Keys and the Date Dimension

# Keys

## Review

Before we discuss keys in our dimensional model, let’s review some of what we already know about keys:

**Key**

A key is a value or group of values that can be used to uniquely identify a record in a database.

**Surrogate Key**

A surrogate key is a key that is generated by the DBMS for the purposes of uniquely identifying records. It does not have meaning outside of the context of the database.

**Natural Key**

A natural key is a key that has a real-world meaning. It is passed into the database rather than being generated by the DBMS. It is also referred to as a **business key**, especially in the context of data warehousing.

**Business Key**

Another name for a natural key.

**Primary Key**

The key that serves as the primary method of identifying records in a table.

**Alternate Key**

Any value or collection of values that could be used to uniquely identify records on a table, but is not the primary key. Often this is a natural key on a table where a surrogate key is being used.

**Smart Key**

A key that is built using a specific pattern based on properties of the record. For example, we might represent an employee record using their first initial, last name, and department code. Smart keys are uncommon, because they have to be based on information that will always be unique and never change.

## Fact Table Keys

As we have learned, in a star schema dimensional model, fact tables reference dimension tables. This means that fact tables often have a large number of foreign key columns, one for each dimension they reference.

We also have learned that the maximum granularity of a fact table is defined by its relationships with dimensions. If we are creating a fact table with a coarser grain than the source data, we must aggregate any data that shares the same values for each of the dimensions.

For example, if we create a fact table to track sales by store by day, but the source has sales by salesperson by store by day, any salespeople in the same store would have their sales added together before being inserted into our fact table.

As a result, every combination of foreign keys in a fact table is unique. Any duplicates would have their values combined in some way. This means that each fact table has an alternate key comprised of all of the foreign keys on that table.

The combination of all foreign keys on a fact table should be an alternate key for that table. This alternate key defines the finest granularity possible on the table.

Let’s review the model we started last class, and add foreign key columns as appropriate:



Our **FactOrders** table references five dimensions – DimCustomers, DimCities, DimProducts, DimSalesPeople, and DimDate. Although we have not fleshed out each of those tables yet, we can create a foreign key column for each of them in anticipation of finishing them off.

Let’s do this in a new database called WWIDM (i.e. WideWorldImporters Data Mart):

CREATE DATABASE WWIDM;

GO

Now let’s start our fact table:

CREATE TABLE dbo.FactOrders (

-- Should we create a surrogate key?

CustomerKey INT NOT NULL,

CityKey INT NOT NULL,

ProductKey INT NOT NULL,

SalespersonKey INT NOT NULL,

DateKey INT NOT NULL,

Quantity INT NOT NULL,

UnitPrice DECIMAL(18,2) NOT NULL,

TaxRate DECIMAL(18,3) NOT NULL,

TotalBeforeTax DECIMAL(18,2) NOT NULL,

TotalAfterTax DECIMAL(18,2) NOT NULL

);

Notice we have called our foreign key columns “key” instead of “id”. This is a common naming convention in data warehousing. Notice we have also omitted the “dim” prefix from the columns. In a dimensional model, foreign keys always reference a dimension table, so the “dim” prefix is not necessary and can be assumed.

## Fact Table Surrogate Keys

In our code above, we haven’t defined a surrogate key and we haven’t defined a primary key. Is a surrogate primary key beneficial?

Since dimensions give fact tables context, it’s not usually meaningful to access individual fact records directly. Most queries against a fact table join to at least one dimension to help identify records. In most cases, a surrogate key is not helpful, so fact tables exclude them and are implemented as a heap.

There are a few notable exceptions:

* Some advanced ETL techniques require a surrogate key.
* In cases where ETL is designed to resume processing, rather than restart from the beginning.
* Rarely, the business process being modelled prevents the dimensions from uniquely identifying a fact record. In these cases, a surrogate key can be used to keep them straight.

None of these cases apply to us, so we will leave our fact table as-is.

**Best Practice**

Do not create surrogate keys or primary keys for your fact tables, unless you have a specific technical limitation that requires you to do so.

## Indexing Fact Tables

Indexing fact tables is usually very straightforward.

We use dimension tables to define our fact table. This means that we will usually seek records in the dimension tables, then seek their matching records in the fact table. So, while our dimension tables might benefit from complex indexes, a fact table typically only needs a simple set of foreign key indexes.

*NOTE: In SQL Server 2014 and later there is a special kind of index that is particularly suited to fact tables called columnstore indexes. These indexes are outside the scope of the course, but are worth researching if you have an interest in data warehousing.*

Let’s create the indexes for our fact table:

CREATE INDEX IX\_FactOrders\_CustomerKey ON dbo.FactOrders(CustomerKey);

CREATE INDEX IX\_FactOrders\_CityKey ON dbo.FactOrders(CityKey);

CREATE INDEX IX\_FactOrders\_ProductKey ON dbo.FactOrders(ProductKey);

CREATE INDEX IX\_FactOrders\_SalespersonKey ON dbo.FactOrders(SalespersonKey);

CREATE INDEX IX\_FactOrders\_DateKey ON dbo.FactOrders(DateKey);

# Date Dimensions

## Basics

The date dimension is perhaps the most common and most fundamental dimension in any dimensional model. It is often the first dimension built in any new data warehouse.

Let’s look at some of the most common elements of a date dimension and start to learn about how dimensions are used. For our example, we are going to use the following structure:

CREATE TABLE dbo.DimDate (

DateKey INT NOT NULL,

DateValue DATE NOT NULL,

Year SMALLINT NOT NULL,

Month TINYINT NOT NULL,

Day TINYINT NOT NULL,

Quarter TINYINT NOT NULL,

StartOfMonth DATE NOT NULL,

EndOfMonth DATE NOT NULL,

MonthName VARCHAR(9) NOT NULL,

DayOfWeekName VARCHAR(9) NOT NULL,

CONSTRAINT PK\_DimDate PRIMARY KEY ( DateKey )

);

There are a few things to note here:

* We have created a DateKey, but not defined it as an identity. Date keys are a good example of where a **smart key** can be very beneficial. By defining an integer as YYYYMMDD, we know we have a key that will always be unique and will never change.
* All of the columns in our date dimension are easily calculable using the date itself. As we have learned, this kind of denormalization is acceptable in a dimensional model. Performance and ease-of-use are higher priorities than write integrity.
* We have a primary key on our table. Dimensions are primary tables for our fact tables, so we need a good primary key on our table for the fact tables to reference.
* We have not created any indexes yet. Indexes for dimensions are usually designed around specific usage patterns, whether that is anticipated through requirements or discovered through examining how end users use the data mart. We will hold off on creating indexes until we know more.
* Columns in a dimension provide us with ways to aggregate associated fact tables. For example, if we wanted to find all our sales for a particular year, we could do that by joining to the date dimension and filtering on the Year column.

## Loading the Date Dimension

Date dimensions are unique in that their data does not usually come from a source system – we can calculate all the values we need.

For our simple example, let’s create a stored procedure that could be used to insert a new record into the date dimension.

CREATE PROCEDURE dbo.DimDate\_Load

@DateValue DATE

AS

BEGIN;

INSERT INTO dbo.DimDate

SELECT CAST( YEAR(@DateValue) \* 10000 + MONTH(@DateValue) \* 100 + DAY(@DateValue) AS INT),

@DateValue,

YEAR(@DateValue),

MONTH(@DateValue),

DAY(@DateValue),

DATEPART(qq,@DateValue),

DATEADD(DAY,1,EOMONTH(@DateValue,-1)),

EOMONTH(@DateValue),

DATENAME(mm,@DateValue),

DATENAME(dw,@DateValue);

END;

We can test it out using the following:

EXECUTE dbo.DimDate\_Load '2017-12-01';

SELECT \* FROM dbo.DimDate;

Note what happens if you run it twice. Does that make sense? Would we ever want to change or overwrite a record in this table?

Every column in our table is based on DateValue. If the date value changes, it’s a new date. It would not be meaningful to ever try to change any of the values here.