Your Data: Friend or Foe?

Eleven Tips for Creating a Reporting-Friendly Database

Here at Windward Studios we've seen numerous examples of how structuring customer data *first* leads to huge time savings in report design *later*. Along the way, we've also seen quite a few common errors.

We're here to help you avoid these mistakes. This paper covers a quick look at the basics of data organization and features eleven useful tips to help you organize your data in a way that will save you time in the long run.

The Principles of Data Access

When you try to access a set of data from another program—any program—the process will run more smoothly when you keep in mind three key principles:

Principle #1: You are using a machine to get information from another machine.

Machines do not speak the language that we do, so working out a problem over coffee isn't going to cut it. You need to have a basic understanding of how machines store and retrieve data.

Principle #2: Duplicate data can occur, but the good news is you can reference it uniquely.

When information on its face appears to be exactly the same but has different definitions by human standards, such as two individuals with the same first and last names and birth date, the machine storing that information usually creates a unique identifier in a separate field or record to make sure you can reference them uniquely.

Principle #3: Relations in data are crucial.

Everyone knows that relationships are important in life, and how you define those relationships and build them directly affects the ease with which you interact with people.

It is no different when it comes to data.
Mapping out a clear structure for your data and knowing how items relate to other items from the start will make it easier to integrate your data with software applications.

Tips to Ensure A Reporting-Friendly Database

Tip #1: Always index your columns.

Indexing is a delicate balance of doing just enough without overdoing it.

Indexing improves the query response time of a select by creating a system-managed table that allows the data to be directly referenced instead of searching for it. But since each modification of data in a user table potentially involves updating the indexes, adding or removing data rapidly can noticeably slow down performance. In addition, not enough indexing will also decrease the performance of SQL selects when querying data.

To achieve this delicate balance, we recommend you:

- **Build your queries with an index order.** If the data being returned is usually ordered by a certain column (e.g., dates) every time, then it makes sense to index the order of that column.
- **Make use of covering indexes.** A covering index consists of all the columns a query needs. This optimizes your query for only the columns contained within it.
- **Rebuild fragmented indexes regularly.** Indexes become fragmented through the modification of table information activity splitting the physical and logical locations, thus creating mismatches.

Tip #2: Separate data into logical pieces and types.

Applications that collect data frequently do not store that data in a logical manner. This most often occurs with text fields. Prime examples are **names**, **addresses**, **dates** and **numerical values**.

If you store a name as a single string, e.g., "First Middle Last," you will encounter problems later when you need to sort by first name only, last name only or a mixture of any of the three. Placing this data into separate columns ensures that you can sort and access your data in an optimized manner.

A trick that DBAs use is to create a query that will assemble and return the full name based on these individual parts using the COALESCE function. This prevents the need for duplicate data by creating an additional "full name" column.

Tip #3: Use views to logically group data from multiple tables.

SQL gives you great power to create complex questions and return a list of organized results. Sadly, the queries themselves are not always easy to construct. Furthermore, if you want to return this data again, you either have to reconstruct the query from scratch or have planned ahead and saved the initial query.

For data grouping that is complex and used often, there is a better way. The SQL 92 specification allows you to save queries in a structure called a "view" so that you can reference it later by its descriptive name.

You do this simply by executing the statement "CREATE VIEW viewname AS [Your Full Select Statement]". You can then reference this VIEW anytime by issuing a "SELECT * FROM viewname" statement. This not only makes

it fast and easy to execute complex queries that you have previously built, but it is simple and error-proof for others in your organization to execute them as well.

We see this often when users are first working with our AutoTag product. They need to correlate data and group results from multiple tables in their database. AutoTag's drag and drop table design makes it easy for them to simply drag a VIEW from our DataBin to the template and pick the columns they want returned in their data set.

Tip #4: Employ NOT NULL unless there is a reason not to.

You might be asking yourself: What is this NULL term?

Keep in mind that even an empty string is a value, and there are times where you need blank items like this in your database fields. **NULL is a placeholder in your database that represents missing or unknown data.**

In most databases that conform to the SQL 92 specification, you have the option to specify adding NULL values for missing or unknown data to your tables and columns. This is useful because you can easily create queries excluding values that are not equal to NULL, therefore removing any blank or incomplete data results from your query.

> NOTE: Remember that NULL is not equal to 0 (zero), and you cannot create this comparison/ filter in an SQL query. This is how NULL helps you keep your incomplete or unknown data labeled with a standard recognizable value, making it very easy to filter out of your queries.

Tip #5: Don't use separators in your data.

When storing long strings of data, end users can be prone to creating entries with separators (e.g., City/State) to signify different segments of data in the string.

First off, just...don't. This is a bad practice because SQL utilizes many special characters in queries themselves, and this could interfere with an otherwise properly functioning query.

Worse yet, the programming language you are using to handle queries and results from your queries also has reserved keywords and characters that may break your code as well. Common examples we have seen:

- City/State
- Comma-delimited lists
- Semicolon-delimited lists

But this applies to more than long strings. It also applies to data types such as:

- Phone numbers
- Social security numbers
- Driver's license numbers

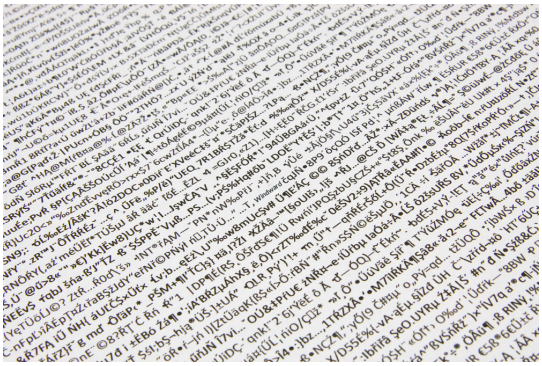
Remember, you can always use SQL functions or your native programming language to manipulate a text string and print it in the form you desire. But when encoding the information in your database, it is always best to stick with storing the information without separators. It is less work for you when parsing it in your native language, and it will cause fewer headaches in your queries and code in the long run.

RECOMMENDATION: If you find your fingers hovering towards the shift key and top row of your keyboard when entering data, take a moment to stop, drop your hands and roll over to a new way of entering this information without separators.

Tip #6: Properly structure your metadata.

Data typing is crucial when organizing your data. Storing dates and currencies in different formats leads to unexpected results when applications interface with your data structure.

If a date is stored as text, you will need to transform it using SQL functions into an SQL DATETIME object. This causes extra processing time and reduces the overall performance of your queries. The same can be said with currencies. Storing the currency symbol or storing the currency in a non-standard format, e.g., 1.000,56 instead of 1000.56, will also cause inconsistencies and errors with applications trying to utilize these currencies.



Therefore, it is best to encode all numerical values in the standard decimal format, which you can later transpose to other formats based on SQL functions. You can define this typing when you create your SQL column by specifying the column encoding. Examples are DATE, TIME, INTEGER, SMALLINT, BIGINT, DECIMAL, FLOAT, and VARCHAR, to mention a few.

Tip # 7: Be aware of the character sets used by your data.

Decades ago, data was encoded in 8 bits, or 256 unique characters, which seemed like a lot back then. But over time, more languages entered the computing landscape—languages like Mandarin Chinese, Thai, Korean, Japanese, Arabic and Russian—and suddenly 256 characters was not nearly enough to house an entire language symbol set.

So the developer powers-that-be got together and implemented an encoding called Universal Transfer Character Set Transformation Format, or as everyone else lovingly calls it, UTF-8. This is basically like a shift key for your keyboard that, in relation to the language set, allows combinations of character set number to be combined to create a new character reference. 32-bit encoding allows much more information to be encoded.

Okay, so great, we can now encode all the symbols that exist in these languages. Why should you care? You set it (your data to UTF-8) and forget it, right?

Not so fast.

The character set can only be defined at the database, schema, table or column level. UTF-8 is a 32-bit character field, which takes up memory and decreases performance, so blanket encoding all fields to UTF-8 would be a waste. The best practice is to reserve setting UTF-8 encoding for large fields of text such as MD5 password hashes, Web addresses and lengthy internal codes or block text.

RECOMMENDATION: When possible, avoid setting your VARCHAR values to UTF-8. It takes 765 bytes to store a single byte of data.

Tip #8: Remember that duplicate data is bad, period.

Data takes up space. Space requires hardware to store it. Hardware costs money. Searching and organizing data costs time, and time is money.

Therefore, the more duplicate data your system has, the more you are costing your organization by storing and processing it.

The most common example of this we see is with storing name objects. Consider an instance where you have a customer list. To add a new customer, you enter the first name and last name, and your customer is created. Then suppose this customer later becomes a reseller of your product, so you enter the first name and last name in the reseller table.

For a few small instances this may work, but as your organization grows you may convert many customers to resellers. Suddenly you have a duplicate data problem on your hands.

You can solve this conundrum easily by creating a person table in the database. Each person is an individual entity with a first name, last name and other personal information. If a person is a customer then we can create a foreign key in the customers table that matches the primary (indexed, see why indexes are helpful?) key in the person table.

The same process can take place in the reseller table. Suddenly we no longer have duplicate data, and we can now more powerfully filter our data because we have applied Tip #2 above to our table structures.

The best way to identify duplicate data is to first look for duplicate entry. If this is occurring, stop and ask yourself if there is a better way to organize the data so it only needs to be entered once and can be referred to many times by multiple items.

Tip #9: Data can be referenced in different dimensions, so reference your data effectively.

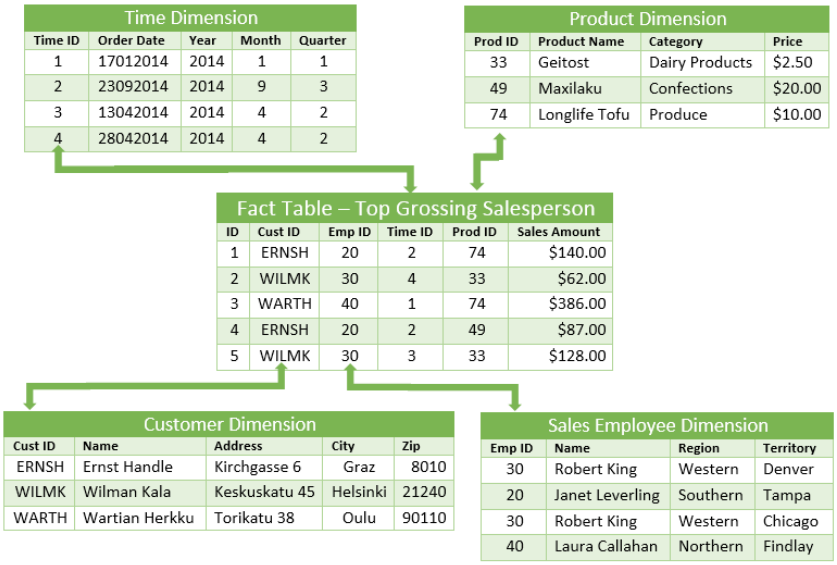
Duplicate data occurs not only at the row/column level but also at a tabular and database level.

You can keep data references between tables in order by utilizing foreign key and primary key relationships, but you may need to filter and correlate this data across different databases as well. The key item to pay attention

to is the fact that data correlation can occur in different dimensions. This can be very powerful—but if you are not careful, it can also generate very confusing data result sets.

Take an example where you have two databases: 1) an Orders database containing order information, and 2) a Sales database containing sales information. There are also four tables that call upon data stored in these databases: a table of orders, a table of order detail, a table of sales people and a table of regions.

Now suppose you want to return the result of the top-grossing sales people, ordered by region, for each month. You need to relate the four tables with information stored in two different databases, and the one key piece of information to filter on is total sales.



Time Dimension

Time ID	Order Date	Year	Month	Quarter
1	17012014	2014	1	1
2	23092014	2014	9	3
3	13042014	2014	4	2
4	28042014	2014	4	2

Product Dimension

Prod ID	Product Name	Category	Price
33	Geitost	Dairy Products	$2.50
49	Maxilaku	Confections	$20.00
74	Longlife Tofu	Produce	$10.00

Fact Table – Top Grossing Salesperson

ID	Cust ID	Emp ID	Time ID	Prod ID	Sales Amount
1	ERNSH	20	2	74	$140.00
2	WILMK	30	4	33	$62.00
3	WARTH	40	1	74	$386.00
4	ERNSH	20	2	49	$87.00
5	WILMK	30	3	33	$128.00

Customer Dimension

Cust ID	Name	Address	City	Zip
ERNSH	Ernst Handle	Kirchgasse 6	Graz	8010
WILMK	Wilman Kala	Keskuskatu 45	Helsinki	21240
WARTH	Wartian Herkku	Torikatu 38	Oulu	90110

Sales Employee Dimension

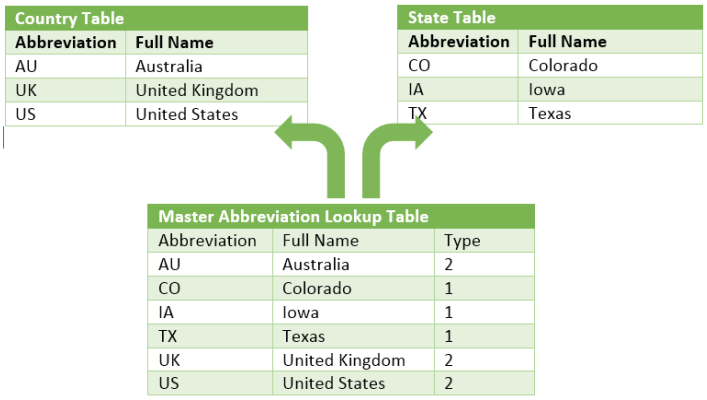
Emp ID	Name	Region	Territory
30	Robert King	Western	Denver
20	Janet Leverling	Southern	Tampa
30	Robert King	Western	Chicago
40	Laura Callahan	Northern	Findlay

Here it makes sense to create what is referred to in the industry as a **fact table**, a single table that contains the figures that tie the different data dimensions together and then makes use of foreign key/primary key relationships to link to these databases and tables.

In our example, we create a table with foreign keys linking to the orders table, sales table and region table. The orders table returns the total amount of each order to a column in the fact table based upon the order details table. You could run a query on the total sales fact table to return all sales people in a filtered region during a date range. We could take this further and create another table that uses SQL SUM functions to create monthly totals for each salesperson grouped by region and filtered by data range.

This prevents the need to create complex relationships directly in a query that only the DBA could assemble, allowing someone referencing the database to access the information in a single table in an intuitive manner.

Tip #10: Make use of lookup tables to prevent grouping different data in a single table.



Country Table

Abbreviation	Full Name
AU	Australia
UK	United Kingdom
US	United States

State Table

Abbreviation	Full Name
CO	Colorado
IA	Iowa
TX	Texas

Master Abbreviation Lookup Table

Abbreviation	Full Name	Type
AU	Australia	2
CO	Colorado	1
IA	Iowa	1
TX	Texas	1
UK	United Kingdom	2
US	United States	2

Let's face it: Organizations need internal codes, and those who work with these codes day in and day out know the codes by heart. The medical billing industry is notorious for this, and we at Windward often encounter abbreviations for states, regions and countries.

But those who don't know the codes by heart need what is called a **lookup table**, or hash table, that relates the shortened code version to the full text version.

Referencing these full name values by their abbreviation equivalent is a great way to keep larger VARCHAR values minimized on other tables while allowing the select to return those large values as needed.

RECOMMENDATION: We advise staying away from combining these lookup tables in a single table and denoting their differences by a type field. This creates a large single table with disparate data and can cause difficulty when trying to extract the values you truly need.

Tip #11: Use proper naming strategies (i.e., human-readable) for your data.

Humans love to be creative when naming things. Programmers and IT staff take this to a whole new level, often searching a popular domain dedicated to Naming Schemes. And while we may love referring to all the items we work with by their Nordic God names, when we expect other people not living in our world to work with it, well, it can become a problem.

Keeping your naming structures human-readable and intuitive is key to getting the most use out of your data sets. Naming your databases, tables and columns obscure names will only cause confusion and sometimes latency. (We have seen table names that are VERY long -- on the order of 80-100 characters. This is not only a pain to look at from a database architecture view but even worse for the poor soul who has to write SQL queries against them.)

So what do you do when you have a creative DBA or you yourself are this DBA with a fixation for non-intuitive naming?

Remember that ALIAS is your friend.

The SQL ALIAS command is structured as follows:

For Column Level Aliases
SELECT column_name AS alias_name FROM table_name;

For Table Level Aliases
SELECT column_name(s) FROM table_name AS alias_name;

An example will show how changing the column or table name into an intuitive alias can not only shorten your SQL queries but also make your database more usable for end users.

EXAMPLE

SELECT eska AS Name, essa AS Home, meena AS Region FROM OneRingToRuleThemAllLordofTheRingsTrilogy as LOTR WHERE LOTR.Home=”Shire” AND LOTR.Name=”Frodo Baggins”

The full text would appear like this without the use of Aliases

SELECT eska, essa, meena FROM OneRingToRuleThemAllLordofTheRingsTrilogy WHERE OneRingToRuleThemAllLordofTheRingsTrilogy.meena =”Shire” AND OneRingToRuleThemAllLordofTheRingsTrilogy.eska =”Frodo Baggins”

You can see that column names labeled in Elvish, as well as very long table names, are difficult to type and not easy to grasp quickly. This makes it easy to make mistakes in queries. Using aliases to keep thing short, simple and intuitive will save you many headaches for yourself and your database users down the road.

Well-Structured Data Leads to Efficient Reporting

We hope you've found these tips helpful. As you put them to use, remember to let your reporting software do as much of the work for you as possible when accessing your data.

If you find that your current database-reporting software makes for a clunky experience, we invite you to see how easy it is to create reports in Windward.

At Windward Studios we believe that reporting and document generation should be simple—not overly complex, tedious and technical. Your reports deserve to look as impressive as the information they contain.

Why can't designing documents linked to your databases be as easy as creating a Word document, Excel spreadsheet or PowerPoint deck?

About Windward

For businesses in document-intensive industries, Windward Studios is the document generation and reporting software company that empowers business professionals to create beautiful, professional reports.

Windward OEM and enterprise customers span over 70 countries and all industries, including financial services, insurance, energy, healthcare, HR and technology. We've been delighting customers since 2004.

Our primary products are AutoTag , a design tool that creates custom templates with Microsoft Office, and a Java or .NET engine that connects to virtually any data source.

Windward delivers exceptional support, training, and documentation, with a 98% satisfaction rating from our customers. We're a Microsoft Gold Partner and an Oracle Gold Partner.

It can. Windward creates software applications that simplify how businesses design and generate professional reports and documents. Windward provides a unique experience using Microsoft Office to format and edit report templates, and the sophisticated engine pulls data from multiple sources and merges that data into those documents. **It's a hassle-free experience that can actually make generating reports fun.**

For details about the technology, its applications, and how it has been successfully deployed to create beautiful reports from nearly every existing data source, please visit the Windward Studios website for an overview video and free trial.

Improve Your Software Product's Reporting and Document Generation

Download a free trial of Windward's products or schedule a live demonstration.