;
COMMIT;
-- Clerk 2: Deposits $300 into account 1 almost at the same time
BEGIN TRANSACTION;
SELECT Balance FROM ACCOUNT WHERE AccountID = 1; -- Suppose it still returns $1000
UPDATE ACCOUNT SET Balance = 1000 + 300 WHERE AccountID = 1;
COMMIT;
```

# Transaction: Durability Property

```
BEGIN TRANSACTION;  
UPDATE ACCOUNT SET Balance = Balance - 500 WHERE AccountID = 1; -- Customer withdraws $500  
COMMIT;
```

Durability

Durable  
Storage

# ACID Properties

**A**

Atomicity

**C**

Consistency

**I**

Isolation

**D**

Durability

# History of “ACID”

Andreas Reuter (1983)

Reliable Management

Multi-User Capability



Photo: Gülay Keskin/Heidelberg Institute for Theoretical Studies (HITS)

# History of “ACID”

Andreas Reuter (1983)

Reliable Management

Multi-User Capability



Photo: Gülay Keskin/Heidelberg Institute for Theoretical Studies (HITS)

# Conclusion

- Illustrative Social Media Analytics
- Limitations of a Flat-file Database System
- Benefits of a Relational Database System
- Relational Algebra
- ACID Properties