

Database systems I

Radim Bača

radim.baca@vsb.cz

dbedu.cs.vsb.cz

 VSB TECHNICAL
UNIVERSITY
OF OSTRAVA

FACULTY OF ELECTRICAL
ENGINEERING AND COMPUTER
SCIENCE

DEPARTMENT
OF COMPUTER
SCIENCE

- Relational algebra
- SQL
 - Inner Join
 - Self Join
 - Outer Join

Relational algebra

- Relational algebra - set of operators on relations
- Operator - takes one or more relations as its input and produce a new relation

Selection	$\sigma_{condition}$
Projection	π
Cartesian product	\times
Join	\bowtie
Theta join	$\bowtie_{condition}$
Union	\cup
Intersection	\cap
Minus	$-$

Example

- Queries consist of elementary operations above relations, whose result is again a relation
- Example of a query:

Find name of those employees working at departments having sales at least 10 millions

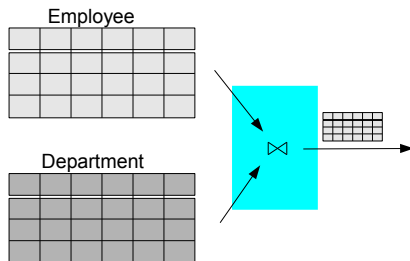
Employee

Department

Example

- Queries consist of elementary operations above relations, whose result is again a relation
- Example of a query:

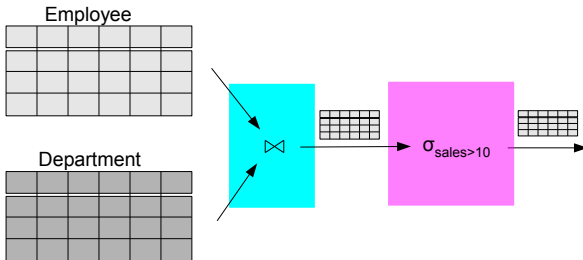
Find name of those employees working at departments having sales at least 10 millions



Example

- Queries consist of elementary operations above relations, whose result is again a relation
- Example of a query:

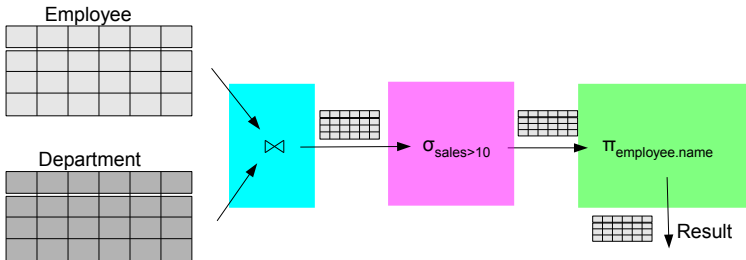
Find name of those employees working at departments having sales at least 10 millions



Example

- Queries consist of elementary operations above relations, whose result is again a relation
- Example of a query:

Find name of those employees working at departments having sales at least 10 millions



Restriction (selection)

- We select some rows from an input relation
- Notation: $\sigma_{condition}(Relation)$ - the condition defines rows that we want to pick
- *Example: Find all employees whose position is 'programmer'.*

Employee

eID	eName	passport no.	position
223	Newman	7905051111	programmer
124	Carter	6901112233	manager
154	Trier	7105029876	programmer

$\Downarrow \sigma_{position = 'programmer'}(Employee)$

eID	eName	passport no.	position
223	Newman	7905051111	programmer
154	Trier	7105029876	programmer

Restriction (selection)

- We select some rows from an input relation
- Notation: $\sigma_{condition}(Relation)$ - the condition defines rows that we want to pick
- *Example: Find all employees whose position is 'programmer'.*

Employee

eID	eName	passport no.	position
223	Newman	7905051111	programmer
124	Carter	6901112233	manager
154	Trier	7105029876	programmer

$\Downarrow \sigma_{position = 'programmer'}(Employee)$

eID	eName	passport no.	position
223	Newman	7905051111	programmer
154	Trier	7105029876	programmer

Projection

- We select some columns from an input relation
- Notation: $\pi_{\text{list of attributes}}(\text{Relation})$
- Example: Find IDs and names of all employees.

Employee

eID	eName	passport no.	position
223	Newman	7905051111	programmer
124	Carter	6901112233	manager
154	Trier	7105029876	programmer

$\Downarrow \pi_{eID, eName}(\text{Employee})$

eID	eName
223	Newman
124	Carter
154	Trier

Projection

- We select some columns from an input relation
- Notation: $\pi_{\text{list of attributes}}(\text{Relation})$
- Example: Find IDs and names of all employees.

Employee

eID	eName	passport no.	position
223	Newman	7905051111	programmer
124	Carter	6901112233	manager
154	Trier	7105029876	programmer

$\Downarrow \pi_{eID, eName}(\text{Employee})$

eID	eName
223	Newman
124	Carter
154	Trier

Duplications

- *Example: Find all positions of employees.*
 - **Relational algebra:** $\pi_{position}(Employee)$ - the resulting relation has two rows {(programmer),(manager)}
 - Duplications are automatically eliminated since we work with sets
 - **SQL:** `SELECT position FROM Employee` - the resulting relation has three rows
 - SQL works with multisets and the elimination of duplications has to be explicitly claimed

Duplications

- *Example: Find all positions of employees.*
 - **Relational algebra:** $\pi_{position}(Employee)$ - the resulting relation has two rows {(programmer),(manager)}
 - Duplications are automatically eliminated since we work with sets
 - **SQL:** `SELECT position FROM Employee` - the resulting relation has three rows
 - SQL works with multisets and the elimination of duplications has to be explicitly claimed

Cartesian Product

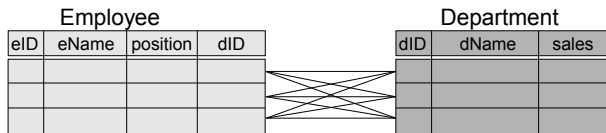
- Cartesian product $R \times S$ is a "combination" of two relations R and S
- Each row of the relation R is combined with each row of S
- *Example:*

eID	eName	position	dID

dID	dName	sales

Cartesian Product

- Cartesian product $R \times S$ is a "combination" of two relations R and S
- Each row of the relation R is combined with each row of S
- *Example:*



↓ $Employee \times Department$

eID	eName	position	dID	dID	dName	sales

Combination of Operations

- *Example: To all employees with the position 'manager' add an information about department at which they work.*

$\sigma_{\text{position}='manager' \wedge \text{Employee.dID}=\text{Department.dID}}(\text{Employee} \times \text{Department})$

$\sigma_{\text{Employee.dID}=\text{Department.dID}}(\sigma_{\text{position}='manager'}(\text{Employee}) \times \text{Department})$

Employee

eID	eName	position	dID

Department

dID	dName	sales

Combination of Operations

- *Example: To all employees with the position 'manager' add an information about department at which they work.*

$\sigma_{\text{position}='manager'} \wedge \text{Employee.dID}=\text{Department.dID} (\text{Employee} \times \text{Department})$

$\sigma_{\text{Employee.dID}=\text{Department.dID}} (\sigma_{\text{position}='manager'} (\text{Employee}) \times \text{Department})$

Employee			
eID	eName	position	dID

Department		
dID	dName	sales

Combination of Operations

- *Example: To all employees with the position 'manager' add an information about department at which they work.*

$\sigma_{\text{position}='manager'} \wedge \text{Employee.dID}=\text{Department.dID} (\text{Employee} \times \text{Department})$

$\sigma_{\text{Employee.dID}=\text{Department.dID}} (\sigma_{\text{position}='manager'} (\text{Employee}) \times \text{Department})$

Employee

eID	eName	position	dID

Department

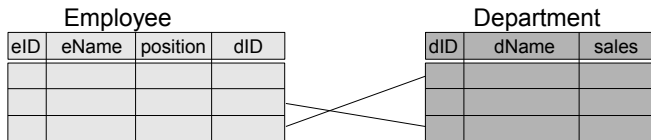
dID	dName	sales

Combination of Operations

- Example: To all employees with the position 'manager' add an information about department at which they work.

$\sigma_{\text{position}='manager'} \wedge \text{Employee.dID}=\text{Department.dID} (\text{Employee} \times \text{Department})$

$\sigma_{\text{Employee.dID}=\text{Department.dID}} (\sigma_{\text{position}='manager'} (\text{Employee}) \times \text{Department})$



Natural Join

- Natural join $R \bowtie S$ - from the product $R \times S$ we pick only rows with the same values of attributes of the same name in both relations
- Having the natural join introduced, the situation from the last example is simplified
- *Example: To all employees the with position 'manager' add an information about department at which they work.*

Original version:

$\sigma_{\text{position}='manager' \wedge \text{Employee.dID}=\text{Department.dID}}(\text{Employee} \times \text{Department})$

Natural join version:

$\sigma_{\text{position}='manager'}(\text{Employee} \bowtie \text{Department})$

Theta Join (JOIN)

- Theta join (abbreviated to join) $R \bowtie_{\Theta} S$ - from the product $R \times S$ we pick only rows satisfying the Θ condition
- Once again, by using theta join, we simplify the notation
- *Example: To all employees the with position 'manager' add an information about department at which they work.*

Original version:

$\sigma_{\text{position}='manager' \wedge \text{Employee.dID}=\text{Department.dID}}(\text{Employee} \times \text{Department})$

Theta join version:

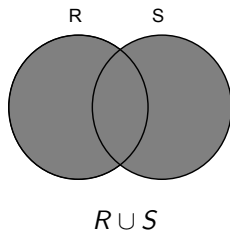
$\text{Employee} \bowtie_{\text{position}='manager' \wedge \text{Employee.dID}=\text{Department.dID}} \text{Department}$

Join Operation

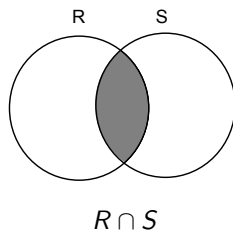
- Although, when using joins, we do not improve the expressing capabilities of language, joins enable us to simplify notation of queries
- Theta join is implemented in SQL as the JOIN operation and it is a very frequent operation!

Standard Set Operations

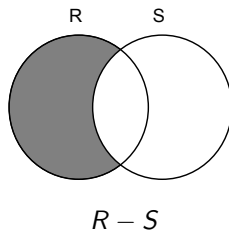
Union



Intersection



Difference



Intersection

- Intersection can be expressed by other set operations of relational algebra

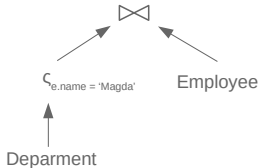
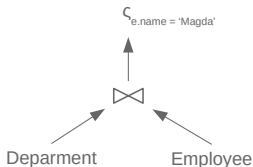
$$R \cap S = R - (R - S)$$

$$R \cap S = R \bowtie S \quad \text{when schemes of } R \text{ and } S \text{ are the same}$$

- So it again mainly simplifies the notation

Final Notes

- Some kind of relational algebra is often used as an internal representation of *query plans*
- Query plan describe how to execute the query
- To better understand a query plan or a relational algebra expression we usually use a tree.
- *Example: Find department names where an employee with name 'Magda' works.*



SQL

- Language for relational database systems
- Declarative
- SQL standards:
 - SQL-92 - Basic SQL constructs
 - SQL-99 - Regex, triggers, OO
 - SQL-2003 - XML, windows, sequences, auto-gen IDs
 - SQL-2008 - Truncate, offset/fetch
 - SQL-2011 - Temporal DB
 - SQL-2016 - JSON

SQL

- Data Definition Language (DDL) - alter database scheme
- Data Manipulation Language (DML) - alter database data & data retrieval
- Data Control Language (DCL) - configuration, access management, ...

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

```
SELECT  $A_1, \dots, A_n$   
FROM  $R_1, \dots, R_m$   
WHERE condition
```

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

SELECT A_1, \dots, A_n - which columns will be returned
FROM R_1, \dots, R_m
WHERE *condition*

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

SELECT A_1, \dots, A_n

- which columns will be returned

FROM R_1, \dots, R_m

- from which tables we will read data

WHERE *condition*

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

SELECT A_1, \dots, A_n

FROM R_1, \dots, R_m

WHERE *condition*

- which columns will be returned
- from which tables we will read data
- which condition has to be satisfied by returned rows

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

SELECT A_1, \dots, A_n
FROM R_1, \dots, R_m
WHERE *condition*



$\pi_{A_1, \dots, A_n} (\sigma_{condition}(R_1 \times \dots \times R_m))$

- The only difference between these two statements concerns duplications

SELECT Statement and Relational Algebra

- Basic structure of the SELECT statement:

SELECT A_1, \dots, A_n
FROM R_1, \dots, R_m
WHERE *condition*



$\pi_{A_1, \dots, A_n} (\sigma_{condition}(R_1 \times \dots \times R_m))$

- The only difference between these two statements concerns duplications

SELECT Statement and Relational Algebra (JOIN)

- Basic structure of the SELECT statement:

```
SELECT A1, ... , An
FROM R1
JOIN R2 ON join_condition    - Theta join between tables
WHERE condition
```



$$\pi_{A_1, \dots, A_n} (\sigma_{condition} (R_1 \bowtie_{join_condition} R_2))$$

SELECT Statement and Relational Algebra (JOIN)

- Basic structure of the SELECT statement:

```

SELECT  $A_1, \dots, A_n$ 
FROM  $R_1$ 
JOIN  $R_2$  ON join_condition      - Theta join between tables
WHERE condition

```

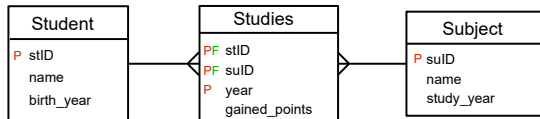


$$\pi_{A_1, \dots, A_n} (\sigma_{condition} (R_1 \bowtie_{join_condition} R_2))$$

Sakila Database

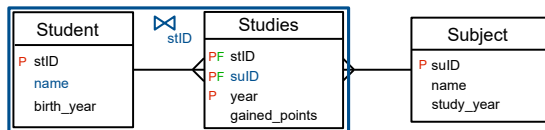
- Sakila database <https://github.com/ivanceras/sakila>
- Open-source, well-designed, support for more database systems
- One database for lectures and exercises

Example: Join of two tables



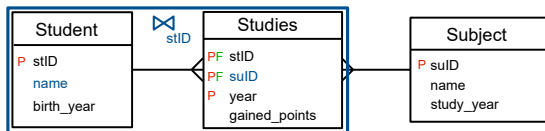
- *To every name of student find also suID of a subject, which he/she studies or studied.*

Example: Join of two tables



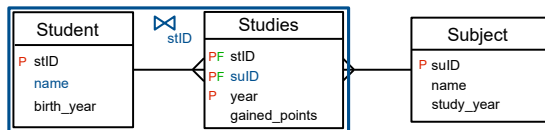
- *To every name of student find also suID of a subject, which he/she studies or studied.*

Example: Join of two tables



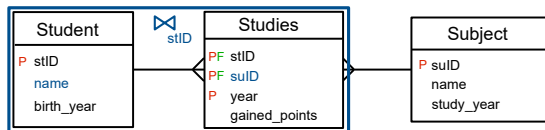
- *To every name of student find also suID of a subject, which he/she studies or studied.*
 - `SELECT name, suID`
`FROM Student`
`JOIN Studies ON Student.stID = Studies.stID`
 - $\pi_{name, suID} (Student \bowtie Studies)$
 - From the duplications perspective, the results of both queries do not have to be the same!

Example: Join of two tables



- *To every name of student find also suID of a subject, which he/she studies or studied.*
 - `SELECT name, suID`
`FROM Student`
`JOIN Studies ON Student.stID = Studies.stID`
 - $\pi_{name, suID} (Student \bowtie Studies)$
 - From the duplications perspective, the results of both queries do not have to be the same!

Example: Join of Two Tables with Condition

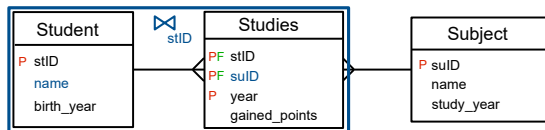


- Find *sulIDs* of all subjects, which students called Petr studied in 2010.
 - ```

SELECT name, sulID
FROM Student
JOIN Studies ON Student.stID = Studies.stID
WHERE year = 2010 and name = 'Petr'

```
  - $$\pi_{sulID} (\sigma_{year=2010 \wedge name='Petr'} (Student \bowtie Studies))$$

## Example: Join of Two Tables with Condition

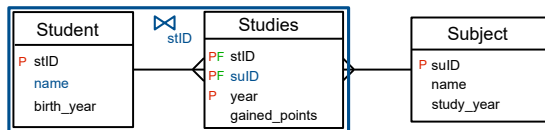


- Find *suIDs* of all subjects, which students called Petr studied in 2010.
  - ```

SELECT name, suID
FROM Student
JOIN Studies ON Student.stID = Studies.stID
WHERE year = 2010 and name = 'Petr'

```
 - $$\pi_{suID} (\sigma_{year=2010 \wedge name='Petr'} (Student \bowtie Studies))$$

Example: Join of Two Tables with Condition



- Find *suIDs* of all subjects, which students called Petr studied in 2010.
 - ```
SELECT name, suID
FROM Student
JOIN Studies ON Student.stID = Studies.stID
WHERE year = 2010 and name = 'Petr'
```
  - $$\pi_{suID} (\sigma_{year=2010 \wedge name='Petr'} (Student \bowtie Studies))$$

# Example: Join of Two Tables with Condition

```
SELECT distinct pID
FROM Student
JOIN Study On Student.sID = Study.sID
WHERE year = 2011 and name = 'Petr'
```

Student

| sID | name  | birth_date |
|-----|-------|------------|
| 1   | Petr  | 1990       |
| 2   | Pavel | 1991       |
| 3   | Ivana | 1990       |

Study

| sID | pID | year | points |
|-----|-----|------|--------|
| 1   | 35  | 2010 | 23     |
| 1   | 35  | 2011 | 55     |
| 1   | 21  | 2010 | 89     |
| 2   | 46  | 2010 | 59     |

 $\pi_{pID}$ 
 $\sigma_{year=2011 \wedge name='Petr'}$ 
 $Student \bowtie Study$

# Example: Join of Two Tables with Condition

SELECT distinct pID

FROM Student

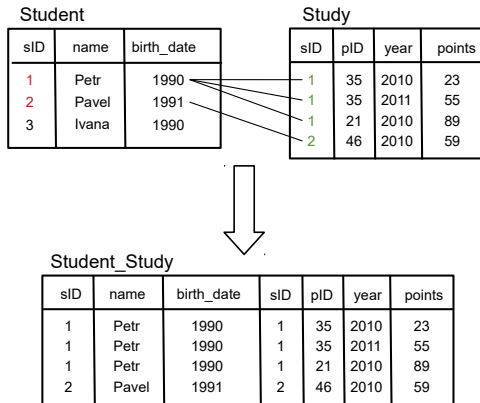
JOIN Study On Student.sID = Study.sID

WHERE year = 2011 and name = 'Petr'

$\pi_{pID}$

$\sigma_{year=2011 \wedge name='Petr'}$

*Student* ⋈ *Study*



# Example: Join of Two Tables with Condition

SELECT distinct pID

FROM Student, Study

JOIN Study ON Student.sID = Study.sID

WHERE year = 2011 and name = 'Petr'

$\pi_{pID}$

$\sigma_{year=2011 \wedge name='Petr'}$

*Student* ⋈ *Study*

Student

| sID | name  | birth_date |
|-----|-------|------------|
| 1   | Petr  | 1990       |
| 2   | Pavel | 1991       |
| 3   | Ivana | 1990       |

Study

| sID | pID | year | points |
|-----|-----|------|--------|
| 1   | 35  | 2010 | 23     |
| 1   | 35  | 2011 | 55     |
| 1   | 21  | 2010 | 89     |
| 2   | 46  | 2010 | 59     |



Student\_Study

| sID | name  | birth_date | sID | pID | year | points |
|-----|-------|------------|-----|-----|------|--------|
| 1   | Petr  | 1990       | 1   | 35  | 2010 | 23     |
| 1   | Petr  | 1990       | 1   | 35  | 2011 | 55     |
| 1   | Petr  | 1990       | 1   | 21  | 2010 | 89     |
| 2   | Pavel | 1991       | 2   | 46  | 2010 | 59     |

# Example: Join of Two Tables with Condition

```
SELECT distinct pID
FROM Student, Study
JOIN Study On Student.sID = Study.sID
WHERE year = 2011 and name = 'Petr'
```

 $\pi_{pID}$ 
 $\sigma_{year=2011 \wedge name='Petr'}$ 

*Student* ⋈ *Study*

| sID | name  | birth_date |
|-----|-------|------------|
| 1   | Petr  | 1990       |
| 2   | Pavel | 1991       |
| 3   | Ivana | 1990       |

| sID | pID | year | points |
|-----|-----|------|--------|
| 1   | 35  | 2010 | 23     |
| 1   | 35  | 2011 | 55     |
| 1   | 21  | 2010 | 89     |
| 2   | 46  | 2010 | 59     |

↓

| sID | name  | birth_date | sID | pID | year | points |
|-----|-------|------------|-----|-----|------|--------|
| 1   | Petr  | 1990       | 1   | 35  | 2010 | 23     |
| 1   | Petr  | 1990       | 1   | 35  | 2011 | 55     |
| 1   | Petr  | 1990       | 1   | 21  | 2010 | 89     |
| 2   | Pavel | 1991       | 2   | 46  | 2010 | 59     |



# Example: Join of Two Tables with Condition

```
SELECT distinct pID
FROM Student, Study
JOIN Study On Student.sID = Study.sID
WHERE year = 2011 and name = 'Petr'
```

 $\pi_{pID}$ 
 $\sigma_{year=2011 \wedge name='Petr'}$ 
 $Student \bowtie Study$ 

Student\_Study

| sID | name  | birth_date | sID | pID | year | points |
|-----|-------|------------|-----|-----|------|--------|
| 1   | Petr  | 1990       | 1   | 35  | 2010 | 23     |
| 1   | Petr  | 1990       | 1   | 35  | 2011 | 55     |
| 1   | Petr  | 1990       | 1   | 21  | 2010 | 89     |
| 2   | Pavel | 1991       | 2   | 46  | 2010 | 59     |



Student\_Study

| sID | name | birth_date | sID | pID | year | points |
|-----|------|------------|-----|-----|------|--------|
| 1   | Petr | 1990       | 1   | 35  | 2011 | 55     |

# Example: Join of Two Tables with Condition

**SELECT** *distinct* pID

FROM Student, Study

JOIN Study ON Student.sID = Study.sID

WHERE year = 2011 and name = 'Petr'

$\pi_{pID}$

$\sigma_{year=2011 \wedge name='Petr'}$

*Student* ⋈ *Study*

Student\_Study

| sID | name | birth_date | sID | pID | year | points |
|-----|------|------------|-----|-----|------|--------|
| 1   | Petr | 1990       | 1   | 35  | 2011 | 55     |

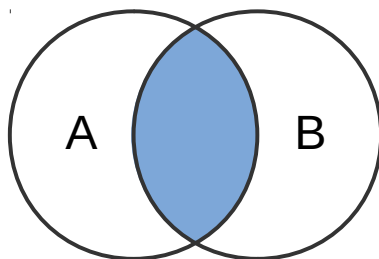


Student\_Study

| pID |
|-----|
| 35  |

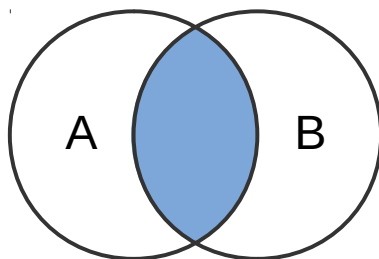
## Inner Join Observation

```
SELECT *
FROM A
JOIN B ON A.k = B.k
```



## Inner Join Observation

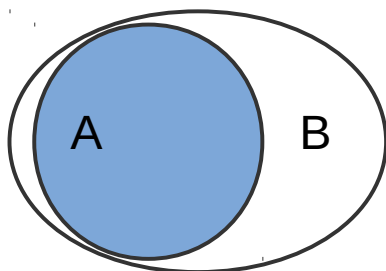
```
SELECT *
FROM A
JOIN B ON A.k = B.k
```



What if  $A.k$  is foreign key and  $B.k$  is a primary key?

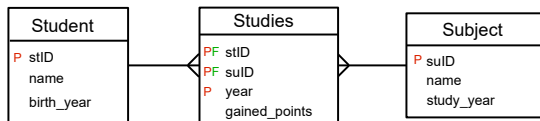
## Inner Join Observation

```
SELECT *
FROM A
JOIN B ON A.k = B.k
```



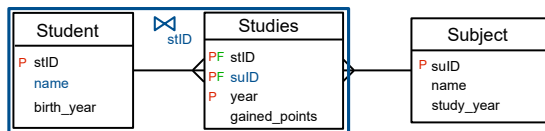
Note that set B is usually smaller (i.e. the visualization is not very accurate)

## Example: Join of Three Tables



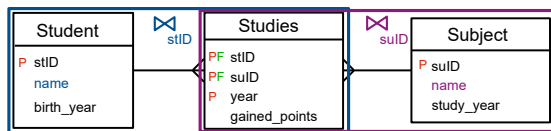
- *To each name of subject find name of student who studies or studied this subject.*

## Example: Join of Three Tables



- *To each name of subject find name of student who studies or studied this subject.*
  - `SELECT distinct Studies.suID, Student.name  
FROM Student  
JOIN Studies ON Student.stID = Studies.stID`

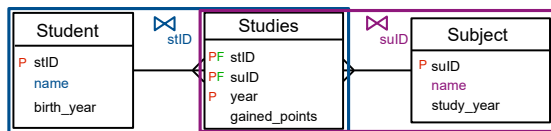
## Example: Join of Three Tables



- *To each name of subject find name of student who studies or studied this subject.*
  - `SELECT distinct Subject.name, Student.name`  
`FROM Student`  
`JOIN Studies ON Student.stID = Studies.stID`  
`JOIN Subject ON Studies.suID = Subject.suID`

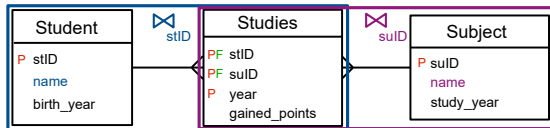


## Example: Join of Three Tables



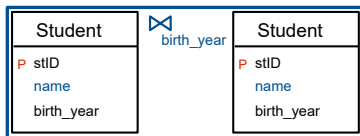
- *To each name of subject find name of student who studies or studied this subject.*
  - `SELECT distinct Subject.name, Student.name`  
`FROM Student`  
`JOIN Studies ON Student.stID = Studies.stID`  
`JOIN Subject ON Studies.suID = Subject.suID`
  - $\pi_{Subject.name, Student.name} (Student \bowtie Studies \bowtie Subject)$

## Example: Join of Three Tables + Renaming a Relation



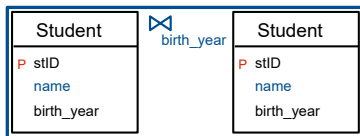
- *To each name of subject find name of student who studies or studied this subject.*
  - `SELECT distinct su.name, st.name`  
`FROM Student st`  
`JOIN Studies se ON st.stID = se.stID`  
`JOIN Subject su ON se.suID = su.suID`
  - $\pi_{Subject.name, Student.name} (Student \bowtie Studies \bowtie Subject)$
  - **For the sake of clarity, we can rename the relations**

## Example: Self Join



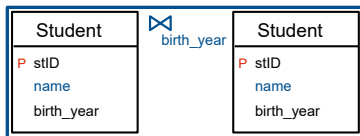
- *Find all pairs of students having the same birth year.*
  - ```
SELECT s1.name, s2.name
FROM Student s1
JOIN Student s2
      ON s1.birth_year = s2.birth_year
```

Example: Self Join



- *Find all pairs of students having the same birth year.*
 - ```
SELECT s1.name, s2.name
FROM Student s1
JOIN Student s2
 ON s1.birth_year = s2.birth_year
```
  - **Not quite correct solution!**

## Example: Self Join



- *Find all pairs of students having the same birth year.*
  - ```
SELECT s1.name, s2.name
FROM Student s1
JOIN Student s2
      ON s1.birth_year = s2.birth_year
      AND s1.stID > s2.stID
```

Inner Join Rules

sID	name	birth_date
1	Petr	1990
2	Pavel	1991
3	Ivana	1990

sID	pID	year	points
1	35	2010	23
1	35	2011	55
1	21	2010	89
2	46	2010	59

- **Multiplication** - Every student is repeated as many times as is the number of his studies
- **Elimination** - A student is eliminated if he did not study anything

Inner Join Rules

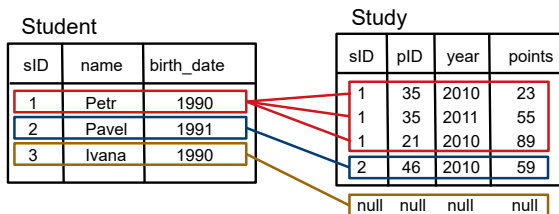
sID	name	birth_date
1	Petr	1990
2	Pavel	1991
3	Ivana	1990

sID	pID	year	points
1	35	2010	23
1	35	2011	55
1	21	2010	89
2	46	2010	59
null	null	null	null

- **Multiplication** - Every student is repeated as many times as is the number of his studies
- **Elimination** - A student is eliminated if he did not study anything

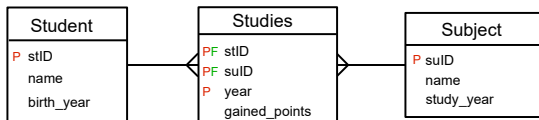
However, sometimes we do not want to eliminate!

Outer Join



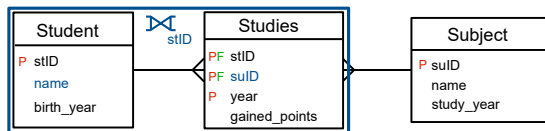
- This is called outer join
- We have more types of outer joins:
 - Left/Right outer join
 - Full outer join

Example: Left Outer Join



- List names of students and sulDs of subjects studied by them; if a student does not study any subject, write NULL to his/her name.

Example: Left Outer Join

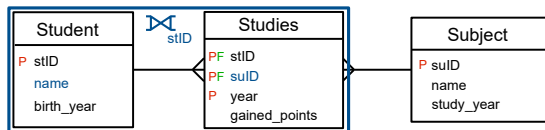


- List names of students and suIDs of subjects studied by them; if a student does not study any subject, write NULL to his/her name.

```

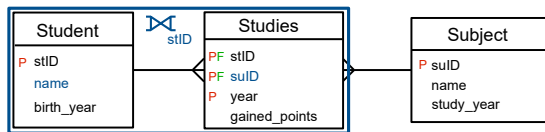
SELECT name, suID
FROM Student
LEFT JOIN Studies ON
    Student.stID = Studies.stID
  
```

Example: Left Outer Join with Condition



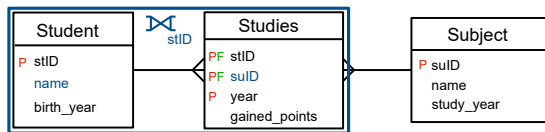
- List student names and suIDs of subjects studied by them in 2011; if a student does not study a subject, write NULL to his name.

Example: Left Outer Join with Condition



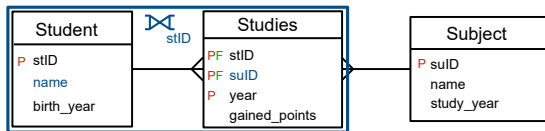
- *List student names and suIDs of subjects studied by them in 2011; if a student does not study a subject, write NULL to his name.*
- ```
SELECT name, suID
FROM Student
LEFT JOIN Studies ON
 Student.stID = Studies.stID
WHERE year = 2011
```
- Is this correct?

## Example: Left Outer Join with Condition



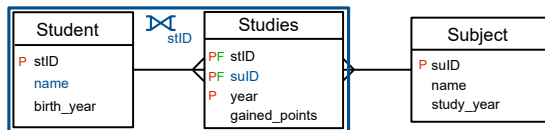
- *List student names and suIDs of subjects studied by them in 2011; if a student does not study a subject, write NULL to his name.*
- ```
SELECT name, suID
FROM Student
LEFT JOIN Studies ON
      Student.stID = Studies.stID
WHERE year = 2011
```
- Is this correct?

Example: LEFT OUTER JOIN with Condition



- *List student names and suIDs of subjects studied by them in 2011; if a student does not study a subject, write NULL to his name.*
- ```
SELECT name, suID
FROM Student
LEFT JOIN Studies ON
 Student.stID = Studies.stID
WHERE year = 2011
```
- **No, it is not.** The WHERE condition removes all students which does not study anything in 2011!

## Example: LEFT OUTER JOIN with Condition



- *List student names and suIDs of subjects studied by them in 2011; if a student does not study a subject, write NULL to his name.*
- ```
SELECT name, suID
FROM Student
LEFT JOIN Studies ON
    Student.stID = Studies.stID
    AND year = 2011
```

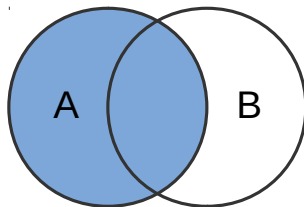
SQL Clause Priority

- ❶ FROM
- ❷ JOIN
- ❸ WHERE
- ❹ SELECT
- ❺ . . .

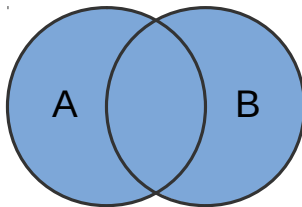
- Semantic order
- Not a query processing order!

Outer Join Visualization

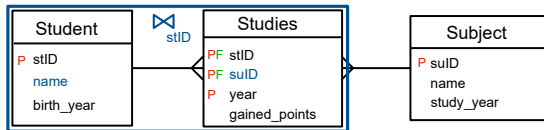
```
SELECT *  
FROM A  
LEFT JOIN B  
    ON A.k = B.k
```



```
SELECT *  
FROM A  
FULL JOIN B  
    ON A.k = B.k
```

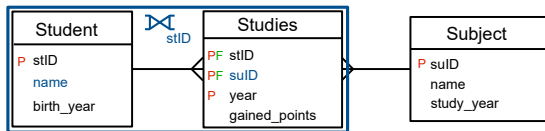


Example: Difference



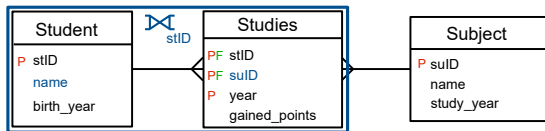
- *List names of students who did not studied anything*

Example: Difference



- *List names of students and sulIDs of subjects studied by them; if a student does not study any subject, write NULL to his/her name.*
- ```
SELECT name
FROM Student
LEFT JOIN Studies ON
 Student.stID = Studies.stID
WHERE Studies.stID IS NULL
```
- We will show other ways how to implement the difference later

## Example: Difference



- *List names of students and sulIDs of subjects studied by them; if a student does not study any subject, write NULL to his/her name.*

- ```
SELECT name
FROM Student
LEFT JOIN Studies ON
      Student.stID = Studies.stID
WHERE Studies.stID IS NULL
```
- We will show other ways how to implement the difference later

References

- Stránky UDBS na <http://dbedu.cs.vsb.cz>
- Andrew Pavlo, CMU Database systems
<https://www.youtube.com/watch?v=KG-mqHoXOXY>