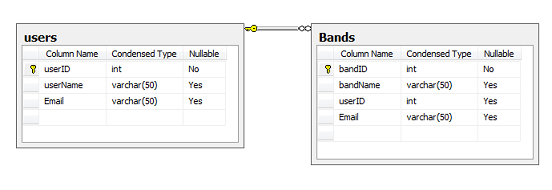
<http://dotnetanalysis.blogspot.com/2012/01/database-normalization-sql-server.html>

**Database Normalization Tutorial with example**

***(If you open this page in google chrome, there is a translation tool that shows up on the left above the article. Please use that to translate this to any language in the world. )***  
 **Normalizing a database means laying out the tables in an efficient way with the main focus being on preserving the integrity of data.**If the database is normalized, there is no repetition of data.  
  
**The need for Normalization**  
  
[](http://3.bp.blogspot.com/-0L-Hdf73sRY/TyYS-9J_WcI/AAAAAAAAANE/tRyImg-jet0/s1600/deNormalized.PNG)  
  
  
  
Consider the image above. If you notice, you could have a different Email associated with the same userID in both the tables. There is no restriction in the database to prevent this inconsistency from happening. The database totally relies on the UI application (that enters data into the database) to prevent this from happening. If the database was normalized it would never accept inconsistent data.  
  
  
**Forms of Normalization**  
  
There are 7 forms of Normalization. The seven forms of Normalization are

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. Boyce-Codd Normal Form
5. Fourth Normal Form
6. Fifth Normal Form
7. Domain/Key Normal Form

Many database designers normalize OLTP databases only up to the 3rd Normal form (and reporting OLAP databases up to 2nd Normal Form). The 3rd Normal form reduces redundancy to a reasonable level without complicating the design too much. In this article we will be proceeding only up to the third normal form.  
  
  
  
**Example of Database Normalization**  
  
  
**The Denormalized table**  
Consider the table below. This database stores data for products purchased by people online. This database  also stores their employer information. **In this example we will assume, that a person can only have one employer**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SSN** | **UserName** | **Product1** | **Product2** | **MoreProducts** | **EmployerName** | **EmployerAddress** |
| 332345432 | Amy | M |  |  | Google | 1 California drive |
| 666666666 | Kevin | A | B | C,D | Facebook | 22nd Street Sanfrancisco |
| 919919919 | Raj | D |  |  | Google | 1 California drive |

In the tables below the columns that make up the primary key are denoted in yellow headings. If you are unfamiliar with SQL keys, please check out this article  
 <http://dotnetanalysis.blogspot.com/2012/01/sql-key-super-key-candidate-key-primary.html>

**First Normal Form:**

1. You can only have one value in a column
2. You should not create multiple columns for a one to many relationship.

(There are more rules to the first normal form, such as two rows cannot be identical etc, which are automatically enforced by the SQL server. Hence I won't be listing them here.)

The column MoreProducts in the denormalized table is in conflict with the first rule. Which is eliminated in the First Normal Form.  
  
The product1, product2, MoreProducts columns in the denomalized table is in conflict with the second rule of first normal form. Hence instead of having multiple columns for products, we put it into multiple rows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SSN** | **UserName** | **EmployerName** | **EmployerAddress** | **Product** |
| 332345432 | Amy | Google | 1 California drive | M |
| 666666666 | Kevin | Facebook | 22nd Street Sanfrancisco | A |
| 666666666 | Kevin | Facebook | 22nd Street Sanfrancisco | B |
| 666666666 | Kevin | Facebook | 22nd Street Sanfrancisco | C |
| 666666666 | Kevin | Facebook | 22nd Street Sanfrancisco | D |
| 919919919 | Raj | Google | 1 California drive | D |

In the above table, following the [rules](http://dotnetanalysis.blogspot.com/2012/01/sql-key-super-key-candidate-key-primary.html)of the primary key, SSN and Product together have been chosen as the primary key.

**Second Normal Form:**

1. The table is in First Normal Form
2. All the non primary key columns in the table should depend on the entire primary key.

*"The following explanations make this more specific:*

* *If the table has a one-column primary key, the attribute must depend on that key.*
* *If the table has a composite primary key, the attribute must depend on the values in all its columns taken as a whole, not on one or some of them.*
* *If the attribute also depends on other columns, they must be columns of a candidate key; that is, columns that are unique in every row."*

  Source: <http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.ddi.doc/ddi54.htm>

In the First Normal Form table above, the SSN+Product columns together form the primary key. The UserName column does not depend on the entire primary key. It only depends on a part of the primary key (ie SSN). The UserName column is clearly in violation of the second rule.  
  
Similarly the reader can also notice that the EmployerName and EmployerAddress columns are also in violation of the second rule.  
  
These violations are fixed in the Second Normal Form shown below.

|  |  |
| --- | --- |
| SSN | UserName |
| 332345432 | Amy |
| 666666666 | Kevin |
| 919919919 | Raj |

|  |  |  |
| --- | --- | --- |
| SSN | EmployerName | EmployerAddress |
| 332345432 | Google | 1 California drive |
| 666666666 | Facebook | 22nd Street Sanfrancisco |
| 919919919 | Google | 1 California drive |

|  |  |
| --- | --- |
| SSN | Product |
| 332345432 | M |
| 666666666 | A |
| 666666666 | B |
| 666666666 | C |
| 666666666 | D |
| 919919919 | D |

In the Second Normal Form, every column is dependent on the entire primary key in that table and not part of the primary key (as was the case in the first normal form).

**Third Normal Form**

1. Database is in Second Normal form
2. There are no transitive dependencies (That is every non primary key is dependent **directly**on the primary key. )

In the Second Normal Form the EmployerAddress column depends on EmployerName which in turn depends on SSN. That is a transitive (indirect) dependency, which needs to be removed in the third normal form.

|  |  |
| --- | --- |
| SSN | UserName |
| 332345432 | Amy |
| 666666666 | Kevin |
| 919919919 | Raj |

|  |  |
| --- | --- |
| SSN | EmployerName |
| 332345432 | Google |
| 666666666 | Facebook |
| 919919919 | Google |

|  |  |
| --- | --- |
| EmployerName | EmployerAddress |
| Google | 1 California drive |
| Facebook | 22nd Street Sanfrancisco |

|  |  |
| --- | --- |
| SSN | Product |
| 332345432 | M |
| 666666666 | A |
| 666666666 | B |
| 666666666 | C |
| 666666666 | D |
| 919919919 | D |

**Third Normal form in the real world applications**  
  
What is explained above is how to lay out a normalized database with natural keys (keys that make sense to the business) . For most practical applications, once this is done, a surrogate key is introduced which replaces the natural key as the primary key. In which case the database would look like this.

|  |  |  |
| --- | --- | --- |
| UserId | SSN | UserName |
| 1 | 332345432 | Amy |
| 2 | 666666666 | Kevin |
| 3 | 919919919 | Raj |

|  |  |  |
| --- | --- | --- |
| UserEmployerId | UserId | EmployerId |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 1 |

|  |  |  |
| --- | --- | --- |
| EmployerId | EmployerName | EmployerAddress |
| 1 | Google | 1 California drive |
| 2 | Facebook | 22nd Street Sanfrancisco |

|  |  |  |
| --- | --- | --- |
| UserProductId | UserId | Product |
| 1 | 1 | M |
| 2 | 2 | A |
| 3 | 2 | B |
| 4 | 2 | C |
| 5 | 2 | D |
| 6 | 3 | D |

Advantages of using surrogate keys:

* Suppose Martha Haley's name changes to to Martha Griffin after marriage. If last name was a part of the primary key, it would be hard to establish that Martha Haley and Martha Griffin are the same person. With surrogate key, this won't happen.
* If one of the columns forming natural keys needs to be updated, the foreign keys can still remain unchanged. (Suppose first name and last name together form a natural key. Even if the person's last name changes, if we are using a surrogate key, the foreign key that connects this table to other tables remains unchanged).
* With surrogate keys joins are faster.
* Popular ORM tools like nhibernate, are more compatible with surrogate keys.

However note that when a surrogate key is introduced, a unique key constraint should be set on the natural key, to preserve the integrity of the data.

The drawback of a normalized database is that data access is slower. As you can see, if I need to find the address of Amy's employer in the 3rd Normal Form, I would have to join 3 tables.  That is why reporting databases should never be normalized beyond 2nd Normal form.