Syracuse University
School of Information Studies

Transaction Management and Concurrency Control

Agenda

- What are database transactions?
- What are the ACID properties of a transaction?
- How to write transaction-safe SQL code.
- What is concurrency control, and how is it related to transactions?
- Understand advanced concurrency issues like versioning, locking, and deadlocks.

Syracuse University
School of Information Studies

Transaction Management and Concurrency Control

The End

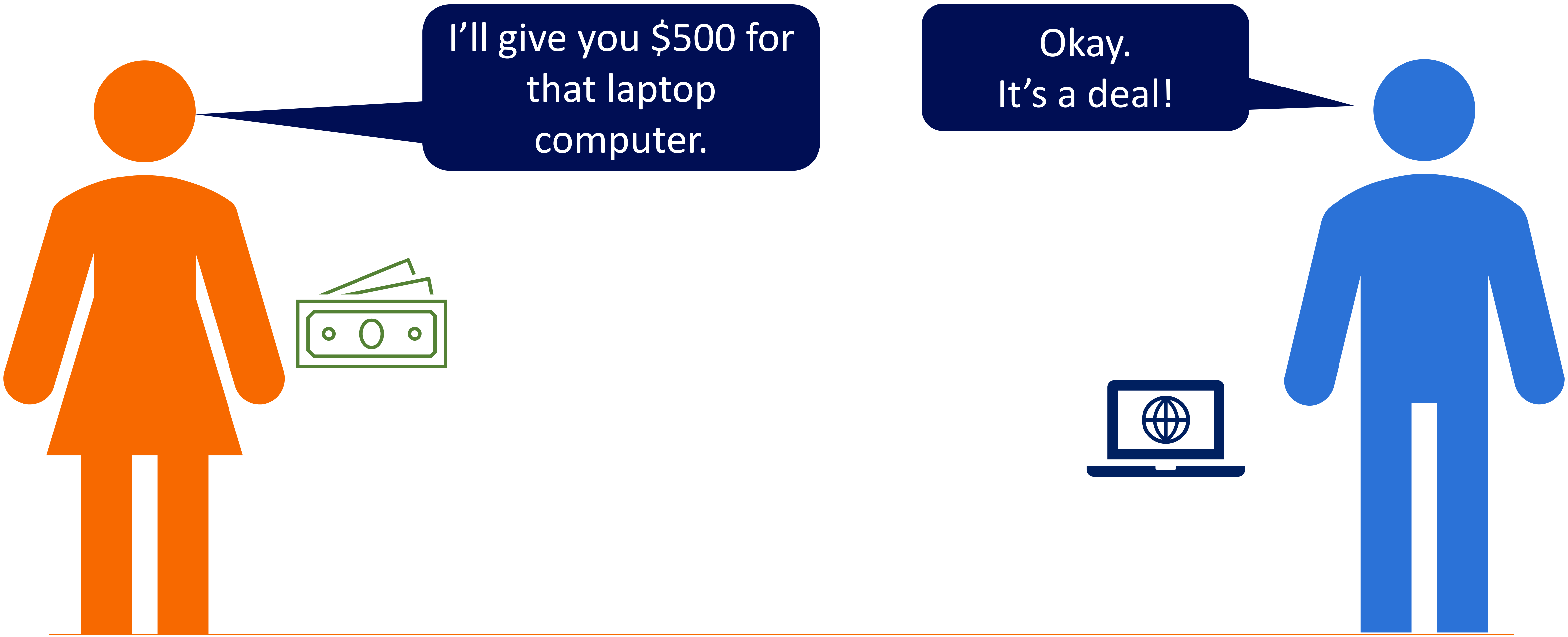
Syracuse University
School of Information Studies

What Are Transactions?

Transaction

- Any logical unit of work in your database management system, read, write, or combination thereof
- It is data logic
- Typically consists of several read/write operations
- Must succeed or fail as a whole
- Operates independently from other units of work
- Doesn’t know about anything other than itself

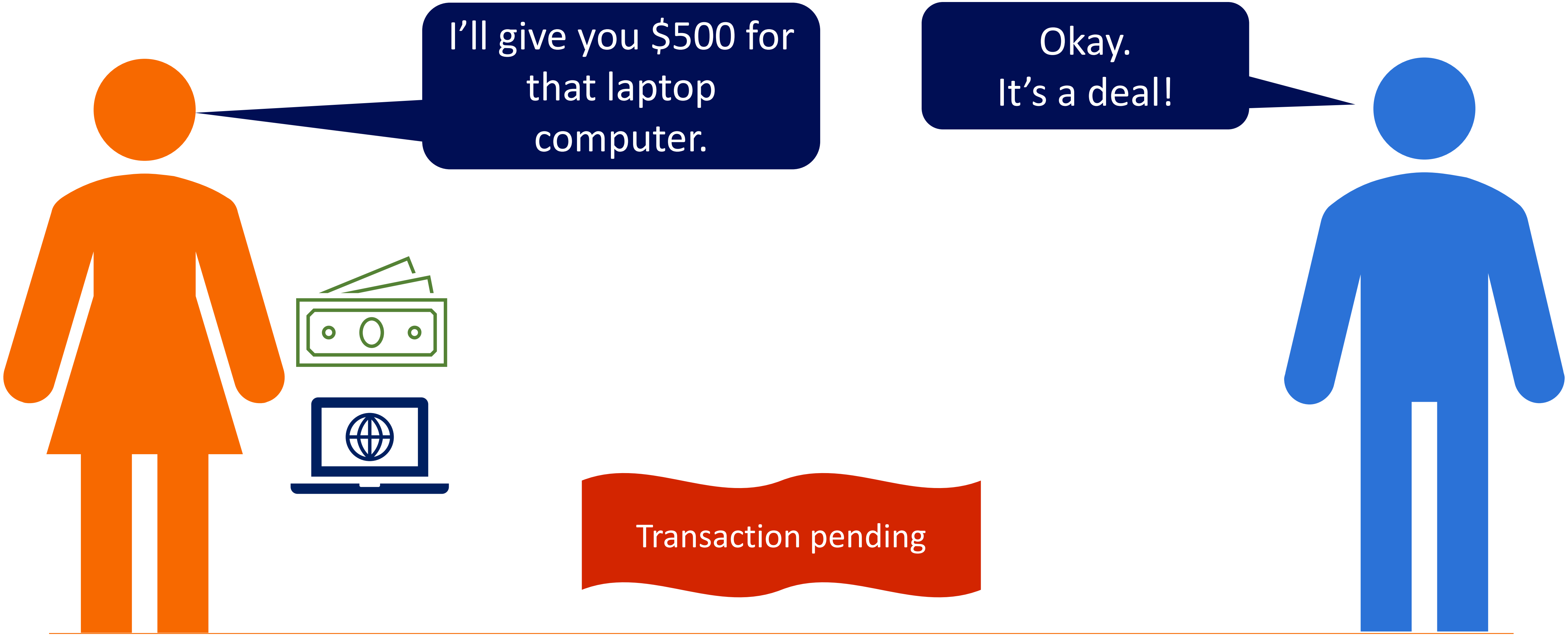
Real-Life Example



I'll give you $500 for that laptop computer.

Okay.
It's a deal!

Real-Life Example

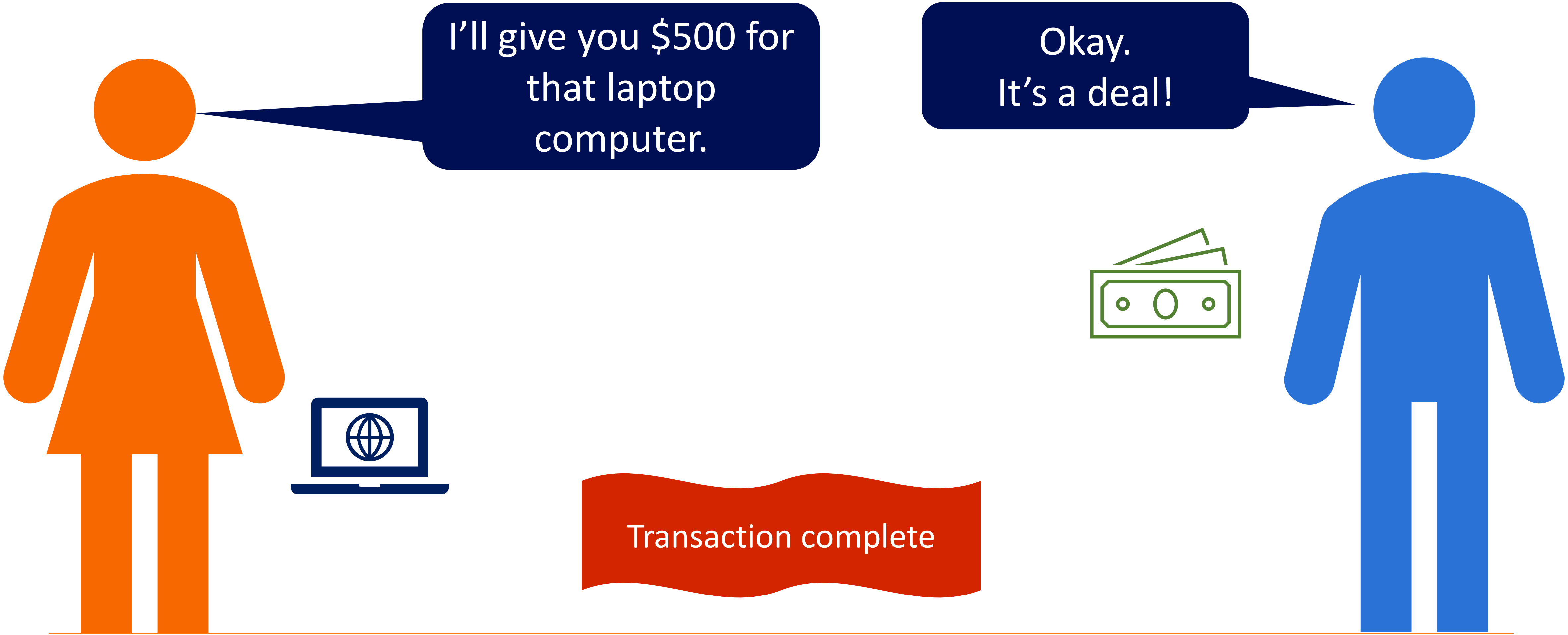


I’ll give you $500 for that laptop computer.

Okay.
It’s a deal!

Transaction pending

Real-Life Example

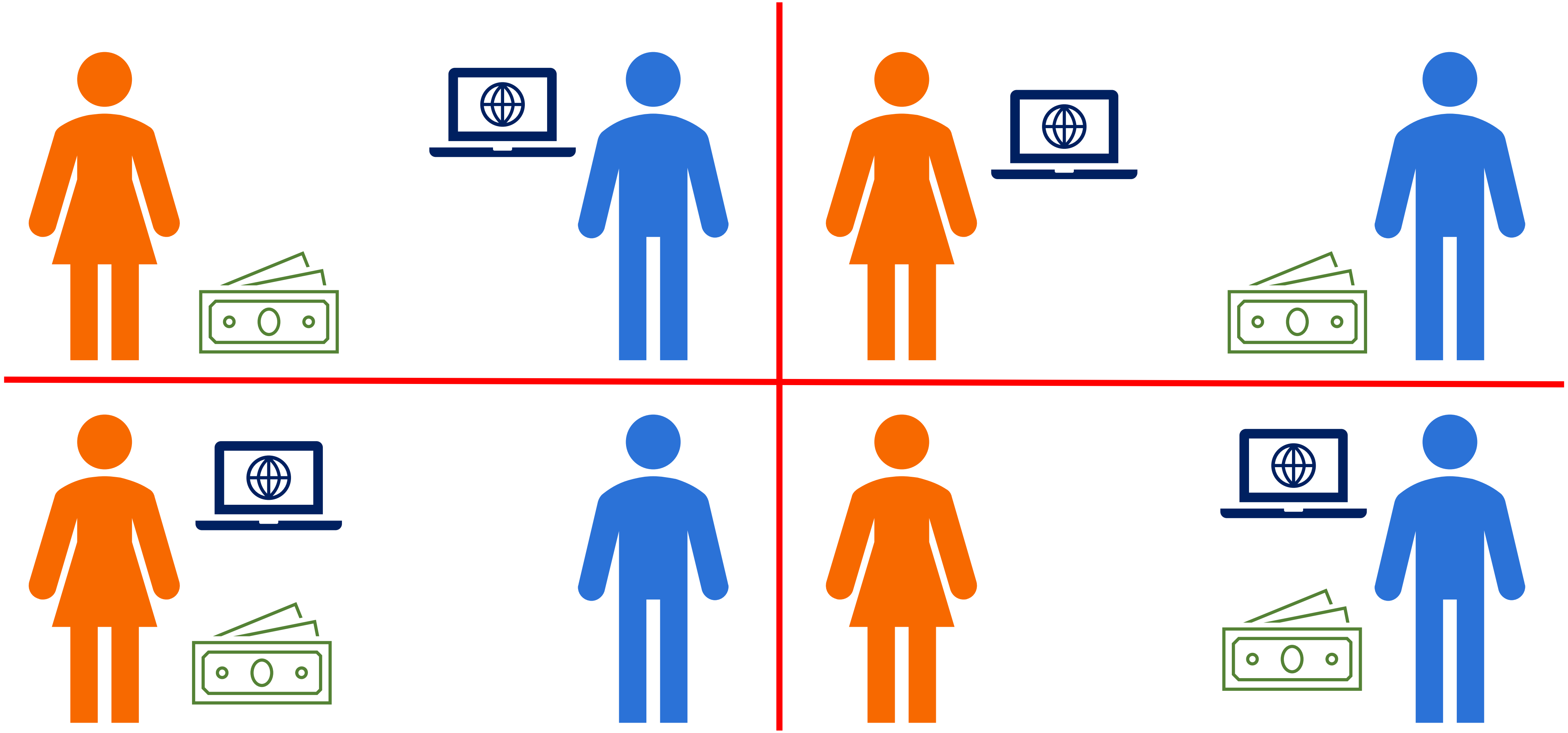


I’ll give you $500 for that laptop computer.

Okay.
It’s a deal!

Transaction complete

Database Transactions
Guarantee No Intermediary States



Syracuse University
School of Information Studies

What Are Transactions?

The End

Syracuse University
School of Information Studies

Demo

The Need for Transactions

Demo: The Need for Transactions

- We will use the Azure Data Studio application
- We will use the demo database
- Accounts table
- Understand the need for transactions and the problem of intermediary states
- This code does not use transactions, but what’s the problem?
    - Check constraint firing
    - No rows affected

Syracuse University
School of Information Studies

Demo: The Need for Transactions

The End

Syracuse University
School of Information Studies

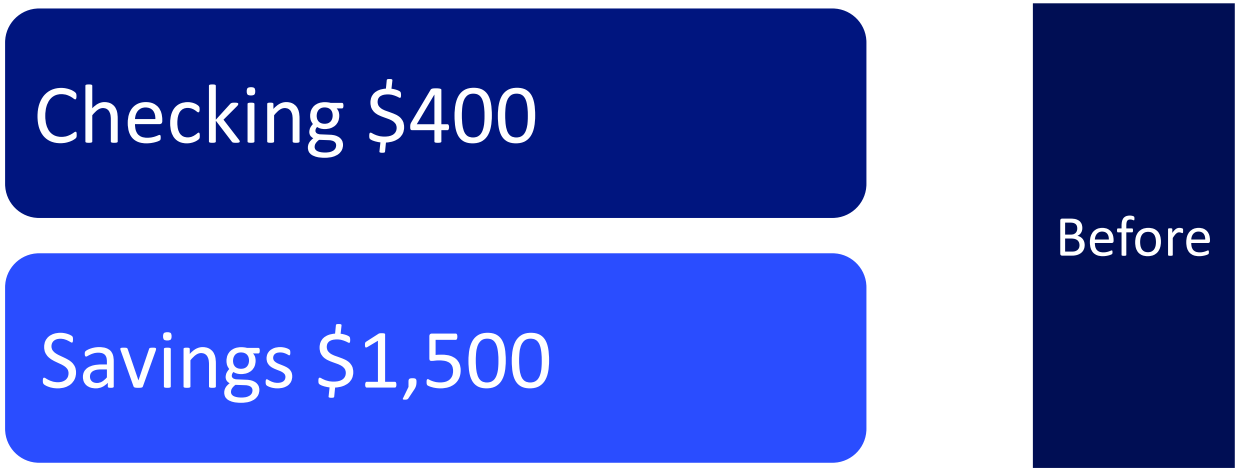
ACID Properties

Transaction ACID Properties

- Atomic
  - Transaction cannot be subdivided; logical unit of work; completes as a single unit or not at all
- Consistent
  - Transitions data from one state to another, constraints intact
- Isolated
  - Database changes not revealed to other users until after transaction has completed
- Durable
  - Database changes are permanent once committed

Example: Transfer Funds

- Transfer $1,000 from savings to checking



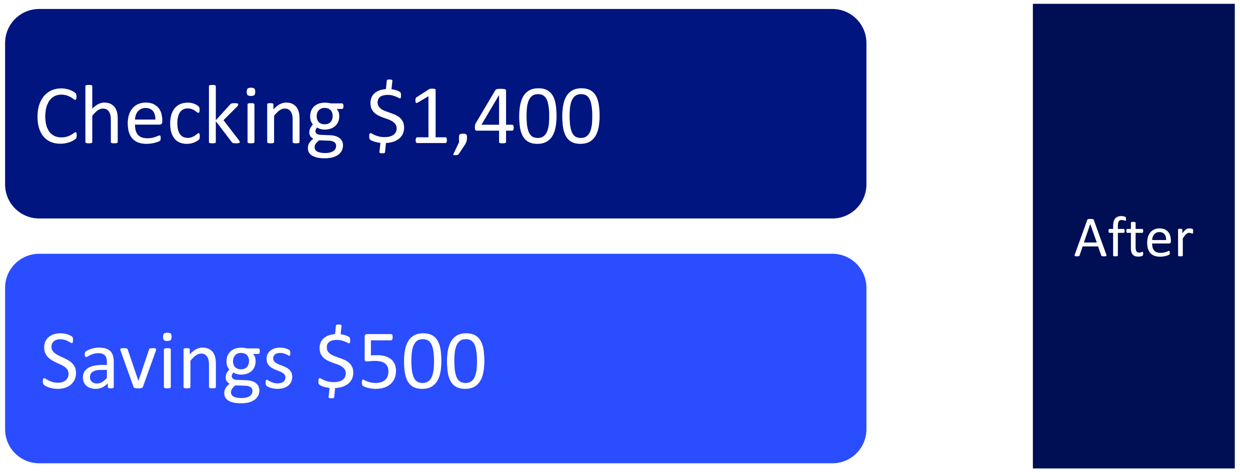
Checking $400

Savings $1,500

Before

Example: Transfer Funds (cont.)

- Transfer $1,000 from savings to checking



Checking $1,400

Savings $500

After

Atomic: Transfer Funds

- Transfer $1,000 from savings to checking
  - Step 1 remove $1,000 from savings
  - Step 2 add $1,000 to checking

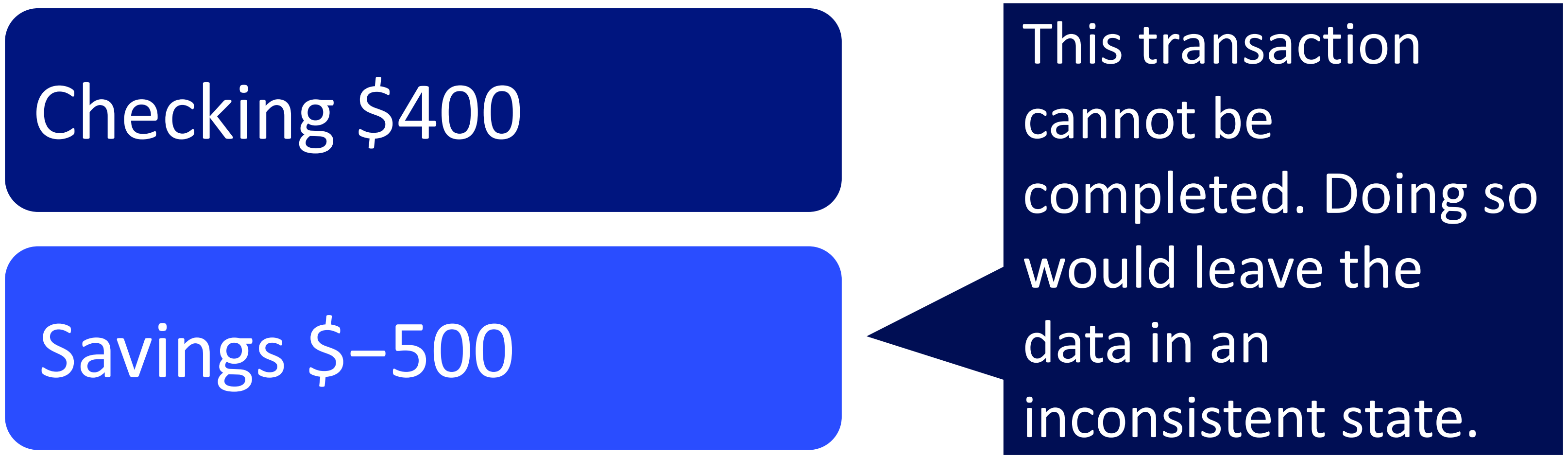
Checking $400

Savings $500

Nobody ever sees this step in the process since the transaction is treated as a single operation.

Consistent: Transfer Funds

- Transfer $2,000 from savings to checking
- Business rule accounts: 0 or more



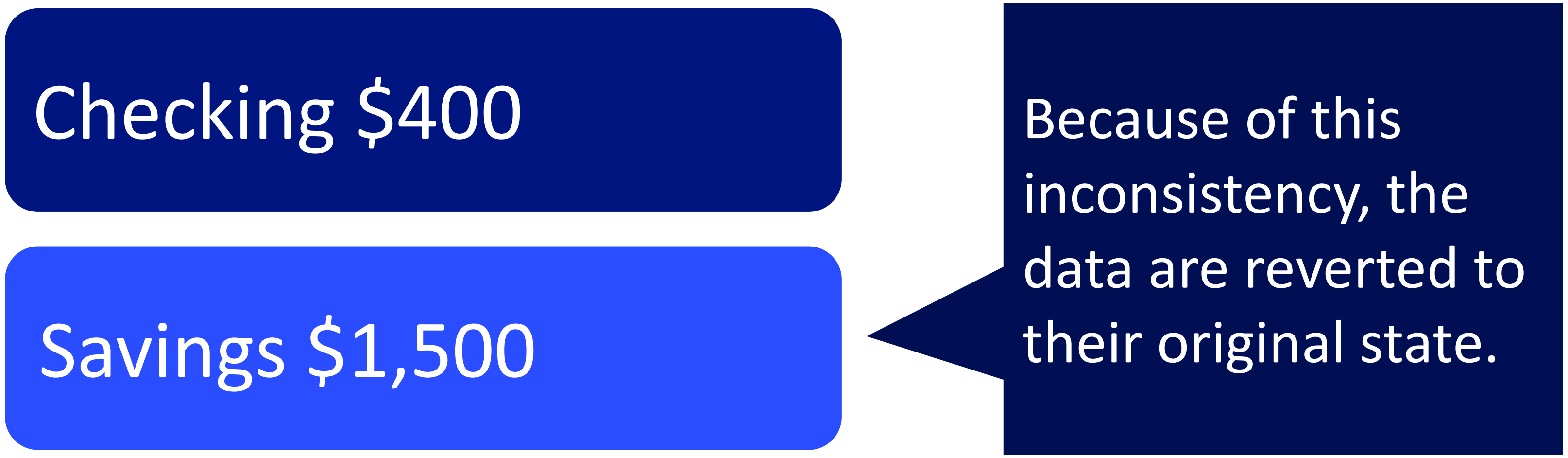
Checking $400

Savings $–500

This transaction cannot be completed. Doing so would leave the data in an inconsistent state.

Consistent: Transfer Funds (cont.)

- Transfer $2,000 from savings to checking
- Business rule accounts: 0 or more



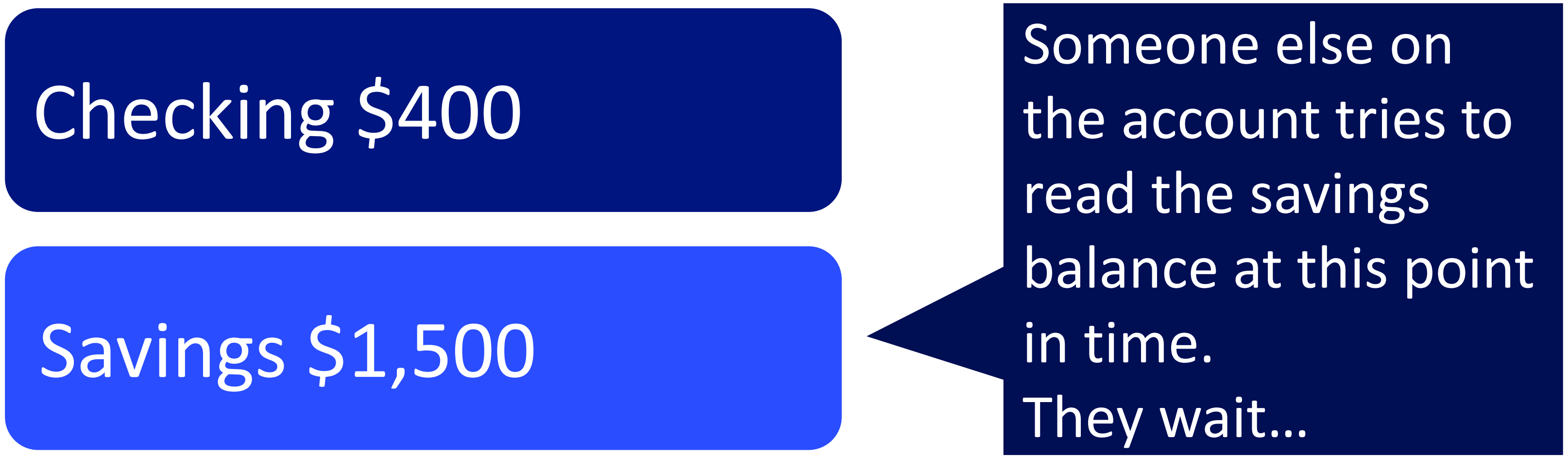
Checking $400

Savings $1,500

Because of this inconsistency, the data are reverted to their original state.

Isolated: Transfer Funds

- Transfer $1,000 from savings to checking



Checking $400

Savings $1,500

Someone else on the account tries to read the savings balance at this point in time.
They wait…

Isolated: Transfer Funds (cont.)

- Transfer $1,000 from savings to checking

Checking $1,400

Savings $500

… until the transaction is completed, then they see the savings balance of $500.

Durable: Transfer Funds

- Transfer $1,000 from savings to checking

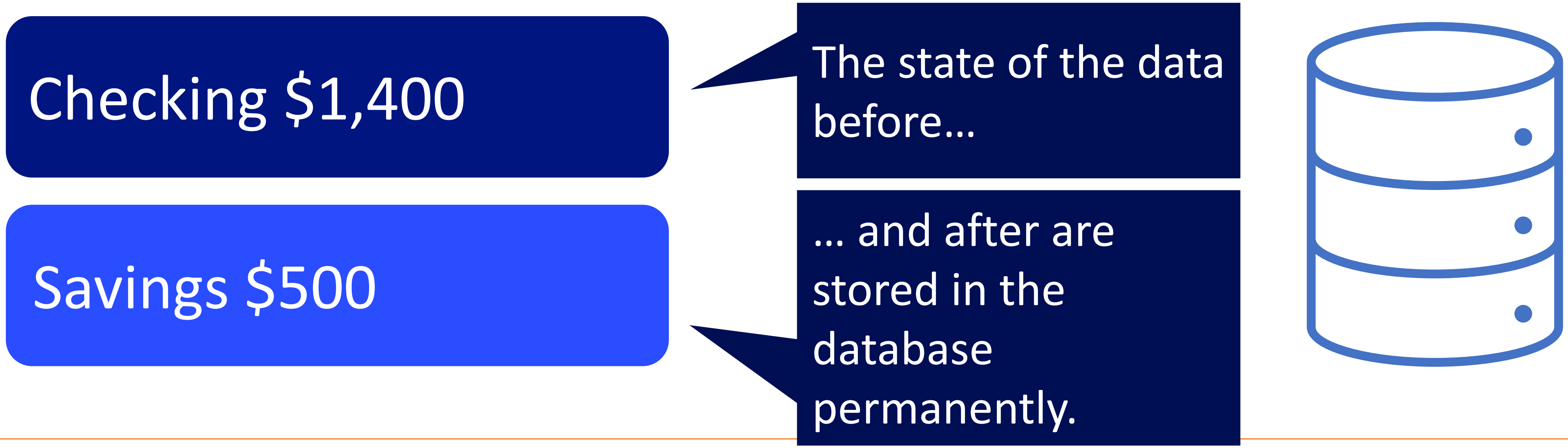
Checking $400

Savings $1,500

The state of the data before…

Durable: Transfer Funds (cont.)

- Transfer $1,000 from savings to checking



Checking $1,400

Savings $500

The state of the data before…

… and after are stored in the database permanently.

Syracuse University
School of Information Studies

ACID Properties

The End

Syracuse University
School of Information Studies

Transaction Support in SQL Server

Transactions in SQL Server

- We simply take the code we have:

```
BEGIN TRANSACTION
UPDATE accounts SET checking …
UPDATE accounts SET checking …
COMMIT TRANSACTION
```

- And tell SQL Server to group them as a transaction—sort of

T-SQL Transaction Commands

- `BEGIN TRAN[SACTION]`
  - Starts the marking point of a transaction; any data manipulation statements after this point are not durable
- `COMMIT [TRAN[SACTION]]`
  - Marks the end of a successful transaction; at this point, all data changes to become durable
- `ROLLBACK [TRAN[SACTION]]`
  - Undoes any data manipulation statements since the beginning of the transaction
- @@TRANCOUNT
  - A T-SQL Internal variable that identifies pending transactions

Syracuse University
School of Information Studies

Transaction Support in SQL Server

The End

Syracuse University
School of Information Studies

Demo

SQL Transactions

Demo: SQL Transations

- We will use the Azure Data Studio application
- We will use the demo database
- Without BEGIN TRANSACTION, all commands are durable
- With BEGIN TRANSACTION, they are not!
- @@TRANCOUNT
- COMMIT and ROLLBACK
- Transactions are isolated, consistent, atomic, and durable

Syracuse University
School of Information Studies

Demo: SQL Transactions

The End

Syracuse University
School of Information Studies

Transaction Safe Code

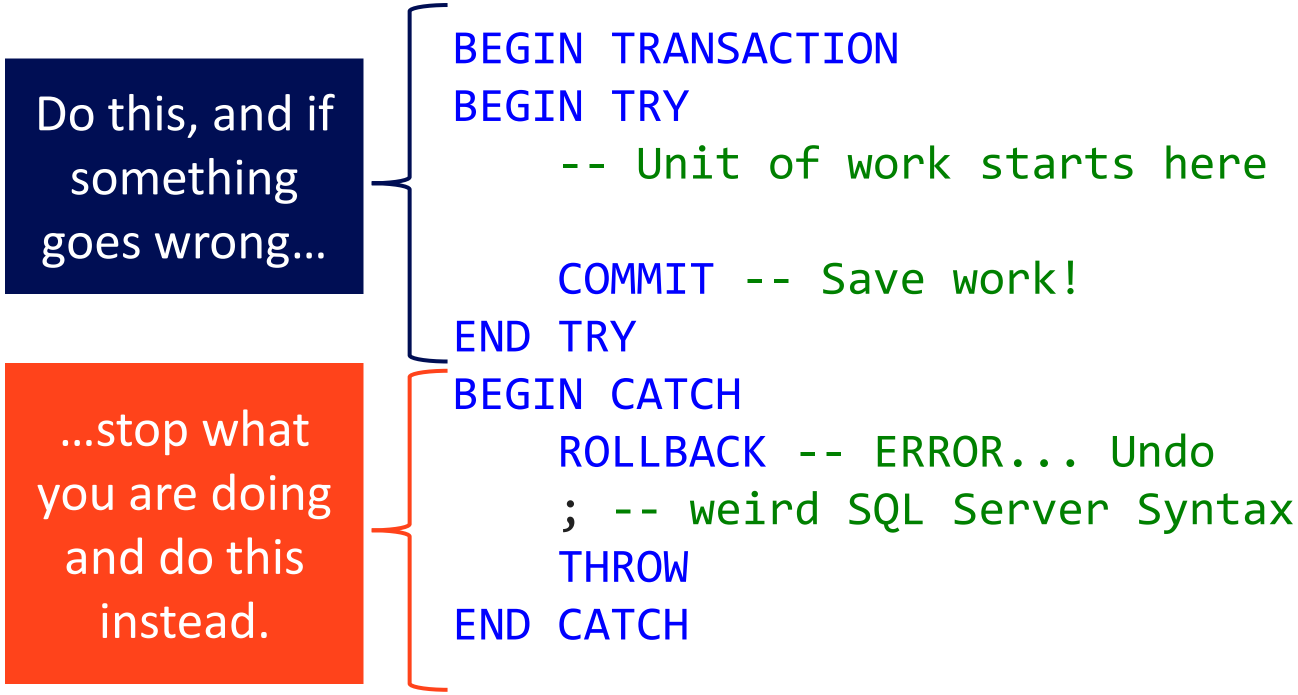
Transaction Safe Code 101: How Do You Know When to Rollback?

We want to rollback when:

- An error occurs
  - Database cannot write to the table (disk is full)
  - There is a physical hardware/operating system issue
  - A database constraint is violated (check, PK, FK, etc.), which will result in an inconsistency
- There is custom data logic
  - The expected number of rows are not affected, for example

SQL Error Handling: TRY… CATCH

The SQL TRY… CATCH statement makes transaction management of errors simple.



Do this, and if something goes wrong…

```
BEGIN TRANSACTION
BEGIN TRY
    -- Unit of work starts here

    COMMIT -- Save work!
END TRY
BEGIN CATCH
    ROLLBACK -- ERROR... Undo
    ; -- weird SQL Server Syntax
    THROW
END CATCH
```

…stop what you are doing and do this instead.

Syracuse University
School of Information Studies

Transaction Safe Code

The End

Syracuse University
School of Information Studies

Demo

Transactions TRY/CATCH

Demo: Transactions TRY/CATCH

- We will use the Azure Data Studio application.
- We will use the demo database.
- Re-write our p_transfer_funds to use transactions.
- Test it out:.
  - Check constraint firing
  - No rows affected
- Not perfect, but better!

Syracuse University
School of Information Studies

Demo: Transactions TRY/CATCH

The End

Syracuse University
School of Information Studies

Rollback on Custom Data Logic

Rollback From Custom Data Logic

- We might want to rollback based on custom data logic
- For example, when you update a row but no rows are affected, how do you handle this?

@@ROWCOUNT—reports number of rows affected from an SQL statement

```
SELECT * FROM accounts
(2 rows affected)
PRINT @@ROWCOUNT
2
```

SQL THROW a Custom Error

Use the SQL THROW statement to launch a custom user-defined error

```
THROW error_number, message, state
```

- `error_number` — a customer number >= 50000
- `Message` — a string error message
- `State` — a number 0–255 for message state, usually a 1

Syracuse University
School of Information Studies

Rollback on Custom Data Logic

The End

Syracuse University
School of Information Studies

Demo

Rollback On Custom Data Logic

Demo: Rollback on Custom Data Logic

- We will use the Azure Data Studio application.
- We will use the demo database.
- Re-write our p_transfer_funds to use transactions.
- Handle custom business logic for rows affected.

Syracuse University
School of Information Studies

Demo: Rollback Custom Data Logic

The End

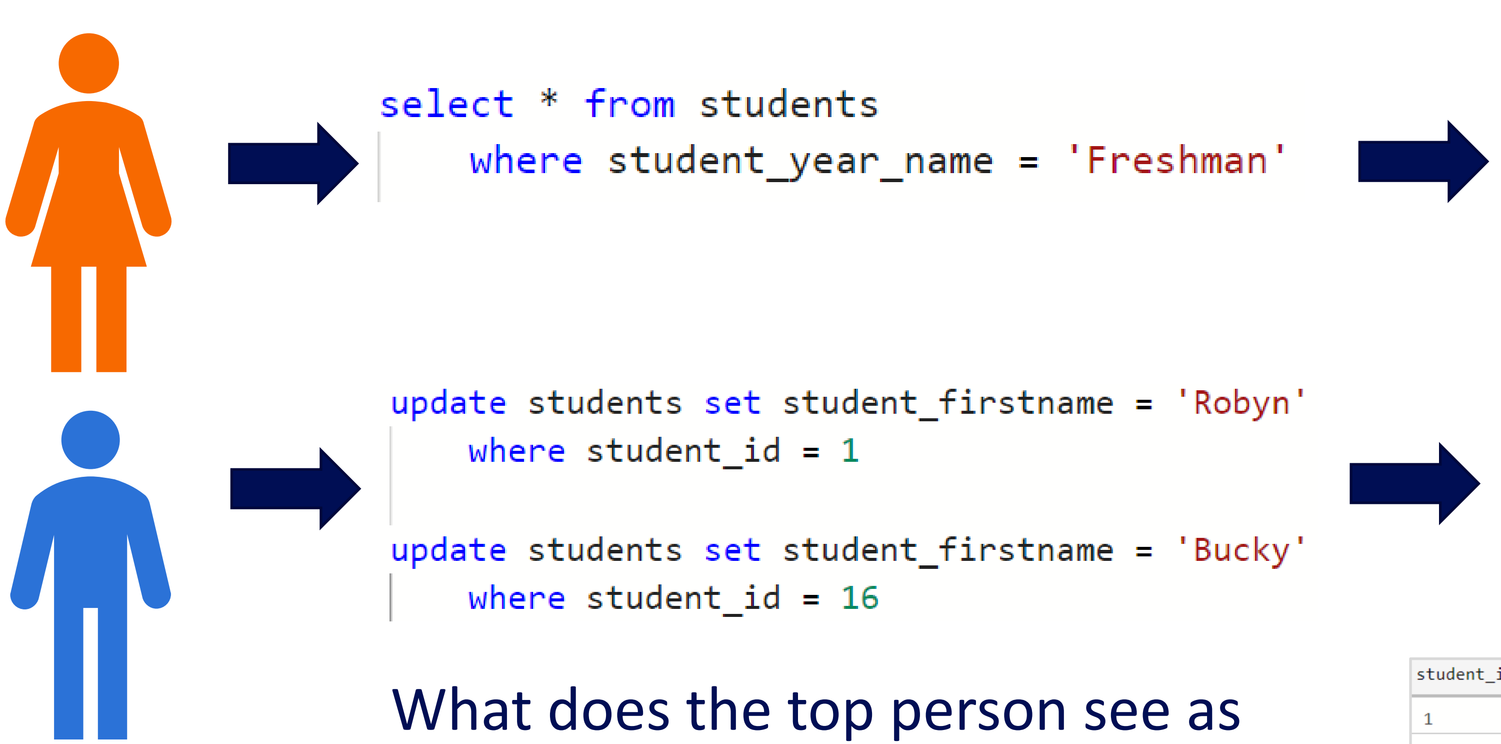
Syracuse University
School of Information Studies

Concurrency Control

Concurrency Control

- Problem: in a multiuser environment, simultaneous access to data can result in interference and data loss
- Solution: concurrency control
  - The process of managing simultaneous transactions against a database so that data integrity is maintained, and the operations do not interfere with each other in a multi-user environment.
  - Concurrency control helps keep transactions isolated.
  - Concurrency control is a part of every modern DBMS.

Example: Concurrency Question



```
select * from students
    where student_year_name = 'Freshman'
```

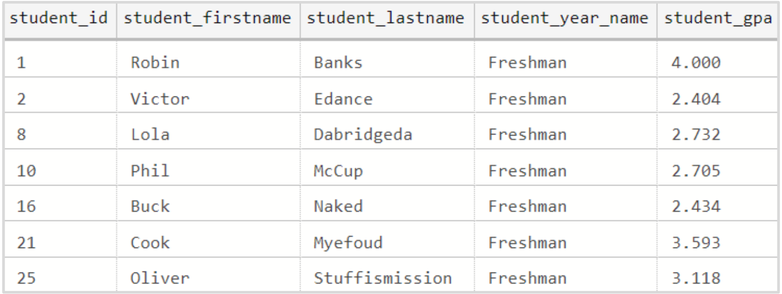
```
update students set student_firstname = 'Robyn'
    where student_id = 1

update students set student_firstname = 'Bucky'
    where student_id = 16
```

What does the top person see as output?

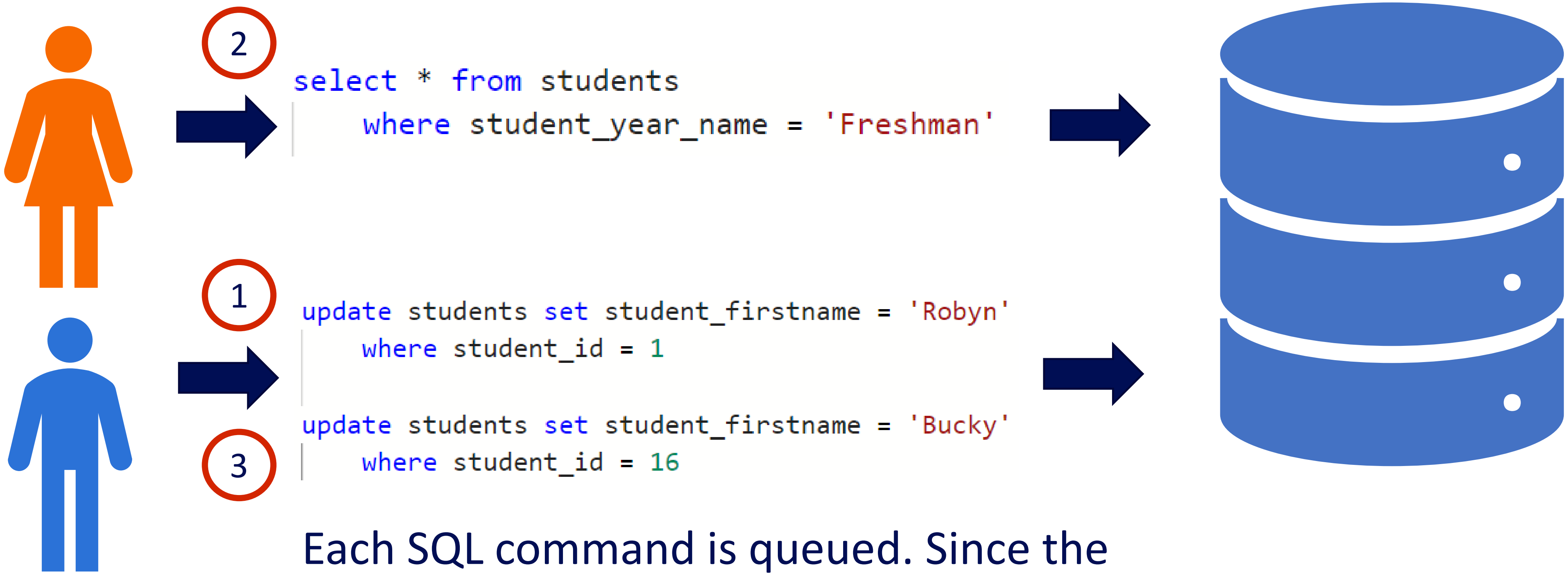
Robin Robyn Robyn

Buck Buck Bucky



student_id	student_firstname	student_lastname	student_year_name	student_gpa
1	Robin	Banks	Freshman	4.000
2	Victor	Edance	Freshman	2.404
8	Lola	Dabridgeda	Freshman	2.732
10	Phil	McCup	Freshman	2.705
16	Buck	Naked	Freshman	2.434
21	Cook	Myefoud	Freshman	3.593
25	Oliver	Stuffismission	Freshman	3.118

Example: Concurrency Answer



```sql
select * from students
    where student_year_name = 'Freshman'
```

```sql
update students set student_firstname = 'Robyn'
    where student_id = 1
```

```sql
update students set student_firstname = 'Bucky'
    where student_id = 16
```

Each SQL command is queued. Since the bottom person did not use a transaction, they could get ordered:

Robyn
Bucky

Syracuse University
School of Information Studies

Concurrency Control

The End

Syracuse University
School of Information Studies

DBMS Without Concurrency Control

Lost Update Problem

Occurs when one update overwrites another

Transaction A	Time	Transaction B
READ Balance = $500	1	---
---	2	READ Balance = $500
---	3	Withdrawal $300 WRITE Balance = $200
Withdrawal $100 WRITE Balance = $400	4	---
	5	READ Balance = $400?!

Balance should be $100, but it’s $400!

Dirty/Inconsistent Read Problem

Occurs when a transaction reads uncommitted data

Transaction A	Time	Transaction B
---	1	READ City = ‘Utica’
---	2	UPDATE City = ‘Rome’
READ City = ‘Rome’	3	---
---	4	FAIL: Rollback

A’s city = ‘Rome’ and B’s city = ‘Utica’

Syracuse University
School of Information Studies

DBMS Without Concurrency Control

The End

Syracuse University
School of Information Studies

Serializability and Locking

How Do DBMS Maintain Concurrency Control?

- Serializability
  - Finish one transaction before starting another
- Locking mechanisms
  - The most common way of achieving serialization
  - Data that are retrieved for the purpose of updating are locked for the updater
  - No other user can perform a write operation until unlocked

Serializability and Locking

This demonstrates an exclusive lock.

Transaction A	Time	Transaction B
Request account, lock acquired	1	---
READ Balance = $500	2	Request account, waiting for lock release
---	3	---
Withdrawal $100 WRITE Balance = $400	4	---
Lock released	5	---
---	6	Lock acquired
---	7	READ Balance = $400

Locking Mechanisms

- Locking granularity
  - Database: used during database updates, ALTER DATABASE
  - Table: used for bulk updates, or ALTER TABLE
  - Block or page: very commonly used
  - Record: only requested row; fairly commonly used
  - Field: requires significant overhead; impractical
- Types of locks
  - Shared lock: read but no update permitted; used when just reading to prevent another user from placing an exclusive lock on the record
  - Exclusive lock: no access permitted; used when preparing to update

Syracuse University
School of Information Studies

Serializability and Locking

The End

Syracuse University
School of Information Studies

Versioning and Isolation Levels

Versioning

- DBMS maintains multiple versions of the data to be modified as part of the transaction.
- Concurrent reads are allowed to the data.
- Attempts to write are allowed, but subsequent writes are rolled back and restarted.

Versioning Example

Transaction A	Time	Transaction B
READ Balance = $500	1	---
---	2	READ Balance = $500
Withdrawal $100 Begin transaction WRITE Balance = $400	3	---
---	4	Withdrawal $300 Begin transaction WRITE Balance = $200
Commit	5	Other version pending, rollback
---	6	Restart transaction

Isolation Levels in SQL Server

- We change concurrency control by setting an isolation level in the database
- Read uncommitted: any transaction can read any uncommitted data; no locks, no concurrency control.
- Read committed: no transaction can read uncommitted data; default
- Serializable: like read committed but also locks related data across foreign keys
- Read committed snapshot: uses versioning so that there are no locks on the read operation

Syracuse University
School of Information Studies

Versioning and Isolation Levels

The End

Syracuse University
School of Information Studies

Demo

Concurrency Control

Demo: Concurrency Control

- We will use the Azure Data Studio application.
- We will use the demo database.
- Let’s demonstrate a dirty read by playing with the isolation levels.
  - Update a row
  - Witness a lock on the table read committed
  - No lock with read uncommitted—but you read pending transaction data
  - When transaction is rolled back—yikes!

Syracuse University
School of Information Studies

Demo: Concurrency Control

The End

Syracuse University
School of Information Studies

Deadlocks

Deadlocks

- A deadlock situation occurs when two or more transactions are waiting for each other to give up locks
- Row level locks help mitigate this issue, but there is still a possibility
- Set the lock timeout in milliseconds
- **`SET LOCK_TIMEOUT 5000–five seconds`**
- Read it with **`@@LOCK_TIMEOUT`**
- Default is –1 (never)

Deadlock Example

Transaction A	Time	Transaction B
Begin transaction	1	---
---	2	Begin transaction
Update users where id = 1	3	---
---	4	Update blogs where id = 7
Update blogs where id = 7 (Locked by B, waiting…)	5	---
---	6	Update users where id = 1 (Locked by A, waiting…)
---	7	---

Syracuse University
School of Information Studies

Deadlocks

The End

Syracuse University
School of Information Studies

Demo

Deadlock

Demo: Deadlock

- We will use the Azure Data Studio application.
- We will use the tinyu database.
- Let’s demonstrate a deadlock scenario.
    - Transaction 1: Update row in table A, then row in table B.
    - Transaction 1: Update row in table B, then row in table A.

Syracuse University
School of Information Studies

Demo: Deadlock

The End

Syracuse University
School of Information Studies

Summary

Summary

- Transactions are multiple SQL statements treated as a single unit of work.
- Transactions are atomic, consistent, isolated, and durable.
- To make SQL transaction safe, you must rollback on error, or when the expected data logic results do not match the actual results.
- Concurrency control is how multiple transactions are managed within the DBMS.
- Deadlocks occur when two or more transactions are waiting to give up locks.

Syracuse University
School of Information Studies

Summary

The End