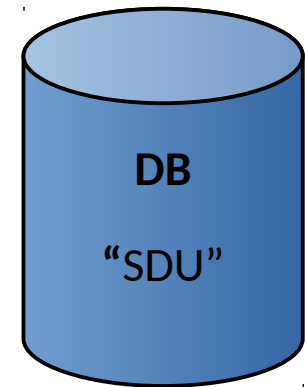


DM534: Introduction to Relational Databases

**10/10/2017
Christian Wiwie**

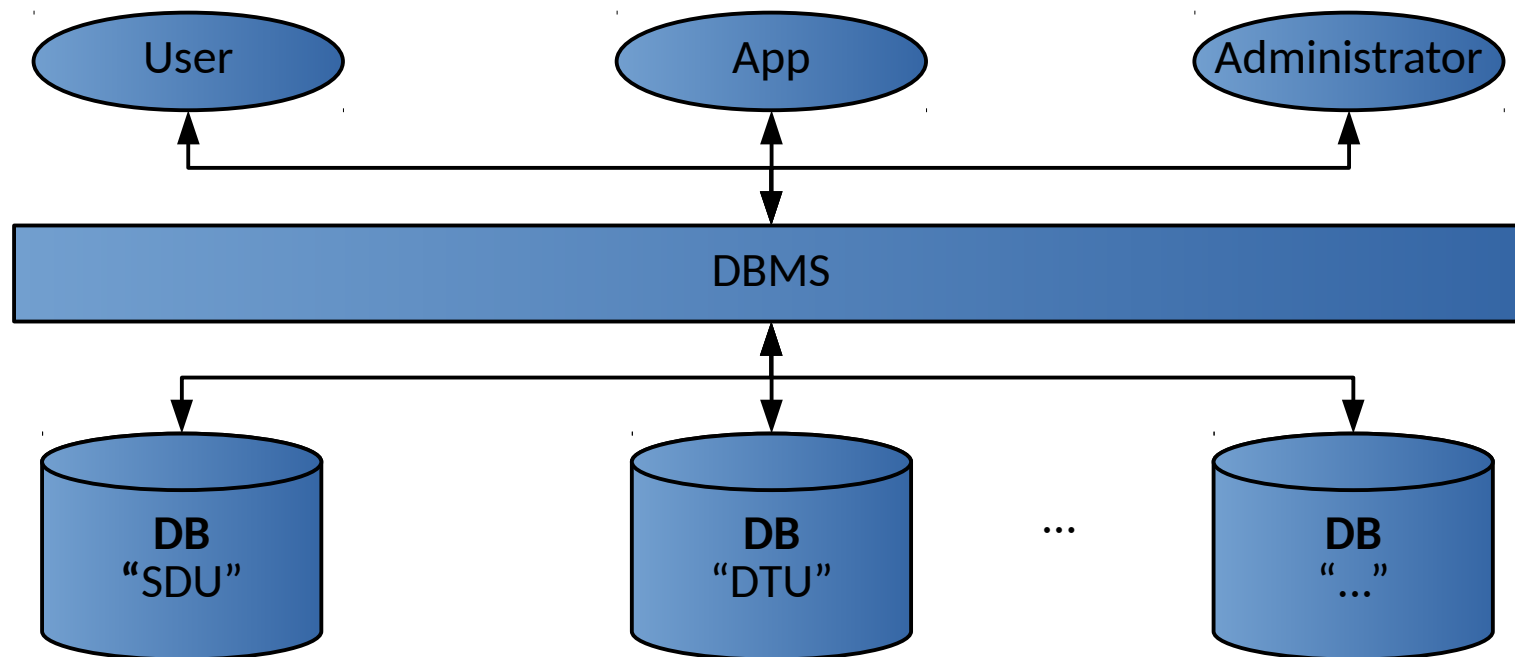
What are Databases?

- Repository for large data amounts
- Describes a logical structure of contained data
- Guarantees data integrity by enforcing constraints
- Allows for efficient access
- Consistent and safe storage



Database Management Systems (DBMS)

- Software that manages databases
- Access to database only via DBMS



Where are Databases used?

- Wherever large amounts of data to be managed
- Examples:
 - Corporate data: payrolls, inventory, sales, customers, ...
 - Banking systems, stock exchanges
 - Airline systems
 - Web search: Google, Live, Yahoo, ...
 - Social networks: Facebook, Twitter, ...
 - Blogs, discussion forums
 - Scientific and medical databases
 - ...

Features of a modern DBMS

- Highly **efficient access** to stored data using Indexes
- Backup/log mechanisms ensure **data safety**
- Security policies to manage access **permissions**
- Data **consistency**: Can enforce complex data constraints, including dependencies
- Flexible **searching, sorting, filtering**
- Ensures all the above, even with parallel multi-user access

Databases vs. storage in files

- File storage does not provide most of these features
 - Need to be realized for each file storage structure individually
 - Highly sophisticated functionalities, not trivial to do right

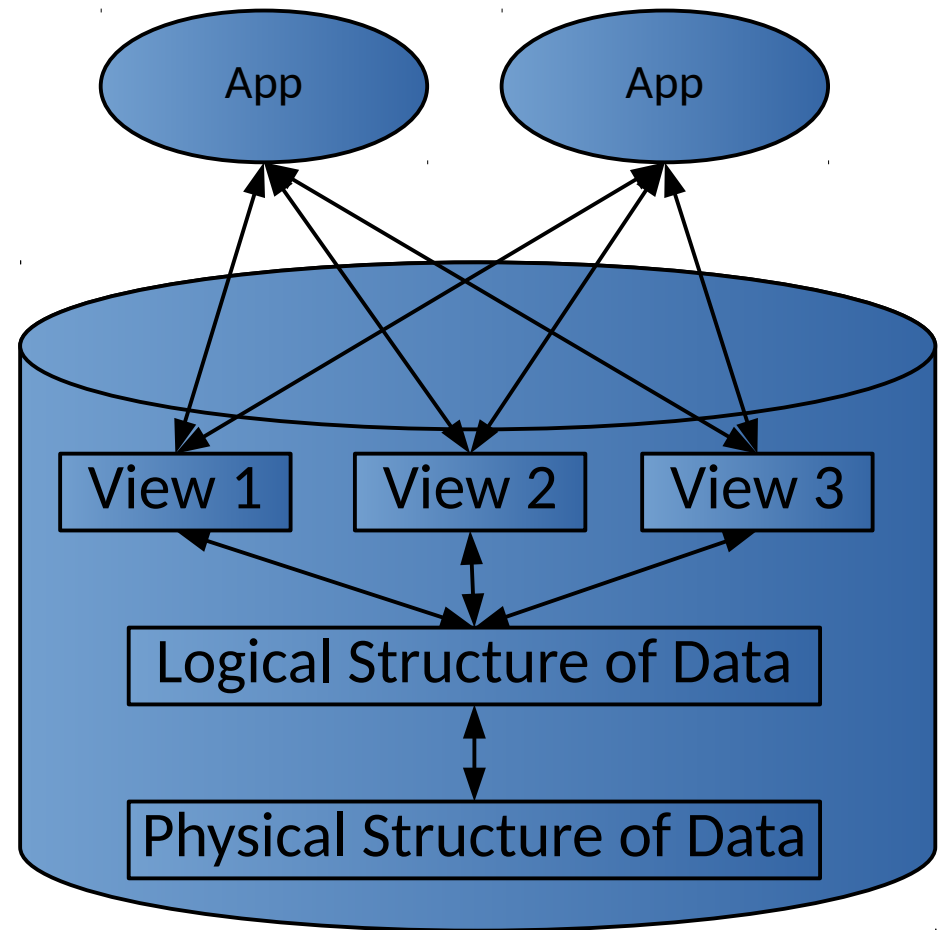
Kinds of DBMS / databases

- Data is modeled and organized differently
- Optimized for specific kinds of operations
- **Relational DBMS (RDBMS) / databases** the most widespread by far
 - Based on mathematical relations
 - Basically, a database is a collection of relations
 - e.g. MySQL, PostgreSQL, ...
- **Graph DBMS / databases**
 - Data is a network, with entities and connections between them
 - e.g. neo4j

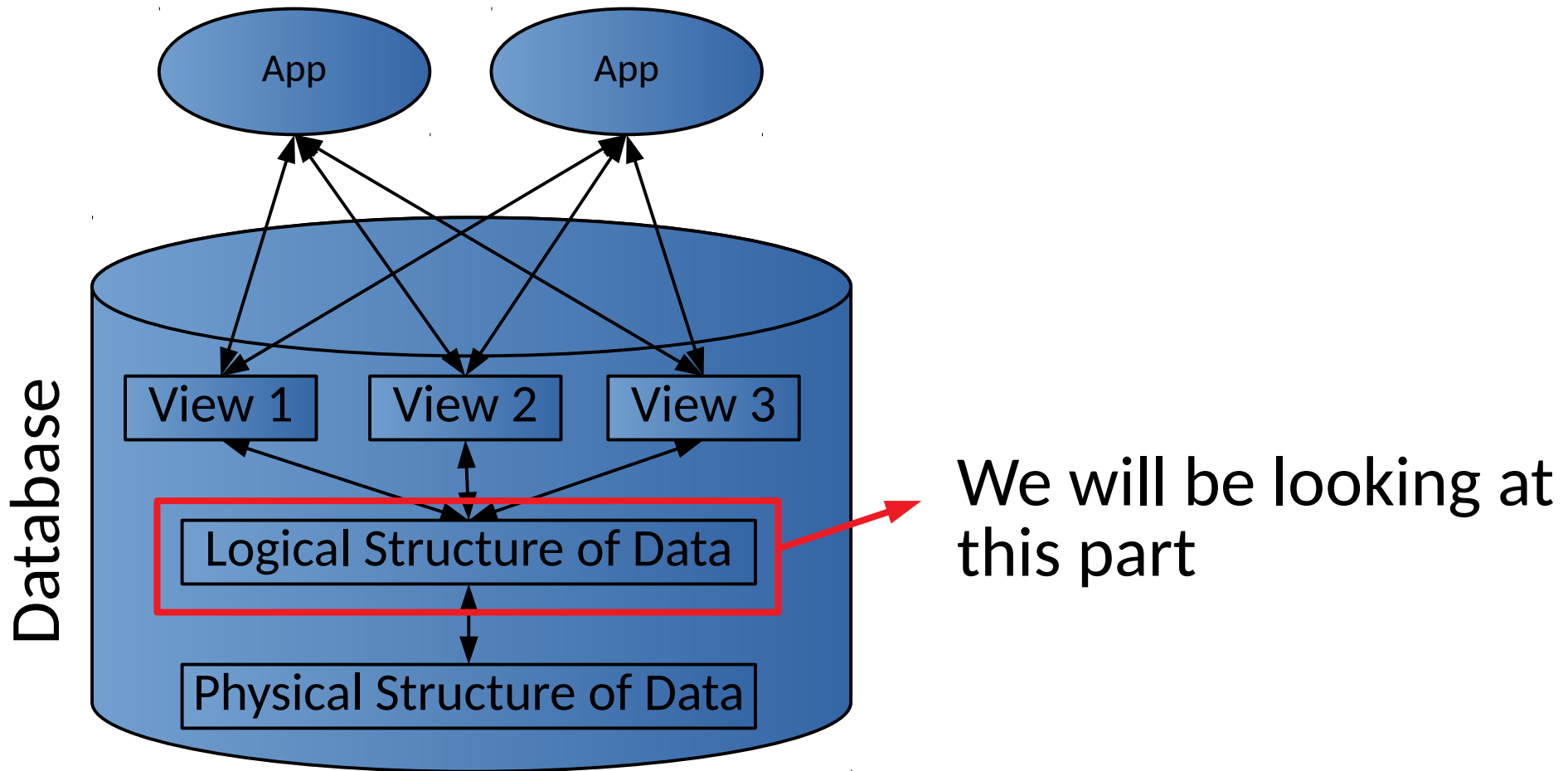
Internal Structure of a Database

- Databases are often tightly integrated with software
- Should be independent from how data is structured and stored
- First level independence
 - Independent of changes in logical structure of data
- Second level independence
 - Independent of changes in physical structure of data

Database



Internal Structure of a Database

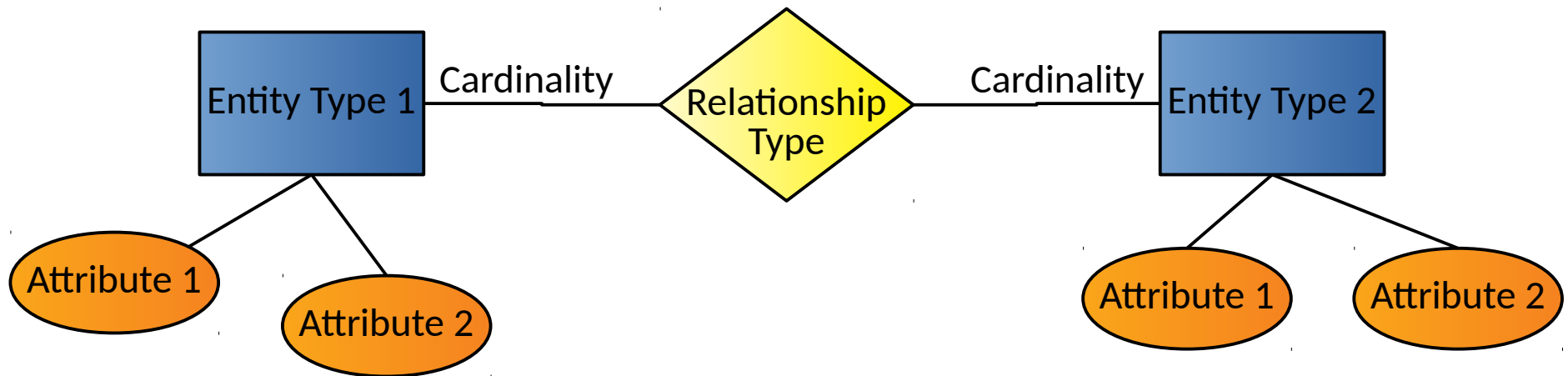


Data Model

- Conceptual description of
 - entity type for which data should be stored
 - relationship types between entity types
- Specific to kind of DBMS
 - Relational databases are based on relational data model
 - For graph databases: the graph itself

Relational Data Model

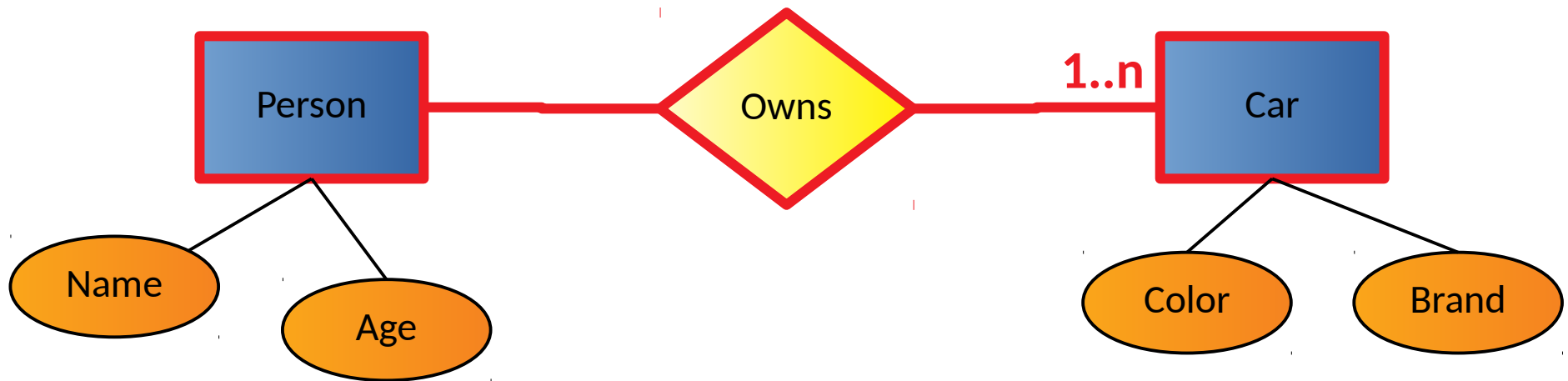
- Visualized with Entity-Relationship (ER) diagrams:



- Cardinality: How many entities are involved in a relationship?

Relational Data Model

- Example Cardinalities

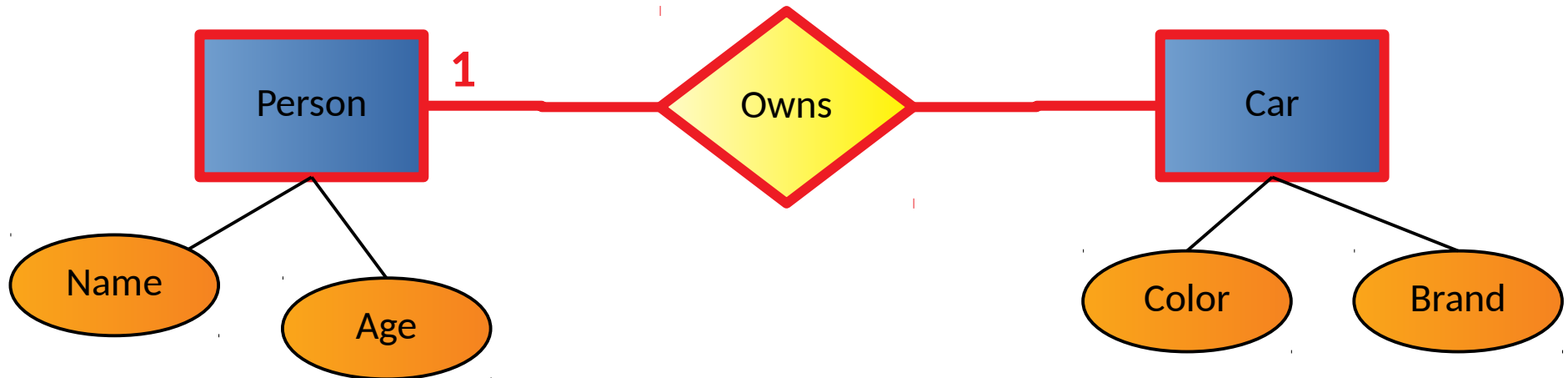


- Read:

- **One person owns one or more cars**

Relational Data Model

- Example Cardinalities



- Read:

- **One car is owned by exactly one person**

- Constraints do not necessarily hold in reality (joint ownership)

Relational Data Model

- Main concept: **relation schemas**
- relation != relationship
 - Relations derived from entities and relationships
- A **relation schema** is defined by
 - a name
 - and a set of attribute names
 - Optionally: attribute types

relation_name(attribute₁, attribute₂, ...) or

relation_name(attribute₁: type₁, attribute₂: type₂, ...)

Relation Schemas

- A **relation schema** usually corresponds to
 - Real world entity types (e.g. car, person, ...)
 - Real world relationship types (e.g. person owns car)
- Example **relation schemas**:
 - *Car(color, brand)*
 - *Person(name: CHAR(20), age: INTEGER)*
 - *Owns(name, age, color, brand)*

Tuples

- A **tuple** is a realization of a relation schema
 - Assigns values to the attributes of the relation
- Example tuples of the relation *Car(color, brand)*:
 - ('red', 'Ford')
 - ('blue', 'Mercedes')
- Example tuples of the relation *Person(name, age)*:
 - ('Henry', 36)
 - ('Thomas', 22)

Relation Instance

- A **relation instance** consists of
 - relation schema
 - all its tuples
- Can be nicely visualized with a table:
Attribute / Column

The diagram shows a table representing a relation instance. The table has five columns and four rows. The first row is the header, with cells containing 'attribute₁', 'attribute₂', '...', '...', and '...'. A bracket above the last three columns is labeled 'Name'. The second and fourth rows are empty. The third row is shaded blue. A bracket to the left of the last three rows is labeled 'Tuple / Row'. A large bracket below the entire table is labeled 'Relation Instance / Table'.

Name				
attribute ₁	attribute ₂

Relation Instance / Table

Relation Instance

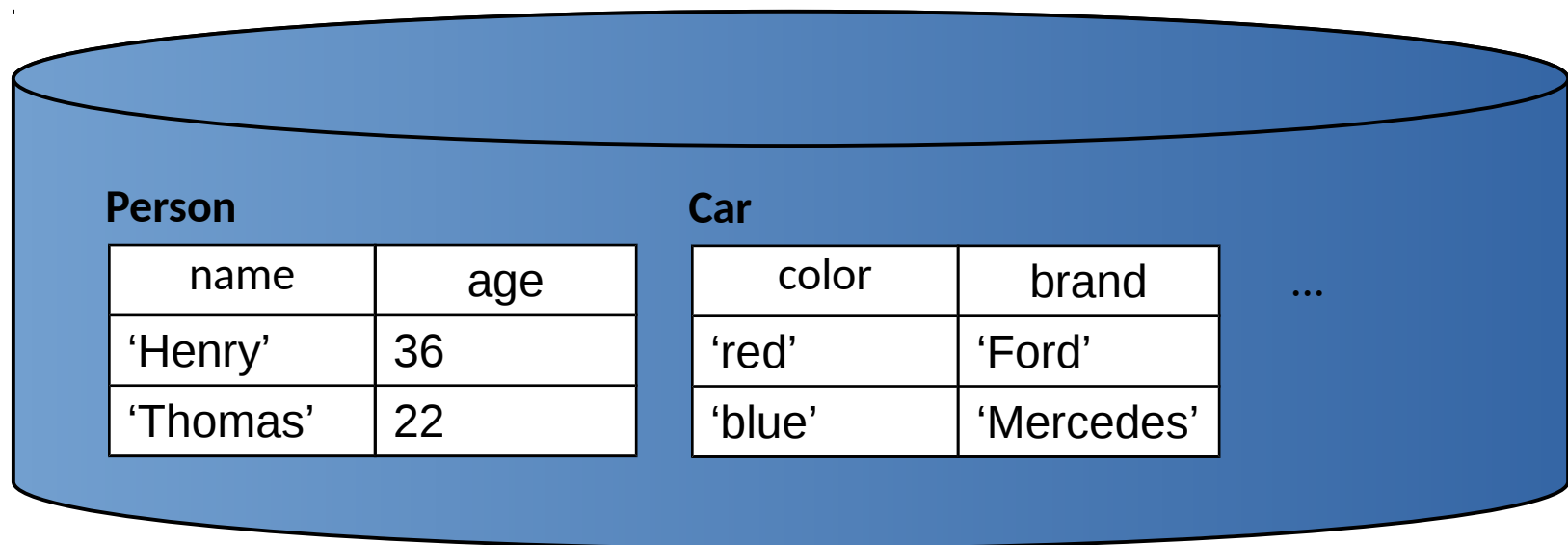
- Example **relation instance** of the person relation

Person

name	age
'Henry'	36
'Thomas'	22

Database Instance

- A **database instance** is the collection of all its relation instances
 - i.e. all relation schemas and their corresponding tuples



Integrity Constraints (ICs)

Integrity Constraints (ICs)

- Condition that must be true for any database instance
- Specified when schemas are defined
- Checked whenever relations are modified
 - i.e., when tuple is added, deleted, or modified

Domain constraints

- Domain of valid values for an attribute
 - e.g., INTEGER, FLOAT, CHAR(20), ...
 - correspond to data types in programming languages
- Example relation schema:

Person(name: CHAR(20), age: INTEGER)

name	age
'Henry'	36
'Mads'	'Doe'

← Domain constraint violation

→ Database will not allow insertion of this tuple

Semantic integrity constraints

- Restrictions on the data
 - e.g., $\text{age} \geq 18$

- Example relation schema:

Person(name: CHAR(20), age: INTEGER)

name	age
'Henry'	36
'Mads'	16

 Constraint violation

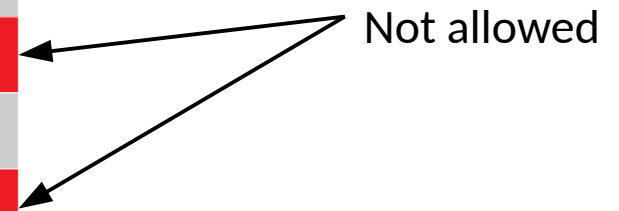
→ Database will not allow insertion of this tuple

Primary Keys

- Set of relation attributes
 - that uniquely identifies tuples of relation
 - all tuples need to have unique values for these attributes
- Example: CPR is primary key of relation **Person**
 - There cannot be two tuples with same CPR number

<u>CPR</u>	Name	Birthday	Address
...
1904651243	Svensson	19.04.1965	...
...
1904651243
...

Not allowed



Primary Keys

- Primary key “points” to exactly one tuple
 - can be used to lookup corresponding tuple
 - e.g., person can be looked up using CPR

What is the name of
the person with
CPR=1904651243 ?

<u>CPR</u>	Name	Birthday	Address
...
1904651243	Svensson	19.04.1965	...
...

Foreign Keys

- Allow to associate tuples in different relations
- Tuple of source relation \rightarrow tuple of target relation
 - Source and target relation can be the same
 - Can only point to a primary key in the target relation

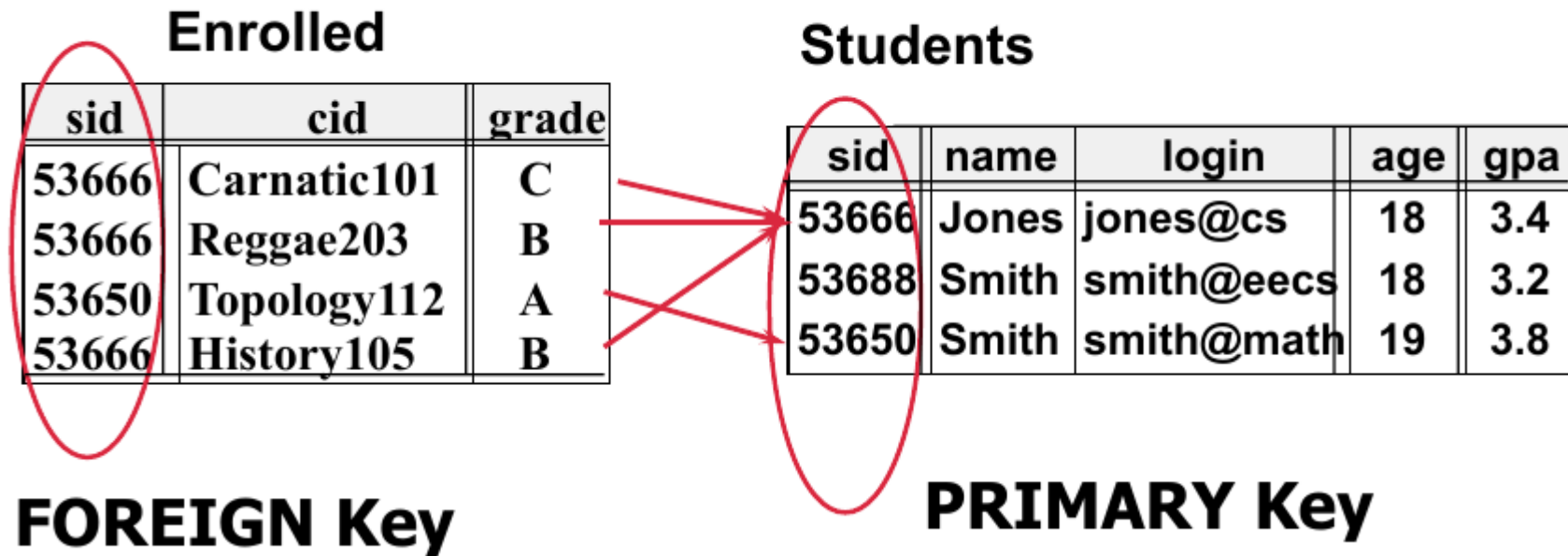
Example: University Database



- Conceptual 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)

Example: Foreign Keys



Query Languages

Query Languages

- Allow manipulation and retrieval of data from a database
- Query languages != programming languages
- not expected to be “turing complete”
 - i.e., not every operation can be expressed
- not intended to be used for complex calculations
- support easy, efficient access to large data sets

Relational Query Languages

- Based on relational algebra
- For relational databases, i.e. relational data model
- Relational model supports simple, powerful QLs:
 - Strong formal foundation based on logic
 - Allows for much optimization
- **SQL**: Most widely used relational query language

→ Understanding Relational Algebra is key to understanding SQL, query processing!

What is an “Algebra”?

- Mathematical system consisting of
 - **Operands:** Variables or values from which new values can be constructed
 - **Operators:** Symbols denoting procedures that construct new values from given values
- Example:
 - Integers [..., -1, 0, 1, ...] as operands
 - Arithmetic operations +/- as operators

What is Relational Algebra?

- An algebra where
 - operands are relations
 - operators are designed to query relations in a database
- Can be used as a query language for relations

Relational Algebra: 5 Basic Operations

- Selection: $\sigma_C(R)$
Selects a subset of tuples from relation R, for which condition C holds (horizontal)
- Projection: $\pi_{A_1, \dots, A_k}(R)$
Retains attributes A_1, \dots, A_k from relation R (vertical)
- Cross-product: **R1 x R2**
Pairwise combination of tuples of relations R1 and R2
- Set-difference: **R1 - R2**
Tuples in relation R1, but not in relation R2
- Union: **R1 U R2**
Tuples in relation R1 and/or in relation R2
- Since each operation returns a relation, operations can be composed (Algebra is “closed”)

Relational Algebra: Example Instances

Example Instances

Sailing Database:
Sailors, Boats, Reserves

Boats:

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

Reserves1:

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

Sailers1:

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Sailers2:

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

DM505 Database Design and Programming

Selection (σ)

Selects rows that satisfy *selection condition*.
Result is a relation.

Schema of result is same as that of the input relation.

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S_2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating > 8}(S_2)$

Projection (π)

Examples: $\pi_{age}(S2)$; $\pi_{sname,rating}(S2)$

Retains only attributes that are in the “*projection list*”.

Schema of result:

- exactly the fields in the projection list,
- with the same names that they had in the input relation.

Projection operator has to *eliminate duplicates* (How do they arise? Why remove them?)

- Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

Projection (π)

Projection

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname, rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Cross Product

S1 x R1: Each row of S1 paired with each row of R1.

Q: How many rows in the result?

***Result schema* has one field per field of S1 and R1, with field names 'inherited' if possible.**

- *May have a naming conflict:* Both S1 and R1 have a field with the same name.
- In this case, can use the *renaming operator*:
 $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

Cross Product

S1	<u>sid</u>	sname	rating	age	R1	<u>sid</u>	<u>bid</u>	<u>day</u>
	22	dustin	7	45.0		22	101	10/10/96
	31	lubber	8	55.5		58	103	11/12/96
	58	rusty	10	35.0				

$$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1) =$$

<u>sid1</u>	<u>sname</u>	<u>rating</u>	<u>age</u>	<u>sid2</u>	<u>bid</u>	<u>day</u>
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Union and Set Difference

All of these operations take two input relations, which must be union-compatible:

- Same number of fields.
- `Corresponding' fields have the same type.

For which, if any, is duplicate elimination required?

Union

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

S1US2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

Set Difference

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
22	dustin	7	45.0

S1 – S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

S2 – S1

Nesting Operators

- Result of a relational algebra operator is a relation
- It can be used as input to another relational algebra operator

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}(S2))$$

Compound Operator: Intersection

- In addition to the 5 basic operators, there are several additional “Compound Operators”
 - Do not add computational power to the language
 - Useful shorthands
 - Can be expressed with basic operations
- Example: **Intersection**
 - Takes two input relations that are union-compatible

$$R \cap S = R - (R - S)$$

Compound Operator: Intersection

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

S1 \cap S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

SQL - A language for Relational DBs

SQL - A language for Relational DBs

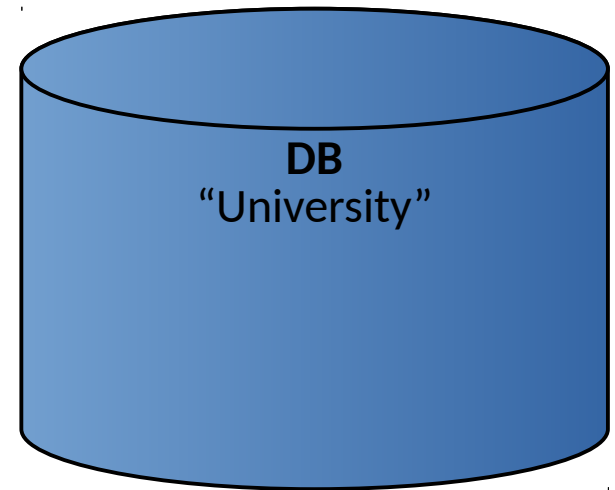
- Say: “ess-cue-ell” or “sequel”
 - But spelled “SQL”
- Data Definition Language (DDL)
 - create, modify, delete relations
 - specify constraints
 - administer users, security, etc.
- Data Manipulation Language (DML)
 - Specify queries to find tuples that satisfy criteria
 - add, modify, remove tuples
- The DBMS is responsible for efficient evaluation

SQL - A language for Relational DBs

- Query language to retrieve data from database
- Includes a data-definition component to define database schemas
- SQL commands have to be terminated with ‘;’
- SQL is standardized
 - some DBMS include their own SQL commands

Creating Databases in SQL

- Create a new, empty database 'University':
`CREATE DATABASE University;`
 - Does not contain any relations upon creation

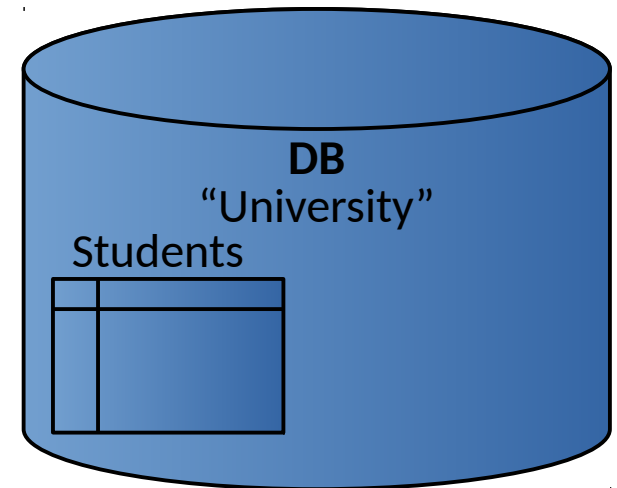


Creating Relations in SQL

- Create a new, empty relation 'Students':

```
CREATE TABLE Students (sid CHAR(20) PRIMARY KEY, name CHAR(20), login CHAR(10), age INTEGER, gpa FLOAT);
```

- Does not contain any tuples upon creation
- Note: the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

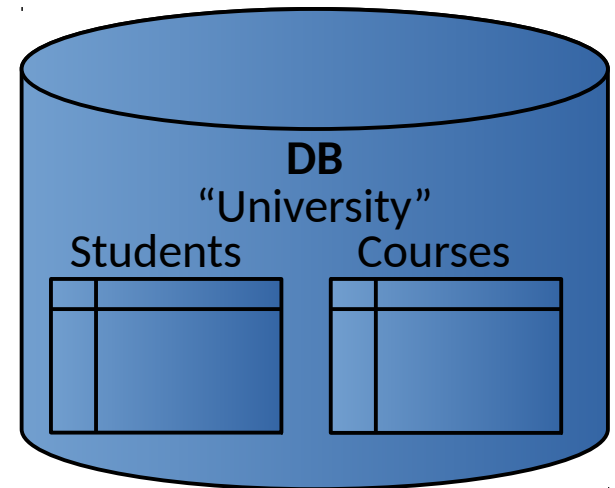


sid	name	login	age	gpa

Creating Relations in SQL

- Similarly:

```
CREATE TABLE Courses  
( cid CHAR(20) PRIMARY KEY, cname CHAR(20), credits  
INTEGER);
```



Adding and Deleting Tuples

- Insert a single tuple:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2);
```

sid	name	login	age	gpa
53688	Smith	smith@ee	18	3.2

- Delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE FROM Students S WHERE S.name = 'Smith';
```

sid	name	login	age	gpa
-----	------	-------	-----	-----

Selecting Tuples in SQL

- Find tuples for all 18 year old students with gpa's above 2.0:

```
SELECT * FROM Students S WHERE S.age=18 AND S.gpa > 2.0;
```

sid	name	login	age	gpa
53688	Smith	smith@ee	18	3.2

- To get just names and logins:

```
SELECT S.name, S.login FROM Students S WHERE S.age=18 AND S.gpa > 2.0;
```

name	login
Smith	smith@ee

Relational Algebra Operators in SQL

- Relational algebra operators can be expressed with SQL

- Selection operator (σ):

```
SELECT * FROM Students S
    WHERE S.age=18 AND S.gpa > 2.0;
```

- Projection operator (π):

```
SELECT S.age,S.gpa FROM Students S;
```

- Union:

```
SELECT * FROM Students S
    WHERE S.age=18 AND S.gpa > 2.0
UNION
SELECT * FROM Students S
    WHERE S.age=20 AND S.gpa > 2.3;
```

Relational Algebra Operators in SQL

- Set Difference:

```
SELECT * FROM Students S  
WHERE S.gpa > 2.0
```

EXCEPT

```
SELECT * FROM Students S  
WHERE S.age=19;
```

- Cross Product:

```
SELECT * FROM Students S, Enrolled E;
```


Primary Keys in SQL

- Single attribute key:

```
CREATE TABLE Students (sid CHAR(20) PRIMARY KEY,  
name CHAR(20), login CHAR(10), age INTEGER, gpa  
FLOAT)
```

- Multi-attribute key:

```
CREATE TABLE Enrolled (sid CHAR(20) cid CHAR(20),  
grade CHAR(2), PRIMARY KEY (sid,cid))
```

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses
 - sid is a foreign key referring to Students
- Students can only enroll for registered courses
 - cid is a foreign key referring to Courses

```
CREATE TABLE Enrolled  
  (sid CHAR(20),cid CHAR(20),grade CHAR(2),  
   PRIMARY KEY (sid,cid),  
   FOREIGN KEY (sid) REFERENCES Students,  
   FOREIGN KEY (cid) REFERENCES Courses);
```

Thank you for your attention!