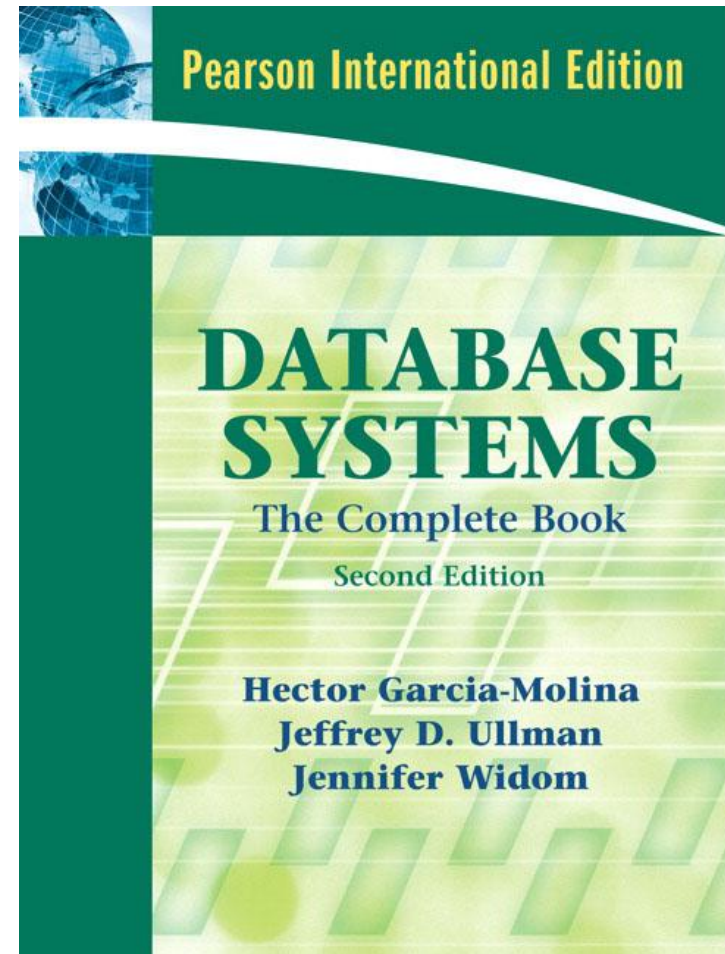


Database Management System I

Introduction to SQL

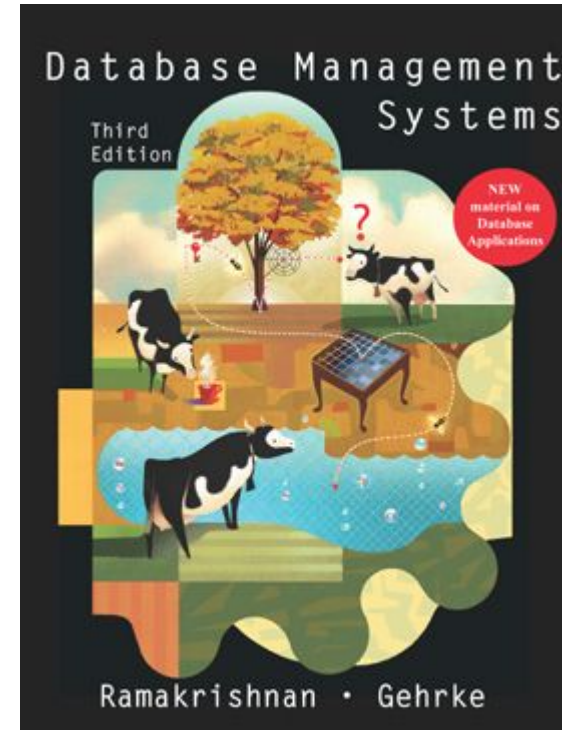
Main Textbook

- Database Systems:
The Complete Book
 - Hector Garcia-Molina
 - Jeffrey D. Ullman
 - Jennifer Widom



Alternative Textbook

- Database Management Systems
 - Raghu Ramakrishnan
 - Johannes Gehrke



Goals of Course

- To obtain a firm background in database systems, e.g.,
 - how to talk to database systems in a standard language
 - how to improve the efficiency of database systems
 - the theories behind database design and some algorithms behind database implementation
- Mostly “basic stuff” about databases

What will NOT be taught

- Advanced database technologies
 - Geographical information systems
 - Data mining
 - ...
 - (This is an **introductory course** only)
- Specific instructions on how to install and use a specific database system on a specific platform
 - Try the user manual or Google

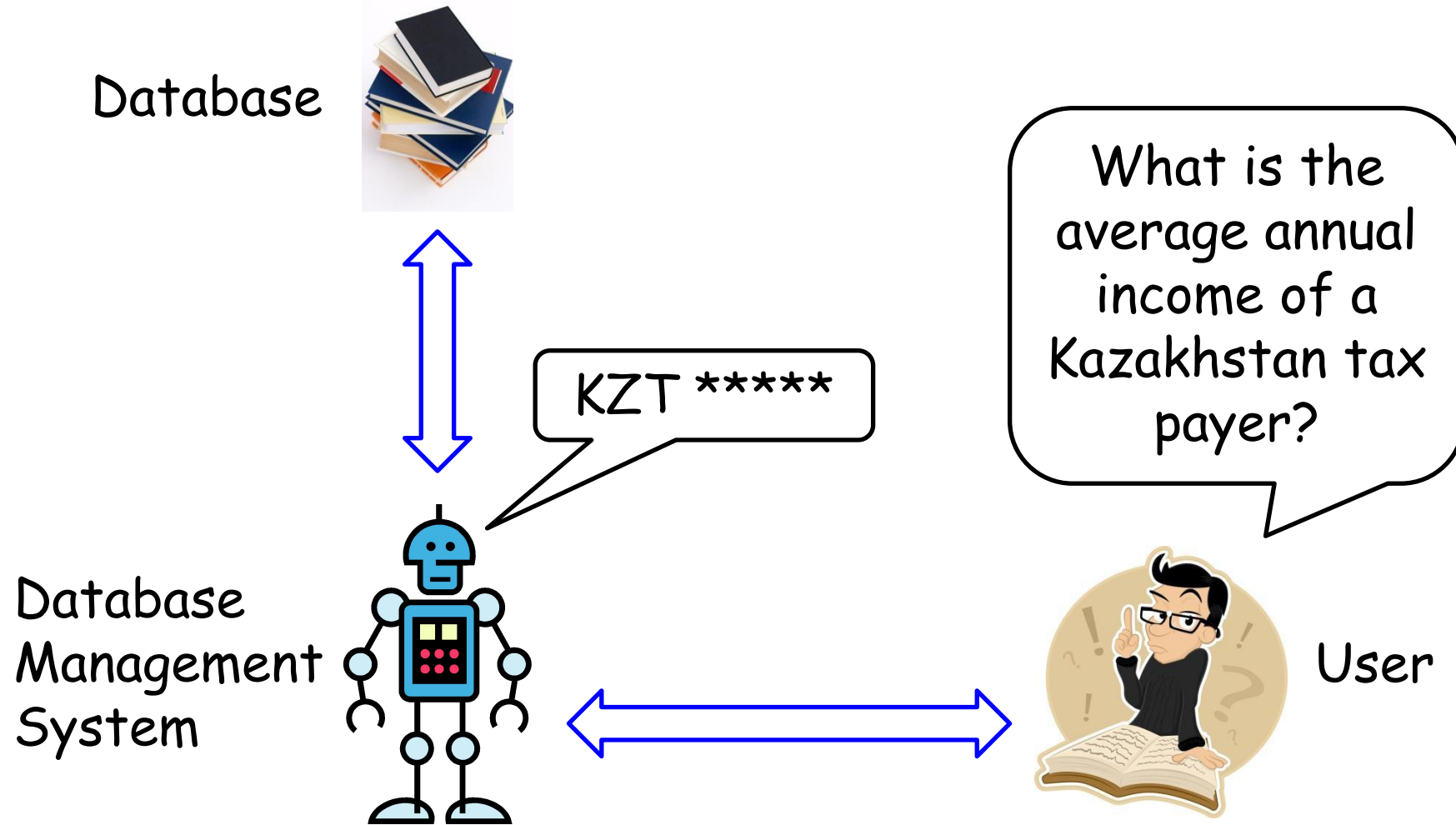
Teaching Style

- There will be **a lot of** in-lecture exercises
- Questions will be welcomed
- Lecture notes will be released on the Drive (at least several days before lectures)

Course Overview

- What is a database?
- A large collection of data organized especially for rapid search and retrieval (as by a computer)
- What is a database system?
(more formally, a database management system, i.e., DBMS)
- A management system that helps us retrieve information from databases

Database and DBMS



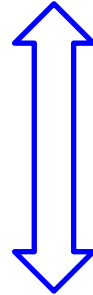
Tables, Relations, Relational Model

Database

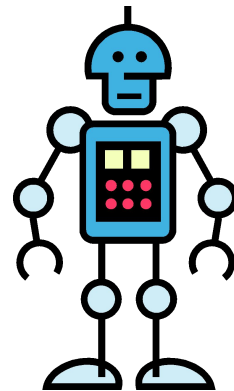


Taxpayer_ID	Annual_Income
51248297	100000
33891634	50000
...	...

Income_Table

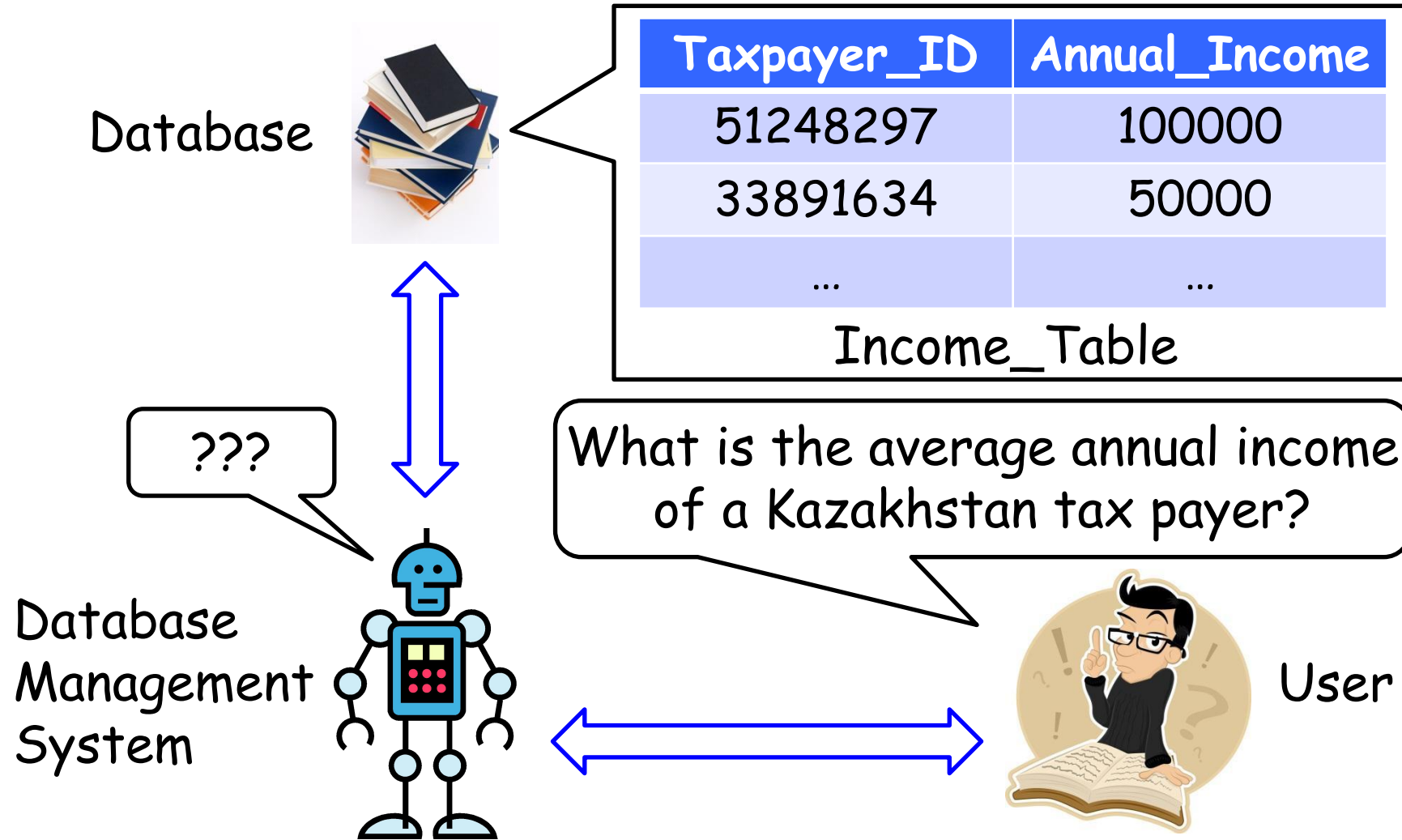


Database
Management
System

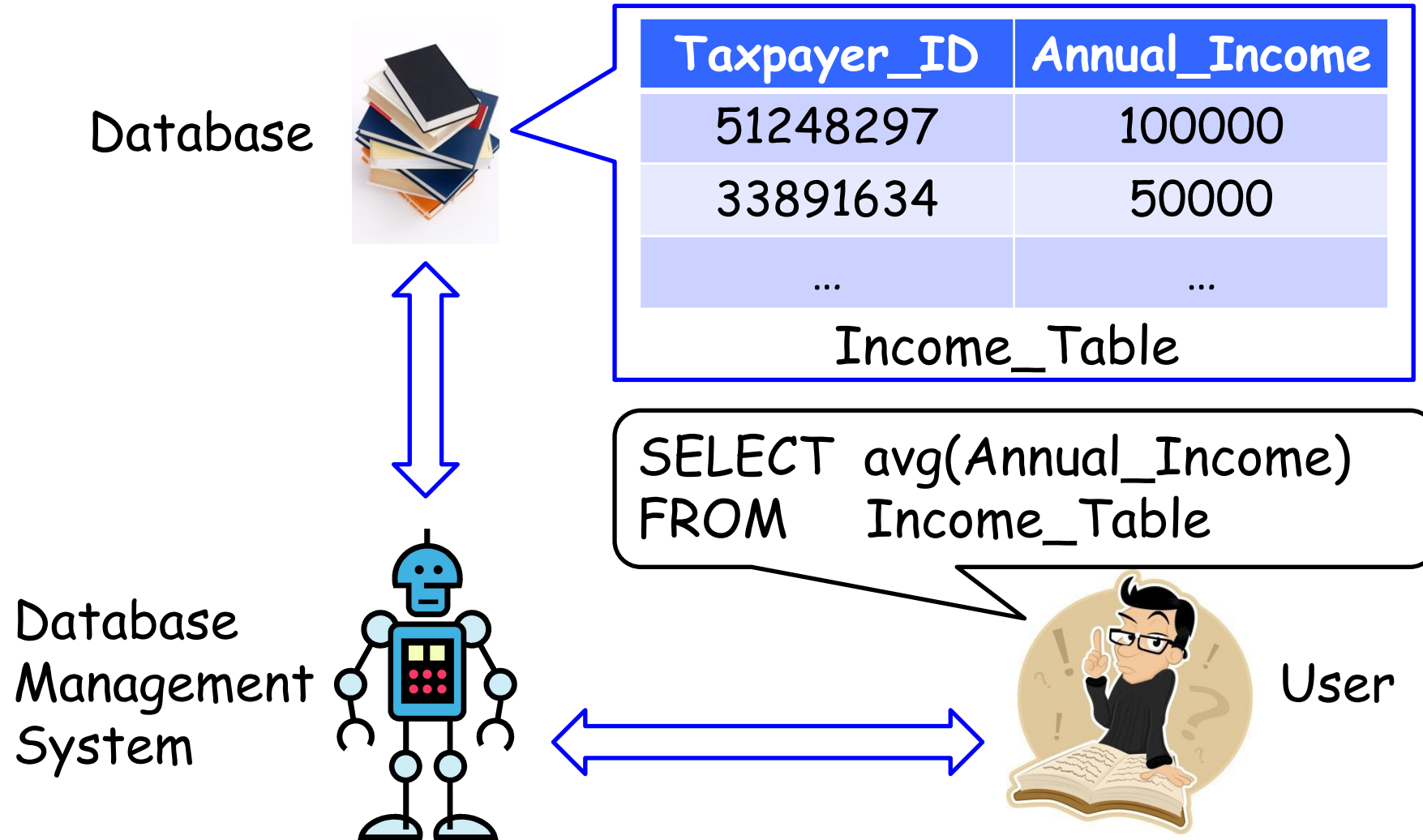


User

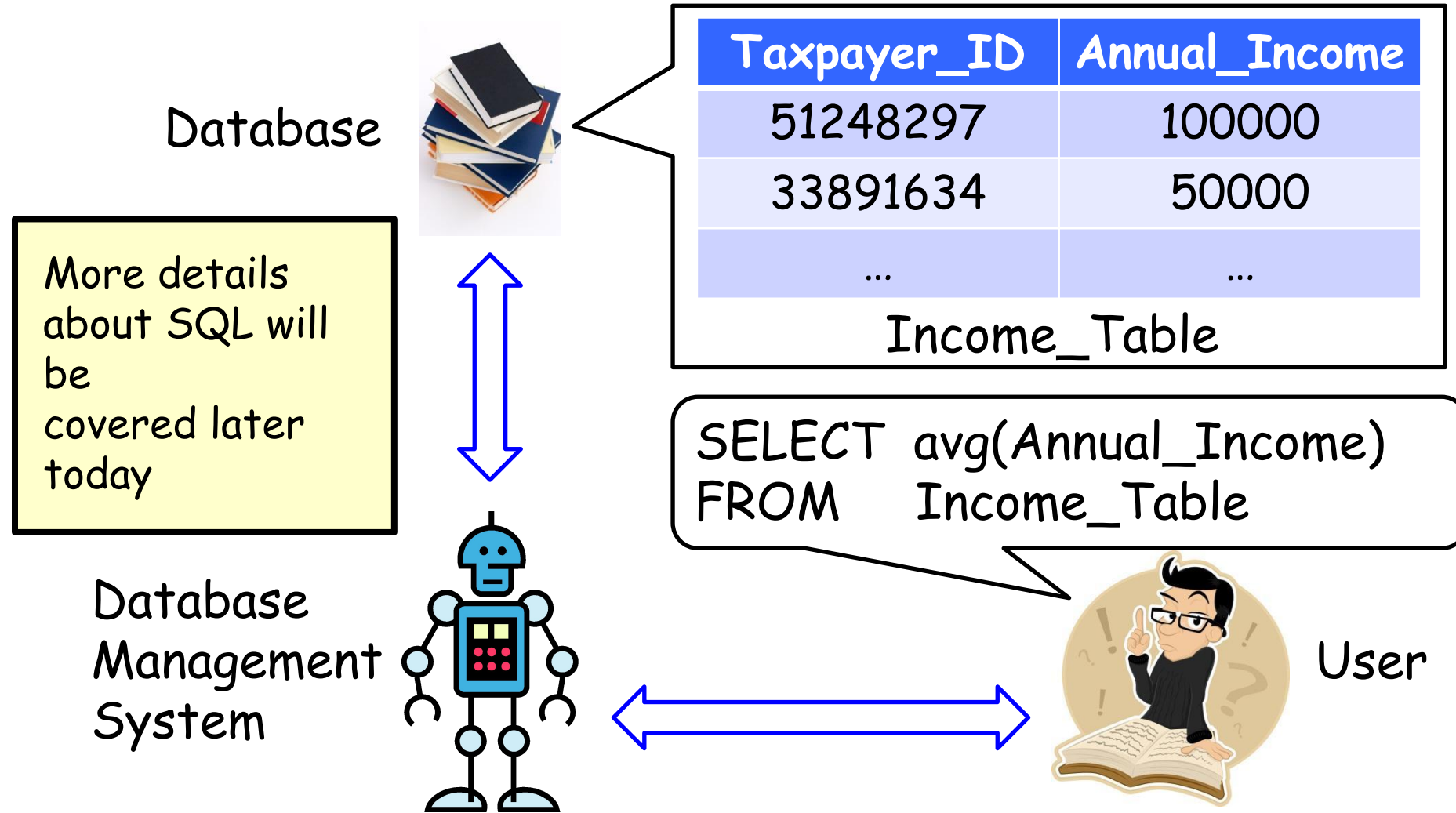
Tables, Relations, Relational Model



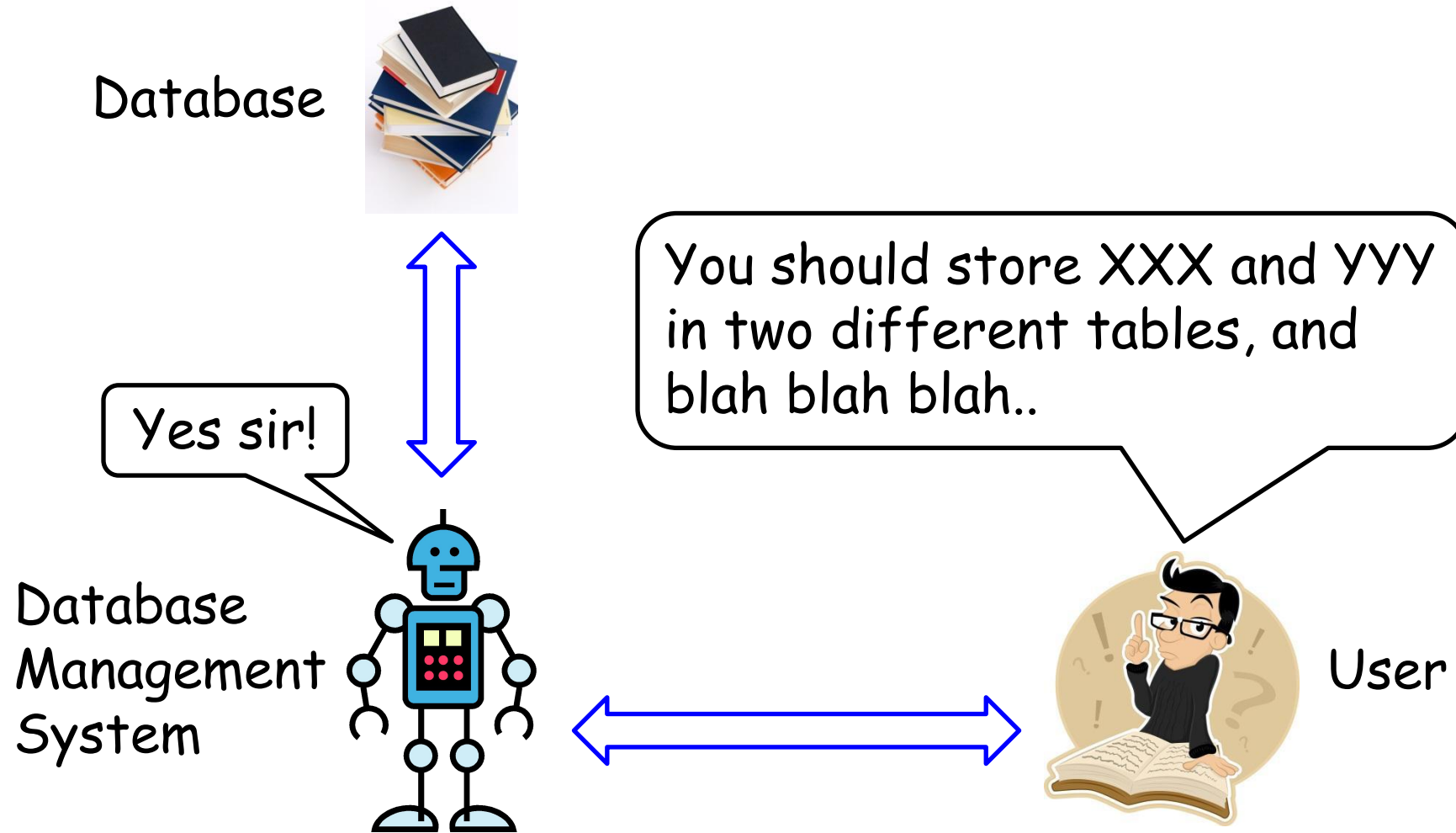
Structured Query Language (SQL)



Structured Query Language (SQL)



Database Schema Design



Database Schema Design

Taxpayer_ID	Annual_Income
51248297	100000
33891634	50000
67904777	70000
...	...

- Assume that we want to capture parent-child relationships

Database Schema Design

Taxpayer_ID	Annual_Income	Child_ID
51248297	100000	
33891634	50000	
67904777	70000	
...	...	

- Is one column enough?

Database Schema Design

Taxpayer_ID	Annual_Income	Child_ID1	Child_ID2
51248297	100000		
33891634	50000		
67904777	70000		
...	...		

- Are two columns enough?
- Assume that two columns are enough
- Does everyone have two children?

Schema designs based on the *Entity-Relationship model*

Course Content

- SQL
- Constraints and Triggers
- Conceptual Design
- Indices
- Relation Algebra
- Query Processing/Optimization
- Concurrency Control
- Recovery
- Current trend (e.g., NOSQL)

**Database
Design**

**Database
Implementation**

What do you want from a DBMS?

Why do we need it?

- Keep data around (persistent)
- Answer queries (questions) about data
- Update data

- Requirements from high-end applications
 - Massive amounts of data (terabytes ~ petabytes)
 - High throughput (thousands ~ millions transactions/min)

The Relational Revolution



The Relational Revolution (1970's)

- IBM and Univ of Berkeley
- A simple data model: Data is stored in **relations** (tables)
- A **declarative** query language: SQL
 - Programmer specifies what answers a query should return, but not how the query is executed
 - DBMS picks the best execution strategy
- **Hide** the physical organization of the database from applications
 - Provided only **logical** view of the data

Turing Award!

Edgar C Codd

- Relational model is the dominating technology today
- Graphs/Streams/Arrays are hot wanna-be!

**“Relational databases are the
foundation of western civilization.”**



Bruce Lindsay
IBM Fellow
IBM Almaden Research Center

- Structured Query Language (SQL)

Structured Query Language (SQL)

- A declarative (computer) language for managing data in a **relational** database management system
- Two parts
 - Data Definition Language (DDL)
 - Create/Alter/Delete tables
 - Will be discussed in the next week
 - Data Manipulation Language (DML)
 - Query one or more tables
 - Insert/Delete/Modify tuples in tables
 - Will be discussed in the following

Tables in SQL

Table name

Product

Key

Attribute name

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

Tuple (i.e. row, record)

Data Types in SQL

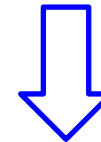
- Character strings
 - CHAR(20)
 - VARCHAR(50)
 - ...
- Numbers
 - INT
 - FLOAT
 - ...
- Others
 - BOOLEAN
 - DATETIME
 - ...

Product		
<u>PName</u>	Price	Category
iPhone 4	888	Phone
iPad 2	668	Tablet
Milestone	798	Phone
EOS 550D	1199	Camera

Simple SQL Query

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE Category = 'Phone'
```



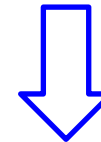
"selection"

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
Milestone	798	Phone	Motorola

Simple SQL Query

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE Category <> 'Phone'
```

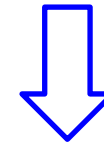


<u>PName</u>	Price	Category	Manufacturer
iPad 2	668	Tablet	Apple
EOS 550D	1199	Camera	Canon

Simple SQL Query

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE Category = 'Phone' AND Price > 800
```

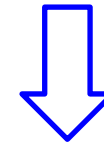


<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

Simple SQL Query

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE Category = 'Tablet' OR Price > 1000
```



<u>PName</u>	Price	Category	Manufacturer
iPad 2	668	Tablet	Apple
EOS 550D	1199	Camera	Canon

Simple SQL Query (cont.)

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 800
```

↓
"selection and projection"

<u>PName</u>	Price	Manufacturer
iPhone 4	888	Apple
EOS 550D	1199	Canon

Details

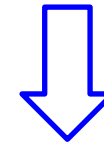
- SQL is NOT case sensitive (when it comes to keywords and names)
 - SELECT = Select = select
 - Product = product

- Constants must use single quotes
 - 'abc' - OK
 - "abc" - NOT OK

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE PName LIKE 'iPh%'
```



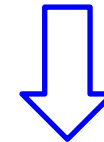
<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

% stands for "any string"

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE PName LIKE '%Ph%'
```



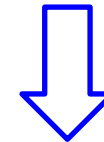
<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

% stands for "any string"

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE PName LIKE '%P%e%'
```



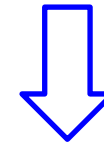
<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

% stands for "any string"

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE PName LIKE '_Phone 4'
```



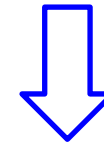
<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

_ stands for "any character"

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT *
FROM Product
WHERE PName LIKE '_Phone__'
```



<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple

_ stands for "any character"

Patterns for Strings

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```

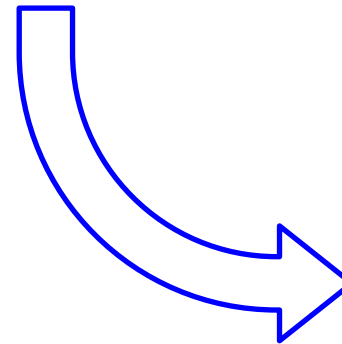
SELECT *
FROM Product
WHERE PName NOT LIKE '_Phone__'

```

Eliminating Duplicates

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT Category
FROM Product
```

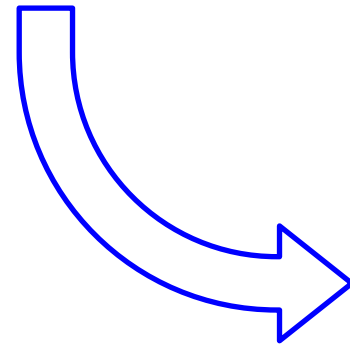


Category
Phone
Tablet
Phone
Camera

Eliminating Duplicates (cont.)

Product	PName	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category
FROM Product
```

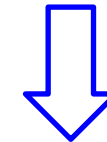


Category
Phone
Tablet
Camera

Ordering the Results

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT PName, Price
FROM Product
WHERE Price < 800
ORDER BY PName
```

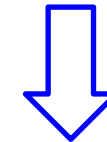


<u>PName</u>	Price
Milestone	798
iPad 2	668

Ordering the Results (cont.)

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT PName, Price
FROM Product
WHERE Price < 800
ORDER BY PName DESC
```

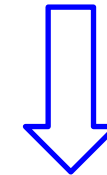


<u>PName</u>	Price
iPad 2	668
Milestone	798

Ordering the Results (cont.)

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT PName, Category
FROM Product
WHERE Price < 1000
ORDER BY Category, PName
```

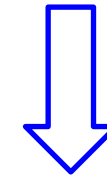


<u>PName</u>	Category
Milestone	Phone
iPhone 4	Phone
iPad 2	Tablet

Ordering the Results (cont.)

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```
SELECT PName, Category
FROM Product
WHERE Price < 1000
ORDER BY Category DESC,
PName
```



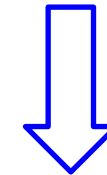
<u>PName</u>	Category
iPad 2	Tablet
Milestone	Phone
iPhone 4	Phone

Ordering the Results (cont.)

Product	<u>PName</u>	Price	Category	Manufacturer
	iPhone 4	888	Phone	Apple
	iPad 2	668	Tablet	Apple
	Milestone	798	Phone	Motorola
	EOS 550D	1199	Camera	Canon

```

SELECT PName, Category
FROM Product
WHERE Price < 1000
ORDER BY Category DESC,
         PName DESC
    
```



<u>PName</u>	Category
iPad 2	Tablet
iPhone 4	Phone
Milestone	Phone

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category
```

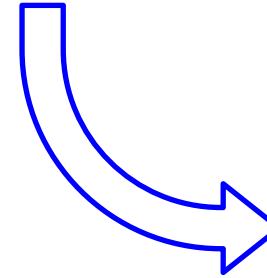
?

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY Category
```



Category
Camera
Phone
Tablet

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category
FROM Product
ORDER BY Category
WHERE Price < 1000
```

?

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category
FROM Product
ORDER BY Category
WHERE Price < 1000
```

Error!

- "WHERE" should always proceed "ORDER BY"

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY PName
```



Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

```
SELECT DISTINCT Category  
FROM Product  
ORDER BY PName
```

Error!

- "ORDER BY" items must appear in the select list if "SELECT DISTINCT" is specified

Joins

Company

<u>CName</u>	StockPrice	Country
Canon	45	Japan
Motorola	40	USA
Apple	374	USA

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

- A user wants to know the names and prices of all products by Japan companies. How?

Joins

Company

<u>CName</u>	StockPrice	Country
Canon	45	Japan
Motorola	40	USA
Apple	374	USA

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

- ```
SELECT PName, Price
FROM Product, Company
WHERE Country = 'Japan'
AND Manufacturer = CName
```

# Joins

Person

| <u>PName</u> | Address | WorksFor |
|--------------|---------|----------|
| ...          | ...     | ...      |

Company

| <u>CName</u> | Address | Country |
|--------------|---------|---------|
| ...          | ...     |         |

- Find the names of the persons who work for companies in USA
- ```
SELECT PName
FROM Person, Company
WHERE Country = 'USA'
AND WorksFor = CName
```

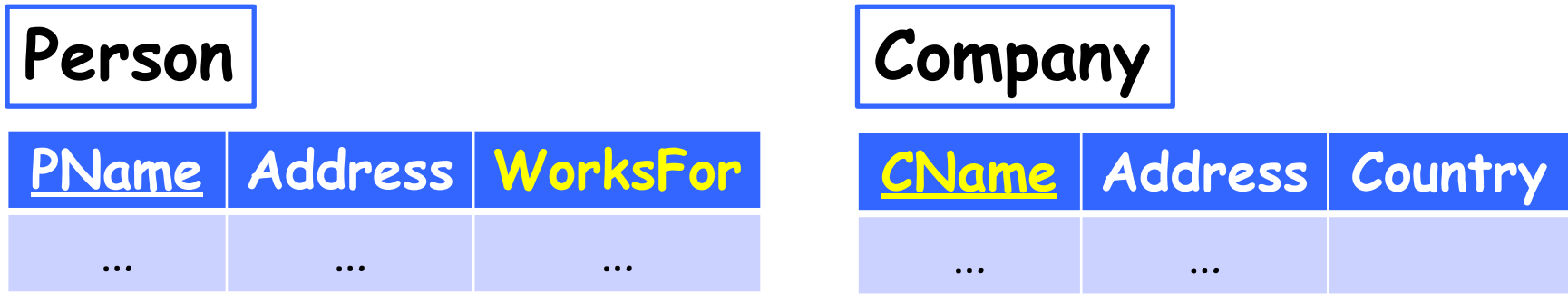
Joins

Person			Company		
<u>PName</u>	Address	WorksFor	<u>CName</u>	Address	Country
...	

- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Address
FROM Person, Company
WHERE Country = 'USA'
AND WorksFor = CName
```

**Error!**

# Joins



- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address
FROM Person, Company
WHERE Country = 'USA'
AND WorksFor = CName
```

Joins

Person

<u>PName</u>	Address	CName
...

Company

<u>CName</u>	Address	Country
...	...	

- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address
FROM Person, Company
WHERE Country = 'USA'
AND CName = CName
```

**Error!**

# Joins

## Person

| <u>PName</u> | Address | CName |
|--------------|---------|-------|
| ...          | ...     | ...   |

## Company

| <u>CName</u> | Address | Country |
|--------------|---------|---------|
| ...          | ...     |         |

- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT PName, Company.Address  
FROM Person, Company  
WHERE Country = 'USA'  
AND Person.CName = Company.CName
```


Joins

Person			Company		
<u>PName</u>	Address	CName	<u>CName</u>	Address	Country
...	

- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT X.PName, Y.Address
FROM Person AS X, Company AS Y
WHERE Y.Country = 'USA'
 AND X.CName = Y.CName
```

# Joins

## Person

| <u>PName</u> | Address | CName |
|--------------|---------|-------|
| ...          | ...     | ...   |

## Company

| <u>CName</u> | Address | Country |
|--------------|---------|---------|
| ...          | ...     |         |

- Find the names the persons who work for companies in USA, as well as their company addresses
- ```
SELECT X.PName, Y.Address
FROM   Person X, Company Y
WHERE  Y.Country = 'USA'
       AND X.CName = Y.CName
```

Exercise

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	Manufacturer

- Exercise: Find the names of the companies in China that produce products in the 'tablet' category
- ```
SELECT DISTINCT CName
FROM Company, Product
WHERE Manufacturer = CName
AND Country = 'China'
AND Category = 'Tablet'
```

# Exercise

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | Manufacturer |
|---------|--------------|-------|----------|--------------|
|         | ...          | ...   | ...      | ...          |

- Exercise: Find the names of the companies in China that produce products in the 'tablet' or 'phone' category
- ```
SELECT DISTINCT CName
FROM Company, Product
WHERE Manufacturer = CName
AND Country = 'China'
AND (Category = 'Tablet'
OR Category = 'Phone')
```

Exercise

Product

<u>PName</u>	Price	Category	Manufacturer
iPhone 4	888	Phone	Apple
iPad 2	668	Tablet	Apple
Milestone	798	Phone	Motorola
EOS 550D	1199	Camera	Canon

- Exercise: Find the manufacturers that produce products in both the 'tablet' and 'phone' categories
- ```
SELECT DISTINCT Manufacturer
FROM Product
WHERE Category = 'Tablet'
AND Category = 'Phone'
```

**Error!**

# Exercise

## Product

| <u>PName</u> | Price | Category | Manufacturer |
|--------------|-------|----------|--------------|
| iPhone 4     | 888   | Phone    | Apple        |
| iPad 2       | 668   | Tablet   | Apple        |
| Milestone    | 798   | Phone    | Motorola     |
| EOS 550D     | 1199  | Camera   | Canon        |

- Exercise: Find the manufacturers that produce products in both the 'tablet' and 'phone' categories
- ```
SELECT DISTINCT X.Manufacturer
FROM Product AS X, Product AS Y
WHERE X.Manufacturer = Y.Manufacturer
AND X.Category = 'Tablet'
AND Y.Category = 'Phone'
```

Subqueries

- A subquery is a SQL query nested inside a larger query
- Queries with subqueries are referred to as **nested queries**
- A subquery may occur in
 - SELECT
 - FROM
 - WHERE

SQL subquery

SQL subquery

A special subquery: Scalar Subquery

Scalar Subquery

- return a **single value** which is then used in a comparison.
- If query is written so that it expects a subquery to return a single value, and it returns multiple values or no values, a **run-time error** occurs.

Example Query

From **Sells**(bar, beer, price), find the bars that serve **Heineken** for the same price **Ku De Ta** charges for **Bud**.

Example Scalar Subquery

Find the price **Ku De Ta** charges for **Bud**.

Sells

Bar	Beer	Price
Clinic	Heineken	8.00
Clinic	Bud	6.60
Ku De Ta	Bud	7.90
MOS	Heineken	7.90
Ku De Ta	Heineken	8.00

```
SELECT price
FROM Sells
WHERE bar = 'Ku De Ta'
AND beer = 'Bud';
```



Find the bars that serve **Heineken** at that price.

```
SELECT bar
FROM Sells
WHERE beer = 'Heineken'
AND price = 7.90;
```



Example Scalar Subquery

```
SELECT bar
FROM Sells
WHERE beer = 'Heineken' AND
      price = (SELECT price
               FROM Sells
               WHERE bar = 'Ku De Ta'
               AND beer = 'Bud' );
```

Subqueries in FROM

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	CName

- Find all products in the 'phone' category with prices under 1000
- ```

SELECT X.PName
FROM (SELECT *
 FROM Product
 WHERE category = 'Phone') AS X
WHERE X.Price < 1000

```

# Subqueries in FROM (cont.)

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | CName |
|---------|--------------|-------|----------|-------|
|         | ...          | ...   | ...      | ...   |

- Find all products in the 'phone' category with prices under 1000
- `SELECT PName  
FROM Product  
WHERE Category = 'Phone'  
AND Price < 1000`
- This is a much more efficient solution

# Subqueries in WHERE (cont.)

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | CName |
|---------|--------------|-------|----------|-------|
|         | ...          | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```

SELECT DISTINCT CName
FROM Company AS X
WHERE X.CName IN
    (SELECT Y.CName
     FROM Product AS Y
     WHERE Y.Price < 100)

```

Subqueries in WHERE (cont.)

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	CName

- Find all companies that make some products with price < 100
- ```
SELECT DISTINCT CName
FROM Company AS X
WHERE EXISTS
 (SELECT * FROM Product AS Y
 WHERE X.CName = Y.Cname
 AND Y.Price < 100)
```
- A nested query is **correlated** with the outer query if it contains a reference to an attribute in the outer query.
- A nested query is **correlated** with the outside query if it must be **re-computed for every tuple** produced by the outside query.

# Subqueries in WHERE (cont.)

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | CName |
|---------|--------------|-------|----------|-------|
|         | ...          | ...   | ...      | ...   |

- Find all companies that make some products with price < 100
- ```

SELECT DISTINCT CName
FROM Company AS X
WHERE X.CName IN
      (SELECT *
       FROM Product AS Y
       WHERE Y.Price < 100)
    
```

Error!

- The number of attributes in the **SELECT** clause in the subquery must match the number of attributes compared to with the comparison operator.

Subqueries in WHERE (cont.)

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	CName

- Find all companies that make some products with price < 100
- `SELECT DISTINCT CName
FROM Company AS X
WHERE 100 > ANY
 (SELECT Price FROM Product AS Y
 WHERE X.CName = Y.Cname)`

Subqueries in WHERE (cont.)

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	CName

- Find all companies that make some products with price < 100
- `SELECT DISTINCT CName
FROM Product
WHERE Price < 100`
- This is more efficient than the previous solutions

Operators in Subqueries

IN

$\langle \text{tuple} \rangle \text{ IN } \langle \text{relation} \rangle$ is true if and only if the tuple is a member of the relation.

ANY

$x = \text{ANY}(\langle \text{relation} \rangle)$ is a boolean cond. meaning that x equals at least one tuple in the relation.

EXISTS

- $\text{EXISTS}(\langle \text{relation} \rangle)$ is true if and only if the $\langle \text{relation} \rangle$ is not empty.
- Returns true if the nested query has 1 or more tuples.

ALL

$x \neq \text{ALL}(\langle \text{relation} \rangle)$ is true if and only if for every tuple t in the relation, x is not equal to t .

Note

The keyword **NOT** can proceed any of the operators (s NOT IN R)

Avoiding Nested Queries

- In general, nested queries tend to be more inefficient than un-nested queries
 - query optimizers of DBMS **do not generally do a good job** at optimizing queries containing subqueries
- Therefore, they should be avoided whenever possible
- But there are cases where avoiding nested queries is hard...

Subqueries in WHERE (cont.)

Company	<u>CName</u>	StockPrice	Country

Product	<u>PName</u>	Price	Category	CName

- Find all companies that do not make any product with price < 100
- ```

SELECT DISTINCT CName
FROM Company AS X
WHERE NOT EXISTS
 (SELECT * FROM Product AS Y
 WHERE X.CName = Y.Cname
 AND Y.Price < 100)

```

# Subqueries in WHERE (cont.)

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | CName |
|---------|--------------|-------|----------|-------|
|         | ...          | ...   | ...      | ...   |

- Find all companies that do not make any product with price < 100
- `SELECT DISTINCT CName  
FROM Company AS X  
WHERE 100 <= ALL  
    (SELECT Price FROM Product AS Y  
      WHERE X.CName = Y.Cname)`

# Subqueries in WHERE (cont.)

| Company | <u>CName</u> | StockPrice | Country |
|---------|--------------|------------|---------|
|         | ...          | ...        | ...     |

| Product | <u>PName</u> | Price | Category | CName |
|---------|--------------|-------|----------|-------|
|         | ...          | ...   | ...      | ...   |

- Find all companies that does not make any products with price < 100
- `SELECT DISTINCT CName  
FROM Company AS X  
WHERE 100 <= ALL  
    (SELECT Price FROM Product AS Y  
      WHERE X.CName = Y.Cname)`

# Exercise

Likes

| Drinker | Beer |
|---------|------|
| ...     | ...  |

Frequent

| Drinker | Bar |
|---------|-----|
| ...     | ... |

Serves

| Bar | Beer |
|-----|------|
| ... | ...  |

- Find all drinkers that frequent some bar that serves some beer they like
- ```
SELECT DISTINCT F.Drinker
FROM   Likes AS L, Frequent AS F,
       Serve AS S
WHERE  L.Drinker = F.Drinker
       AND F.Bar = S.Bar
       AND L.Beer = S.Beer
```

Exercise

Likes

Drinker	Beer
...	...

Frequent

Drinker	Bar
...	...

Serves

Bar	Beer
...	...

- Find all drinkers that frequent some bar that does not serve any beer they like
- ```
SELECT DISTINCT F.Drinker
FROM Frequent AS F, Serves AS S
WHERE F.Bar = S.Bar AND NOT EXIST
 (SELECT *
 FROM Likes as L
 WHERE L.Beer = S.Beer
 AND L.Drinker = F.Drinker)
```



# Exercise

Likes

| Drinker | Beer |
|---------|------|
| ...     | ...  |

Frequent

| Drinker | Bar |
|---------|-----|
| ...     | ... |

Serves

| Bar | Beer |
|-----|------|
| ... | ...  |

- Find all drinkers that do not frequent any bar that serve some beer they like
- ```
SELECT DISTINCT F.Drinker
FROM   Frequent AS F
WHERE  NOT EXIST
      (SELECT *
       FROM Likes AS L, Serves AS S
       WHERE L.Beer = S.Beer
            AND L.Drinker = F.Drinker
            AND S.Bar = F.Bar)
```

Roadmap --SQL

- Table
- SELECT
FROM
WHERE
- ORDER BY
- Joins
- Subqueries
- Aggregations
- UNION, INTERSECT, EXCEPT
- NULL
- Outerjoin
- Insert/Delete tuples
- Create/Alter/Delete tables
- View

Exercise

Likes

Drinker	Beer
John	A2

Frequent

Drinker	Bar
John	B1

Serves

Bar	Beer
B1	A1

- Find all drinkers that frequent some bar that does not serve any beer they like
- ```

SELECT DISTINCT F.Drinker
FROM Frequent AS F
WHERE NOT EXIST
 (SELECT *
 FROM Serves as S, Likes as L
 WHERE L.Beer = S.Beer
 AND L.Drinker = F.Drinker
 AND F.Bar = S.Bar)

```

# Exercise

Likes

| Drinker | Beer |
|---------|------|
| ...     | ...  |

Frequent

| Drinker | Bar |
|---------|-----|
| ...     | ... |

Serves

| Bar | Beer |
|-----|------|
| ... | ...  |

- Find all drinkers that do not frequent any bar that serve some beer they like
- ```
SELECT DISTINCT F.Drinker
FROM   Frequent AS F
WHERE  NOT EXIST
      (SELECT *
       FROM   Likes AS L, Serves AS S
       WHERE  L.Beer = S.Beer
              AND L.Drinker = F.Drinker
              AND S.Bar = F.Bar)
```