Database Design

Database Terminology

Database: A shared collection of logically related data stored in a structured format.

Table (Entity): A two-dimensional arrangement of rows and columns that contains data.

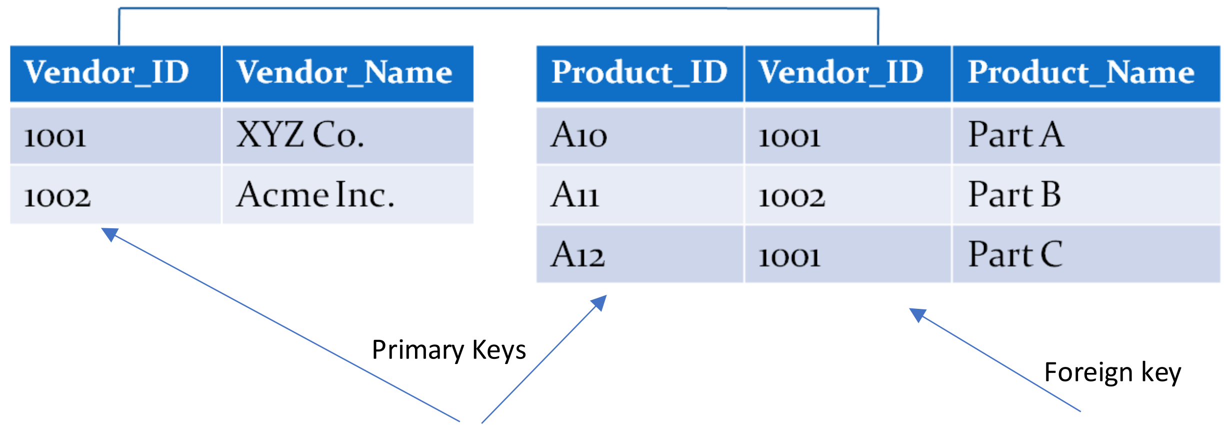
Student_ID	Student_lName	Student_fName	Student_Phone
1001	Smith	Jane	416-555-7656
1002	Jones	Bill	905-555-4736
1003	Jackson	Mary	416-555-2134

Record (Row or Tuple): Each record contains all the information about a single member or item.

Field (Column or Attribute): A record comprises of fields where each field is a single piece of information relating to the record for example your STUDENT record may contain individual fields for first name, last name, student number, date of birth, etc. All records in a table have the same structure and so contain the same fields but obviously each field has its own unique data.

Primary Key: A field or combination of fields that uniquely identifies a record in a table and has a constraint of NOT NULL. A primary key is the minimal fields needed to uniquely identify a record in a table.

Foreign Key: A field that relates to a primary key in an associated table or to a Unique column in another table. A foreign key can sometimes contain NULL values depending upon the constraints/integrity set on the field. A PK and FK are used to "link" tables together.



Vendor_ID	Vendor_Name
1001	XYZ Co.
1002	Acme Inc.

Product_ID	Vendor_ID	Product_Name
A10	1001	Part A
A11	1002	Part B
A12	1001	Part C

Composite Primary Key (compound or concatenated key): A key that consists of 2 or more fields that uniquely identifies a record.

Candidate Key: A field or combination of fields which can uniquely identify a record in a table. A table may have one or more candidate keys and one of them can be used as a Primary Key of the table.

For Example, In an Employee Table, we may have columns like Employee ID, Employee Name, and Employee SSN. We can consider either Employee ID or Employee SSN as Candidate Key's.

Database Integrity

Entity Integrity: Refers to the fact that there are no duplicate rows in a table.

In an Employee table we would not want to store the same information for one employee more than once.

Domain Integrity: Ensures valid entries in a column by restricting the type, range or format of the data values possible. Ie: Setting constraints and data types.

In a Student table we would ensure a student is designated either M or F for male/female respectively. We would therefore set the data type for the field to Char(1), NOT NULL, and constraints of M or F only.

Referential Integrity: Where row data is shared between more than one record this ensures that the data cannot be deleted while there are still dependent records.

In an Invoicing database we will not allow the system to delete a Customer record if that customer has any Invoices.

User-Defined Integrity: You may have your own business rules that do not fall into the other data integrity categories, these can be enforced by "User-Defined Integrity"

These would be based on the business rules defined and the function of the database.

Relationship Types

Definition: A relationship describes an association among tables.

Types: There are 3 types of relationships. 1:1, 1:M, M:N

1. One-to-One (1:1): A retail company may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. The relationship between EMPLOYEE manages STORE is 1:1

2. Many-to-Many (M:N): A student can take many classes and each class can have many students. The relationship between STUDENT takes CLASS is M:N.

3. One-to-Many (1:M): A painter paints many paintings and each painting is painted by one painter. The relationship between PAINTER paints PAINTING is 1:M

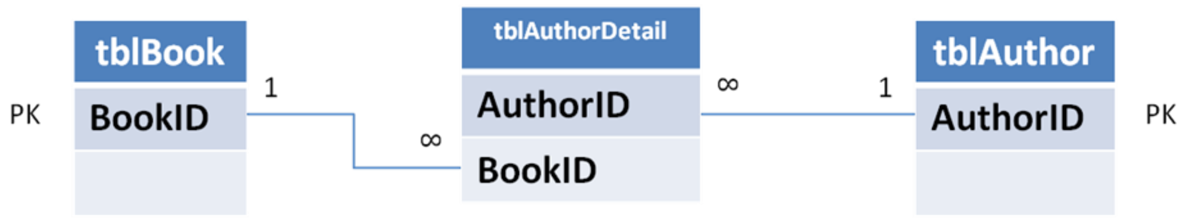
A proper database design will ALWAYS remove the M:N relationship and convert it to two 1:M relationships by the use of a bridge table. Note: new software will allow a M:N but then the design will not be portable and it is very difficult to manage.

Business Rules

Definition: A brief and precise description of a policy, procedure, or principle within the organization.

Why: Business rules are used to define entities (tables), attributes (fields), relationships, and constraints.

How many: Each set of related tables need at least two rules to define their relationship type.



1. Each Author can write many Books (1:M)
2. Each Book can be written by many Authors (1:M)

Note: With the 1st business rule the Book table has a Many relationship. With the 2nd business rule the Author table also has a Many relationship. We therefore have a many to many (M:N) relationship between the tables Book and Author. We remove the M:N relationship by inserting a bridge table between the two tables.

Bridge tables are used to eliminate M:N relationships. The minimum number of fields in any bridge table is the two Primary Keys from the related tables.

How to determine them: The main source of business rules come from company managers, policy makers, department managers, written documentation, and direct interviews with end users (end user sources are less reliable).

Naming Conventions

When designing your database, like any programming, you should get used to using consistant naming conventions. The following are some naming conventions in common use – some are required and some are optional.

1. Precede Objects with prefixes that describe the object. (**optional**)
    i. Tables – tblName
    ii. Forms – frmName
    iii. Queries – qryName
    iv. Reports – rptName
    v. Controls - intName, binName, cboName, etc.

2. Precede field names with the table name minus the prefix (**optional**)
    i. StudentFName (table name is tblStudent)
    ii. EmployeeSIN (table name is tblEmployee)

3. Do not use reserved words. ie: Date, Integer (**required**)

4. Do not use spaces – use underscore. Ie: Client_fName or ClientfName (**optional**)

5. Only use alphanumeric characters (**optional**)

6. Always start an object name with a letter NOT a number (**required**)

Normalization

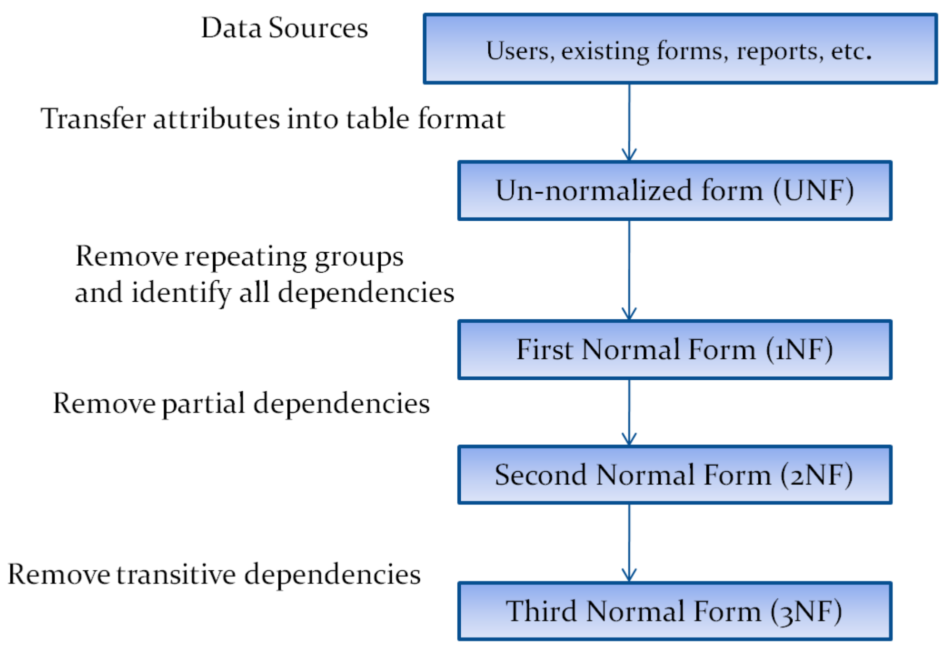
Definition: is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.

Goal: is to reduce redundancies, generate fewer data anomalies, and improve efficiency.

Concepts:

- Normalization works through a series of stages called normal forms.
- The first 3 stages are known as first normal form (1NF), second normal form (2NF), and third normal form (3NF).
- Generally each successive normal form produces more tables in your schema thus producing more joins.
- The more relational joins in a schema the more resources are required by the database system to respond to end-user queries.
    - Thus a good design must also consider end-user demand for fast performance.
- Reducing a normal form to a previous normal form is called denormalization. (The price you pay for increased performance (by denormalization) is greater data redundancy thereby more data anomalies.)

Illustration of the Process



Data Sources
Users, existing forms, reports, etc.
Transfer attributes into table format
Un-normalized form (UNF)
Remove repeating groups and identify all dependencies
First Normal Form (1NF)
Remove partial dependencies
Second Normal Form (2NF)
Remove transitive dependencies
Third Normal Form (3NF)

A Properly Normalized Database Will Result In:

- Each table represents a single subject
- No data item will be unnecessarily stored in more than one table. (Ensures data is updated in only one place)
- All attributes in a table are dependent on the primary key and only the primary key
- All attributes are atomized

The Normalization Process

UNF

Process:

1. Create column headings (ignoring any calculated fields)

2. Enter sample data into table

3. Identify a key for table (and underline it)

4. Remove duplicate data

1NF

Rule: Remove any repeating attributes to a new table

Process:

1. Identify repeating attributes

2. Remove repeating attributes to a new table together with a copy of the key from the UNF table

3. Assign a key to the new table (and underline it). The key from the unnormalised table always becomes part of the key of the new table. A compound key is created. The value for this key must be unique for each entity occurrence.

2NF

Rule: Remove any non-key attributes that only depend on part of the table key to a new table

Ignore tables with a) a simple key or b) with no non-key attributes (these go straight to 2NF with no conversion)

Process:

1. Take each non-key attribute in turn and ask the question
   - is this attribute dependent on one part of the key?

2. If yes, remove attribute to new table with a **copy** of the **part** of the key it is dependent upon. The key it is dependent upon becomes the key in the new table. Underline the key in this new table.

3. If no, check against other part of the key and repeat above process.

4. If still no, ie not dependent on either part of key, keep attribute in current table.

3NF

Rule: Remove to a new table any non-key attributes that are more dependent on other non-key attributes than the table key

Ignore tables with zero or only one non-key attribute (these go straight to 3NF with no conversion).

Process:

- If a non-key attribute is more dependent on another non-key attribute than the table key
- Move the **dependent** attribute, together with a **copy** of the non-key attribute upon which it is dependent, to a new table
- Make the non-key attribute, upon which it is dependent, the key in the new table. Underline the key in this new table.
- **Leave** the non-key attribute, upon which it is dependent, in the original table and mark it as a **foreign key** (*).

Example:

Needs to be Added