

# Overview of the Class and Introduction to DB schemas and queries

Lois Delcambre

# CS 386/586 Introduction to Databases

**Instructor:** Lois Delcambre [imd@cs.pdx.edu](mailto:imd@cs.pdx.edu) 503 725-2405

**TA:** TBA

**Office Hours:** Immediately after class (in/near the classroom)  
Mondays in my office from 4 to 5:30PM

This is a first class in relational database management systems (using SQL and relational algebra). If you've already had a class in relational database systems, you should talk to me; you probably don't need to take this class.

**Prerequisites:** CS 161 Intro to CS 1 (programming) and CS 250 Discrete Structures 1 (discrete math).

If you are a post-bac student, you could enroll as an undergraduate (in 386) or graduate (in 586). If you plan to work on an MS CS degree, you should consider taking 586.

# Why is This Course in the Curriculum?

- It teaches valued job skills
- It integrates CS concepts
  - languages, data structures, concurrency
- The (digital) world runs on data
- It provides an example of the practical power (query optimizers) of an underlying theory (relational algebra)
- It is one of the few (only?) topics that focuses on information that is stored persistently, e.g., on disk (not in main memory).

# Class web page

Syllabus available at:

[www.cs.pdx.edu/~lmd/cs386](http://www.cs.pdx.edu/~lmd/cs386)

Contains complete class schedule including reading assignments, HW assignments, suggested answers for completed assignments, handouts for lectures, and so forth.

New information appears frequently, so **reload** the page ....  
Handouts of slides will be posted on the web page sometime before class – hopefully a day ahead. (One slide set per week)

General structure of the class and the grading is set but the details may be modified, if necessary.

# Overview of the Syllabus

- **Eight Assignments, assigned on Tuesdays, due 1 week later – on Tuesday (40% for 386 students, 32% for 586):**  
Each assignment is worth 5% of your grade (for 386 students) or 4% of your grade (for 586 students). Work by yourself or with a partner. Must be your own work or the work of you and your partner.
- **Project (8%) for 586 (graduate) students** TBA
- **First Exam (20%) OPEN BOOK:** In class. Work by yourself. Ask questions only of the instructor or exam monitor.
- **Second Exam (20%) OPEN BOOK:** In class. Work by yourself. Ask questions only of the instructor or exam monitor.
- **Third Exam (20%) OPEN BOOK:** During finals week. Work by yourself. Ask questions only of the instructor or exam monitor.

## Course Text

***Database Management Systems, 3rd Edition.*** By Raghu Ramakrishnan and Johannes Gehrke, McGraw Hill, 2000, ISBN 0-07-246563-8.

- Should be able to get used copies
- Make sure it's the correct edition

You may find it useful to have a SQL reference, e.g., ***SQL:1999*** by Melton – or other web-based resources.

You may find it useful to do the reading after the topics are covered in class.

# Academic Integrity

You are responsible for knowing the PSU Academic Integrity Policy.

I feel that it is very important to enforce the academic integrity policy; please make sure you know what the rules are and follow them – for assignments and for tests.

# Communication Mechanisms

- **Class web page:** Complete class schedule with links to assignments, lectures, and (often) sample answers for assignments. RELOAD the page every time you visit it. <http://www.cs.pdx.edu/~lmd/cs386>
- **Announcements, Questions, and Answers:** You must register to use the CS386/CS586 site on Piazza. To register, go here: <http://piazza.com/pdx/winter2012/cs386cs586>  
Be sure to supply your name; use an e-mail address that you check regularly. You can decide how often you want to receive notifications.
- **Office hours:**
  - In person and telephone meetings, by request.
  - My office hours: immediately following class (in or near the classroom) plus Mondays from 4 to 5:30PM in FAB 115-12
  - TA office hours: TBA

## Piazza – what can a student do?

- Post a new question.
- “Improve” a question. Students and instructors can change the question, e.g., to make it more clear.
- Provide a “student answer”. Any student can answer a question.
- “Improve” a student answer. Students can change the student answer to a question.
- Mark a question or answers as “good”. Students and instructors can mark questions and answers as “good”.

You can see the history of how it was changed; use the slider.

- Tag a question. For example, put #assignment\_1 at the end of a question, if the question is about Assignment 1.
- Post a follow-up to a question.

# Piazza – what can an instructor do?

- Post questions (just as students can) but questions will mostly come from students.
- Answer a question. Each question has a separate place for the instructor answer. The instructor(s) can also modify or improve the instructor answer.
- Post a follow-up to a question.
- Important: instructors can post “Notes”; I will use these Notes to make announcements about the class – including changes to deadlines (if they ever occur), corrections, etc. This is the only place where announcements will be made. Search for “[tag:instructor-note](#)” or click on the [#instructor-note](#) tag to see all announcements that either I or the TA have made.

# Class Courtesy

Please ...

- Be prompt
- Turn your cell phones off
  - I get to answer any incoming calls ...*
- No headphones or earbuds
- One person talking at a time (except during in-class exercises)

# Motivation for relational databases (and queries)

Using the sailors database from the book

## Consider the following files (Fig. 5.1-5.3 in text)

sailors			
sid	sname	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	40
85	Art	3	25.5
95	Bob	3	63.5

reserves		
sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

boats		
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

## How would you answer these questions? In a spreadsheet ... or in a program ...

- List the name of the red boats.
- List the id and name of the sailors who are over age 36.
- List the id and name of the sailors who are over age 36 or who have a rating that is less than 8.
- List the id and name of the sailors who reserved a boat on 9/8/98.
- List the name of the boats that were reserved on 10/10/98.
- For every reservation, list the reservation date, the sailor id, the sailor name, the boat id, the boat name, and the boat color.

# Can you write programs to answer these questions generically?

- Original: List the names of the red boats.
- List the names of the **<color>** boats.
- List the **<attributes>** of the **<color>** boats.
  
- Original: List the id and name of the sailors who are over age 36.
- List the **<attributes>** of **<table name>** where **<attribute name>** **<comparator>** **<constant>**.
  
- Notice: the last form of query shown just above could answer the first question above about about red boats.

# One assumption that we make for DB tables

For each table, every row has values for the same attributes – in the proper position. (Null values are okay.)


The third row doesn't match the other rows; it has an extra value that's not part of the schema (structure) for this table.

boats		
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

boats			
bid	name	color	
101	Interlake	blue	
102	Interlake	red	
103	Clipper	green	33'
104	Marine	red	



## Another assumption (referential integrity)

- For a table that references another table (like bid in reserves), it must point to a valid row (e.g., to a bid that is in the boats table).
- The second row has an invalid bid (107) because there is no row in the boats table with bid of 107. 

(Only part of the reserves table is shown here.)

boats		
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

reserves		
sid	bid	day
22	101	10/10/98
22	107	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
...		

# SQL Demo

- Class web page: <http://cs.pdx.edu/~lmd/cs386>
- Click on “link to DB information” (near the top)
- Click on “PostgreSQL query page”
- Click on “Database Class” in left panel
- Log in ... with userid: `introdb_readonly`  
and password: `introdb`
- Click on “`introdb_sailors`”
- Click on “public”
- You’ll see the three tables in the Sailors DB

## SQL demo (cont.)

- Click “Browse” on each of the three tables; you can see the data in the database
- Click “SQL” in the upper right corner; you should see a separate, small window.
- Try out some queries like this:
- `SELECT name FROM boats WHERE color = 'red'`
- `SELECT sid, sname FROM sailors WHERE age > 36`
- `SELECT sid, sname FROM sailors WHERE age > 36 OR rating < 8`

## SQL demo (cont.) – with two tables

- `SELECT sailors.sid, sailors.sname  
FROM sailors JOIN reserves USING (sid)  
WHERE day = '10/10/98'`
- This query matches a row from the sailors table with a row from the reserves table where the sid has the same value in each of the rows.
- This syntax only works when the attribute name that you want to join on (sid in this case) is the same in both tables.
- The WHERE clause further restricts the query answer to be only those sailors row/reserves row pairs where the day = '10/10/98'

## SQL demo (cont.) – using two tables

- **SELECT bname**  
**FROM boats JOIN reserves USING (bid)**  
**WHERE day = '10/10/98'**
- Here a row from the boats table is matched with a row from the reserves table when they have the same bid value.
- This syntax works because the join attribute (bid) has the same name in both tables.
- **SELECT boats.bname**  
**FROM boats JOIN reserves USING (bid)**  
**WHERE day = '10/10/98'**  
is equivalent to above query; boats.bname or bname can be used because there's only one bname.

## SQL demo (cont.) – using three tables

- SELECT day, sailors.sid, sname, boats.bid, bname, color  
FROM sailors JOIN reserves USING (sid) JOIN boats USING (bid)
- 
- SELECT day, sailors.sid, sname, boats.bid, bname, color  
FROM sailors JOIN reserves USING (sid) JOIN boats USING (bid)  
WHERE color = 'green'

# Joins when the attribute names are different

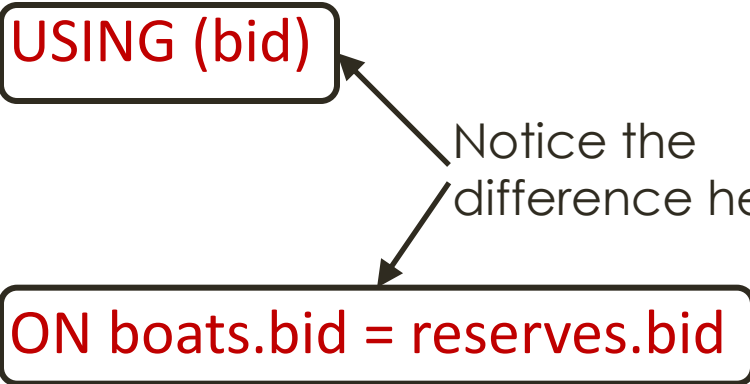
- Consider the following query:

```
SELECT bname  
FROM boats JOIN reserves USING (bid)  
WHERE day = '10/10/98'
```

- This query is equivalent:

```
SELECT bname  
FROM boats JOIN reserves ON boats.bid = reserves.bid  
WHERE day = '10/10/98'
```

Notice the  
difference here



- The “ON” clause allows you to join when the attribute in one table has a different name than attribute in the other table.

## Example – different attribute names

Note: they are STILL foreign keys to sid and bid!

sailors				reserves			boats		
sid	sname	rating	age	sailor	boat	day	bid	bname	color
22	Dustin	7	45	22	101	10/10/98	101	Interlake	blue
29	Brutus	1	33	22	102	10/10/98	102	Interlake	red
31	Lubber	8	55.5	22	103	10/8/98	103	Clipper	green
32	Andy	8	25.5	22	104	10/7/98	104	Marine	red
58	Rusty	10	35	31	102	11/10/98			
64	Horo								
71	Zork								
74	Horo								
85	Art								
95	Bob								

Query for the original schema (that won't work now):

```
SELECT bname  
FROM boats JOIN reserves USING (bid)  
WHERE day = '10/10/98'
```

Equivalent query for the new schema (that will work now):

```
SELECT bname  
FROM boats JOIN reserves ON boats.bid = reserves.boat  
WHERE day = '10/10/98'
```

# Introduction to DB Schemas

Using the sailors database from the book

# Tables can have keys and foreign keys

sailors			
sid	sname	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35
64	Horo		
71	Zork		
74	Horo		
85	Art		
95	Bob		

reserves		
sailor	boat	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98

boats		
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

A **key** for a table: one or more attributes whose values uniquely identify the rows in the table (for all future data). `sid` is unique for all sailors. The combination of (`sailor`, `boat`) is unique for all reserves

A **foreign key** in a table: one or more attributes whose values must match the values of a key in some table. `reserves.sailor` is a foreign key that references `sailors.sid`

## Drawing the schema (1/3)

Here we see tables – with the table name and the attribute names.

sailors

sid	sname	rating	age
-----	-------	--------	-----

reserves

sid	bid	day
-----	-----	-----

boats

bid	name	color
-----	------	-------

Here we see tables with the key for each table underlined and the foreign keys represented by arrows from the foreign key to the key that it references.

sailors

<u>sid</u>	sname	rating	age
------------	-------	--------	-----

reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
------------	------------	------------

boats

<u>bid</u>	name	color
------------	------	-------

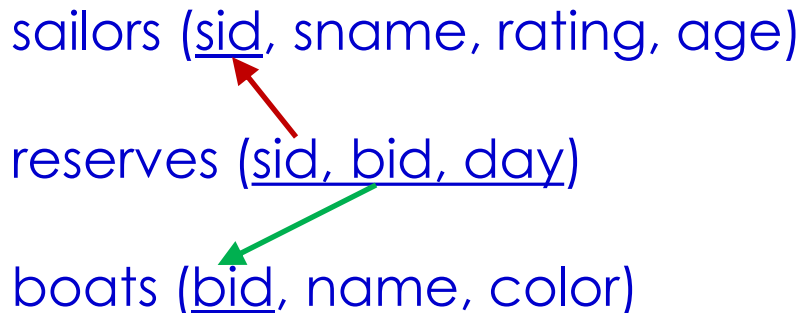
## Drawing the schema (2/3)

Here we see tables –  
with the table name and the attribute names.

sailors(sid, sname, rating, age)  
reserves(sid, bid, day)  
boats (bid, name, color)

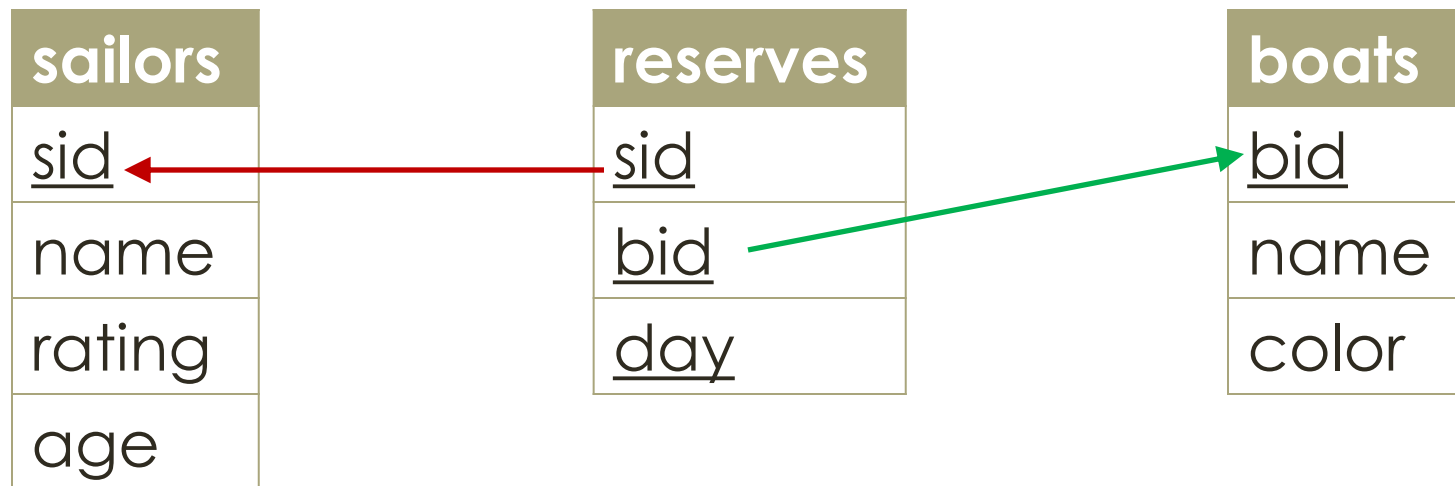
Here we see tables with the key for each table  
underlined and the foreign keys  
represented by arrows from the foreign key to the  
key that it references.

sailors (sid, sname, rating, age)  
reserves (sid, bid, day)  
boats (bid, name, color)



## Drawing the schema for sailors DB (3/3)

Here we see tables with the key for each table underlined and the foreign keys represented by arrows from the foreign key to the key that it references.



Note: attribute names are listed here (NOT data); compare to slide 13, for example.

## Notes on the first assignment

- You will be asked to draw a diagram – exactly in the form of the previous slide for the Spy database in the `introdb_readonly` account. You must show each table in a box, with the table name listed first followed by all attributes in the table. The key for the table must be underlined. And, most importantly, each foreign key must be represented as an arrow that originates at the attribute or attributes that comprise the foreign key and must point to the attribute or attributes that comprise the key of the table that is being references.
- You will also be asked to write a set of queries for the Spy database.

# High-level introduction to databases

# What is computer science?

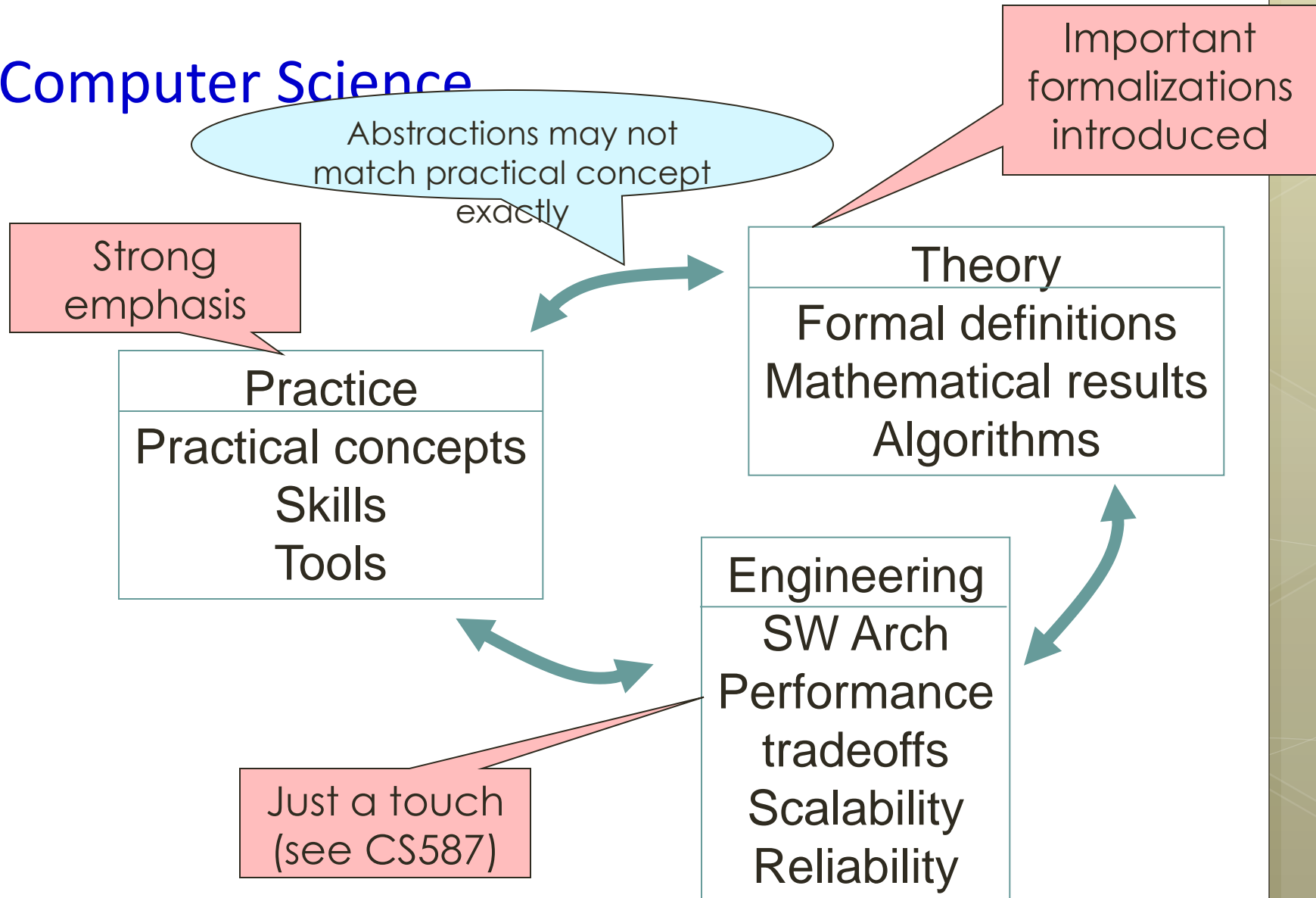
All computer science students must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design.

Final Report of the Joint ACM/IEEE-CS Task Force on Computing Curricula 2001 for Computer Science - a joint undertaking of the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM).

This volume outlines a set of recommendations for undergraduate programs in computer science.

<http://www.computer.org/education/cc2001/final/index.htm>

# Computer Science



# Practice and Theory

## Practice

- Tables, columns, rows, keys
- SQL
- Application structure
- Logical & physical database design
- Transactions
- Security

## Theory

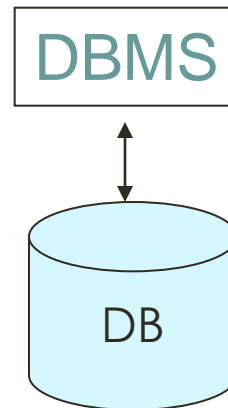
- Relational model: relations, attributes, tuples
- Relational algebra, equivalences
- Functional dependencies, normalization
- Schedules, serializability

# Engineering

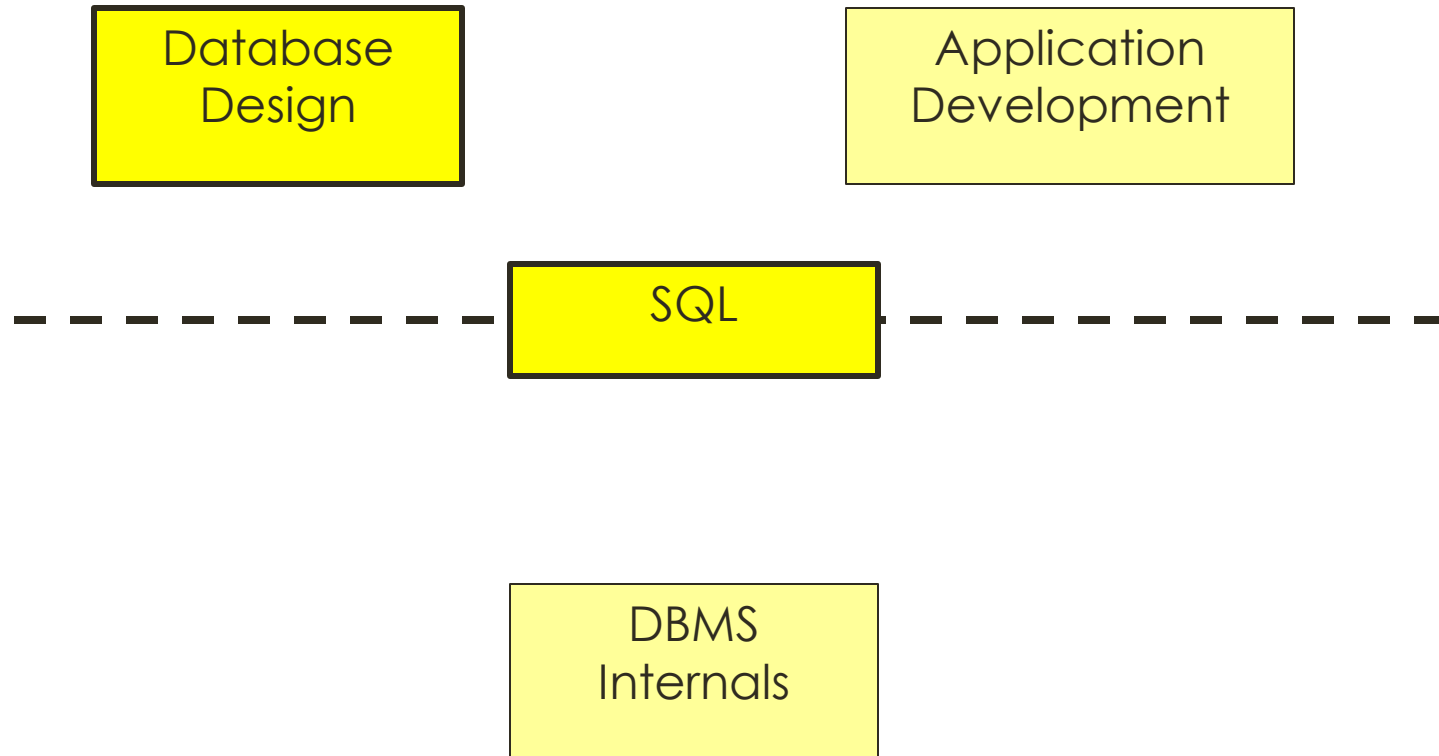
- Basic system structure
- Storage and Indexing
- Query evaluation (operators, optimization)

# What's a DB?

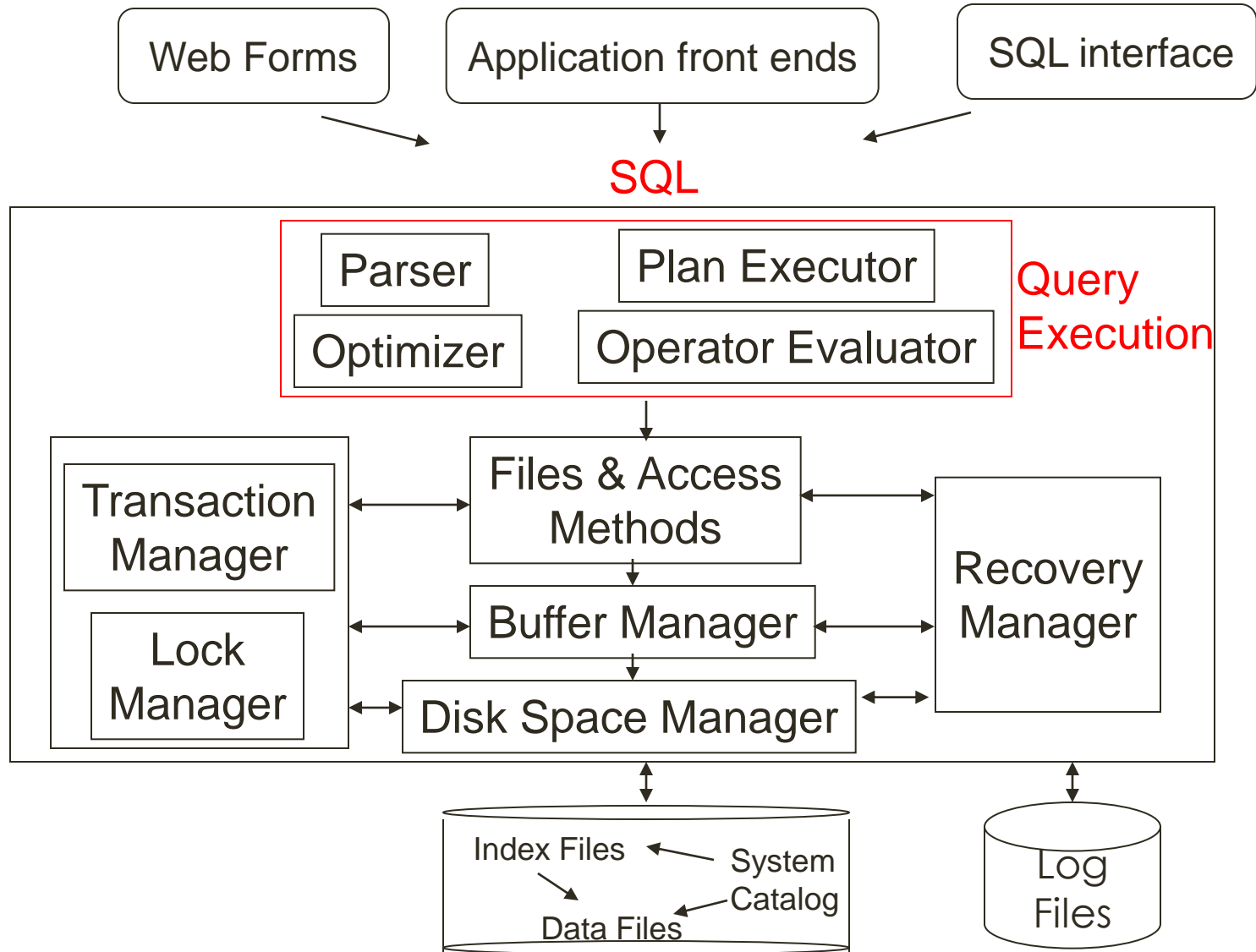
- **database** (DB) - a collection of persistent data  
Persistent: Lifetime not bound to a process
- **database management system** (DBMS) - a software system that supports the definition, population, and query of a database.



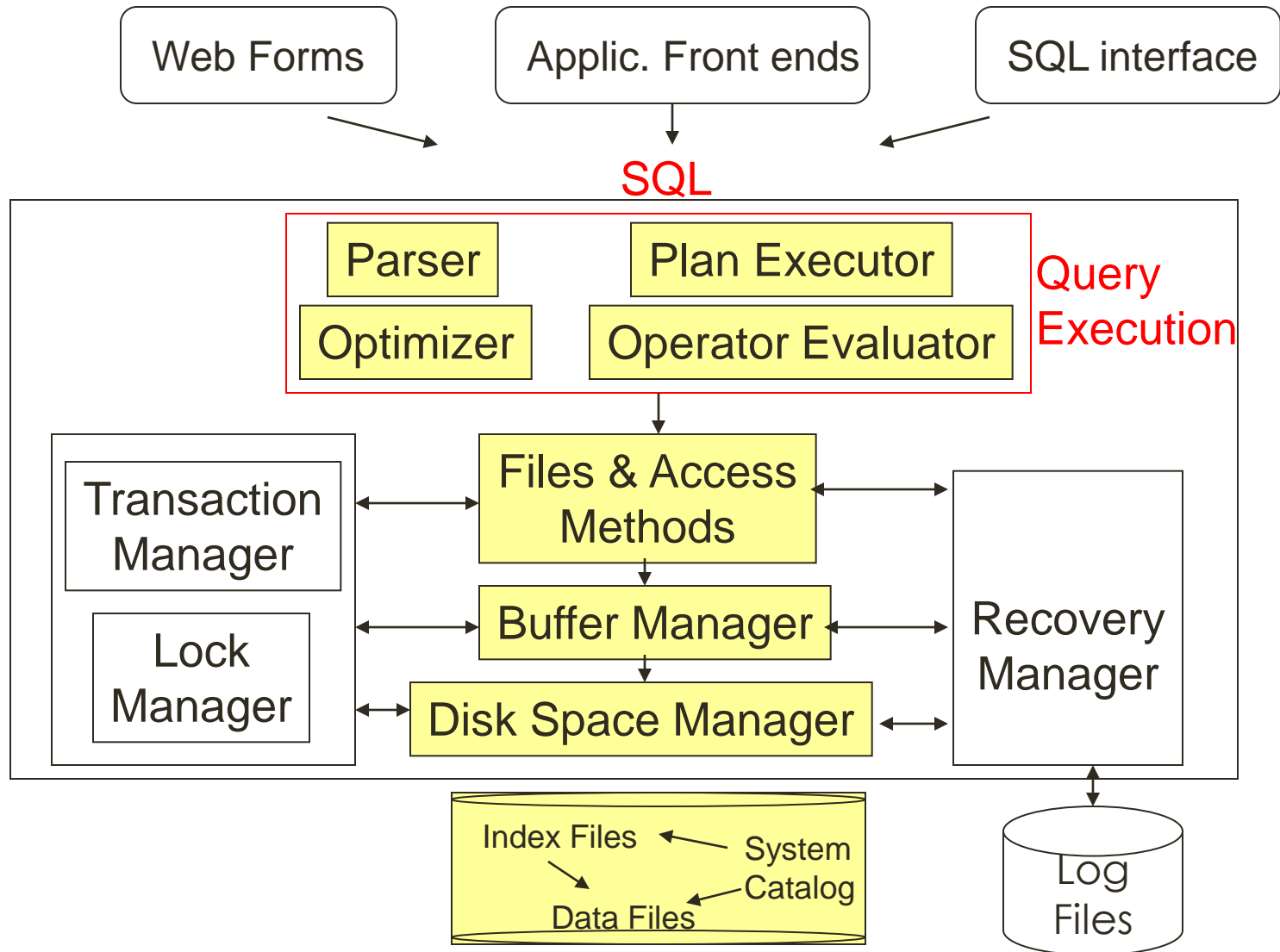
# Where Does This Course Focus



# Database Architecture (Figure 1.3, p. 20)



# Query Processing! (shown in yellow)



Introduce:  
Database terminology  
SQL query language  
from a mostly practical point of view (this week)!

# Introduction to Relational Databases

## Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Imagine that this table (or relation) has been defined to help keep track of bank accounts.

# Table Structure

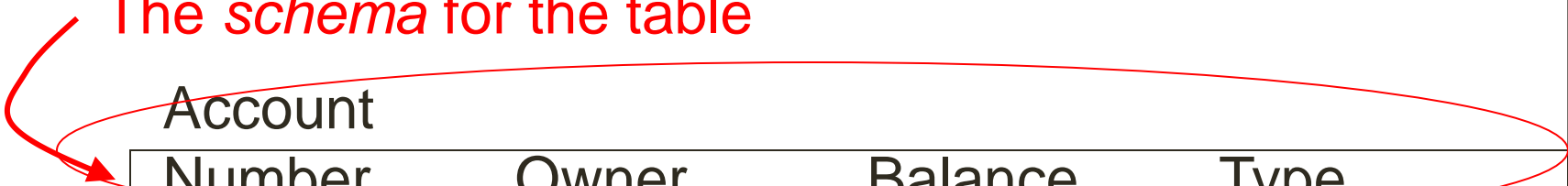
The *name* of the table

The name of the *columns (attributes)*

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

# Table Schema

The *schema* for the table



Account				
Number	Owner	Balance	Type	
101	J. Smith	1000.00	checking	
102	W. Wei	2000.00	checking	
103	J. Smith	5000.00	savings	
104	M. Jones	1000.00	checking	
105	H. Martin	10,000.00	checking	

The **schema** sets the structure of the table. You can think of the schema as the *definition* of the table. (Note, the schema specifies more information than what is shown.)

# Table Rows

Account				
Number	Owner	Balance	Type	
101	J. Smith	1000.00	checking	
102	W. Wei	2000.00	checking	
⋮	J. Smith	5000.00	savings	
⋮	M. Jones	1000.00	checking	
105	H. Martin	10,000.00	checking	

Each entry in the table is called a *row (tuple)*.  
Sometimes an entry in the table is called a record.

# Table Instance

An *instance* of the table...

the current contents or data in the table.

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

## Another Table Instance

Another *instance* of the table  
(two rows added, one (103) deleted)

### Account

Number	Owner	Balance	Type
101	J. Smith	1,000.00	checking
102	W. Wei	2,000.00	checking
104	M. Jones	1,000.00	checking
105	H. Martin	10,000.00	checking
107	W. Yu	7,500.00	savings
109	R. Jones	432.55	checking

new

# Intension vs. Extension

The *intension* of the table

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

The *extension* of the table. Also called the *extent*.

# “Size” of a Table

Degree or arity of a table is the number of columns

Degree of this relation (or table) is 4  
because there are 4 attributes

	Account	Owner	Balance	Type
Cardinality of this instance is 5 (because there are 5 rows)	Number			
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

**Cardinality** of a table = the number of rows in the  
current instance

## Database (One or More Tables)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/00	125.00
	101	925	10/24/00	23.98

# Table Keys

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98

Each table has a key.... where the values must be unique.

## Table Keys (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98

Key may consist of one column or two (or more) columns.

# Connections between Tables

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00
	106	5	12/05/00	555.00

Is this legal?

If not, how do we prevent it from happening?

# Foreign Key

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00
	<del>106</del>	<del>5</del>	<del>12/05/00</del>	<del>555.00</del>

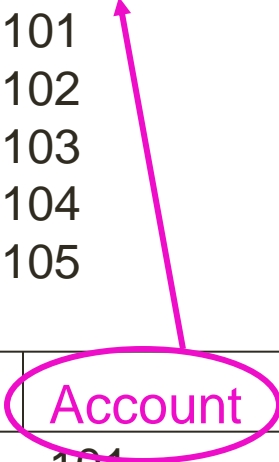
We say that **Deposit.Account** is a *foreign key* that *references* **Account.Number**. If the DBMS enforces this constraint, we have *referential integrity*.

## Foreign Keys (cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98



Are there any foreign keys in the Check table?

Yes, Check.Account is a foreign key that references Account.Number.

# Foreign keys might or might not be part of the key for the referring table

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	110/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/98	23.98

Deposit.Account is **NOT** part of key for Deposit.

Check.Account **IS** part of key for Check.

## Keys for a Table

Consider the following sample data from a table:

1	Jones	28	\$50,000.00

Can you tell what the key for this table is?

## Keys for a Table

Consider the following sample data from a table:  
Can you guess the table name? The attribute names?

1	Jones	28	\$50,000
2	Smith	28	\$60,000

Can you tell what the key for this table might be?  
(Or which attributes can't be the key?)

## Keys for a Table

One possibility:

Person table with Id as the key

<u>Id</u>	Name	Age	Salary
1	Jones	28	\$50,000
2	Smith	28	\$60,000

Use underline to indicate key columns

## Keys, Table Names, Column Names tell us something about the meaning of a table

Another possibility:

Sales Table, by client company, per day

<u>Salesperson</u>	Customer	<u>Day</u>	Volume
1	Jones	28	\$50,000
2	Smith	28	\$60,000

## Database Domains for Columns

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	...			

For every column of every table, **the schema specifies allowable values**. For example,

**Number** must be a 3-digit number

**Owner** must be a 30-character string


**Type** must be “checking” or “savings”

The set of allowable values for an column is called the **domain** of the column.

# Specification of a Relational Schema

- Select the tables, with a **name for each table**.
- Select **column names for each table** and give the **domain for each column**.
- Specify the **key(s)** for each table.
- Specify all appropriate **foreign keys**.

There  
can be  
more than  
one key  
for a  
table.



## Another Example Database

(Keys are underlined. Each table has one key.)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Class-Offering (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Quarter, Course, Section, Grade)

*Do Name and Number mean the same thing everywhere they appear?*

## Example Database (cont.)

(with some foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Quarter, Course, Section, Grade)

What foreign keys are present in the Completed table?

## Example Database (cont.)

(with foreign keys shown informally, with arrows)

Teacher (Number, Name, Office, E-mail)

Course (Number, Name, Description)

Taught-By (Quarter, Course, Section, Teacher, TimeDays)

Student (Number, Name, Major, Advisor)

Completed (Student, Quarter, Course, Section, Grade)

Foreign keys in the Completed table are shown above.

# What are the keys? Do we need more tables?

Recipe (id, name, servings, prep-time)

Ingredient (id, name)

# Possible Keys

Recipe (id, name, servings, prep-time)

Ingredient (id, name)

But...one recipe uses lots of ingredients and one ingredient can be used in lots of recipes..

What can we do?

## Possible tables

Recipe (id, name, servings, prep-time, **ingred-id**)

**Will this work?**

Ingredient (id, name, **recipe-id**)

**Will this work?**

**Should we do both of these?**

# Possible tables

Recipe (id, name, servings, prep-time)

Used-In (recipe-id, ingredient-id)

What's the key for this table?

Ingredient (id, name)

## Another Possibility

Recipe (id, name, servings, prep-time)

Used-In (recipe-id, ingredient-id, quantity)

We need for recipe-id and ingredient-id, together, to be the key for this table.

Ingredient (id, name)

In general, we always need to introduce a new table for a many-to-many relationship

# SQL – the language we use to talk to the Database Management System

SQL can be used for lots of purposes including:

To define tables -

```
CREATE TABLE Account
  (Number integer NOT NULL,
   Owner character,
   Balance currency,
   Type character,
   PRIMARY KEY (Number));
```

To query the database –

```
SELECT      *
FROM        Account
WHERE       Type = "checking ";
```

Notice that all SQL statements end with a semicolon (but PostgreSQL doesn't allow a semicolon at the end of a query).

## SQL (cont.)

To insert rows into a table:

```
INSERT INTO Account  
VALUES (106, " H. Martinez ", 10,000, " savings ");
```

and so forth

SQL is a standard...

and there have been a series of SQL standards: 1986,  
1989, 1992 (SQL2), 1999 (SQL3), ...

But DBMS products differ in how much of the standard they support ... and how many extra features they have.

# Database Schema (first version)

ACCOUNT	<u>Number</u>	Owner	Balance	Type
---------	---------------	-------	---------	------

DEPOSIT	Account	<u>Transaction-id</u>	Date	Amount
---------	---------	-----------------------	------	--------

CHECK	<u>Account</u>	<u>Check-number</u>	Date	Amount
-------	----------------	---------------------	------	--------

# Database Schema (second version)

What are the foreign keys here?

ACCOUNT	<u>Number</u>	Owner	Balance	Type
---------	---------------	-------	---------	------

DEPOSIT	Account	<u>Transaction-id</u>	Date	Amount
---------	---------	-----------------------	------	--------

CHECK	<u>Account</u>	<u>Check-number</u>	Date	Amount
-------	----------------	---------------------	------	--------

ATMWITHDRAWAL	<u>TransactionID</u>	CustId	AcctNo	Amount	WithdrawDate
---------------	----------------------	--------	--------	--------	--------------

CUSTOMER	<u>ID</u>	Name	Phone	Address
----------	-----------	------	-------	---------

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

```
SELECT AcctNo, Amount  
FROM ATMWithdrawal  
WHERE Amount < 50;
```

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

```
SELECT AcctNo, Amount  
FROM ATMWithdrawal  
WHERE Amount < 50;
```



**This is the WHERE clause.**  
**The WHERE clause is evaluated for each row in the table.**

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

Is the amount field of this row  
less than \$50? **YES!**

Amount < 50

## Intermediate Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

Is the amount field of this record less than \$50? **NO!**

Amount < 50

**Ignore this record!**

## Intermediate Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

Is the amount field of this record less than \$50? **YES!**

Amount < 50

## Intermediate Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
3	2	101	\$40.00	11/01/2000 10:05:00

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

Is the amount field of this record less than \$50? **YES!**

Amount < 50

## Intermediate Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00

## ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00
5	2	100	\$200.00	11/08/2000 14:14:00

Is the amount field of this record less than \$50? **NO!**

Amount < 50

**Ignore this record!**

## Intermediate Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/01/2000 9:45:00
3	2	101	\$40.00	11/01/2000 10:05:00
4	2	100	\$40.00	11/01/2000 10:07:00

## Intermediate Query Answer table

<del>TransactionID</del>	<del>CustId</del>	AcctNo	Amount	<del>WithdrawDate</del>
<del>1</del>	<del>1</del>	102	\$25.00	<del>11/01/2000 9:45:00</del>
<del>3</del>	<del>2</del>	101	\$40.00	<del>11/01/2000 10:05:00</del>
<del>4</del>	<del>2</del>	100	\$40.00	<del>11/01/2000 10:07:00</del>

```
SELECT AcctNo, Amount
FROM ATMWithdrawal
WHERE Amount < 50;
```

*Consider the attributes listed in the SELECT clause.*

*Throw away attributes that are not listed.*

*Thus the final query answer is:*

## Final Query Answer table

AcctNo	Amount
102	\$25.00
101	\$40.00
100	\$40.00

## Another SQL Query (using one table)

ATMWithdrawal					
TransactionId	CustId	AcctNo	Amount	WithdrawDate	
1	1	102	\$25.00	11/01/00	9:45:00 AM
2	1	102	\$150.00	11/10/00	1:15:00 PM
3	2	101	\$40.00	11/01/00	10:05:00 AM
4	2	100	\$40.00	11/01/00	10:07:00 AM
5	2	100	\$200.00	11/08/00	2:14:00 PM

```
SELECT *  
FROM ATMWithdrawal  
WHERE TransactionId = 3;
```

The five rows are considered, one by one, to see if **TransactionId = 3** (to see if the WHERE clause evaluates to true).

```
SELECT *  
FROM ATMWithdrawal  
WHERE TransactionId = 3;
```

Note: "\*" in  
SELECT clause  
means "all attributes"

ATMWithdrawal					
TransactionId	CustId	AcctNo	Amount	WithdrawDate	
1	1	102	\$25.00	11/01/00	9:45:00 AM
2	1	102	\$150.00	11/10/00	1:15:00 PM
3	2	101	\$40.00	11/01/00	10:05:00 AM
4	2	100	\$40.00	11/01/00	10:07:00 AM
5	2	100	\$200.00	11/08/00	2:14:00 PM

Query Answer is:

TransactionId	CustId	AcctNo	Amount	WithdrawDate	
3	2	101	\$40.00	11/01/00	10:05:00 AM

## Example Query

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT *  
FROM Account  
WHERE Type = "checking";
```

## Example Query with Answer

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT *  
FROM Account  
WHERE Type = "checking";
```

	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

## Another Query

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT *  
FROM Account  
WHERE Type = "savings";
```

## ...with its Query Answer

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT *  
FROM Account  
WHERE Type = "savings";
```

	Number	Owner	Balance	Type
	103	J. Smith	5000.00	savings

## Yet Another Query (what's different?)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT      Owner
FROM        Account
WHERE       Type = "checking";
```

## ...the query answer

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT      Owner
FROM        Account
WHERE       Type = "checking";
```

Owner
-------

J. Smith

W. Wei

M. Jones

H. Martin

## Example (Useless) Query

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

```
SELECT *  
FROM Account  
WHERE Type = "checking" AND  
Type = "savings";
```

## Example (Useless) Query with Answer

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Query  
answer  
is empty.  
But that's  
allowed.

```
SELECT *
FROM Account
WHERE Type = "checking" AND
Type = "savings";
```

Number	Owner	Balance	Type
--------	-------	---------	------

So... why is this a “useless” query?

# How an SQL query is evaluated

Third, the SELECT clause tells us which columns to keep in the query answer.

SELECT  
FROM  
WHERE

AcctNo, Amount  
ATMWithdrawal  
Amount < 50;

First, the FROM clause tells us the input tables.

Second, the WHERE clause is evaluated for all possible combinations from the input tables.

# Quick Exercise

Using Account table

ACCOUNT	<u>Number</u>	Owner	Balance	Type
---------	---------------	-------	---------	------

Write SQL queries that determine

1. What types of accounts does Smith have?
2. What is the account number of savings accounts with a balance less than \$1000

*And find the answers to your queries*

Is Query 2. a useless query?

## SQL query using two tables

```
SELECT      A.Owner, A.Balance  
FROM        Account A, Deposit D  
WHERE       D.Account = A.Number and A.Balance > 1000;
```

**How does this work?**  
**Which rows, from which tables,  
are evaluated in the WHERE clause?**

## SQL query using two tables

```
SELECT      A.Owner, A.Balance
FROM        Account A, Deposit D
WHERE       D.Account = A.Number and A.Balance > 1000;
```

“A” is a correlation name for Account

“D” is a correlation name for Deposit.

Correlation names are like local variables – they hold one tuple or row from the corresponding table.

You choose correlation names when you write the query.

Or – you can use the entire table name.

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

```

SELECT      A.Owner, A.Balance
FROM        Account A, Deposit D
WHERE       D.Account = A.Number and A.Balance > 1000;

```

We must check every combination of one row from Account with one row from Deposit.

## Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

## Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

No! Throw  
it away.

WHERE D.Account = A.Number and A.Balance > 1000;



Number	Owner	Balance	Type	Account	T-id	Date	Amount
--------	-------	---------	------	---------	------	------	--------

notice  
the  
columns

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Yes! Place in query answer.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Yes! Place in query answer.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

All combinations fail! →

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

No! Throw it away. Why?

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

The first three fail.

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

<b>Account</b>			
Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

<b>Deposit</b>			
Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/02/00	10,000.00

Yes! Place in query answer.

WHERE D.Account = A.Number and A.Balance > 1000;

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10,000.00

## Intermediate result (after processing the FROM & WHERE clauses)

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10,000.00

SELECT            A.Owner, A.Balance            Process the SELECT  
FROM              Account A, Deposit D  
WHERE             D.Account = A.Number and A.Balance > 1000;

Final query  
answer:  
(notice that  
W. Wei appears twice)

Owner	Balance
W. Wei	2000.00
W. Wei	2000.00
H. Martin	10,000.00

## Intermediate result

(after processing the FROM & WHERE clauses)

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10,000.00

Process the SELECT

```
SELECT  DISTINCT A.Owner, A.Balance
FROM    Account A, Deposit D
WHERE   D.Account = A.Number and A.Balance > 1000;
```

If we use the word  
DISTINCT, then  
duplicate rows are removed  
from the query answer.  
W. Wei only appears once.

Owner	Balance
W. Wei	2000.00
H. Martin	10,000.00

# Another SQL query using two tables

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

```
SELECT      A.Number, A.Owner
FROM        Account AS A, Deposit AS D
WHERE       A.Number = D.Account and D.Amount > 300;
```

How many rows will be in the query answer?

How many columns will be in the query answer?

# SQL query using two tables(cont.)

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

```
SELECT A.Number, A.Owner
FROM Account AS A, Deposit AS D
WHERE A.Number = D.Account and D.Amount > 300;
```

	Number	Owner
	102	W. Wei
	104	M. Jones
	105	H. Martin

# Queries

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Notice that a query is expressed against the schema.

SELECT  
FROM  
WHERE

Owner  
Account  
Type = "checking";



But the query **runs** or **executes** against the **instance (the data)**

Owner
-------

J. Smith  
W. Wei  
M. Jones  
H. Martin



*And may give different answers on different instances*

# Comments on Queries

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Notice that **the answer to a query is always a table!**  
It doesn't have a name (for the table) unless you provide it.  
The attribute names are deduced from the input tables (or supplied by the query author). It might or might not have any rows.



	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

# Creating temporary tables using INTO

We can create a name for the query answer:

```
SELECT      Owner INTO temp3
FROM        Account
WHERE       Type = "checking";
```



temp3	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

temp3 can be used as a table in subsequent queries!  
**REMEMBER TO DELETE YOUR TEMPORARY TABLES!!**

## Comments on Queries

Because **the answer to a relational query is always a table**

.....

we can use the answer from one query as input to another query.

This means that we can create arbitrarily complex queries!

A relational query languages is **closed** if it has this property.