Introduction to SQL

- During this week we will learn:
  - How to use a graphical interface to perform database operations
  - How to create database tables using SQL
  - How to define primary and foreign key constraints using SQL
  - How to insert data into a table
  - How to write simple database queries using SQL

SQL

- *SQL* is the standard database language for relational databases. With SQL we can:
    - Create the database and table structures
    - Perform insertion, modification, and deletion of data from the tables
    - Perform database queries
- The query operates on tables and builds a result table from one or more tables in the database
- An SQL query is a single statement in which you describe what you want from the database

SQL

- SQL is used with relational database management systems (RDMS), such as *Microsoft SQL Server*, which we will be using during the course
- RDMS software can be running on local computer on a server on the internet
- We can send database queries to a RDMS using e.g. programming interfaces, command line interfaces or graphical interfaces

Communicating with a RDMS

Here's an example on performing a database query in Python programming language:

```python
import psycopg2

# Establish connection with the RDMS
connection = psycopg2.connect(
    # ...
)

cursor = connection.cursor()
# Execute the database query
cursor.execute("SELECT course_code, course_name, credits FROM Course")
courses = cursor.fetchall()
```

Communicating with a RDMS

- During this course we will be using a graphical interface called *SQL Server Management Studio* to communicate with the SQL Server
- With SQL Server Management Studio we can for example inspect and manage database related information, perform database queries and visualize the structure of the database tables

SQL as a data definition language

Create database

- Database is a named collection of tables
- In addition to tables, database holds different kinds of configuration, for example related to access control
- We can create a database with the `CREATE DATABASE` statement

```
CREATE DATABASE University
```

Create table

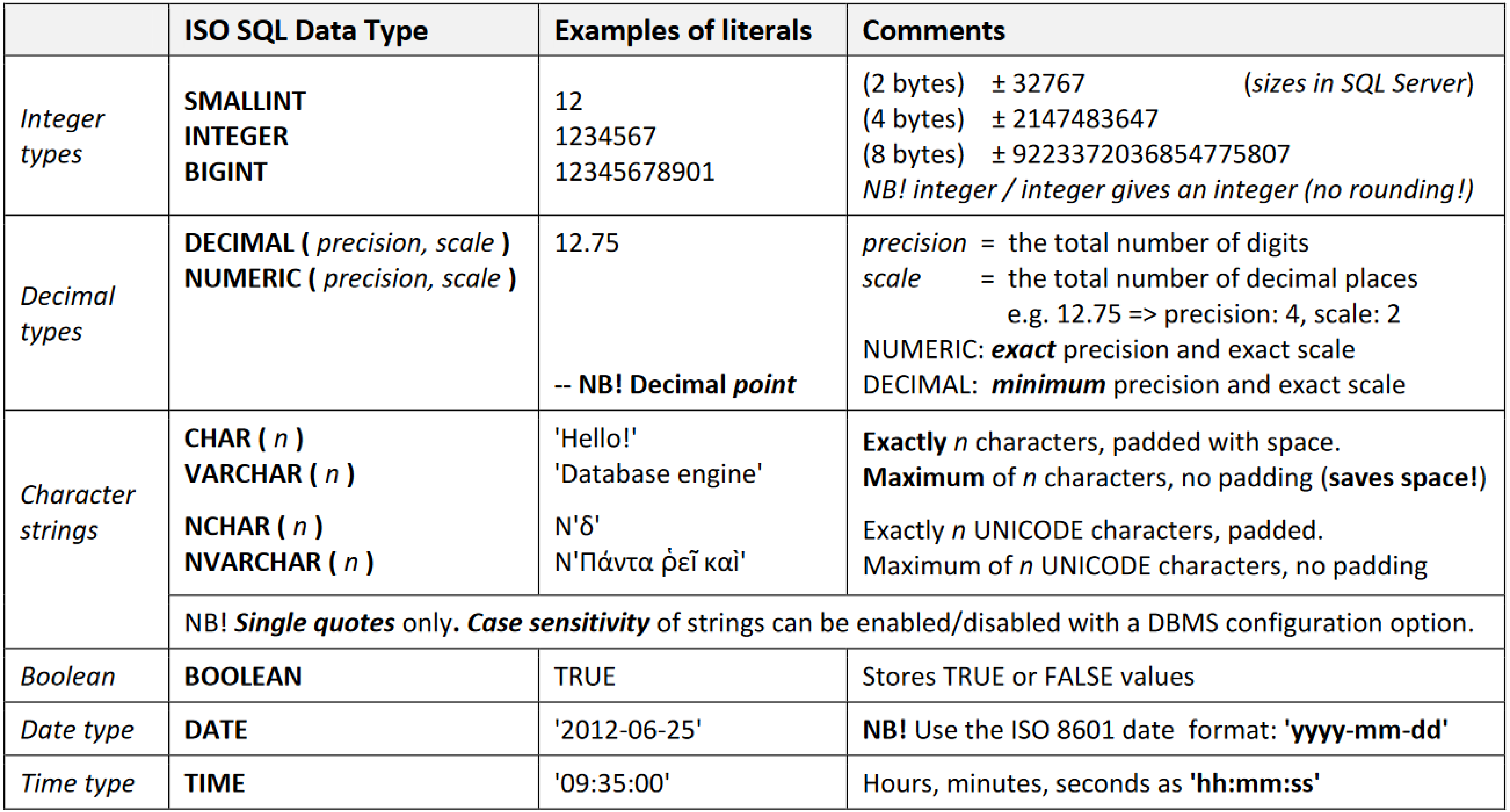
- The actual data of a database lives inside *tables*
- Table has a name and a collection of *columns*
- We can create a table with the `CREATE TABLE` statement

```sql
CREATE TABLE Student (
    student_number INTEGER,
    first_name VARCHAR(50),
    surname VARCHAR(50)
)
```

Create table

- Table and column names should describe the information they store
  - The "Student" table contains rows that represent students
  - The "first_name" column contains the family name of the student
- Table and column names should consist of letters, digits or underscores. They *should not contain whitespace*
- In column names, underscode is commonly used instead of whitespace. For example "first_name" instead of "first name"
- Table names are commonly in *singular format*, for example "Student"
- Each column has a *type* that determines the kind of values the column can have
- For example an `INTEGER` type of column can only contain integer values

Data types



	ISO SQL Data Type	Examples of literals	Comments
Integer types	**SMALLINT** **INTEGER** **BIGINT**	12 1234567 12345678901	(2 bytes) ± 32767 (*sizes in SQL Server*) (4 bytes) ± 2147483647 (8 bytes) ± 9223372036854775807 *NB! integer / integer gives an integer (no rounding!)*
Decimal types	**DECIMAL (** *precision, scale* **)** **NUMERIC (** *precision, scale* **)**	12.75 -- **NB! Decimal *point***	*precision* = the total number of digits *scale* = the total number of decimal places e.g. 12.75 => precision: 4, scale: 2 NUMERIC: ***exact*** precision and exact scale DECIMAL: ***minimum*** precision and exact scale
Character strings	**CHAR (** *n* **)** **VARCHAR (** *n* **)** **NCHAR (** *n* **)** **NVARCHAR (** *n* **)**	'Hello!' 'Database engine' N'δ' N'Πάντα ῥεῖ καὶ'	**Exactly** *n* characters, padded with space. **Maximum** of *n* characters, no padding (**saves space!**) Exactly *n* UNICODE characters, padded. Maximum of *n* UNICODE characters, no padding
	NB! ***Single quotes*** only**.** ***Case sensitivity*** of strings can be enabled/disabled with a DBMS configuration option.		
Boolean	**BOOLEAN**	TRUE	Stores TRUE or FALSE values
Date type	**DATE**	'2012-06-25'	**NB!** Use the ISO 8601 date format: **'yyyy-mm-dd'**
Time type	**TIME**	'09:35:00'	Hours, minutes, seconds as **'hh:mm:ss'**

Example of a table creation

- Let's consider a table named "Country" that stores information about countries
- The table needs the following columns:
    - "country_code", the three characters long code that identifies the country. This is the table's primary key
    - "country_name", the name of the country
    - "population" the number of people living the country
- *What is the SQL statement that creates the "Country" table with the mentioned columns?*

Constraints

- *Constraints* specify rules for the data in a table
- For example `NOT NULL` constraint ensures that a column cannot have a `NULL` (empty) value
- The `NOT NULL` constraint is defined *after the column type* in the `CREATE TABLE` statement

```sql
CREATE TABLE Student (
    student_number INTEGER NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL
)
```

Primary key constraint

- *Primary key* uniquely identifies each row in the table
- *Primary key constraint* prevents duplicate rows to exist for the table
- Primary key constraint is defined with the `PRIMARY KEY` constraint *after the column definitions* in the `CREATE TABLE` statement

```sql
CREATE TABLE Student (
    student_number INTEGER NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,

    -- The primary key is the student_number column
    CONSTRAINT Pk_Student PRIMARY KEY (student_number)
)
```

Foreign key constraint

- *Foreign key* is a column or group columns whose values are required to match those of the primary key of the referenced table
- *Foreign key constraint* prevents foreign key not being matched by a primary key in the referenced table
- Foreign key constraint is defined with the `FOREIGN KEY` constraint *after the column definitions* in the `CREATE TABLE` statement

```
CREATE TABLE Laptop (
    serial_number VARCHAR(10) NOT NULL,
    student_number INTEGER NOT NULL,

    -- The primary key is the serial_number column
    CONSTRAINT Pk_Laptop PRIMARY KEY (serial_number),
    --- The foreign key student_number references the primary key student_number in the Student table
    CONSTRAINT Fk_Student FOREIGN KEY (student_number)
    REFERENCES Student(student_number)
)
```

Drop table

- We can delete a table in the database with the `DROP TABLE` statement
- ⚠️ This operation will delete all rows in the table

```
DROP TABLE Laptop
```

SQL as a data manipulation language

Insert

- We insert a new row into a table by defining the table name and the values for the columns
- A new row can be inserted with the `INSERT INTO` statement
- ⚠️ String literals are defined with *single quotes*, for example `'Kalle'`

```sql
INSERT INTO Student (student_number, first_name, surname) VALUES (1, 'Kalle', 'Ilves')
```

Insert

- Constraits are checked once a new row if inserted
- For example if `NOT NULL` constraint of a column is violated, there will be an error

```sql
-- ❌ surname columns has a NOT NULL constraint, omitting it will cause an error
INSERT INTO Student (student_number, first_name) VALUES (1, 'Kalle')
```

Select

- The `SELECT` statement is used to select rows from a table
- With the `SELECT` statement we define a group of columns we want to select the data from and the name of the target table
- The result is a result table containing the rows from the target table with the specified columns

```sql
SELECT first_name, surname FROM Student
```

Where

- We can filter the selected rows of a table with a `WHERE` clause
- With the `WHERE` clause we define a condition which the selected rows should satisfy
- The result table only contains the rows that satisfy the condition

```sql
SELECT first_name, surname FROM Student WHERE first_name = 'Matti'
```

Comparison operators

- The `WHERE` clause conditions support similar *comparison operators* as many programming languages

```
WHERE first_name = 'Matti' -- equal to. ⚠️ Note, just a single = symbol
WHERE first_name <> 'Matti' -- not equal to
WHERE age > 18 -- greater than
WHERE age >= 30 -- greater than or equal
WHERE age < 18 -- less than
WHERE age <= 30 -- less than or equal
```

Logical operators

- Comparisons can be combined with *logical operators* to achieve conditions such as "age is greater than 18 *and* age is less than 30"

```
WHERE age > 18 AND age < 30  -- AND operator
WHERE first_name = 'Matti' OR first_name = 'Kaarina' -- OR operator
WHERE NOT age < 18 -- NOT operator
```

Logical operators

- We can use brackets to determine in which order the logical operators should be applied

```
WHERE (skill = 1 OR skill = 2) AND salary > 10000
```

Order by

- The order of result table's row is unpredictable, it might not bee the same each time we execute the query
- We can use the `ORDER BY` clause to define in which order we want the rows to be in the result table
- The sorting is done based on columns

```sql
SELECT course_name, credits
FROM Course
ORDER BY credits -- rows will be sorted by the credits column's value
```

Order by

- Table might contain multiple rows with the same value in the column used in the `ORDER BY` clause
- To determine the order of such rows we can provide multiple columns to the `ORDER BY` clause

```sql
SELECT course_name, credits
FROM Course
-- when the credits is the same, the course_name is used to determine the order
ORDER BY credits, course_name
```

Order by

- The `ORDER BY` sorts the records in *ascending order* (smallest value first) by default
- We can change the order by using either `ASC` (ascending order) or `DESC` (descending order) keyword

```
SELECT course_name, credits
FROM Course
-- use descending order for credits and ascending order for course_name
ORDER BY credits DESC, course_name ASC
```

Summary

- We can create database tables using the `CREATE TABLE` statement
- `PRIMARY KEY` constraint is used to define the table's primary key
- `FOREIGN KEY` constraint is used to define a foreign key referencing primary key column of another table
- `INSERT INTO` statement is used to insert a new row for the table
- `SELECT` statement is used to select rows from a table
- `WHERE` clause can be used to filter the rows of a table
- We can use comparison and logical operators to define a condition for the `WHERE` clause, for example `WHERE first_name = 'Kalle' OR first_name = 'Elina'`
- We can use `ORDER BY` clause to determine the order of rows in the result table