Databases and Info Systems

Normalisation Example

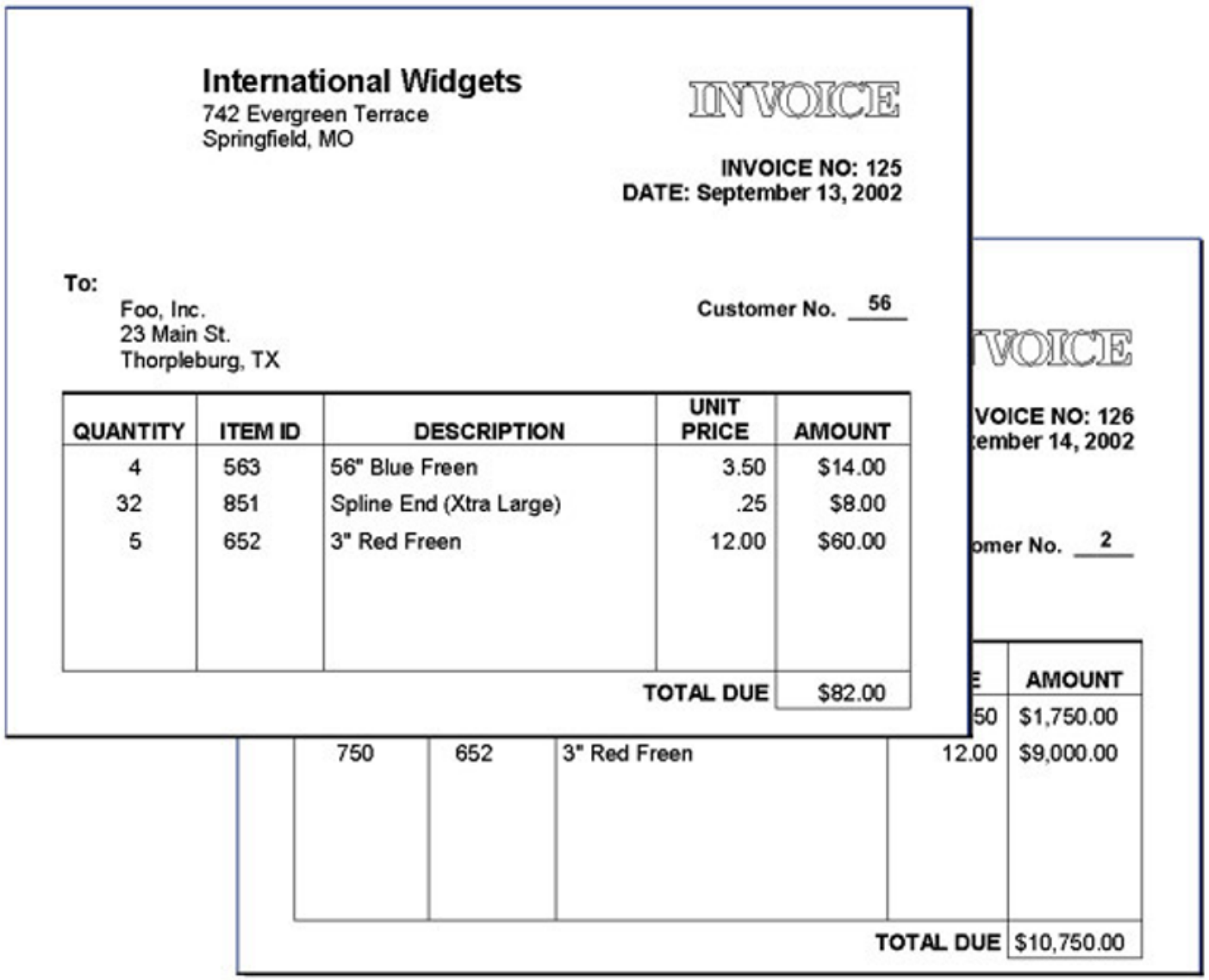
Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

March 30, 2020

Table of Contents

- A company wishes to keep track of sales invoices.
- Each invoice contains data about the sale of a number of items to a particular customer
- Based on the tutorial at: `http://www.phlonx.com/resources/nf3`



International Widgets
742 Evergreen Terrace
Springfield, MO

INVOICE

INVOICE NO: 125
DATE: September 13, 2002

To:
Foo, Inc.
23 Main St.
Thorpleburg, TX

Customer No. 56

QUANTITY	ITEM ID	DESCRIPTION	UNIT PRICE	AMOUNT
4	563	56" Blue Freen	3.50	$14.00
32	851	Spline End (Xtra Large)	.25	$8.00
5	652	3" Red Freen	12.00	$60.00
			TOTAL DUE	$82.00

VOICE

VOICE NO: 126
ember 14, 2002

omer No. 2

			E	AMOUNT
			50	$1,750.00
750	652	3" Red Freen	12.00	$9,000.00
			TOTAL DUE	$10,750.00

- Previously, the company kept track of this data using a spreadsheet (below).
- Because the amount of data is getting large, they wish to record it in a database.
- This will allow them to do more complex calculations easily, using SQL.

orders.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Invoice No.**	**Date**	**Cust. No.**	**Cust. Name**	**Cust. Address**	**Cust. City**	**Cust. State**	**Item ID**	**Item Descri**	**Item Qty.**	**Item Price**	**Item Total**	**Order Total Price**
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563	56" Blue Fre	4	$ 3.50	$ 14.00	$ 82.00
3								851	Spline End (	32	$ 0.25	$ 8.00	$ 82.00
4								652	3" Red Free	5	$ 12.00	$ 60.00	$ 82.00
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenu	Washington	DC	563	56" Blue Fre	500	$ 3.50	$1,750.00	$ 10,750.00
6								652	3" Red Free	750	$ 12.00	$9,000.00	$ 10,750.00

- First Normal Form (1NF) demands that there are no repeating groups.
- Here, there is data relating to items repeating for each invoice, so it's not in 1NF.
- To avoid this, one option is to "flatten" the table.

orders.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Invoice No.**	**Date**	**Cust. No.**	**Cust. Name**	**Cust. Address**	**Cust. City**	**Cust. State**	**Item ID**	**Item Descri**	**Item Qty.**	**Item Price**	**Item Total**	**Order Total Price**
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563	56" Blue Fre	4	$ 3.50	$ 14.00	$ 82.00
3								851	Spline End (	32	$ 0.25	$ 8.00	$ 82.00
4								652	3" Red Free	5	$ 12.00	$ 60.00	$ 82.00
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenu	Washington	DC	563	56" Blue Fre	500	$ 3.50	$1,750.00	$ 10,750.00
6								652	3" Red Free	750	$ 12.00	$9,000.00	$ 10,750.00

- We flatten the table by filling in all the fields in each row.
- There is now quite a bit of duplicated data (i.e. **redundancy**) in our table, but we can deal with that later.
- Since this is to go into a database, we need to identify a **primary key** for this table.

orders.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Invoice No.	Date	Cust. No.	Cust. Name	Cust. Address	Cust. City	Cust. State	Item ID	Item Descri	Item Qty.	Item Price	Item Total	Order Total Price
2	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	563	56" Blue Fre	4	$ 3.50	$ 14.00	$ 82.00
3	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	851	Spline End (	32	$ 0.25	$ 8.00	$ 82.00
4	125	9/13/2002	56	Foo, Inc.	23 Main St., Thorpleburg	Thorpleburg	TX	652	3" Red Free	5	$ 12.00	$ 60.00	$ 82.00
5	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenu	Washington	DC	563	56" Blue Fre	500	$ 3.50	$1,750.00	$ 10,750.00
6	126	9/14/2002	2	Freens R Us	1600 Pennsylvania Avenu	Washington	DC	652	3" Red Free	750	$ 12.00	$9,000.00	$ 10,750.00

- The order_id cannot be used by itself as a primary key, since each order is represented by several rows.

- Similarly, the item_id cannot be used by itself either, since each item occurs several times (for different orders).

orders : Table

order_id	order_date	customer_id	customer_name	customer_addre	customer_city	customer_	item_id	item_description	item_qty	item_price	item_total_price	order_total_price
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	563	56" Blue Freen	4	$3.50	$14.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	851	Soline End (Xtra	32	$0.25	$8.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	652	3' Red Freen	5	$12.00	$60.00	$82.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	563	56" Blue Freen	500	$3.50	$1,750.00	$10,750.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	652	3' Red Freen	750	$12.00	$9,000.00	$10,750.00

Record: 6 of 6

- Instead, we can use both the order_id and item_id together as a compound (or composite) primary key.

- The combination of these two attributes uniquely identifies each row.

orders : Table

order_id	order_date	customer_id	customer_name	customer_addre	customer_city	customer_	item_id	item_description	item_qty	item_price	item_total_price	order_total_price
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	563	56" Blue Freen	4	$3.50	$14.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	851	Spline End (Xtra	32	$0.25	$8.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	652	3" Red Freen	5	$12.00	$60.00	$82.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	563	56" Blue Freen	500	$3.50	$1,750.00	$10,750.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	652	3" Red Freen	750	$12.00	$9,000.00	$10,750.00

Record: 6 of 6

- orders(order_id, order_date, customer_id, customer_name, customer_address, customer_city, customer_state, item_id, item_description, item_qty, item_price, item_total_price, order_total_price)

- We have now succeeded in converting our data into 1NF.

- This was done by:
  1. Flattening the table.
  2. Extending the primary key

- What about 2NF?
- 2NF demands that there are **no partial key functional dependencies**.
  - This is only an issue where we have compound primary keys.
  - The use of order_id and item_id as a compound primary key in our example means that this is something we need to examine

- Key fields:

order_id Identifies the invoice

item_id Identifies the items that have been sold.

- We need to examine all the other attributes in the table to see if they have a functional dependency on one or both of these.

- order_date
- customer_id
- customer_name
- customer_address
- customer_city
- customer_state

- item_description
- item_qty
- item_price
- item_total_price
- order_total_price

- Dependent only on order_id:
  - order_date
  - customer_id
  - customer_name
  - customer_address
  - customer_city
  - customer_state
- Dependent only on item_id:
  - item_description
  - item_price
- Dependent on both:
  - item_qty
  - item_total_price
  - order_total_price

- What about item_total_price and order_total_price?
- On first glance, it appears that these are both dependent on both order_id and item_id.
- However, we can see that these are derived values, which can be calculated using data that we are already storing (i.e. the price of items, and their quantity).
- Derived values in a database add redundancy!
- For this reason, they should not be included

- We now need to perform decomposition, to split this into multiple tables.
- We need to take every part of the compound key in turn, and split it (and any attributes that solely depend on it) into a different table.
- The key itself is left behind in the original table to act as a foreign key (and that is still part of the original table's primary key).
- This table has two parts to the compound key: order_id and item_id.
- We need to deal with each of these

- We begin with order_id and get tables that look like this:
  - `orders(order_id, order_date, customer_id, customer_name, customer_address, customer_city, customer_state)`
  - `orders_items(order_id, item_id, item_description, item_qty, item_price)`
- Everything that depends only on order_id has been moved to the orders table.
- Everything else is left behind in the original table (order_items).

- Now we do the same with item_id:
  - orders(<u>order_id</u>, order_date, customer_id, customer_name, customer_address, customer_city, customer_state)
  - orders_items(<u>order_id</u>, <u>item_id</u>, item_qty)
  - items(<u>item_id</u>, item_description, item_price)

- Are we now in 2NF?
  - `orders(order_id, order_date, customer_id, customer_name, customer_address, customer_city, customer_state)`
  - `items(item_id, item_description, item_price)`
- The orders and items tables only have 1 attribute as their primary key.
- They must be in 2NF, because it is impossible to have a partial key dependency with a simple primary key.

- Are we now in 2NF?
  - `orders_items(order_id, item_id, item_qty)`
- The order_items table has a compound key.
- The only non-key attribute is item_qty, which is functionally dependent on both parts of the primary key.
- We are now in 2NF

- What about 3NF?
- 3NF insists that we have no transitive functional dependencies.
- This is when one non-key attribute has a functional dependency on another non-key attribute.
- This is not possible if there is only one non-key attribute.
- We must examine all three tables to see if they are in 3NF.

- `orders_items(order_id, item_id, item_qty)`
- We begin with order_items.
- This has only one non-key attribute (item_qty) so it is in 3NF.

- `items(item_id, item_description, item_price)`
- The items table is next.
- This has two non-key attributes, so we need to be careful.
- Does item_price depend on item_description (or vice-versa)?
- No! They both depend only on item_id (which is the primary key) so this is also in 3NF.

- `orders(order_id, order_date, customer_id, customer_name, customer_address, customer_city, customer_state)`
- Finally, we examine the orders table.
- Again, there are multiple non-key attributes, so we need to check for 3NF.
- order_date depends on order_id so that's OK.
- BUT, all the data about customers (name, address, city, state) depends on the customer_id rather than the order_id.

- We need to split these attributes into another table (leaving the attribute they depend on behind as a foreign key).
  - `customers(`<u>`customer_id`</u>`, customer_name, customer_address, customer_city, customer_state)`
  - `orders(`<u>`order_id`</u>`, order_date, `*`customer_id`*`)`

Final Database Design

- customers(customer_id, customer_name, customer_address, customer_city, customer_state)
- orders(order_id, order_date, *customer_id*)
- items(item_id, item_description, item_price)
- orders_items(order_id, item_id, item_qty)