Creating Tables

```
CREATE TABLE <Table_name>
(
      Column_name Data_type [ NULL | NOT NULL ]
      [ IDENTITY ( Initial_value , Increment ) | DEFAULT (Constant | Expression) ]
      [ Constraint ] [ Constraint ] … ,

      Column_name Data_type [ NULL | NOT NULL ]
      [ IDENTITY ( Initial_value , Increment ) | DEFAULT (Constant | Expression) ]
      [ Constraint ] [ Constraint ] … ,

      …

      [ Constraint ] ,
      [ Constraint ] ,
      …
)
```

Data Types

1. Numerical types :
   a) Whole numbers :
      - INTEGER
      - INT (4 octets)
      - SMALLINT(2 octets)
      - TINYINT(1 octet)

   b) Real numbers :
      - REAL (4 octets)
      - FLOAT(8 octets)
      - DECIMAL(nbDigits,nbDecimals)
      - MONEY
2. Character types :
   - CHAR(n) : a string of characters of fixed length. Each character occupies 1 octet, ***n*** denotes the maximum length of the string, with a hard max of 255 characters maximum.
   - VARCHAR : a string of characters with a variable length.
3. Date types :
   a) DateTime : dates between the 1st January 1753 and the 31 December 9999 (8 octets).

   b) SmallDateTime : dates between January 1st 1900 and December 1st 2079 (4 octets).

Data Integrity

The concept of data integrity deals with the consistency and accuracy of the data being stored in the tables of our database. The following keywords allow us to implement the concepts of data integrity :

1. Entity Integrity
   a) NULL, NOT NULL
   b) PRIMARY KEY
   c) UNIQUE
   d) IDENTITY
2. Referential Integrity
   a) FOREIGN KEY
3. Domain Integrity
   a) DEFAULT
   b) CHECK

NULL, NOT NULL

- **NULL** : Allows the values of a column to not be defined when inserting a row.
- **NOT NULL** : Forces the values of a column to be defined when inserting a row.

Example :

```
CREATE TABLE table1
(
    code int            NOT NULL,
    name char(30)       NOT NULL,
    city char(10)       NULL,
)
```

NULL is not a valid value for the columns "code" and "name"

NULL is a valid value for the column "city"

PRIMARY KEY

The PRIMARY KEY constraint defines which column(s) in your table function as its primary key.

```
CREATE TABLE table1 (
      code int NOT NULL ,
      name char(30) NOT NULL,
      city char(10) NULL,
      CONSTRAINT PK_code PRIMARY KEY CLUSTERED (code)
)
```

- The previous CREAT TABLE statement defines the “code” column as the primary key of the table “table1”. It must be assigned and no other values in the “code” column can share the same value.
- CLUSTERED ensures that the table indices will be ordered based on the values in the “code” column.

The PRIMARY KEY constraint can also be declared inline when defining a column :

```
CREATE TABLE table1 (
            code int NOT NULL PRIMARY KEY CLUSTERED,
            name char(30) NOTNULL,
            city char(10) NULL
)
```

UNIQUE

The UNIQUE constraint ensures that all values in a column are different from one another.

```
CREATE TABLE table1(
            code int NOT NULLPRIMARY KEYCLUSTERED,
            SIS char(9),
            name char(30) NOT NULL,
            CONSTRAINT Unique_SIS UNIQUE NONCLUSTERED (SIS)
)
```

- The above CREATE TABLE statement defines SIS as a column with unique values, and ensures that no values of the column will share the same value.
- NONCLUSTERED is used to define a non ordered index for the “SIS” column

The UNIQUE constraint can also be declared inline when defining a column :

```
CREATE TABLE table1(
            code int NOT NULL  PRIMARY KEY CLUSTERED,
            SIS char(9) UNIQUE NONCLUSTERED,
            name char(30) NOT NULL
)
```

IDENTITY

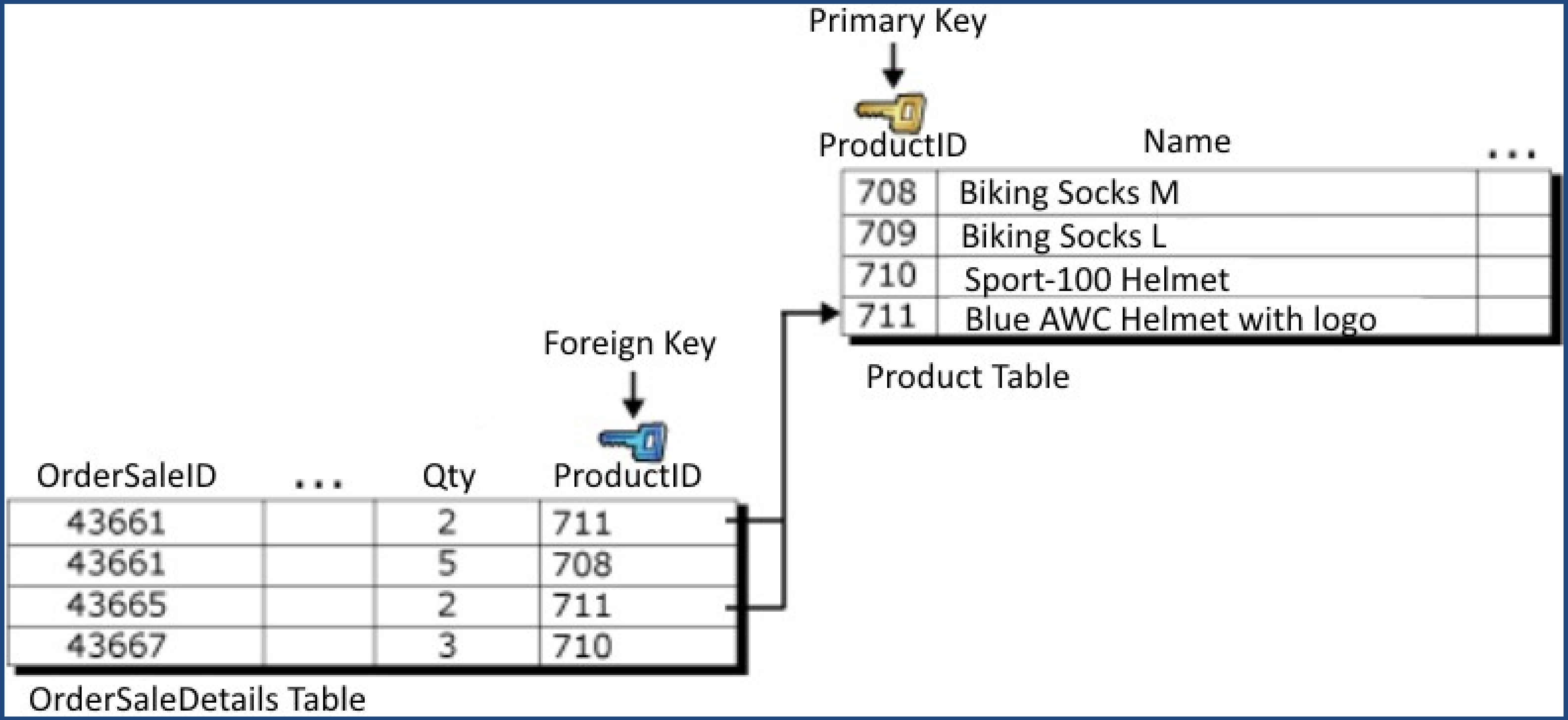
The **IDENTITY(Initial_value, Increment)** property defines a starting value ("**Initial_value**") for the values of a column, and ensure that these values will be automatically incremented by the "**Increment**" amount.

```
CREATE TABLE table1  (
     code int NOT NULL IDENTITY(1,10),
     name char(30) NOT NULL,
     city char(10) NULL
)
```

The above code causes values in the "code" column to be initially set to 1, but automatically increment in steps of 10.

FOREIGN KEY

- We use referential integrity to ensure that the data we are storing in different tables remains accurate and consistent.
- The FOREIGN KEY constraint maintains referential integrity by ensuring that any column declared a foreign key can only store a primary or unique key of another table.
- The FOREIGN KEY constraint ensures that the column has no references to non-existent values and that, if the value of a key changes, all the references made to that key are updated accordingly.



Product Table

ProductID	Name	. . .
708	Biking Socks M	
709	Biking Socks L	
710	Sport-100 Helmet	
711	Blue AWC Helmet with logo	

OrderSaleDetails Table

OrderSaleID	. . .	Qty	ProductID
43661		2	711
43661		5	708
43665		2	711
43667		3	710

FOREIGN KEY

The FOREIGN KEY constraint defines one or more columns of a table whose values must correspond to the primary key of another table.

```
CREATE TABLE father(
		fatherNb int NOT NULL  PRIMARY KEY CLUSTERED,
		fatherName char(30) NOT NULL
)
CREATE TABLE child(
		childNb int NOT NULL PRIMARY KEY CLUSTERED,
		childLastName char(30) NOT NULL,
		papaNb int NULL,
		CONSTRAINT FK_SonFather FOREIGN KEY (papaNb) REFERENCES father (fatherNb)
)
```

In the tables defined above, the values in the column child.papaNb must store the same value as a fatherNb being stored in the father table.

The FOREIGN KEY constraint can also be declared inline when defining a column :

```
CREATE TABLE child(
		childNb int NOT NULL  PRIMARY KEY CLUSTERED,
		childLastName char(30) NOT NULL,
		papaNb int NULL FOREIGN KEY REFERENCES father (fatherNb)
)
```

DEFAULT

The DEFAULT constraint defines the default value of a column. If no value is specified for the column when a row is being inserted, then the value of the column will be set to the default value.

```
CREATE TABLE table1(
	num int NOT NULL  PRIMARY KEY  CLUSTERED,
	name char(30) NOT NULL,
	city char (10) DEFAULT('Montréal'),
	birthDate  DateTime  DEFAULT(GETDATE())
)
```

In this example, the value 'Montreal' will be inserted in the 'city' column if no value is specified when the row is inserted.

The value in column “birthDate” will be set to the current date (GETDATE()) if no value is specified when the row is inserted.

CHECK

The CHECK constraint defines a range of allowed values that limits which values can be stored in the column.

```
CREATE TABLE table1 (
      num int NOT NULL  PRIMARY KEY CLUSTERED,
      name char(30) NOT NULL,
      Tel char(10) NULL,
      CONSTRAINT C_tel CHECK (Tel Like “514[0-9][0-9][0-9][0-9][0-9][0-9][0-9]")
)
```

In this example, the "Tel" column can not contain a phone number that does not match the expression Like “514 [0-9] [0-9] [0-9] [0-9] [0- 9] [0-9] [0-9]”; valid phone numbers must start with 514 and consist of a sequence of 10 1-digit numeric characters.

The CHECK constraint can also be declared inline when defining a column :

```
 CREATE TABLE table1 (
      num int NOT NULL  PRIMARY KEY CLUSTERED,
      name char(30) NOT NULL,
      Tel char(10) NULL CHECK (Tel Like "514[0-9][0-9][0-9][0-9][0-9][0-9][0-9]")
   )
```