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

- Data types
- SQL DDL
  - Create Table
  - Alter Table
  - Create Index
- SQL DML

Data types

- Database systems implement several cathegories of data types:
  - Integers
  - Floats
  - Strings
  - Date and time
  - Binary strings & BLOBs (large objects)
- Different database systems often implement some data types differently (data types' names)
- Therefore, in what follows, we will focus on SQL Server 2012/16

Data types

- Database systems implement several cathegories of data types:
  - Integers
  - Floats
  - Strings
  - Date and time
  - Binary strings & BLOBs (large objects)
- Different database systems often implement some data types differently (data types' names)
- Therefore, in what follows, we will focus on SQL Server 2012/16

Table creation

- The statement

```
CREATE TABLE table (
     attr1 int,
     attr2 date
)
```

  creates a table with an integer attribute `attr1` and an attribute `attr2` of the type date

- In the second part of this lecture, we will return to this statement, nevertheless, as for data types, the above awareness is sufficient

Numbers

- `int` (4 byte) - integer number
- `bit` - can gain these values: 0, 1, or NULL
- `decimal(d,p)/numeric(d,p)` - numbers with specified number of digits (d) and precision after the decimal point (p)
- `float` (8 byte) - numbers with decimal point
- In the previous examples, we often worked with integer data types

Float

- IEEE 754 standard
- $sign * mantisa * 2^{exponent}$
- There can be rounding issues:
  - `SELECT 1.0 / 3 * 3`

Strings

- Strings can have
  - fixed length
  - variable length
- Moreover, a string can be coded by
  - 8-bit code
  - UNICODE

	fixed length	variable length
8-bit code	`char[n]`	`varchar[n]`
UNICODE	`nchar[n]`	`nvarchar[n]`

Strings - Variable vs. Fixed

- Fixed length strings
  - – Can occupy more space
  - \+ Fast value update
- Variable length strings needs to ‘create’ a space during a value update

Overflow Pages

- Most DBMS don't allow a tuple to exceed a size of a page (4 - 16kB)
- SQL Server requires a fixed size attributes of a row to fit into a page
- However, variable length values can exceed the page size
- Such data are stored separately in a *overflow pages*:
  - PostgreSQL: TOAST
  - MySQL: Overflow
  - SQL Server: Overflow

Strings and functions

- Many functions (often specific for particular database systems) relate to strings: `http://msdn.microsoft.com/en-us/library/ms181984.aspx`
- You can take a look at a description of basic functions as:
  - `CONCAT` - concatenation
  - `LEN` - length of a string
  - `LOWER` - switches all letters to lower cases
  - `LTRIM/RTRIM` - skips all left/right blanks of a string
  - `STR` - transfers a string to a number
  - `SUBSTRING` - returns a substring

Strings and functions

- Many functions (often specific for particular database systems) relate to strings: `http://msdn.microsoft.com/en-us/library/ms181984.aspx`
- You can take a look at a description of basic functions as:
  - `CONCAT` - concatenation
  - `LEN` - length of a string
  - `LOWER` - switches all letters to lower cases
  - `LTRIM/RTRIM` - skips all left/right blanks of a string
  - `STR` - transfers a string to a number
  - `SUBSTRING` - returns a substring

Data type change

- Some data type changes happen implicitly (usually numeric data types)

```
CREATE TABLE table (
     attr1 int,
     attr2 float
)

INSERT INTO table VALUES (2, NULL)

UPDATE table SET attr2 = attr1
```

- The assignment runs automatically without any need to change `int` to `float` (and the other way around)

CAST

- Sometimes, the implicit data type change is not defined or is unclear from context
- Then it is necessary to change explicitly the data type by using the CAST statement

```
CAST (expression AS data_type)
```

- *Find an average birth year of students.*

```
SELECT AVG(CAST (birth_year as FLOAT))
FROM Student
```

CAST

- Sometimes, the implicit data type change is not defined or is unclear from context
- Then it is necessary to change explicitly the data type by using the CAST statement

```
CAST (expression AS data_type)
```

- *Find an average birth year of students.*

```
SELECT AVG(CAST (birth_year as FLOAT))
FROM Student
```

Date and time

- We distinguish the following time data types:
  - those complying with time zones (`datetimeoffset`)
  - those without any information about time zones (`date`, `time`, `datetime2`)
- We will be concerned only with data types without time zones
- Date and time data types are sometimes confusing even for experienced programmer

Date and time

- `date` (3 byte) - stores a date in an implicit format YYYY-MM-DD
- `datetime2` (6-8 byte) - stores both date and time in an implicit format YYYY-MM-DD HH:MM:SS
- `time` (5 byte) - stores a time in an implicit format HH:MM:SS

- `datetime2` is an extension of `datetime`; `datetime` is implemented in SQL Server only to keep the backward compatibility

Dates and functions

- There are many functions for dates and times: `http://msdn.microsoft.com/en-us/library/ms186724.aspx`
- Let us mention only some of them:
  - `SYSDATETIME` - returns current date and time
  - `GETDATE` - returns current date
  - `DAY(attr)` - returns day from the attribute `attr`
  - `MONTH(attr)` - returns month from the attribute `attr`
  - `YEAR(attr)` - returns year from the attribute `attr`

BLOB

- Binary large object (BLOB)
- It is a reference to an external object (File in file system)
- Different data types in different database systems:
  - Oracle - BFILE
  - SQL Server - FILESTREAM
  - MySQL - BLOB
- Most of these database systems has several others data types for large objects (some deprecated, some with specific features)

BLOB

- Database system cannot manipulate the content of an external file
  - NO durability protection
  - NO transaction protection
- Objects above 256KB should be stored as a BLOB [1]
- The major difference is file fragmentation

[1] Sears, Russell, Catharine Van Ingen, and Jim Gray. To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?

Data Definition Language (DDL)

- We will be mainly concerned with these statements:
  - `CREATE TABLE` - creates a table
  - `ALTER TABLE` - modifies a table scheme
  - `DROP TABLE` - deletes a table

The CREATE TABLE statement

- Creation of a table:

```
CREATE TABLE table (attr1 data_type
[constraints, attr2 ..., attr3 ...])
```

- Example:

```
CREATE TABLE Person (
      id int PRIMARY KEY,
      name varchar(20),
      born date
)
```

Constraints

- Contraints that can appear after data type:
  - `NOT NULL` – the attribute cannot gain the NULL value
  - `DEFAULT value` – an implicit value of the attribute
  - `UNIQUE` – values of the attribute have to be unique throughout the table
  - `PRIMARY KEY` – the attribute is a primary key of the table
  - `REFERENCES tab(attr)` – the attribute is a foreign key and it referes to the atribut `attr` of the `tab` table

Constraints - Example 1

- The previous table `Person` with more constraints:

```
CREATE TABLE Person (
      id int PRIMARY KEY,
      name varchar(20) NOT NULL,
      born date DEAFULT NULL,
      cat int REFERENCES cathegory
)
```

- We can see that REFERENCES does not have to specify any attribute - and in this case, a primary key of the referred table (the `cathegory` table in the above example) is considered

Constraints - Example 1

- The previous table `Person` with more constraints:

```
CREATE TABLE Person (
      id int PRIMARY KEY,
      name varchar(20) NOT NULL,
      born date DEAFULT NULL,
      cat int REFERENCES cathegory
)
```

- We can see that REFERENCES does not have to specify any attribute - and in this case, a primary key of the referred table (the `cathegory` table in the above example) is considered

Constraints - Example 2

- The previous table `Person` (an alternative notation):

```
CREATE TABLE Person (
      id int,
      name varchar(20) NOT NULL,
      born date DEAFULT NULL,
      cat int,
      PRIMARY KEY (id),
      FOREIGN KEY (cat) REFERENCES cathegory
)
```

- Constraints concerning the primary and foreign keys are written in separate lines

Constraints - Example 2

- The previous table `Person` (an alternative notation):

```
CREATE TABLE Person (
        id int,
        name varchar(20) NOT NULL,
        born date DEAFULT NULL,
        cat int,
        PRIMARY KEY (id),
        FOREIGN KEY (cat) REFERENCES cathegory
)
```

- Constraints concerning the primary and foreign keys are written in separate lines

Constraints - CHECK

- A constraint can be defined by a logical expression which has to be satisfied for the given attribute
- *Let us demand that any birth year cannot be under 1950.*

```
CREATE TABLE Person (
        ...,
        born date CHECK(YEAR(born) >= 1950),
        ...
)
```

Constraints - CHECK

- A constraint can be defined by a logical expression which has to be satisfied for the given attribute
- *Let us demand that any birth year cannot be under 1950.*

```
CREATE TABLE Person (
        ...,
        born date CHECK(YEAR(born) >= 1950),
        ...
)
```

Table types

- Two major types of tables:
  - Heap
  - Sorted table (primary index)
- Primary key
  - SQL Server – table sorted according to the primary key
  - Oracle, MySQL, PostgreSQL – heap table + secondary index

The ALTER TABLE Statement

- modifies a table scheme
- This modification can have many forms; let us mention at least basic ones:
  - addition of a column (`ALTER TABLE tab ADD ...`)
  - deletion of a column (`ALTER TABLE tab DROP COLUMN ...`)
  - modification of a column (`ALTER TABLE tab ALTER COLUMN ...`)
- This is a syntax used by SQL Server
- Oracle and MySQL use `MODIFY` istead of `ALTER COLUMN`

The ALTER TABLE Statement

- modifies a table scheme
- This modification can have many forms; let us mention at least basic ones:
  - addition of a column (`ALTER TABLE tab ADD ...`)
  - deletion of a column (`ALTER TABLE tab DROP COLUMN ...`)
  - modification of a column (`ALTER TABLE tab ALTER COLUMN ...`)
- This is a syntax used by SQL Server
- Oracle and MySQL use `MODIFY` istead of `ALTER COLUMN`

The ALTER TABLE Statement - examples

- *Let us add a column of the type* `numeric(6,2)` *for a salary to the table* `Person`*.*

```
ALTER TABLE Person
ADD salary numeric(6,2)
```

- *However, the* `salary` *column is not enough now and we need to extend the range by 2 digits.*

```
ALTER TABLE Person
ALTER COLUMN salary numeric(8,2)
```

- *But finally, we decide to delete the* `salary` *column for some reason.*

```
ALTER TABLE Person
DROP COLUMN salary
```

The ALTER TABLE Statement - examples

- *Let us add a column of the type* `numeric(6,2)` *for a salary to the table* `Person`*.*

```
ALTER TABLE Person
ADD salary numeric(6,2)
```

- *However, the* `salary` *column is not enough now and we need to extend the range by 2 digits.*

```
ALTER TABLE Person
ALTER COLUMN salary numeric(8,2)
```

- *But finally, we decide to delete the* `salary` *column for some reason.*

```
ALTER TABLE Person
DROP COLUMN salary
```

The ALTER TABLE Statement - examples

- *Let us add a column of the type* `numeric(6,2)` *for a salary to the table* `Person`*.*

```
ALTER TABLE Person
ADD salary numeric(6,2)
```

- *However, the* `salary` *column is not enough now and we need to extend the range by 2 digits.*

```
ALTER TABLE Person
ALTER COLUMN salary numeric(8,2)
```

- *But finally, we decide to delete the* `salary` *column for some reason.*

```
ALTER TABLE Person
DROP COLUMN salary
```

The DROP TABLE Statement

- Deletion of a table including its definition:

  ```
  DROP TABLE table
  ```

- Deletion of a table is not allowed when some foreign key refers to it

The DROP TABLE Statement

- Deletion of a table including its definition:

  ```
  DROP TABLE table
  ```

- Deletion of a table is not allowed when some foreign key refers to it

Renaming of a Table or a Column

- How to rename a table:

```
RENAME TABLE original_table TO new_table
```

- How ro rename a column:

```
RENAME COLUMN original_column TO new_column
```

- However, these SQL commands are not supported by SQL Server and they are replaced by `sp_rename`:
  - `sp_rename 'puvodni_tabulka', 'nova_tabulka'`
  - `sp_rename 'tab.puvodni_sloupec', 'nova_sloupec'`

Renaming of a Table or a Column

- How to rename a table:

```
RENAME TABLE original_table TO new_table
```

- How ro rename a column:

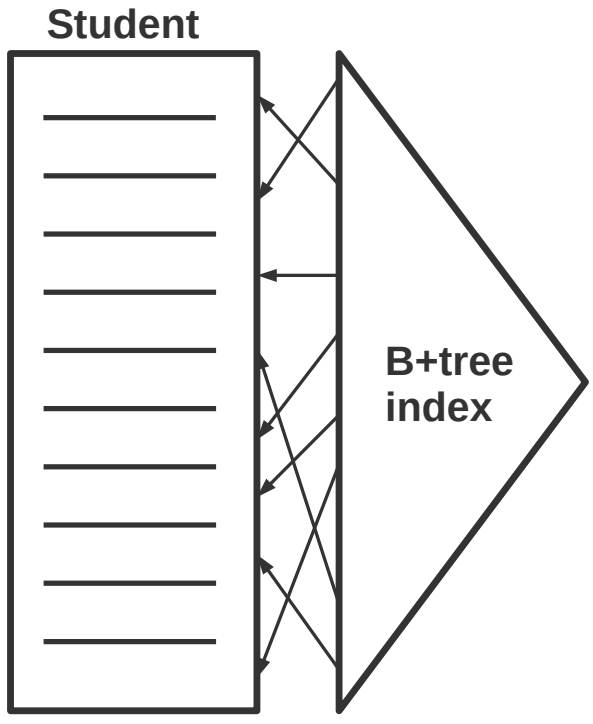
```
RENAME COLUMN original_column TO new_column
```

- However, these SQL commands are not supported by SQL Server and they are replaced by `sp_rename`:
  - `sp_rename 'puvodni_tabulka', 'nova_tabulka'`
  - `sp_rename 'tab.puvodni_sloupec', 'nova_sloupec'`

CREATE INDEX

```
CREATE INDEX index_name ON table(attributes)
```

- It creates a B+tree having `attributes` as the key
- B+tree helps fast seek of a row according to the key
- It is used automatically

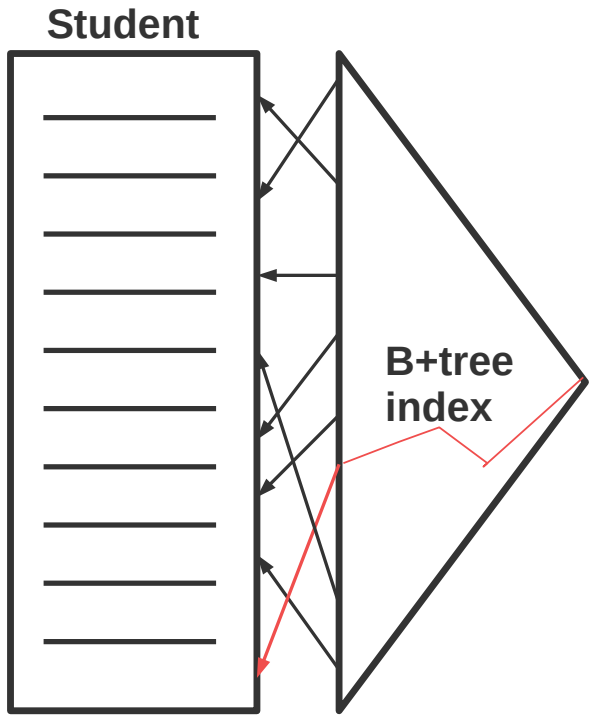


CREATE INDEX

```
CREATE INDEX index_name ON table(attributes)
```

- It creates a B+tree having `attributes` as the key
- B+tree helps fast seek of a row according to the key
- It is used automatically
- ```
  CREATE INDEX ix_student
  ON Student(name)

  SELECT * FROM Student
  WHERE name = 'Tonda'
  ```


  ```

Indexes

- There are many types of indexes:
  - B+tree
  - Hash table
  - Bitmap
  - R-tree and much more

\+ Speed up queries

− Slow down insert/updates/deletes

INSERT - adds records

- Adding one record:

  INSERT INTO *table*
  VALUES ($value_1$, ... , $value_m$)

  where the *table* has m attributes

- Adding records from an existing relation:

  INSERT INTO *table*
  SELECT ... FROM ...

  where the result of the query has to have the same number of attributes as the *table* and also the data types have to mutually correspond

INSERT - adds records

- Adding one record:

  INSERT INTO *table*
  VALUES ($value_1$, ... , $value_m$)

  where the *table* has m attributes

- Adding records from an existing relation:

  INSERT INTO *table*
  SELECT ... FROM ...

  where the result of the query has to have the same number of attributes as the *table* and also the data types have to mutually correspond

DELETE - removes records

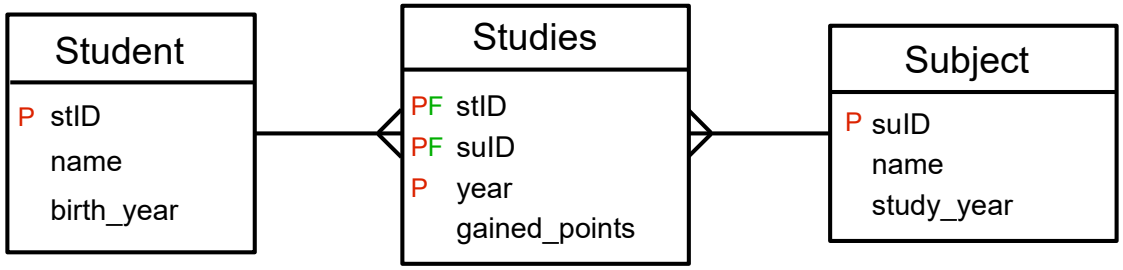
- DELETE FROM *table*
  WHERE *condition*
- The condition can be relatively complicated
- The condition can contain subqueries, aggregate functions, etc.

UPDATE - changes records

- UPDATE *table*
  SET $A_1 = expr_1, \ldots, A_n = expr_n$
  WHERE *condition*

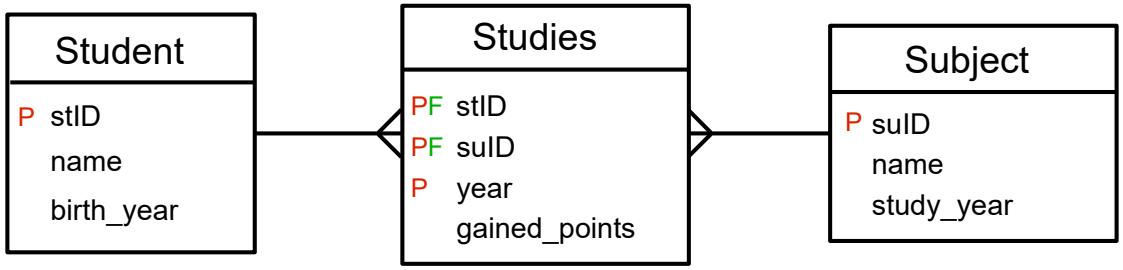
- Similarly to DELETE, the condition can be relatively complicated

Example 1: Simple adding



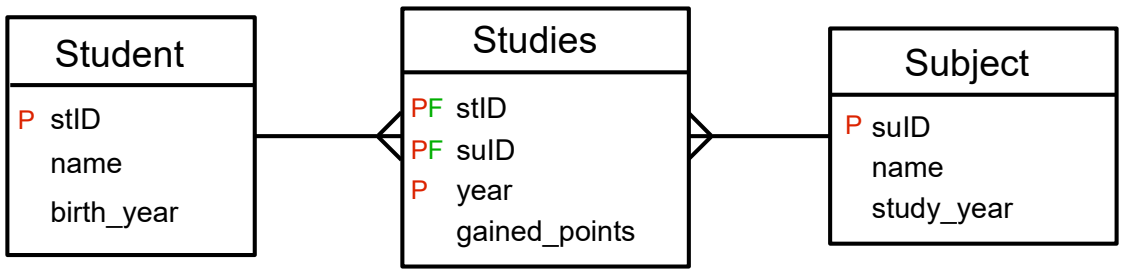
- *Insert into the database a student called Alex.*
  - `INSERT INTO Student VALUES (9, 'Alex', NULL)`
  - The `sID` of the student is chosen as the first available (unused) `sID` in the `Student` table
  - When running this statement repeatedly, we get an error reporting a conflict with the primary key constraint

Example 1: Simple adding



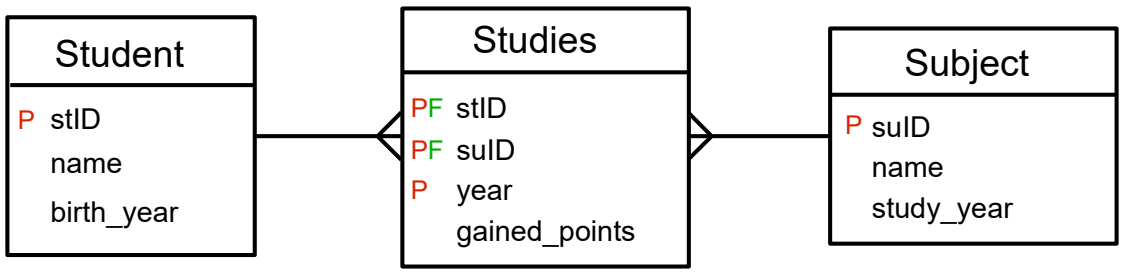
- *Insert into the database a student called Alex.*
  - `INSERT INTO Student`
    `VALUES (9, 'Alex', NULL)`
  - The `sID` of the student is chosen as the first available (unused) `sID` in the `Student` table
  - When running this statement repeatedly, we get an error reporting a conflict with the primary key constraint

Example 2: INSERT and SELECT



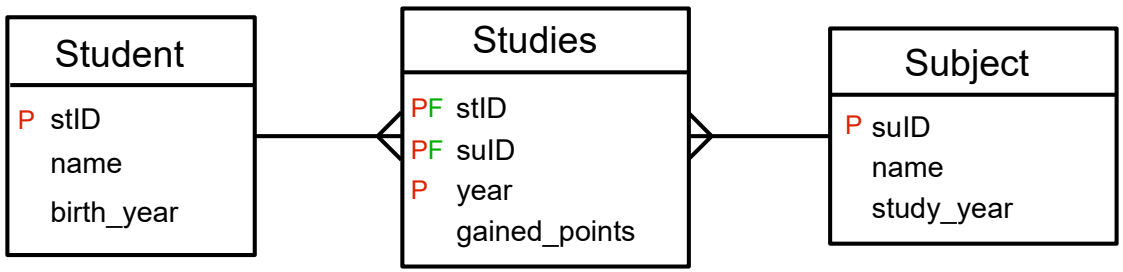
- *Add all students to a subject with suID 4 for the year 2012.*
  - `INSERT INTO Studies`
    `SELECT sID, 4, 2012, NULL FROM Student`
  - We can see that the scheme of the result of the SELECT query has to correspond to the scheme of the `Studies` table
  - Therefore, four attributes of the integer data type have to be in the result

Example 2: INSERT and SELECT



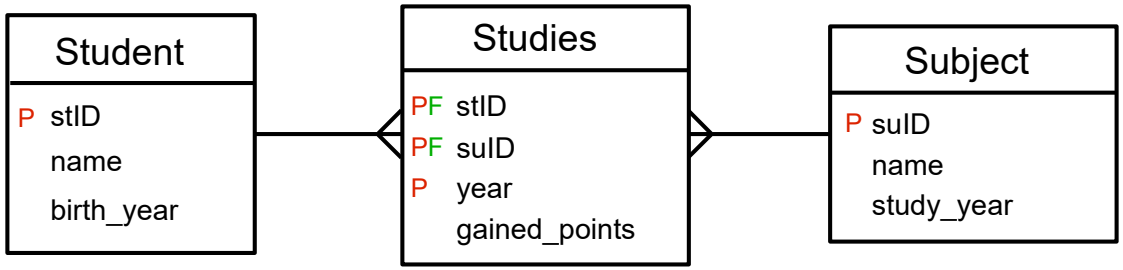
- *Add all students to a subject with suID 4 for the year 2012.*
  - `INSERT INTO Studies`
    `SELECT sID, 4, 2012, NULL FROM Student`
  - We can see that the scheme of the result of the SELECT query has to correspond to the scheme of the `Studies` table
  - Therefore, four attributes of the integer data type have to be in the result

Example 2: INSERT and SELECT



- *Add all students to a subject with suID 4 for the year 2012.*
  - `INSERT INTO Studies`
    `SELECT sID, 4, 2012, NULL FROM Student`
  - We can see that the scheme of the result of the SELECT query has to correspond to the scheme of the `Studies` table
  - Therefore, four attributes of the integer data type have to be in the result

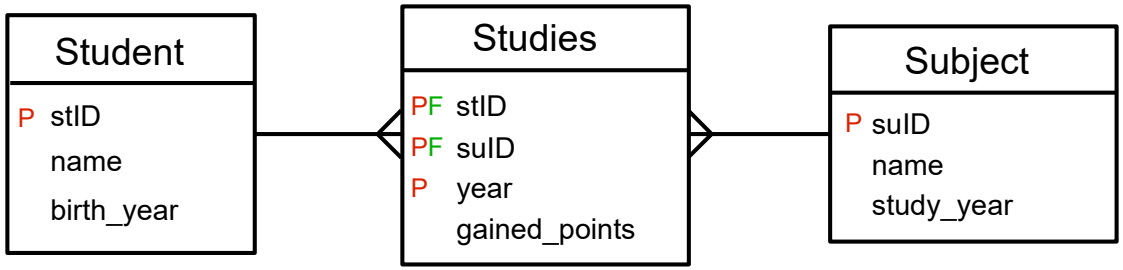
Example 3: INSERT and SELECT



- *To the UDBS subject and for the year 2012, add all students who gained less than 51 points in 2011.*

```
INSERT INTO Studies
SELECT S.sID, S.suID, 2012, NULL
FROM Studies S, Subject Su
WHERE S.suID = Su.suID and Su.name = 'UDBS'
      and S.year = 2011
      and S.gained_points < 51
```

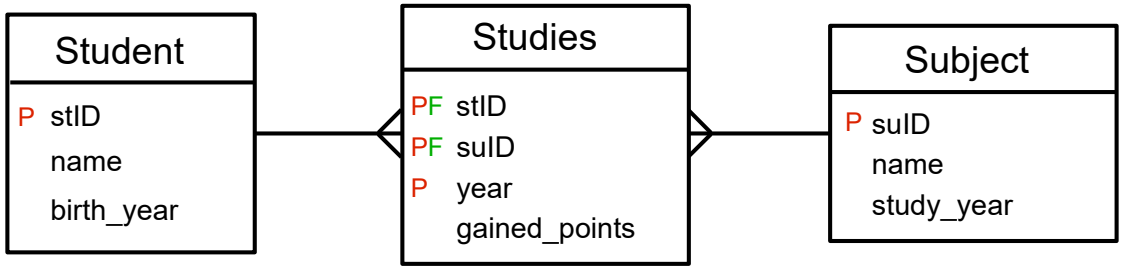
Example 3: INSERT and SELECT



- *To the UDBS subject and for the year 2012, add all students who gained less than 51 points in 2011.*

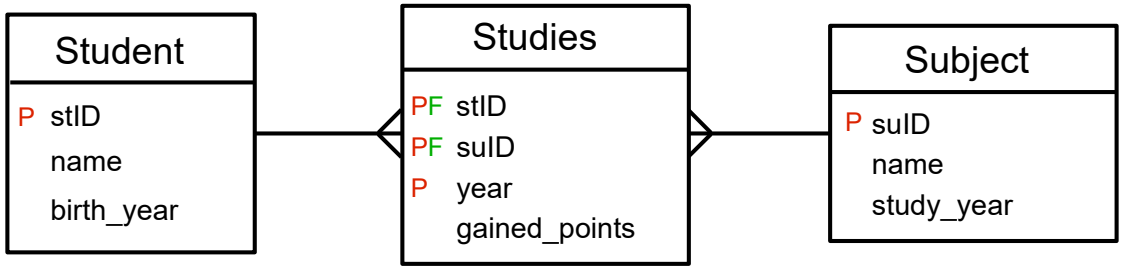
```
INSERT INTO Studies
SELECT S.sID, S.suID, 2012, NULL
FROM Studies S, Subject Su
WHERE S.suID = Su.suID and Su.name = 'UDBS'
      and S.year = 2011
      and S.gained_points < 51
```

Example 4: Simple delete



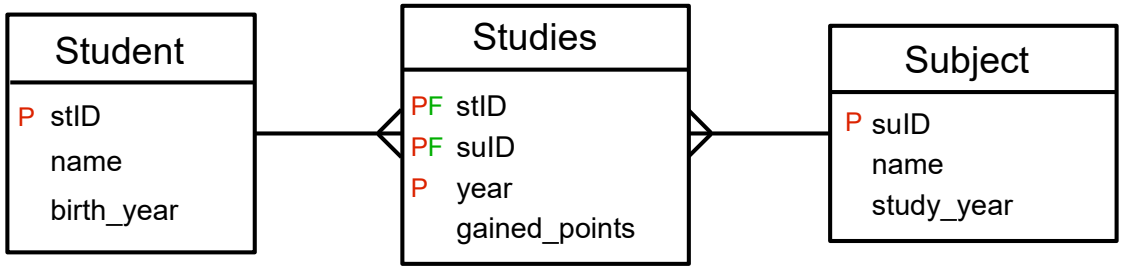
- *Remove a student with the sID 1 from the database.*
  - `DELETE FROM Studies WHERE sID = 1`
  - `DELETE FROM Student WHERE sID = 1`
  - First we have to delete the corresponding records from the `Studies` table and only then we can remove the student from the `Student` table

Example 4: Simple delete



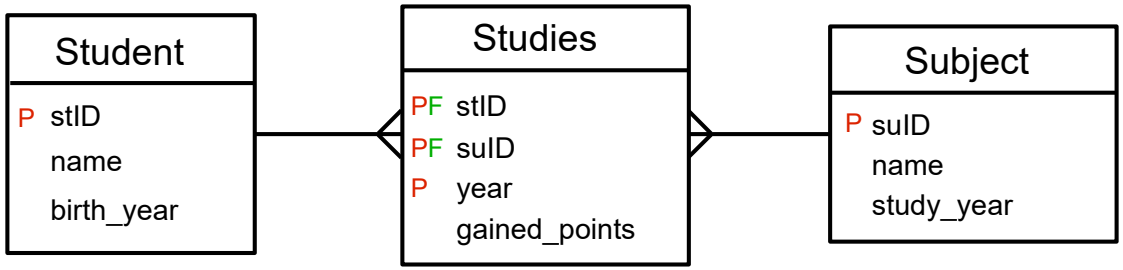
- *Remove a student with the sID 1 from the database.*
  - `DELETE FROM Studies WHERE sID = 1`
  - `DELETE FROM Student WHERE sID = 1`
  - First we have to delete the corresponding records from the `Studies` table and only then we can remove the student from the `Student` table

Example 4: Simple delete



- *Remove a student with the sID 1 from the database.*
  - `DELETE FROM Studies WHERE sID = 1`
  - `DELETE FROM Student WHERE sID = 1`
  - First we have to delete the corresponding records from the `Studies` table and only then we can remove the student from the `Student` table

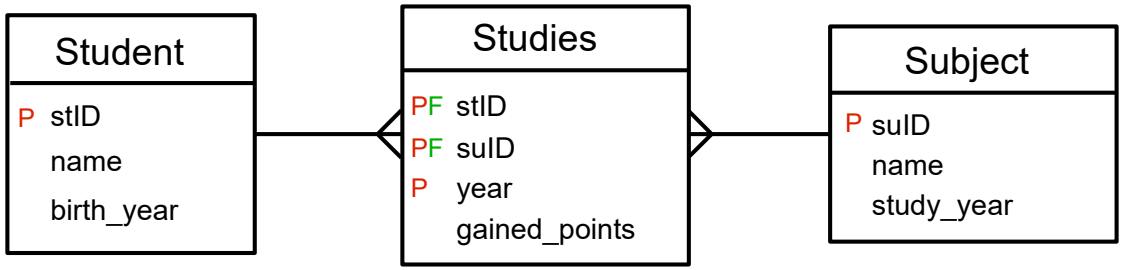
Example 5: DELETE and SELECT



- *Remove all students who gained twice less than 51 points in some subject.*

```
DELETE FROM Studies WHERE sID in
  (SELECT Ss.sID FROM Student S, Studies Ss
  WHERE S.sID = Ss.sID and
    Ss.gained_points < 51
  GROUP BY Ss.sID, suID
  HAVING COUNT(*) >= 2)
```

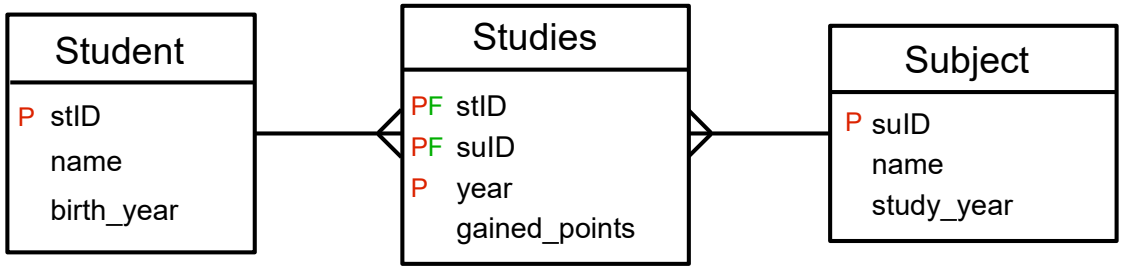
Example 5: DELETE and SELECT



- *Remove all students who gained twice less than 51 points in some subject.*

```
DELETE FROM Studies WHERE sID in
   (SELECT Ss.sID FROM Student S, Studies Ss
   WHERE S.sID = Ss.sID and
      Ss.gained_points < 51
   GROUP BY Ss.sID, suID
   HAVING COUNT(*) >= 2)
```

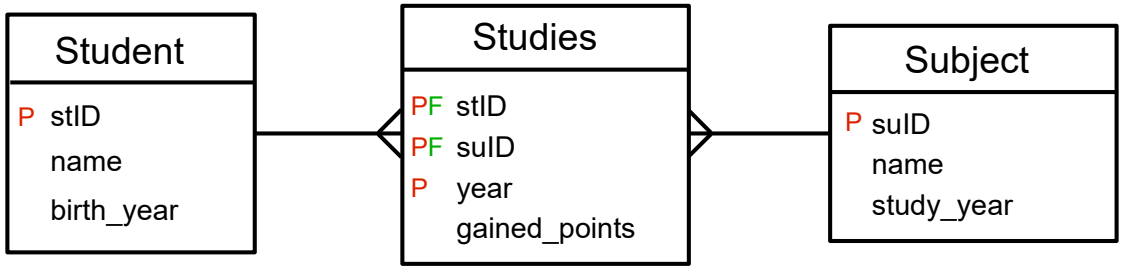
Example 6: Simple update



- *Set all students' gained_points to 90 for all subjects they studied in 2011.*

```
UPDATE Studies
SET gained_points = 90
WHERE year = 2011
```

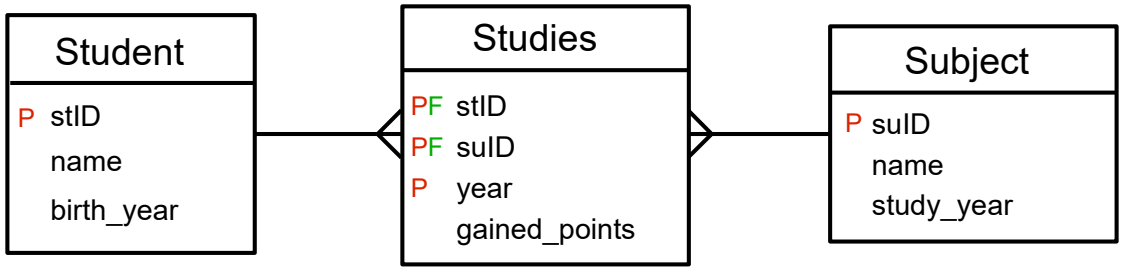
Example 6: Simple update



- *Set all students' gained_points to 90 for all subjects they studied in 2011.*

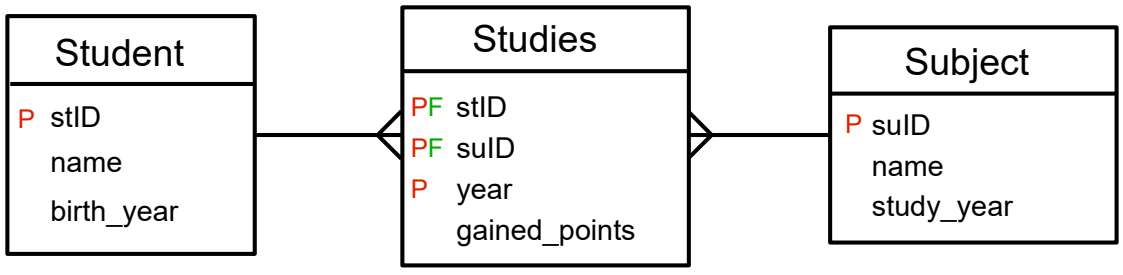
```
UPDATE Studies
SET gained_points = 90
WHERE year = 2011
```

Example 7: UPDATE and SET with SELECT



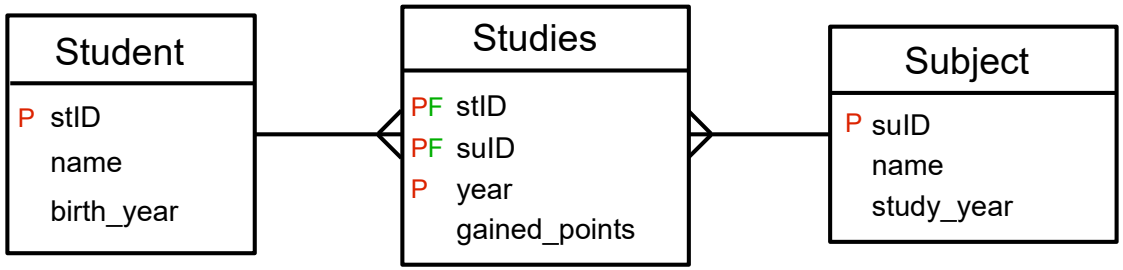
- *For all subjects studied in 2011 and all students, set gained_points to the minimum of points gained (among all students) in 2010.*
  - ```
    UPDATE Studies
    SET gained_points =
        (SELECT MIN(gained_points) FROM Studies
        WHERE year = 2010)
    WHERE year = 2011
    ```
  - Note that the second condition corresponds to the UPDATE statement, while the first one is a part of SELECT
    ```

Example 7: UPDATE and SET with SELECT



- *For all subjects studied in 2011 and all students, set gained_points to the minimum of points gained (among all students) in 2010.*
  - ```
    UPDATE Studies
    SET gained_points =
          (SELECT MIN(gained_points) FROM Studies
          WHERE year = 2010)
    WHERE year = 2011
    ```
  - Note that the second condition corresponds to the UPDATE statement, while the first one is a part of SELECT
    ```

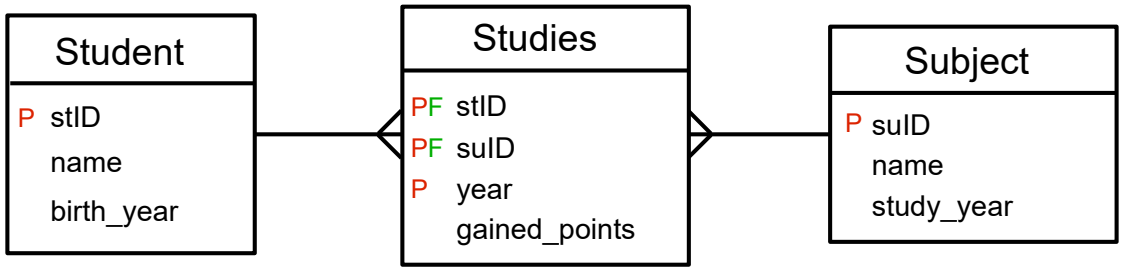
Example 8: UPDATE and SET with alias



- *For all subjects studied in 2011 and all students, set gained_points to the minimum of points gained by the corresponding student in 2010.*

```
UPDATE s1
SET s1.gained_points =
    (SELECT MIN(gained_points) FROM Studies
    WHERE year = 2010 and sID = s1.sID)
FROM Studies s1
WHERE s1.year = 2011
```

Example 8: UPDATE and SET with alias



- *For all subjects studied in 2011 and all students, set gained_points to the minimum of points gained by the corresponding student in 2010.*

```
UPDATE s1
SET s1.gained_points =
     (SELECT MIN(gained_points) FROM Studies
     WHERE year = 2010 and sID = s1.sID)
FROM Studies s1
WHERE s1.year = 2011
```

DELETE, UPDATE, and foreign keys

- In some cases, removing or changing records can be forbidden by the database system
- For example, if we run the statement

  ```
  DELETE FROM Studies
  ```

  we get the following error report in the SQL Server:
  The DELETE statement conflicted with the REFERENCE constraint ...
- We cannot remove students if there are records about their study in the `Studies` table which refer to them
- So first it is necessary to remove the corresponding records from the `Studies` table
- The above holds analogously also for UPDATE

References

- W3C Schools, SQL Tutorial.
  `http://www.w3schools.com/sql/default.asp`
- Jennifer Widom. Introduction to Databases.
  `https://www.coursera.org/course/db`
- Course home pages `http://dbedu.cs.vsb.cz`