

Databases and SQL

an introduction

Denis Khryashchev / Huy T. Vo
(many slides were derived from Prof. Freire's)



The City College
of New York



Agenda

- Why study databases?
- Why use databases?
- Introduction to Relational Databases
- Representing structured data with the Relational Model
- Accessing and querying data using SQL



Why study databases?

- Databases used to be specialized applications, now they are a central component in computing environments
- Knowledge of a database concept is essential for computer scientists, software engineers, and anyone who needs to manipulate data



Why study databases?

- **Databases are everywhere, even when you don't see them: most activities involve data**
 - Banking + credit cards: all transactions
 - Airlines: reservations, schedules
 - Universities: registration, grades
 - Sales: customers, products, purchases
 - Manufacturing: production, inventory, orders, supply chain
 - Human resources: employee records, salaries, tax deductions
 - Web sites: front end to information stored in databases; e.g., Google, YouTube, Flickr, Amazon...
 - Scientific research, e.g., studying the environment, cities, ...
- Data need to be ***managed***

Why study databases?

- **Data are valuable:**
 - E.g., bank account records, tax records, student records, your videos and photos...
 - Data must be ***protected*** - no matter what happens whether we have machine crashes, disk crashes, hurricanes/floods;
 - They also need to be protected from people



Why study databases?

- Data are often structured:


Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	
1	20361767	08/24/2013 02:37:36 AM	08/24/2013 03:09:54 AM	NYPD	New York City Police Depa	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10304	645 BAY STREET
2	20361829	08/24/2013 02:34:30 AM	08/24/2013 03:17:14 AM	NYPD	New York City Police Depa	Blocked Driveway	No Access	Street/Sidewalk	11205	31 ELLIOTT PLACE
3	20366804		02:32:46 AM	NYPD	New York City Police Depa	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk	10012	620 BROADWAY
4	20362779		03:15:10 AM	NYPD	New York City Police Depa	Blocked Driveway	Partial Access	Street/Sidewalk	11218	562 EAST 9 STREET
5	20364390			NYPD	New York City Police Depa	Blocked Driveway	Partial Access	Street/Sidewalk	11379	58-15 86 STREET
6	20366039			DOHMH	Department of Health and	Indoor Air Quality	Chemical Vapors/Gases/Odors	3+ Family Apartment Building	10031	541 WEST 142 STREET
7	20365752		02:28:36 AM	NYPD	New York City R					2357 CROTONA AVENUE
8	20364370			DOHMH	Department of					605 MORRIS PARK AVENUE
9	20361880		03:04:45 AM	NYPD	New York City R					20-15 74 STREET
10	20364368			DOHMH	Department of					584 MORRIS PARK AVENUE
11	20364887			DOHMH	Department of					562 MORRIS PARK AVENUE
12	20360837			NYPD	New York City R					87-06 17TH STREET
13	20367406			DOHMH	Department of					534 MORRIS PARK AVENUE
14	20361809			DOHMH	Department of					1750 VAN BUREN STREET
15	20362539			DOHMH	Department of					461 MORRIS PARK AVENUE
16	20364157			DOHMH	Department of					518 MORRIS PARK AVENUE

Steven Spielberg

From Wikipedia, the free encyclopedia

Steven Allan Spielberg (born December 18, 1946)^(?) is an American **film director**, **screenwriter**, and **film producer**. In a career spanning six decades, Spielberg's films have taken up many themes and genres. Spielberg's early **science-fiction** and **adventure films** were seen as an archetype of modern **Hollywood blockbuster** filmmaking. In later years, his films began addressing such issues as the **Holocaust**, **slavery**, **war** and **terrorism**.

Spielberg won the **Academy Award for Best Director** for *Schindler's List* (1993) and *Saving Private Ryan* (1998). Three of Spielberg's films - *Jaws* (1975), *E.T. the Extra-Terrestrial* (1982), and *Jurassic Park* (1993) - achieved **box office records**, each becoming the highest-grossing film made at the



Spielberg at the Pentagon (August 11, 1995).

Born	Steven Allan Spielberg
	December 18, 1946 (age 63)
	Cincinnati, Ohio, U.S.
Occupation	Film director, producer, screenwriter
Years active	1964-present
Spouse(s)	Amy Irving (m. 1985–1989)
	Kate Capshaw (m. 1991–present)

Hitachi Deskstar 7K500 - hard drive - 500 GB - SATA-300

\$53 and up (6 stores) **cashback** · 2%
★★★★☆ **user reviews** (1)




The Hitachi Deskstar 7K500 hard disk drive extends the company's long-standing tradition of performance and reliability leadership. Hitachi's standardized features in desktop solutions enable fast transfer rates, low power utilization and quiet acoustics.... [more...](#)

Share Facebook Twitter Email

[See larger photo](#)

See also: [Product Summary](#) · [Where to Buy](#) · [User Reviews](#) · [Expert Reviews](#) · [Specifications](#)

WHERE TO BUY »

PRODUCT	SELLER	PRICE
 Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal	ServerSupply.com	\$53.00
Go to store		
 Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal	ALLHDD.COM	\$64.00
Go to store		
 Deskstar 7K500 Hard Drive - 500GB - 7200rpm - Internal	Assembly Alliance Electronics	\$69.69
Go to store		

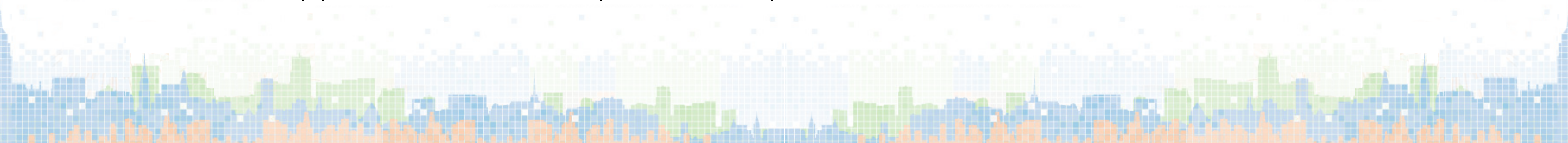
Why study databases?

- Data are often structured
- We can exploit this regular structure to store data efficiently
- Data should be ***retrieved*** in useful ways (that is, we can use a query language)



Why study databases?

- Because the database field has made a number of contributions to basic computer science
 - Databases are behind many of important contributions and impact that CS has had
 - Find, gather, analyze and understand data, e.g., Banks, human genome, e-commerce
- Understand concepts and apply to different problems and different areas, e.g., Big Data
- Because DBMS software is highly successful as a commercial technology (Oracle, DB2, MS SQL Server...)
- Because DB research is highly active and **very** interesting!
 - Lots of opportunities to have practical impact



Why study databases?

- Big data platforms use the structured query language of databases (SQL)
 - both human and machine “readable”
- A common interface for accessing data, e.g. an abstraction layer for computer applications to deal with data

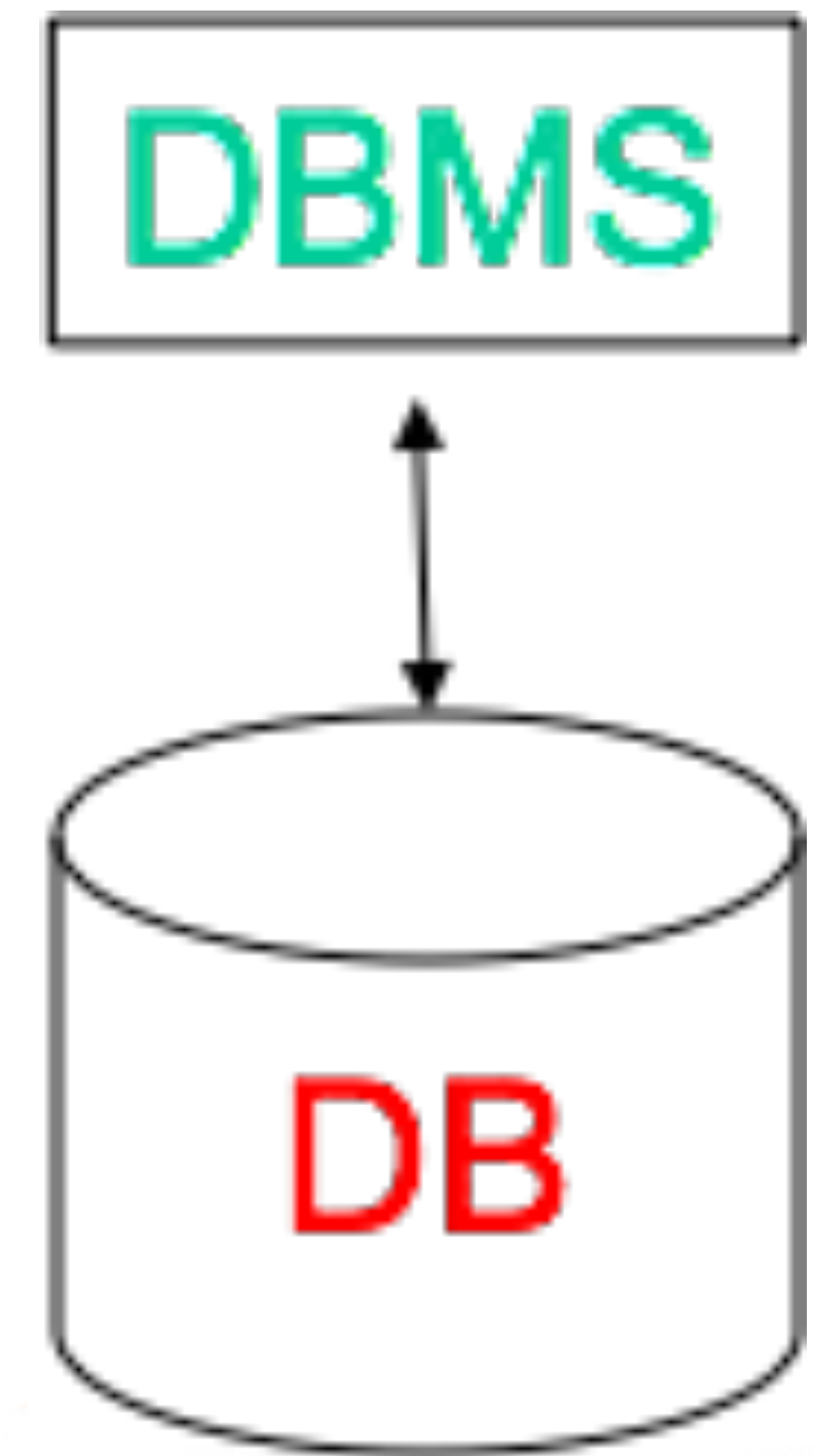


Database Systems: the Basics



Databases and Database Management Systems

- Database (DB) is an integrated collection of data
 - Models real-world objects
 - Entities (e.g., people, residence, Christmas cards)
 - Relationships (e.g., John Doe lives on 123 Elm St)
 - Captures structure — allows data to be **queried**
- A Database (Management) System (DBMS) is a software suite designed to store and manage databases
 - Provides environment that is both convenient and efficient to use
 - Address all complications discussed



Storing Data: Database vs. File System

- Once upon a time database applications were built on top of file systems...
- But this has many drawbacks:
 - Data redundancy, inconsistency and isolation
 - Multiple file formats, duplication of information in different files
 - Difficulty in accessing data: need to write a new program to carry out each new task, e.g., search people by zip code or last name; update telephone number
 - Integrity problems: integrity constraints (e.g., num_residence = 1) become part of program code -- hard to add new constraints or change existing ones
- Atomicity of updates
 - Failures may leave database in an inconsistent state with partial updates carried out, e.g., *John and Mary get married, add new residence, update John's entry, and database crashes while Mary's entry is being updated...*

Storing Data: Database vs. File System

- Concurrent access by multiple users
 - Needed for performance: can you imagine if only 1 person at a time could buy a ticket from Delta?
 - Uncontrolled concurrent access can lead to inconsistencies, e.g., the same seat could be sold multiple times...
- There are 3 seats left; I buy 2 seats; John buys 3 seats at the same time
- If I hit enter 1st there will be 1 seat left; if John is faster there will be 0; but 5 seats have been sold and we will fight at the airport!

Database systems offer solutions to all the above problems



Why use Database Systems?

- Data independence and efficient access
 - Easy + efficient access through declarative query languages and optimization
- Data integrity and security
 - Safeguarding data from failures and malicious access
- Concurrent access
- Reduced application development time
- Uniform data administration
- ***When should you not use a database?***

What's in a DBMS?

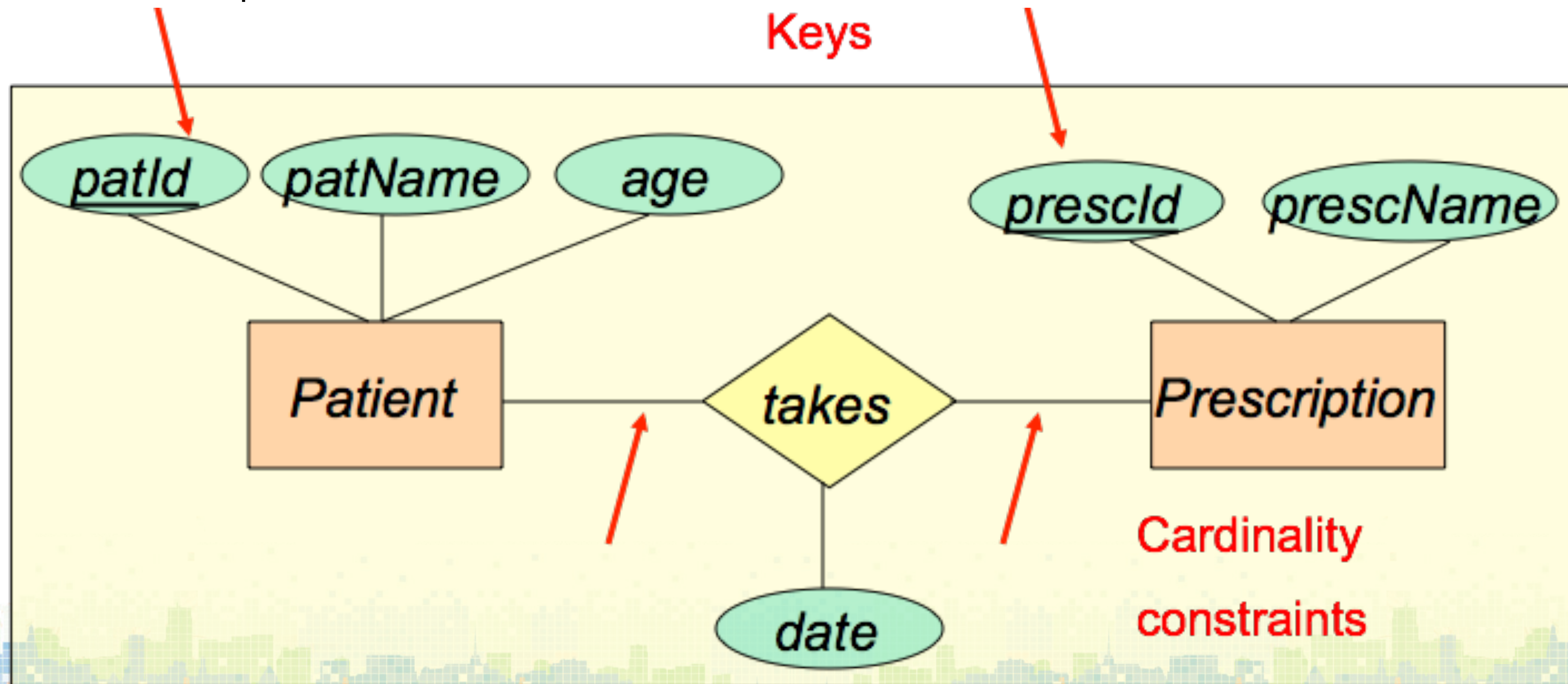
	<i>Data model</i>	<i>Query language</i>	<i>Transactions and crash recovery</i>
<i>“above the water”</i> ↑	Logical DB design	SQL, QBE, views	Transactions
	Relational Model	XPath, XQuery	
XML data model			
<i>“below the water”</i> ↓	Map data to files	Query optimization	Locking
	Clustering	Query evaluation	Concurrency control
	Indexes		Recovery
			Logs

Designing a database: The Conceptual Model

- What are the **entities** and **relationships** among these entities in the application?
- What information about these entities and relationships should we store in the database?
- What are the integrity constraints or business rules that hold?
- Different applications have different needs, and different perspectives – even to model the same object
 - billing department:
 - `patient(id, name, insurance, address)`
 - `visit(patientId, procedure, date, charge)`
 - in-patient:
 - `patient(id, name, age, address)`
 - `alergies(id, alergies)`
 - `prescription(patientId, date, medicine)`

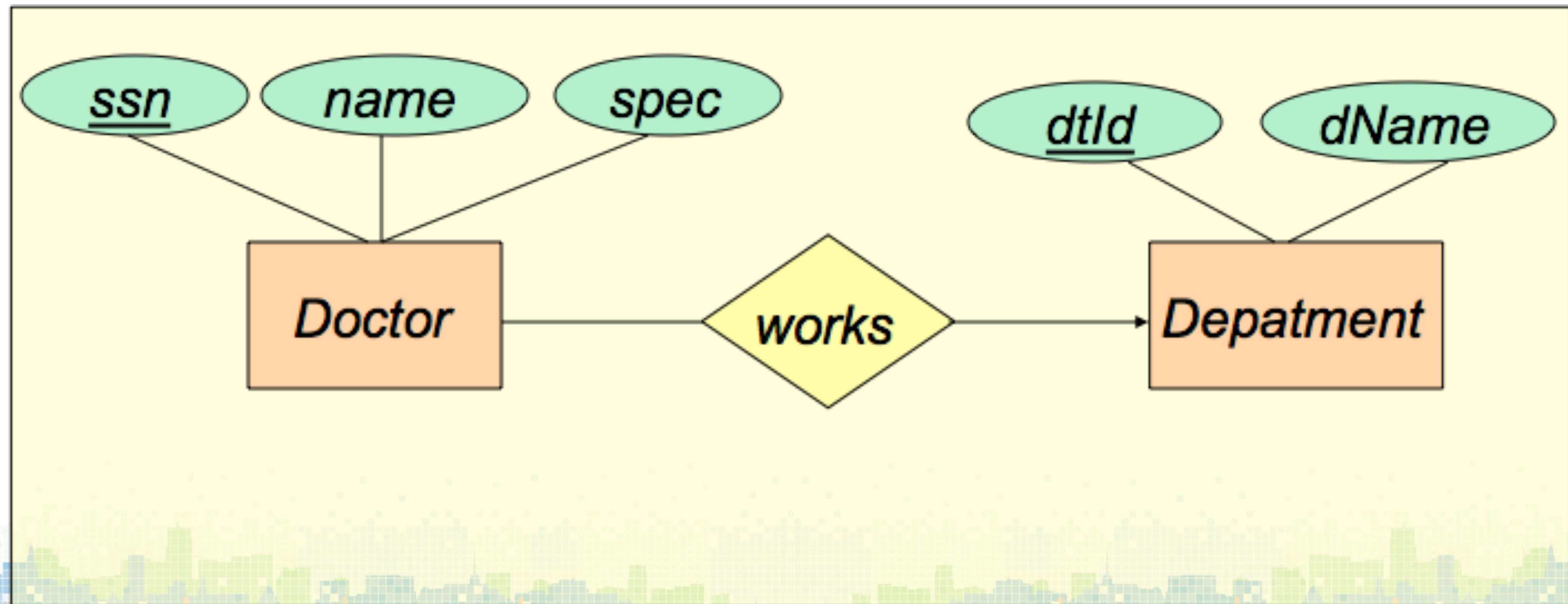
The Entity Relationship (ER) Data Model

- A ***data model*** is a collection of concepts for describing data, relationships, semantics and constraints



ER: Another Example

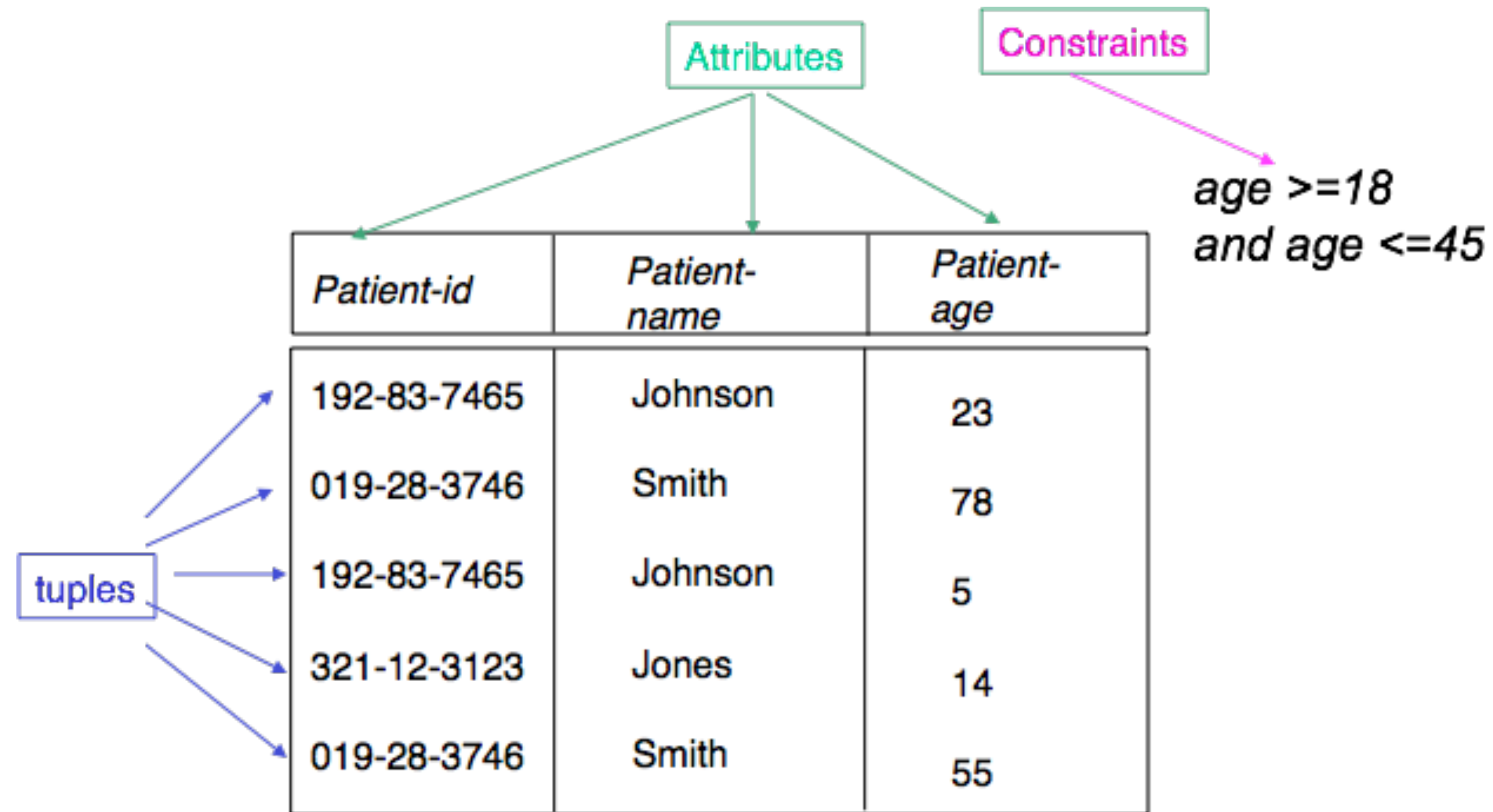
- A department has many doctors, but a doctor can only work in one department



Relational Data Model

- ER used for conceptual design is then mapped into the relational model
- The ***relational model of data*** is the most widely used model today
 - Main concept: ***relation***, basically a table with rows and columns
 - Every relation has a ***schema***, which describes the columns, or fields
- A ***schema*** is a description of a particular collection of data, using a given data model
 - `Patient(patientId : int, patientName : str, age : int)`
 - `Takes(patientId : int, prescId, prescDate : date)`
 - `Prescription(prescId : int, presName : str)`

Relational Model: Terminology



schema

Patient(patientId:int, patientName:str, age: int)

Pitfalls in Relational Database Design

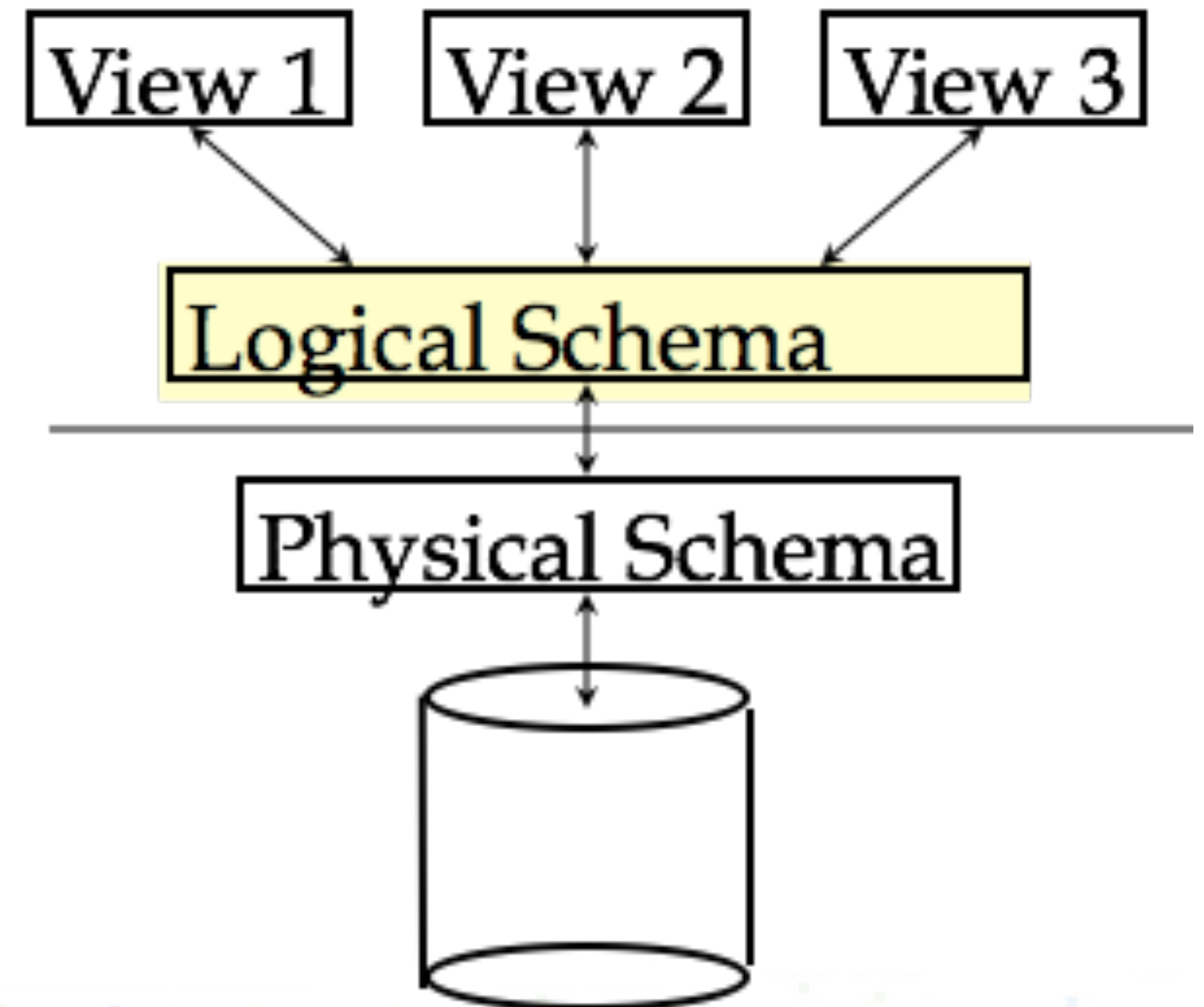
- Find a **“good” collection of relation schemas**
- Bad design may lead to
 - Repetition of information inconsistencies!
 - E.g., keeping people and addresses in a single file
 - **Inability to represent certain information**
 - E.g., a doctor that is both a cardiologist and a pediatrician
- Design Goals:
 - Avoid redundant data
 - Ensure that relationships among attributes are represented
 - Ensure constraints are properly modeled: updates check for violation of database integrity constraints

Query Languages

- **Query languages:** Allow *manipulation* and *retrieval* of data from a database
- Queries are posed with respect to *data model*
 - Operations over objects defined in data model
- Relational model supports simple, powerful QLs:
 - Strong formal foundation based on logic
 - Allows for optimization
- Query Languages **!=** programming languages
 - QLs support easy, efficient access to large data sets
 - QLs not expected to be “*Turing complete*”
 - QLs not intended to be used for complex calculations

Level of Abstraction

- Many **views**, single **conceptual (logical) schema** and **physical schema**
 - Views describe how users see the data
 - Logical schema defines logical structure
 - Physical schema describes the files and indexes used



Example: University Catalog

- Physical schema:
 - Students stored in id order
 - Index on last name
- Logical schema:
 - Students (sid: string, name: string, login: string, age: integer, gpa: real)
 - Courses (cid: string, cname: string, credits: integer)
 - Enrolled (sid : string, cid: string, grade: string)
- External Schema (View):
 - Course_info (cid: string, enrollment: integer)

Data Independence

- Applications insulated from how data is structured and stored
- Logical data independence: Protection from changes in logical structure of data
 - Changes in the logical schema do not affect users as long as their views are still available
- Physical data independence: Protection from changes in physical structure of data
 - Changes in the physical layout of the data or in the indexes used do not affect the logical relations

One of the most important benefits of using DBMS!

Storage and Indexing

- The DB administrator designs the physical structures
- Nowadays, database systems can do (some of) this automatically
- File structures: sequential, hashing, clustering, single or multiple disks, etc.
- Indexes:
 - Select attributes to index
 - Select the type of index
- ***Storage manager*** is a module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system:
 - interaction with the file manager
 - efficient storing, retrieving and updating of data

Query Optimization and Evaluation

- DBMS must provide efficient access to data
 - In an emergency, can't wait 10 minutes to find patient allergies
- Declarative queries are translated into imperative query plans
 - Declarative queries — logical data model
 - Imperative plans — physical structure
- Relational optimizers aim to find the best imperative plans (i.e., shortest execution time)
 - In practice they avoid the worst plans...

Transaction: an Execution of a DB program

- Key concept is **transaction**, which is an **atomic** sequence of database actions (reads/writes)
- Each transaction, executed completely, must leave the DB in a **consistent state** if DB is consistent when the transaction begins
 - Ensuring that a transaction (run alone) preserves consistency is ultimately the programmer's responsibility!
- **Transaction-management** component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures
 - DBMS ensures **atomicity** (all-or-nothing property)

Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance
- But interleaving actions of different user programs can lead to inconsistency
 - e.g., nurse and doctor can simultaneously edit a patient record
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system



Databases make these folks happy...

- End users
- DBMS vendors: \$20B+ industry
- DB application programmers
- Database administrator (DBA)
 - Designs logical /physical schemas
 - Handles security and authorization
 - Data availability, crash recovery
 - Database tuning as needs evolve




Summary

- DBMS used to maintain and query (large) structured datasets
- Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security
- Levels of abstraction give data independence



Common Database Model

- Relational
 - Semi-structured/XML
 - Object-oriented
 - Object-relational
 - Hierarchical
 - Network
 - ...
- 

Why Study the Relational DB Model?

- Extremely useful and simple
 - Single data-modeling concept: relations = 2-D tables
 - Allows clean yet powerful manipulation languages
- Most widely used model
 - Vendors: IBM, Microsoft, Oracle
 - Open source: MySQL
- Some competitors: object-oriented model, semi-structured (XML) model
- A synthesis emerging:
 - Object-relational model: Informix Universal Server, UniSQL, O2, Oracle, DB2
 - XML-enabled databases: Oracle, DB2, SQLServer

Example: A Relation

*Relation
name*

Account

The Account relation keeps track of bank accounts. Facts about real-world entities: *J. Smith owns a checking account whose number is 101 and balance is 1000.00*

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

Example: Attributes of a Relation

Attribute **domains**:

Number must be a 3-digit number

Owner must be a 30-character string

Type must be “checking” or “savings”

The name of the attributes (columns)

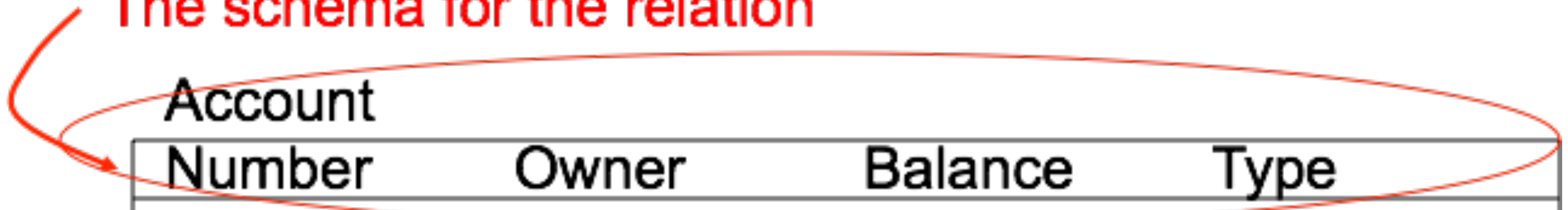
Account Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Example: Relation Schema

The schema sets the structure of the relation---it is the definition of the relation.

Note: the schema specifies more information than what is shown, e.g., keys, constraints...

The schema for the relation



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

Terminology of Relational Database

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

rows/tuples

The extension of the table

Orders of tuples is irrelevant. Relation is a set!

Why do we “need” this model?

- Why not use tables defined in Java or C?
- The relational model provides physical independence
 - Tables can be stored in many different ways, but they have the same logical representation
- Operations can be expressed in *relational algebra*
 - Table-oriented operations — simple
 - Limited set of operations is a strength: queries can be automatically optimized

Relational Database

- A database consists of multiple relations
- Information is distributed among relations. E.g.:
 - account: information about accounts
 - deposit: information about deposits into accounts
 - check : information about checks
- What would happen if we stored all information in a single table?
 - repetition of information (e.g., two customers own an account, two deposits on the same account)
 - the need for null values (e.g., represent a customer without an account)

Example: Relational Database

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	11/2/00	10,000.00

Check	Account	Check-number	Date	Amount
	101	924	10/23/98	125.00
	101	925	10/24/99	23.98

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

Example: Relational Database

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	11/2/00	10,000.00
	106	5	12/5/00	555.00

Is this legal?

If not, how do we prevent it from happening?

Example: Relational Database

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	11/2/00	10,000.00
	106	5	12/5/00	555.00

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

Primary Key vs. Unique

- There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes
- No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.



Relational Model vs. XML

- Relations: Schema must be fixed in advance
XML: Does not require predefined, fixed schema
- Relations: Rigid flat table structure
XML: Flexible hierarchical structure (defined by regular expressions)
- Relations: Easy to understand, simple query language
XML: Can be harder to understand, more complex query language
- Relations: Ordering of data not relevant (tuple ordering or attribute ordering)
XML: Ordering forced by document format, may or may not be relevant
- Relations: Transmission and sharing can be problematic
XML: Designed for easy representation and exchange
- Relations: "Native" data model for all current widely-used commercial DBMSs
XML: "Add-on," often implemented on top of relations

Structured Query Language (SQL)



What is SQL?

- SQL = *data manipulation* + *data definition* + *control*
- Data manipulation: ad-hoc queries and updates

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

- Data definition: creation of tables and views

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

- Control: assertions to protect data integrity

```
CHECK (Owner IS NOT NULL)
```



SQL in Action

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

**Attributes of the
resulting relation**

```
SELECT *  
FROM ATMWithdrawal  
WHERE Amount < 50;
```

**Relation to which
the query refers**

**Condition that
must be satisfied**

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

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

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

Amount < 50

Query Answer table

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

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

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

Amount < 50

Ignore this record!

Query Answer table

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

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

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

Amount < 50

Query Answer table

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

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

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

Amount < 50

Query Answer table

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

ATMWithdrawal table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	\$25.00	11/1/2000 9:45:00
2	1	102	\$150.00	11/10/2000 13:15:00
3	2	101	\$40.00	11/1/2000 10:05:00
4	2	100	\$40.00	11/1/2000 10:07:00
5	2	100	\$200.00	11/8/2000 14:14:00

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

Amount < 50

Ignore this record!

Query Answer table

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

Conditions in the WHERE clause

- Conditions evaluate to a Boolean value: TRUE or FALSE (and also UNKNOWN...)
- Expressions built with comparison operators: =, <>, <, >, <=, and >=
 - E.g., Amount = 50; Amount <> 50
- Values to be compared can be
 - Attributes of relations in FROM clause
 - Constraints
 - Arithmetic expressions, e.g., Amount < Credit - Balance
- Expressions composed with logical connectives: and, or, not
 - E.g., Amount < 50 and CustID <> 1

Projection in SQL

Query Answer table (Amount < 50)

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

```
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

```
SELECT AcctNo AS Number, Amount AS Amt
FROM ATMWithdrawal
WHERE Amount < 50;
```

Renaming
attributes

- Result will be the same, but with different column headers

Number	Amt
102	\$25.00
101	\$40.00
100	\$40.00

Generalized
Projection

```
SELECT AcctNo AS Number, Amount*10 AS Amt
FROM ATMWithdrawal
WHERE Amount < 50;
```

Number	Amt
102	\$250.00
101	\$400.00
100	\$400.00

How is an SQL query evaluated?

Third, the **SELECT** clause tells us which attributes 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.

More extensions: aggregation

We can use *aggregate* operators in the SELECT clause:

COUNT, **SUM**, **MIN**, **MAX**, and **AVG**

- Produces the SUM, MIN, MAX, AVG of the values in a column
 - For SUM and AVG, values must be numeric
- COUNT produces the number of values in a column

```
SELECT AVG (CBalance)
```

```
FROM Customer;
```

```
SELECT MIN(CBalance), MAX(CBalance)
```

```
FROM Customer;
```

```
SELECT MIN(CBalance), MAX(CBalance), AVG (CBalance)
```

```
FROM Customer
```

```
WHERE age > 35;
```


More extensions: aggregation

DISTINCT clause tells the operator to eliminate duplicates from the set *before* computing the function over the set

What is the difference between these two queries?

```
SELECT COUNT(Name)  
FROM Customer;
```

```
SELECT COUNT(DISTINCT Name)  
FROM Customer;
```

How many tuples are returned?



and even more!

- Clauses:
 - ORDER BY
 - GROUP BY
 - HAVING
 - LIMIT
- Operators:
 - Date/time
 - GIS (vendor-specific, e.g. PostGIS)

SQL: The Many Versions

- Watch out: database products may differ in which features of SQL they supports
- There are two major standards:
 - ANSI
 - SQL92 = SQL2
- SQL3: many new features
 - Recursion, triggers, objects
- Vendor-specific dialects
 - Include ANSI, and most of SQL2
 - Some of SQL3

Thank you!



Practice session Objectives

- 1) Learn to connect to a remote database through API (Carto, former CartoDB) with Python 2 / 3;
- 2) Learn to create SQL queries for filtering out data, calculating summary statistics, etc;
- 3) Develop intuition of SQL queries.



Task 1 – simple queries

- 1) Sort data by both: start_station_id ascending, and tripduration descending (hint: ASC, DESC);
- 2) Select last 10 records of the table. Hint: use the table's main id field;
- 3) List all unique birth years. Hint: distinct;
- 4) Find minimal, maximal and average trip duration
hint: min(), max(), avg().



Task 2 – date/time and conditions


1) Select only trips that started at 1 AM

hint: `EXTRACT(HOUR FROM fieldname::time);`

2) What is the average birth year of people that ride bikes at 2 AM?;

3) What is the age of the oldest person riding at 3 AM?

Hint: `age = 2018 - birth_year`. (For this task we assume they were all born on the same day).



Task 3 – aggregation

1) Find the “start_station_id” that had the highest number of bikes taken from it

hint: GROUP BY station id, COUNT();

2) Show top 3 “end_station_id” with the largest total “tripduration”. Hint: GROUP BY station id, SUM();

3) Find the “start_station_id” with the shortest average trip duration during 1 AM.

