CSE 3104
DATABASE LAB

SQL Constraints

- Constraints are used to limit the type of data that can go into a table.
- Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).
- We will focus on the following constraints:
  - **NOT NULL**
  - **UNIQUE**
  - **PRIMARY KEY**
  - **FOREIGN KEY**
  - **CHECK**
  - **DEFAULT**

SQL NULL Values

- If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.
- NULL values are treated differently from other values.
- NULL is used as a placeholder for unknown or inapplicable values.

SQL Working with NULL Values

- Look at the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari		Stavanger

- Suppose that the "Address" column in the "Persons" table is optional. This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

SQL NOT NULL Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.
- The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.
-
```
CREATE TABLE table_name
    (
    column_name1 data_type NOT NULL,
    column_name2 data_type NULL,
    column_name3 data_type NOT NULL,
    );
```

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

- SYNTAX:
-
```
CREATE TABLE table_name
    (
    column_name1 data_type NOT NULL,
    column_name2 data_type NULL,
    column_name3 data_type NOT NULL,
    UNIQUE (column_name1)
    );
```
-
```
CREATE TABLE table_name
    (
    column_name1 data_type NOT NULL UNIQUE,
    column_name2 data_type NULL,
    column_name3 data_type NOT NULL,
   );
```

```
CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint on ALTER TABLE

- To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD UNIQUE (P_Id)
```

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Each table should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

- SYNTAX:
- ```
  CREATE TABLE table_name
      (
      column_name1 data_type NOT NULL,
      column_name2 data_type NULL,
      column_name3 data_type NOT NULL,
      PRIMARY KEY (column_name1)
       );
  ```
- ```
  CREATE TABLE table_name
      (
      column_name1 data_type NOT NULL PRIMARY KEY,
      column_name2 data_type NULL,
      column_name3 data_type NOT NULL,
     );
  ```

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

Note: In the example above there is only ONE PRIMARY KEY (pk_PersonID). However, the value of the pk_PersonID is made up of two columns (P_Id and LastName).

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY(P_Id,LastName)
)
```

SQL PRIMARY KEY Constraint on ALTER TABLE

- To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD PRIMARY KEY (P_Id)
```

- To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons
ADD CONSTRAINT pK_ PRIMARY KEY (P_Id,LastName)
```

Create table & Insert Data

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

O_Id	OrderNo
1	34455
2	33445
3	12345

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

SQL FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents that invalid data form being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
- Syntax

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

SQL CHECK Constraint

- SQL CHECK Constraint on CREATE TABLE
- The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.
- Syntax

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL CHECK Constraint

- SQL CHECK Constraint on CREATE TABLE
- The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.
- Syntax

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL CHECK Constraint on ALTER TABLE

To create a CHECK constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (P_Id>0)
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
```

To DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT chk_Person
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK chk_Person
```

SQL DEFAULT Constraint

- The DEFAULT constraint is used to insert a default value into a column.The default value will be added to all new records, if no other value is specified.
- The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:
- **CREATE TABLE Persons**
  (
  P_Id int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255) DEFAULT 'Sandnes'
)

SQL DEFAULT Constraint on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'SANDNES'
```

SQL Server / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'SANDNES'
```

SQL AUTO INCREMENT Field

Syntax for SQL Server

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sorts the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

```
SELECT * FROM Persons
ORDER BY LastName
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

The TOP Clause

- The TOP clause is used to specify the number of records to return.
- The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.
- **Note:** Not all database systems support the TOP clause.
- SQL Server Syntax

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

SQL TOP Example

- The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tove	Bakken 2	
5	Tjessem	Jakob		

SQL TOP Example

- Now we want to select only the two first records in the table above.
- We use the following SELECT statement:

```
SELECT TOP 2 * FROM Persons
```

- The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timotervn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

The LIKE Operator

- The LIKE operator is used to search for a specified pattern in a column.
- SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

LIKE Operator Example

- The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tove	Bakken 2	
5	Tjessem	Jakob		

LIKE Operator Example

- Now we want to select the persons living in a city that starts with "s" from the table above.
- We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 's%'
```

- The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist] or [!charlist]	Any single character not in charlist

SQL Wildcard Examples

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 'sa%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Next, we want to select the persons living in a city that contains the pattern "nes" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%nes%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Using the _ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "la" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName LIKE '_la'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

```
SELECT * FROM Persons
WHERE LastName LIKE 'S_end_on'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

The IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

- Now we want to select the persons with a last name equal to "Hansen" or "Pettersen" from the table above.
- We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')
```

The BETWEEN Operator

- The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

- Now we want to select the persons with a last name alphabetically between "Hansen" and "Pettersen" from the table above.
  - **SELECT * FROM Persons**
    WHERE LastName
    BETWEEN 'Hansen' AND 'Pettersen'