Normalization

Well normalized data makes programming (relatively) easy, and works very well in multi-platform, enterprise wide environments. Non-normalized data leads to heartbreak.

Normalization: The first three forms

Invoice Table Violate's Normalization Form 1

	Customer Information											
Invoice#	Cust#	Name	Addr	Quant1	Part1	Amt1	Quant2	Part2	Amt2	Quant3	Part3	Amt3
1001	43	Jones	121 1st	200	Screw	2.00	300	Nut	2.25	100	Washr	0.75
1002	55	Smith	222 2nd	1	Motor	52.00	5	Brace	44.44			
1003	43	Jones	121 1st	10	Saw	121.00						

First Normal Form:

No repeating groups. As an example, it might be tempting to make an invoice table with columns for the first, second, and third line item (see above). This violates the first normal form, and would result in large rows, wasted space (where an invoice had less than the maximum number of line items), and *horrible* SQL statements with a separate join for each repetition of the column. First form normalization requires you make a separate line item table, with it's own key (in this case the combination of invoice number and line number) (See below).

Complies with Normalization Form 1, Violate's Normalization Form 2

Invoice table

Invoice#
1001
1002
1003

Line item table

		Customer Information					
Invoice#	LIne#	Cust#	Name	Address	Quant1	Part1	Amt1
1001	1	43	Jones	121 1st	200	Screw	2.00
1001	2	43	Jones	121 1st	300	Nut	2.25
1001	3	43	Jones	121 1st	100	Washr	0.75
1002	1	55	Smith	222 2nd	1	Motor	52.00
1002	2	55	Smith	222 2nd	10	Saw	121.00
1003	1	43	Jones	121 1st	5	Brace	44.44

Second Normal Form:
Each column must depend on the *entire* primary key. As an example, the customer information could be put in the line item table (see above). The trouble with that is that the customer goes with the invoice, not with each line on the invoice. Putting customer information in the line item table will cause redundant data, with it's inherant overhead and difficult modifications. Second form normalization requires you place the customer information in the invoice table (see below).

Complies with Normalization Form 2, Violate's Normalization Form 3

Invoice table

	Customer Information		
Invoice#	Cust#	Name	Address
1001	43	Jones	121 1st
1002	55	Smith	222 2nd
1003	43	Jones	121 1st

Line item table

Invoice#	LIne#	Quant1	Part1	Amt1
1001	1	200	Screw	2.00
1001	2	300	Nut	2.25
1001	3	100	Washr	0.75
1002	1	1	Motor	52.00
1002	2	10	Saw	121.00
1003	1	5	Brace	44.44

Third Normal Form:
Each column must depend on *directly* on the primary key. As an example, the customer address could go in the invoice table (see above), but this would cause data redundancy if several invoices were for the same customer. It would also cause an update nightmare when the customer changes his address, and would require extensive programming to insert the address every time an existing customer gets a new invoice. Third form normalization requires the customer address go in a separate customer table with its own key (customer), with only the customer identifier in the invoice table (see below).

Complies with Normalization Form 3

Invoice table

Invoice#	Cust#
1001	43
1002	55
1003	43

Customer table

Cust#	Name	Address
43	Jones	121 1st
55	Smith	222 2nd

Line item table

Invoice#	LIne#	Quant1	Part1	Amt1
1001	1	200	Screw	2.00
1001	2	300	Nut	2.25
1001	3	100	Washr	0.75
1002	1	1	Motor	52.00
1002	2	10	Saw	121.00
1003	1	5	Brace	44.44

Additional normalization tips

Make a table for each list
Do this right away. It will save a fortune in time. Go through the department or enterprise, ferreting out all lists. Document them. Each should be a table if their information is needed, and if practical.

Use non-meaningful primary keys
If employee numbers starting with C mean the person's stationed in Chicago, and the person moves to Los Angeles, what do you do with his employee number. Making primary keys non-meaningful means changes in environment or business rules can't render them ineffective.