Overview of the Relational data model

1

What you will learn about in this section

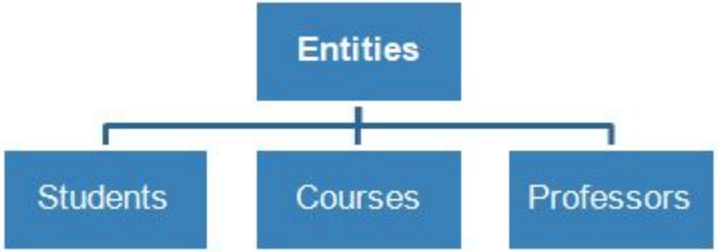
1. Data models & the relational data model
2. Schemas & data independence

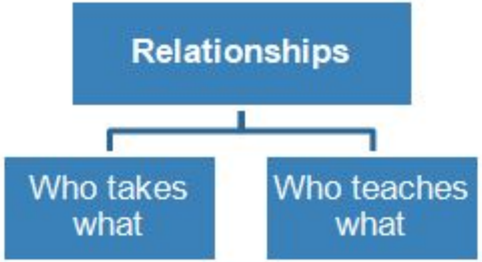
A Motivating, Running Example

Consider building a course management system (CMS):

Entities (e.g., Students, Courses)

Relationships (e.g., Alice is enrolled in 145)





Students

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

100% $ % .0 .00 123 Arial 10 B I S A

	A	B	C	D	E	F	G
5			**Student**	**SID**	**Address**		
6			Mickey	40001	43 Toontown		
7			Daffy	40002	147 Main St		
8			Donald	50003	312 Escondido		
9			Minnie	50004	451 Gates		
10			Pluto	10008	97 Packard		
11							
12							
13							
14			**Course**	**Description**	**Room**	Class size	
15			cs145	Toon systems	Nvidia	300	
16			cs161	Animation art	Gates 300	145	
17			cs245	Painting town rec	Packard 45	27	

‘Modeling’ the CMS

Logical Schema

Students(sid: *string*, name: *string*, gpa: *float*)

Courses(cid: *string*, cname: *string*, credits: *int*)

Enrolled(sid: *string,* cid*: string,* grade*: string*)

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Corresponding *keys*

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Key concept

Data model

Relational model (aka tables)

Simple and most popular

Elegant algebra (E.F. Codd et al)

Data model:

Organizing principle of data + operations

Every relation has a schema

Logical Schema: describes types, names

Physical Schema: describes data layout

Virtual Schema (Views): derived tables

Schema:

Describes blueprint of table (s)

Key concept

Data independence

Data independence

Logical Data Independence
Protection from changes in the Logical Structure of the data

ie. Should not need to ask : Can we add a new entity or attribute without rewriting the application

Physical Data Independence
Protection from Physical Layout Changes

ie. Should not need to ask : Which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS

Overview of

Modify/Writes in DBMS

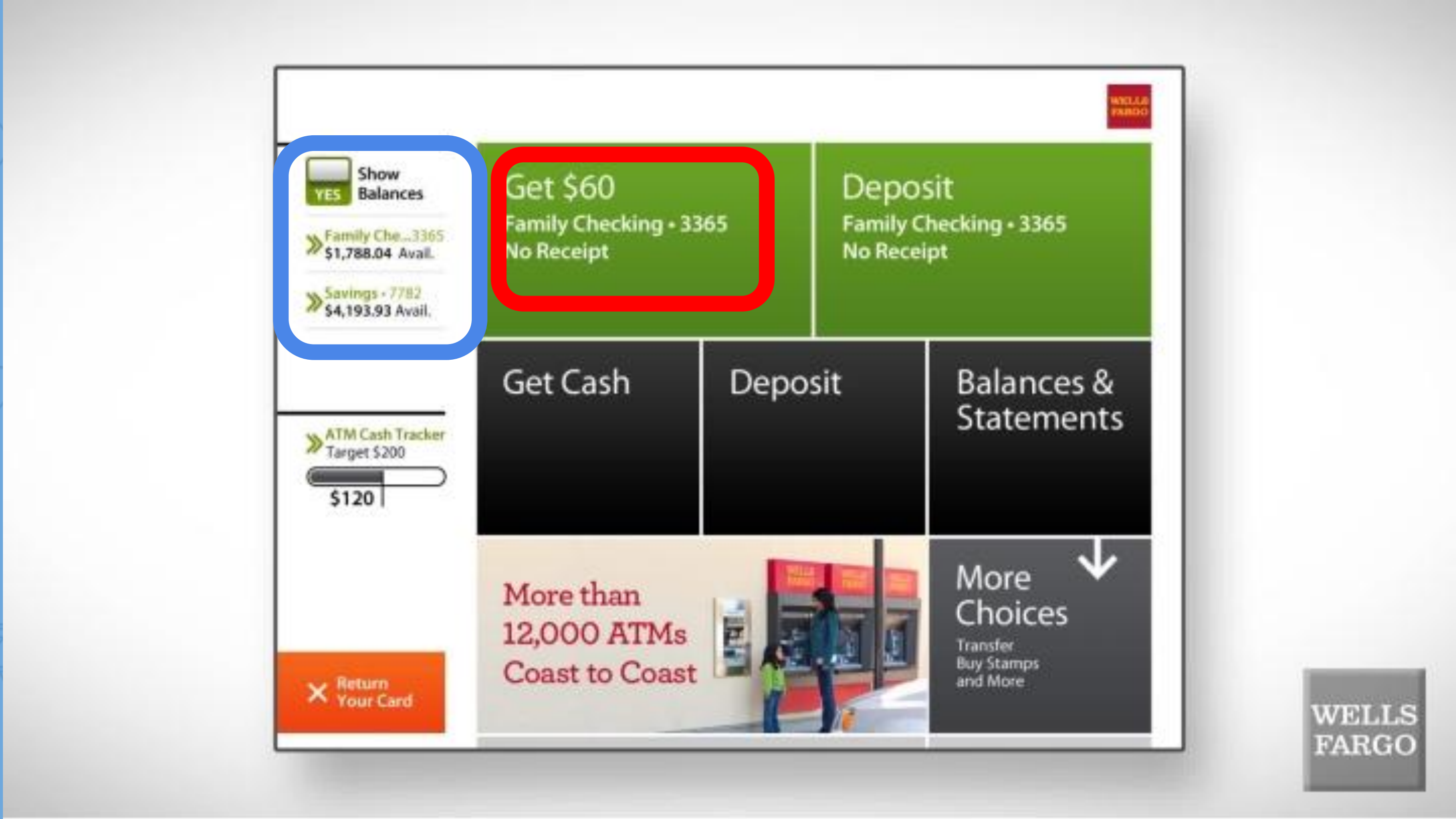
What you will learn about in this section

1. Transactions
2. Data Concurrency & locking
3. Atomicity & logging
4. Summary

Recall

ATM DB:

Transaction



Read Balance Give money Update Balance	vs	Read Balance Update Balance Give money

Transactions

Example

Transfer $3k from a10 to a20:

1 Debit $3k from a10
2 Credit $3k to a20

Crash before 1,
After 1 but before 2,
After 2.

Acct	Balance
a10	20,000
a20	15,000

Acct	Balance
a10	17,000
a20	18,000

Challenges with Many Users

Suppose that our application serves millions of users or more- what are some challenges?

Security

Different users, different roles

Performance

Need to provide concurrent access

Consistency

Concurrency can lead to update problems

Disk/SSD access is slow, DBMS hide the latency by doing more CPU work concurrently

DBMS allows user to write programs as if they were the only user

Key concept

Transactions

An atomic sequence of db actions (reads/writes)

Leaves DB in a **consistent** state

(e.g., bank $$ sum stays same, two courses cannot be in one room, airline seats can't be doublebooked)

Atomicity: An action either completes entirely or not at all

Consistency: An action results in a state which conforms to all integrity constraints

Integrity constraints Data properties to assert, for application's needs

Challenge: Scheduling Concurrent Transactions

- The DBMS ensures that the execution of $\{T_1,\ldots,T_n\}$ is equivalent to some **serial** execution

- One way to accomplish this: **Locking**
  - Before reading or writing, transaction requires a lock from DBMS, holds until the end

A set of TXNs is isolated if their effect is as if all were executed serially

Challenge: Scheduling Concurrent Transactions

Key Idea: If Ti wants to write to an item x and Tj wants to read x, then Ti, Tj conflict. Solution via locking:

- only one winner gets the lock
- loser is blocked (waits) until winner finishes

What if Ti and Tj need X and Y, and Ti asks for X before Tj, and Tj asks for Y before Ti?

-> Deadlock! One is aborted…

All concurrency issues handled by the DBMS…

Ensuring Atomicity & Durability

- DBMS ensures **atomicity** even if a TXN crashes!
- One way to accomplish this: **Write-ahead logging (WAL)**
- **Key Idea**: Keep a log of all the writes done.
  - After a crash, the partially executed TXNs are undone using the log

Write-ahead Logging (WAL): Before any action is finalized, a corresponding log entry is forced to disk

We assume that the log is on "stable" storage

All atomicity issues also handled by the DBMS...

A Well-Designed DBMS makes many people happy!

End Users and DBMS Vendors	DB Application Programmers	DataBase Administrators
Reduces Cost and Makes Money	Can handle more users, faster, for cheaper, and with better reliability / Security Guarantees	Easier time of Designing Logical / Physical Schema,Handling Security / Authorization, tuning, Crash recovery & more…...

Summary of DBMS

DBMS are used to maintain, query, and manage large datasets. Provide concurrency, recovery from crashes, quick application development, integrity, and security

Key abstractions give **data independence**

DBMS R&D is one of the broadest, most exciting fields in CS. **Fact!**

THANK YOU!