Give UML and SQL CREATE TABLE statements that define a relational DB that is roughly equivalent to that defined by this XML DTD.

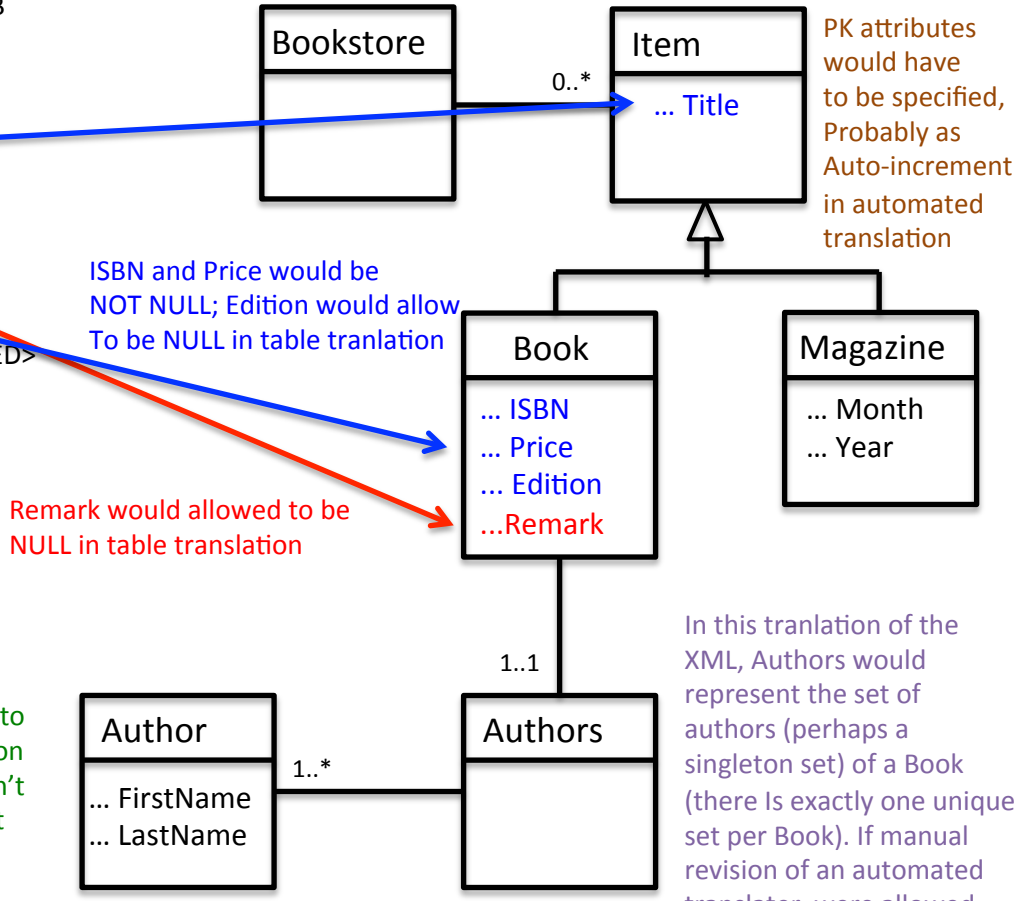
```
<!DOCTYPE Bookstore [
  <!ELEMENT Bookstore (Book | Magazine)*>
  <!ELEMENT Book (Title, Authors, Remark?)>
  <!ATTLIST Book ISBN CDATA #REQUIRED
            Price CDATA #REQUIRED
            Edition CDATA #IMPLIED>
  <!ELEMENT Magazine (Title)>
  <!ATTLIST Magazine Month CDATA #REQUIRED Year CDATA #REQUIRED>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Authors (Author+)>
  <!ELEMENT Remark (#PCDATA)>
  <!ELEMENT Author (First_Name, Last_Name)>
  <!ELEMENT First_Name (#PCDATA)>
  <!ELEMENT Last_Name (#PCDATA)>
]>
```

Give UML and SQL CREATE TABLE statements that define a relational DB that is roughly equivalent to that defined by this JSON Schema.

```
{ "type":"object",
  "properties": {
    "Books": {
      "type":"array",
      "items": {
        "type":"object",
        "properties": {
          "ISBN": { "type":"string", "pattern":"ISBN*" },
          "Price": { "type":"integer",
                  "minimum":0, "maximum":200 },
          "Edition": { "type":"integer", "optional": true },
          "Remark": { "type":"string", "optional": true },
          "Title": { "type":"string" },
          "Authors": {
            "type":"array",
            "minItems":1,
            "maxItems":10,
            "items": {
              "type":"object",
              "properties": {
                "First_Name": { "type":"string" },
                "Last_Name": { "type":"string" }}}}}},
}}
```

Give UML and SQL CREATE TABLE statements that define a relational DB that is roughly equivalent to that defined by this XML DTD.

```
<!DOCTYPE Bookstore [
  <!ELEMENT Bookstore (Book | Magazine)*>
  <!ELEMENT Book (Title, Authors, Remark?)>
  <!ATTLIST Book ISBN CDATA #REQUIRED
          Price CDATA #REQUIRED
          Edition CDATA #IMPLIED>
  <!ELEMENT Magazine (Title)>
  <!ATTLIST Magazine Month CDATA #REQUIRED Year CDATA #REQUIRED>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Authors (Author+)>
  <!ELEMENT Remark (#PCDATA)>
  <!ELEMENT Author (First_Name, Last_Name)>
  <!ELEMENT First_Name (#PCDATA)>
  <!ELEMENT Last_Name (#PCDATA)>
]>
```



In this exercise, I didn't specify deterministic translation rules from XML to UML and SQL table definitions, but if we were to automate the translation process, developers would agree on deterministic translation rules. I don't show the table translations, but ask if the UML-to-tables translation isn't clear. This UML assumes

(a) that every non-leaf element translates to a UML class

(a) that every XML attribute or leaf element translates to a UML class attribute (and would translate to a table attribute too)

(b) #IMPLIED XML attribites translates to allowing NULL values for corresponding table attributes

(c) #REQUIRED XML attributes translate to NOT NULL declarations in tables

(d) '?' After an XML **leaf element** translates to allowing NULL for the corresponding table attribute; '?' after a non-leaf element translates to a 0..1 if an associated class for an associated class (there is no example of this in this diagram); if a leaf element is not qualified by a '?' then is it NOT NULL; if a non-leaf element is not qualified by a '?' then this translates to a 1..1 cardinality (see Book to Authors) and see the explanation of this construct to right.

(e) '*' translates to a cardinality of 0..* (see Bookstore to Item) and '+' tranlates to 1..* (see Authors to Author)

(f) The '|' (or) tranlates to superclass with complete coverage, and perhaps with a default of disjoint

(g) Since Title was a leaf element of both Book and Magazine, I put Title in the parent

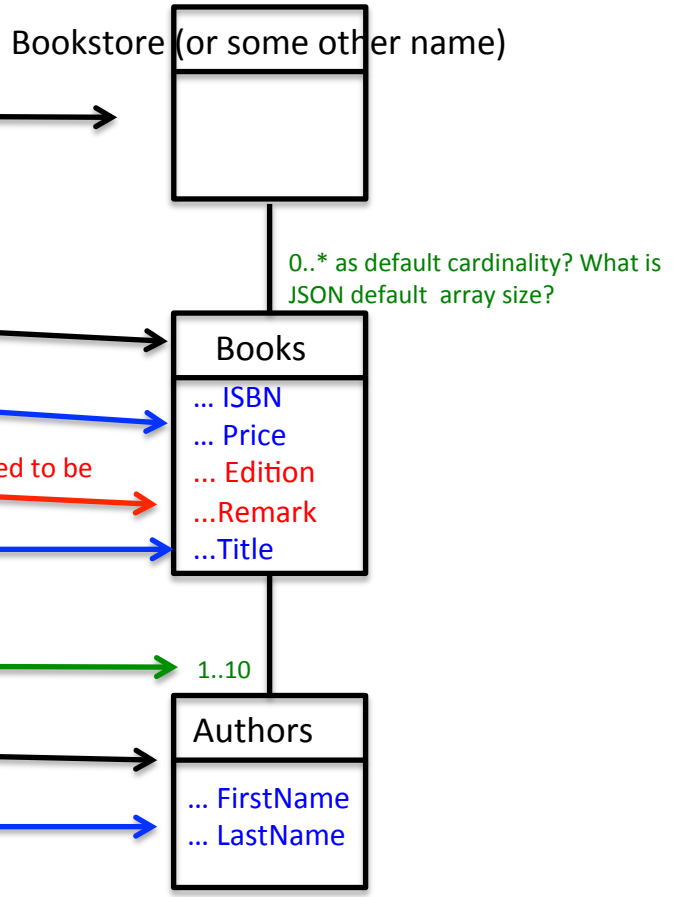
(h) I have not specified other cardinalities that are not implied by the XML to UML translation policy above, but presumably a good default would be 0..*

In this tranlation of the XML, Authors would represent the set of authors (perhaps a singleton set) of a Book (there Is exactly one unique set per Book). If manual revision of an automated translater were allowed, then we might we decide to delete the Authors class in the translation (unless there were attributes associated with the Authors element in the XML) , and have a direct Association from Book to 1..* Author

Give UML and SQL CREATE TABLE statements that define a relational DB that is roughly equivalent to that defined by this JSON Schema.

```
{ "type":"object",
  "properties": {
    "Books": {
      "type":"array",
      "items": {
        "type":"object",
        "properties": {
          "ISBN": { "type":"string", "pattern":"ISBN*" },
          "Price": { "type":"integer",
                "minimum":0, "maximum":200 },
          "Edition": { "type":"integer", "optional": true },
          "Remark": { "type":"string", "optional": true },
          "Title": { "type":"string" },
          "Authors": {
            "type":"array",
            "minItems":1,
            "maxItems":10,
            "items": {
              "type":"object",
              "properties": {
                "First_Name": { "type":"string" },
                "Last_Name": { "type":"string" }}}}}}}},
}}
```

These would be attribute constraints in SQL table definitions



Much the same as in XML case, but translate, but translate each JSON object to a class, and translate arrays into cardinality constraints