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

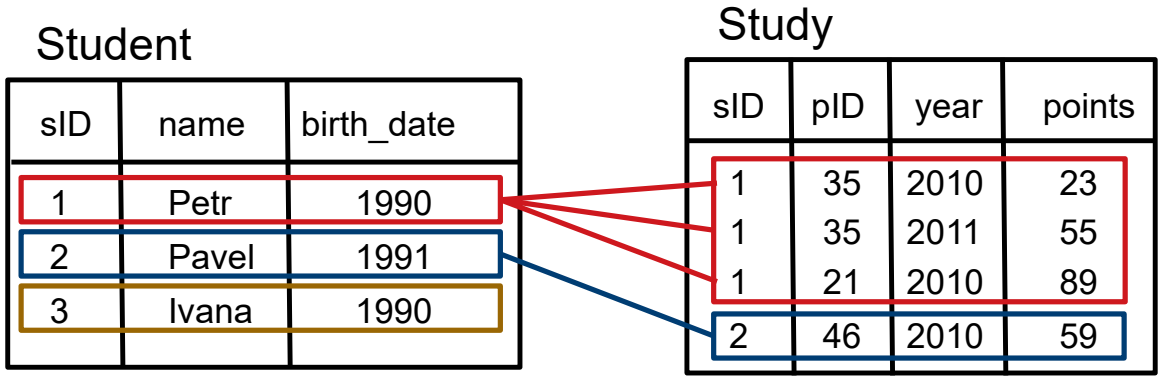
EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Content

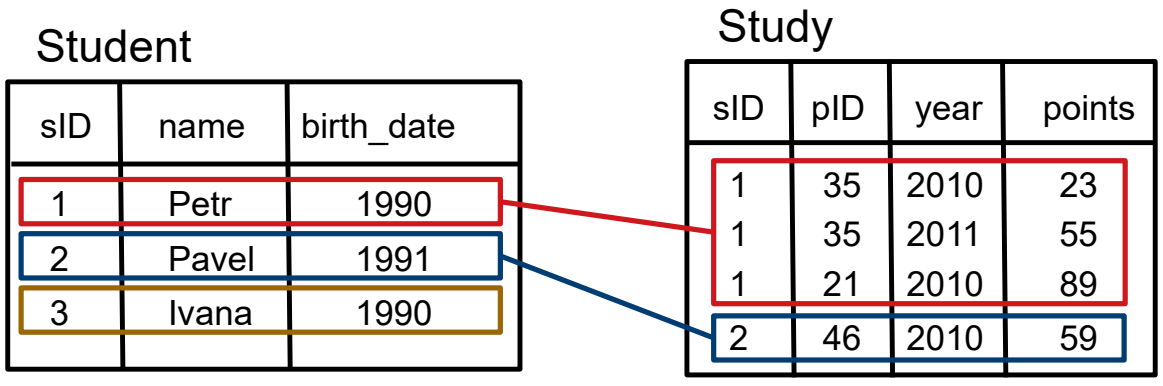
- Subqueries & Semi-joins (IN, EXISTS, ALL)
- Set operations
- Subqueries following FROM
- How to handle complex queries

Inner Join Rules



- **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

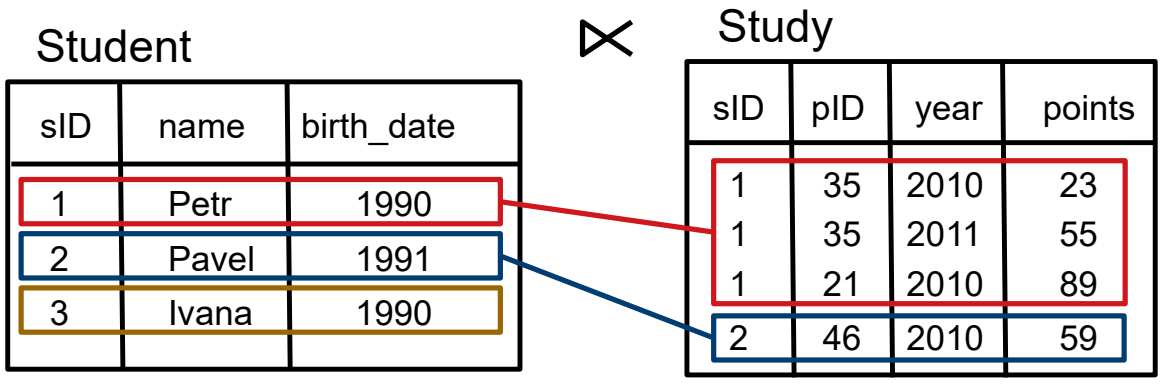
Inner Join Rules



- **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 multiply!

Inner Join Rules



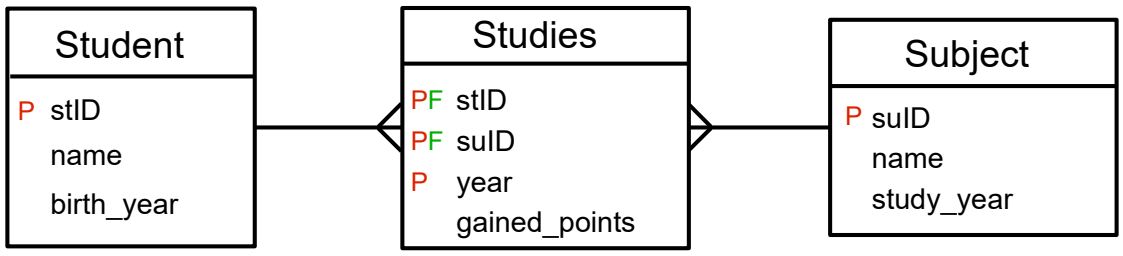
- **Multiplication** - Every student is repeated as many times as is the number of his studies
- **Elimination** - A student is eliminated if he did not study anything

Here comes a semi join!

Semi Join

- There are several ways how to express a semi join in SQL:
  - IN subquery
  - EXISTS subquery

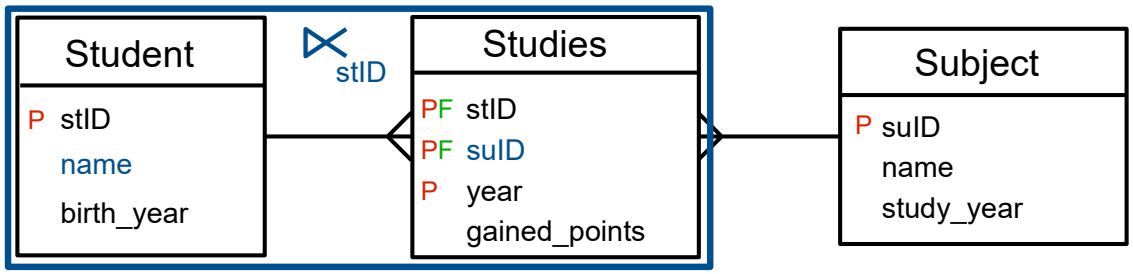
Example: IN subquery



- *Find names of all students who studied some subject in 2010.*

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE year=2010
)
```

Example: IN subquery



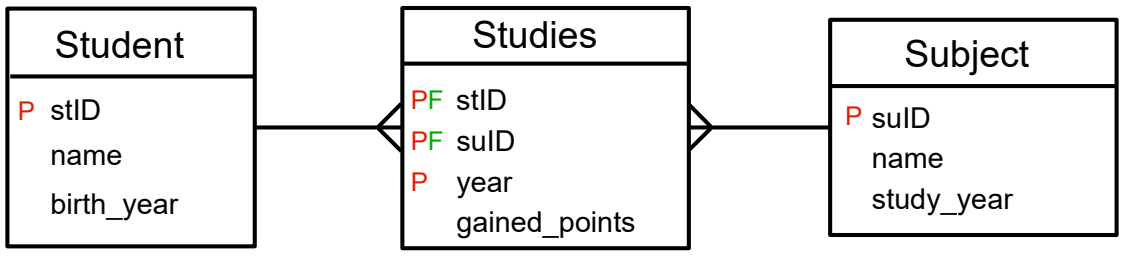
- *Find names of all students who studied some subject in 2010.*

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE year=2010
)
```

?
⇔

```
SELECT name
FROM Student st
JOIN Studies ss
   ON st.stID = ss.stID
WHERE year=2010
```

Example: IN subquery



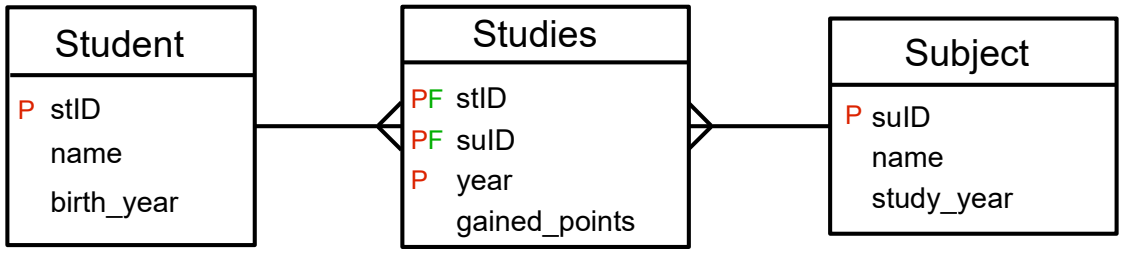
- *Find names of all students who studied some subject in 2010.*

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE year=2010
)
```

⇔

```
SELECT distinct name
FROM Student st
JOIN Studies ss
   ON st.stID = ss.stID
WHERE year=2010
```

Example: Exists Subquery



- *Find names of all students who studied some subject in 2010.*

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE study_year=2010
)
```

⇔

```
SELECT name
FROM Student st
WHERE EXISTS(
   SELECT 1 FROM Studies se
   WHERE se.year=2010
      st.stID = se.stID
)
```

Dependent vs. Independent Subqueries

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE study_year=2010
)
```

⇔

```
SELECT name
FROM Student st
WHERE EXISTS(
   SELECT 1 FROM Studies se
   WHERE se.year=2010
      st.stID = se.stID
)
```

- Independent subquery - can be processed separately
- Dependent subquery - some value from the outer query is used
- Some database systems use just nested-loop joins to process dependent subqueries (last lecture example)

Dependent vs. Independent Subqueries

```
SELECT name
FROM Student
WHERE stID IN(
   SELECT stID
   FROM Studies
   WHERE study_year=2010
)
```

$\Leftrightarrow$

```
SELECT name
FROM Student st
WHERE EXISTS(
   SELECT 1 FROM Studies se
   WHERE se.year=2010
      st.stID = se.stID
)
```

- Independent subquery - can be processed separately
- Dependent subquery - some value from the outer query is used
- Some database systems use just nested-loop joins to process dependent subqueries (last lecture example)

Operator IN

- `SELECT * FROM R WHERE R.b IN (list of values)`
- For each row of relation `R` we check whether a value of attribute `b` is in the list of values
- We usually obtain the list of values by some nested query
- The nested query usually return values of attributes which are related to $R.b$
- Common bug are queries like this one:

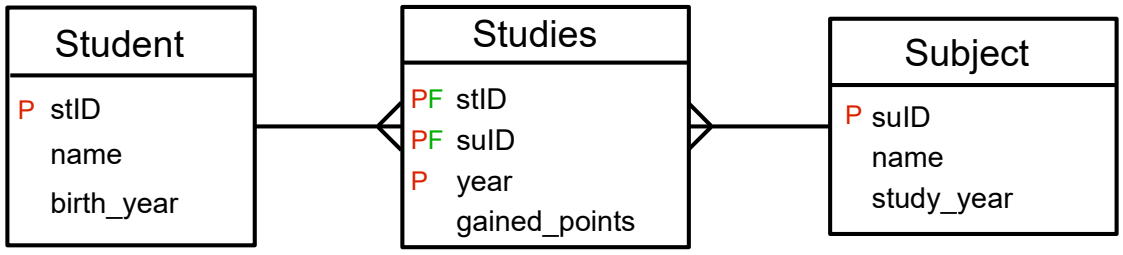
```
SELECT * FROM Student
WHERE stID IN (SELECT suID FROM Studies)
```

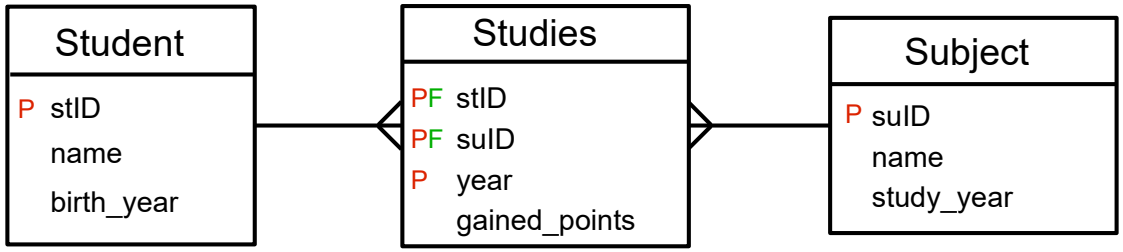
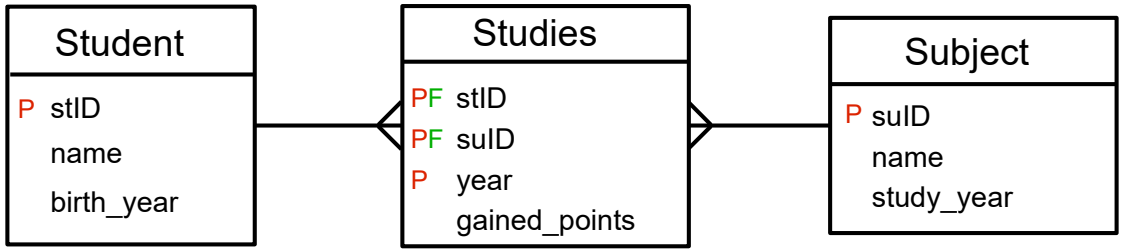
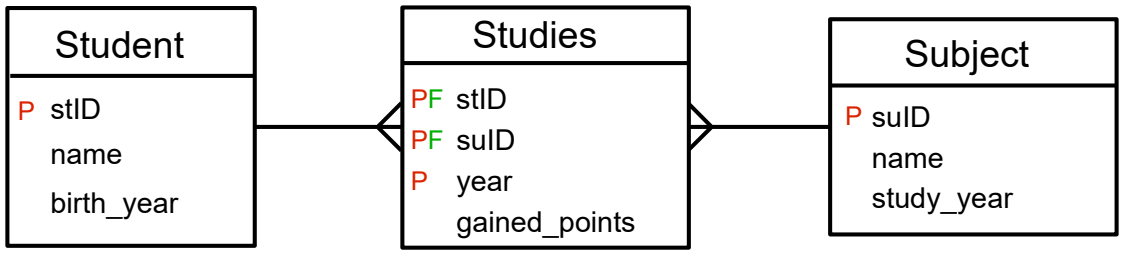
Operator IN

- `SELECT * FROM R WHERE R.b IN (list of values)`
- For each row of relation `R` we check whether a value of attribute `b` is in the list of values
- We usually obtain the list of values by some nested query
- The nested query usually return values of attributes which are related to $R.b$
- Common bug are queries like this one:

```
SELECT * FROM Student
WHERE stID IN (SELECT suID FROM Studies)
```

Example: Difference using operator IN



- *Find suIDs of all subjects which are studied only by students born after 1985.*
  - ```
    SELECT DISTINCT suID FROM Studies
    WHERE Studies.suID NOT IN (
       SELECT suID FROM Student st
       JOIN Studies ss ON st.stID = ss.stID and
       WHERE birth_year<=1985 or birth_year IS NULL
    )
    ```
  - $\pi_{suID}(Studies) - \pi_{suID}(Student \bowtie_{birth\_year<=1985} Studies)$
    ```

Problems of NOT IN Operator

- `SELECT * FROM R`
  `WHERE R.b NOT IN (list of values)`
- Condition is evaluated as a `true` if comparison of `R.b` with every value in the list is `false`.
- The problem is when the list of values contains `NULL`!
- The condition is then always evaluated to `unknown` and the query result is empty.
- If the list of values is obtained through an SQL, this mistake may not be easy to recognize from the beginning; due to this, it is more safe to use `NOT EXISTS`.

Problems of NOT IN Operator

- `SELECT * FROM R`
  `WHERE R.b NOT IN (list of values)`
- Condition is evaluated as a `true` if comparison of `R.b` with every value in the list is `false`.
- The problem is when the list of values contains `NULL`!
- The condition is then always evaluated to `unknown` and the query result is empty.
- If the list of values is obtained through an SQL, this mistake may not be easy to recognize from the beginning; due to this, it is more safe to use `NOT EXISTS`.

Example: Operator Exists



- *Find all subjects for which there is another subject taught in the same study year.*

```
SELECT * FROM Subject s1
WHERE EXIST (
   SELECT 1 FROM Subject s2
   WHERE s1.study_year = s2.study_year
         and s1.suID <> s2.suID
)
```

Example: Operator Exists



- *Find all subjects for which there is another subject taught in the same study year.*

```
SELECT * FROM Subject s1
WHERE EXIST (
   SELECT 1 FROM Subject s2
   WHERE s1.study_year = s2.study_year
         and s1.suID <> s2.suID
)
```

- Can be expressed using self-join (see second lecture)

Example: Not Exists



- *Find the oldest student.*
  - ```
    SELECT * FROM Student s1
    WHERE NOT EXISTS (
       SELECT 1 FROM Student s2
       WHERE s1.birth_year > s2.birth_year
    ) and birth_year is not null
    ```
  - Can be computed by using aggregation (see previous lecture)
    ```

EXISTS and NOT EXISTS Operators

- `SELECT * FROM R WHERE EXISTS (subquery)`
- For each row of `R` we check whether the subquery returns a result or not
- Predicate returns `true` for a row if query result is non-empty, otherwise `false`
- In order to get meaningful results we have to use a correlated subquery
- What is meaning of the following query?

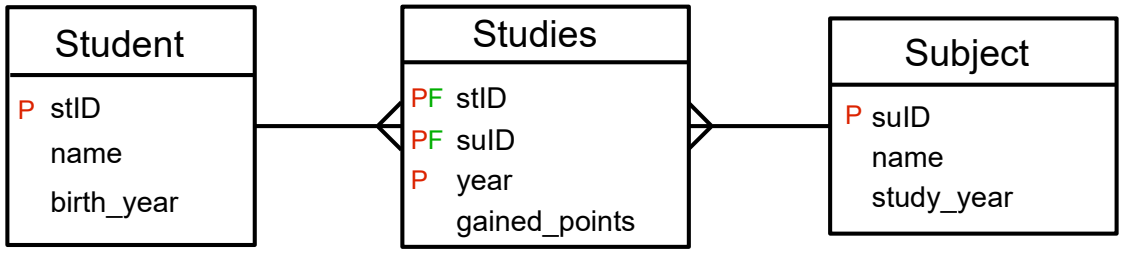
```
SELECT * FROM Student s1
WHERE EXISTS (SELECT 1 FROM Student s2
              WHERE s2.jmeno = 'Petr')
```

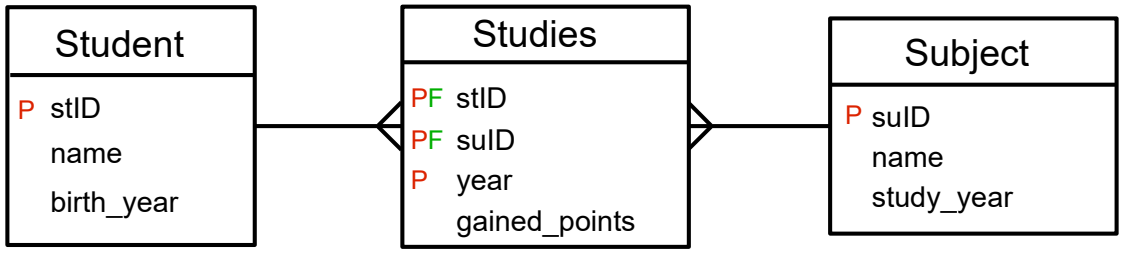
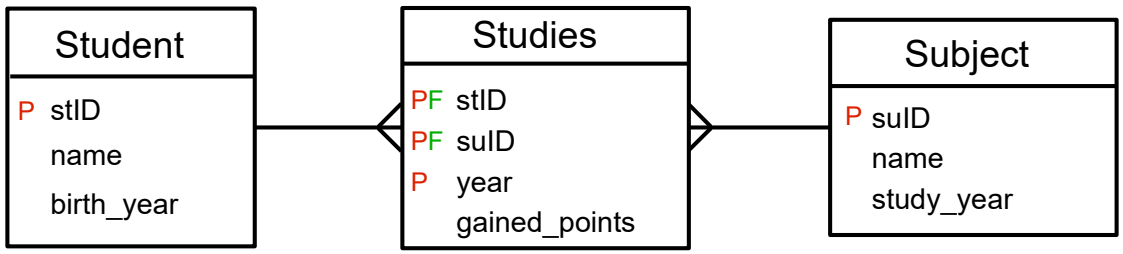
EXISTS and NOT EXISTS Operators

- `SELECT * FROM R WHERE EXISTS (subquery)`
- For each row of `R` we check whether the subquery returns a result or not
- Predicate returns `true` for a row if query result is non-empty, otherwise `false`
- In order to get meaningful results we have to use a correlated subquery
- What is meaning of the following query?

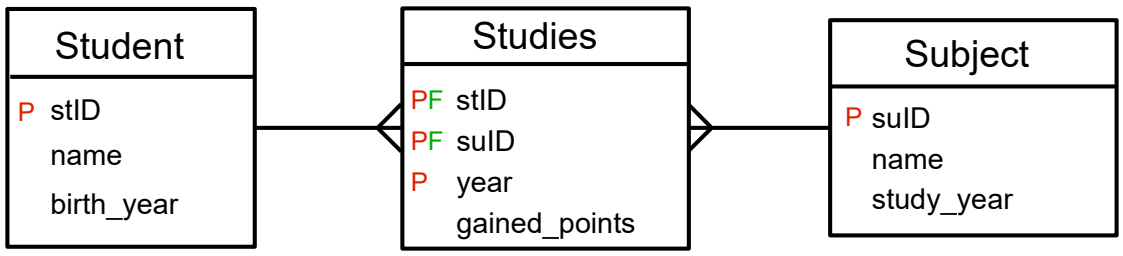
```
SELECT * FROM Student s1
WHERE EXISTS (SELECT 1 FROM Student s2
                WHERE s2.jmeno = 'Petr')
```

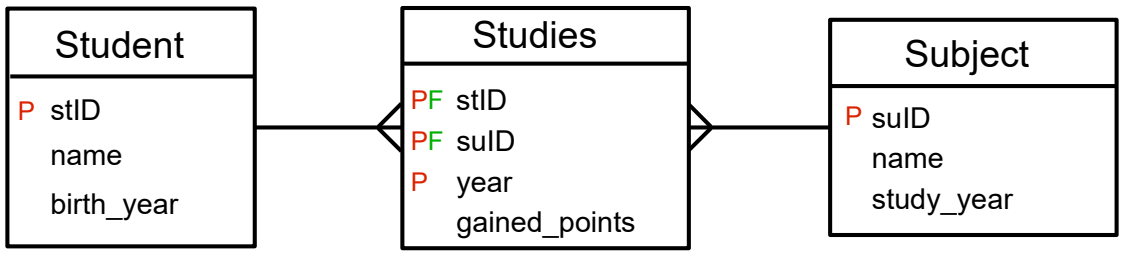
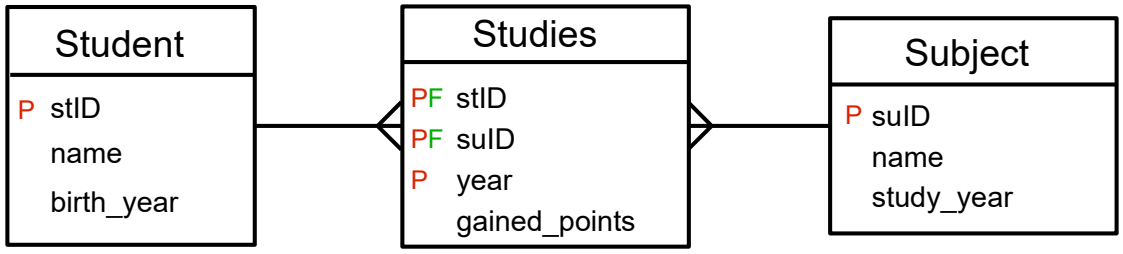
Example: All



- *Find the oldest student.*
  - ```
    SELECT * FROM Student S1
    WHERE S1.birth_year <= all(
          SELECT S2.birth_year FROM Student S2
          WHERE S2.birth_year is not null
    )
    ```
  - The `ALL` operator says that the operation has to be satisfied for all entries in the parentheses
  - Similarly, there is an `ANY` operator saying that the operation has to be satisfied at least for one entry in the parentheses
    ```

Example: All



- *Find the oldest student.*
  - `SELECT * FROM Student S1`
    `WHERE S1.birth_year <= all(`
    `SELECT S2.birth_year FROM Student S2`
    `WHERE S2.birth_year is not null`
    `)`
  - The `ALL` operator says that the operation has to be satisfied for all entries in the parentheses
  - Similarly, there is an `ANY` operator saying that the operation has to be satisfied at least for one entry in the parentheses

Exists vs. All, Any

- On the previous slide, we see how we can rewrite a query using `NOT EXISTS` as a query using `ALL`
- In general, most queries using `EXISTS` and `NOT EXISTS` can be rewritten by using `ANY` and `ALL`

Example: Union



- *Write together names of all students and subjects.*
- ```
  SELECT name FROM Student
  UNION ALL
  SELECT name FROM Subject
  ```
  ```

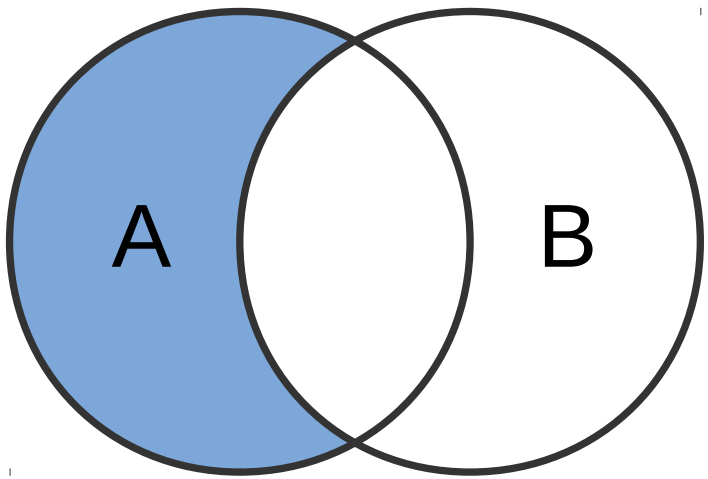
Example: Intersect



- *Find all students who study or studied both subjects with suIDs 1 and 5.*
  - ```
    SELECT St.name FROM Student St
    JOIN Studies Ss ON St.stID = Ss.stID
    WHERE Ss.suID = 1
    INTERSECT
    SELECT St.name FROM Student St
    JOIN Studies Ss ON St.stID = Ss.stID
    WHERE Ss.suID = 5
    ```
    ```

Example: Intersect



- *Find all students who study or studied both subjects with suIDs 1 and 5.*
  - There is another way to solve it using IN/EXISTS subqueries
  - or `HAVING count(distinct Ss.suID)`
  - or several others ...

Example: Except (Difference)



- *Find suIDs of all subjects which are studied only by students born after 1985.*
  - ```
    SELECT suID FROM Studies
    EXCEPT
    SELECT suID FROM Studies Se, Student St
    WHERE Se.stID = St.stID AND St.birth_year <= 1985
       AND St.birth_year is NULL
    ```
  - $\pi_{suID}(Studies) -$
    $\pi_{suID}(Student \bowtie_{birth\_year<=1985} Studies)$
    ```

Difference

- There are many ways how to express difference in SQL
  - Outer join + IS NULL predicate
  - NOT IN/NOT EXISTS
  - EXCEPT (MINUS)
- The most important part is to *identify the sets we use during the difference*



Set Operations

- There are database systems that support just UNION operator
- Set operations are sometimes not optimized very well

Subqueries following FROM

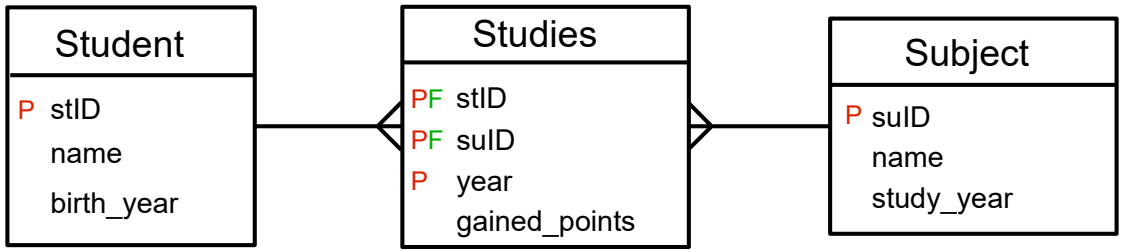
SELECT $A_1, \ldots, A_n$
FROM R ← instead of name of a concrete relation
WHERE *condition* we define a subquery

Subqueries following FROM

SELECT $A_1, \ldots, A_n$
FROM (SELECT ... FROM ...) R
WHERE *condition*

- Subqueries can be useful when we need to exploit the result of some query for further processing
- Sometimes queries nested into the FROM clause are called inline views

Example: Subqueries following FROM



- *Find an average number of students per subject.*

```
SELECT AVG(R.studentCount) FROM (
  SELECT COUNT(*) studentCount
  FROM Subject su
  JOIN Studies ss ON P.suID = S.suID
  GROUP BY P.suID
) R
```

- We handle subquery result like it is a table

How to handle more complex queries

- There are three rules that may simplify the SQL debuging
  - Try to split the query into smaller pieces
  - Complete independent nested queries first
  - Use small data where the result is known

References

- Course home pages `http://dbedu.cs.vsb.cz`