Missing Information

Database Systems Lecture 10
Natasha Alechina

In This Lecture

• Missing Information
  • NULLs and the relational model
  • OUTER JOINs
  • Default values
• For more information
  • Not really covered by Connolly and Begg
  • Some information in Chapter 3.3, 5, and 6
  • Ullman and Widom 6.1.5, 6.1.6, 6.3.8

Missing Information

• Sometimes we don’t know what value an entry in a relation should have
  • We know that there is a value, but don’t know what it is
  • There is no value at all that makes any sense

• Two main methods have been proposed to deal with this
  • NULLs can be used as markers to show that information is missing
  • A default value can be used to represent the missing value

NULLs

• NULL is a placeholder for missing or unknown value of an attribute. It is not itself a value.
• Codd proposed to distinguish two kinds of NULLs:
  • A-marks: data Applicable but not known (for example, someone’s age)
  • I-marks: data is Inapplicable (telephone number for someone who does not have a telephone, or spouse’s name for someone who is not married)

Problems with NULLs

• Problems with extending relational algebra operations to NULLs:
  • Defining selection operation: if we check tuples for some property like Mark > 40 and for some tuple Mark is NULL, do we include it?
  • Defining intersection or difference of two relations: are two tuples <John,NULL> and <John,NULL> the same or not?
  • Additional problems for SQL: do we treat NULLs as duplicates? Do we include them in count, sum, average and if yes, how? How do arithmetic operations behave when an argument is NULL?

Theoretical solutions 1

• Use three-valued logic instead of classical two-valued logic to evaluate conditions.
• When there are no NULLs around, conditions evaluate to true or false, but if a null is involved, a condition will evaluate to the third value (‘undefined’, or ‘unknown’).
• This is the idea behind testing conditions in WHERE clause of SQL SELECT: only tuples where the condition evaluates to true are returned.
3-valued logic

- If the condition involves a boolean combination, we evaluate it as follows:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x AND y</th>
<th>x OR y</th>
<th>NOT x</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>unknown</td>
<td>unknown</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>unknown</td>
<td>unknown</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

false=0, true=1, unknown=1/2, NOT(x)=1-x, AND(x,y) = min(x,y), OR(x,y) = max(x,y):

Theoretical solutions 2

- Use variables instead of NULLs to represent unknown values.
- Different unknown values correspond to different variables.
- When we apply operations such as selection to tables with variables, variables may acquire side conditions (constraints), for example x > 40 if x was unknown value of Mark and we include it in result of selection Mark > 40.
- This works out fine, but has high computational complexity and is not used in practice.

SQL solution: NULLs in conditions

- Salary > 15,000 evaluates to 'unknown' on the last tuple - not included

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>25,000</td>
</tr>
<tr>
<td>Mark</td>
<td>15,000</td>
</tr>
<tr>
<td>Anne</td>
<td>20,000</td>
</tr>
<tr>
<td>Chris</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SQL solution: arithmetic

- Arithmetic operations applied to NULLs result in NULLs

<table>
<thead>
<tr>
<th>Name</th>
<th>NewSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>27,500</td>
</tr>
<tr>
<td>Mark</td>
<td>16,500</td>
</tr>
<tr>
<td>Anne</td>
<td>22,000</td>
</tr>
<tr>
<td>Chris</td>
<td>NULL</td>
</tr>
</tbody>
</table>
SQL solution: aggregates

```sql
SELECT
    AVG(Salary) AS Avg,
    COUNT(Salary) AS Num,
    SUM(Salary) AS Sum
FROM Employee
```

- Avg = 20,000
- Num = 3
- Sum = 60,000

```sql
SELECT COUNT(*)...
gives a result of 4
```

Example: inner join

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>123 DBS</td>
<td>60</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
<td>124 PRG</td>
<td>70</td>
</tr>
<tr>
<td>125</td>
<td>Mark</td>
<td>125 DBS</td>
<td>50</td>
</tr>
<tr>
<td>126</td>
<td>Jane</td>
<td>126 DBS</td>
<td>80</td>
</tr>
</tbody>
</table>

Example: full outer join

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>123 DBS</td>
<td>60</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
<td>124 PRG</td>
<td>70</td>
</tr>
<tr>
<td>125</td>
<td>Mark</td>
<td>125 DBS</td>
<td>50</td>
</tr>
<tr>
<td>126</td>
<td>Jane</td>
<td>126 DBS</td>
<td>80</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Example: left outer join

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>123 DBS</td>
<td>60</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
<td>124 PRG</td>
<td>70</td>
</tr>
<tr>
<td>125</td>
<td>Mark</td>
<td>125 DBS</td>
<td>50</td>
</tr>
<tr>
<td>126</td>
<td>Jane</td>
<td>126 DBS</td>
<td>80</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Example: right outer join

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>123 DBS</td>
<td>60</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
<td>124 PRG</td>
<td>70</td>
</tr>
<tr>
<td>125</td>
<td>Mark</td>
<td>125 DBS</td>
<td>50</td>
</tr>
<tr>
<td>126</td>
<td>Jane</td>
<td>126 DBS</td>
<td>80</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Outer Joins

- When we take the join of two relations we match up tuples which share values
- Some tuples have no match, and are 'lost'
- These are called 'dangles'

- Outer joins include dangles in the result and use NULLs to fill in the blanks
  - Left outer join
  - Right outer join
  - Full outer join
Outer Join Syntax in Oracle

```sql
SELECT <cols>
FROM <table1> <type> OUTER JOIN <table2>
ON <condition>
```

Where `<type>` is one of LEFT, RIGHT, or FULL

Example:
```sql
SELECT * 
FROM Student FULL OUTER JOIN Enrolment 
ON Student.ID = Enrolment.ID
```

Default Values

- Default values are an alternative to the use of NULLs
  - If a value is not known a particular placeholder value - the default - is used
  - These are actual values, so don’t need 3VL etc.

Example:
```sql
UPDATE Parts 
SET Qty = Qty + 5
```

Problems With Default Values

- Since defaults are real values
  - They can be updated like any other value
  - You need to use a value that won’t appear in any other circumstances
  - They might not be interpreted properly

Also, within SQL defaults must be of the same type as the column
- You can’t have a string such as ‘unknown’ in a column of integers

Splitting Tables

- NULLs and defaults both try to fill entries with missing data
  - NULLs mark the data as missing
  - Defaults give some indication as to what sort of missing information we are dealing with

- Often you can remove entries that have missing data
  - You can split the table up so that columns which might have NULLs are in separate tables
  - Entries that would be NULL are not present in these tables

Example:
```sql
UPDATE Parts 
SET Qty = Qty + 5
```
Problems with Splitting Tables

- Splitting tables has its own problems
  - We might introduce many extra tables
  - Information gets spread out over the database
  - Queries become more complex and require many joins
- We can recover the original table, but
  - We need to do an outer join to do so
  - This introduces NULLs, which brings in all the associated problems again

SQL Support

- SQL allows both NULLs and defaults:
  - A table to hold data on employees
  - All employees have a name
  - All employees have a salary (default 10000)
  - Some employees have phone numbers, if not we use NULLs

```
CREATE TABLE Employee
(
  Name CHAR(50) NOT NULL,
  Salary INT DEFAULT 10000,
  Phone CHAR(15) NULL
)
```

- SQL allows you to insert NULLs
  - INSERT INTO Employee
    VALUES ('John', 12000, NULL)
  - UPDATE Employee
    SET Phone = NULL
    WHERE Name = 'Mark'

- You can also check for NULLs
  - SELECT Name FROM Employee
    WHERE Phone IS NULL
  - SELECT Name FROM Employee
    WHERE Phone IS NOT NULL

SQL Support

- Which Method to Use?
  - Often a matter of personal choice, but
  - Default values should not be used when they might be confused with 'real' values
  - Splitting tables shouldn't be used too much or you'll have lots of tables
  - NULLs can (and often are) used where the other approaches seem inappropriate
  - You don't have to always use the same method - you can mix and match as needed

Example

- For an online store we have a variety of products - books, CDs, and DVDs
  - All items have a title, price, and id (their catalogue number)
  - Any item might have an extra shipping cost, but some don't
- There is also some data specific to each type
  - Books must have an author and might have a publisher
  - CDs must have an artist
  - DVDs might have a producer or director

Example

- We could put all the data in one table
  - There will be many entries with missing information
  - Every row will have missing information
  - We are storing three types of thing in one table
Example

- It is probably best to split the three types into separate tables
- We’ll have a main Items table
- Also have Books, CDs, and DVDs tables with FKs to the Items table

<table>
<thead>
<tr>
<th>Item</th>
<th>ID</th>
<th>Title</th>
<th>Price</th>
<th>Shipping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books</td>
<td></td>
<td>Author</td>
<td>Publisher</td>
<td></td>
</tr>
<tr>
<td>CDs</td>
<td></td>
<td>Artist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVDs</td>
<td></td>
<td>Producer</td>
<td>Director</td>
<td></td>
</tr>
</tbody>
</table>

Example

- Each of these tables might still have some missing information
- Shipping cost in items could have a default value of 0
- This should not disrupt computations
- If no value is given, shipping is free

- Other columns could allow NULLs
  - Publisher, director, and producer are all optional
  - It is unlikely we’ll ever use them in computation

Next Lecture

- Normalisation to 3NF
  - Data redundancy
  - Functional dependencies
  - Normal forms
  - First, Second and Third Normal Forms

- For more information
  - Connolly and Begg chapter 13
  - Ullman and Widom 1.1.4 (2nd edition), more in 3rd edition (3.5).