Missing Information

Database Systems

Michael Pound

This Lecture

- Missing Information
  - Nulls and the Relational Model
  - Outer Joins
  - Default Values
- Further reading
  - The Manga Guide to Databases, Chapter 2
  - Database Systems, Chapter 4

Coursework

- The coursework will be released at 12pm on Friday
- The coursework will involve designing, creating and using a database
- The coursework is worth 25% of this module
- The deadline is Midnight on Friday 25th March
- Labs on 11th, 18th and 25th will not have additional exercises, so there will be time for coursework
- The late penalty is 5% per working day. As always, don't plagiarise (working together counts)

Coursework

- There are three parts to the coursework, and three files to submit:
  - Designing the database: Draw E/R diagrams based on a problem specification
    - Submission: **cwpart1.doc**
  - Creating the database: Create and populate tables based on the E/R diagrams you've designed
    - Submission: **cwpart2.sql**
  - Using the database: Create a webpage based information in the database from part 2
    - Submission: **index.php**

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?
  - Comparing tuples in 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

- 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 might 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:

a	b	a OR b	a AND b	a == b
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

SQL NULLs in Conditions

```
SELECT *
 FROM Employee
 Where Salary > 15,000;
```

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

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

Name	Salary
John	25,000
Anne	20,000

SQL NULLs in Conditions

```
SELECT *
 FROM Employee
 Where Salary > 15,000
 OR Name = 'Chris';
```

- **`Salary > 15,000 OR Name = 'Chris'`** is essentially **`Unknown OR TRUE`** on the last tuple

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

Name	Salary
John	25,000
Anne	20,000
Chris	NULL

SQL NULLs in Arithmetic

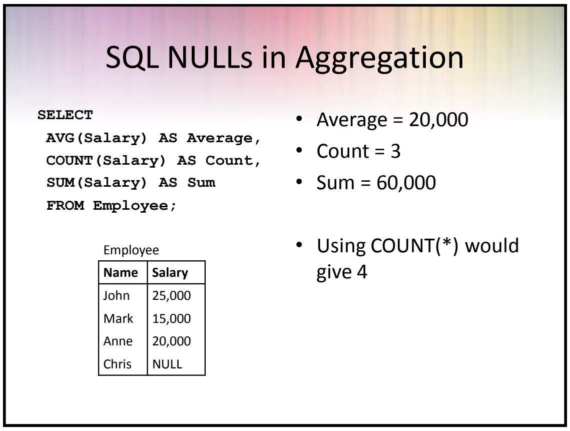
```
SELECT
 Name,
 Salary * 0.05 AS Bonus
 FROM Employee;
```

- Arithmetic operations applied to NULLs result in NULLS

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

Name	Bonus
John	1,250
Mark	750
Anne	1,000
Chris	NULL



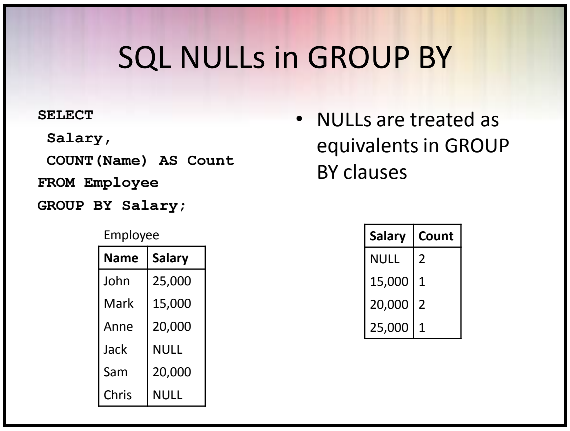
SQL NULLs in Aggregation

```
SELECT
 AVG(Salary) AS Average,
 COUNT(Salary) AS Count,
 SUM(Salary) AS Sum
 FROM Employee;
```

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

- Average = 20,000
- Count = 3
- Sum = 60,000
- Using COUNT(*) would give 4



SQL NULLs in GROUP BY

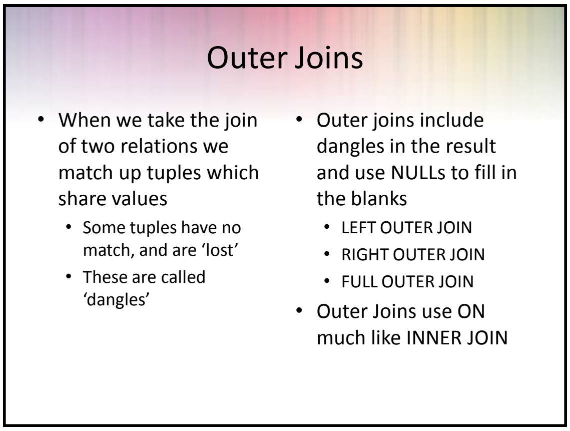
```
SELECT
 Salary,
 COUNT(Name) AS Count
FROM Employee
GROUP BY Salary;
```

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Jack	NULL
Sam	20,000
Chris	NULL

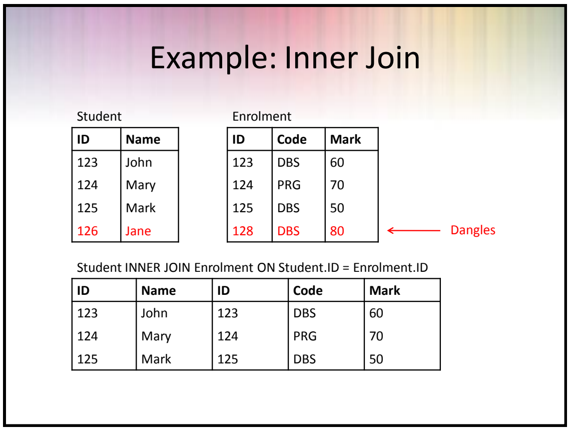
- NULLs are treated as equivalents in GROUP BY clauses

Salary	Count
NULL	2
15,000	1
20,000	2
25,000	1



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 Joins use ON much like INNER JOIN



Example: Inner Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

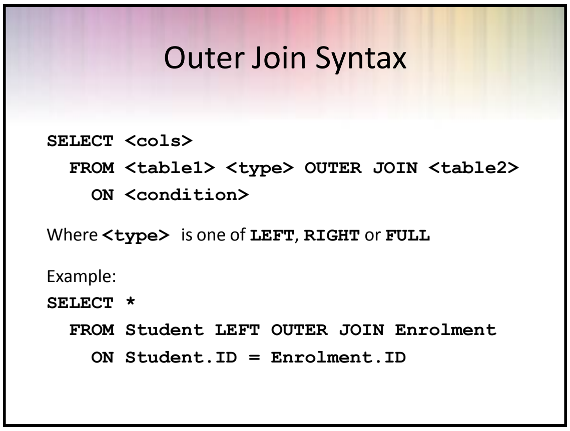
Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

Dangles

Student INNER JOIN Enrolment ON Student.ID = Enrolment.ID

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50



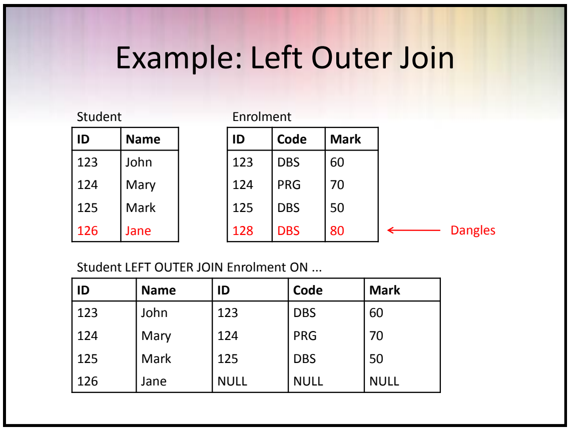
Outer Join Syntax

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

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

Example:

```
SELECT *
  FROM Student LEFT OUTER JOIN Enrolment
    ON Student.ID = Enrolment.ID
```



Example: Left Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

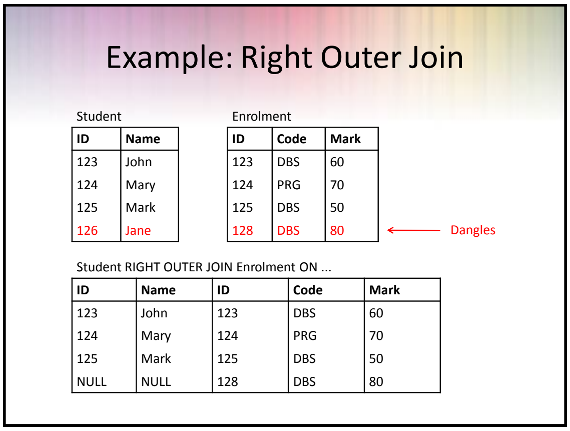
Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

Dangles

Student LEFT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
126	Jane	NULL	NULL	NULL



Example: Right Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

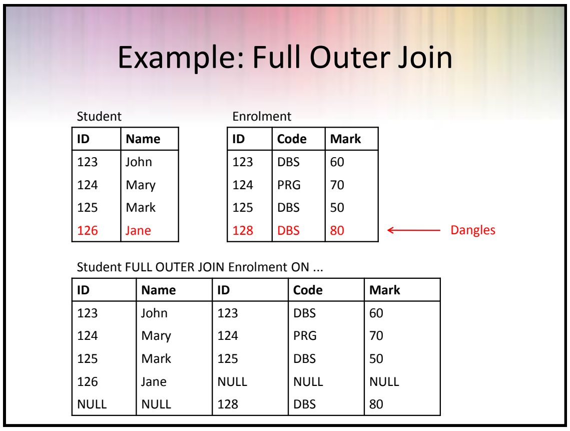
Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

Dangles

Student RIGHT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
NULL	NULL	128	DBS	80



Example: Full Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

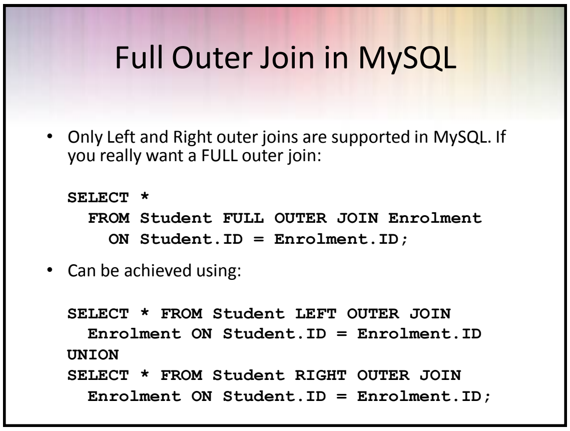
Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

Dangles

Student FULL OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
126	Jane	NULL	NULL	NULL
NULL	NULL	128	DBS	80



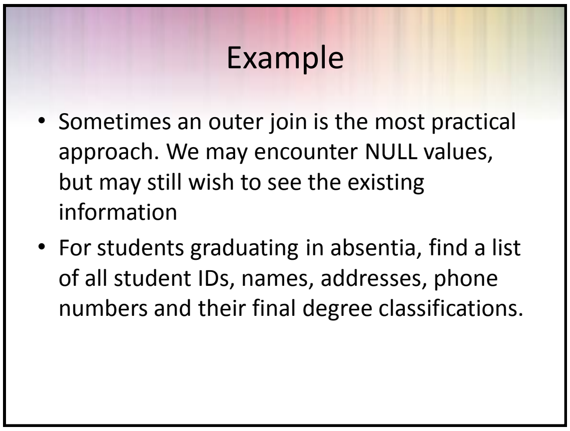
Full Outer Join in MySQL

- Only Left and Right outer joins are supported in MySQL. If you really want a FULL outer join:

```
SELECT *
  FROM Student FULL OUTER JOIN Enrolment
    ON Student.ID = Enrolment.ID;
```

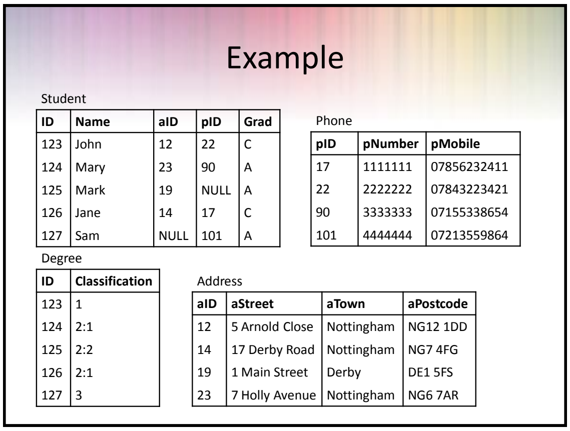
- Can be achieved using:

```
SELECT * FROM Student LEFT OUTER JOIN
  Enrolment ON Student.ID = Enrolment.ID
UNION
SELECT * FROM Student RIGHT OUTER JOIN
  Enrolment ON Student.ID = Enrolment.ID;
```



Example

- Sometimes an outer join is the most practical approach. We may encounter NULL values, but may still wish to see the existing information
- For students graduating in absentia, find a list of all student IDs, names, addresses, phone numbers and their final degree classifications.



Example

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Phone

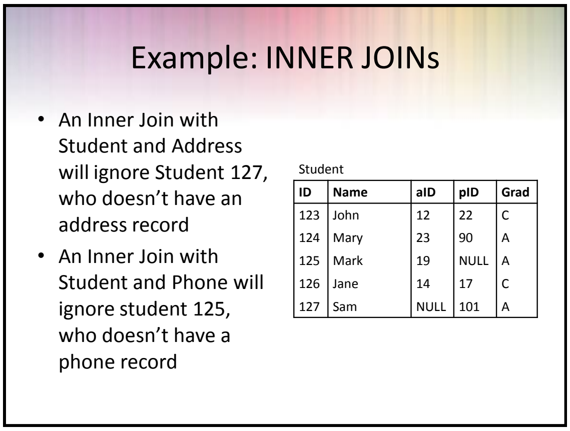
pID	pNumber	pMobile
17	1111111	07856232411
22	2222222	07843223421
90	3333333	07155338654
101	4444444	07213559864

Degree

ID	Classification
123	1
124	2:1
125	2:2
126	2:1
127	3

Address

aID	aStreet	aTown	aPostcode
12	5 Arnold Close	Nottingham	NG12 1DD
14	17 Derby Road	Nottingham	NG7 4FG
19	1 Main Street	Derby	DE1 5FS
23	7 Holly Avenue	Nottingham	NG6 7AR

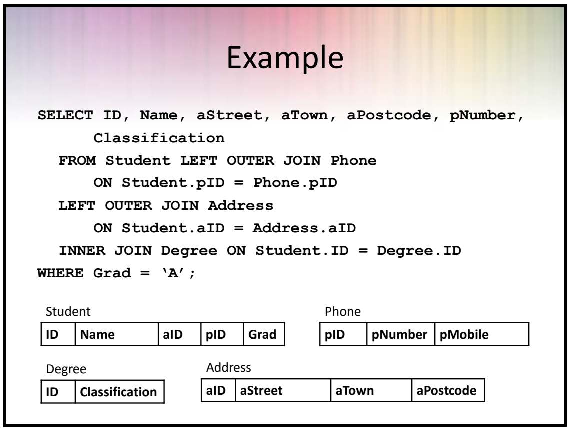


Example: INNER JOINs

- An Inner Join with Student and Address will ignore Student 127, who doesn't have an address record
- An Inner Join with Student and Phone will ignore student 125, who doesn't have a phone record

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A



Example

```
SELECT ID, Name, aStreet, aTown, aPostcode, pNumber,
      Classification
  FROM Student LEFT OUTER JOIN Phone
      ON Student.pID = Phone.pID
  LEFT OUTER JOIN Address
      ON Student.aID = Address.aID
  INNER JOIN Degree ON Student.ID = Degree.ID
WHERE Grad = 'A';
```

Student

ID	Name	aID	pID	Grad

Phone

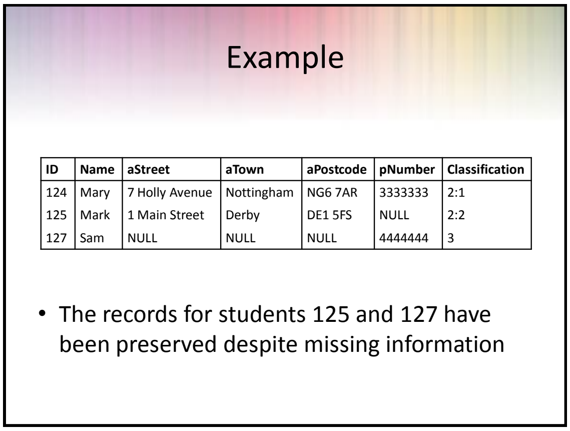
pID	pNumber	pMobile

Degree

ID	Classification

Address

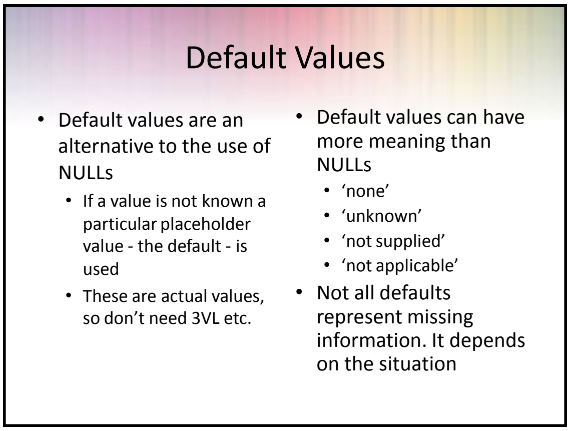
aID	aStreet	aTown	aPostcode



Example

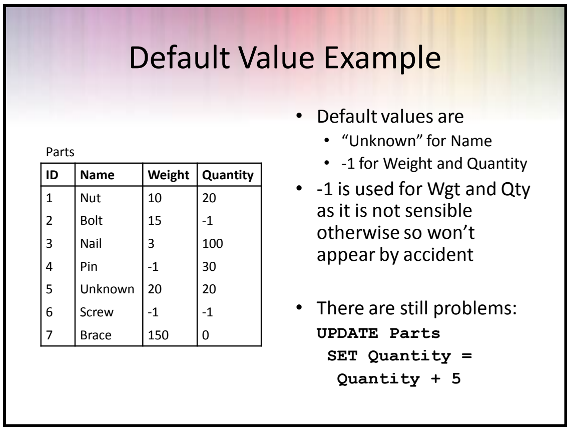
ID	Name	aStreet	aTown	aPostcode	pNumber	Classification
124	Mary	7 Holly Avenue	Nottingham	NG6 7AR	3333333	2:1
125	Mark	1 Main Street	Derby	DE1 5FS	NULL	2:2
127	Sam	NULL	NULL	NULL	4444444	3

- The records for students 125 and 127 have been preserved despite missing information



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.
- Default values can have more meaning than NULLs
  - 'none'
  - 'unknown'
  - 'not supplied'
  - 'not applicable'
- Not all defaults represent missing information. It depends on the situation



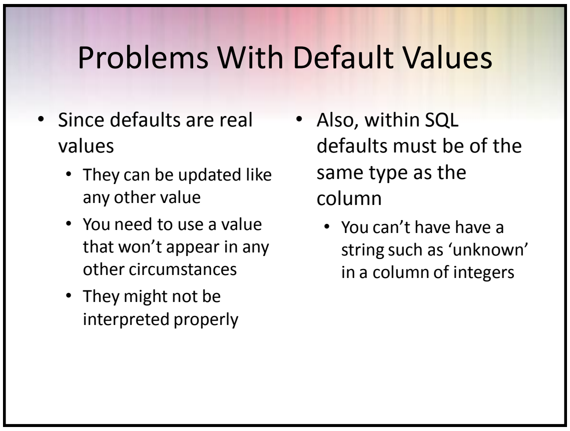
Default Value Example

Parts

ID	Name	Weight	Quantity
1	Nut	10	20
2	Bolt	15	-1
3	Nail	3	100
4	Pin	-1	30
5	Unknown	20	20
6	Screw	-1	-1
7	Brace	150	0

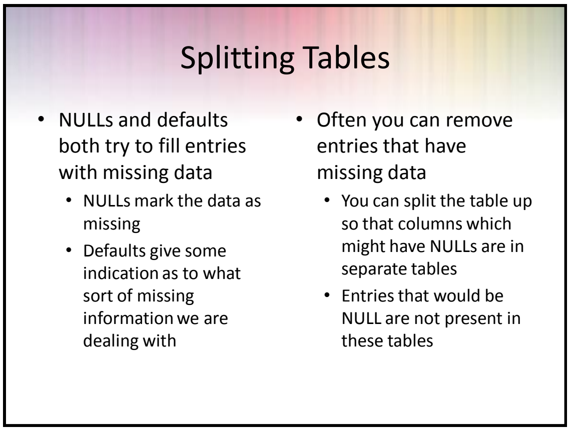
- Default values are
  - "Unknown" for Name
  - -1 for Weight and Quantity
- -1 is used for Wgt and Qty as it is not sensible otherwise so won't appear by accident
- There are still problems:

```
UPDATE Parts
  SET Quantity =
    Quantity + 5
```



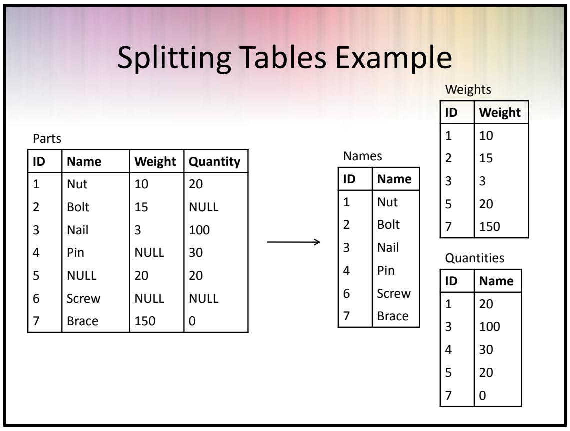
Problems With Default Values

- 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 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



Splitting Tables Example

Parts

ID	Name	Weight	Quantity
1	Nut	10	20
2	Bolt	15	NULL
3	Nail	3	100
4	Pin	NULL	30
5	NULL	20	20
6	Screw	NULL	NULL
7	Brace	150	0

Names

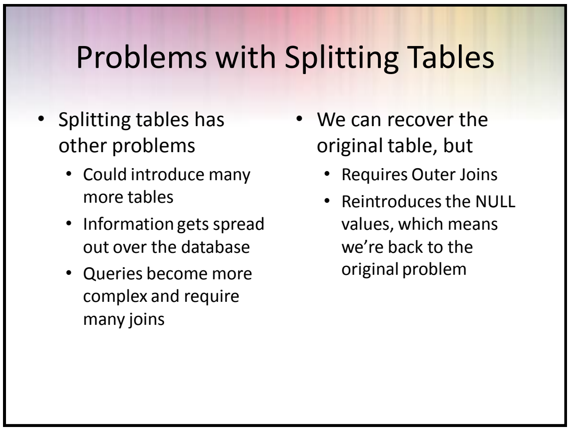
ID	Name
1	Nut
2	Bolt
3	Nail
4	Pin
6	Screw
7	Brace

Weights

ID	Weight
1	10
2	15
3	3
5	20
7	150

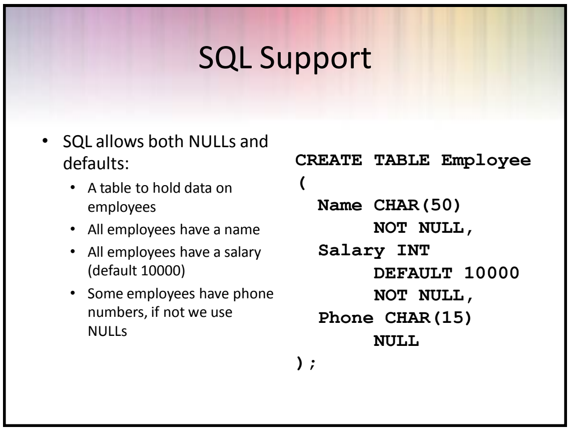
Quantities

ID	Name
1	20
3	100
4	30
5	20
7	0



Problems with Splitting Tables

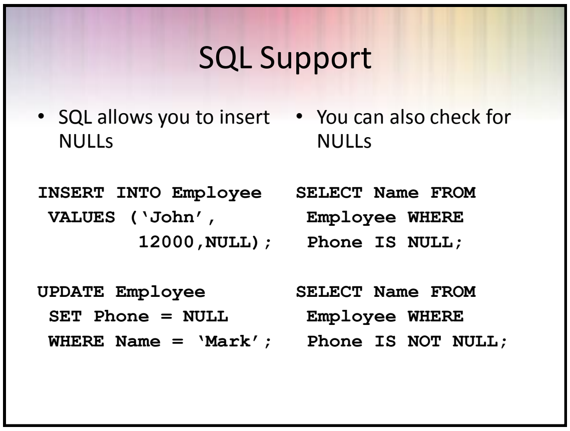
- Splitting tables has other problems
  - Could introduce many more tables
  - Information gets spread out over the database
  - Queries become more complex and require many joins
- We can recover the original table, but
  - Requires Outer Joins
  - Reintroduces the NULL values, which means we're back to the original problem



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
       NOT NULL,
  Phone CHAR(15)
       NULL
);
```



SQL Support

- SQL allows you to insert NULLs

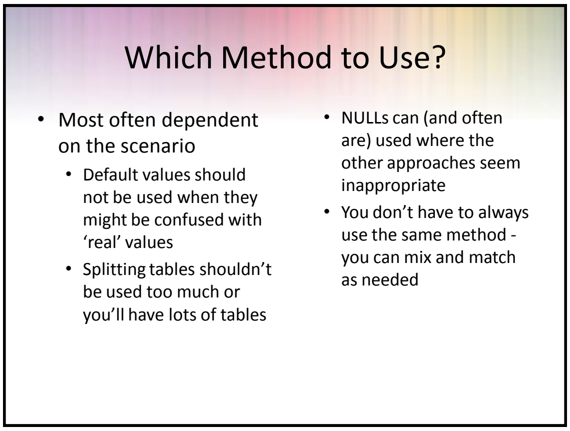
```
INSERT INTO Employee
 VALUES (‘John’,
         12000,NULL);

UPDATE Employee
 SET Phone = NULL
 WHERE Name = ‘Mark’;
```

- You can also check for NULLs

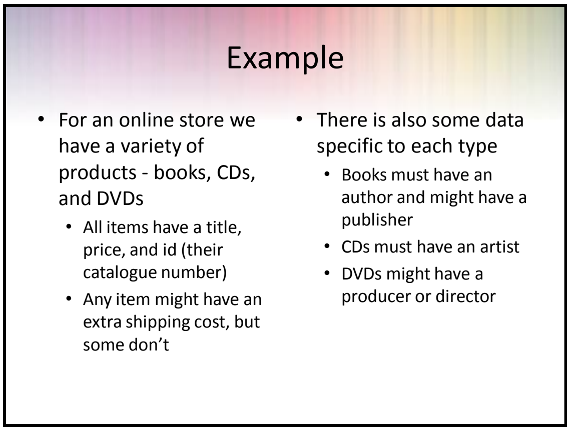
```
SELECT Name FROM
 Employee WHERE
 Phone IS NULL;

SELECT Name FROM
 Employee WHERE
 Phone IS NOT NULL;
```



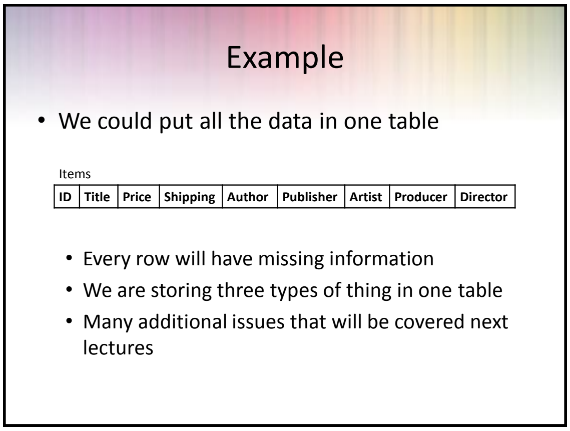
Which Method to Use?

- Most often dependent on the scenario
  - 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



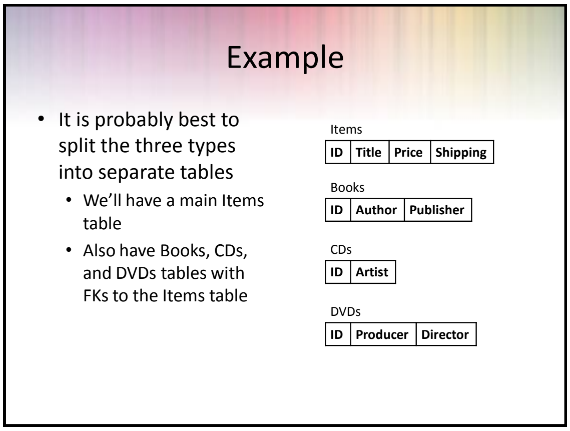
Example

- We could put all the data in one table

Items

ID	Title	Price	Shipping	Author	Publisher	Artist	Producer	Director

  - Every row will have missing information
  - We are storing three types of thing in one table
  - Many additional issues that will be covered next lectures



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

Items

ID	Title	Price	Shipping

Books

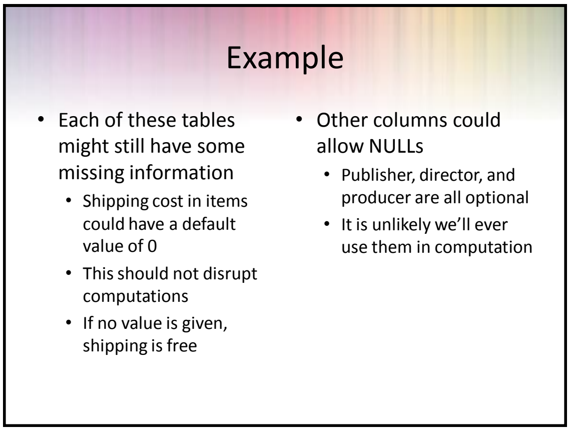
ID	Author	Publisher

CDs

ID	Artist

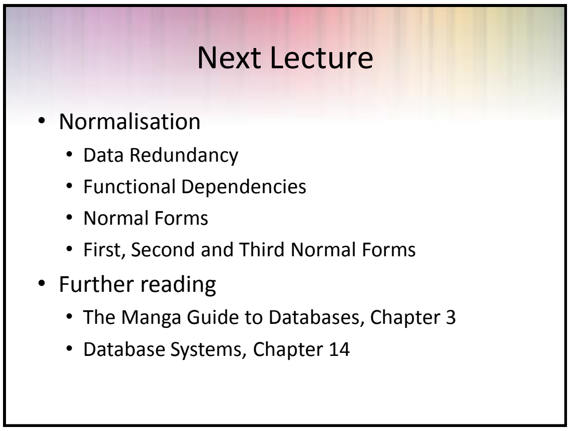
DVDs

ID	Producer	Director



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
  - Data Redundancy
  - Functional Dependencies
  - Normal Forms
  - First, Second and Third Normal Forms
- Further reading
  - The Manga Guide to Databases, Chapter 3
  - Database Systems, Chapter 14