# How-to: Constraints

## Example 1

### Condition

There is a book for which a price is defined.

```
CLASS Book 'Book';
name 'Name' = DATA ISTRING[50] (Book) IN id;
price 'Price' = DATA NUMERIC[14,2] (Book);
```

We need to do so that it will be impossible to enter prices higher than 100.

### Solution

```
// Option 1
CONSTRAINT price(Book b) > 100
    MESSAGE 'The book price cannot be more than 100 rubles';

// Option 2
CONSTRAINT SET(price(Book b) > 100)
    MESSAGE 'The book price cannot be more than 100 rubles';
```

If a user tries to save a book costing over 100 on any form, the user will see a message with a corresponding text. This message will also contain all objects of the **Book** class for which the constraint is violated. Values of properties from the **id** group will be shown for each object.

Both options are identical from the execution perspective. If the platform does not find any change operator in a constraint, the entire expression is automatically "wrapped" into a **SET** operator.

## Example 2

### Condition

We have an order with a date, ID and a posted/not posted flag.

```
CLASS Order 'Order';
date 'Date' = DATA DATE (Order) IN id;
number 'Number' = DATA INTEGER (Order) IN id;
posted 'Completed' = DATA BOOLEAN (Order) IN id;
```

You need to prohibit the change of the order date.

### Solution

```
CONSTRAINT CHANGED(date(Order o)) AND posted(o)
    MESSAGE 'It is forbidden to change the date of a completed order';
```

## Example 3

### Condition

Identical to **Example 2**.

You need to prohibit the deletion of a posted order.

### Solution

```
CONSTRAINT DROPPED(Order o IS Order) AND PREV(date(o)) < currentDate()
    MESSAGE 'It is forbidden to delete old orders';
```

When an order is deleted, all of its properties will be **NULL**. That is why you need to you use the **PREV** operator to access its property values.

## Example 4

### Condition

Similar to **Example 1** and **Example 2**. Also, the order contains lines with a price and a link to the book.

```
CLASS OrderDetail 'Order line';
order 'Order' = DATA Order (OrderDetail) NONULL DELETE;

book 'Book' = DATA Book (OrderDetail) NONULL;
nameBook 'Book' (OrderDetail d) = name(book(d)) IN id;

price 'Price' = DATA NUMERIC[14,2] (OrderDetail);
```

You need to prohibit the entry of order price values exceeding the price of the book by 10%.

### Solution

```
CONSTRAINT price(OrderDetail d) > price(book(d)) * 1.1
    MESSAGE 'The price in the order cannot exceed the price of the book by 10%';
```

Since the expression contains no change operators, this constraint will be triggered when the price changes for a line, book or book price. Therefore, the user will not be able to change the book price if there have been orders for it with a price lower by up to 10%. If such behavior is not required, you need to explicitly apply change operators to those properties that should trigger constraints upon change.

## Example 5

### Condition

Similar to **Example 4**. Here are added the concept of a customer and the possibility to select books that will be available to the customer.

```
CLASS Customer 'Customer';
name 'Name' = DATA ISTRING[100] (Customer);

in 'On' = DATA BOOLEAN (Customer, Book);

customer 'Customer' = DATA Customer (OrderDetail);
```

We need to prohibit the entry of books that are unavailable to the buyer for the order line.

## Solution

```
CONSTRAINT book(OrderDetail d) AND NOT in(customer(d), book(d))
    CHECKED BY book[OrderDetail]
    MESSAGE 'A book is selected in the order line that is not allowed for the customer';
```

It is important to check that the **book** property for the order line is set because otherwise, the constraint will be applied to all order lines with the yet unselected book. The **CHECKED BY** block adds the filter for the order line on the book selection form in order to avoid violating the defined constraint. This way, the user will only see books available to the buyer.

## Example 6

### Condition

Identical to **Example 4**.

We need to prohibit the entry of books that are unavailable to the buyer for the order line, but only for posted orders.

### Solution

```
// Option 1
CONSTRAINT (CHANGED(book(OrderDetail d)) OR CHANGED(price(d)) OR CHANGED(posted(order(d)))) AND posted(order(d))
           AND price(d) > price(book(d)) * 1.1
           MESSAGE 'The price in the order cannot exceed the price of the book by 10%';

// Option 2
constraintBook (OrderDetail d) =
    (CHANGED(book(d)) OR CHANGED(price(d)) OR CHANGED(posted(order(d)))) AND posted(order(d)) AND price(d) > price(book(d)) * 1.1;
WHEN (GROUP MAX constraintBook(OrderDetail d)) DO {
    MESSAGE 'A book is selected in the order line that is not allowed for the customer by lines: \n' +
            (GROUP CONCAT ('Date ' + date(order(OrderDetail d)) + '; Number ' + number(order(d))) IF constraintBook(d), ',') NOWAIT;
    CANCEL;
}
```

The second scenario is similar to the first one, but lets you modify the message shown to the user and the logic of constraint handling.