SQL Fundamentals

Course Objectives

Basic SQL Queries

Retrieve and filter data from a relational database using basic SQL syntax.

Manipulating Values

Transform your data with the help of numerical, date and text functions.

SQL Theory

Understand essential terminology related to SQL, databases and data warehouses.

Working With Multiple Tables

Write queries that combine data from multiple tables.

SQL for Reporting

Use SQL in popular BI tools, and learn how to summarize the results of SQL queries.

SQL Fundamentals
Basic SQL Queries

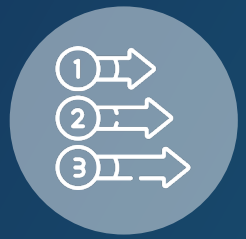
Basic SQL Queries - Section Objectives

Tasks

01. Retrieve data from a relational database using **basic SQL queries**.

02. **Filter** the results and **remove duplicate** values

Skills



Order of Operations

SELECT & FROM Clause

WHERE Clause

GROUP BY and HAVING

ORDER BY

TOP N, OFFSET-FETCH, DISTINCT

Applied Steps

Video – Install and Intro to Azure Data Studio	
1	Do a web search for **'Download and install azure data studio'**
2	**Download the latest release** and install it. (Do not click Download SQL Server)

Video – Creating a connection to a database	
1	Open SQL Operations Studio/Azure Data Studio
2	Click new connection & enter the credentials provided in the student files folder.
5	Expand the database folder, you should see **AdventureWorksDW**
6	Right click on the newly connected query icon and **Select New Query**

Adventure Works Intro

Company Overview

- Large, multinational manufacturing company
- Focused on metal and composite bicycles to North American, European, and Australian commercial markets
- 290 employees
- Several regional sales teams

Business Goals:

- Broaden market share by targeting their sales
- Extend product availability through an external Website

Database Connection Details

Please refer to the SQL Credentials text file in your **SQL Student Files** folder

Example SQL Code

Create a **new query** and **type the following code**. Run the code using the **PLAY** button.

SQL Code	
Query	**SELECT** CustomerKey **AS** CustomerID, **SUM**(SalesAmount) **AS** SalesAmount **FROM** FactInternetSales **WHERE** YEAR(OrderDate) > 2020 **GROUP BY** CustomerKey **HAVING SUM**(SalesAmount) > 10000 **ORDER BY** SalesAmount **DESC**

Saving your queries in Notepadd++

An advanced notepad is a great way to **create a code repository**, for SQL and other coding languages.

NOTEPAD++ (Windows)

Bbedit 13 (MAC)

SELECT and FROM

SELECT and **FROM** are the two most basic key words that form part of any SQL query.

SELECT * allows us to see all the columns in a table, however we should only use this for testing.

SQL Code	
Query	**SELECT** * **FROM** FactInternetSales

SELECT specific columns

We can **include specific columns** in our SELECT statement to make our query more specific.

SQL Code	
Query	**SELECT** SalesOrderNumber, OrderDate, SalesAmount, TaxAmt, OrderQuantity **FROM** FactInternetSales

Creating a column alias

A column alias allows us to **modify the database name** of a column**.**

SQL Code	
Code	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, SalesAmount, TaxAmt, OrderQuantity **FROM** FactInternetSales

Using WHERE to filter rows

WHERE allows us to filter the rows that are selected in the **FROM** tables.

SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, SalesAmount, TaxAmt, OrderQuantity, SalesTerritoryKey **FROM** FactInternetSales **WHERE** SalesTerritoryKey = 6

After using a WHERE filter, use the **ROW count** at the bottom of the query to see how many rows remain.

GROUP BY

FactSales

SalesNumber	LineNumber	Amount
SO24982	1	3255
SO24982	2	40
SO24982	3	565
SO34506	1	25
SO32452	1	99.99
SO53142	1	27.99
SO53142	2	48.99

```
SELECT

SalesNumber,
SUM(Amount) AS SalesAmount,

FROM FactSales

WHERE SalesNumber = 'SO24982'

GROUP BY SalesNumber
```

Used to group the data

Groups are determined by the attributes we specify in the GROUP BY

Aggregate function are calculated for each group

SalesNumber	SalesAmount
SO24982	3860

Limiting results to 1 invoice number for testing

We can use **WHERE** to **filter to a single invoice number** to help us visualize the outcome.

SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, SalesAmount, TaxAmt, OrderQuantity **FROM** FactInternetSales --WHERE SalesTerritoryKey = 6 **WHERE** SalesOrderNumber = 'S051182'

Using GROUP BY to combine invoice numbers

The **GROUP BY** function allows us to **combine rows** in our output table.

SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, **SUM**(SalesAmount) **AS** InvoiceSubTotal, **SUM**(TaxAmt) **AS** TaxAmount, **SUM**(OrderQuantity) **AS** TotalQuantity **FROM** FactInternetSales --WHERE SalesTerritoryKey = 6 **WHERE** SalesOrderNumber = 'S051182' **GROUP BY** SalesOrderNumber, OrderDate

Filtering grouped invoices with HAVING

The **HAVING** statement allows us to **filter grouped rows** in our output table.

SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, **SUM**(SalesAmount) **AS** InvoiceSubTotal, **SUM**(TaxAmt) **AS** TaxAmount, **SUM**(OrderQuantity) **AS** TotalQuantity **FROM** FactInternetSales **WHERE** SalesTerritoryKey = 6 **--WHERE** SalesOrderNumber = 'S051182' **GROUP BY** SalesOrderNumber, OrderDate **HAVING SUM**(SalesAmount) > 1000

SQL Order of Operations

SQL code is not processed in the order it is written.

In particular, the **SELECT** statement is processed later on.

This means the results of the SELECT statement such as column aliases cannot be used in earlier steps.

Using ORDER BY to sort query rows

The **ORDER BY** statement allows us to **sort the final rows** in our table.

SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, **SUM**(SalesAmount) **AS** InvoiceSubTotal, **SUM**(TaxAmt) **AS** TaxAmount, **SUM**(OrderQuantity) **AS** TotalQuantity **FROM** FactInternetSales **WHERE** SalesTerritoryKey = 6 **--WHERE** SalesOrderNumber = 'S051182' **GROUP BY** SalesOrderNumber, OrderDate **HAVING SUM**(SalesAmount) > 1000 **ORDER BY** InvoiceSubTotal ASC

Recap and common errors

The **SELECT** statement is processed after the **HAVING** statement.

Derived columns in the SELECT statement **cannot refer to other column aliases**.

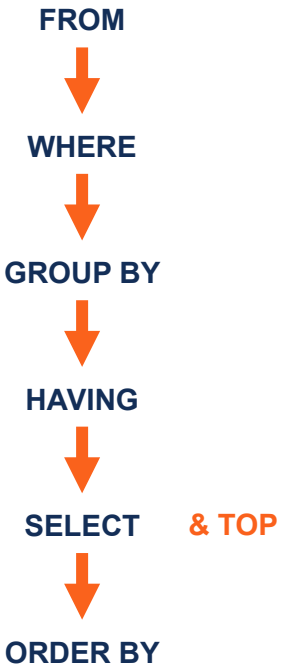
SQL Code	
Query	**SELECT** SalesOrderNumber **AS** InvoiceNumber, OrderDate, **SUM**(SalesAmount) **AS** InvoiceSubTotal, **SUM**(TaxAmt) **AS** TaxAmount, **SUM**(OrderQuantity) **AS** TotalQuantity, **SUM**(SalesAmount) + **SUM** (TaxAmt) AS InvoiceTotal **FROM** FactInternetSales **WHERE** SalesTerritoryKey = 6 **--WHERE** SalesOrderNumber = 'S051182' **GROUP BY** SalesOrderNumber, OrderDate **HAVING SUM**(SalesAmount) > 1000 **ORDER BY** InvoiceSubTotal ASC

Filtering Rows: TOP N

The **TOP (N)** statement allows us to return the **first N rows** from our output table.

You will need to use **ORDER BY** to ensure you get the true **TOP rows**.

SQL Code	
Query	**SELECT TOP(10)** SalesOrderNumber **AS** InvoiceNumber, OrderDate, **SUM**(SalesAmount) **AS** InvoiceSubTotal, **SUM**(TaxAmt) **AS** TaxAmount, **SUM**(OrderQuantity) **AS** TotalQuantity, **SUM**(SalesAmount) + **SUM** (TaxAmt) AS InvoiceTotal **FROM** FactInternetSales **WHERE** SalesTerritoryKey = 6 **--WHERE** SalesOrderNumber = 'S051182' **GROUP BY** SalesOrderNumber, OrderDate **HAVING SUM**(SalesAmount) > 1000 **ORDER BY** InvoiceSubTotal ASC



Filtering Rows: TOP N Percent

The **TOP (N) PERCENT** statement allows us to return the **first N % of rows** from our output table.

SQL Code

Query

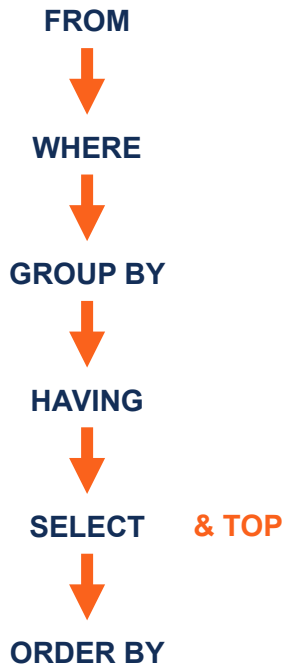
```
SELECT TOP(10) PERCENT

SalesOrderNumber AS InvoiceNumber,
OrderDate,
SUM(SalesAmount) AS InvoiceSubTotal,
SUM(TaxAmt) AS TaxAmount,
SUM(OrderQuantity) AS TotalQuantity,
SUM(SalesAmount) + SUM (TaxAmt) AS InvoiceTotal

FROM FactInternetSales
WHERE SalesTerritoryKey = 6
--WHERE SalesOrderNumber = 'S051182'

GROUP BY SalesOrderNumber, OrderDate
HAVING SUM(SalesAmount) > 1000

ORDER BY InvoiceSubTotal ASC
```



Filtering Rows: OFFSET FETCH

The **OFFSET FETCH** statement is similar to TOP but allows us to **skip the first N rows** from our output table.

SQL Code

Query

```
SELECT
SalesOrderNumber AS InvoiceNumber,
OrderDate,
SUM(SalesAmount) AS InvoiceSubTotal,
SUM(TaxAmt) AS TaxAmount,
SUM(OrderQuantity) AS TotalQuantity,
SUM(SalesAmount) + SUM (TaxAmt) AS InvoiceTotal

FROM FactInternetSales
WHERE SalesTerritoryKey = 6
--WHERE SalesOrderNumber = 'S051182'

GROUP BY SalesOrderNumber, OrderDate
HAVING SUM(SalesAmount) > 1000

ORDER BY InvoiceSubTotal ASC

OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
```

FROM
↓
WHERE
↓
GROUP BY
↓
HAVING
↓
SELECT
↓
ORDER BY
↓
OFFSET FETCH

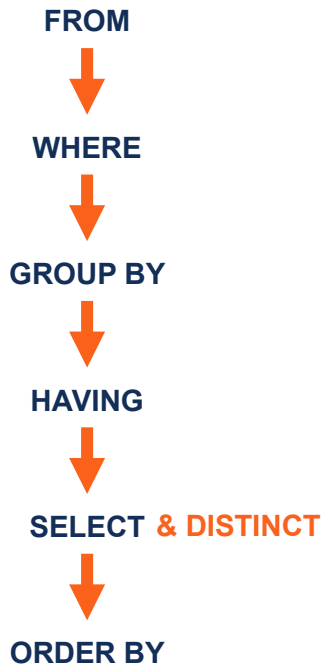
Filtering Rows: DISTINCT

The **DISTINCT** statement allows us to return **a unique list of rows** in the output table.

SQL Code

Query

```
SELECT  DISTINCT
CustomerKey
FROM FactInternetSales
ORDER BY CustomerKey
```



ORDER BY another column

We can order our final output table **using a column that is not present** in our output table.

SQL Code	
Query	**SELECT** CustomerKey **FROM** FactInternetSales **ORDER BY** SalesAmount

SQL Fundamentals: Student Exercise 1a

Create a list of product costs, grouped by invoice numbers.

1. Write a query to return InvoiceNumber and TotalProductCost from the FactInternetSales table.

1. Return only invoices that HAVE a total product cost per Invoice Number > 2000.

Hint: You first need to group by the invoice to get the total and then filter.

	InvoiceNumber	TotalProductCost
1	SO43697	2171.2942
2	SO43702	2171.2942
3	SO43703	2171.2942
4	SO43706	2171.2942
5	SO43707	2171.2942
6	SO43709	2171.2942
7	SO43710	2171.2942
..		
1551	SO46602	2171.2942

SQL Fundamentals: Student Exercise 1b

We need a detailed list of invoices and invoice line numbers, but we're only interested in currency key 100.

1. Write a query to return InvoiceNumber, Invoice LineNumber and SalesAmount from the FactInternetSales table.

1. Return only lines WHERE the currency key is 100.

Hint: Since line number is the lowest level of detail in the FactInternetSales table, you won't need to use GROUP BY.

	InvoiceNumber	InvoiceLineNumber	SalesAmount
1	SO43699	1	3399.99
2	SO43700	1	699.0982
3	SO43702	1	3578.27
4	SO43706	1	3578.27
5	SO43707	1	3578.27
6	SO43711	1	3578.27
7	SO43713	1	3578.27
...			
33400	SO75123	3	8.99

SQL Fundamentals: Student Exercise 1c

We have a new data analyst in the team who wants to see a unique list of sales territory keys. This will help her to better understand the database.

1. Write a query to return the sales territory column from the FactInternetSales table.

1. Return a unique list of territories only.

1. Order the results alphabetically for ease.

	SalesTerritory
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

SQL Fundamentals
Manipulating Values

Manipulating Values - Section Objectives

Tasks

01. Transform your data with the help of **numerical, date and text functions**.

02. Filter our data by **creating conditions** that test if something is true or false.

Skills

Data Types

Functions

Comparison Operators

Logical Operators

Logical Functions

Data Type Conversion

Aggregate Functions

Aggregate functions help define how we want values to be treated when we use a GROUP BY in our query.

Function	Description
SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM works with numeric columns only. Null values are ignored.
AVG	Returns the average of the values in a group. It ignores null values.
COUNT	Returns the number of items found in a group.
MAX	Returns the maximum value in a group.
MIN	Returns the minimum value in a group

Counting rows with COUNT(*) aggregation

COUNT(*) is a special aggregated function that allows us to **count the rows** in a table.

Aggregated functions cannot be used with non-aggregated columns.

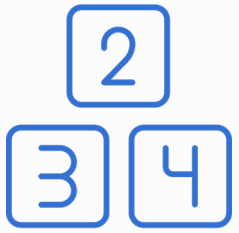
SQL Code	
Query	**SELECT** **COUNT**(*) **AS** TotalCustomers, **AVG**(YearlyIncome) **AS** AverageIncome, --YearlyIncome **AS** TotalIncome –Non aggregated columns cannot be used alongside aggregated columns **FROM** DimCustomer

How aggregate functions respond to NULL values

Aggregate functions **ignore NULL** values**.** For example **COUNT will not include NULL values.**

SQL Code	
Query 1	--Use this query to look at everything in the customer table. **SELECT** * **FROM** DimCustomer
Query 2	--Use this query to count the total number of customers **SELECT** COUNT(*) **AS** TotalCustomers **FROM** DimCustomer
Query 3	--Notice that when you count the same table by middle name, you get a lower count. That's because it ignore the NULLs. **SELECT** COUNT(MiddleName) AS MiddleNameCount **FROM** DimCustomer

Data Types



Numeric

Date & Time

String

Other

Numeric Data Types

Data Type	Description	Storage
bit	An integer data type that can take a value of 1, 0, or NULL	1 byte
tinyint	whole numbers from 0 to 255	1 byte
smallint	whole numbers between -32,768 and 32,767	2 bytes
int	whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
Decimal(p,s)	P = The maximum number of decimal digits to be stored. This number includes both the left and the right sides of the decimal point. S = The number of decimal digits that are stored to the right of the decimal point. Ex. decimal (4,2), 2 digits before the decimal point, two digits after. Allows up to +/- 10^38.	5-17 bytes
Numeric(p,s)	Same as above	5-17 bytes
Smallmoney	Accurate to a ten-thousandth of the monetary units, from - 214,748.3648 to 214,748.3647	4 bytes
money	Accurate to a ten-thousandth of the monetary units, up to +/- 922,337,203,685,477.5808	8 bytes
Float(n)	n is the number of bits that are used to store the mantissa of the float number and, therefore, dictates the precision and storage size. If n is specified, it must be a value between 1 and 53. The default value of n is 53. Takes values from -1.79E + 308 to 1.79E + 308.	4-8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Numeric Functions

Function	Description	Statement	Result
ROUND	Rounds a number to a specified number of decimal places	SELECT ROUND(2.36, 1)	2.4
ABS	Returns the absolute value of a number	SELECT ABS (-353)	353
CEILING	Returns the smallest integer value that is >= a number	SELECT CEILING (5.75)	6
FLOOR	Returns the largest integer value that is <= to a number	SELECT FLOOR (5.75)	5

Numeric Functions

Numeric functions are used to **manipulate number type data** in a column.

SQL Code	
Query	**SELECT TOP**(10) **PERCENT** SalesOrderNumber AS InvoiceNumber, OrderDate, SUM(SalesAmount) AS InvoiceSubTotal, ROUND(SUM(SalesAmount),1) AS InvoiceSubTotalRounded, SUM(TaxAmt) AS TaxAmount, FLOOR(SUM(TaxAmt)) AS TaxAmountFloor, SUM(OrderQuantity) AS TotalQuantity, SUM(SalesAmount) + SUM(TaxAmt) AS InvoiceTotal FROM FactInternetSales WHERE SalesTerritoryKey = 6 GROUP BY SalesOrderNumber, OrderDate HAVING SUM(SalesAmount) > 1000 ORDER BY InvoiceSubTotal DESC

Where is the BOOLEAN data type

The **BOOLEAN** data type is represented by the **BIT number type** in SQL.

A BIT number type = **1 for TRUE** and **0 for FALSE**.

A BIT number can also be **NULL**.

SQL Code	
Query	SELECT * FROM DimProduct WHERE FinishedGoodsFlag = 1

Date and Time Data Types

Data Type	Format	Storage	Date range
DATETIME*	'YYYYMMDD hh:mm:ss.nnn' 2020-01-01 11:45:32.547	8 bytes	January 1st ,1753 though December 31st ,9999
SMALLDATETIME*	'YYYYMMDD hh:mm' 2020-01-01 11:45	4 bytes	January 1st,1900 through June 6th,2079
DATE	'YYYY-MM-DD' 2020-01-01	3 bytes	January 1st,0001 through Dec 31,9999
TIME	'hh:mm:ss:nnnnnnn' 11:45:42.4356456	3-5 bytes	Stores times only to an accuracy of 100 nanoseconds
DATETIME2	'YYYYMMDD hh:mm:ss:nnnnnnn' 2020-01-01 11:45:42.4356456	6 to 8 bytes	January 1st,0001 through Dec 31,9999
DATETIMEOFFSET	'YYYYMMDD hh:mm:ss:nnnnnnn' +/-hh:mm' 2020-01-01 11:45:42.4356456 + 08:00	8 to 10 bytes	January 1st,0001 through Dec 31,9999

*These data types are legacy types, meaning they are still supported, but are not as up to date or accurate.

Datepart Values

Parameter	Description
year, yyyy, yy	Returns the year
quarter, qq, q	Returns the quarter
month, mm, m	Returns the year
day, dd, d	Returns the day of the month
dayofyear, dy, y	Returns the day of the year
week, ww, wk	Returns the week
weekday, dw, w	Returns the weekday
hour, hh	Returns the hour

Date and Time Functions

Function	Description	Statement	Result
GETDATE()	Returns the current date and time	SELECT GETDATE()	2020-12-03 04:16:09.247
DATENAME *(datepart, date)*	Returns a character string representing a specified datepart of a specified date.	SELECT DATENAME(month,'20200101')	January
DATEPART *(datepart, date)*	Returns an integer representing the specified datepart of the specified date.	SELECT DATEPART(year,'20200101')	2020
MONTH (*date*)	Returns an integer representing the month part of a specified date.	SELECT MONTH('20200301')	3
YEAR (*date*)	Returns an integer representing the year part of a specified date.	SELECT YEAR('20190301')	2019
DATEDIFF (*datepart, startdate, enddate*)	Returns the number of days or time datepart boundaries, crossed between two specified dates.	SELECT DATEDIFF(day,'20201201','20201230')	29
DATEADD (*datepart, number, date*)	Returns a new datetime value by adding an interval to the specified *datepart* of the specified *date*.	SELECT DATEADD(day,29,'20201201')	2020-12-30

Date and time functions in practice

Date and time functions allow us to **manipulate date and time values in a column.**

SQL Code	
Query	**SELECT** **GETDATE**() **AS** DateTimeStamp DueDate, ShipDate, **DATEDIFF** (day, ShipDate, DueDate) **AS** DaysBetweenShippedAndDueDate, **DATEDIFF** (hour, ShipDate, DueDate) **AS** HoursBetweenShippedAndDueDate **FROM** FactInternetSales

DATEADD

The **DATEADD function** allows us to **add N date parts** to a date.

SQL Code	
Query	**SELECT** **GETDATE**() **AS** DateTimeStamp DueDate, ShipDate, **DATEDIFF** (day, ShipDate, DueDate) **AS** DaysBetweenShippedAndDueDate, **DATEDIFF** (hour, ShipDate, DueDate) **AS** HoursBetweenShippedAndDueDate, **DATEADD** (day, 10, DueDate) **AS** DueDatePlusTenDays, **DATEADD** (day, -10, DueDate) **AS** DueDateLessTenDays **FROM** FactInternetSales

Working with specific dates

To avoid confusion or mis-interpretation of regions, you should use **YYYY-MM-DD** format.

SQL Code	
Query 1	**SELECT** **MONTH**('20201011') **AS** MonthNumerical, **MONTH**('2020-10-11') **AS** MonthNumerical
Query 2	**SET LANGUAGE** British **SELECT** **DATENAME**(month, '02/12/2020') AS MonthName

Server Properties: Collation

Collation tells us how our database deals with **case sensitivity** and what **languages we can use**.

SQL Code	
Query 1	**SELECT CONVERT (varchar(256), SERVERPROPERTY('collation'))** RESULT: SQL_Latin1_General_CP1_CI_AS
Query 2	Right click on the **dimProduct** table and **SELECT TOP 1000**
Query 3	**SELECT** * **FROM** DimProduct **WHERE** Color = 'silver'
Query 4	**SELECT** * **FROM** DimProduct **WHERE** Color = 'Silver'

String or Text Data Types

Text, **String** and **Char** are all used interchangeably when referring to text.

Data Type	Description	Max Size (characters)	Storage (bytes)
Char(n)	REGULAR: Fixed length	8000	1x Defined length
varchar(n)	REGULAR: Variable length	8000	Number of characters + 2
varchar(max)	REGULAR: Variable length	Very Large	Number of characters + 2
text	REGULAR: Variable length	Very Large	Number of characters + 4
nchar(n)	UNICODE: Fixed length, multiple languages	4000	2x Defined length
nvarchar(n)	UNICODE: Variable length, multiple languages	4000	2x number of characters + 2
nvarchar(max)	UNICODE: Variable length, multiple languages	Very Large	2x number of characters + 2
ntext	UNICODE: Variable length, multiple languages	Very Large	2x Defined length
binary(n)	BINARY: Fixed length	8000	Defined length
varbinary(n)	BINARY: Variable length	8000	Defined length + 2
varbinary(max)	BINARY: Variable length	Very Large	Defined length + 2

String Functions

Function	Description	Statement	Result
CONCAT	returns a string resulting from the concatenation, or joining, of two or more string values	SELECT CONCAT(‘Short, ‘sell’’)	Shortsell
LEFT	Returns the left n characters of a string.	SELECT LEFT('Amazon', 3)	Ama
RIGHT	Returns the right characters of a string.	SELECT RIGHT('Amazon', 3)	zon
REPLACE	Replaces all occurrences of a specified string value with another string	SELECT REPLACE('Buy Stock', 'Buy', 'Sell')	Sell Stock
UPPER	Changes the format to UPPERCASE	SELECT UPPER(‘futures”)	FUTURES
LOWER	Changes the format to LOWERCASE	SELECT LOWER(‘FUTURES’)	futures
LEN	Returns the string length, excluding trailing spaces.	SELECT LEN (’stock ’)	5

The CONCAT String Function

The CONCAT function **joins several strings** together into one long string.

SQL Code	
Query 1	**SELECT** EnglishProductName **AS** ProductName, EnglishDescription **AS** ProductDescription, **CONCAT**(EnglishProductName, **'-'**, EnglishDescription) **AS** ProductNameAndDescription **FROM** DimProduct **WHERE** ProductKey=**555**
Query 2	**SELECT** * **FROM** DimProduct

LEN UPPER LOWER REPLACE string functions

These are the **most common string functions**, allowing us to perform manipulations on text.

SQL Code	
Query 1	**SELECT** EnglishProductName **AS** ProductName, EnglishDescription **AS** ProductDescription, **CONCAT**(EnglishProductName, **'-'**, EnglishDescription) **AS** ProductNameAndDescription, **LEN**(EnglishDescription) **AS** DescriptionLength, **UPPER**(EnglishProductName) **AS** UpperProductName, **LOWER**(EnglishProductName) **AS** LowerProductName, **REPLACE**(EnglishProductName, **'Front'**, **'Ultra Durable Front'**) **AS** EnglishProductNameReplaced **FROM** DimProduct **WHERE** ProductKey=**555**
Query 2	**SELECT** * **FROM** DimProduct

Flexible LEFT and RIGHT functions

We can use LEFT and RIGHT to return a **dynamic number of characters** from a string.

SQL Code	
Query	**SELECT** ProductKey, ProductAlternateKey, EnglishProductName **AS** ProductName, EnglishDescription **AS** ProductDescription, **CONCAT**(EnglishProductName, **'-'**, EnglishDescription) **AS** ProductNameAndDescription, **LEN**(EnglishDescription) **AS** DescriptionLength, **UPPER**(EnglishProductName) **AS** UpperProductName, **LOWER**(EnglishProductName) **AS** LowerProductName, **REPLACE**(EnglishProductName, **'Front'**, **'Ultra Durable Front'**) **AS** EnglishProductNameReplaced **LEFT**(ProductAlternateKey,2) **AS** ProductShort, **RIGHT**(ProductAlternateKey,**LEN**(ProductAlternateKey)-3) **AS** ProductSize **FROM** DimProduct **WHERE** ProductKey=**555**

Comparison Operators

Comparison operators are usually used to determine if a certain condition is **TRUE OR FALSE**.

However, comparisons that involve a NULL return **UNKNOWN.**

	= Equal To	> Greater than	< Less than	≠ Not equal to	≥ Greater or equal to	≤ Less or equal to
Syntax	=	>	<	!= OR <>	>=	<=
Example Code	Date = '2020-08-01'	SUM(SaleAmount) > 25000	SUM(TaxAmt)< 9,000	[ProductName] <> 'Mountain-500'	SUM(SaleAmount) >= 3000	SUM(TaxAmt) <= 2000

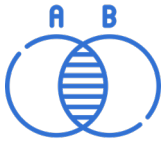
Comparison Operators – Dealing with NULL

NULLs are not captured with regular comparison operators. We have to identify them with **IS NULL**.

SQL Code			
Query 1	**SELECT *** **FROM** DimProduct RESULT 606 ROWS	**SELECT *** **FROM** DimProduct **WHERE** Class <> **'H'** RESULT X ROWS	**SELECT *** **FROM** DimProduct **WHERE** Class <> **'H'** RESULT X ROWS
	In the above examples, the number of rows where **Class = H** and **Class does not = H**, is not equal to the total rows. This is because of the NULLS that exist in the Class column.		
Query 2 (a, b, c)	**SELECT *** **FROM** DimProduct RESULT 606 ROWS	**SELECT *** **FROM** DimProduct **WHERE** Class **IS NULL** RESULT 276 ROWS	**SELECT *** **FROM** DimProduct **WHERE** Class **IS NOT NULL** RESULT 330 ROWS
	In the above examples, the number of rows where **Class IS NULL** and **Class IS NOT NULL**, does infact equal the total.		
Query 3	**SELECT *** **FROM** DimProduct **WHERE** Class <> **'H'** **OR** Class **IS NULL**		

Logical Operators

Logical operators allow us to test multiple conditions at once, or to reverse a condition.

	 **OR**	 **AND**	 **NOT**
	TRUE if either Boolean expression is TRUE	TRUE if both Boolean expressions are TRUE	Used to reverse a condition
Example	**Class <> 'H' OR Class IS NULL**	**Size = 'XL' AND Color = 'Black'**	**NOT(Color = 'Black')**

Logical Operators – Common Errors

We can use **parenthesis** to **explicitly decide** on the **order of logical operators.**

SQL Code	
Query	**SELECT** EnglishProductName, EnglishDescription, Color, [Status], Class **FROM** DimProduct **-- Incorrect use of logical operators** **-- WHERE** Class <> **'H' OR** Class **IS NULL AND** [Status] **IS NOT NULL** **Correct use of parenthesis to make the order of** **WHERE (**Class <> **'H' OR** Class **IS NULL) AND** [Status] **IS NOT NULL**

Advanced Logical Operators - Simplifying Code

	IN	BETWEEN
	TRUE if the value is equal to any item in a list	TRUE if the value is within a range (inclusive).
Example	**Color IN ('Red', 'Blue','White')**	**SalesAmount BETWEEN 500 AND 1000**
Same as...	Color = 'Red' OR Color = 'Blue' OR Color = 'White'	SalesAmount >= 500 AND SalesAmount <= 1000

IN and BETWEEN in practice

xx

SQL Code	
Query 1	**SELECT** EnglishProductName, EnglishDescription, Color, [Status], Class, SafetyStockLevel **FROM** DimProduct **WHERE** (SafetyStockLevel **BETWEEN** 500 **AND** 1000) **AND** [Status] **IS NOT NULL --BETWEEN IS INCLUSIVE OF BOTH END** **-- WHERE** (SafetyStockLevel >= 500 **AND** SafetyStockLevel <= 1000) **AND** [Status] **IS NOT NULL**
Query 2	**SELECT** EnglishProductName, EnglishDescription, Color, [Status], Class, SafetyStockLevel **FROM** DimProduct **WHERE** Color IN ('Black', 'Silver', 'White' , 'Yellow') **WHERE** Color = 'Black' **OR** Color = 'Silver' **OR Color =**'White' **AND** 'Color'='Yellow'

Add these two queries to **Manipulating Data – Logical Operators**

Advanced Logical Operators - Partial Matches

LIKE

TRUE if the value matches a specified pattern

Example	EnglishProductName LIKE '%BRAKE%'

WILDCARDS

% represents **any number of characters**
_ represents **one character**

LIKE in practice

The **LIKE operator** can be used to **find partial matches.**

The _ represents a single wildcard character.

The % represents any number of wildcard characters.

SQL Code	
Query	**SELECT** FirstName, EmailAddress **FROM** DimCustomer **WHERE** FirstName **LIKE '_R%'**

Using IIF statements to create a conditional column

We can use **IIF statements** in the same way as we do in Excel.

SQL Code	
Query	**SELECT** FirstName, LastName, YearlyIncome, EmailAddress, **IIF**(YearlyIncome > 50000, **'Above Average'** , **'Below Average'**) **AS** IncomeCategory **FROM** DimCustomer

Using a CASE statement for multiple conditions

A **CASE statement** helps us define **multiple conditional scenarios** in one column.

SQL Code

Query

```
SELECT

    FirstName,
    LastName,
    YearlyIncome,
    EmailAddress,
    IIF(YearlyIncome > 50000, 'Above Average' , 'Below Average' ) AS IncomeCategory,
    CASE
        WHEN NumberChildrenAtHome = 0 THEN '0'
        WHEN NumberChildrenAtHome = 1 THEN '1'
        WHEN NumberChildrenAtHome BETWEEN 2 AND 4 THEN '2-4'
        WHEN NumberChildrenAtHome >=5 THEN '5+'
        ELSE 'UNKN'
    END AS NumberOfChildrenCategory,
    NumberOfChildrenCategory AS ActualChildren

FROM DimCustomer
```

Basic SQL Formatting

Indenting SQL code helps to **keep sections clear.**

SQL Code	
Query	**SELECT** FirstName, LastName, YearlyIncome, EmailAddress, **IIF**(YearlyIncome > 50000, **'Above Average'** , **'Below Average'**) **AS** IncomeCategory, **CASE** **WHEN** NumberChildrenAtHome = **0 THEN '0'** **WHEN** NumberChildrenAtHome = **1 THEN '1'** **WHEN** NumberChildrenAtHome **BETWEEN 2 AND 4 THEN '2-4'** **WHEN** NumberChildrenAtHome >=**5 THEN '5+'** **ELSE 'UNKN'** **END AS** NumberOfChildrenCategory, NumberOfChildrenCategory **AS** ActualChildren **FROM** DimCustomer

Using IF in a WHERE statement

We can **replicate a conditional IIF column** for use in a **WHERE statement**.

SQL Code	
Query	**SELECT** FirstName, LastName, YearlyIncome, EmailAddress, **IIF**(YearlyIncome > 50000, **'Above Average'** , **'Below Average'**) **AS** IncomeCategory, **CASE** **WHEN** NumberChildrenAtHome = **0 THEN '0'** **WHEN** NumberChildrenAtHome = **1 THEN '1'** **WHEN** NumberChildrenAtHome **BETWEEN 2 AND 4 THEN '2-4'** **WHEN** NumberChildrenAtHome >=**5 THEN '5+'** **ELSE 'UNKN'** **END AS** NumberOfChildrenCategory, NumberOfChildrenCategory **AS** ActualChildren **FROM** DimCustomer **WHERE IIF**(YearlyIncome > 50000, **'Above Average'** , **'Below Average'**) = **'Above Average'**

Replacing NULL using IIF, ISNULL and COALESCE

We can **test for and replace NULL values** in multiple ways.

SQL Code

Query

```
SELECT
    FirstName,
    IIF(MiddleName IS NULL, 'UNKN' , MiddleName) AS MiddleName,
    ISNULL(MiddleName, 'UNKN' ) AS MiddleName2,
    COALESCE(MiddleName, 'UNKN' ) AS MiddleName3,
    LastName,
    YearlyIncome,
    EmailAddress,
    IIF(YearlyIncome > 50000, 'Above Average' , 'Below Average' ) AS IncomeCategory
    CASE
        WHEN NumberChildrenAtHome = 0 THEN '0'
        WHEN NumberChildrenAtHome = 1 THEN '1'
        WHEN NumberChildrenAtHome BETWEEN 2 AND 4 THEN '2-4'
        WHEN NumberChildrenAtHome >=5 THEN '5+'
        ELSE 'UNKN'
    END AS NumberOfChildrenCategory,
    NumberOfChildrenCategory AS ActualChildren

FROM DimCustomer

WHERE IIF(YearlyIncome > 50000, 'Above Average' , 'Below Average' ) =
```

Using CAST to change the data type

The **CAST operator** allows us to **change the data type.**

SQL Code	
Query	**SELECT** SalesAmount, **CAST**(SalesAmount **AS INT**) **AS** SalesAmountCast, OrderDate, **CAST**(OrderDate **AS DATE**) **AS** OrderDateCast **FROM** FactInternetSales

Practical examples using CAST

In this example, the **CAST operator** is used to change two **INTEGERS into DECIMALS.**

SQL Code	
Query	**/*SELECT** SalesAmount, **CAST**(SalesAmount **AS INT**) **AS** SalesAmountCast, OrderDate, **CAST**(OrderDate **AS DATE**) **AS** OrderDateCast **FROM** FactInternetSales*/ **SELECT** EnglishProductName, ReOrderPoint, SafetyStockLevel **CAST**(ReOrderPoint **AS DECIMAL**(**8**,**4**)) / **CAST**(SafetyStockLevel **AS DECIMAL**(8,4)) **AS** PctOfTotalSafetyStock **FROM** DimProduct **WHERE** [Status] = **'Current'**

NATIONAL for Unicode best practice

The **National 'N'** should be used before all strings interacting with a **UNICODE column type.**

SQL Code	
Query	**SELECT** EnglishProductName, EnglishDescription, Color, [Status], Class, SafetyStockLevel **FROM** DimProduct **WHERE** Color IN (N'Black', N'Silver', N'White' , N'Yellow') **WHERE** Color = N'Black' **OR** Color = N'Silver' **OR Color =** N'White' **AND** Color= N'Yellow'

Precedence Among Operators

Earlier, we mentioned that the AND operator is executed before the OR operator.

Here is a **full list of precedence** for operators in SQL.

1. ( ) (Parentheses)
2. * (Multiplication), / (Division), % (Modulo)
3. + (Positive), – (Negative), + (Addition), + (Concatenation), – (Subtraction)
4. =, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
5. NOT
6. AND
7. BETWEEN, IN, LIKE, OR
8. = (Assignment)

SQL Fundamentals: Student Exercise 2a

Sales territory 1 need a summary of their sales for the lead up period to Christmas.

1. Write a query against the FactInternet Sales table that returns orders placed in December for the Sales Territory 1

1. The query should include SalesOrderNumber, SalesOrderLineNumber, SalesAmount and TaxAmount.

	InvoiceNumber	InvoiceLineNumber	SalesAmount	TaxAmount
1	SO43699	1	3399.99	271.9992
2	SO46406	1	3578.27	286.2616
3	SO46431	1	3578.27	286.2616
4	SO46445	1	3578.27	286.2616
5	SO46446	1	3578.27	286.2616
6	SO46452	1	3374.99	269.9992
7	SO46466	1	3578.27	286.2616
..	...	...	...	...
919	SO74251	3	34.99	2.7992

SQL Fundamentals: Student Exercise 2b

Marketing need a list of homeowner customers, along with the number of cars owned.

1. Write a query against the dimCustomer table that returns all customers that are homeowners and have more than 1 car.

1. The query should include full customer names, number of cars owned, and email.

1. The numbers of cars owned should categorize customers into groups:
    - 2-3
    - 4+

	CustomerName	NumberOfCarsOwned	Email
1	Elizabeth Johnson	4+	elizabeth5@adventure-works.com
2	Marco Mehta	2-3	marco14@adventure-works.com
3	Rob Verhoff	2-3	rob4@adventure-works.com
4	Curtis Lu	4+	curtis9@adventure-works.com
5	Lauren Walker	2-3	lauren41@adventure-works.com
6	Ian Jenkins	2-3	ian47@adventure-works.com
7	Shannon Wang	2-3	shannon1@adventure-works.com
..	...	...	...
6126	Colin Xu	2-3	colin28@adventure-works.com

SQL Fundamentals

SQL Theory

SQL Theory - Section Objectives

Tasks

01. Understand the difference between **OLTP** and **Data Warehouse**

02. Learn the **most important terminology** relating to SQL

Skills

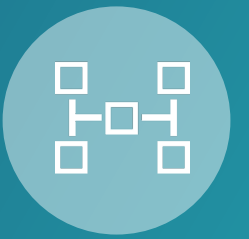
Server vs Instance vs Database

Cloud vs On Premise

OLTP vs Data Warehouse

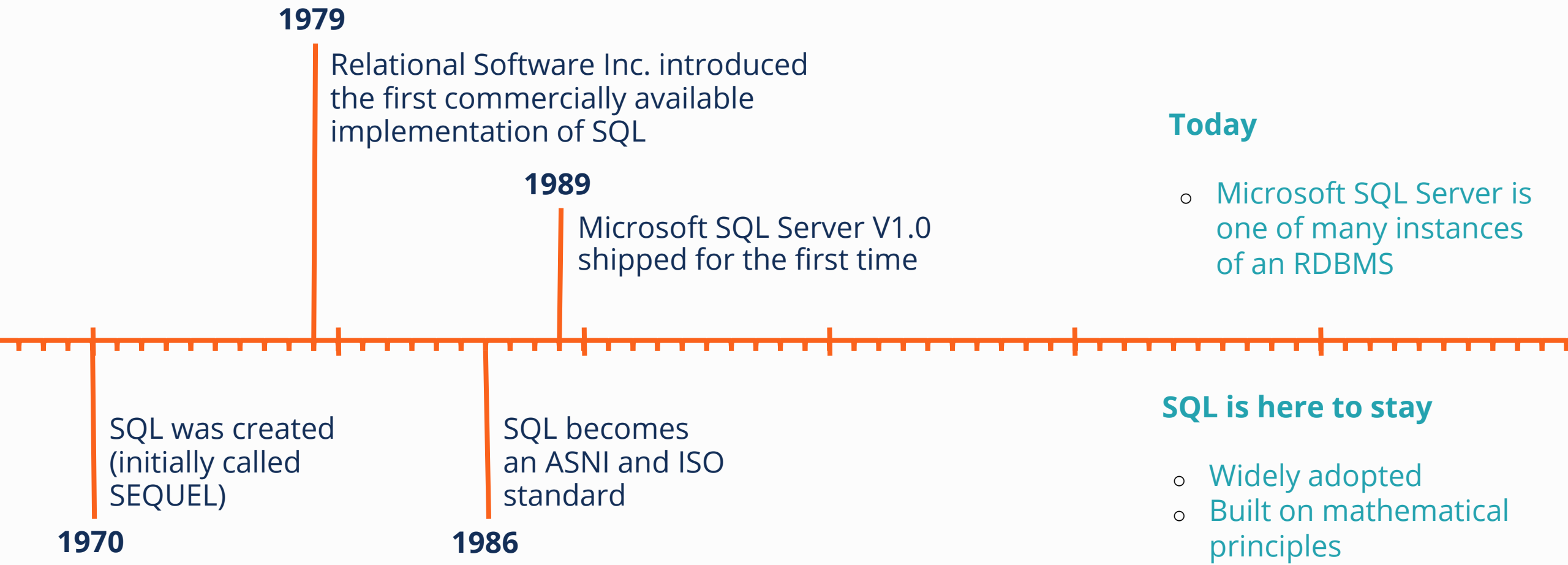
Database Normalization

Fact vs Dimension Tables



Star, Snowflake and Hybrid Schema

A Timeline of SQL

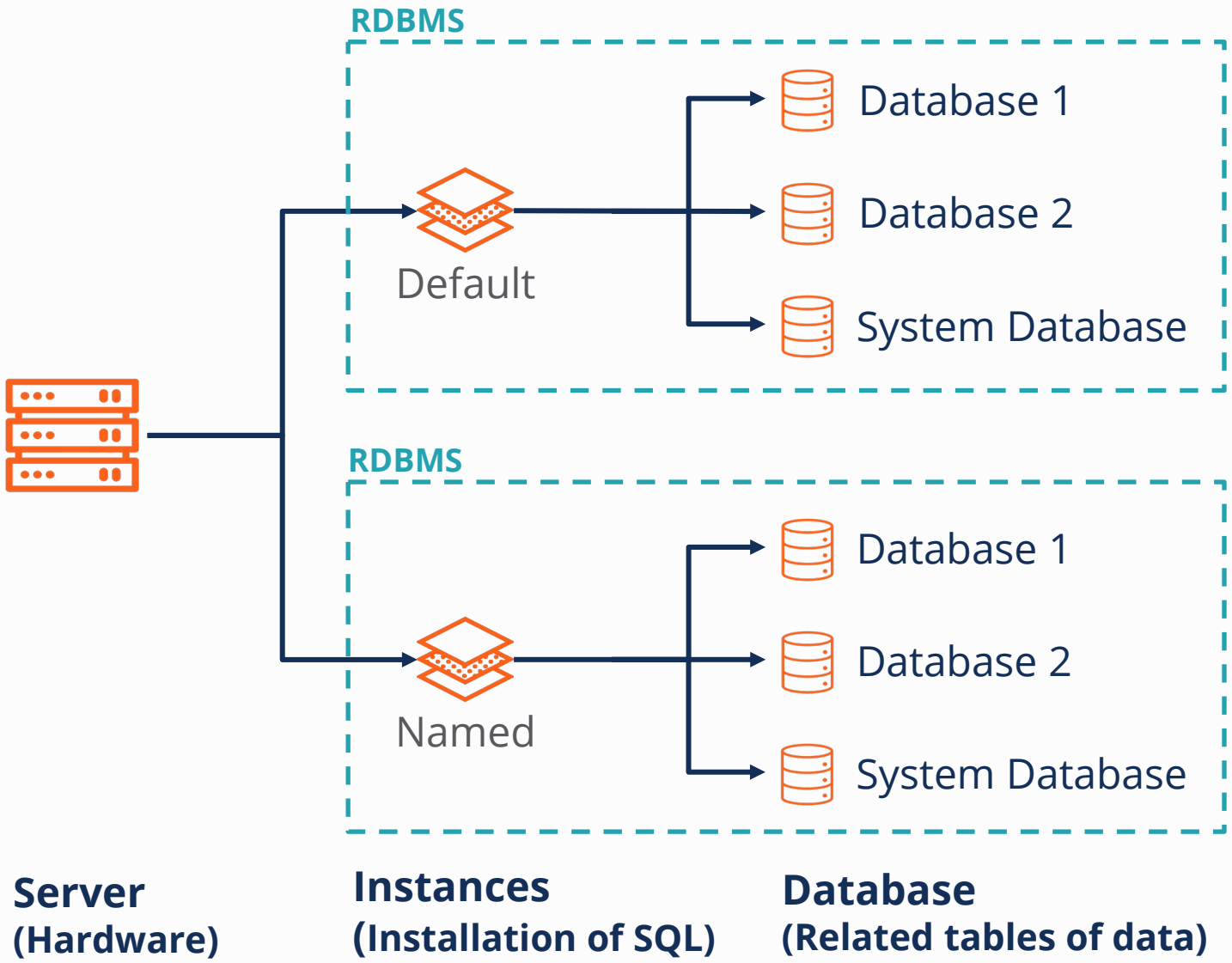


SQL, Servers, Instances & Databases

SQL =
Structured
Query
Language

RDBMS =
Relational
DataBase
Management
System

Popular RDBMS
Microsoft SQL Server
Oracle
MY SQL



RDBMS Installation Options

Box (On Prem)

Business responsible for server and software

Advantages
- Security
- Flexibility

Disadvantages
- Upfront cost
- Requires expertise

IAAS - Infrastructure As A Service (hardware only)

PAAS - Platform As A Service (hardware & software)

Host looks after server hardware and/or software

Advantages
- No upfront cost
- Easy to scale

Disadvantages
- Targeted by hackers
- Cost escalation

Data Security

Personal Data

PII
(Personally
Identifiable Information)

Name, Email, Address, SSN
IP Address, Phone Number
Date of Birth, Place of Work,
Company Phone Number

Privacy Laws

Data Security

First Name	Last Name	Email	User ID	Country	Visit Count
~~John~~	~~Xi~~	~~xigi777@yahuu.com~~	~~000021~~	~~US~~	~~1~~
Nathan	Wilson	Nathan45.Wilson@bmail.com	000022	CAN	5
Alex	Lee	Alex@youtax.com	000023	MEX	2
Sophia	Smith	Fifi-foever@me.com	000024	CAN	2
Mia	MacDonald	Mia23@bmail.com	000025	US	15
				Total Visits	**???**

Types of Database Systems

- Designed for efficient data entry
- Reduce data redundancy
- Fast at reading / writing data

- Efficient for analysis and reporting
- Data Warehouse Organization
- Data Mart Specific team

Data Normalization in OLTP Systems

Normalization is the process of organizing data to minimize redundancy.

First Name	Last Name	Salary	Hourly Salary	Department	Department Manager	Location
John	Xi	52000	30	IT	Thomas	Vancouver
Nathan	Wilson	45000	25.96	Finance	Felix	Toronto
Alex	Lee	60000	34.62	Finance	Felix	Toronto
Sophia	Smith	47000	27.12	IT	Thomas	Vancouver
Mia	MacDonald	50000	28.85	Finance	Felix	Toronto

Data Normalization in OLTP Systems

First Name	Last Name	Salary	Hourly Salary	Department	Department Manager	Location
John	Xi	52000	30	IT	Thomas	Vancouver
Nathan	Wilson	45000	25.96	Finance	Felix	Toronto
Alex	Lee	60000	34.62	Finance	Felix	Toronto
Sophia	Smith	47000	27.12	IT	Thomas	Vancouver
Mia	MacDonald	50000	28.85	Finance	Felix	Toronto

Id	First Name	Last Name	Salary	DepID
1	John	Xi	52000	1
2	Nathan	Wilson	45000	2
3	Alex	Lee	60000	2
4	Sophia	Smith	47000	1
5	Mia	MacDonald	50000	2

DepId	Department	Department Manager	Location
1	IT	Thomas	~~Vancouver~~ San Francisco
2	Finance	Felix	Toronto

Denormalization in DW Systems

Denormalization is used to simplify **multiple related tables into one**.

Category

CategoryID	Category
1	Cell Phone
2	Tablet

Brand

BrandID	Brand Name
1	Apple
2	Samsung

Product Name

Product ID	Cateogy ID	BrandID	Product Name
1	1	1	Iphone X
2	1	1	Iphone XI
3	2	1	Ipad 10.2
4	1	2	Galaxy Note20

DimProduct

Product ID	Category	Brand	Product Name
1	Cell Phone	Apple	Iphone X
2	Cell Phone	Apple	Iphone XI
3	Tablet	Apple	Ipad 10.2
4	Cell Phone	Samsung	Galaxy Note20

Fact Tables

Fact tables contain measurements about a particular business event.

OrderID	CutomerID	OrderDate	Revenue	Quantity	Discount	Total Cost
544122	1	2020-5-12	250.12	1	0	190
545428	2	2020-06-11	5211.45	24	25	2500
546584	3	2020-06-15	2000.24	8	50	940
547514	4	2020-08-11	5201.2	18	0	2800

Fact tables contain **IDs (or Keys)**.

Dimension Tables

Dimension tables provide descriptive information about the attributes in the fact table.

CustomerID	First Name	Last Name	Education Level	Occupation
1	John	Brooks	Bachelors	Professional
2	Lilly	Xi	Graduate Degree	Management
3	Taylor	Hess	Partial College	Skilled Manual
4	Tina	Navarro	High School	Manual

IDs are used to connect dimension tables to fact tables.

Relationships & Keys

Date Table (Dimension Table)

Date (PK)	Day of Week
01/12/2020	Tuesday
02/12/2020	Wednesday
03/12/2020	Thursday

Customer Table (Dimension Table)

CustomerID (PK)	Customer Name
1	John Brooks
2	Lilly Xi
3	Taylor Hess

Sales Transaction Table (Fact Table)

Order ID (PK)	Date (FK)	Revenue	CustomerID (FK)	ProductID (FK)
152156	01/12/2020	261.24	1	3
138688	02/12/2020	22.99	2	1
108966	02/12/2020	350.99	1	2
115812	03/12/2020	350.99	3	2

Columns used to join tables are known as **keys.**

A **Primary Key (PK)** identifies a row in the current table.

A **Foreign Key (FK)** identifies a row in another table.

Product Table (Dimension Table)

ProductID (PK)	ProductName	SubCategory	Category
1	Water Bottle (Blue)	Water Bottles	Accessories
2	Winter Jacket	Jackets	Clothing
3	Rain Jacket	Jackets	Clothing

Star Schema

Date Table (Dimension Table)

Date (PK)	Day of Week
01/12/2020	Tuesday
02/12/2020	Wednesday
03/12/2020	Thursday

Customer Table (Dimension Table)

CustomerID (PK)	Customer Name
1	John Brooks
2	Lilly Xi
3	Taylor Hess

Sales Transaction Table (Fact Table)

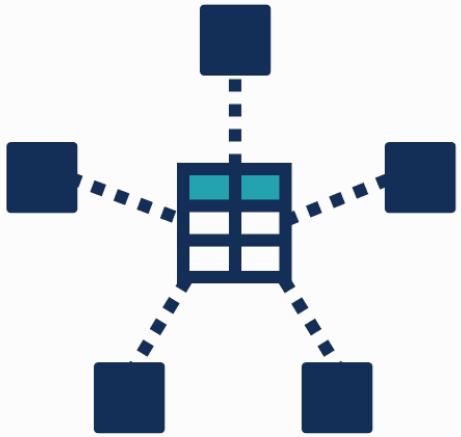
Order ID (PK)	Date (FK)	Revenue	CustomerID (FK)	ProductID (FK)
152156	01/12/2020	261.24	1	3
138688	02/12/2020	22.99	2	1
108966	02/12/2020	350.99	1	2
115812	03/12/2020	350.99	3	2

Product Table (Dimension Table)

ProductID (PK)	ProductName	SubCategory	Category
1	Water Bottle (Blue)	Water Bottles	Accessories
2	Winter Jacket	Jackets	Clothing
3	Rain Jacket	Jackets	Clothing

Star Schema

A simple **star schema** has one central fact table, and a number of single dimension tables.



Snowflake/Hybrid Schema

Date Table (Dimension Table)

Date (PK)	Day of Week
01/12/2020	Tuesday
02/12/2020	Wednesday
03/12/2020	Thursday

Customer Table (Dimension Table)

Customer ID (PK)	Customer Name
1	John Brooks
2	Lilly Xi
3	Taylor Hess

Product Table (Dimension Table)

Product ID (PK)	ProductName	SubCategoryID(FK)
1	Water Bottle (Blue)	1
2	Winter Jacket	2
3	Rain Jacket	2

Order ID (PK)	Date (FK)	Revenue	CustomerID (FK)	ProductID (FK)
152156	01/12/2020	261.24	1	3
138688	02/12/2020	22.99	2	1
108966	02/12/2020	350.99	1	2
115812	03/12/2020	350.99	3	2

Sales Transaction Table (Fact Table)

Sub-Category Table (Dimension Table)

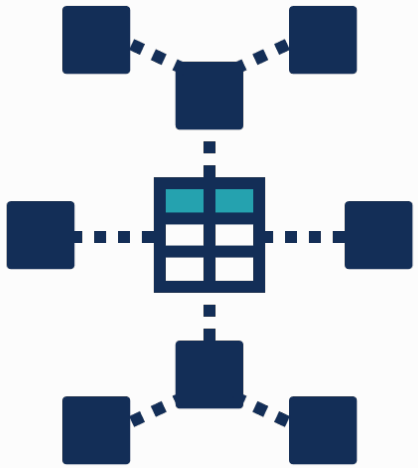
SubCategory ID (PK)	SubCategory	CategoryID(FK)
1	Water Bottles	1
2	Jackets	2

Category Table (Dimension Table)

CategoryID (PK)	Category
1	Accessories
2	Clothing

Star Schema

A **snowflack schema** has one central fact table and includes dimension tables which are further normalized with additional dimension tables.



SQL Fundamentals

Working with Multiple Tables

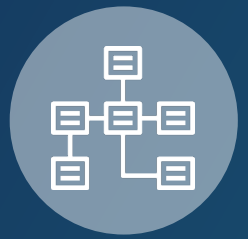
Working with Multiple Tables - Section Objectives

Tasks

01. Combine data from **multiple tables** into a single query

02. Understand how to deal with **MANY to MANY** relationships

Skills



Learn how to read an ERD

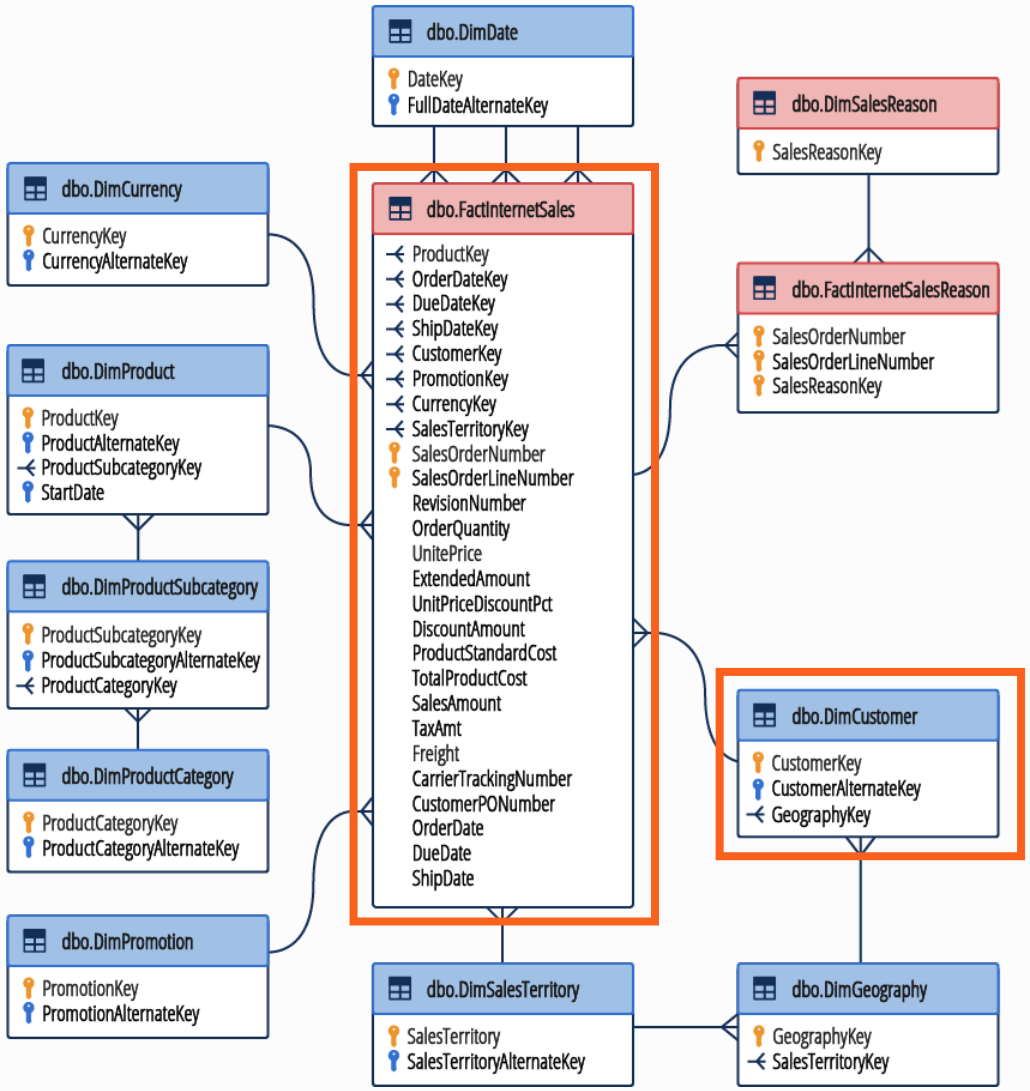
JOINS and UNIONS

Bridge Tables

Create Views

Create dynamic results using subqueries

Relationships and ER diagrams



Scenario

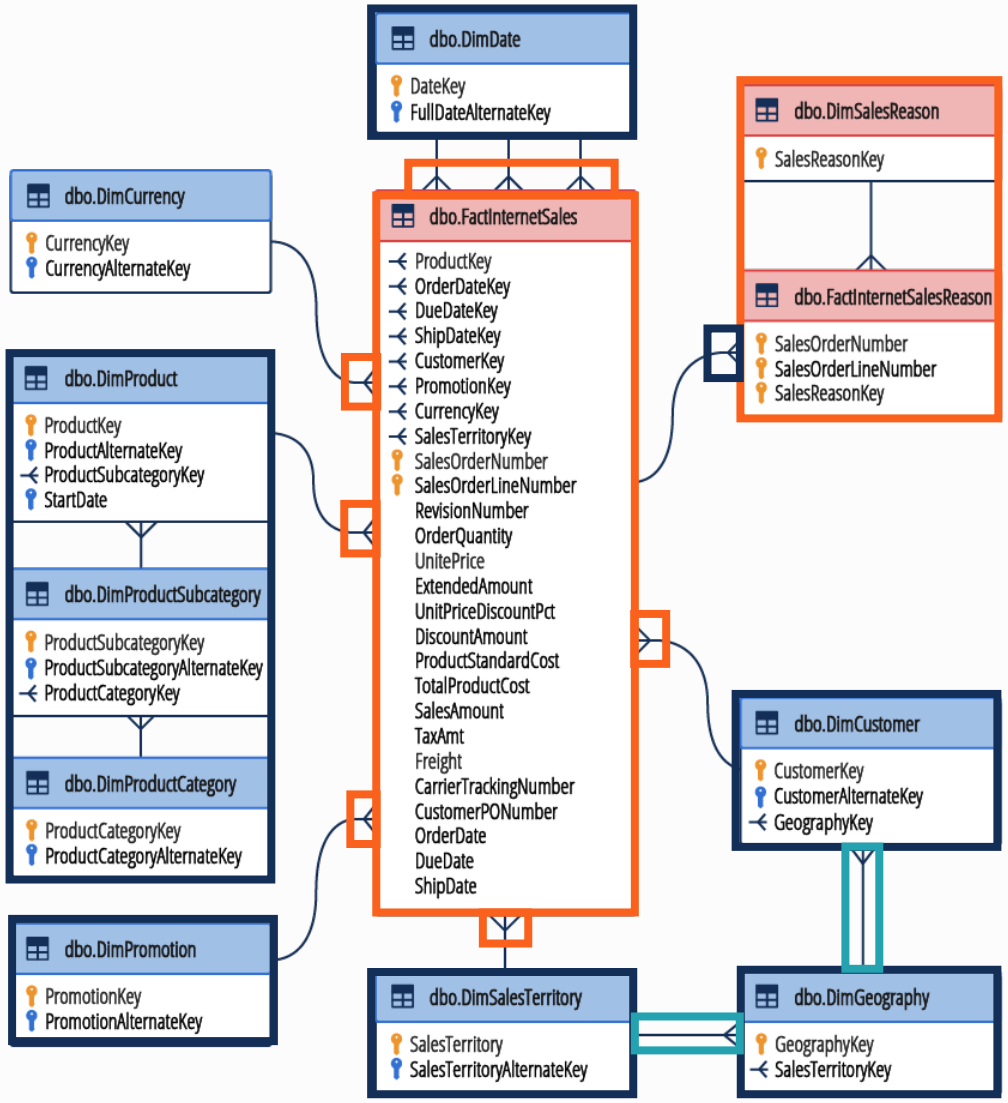
Marketing would like a list of top customers by sales, along with email addresses.

Primary keys uniquely identify rows in a table.

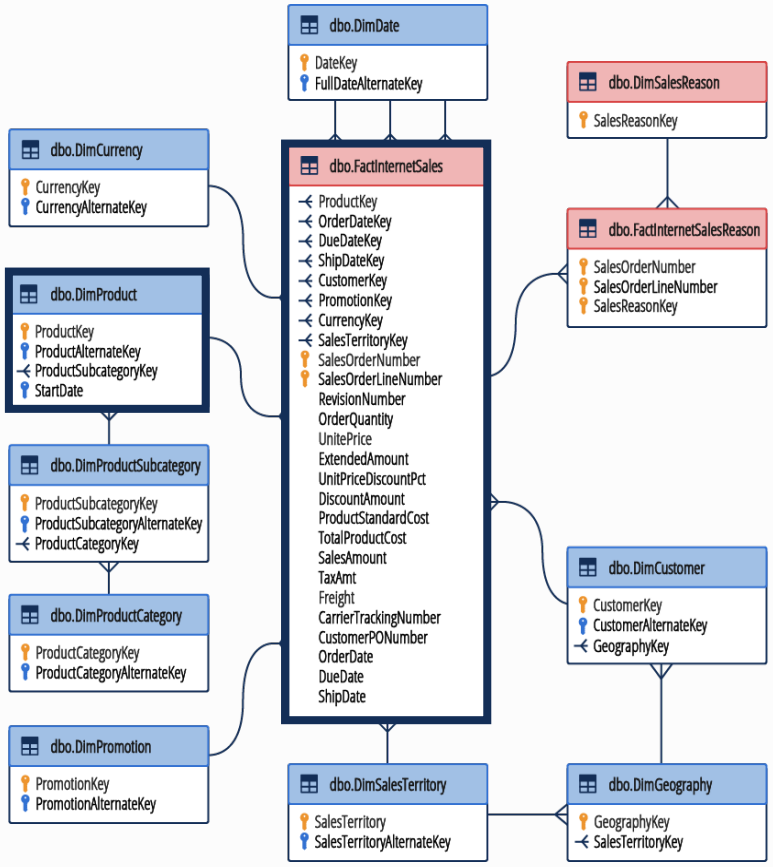
Multiple primary keys mean that the combination of both columns is used to uniquely identify a row.

Indicates a foreign key. This tells us we can join these tables together.

Internet Sales Schema



Purpose of DW Relationships



DW relationships help users **understand the connections** between tables.

DW relationships also help **maintain data integrity**.

1) Relationship cardinality must be respected:

Many to One
Each key can only appear once in one table, but may appear many times in the other.
E.g. Product tbl to Sales tbl

One to One
A single occurrence of the key in each table.
E.g. Tax Payers tbl to Social Security tbl

Many to Many
Potentially many occurrences of each key in each table.
E.g. Customer tbl to Addresses tbl

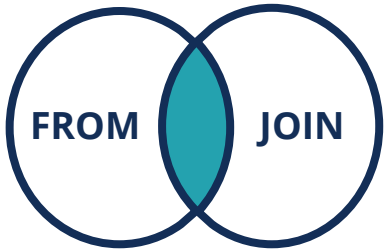
2) Primary keys must exist in dimensions before we add the same foreign key in fact tables.

eg. **ProductID 922 must exist** in DimProduct PK

before

ProductID 922 can be used in FactInternetSales FK

Inner Join



Inner Join

Returns **only the rows where the linking value(s) match** in both tables.

FROM

OrderID	Revenue	CustomerID
152156	261.24	CG-12520
138688	14.62	DV-13045
108966	957.57	SO-20335
115812	1706.18	BH-11710

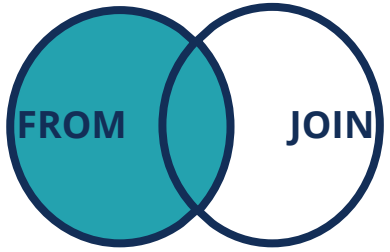
JOIN

CustomerID	Customer Name
CG-12520	Claire Gute
DV-13045	Darrin Van Huff
SW-14531	Sara Wei

OrderID	Revenue	CustomerID	Customer Name
152156	261.24	CG-12520	Claire Gute
138688	14.62	DV-13045	Darrin Van Huff

In SQL, the INNER JOIN can be written as **INNER JOIN** or **JOIN**

Left Join



Left Join

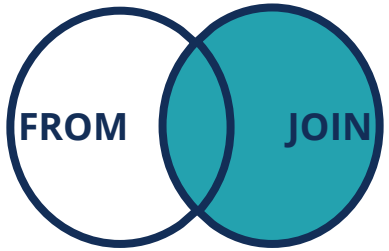
Returns **all records from the left (primary) table,** but only the matched rows from the JOIN table.

OrderID	Revenue	CustomerID
152156	261.24	CG-12520
138688	14.62	DV-13045
108966	957.57	SO-20335
115812	1706.18	BH-11710

CustomerID	Customer Name
CG-12520	Claire Gute
DV-13045	Darrin Van Huff
SW-14531	Sara Wei

OrderID	Revenue	CustomerID	Customer Name
152156	261.24	CG-12520	Claire Gute
138688	14.62	DV-13045	Darrin Van Huff
108966	957.57	SO-20335	NULL
115812	1706.18	BH-11710	NULL

Right Join



Right Join

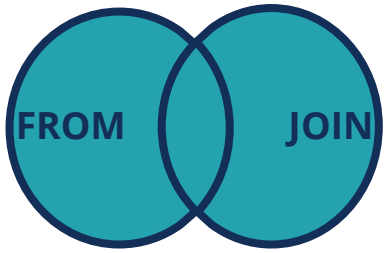
Returns **all records from the right (JOIN) table,** but only the matched rows from the primary table table.

OrderID	Revenue	CustomerID
152156	261.24	CG-12520
138688	14.62	DV-13045
108966	957.57	SO-20335
115812	1706.18	BH-11710

CustomerID	Customer Name
CG-12520	Claire Gute
DV-13045	Darrin Van Huff
SW-14531	Sara Wei

OrderID	Revenue	CustomerID	Customer Name
152156	261.24	CG-12520	Claire Gute
138688	14.62	DV-13045	Darrin Van Huff
NULL	NULL	SW-14531	Sara Wei

Full Outer Join



Full Outer Join

Returns all records from BOTH tables, matching rows where possible.

OrderID	Revenue	CustomerID
152156	261.24	CG-12520
138688	14.62	DV-13045
108966	957.57	SO-20335
115812	1706.18	BH-11710

CustomerID	Customer Name
CG-12520	Claire Gute
DV-13045	Darrin Van Huff
SW-14531	Sara Wei

OrderID	Revenue	CustomerID	Customer Name
152156	261.24	CG-12520	Claire Gute
138688	14.62	DV-13045	Darrin Van Huff
108966	957.57	SO-20335	NULL
115812	1706.18	BH-11710	NULL
NULL	NULL	SW-14531	Sara Wei

A basic INNER JOIN using sales and customers

The INNER JOIN here matches the Customer Key **(FK) from FactInternetSales** to the Customer Key **(PK) from DimCustomer.**

SQL Code	
Query	**SELECT *** **FROM** FactInternetSales **AS** f **INNER JOIN** DimCustomer **AS** c **ON** f.CustomerKey = c.CustomerKey

Returning only the TOP 100 customers

Here we **GROUP BY** Customer Names and Email Addresses, to ensure we have unique customers.

We then used **ORDER BY – DESC** to rank the unique customers, and TOP to select only 100.

We could also have grouped by **Customer Key.**

SQL Code	
Query	**SELECT TOP(100)** **CONCAT**(c.FirstName,**' '**, c.LastName) **AS** CustomerName, c.EmailAddress **AS** EmailAddress, **SUM**(f.SalesAmount) **AS** AmountSpent **FROM** FactInternetSales **AS** f **INNER JOIN** DimCustomer **AS** c **ON** f.CustomerKey = c.CustomerKey **GROUP BY** dc.FirstName, dc.LastName, dc.EmailAddress **ORDER BY** AmountSpent **DESC**

INNER JOIN the currency table

We can use INNER JOIN again, to **JOIN a second table** to our fact table.

SQL Code	
Query	**SELECT TOP(100)** **CONCAT**(dc.FirstName,**' '**, dc.LastName) **AS** CustomerName, dc.EmailAddress **AS** EmailAddress, **SUM**(fs.SalesAmount) **AS** AmountSpent, dcy.CurrencyName **AS** Currency **FROM** FactInternetSales **AS** fs **INNER JOIN** DimCustomer **AS** dc **ON** fs.CustomerKey = dc.CustomerKey **INNER JOIN** DimCurrency **AS** dcy **ON** fs.CurrencyKey = dcy.CurrencyKey **GROUP BY** dc.FirstName, dc.LastName, dc.EmailAddress, dcy.CurrencyName **HAVING** dcy.CurrencyName = **N'US Dollar'** **ORDER BY** AmountSpent **DESC**

HAVING or WHERE

In this example, it is **more efficient to filter** the original data using **WHERE**.

SQL Code	
Query	**SELECT TOP(100)** **CONCAT**(dc.FirstName,**' '**, dc.LastName) **AS** CustomerName, dc.EmailAddress **AS** EmailAddress, **SUM**(fs.SalesAmount) **AS** AmountSpent, --dcy.CurrencyName **AS** Currency **FROM** FactInternetSales **AS** fs **INNER JOIN** DimCustomer **AS** dc **ON** fs.CustomerKey = dc.CustomerKey **INNER JOIN** DimCurrency **AS** dcy **ON** fs.CurrencyKey = dcy.CurrencyKey **WHERE** dcy.CurrencyName = **N'US Dollar'** **GROUP BY** dc.FirstName, dc.LastName, dc.EmailAddress --, dcy.CurrencyName **--HAVING** dcy.CurrencyName = **N'US Dollar'** **ORDER BY** AmountSpent **DESC**

RIGHT JOIN to retrieve full product catalogue

The RIGHT JOIN **keeps the whole table right of the join (DimProduct)**

SQL Code	
Query	**SELECT** dp.EnglishProductName AS ProductName, dp.Color **AS** ProductColor, **ISNULL**(dp.Size,'UNKN') **AS** ProductSize, **ISNULL**(**SUM**(fs.SalesAmount),0) **AS** SalesAmount **FROM** FactInternetSales **AS** fs **RIGHT JOIN** DimProduct **AS** dp **ON** fs.ProductKey = dp. ProductKey **WHERE** dp.Status = **N'Current'** **GROUP BY** dp.EnglishProductName, dp.Color, dp.Size **ORDER BY** SalesAmount **DESC**

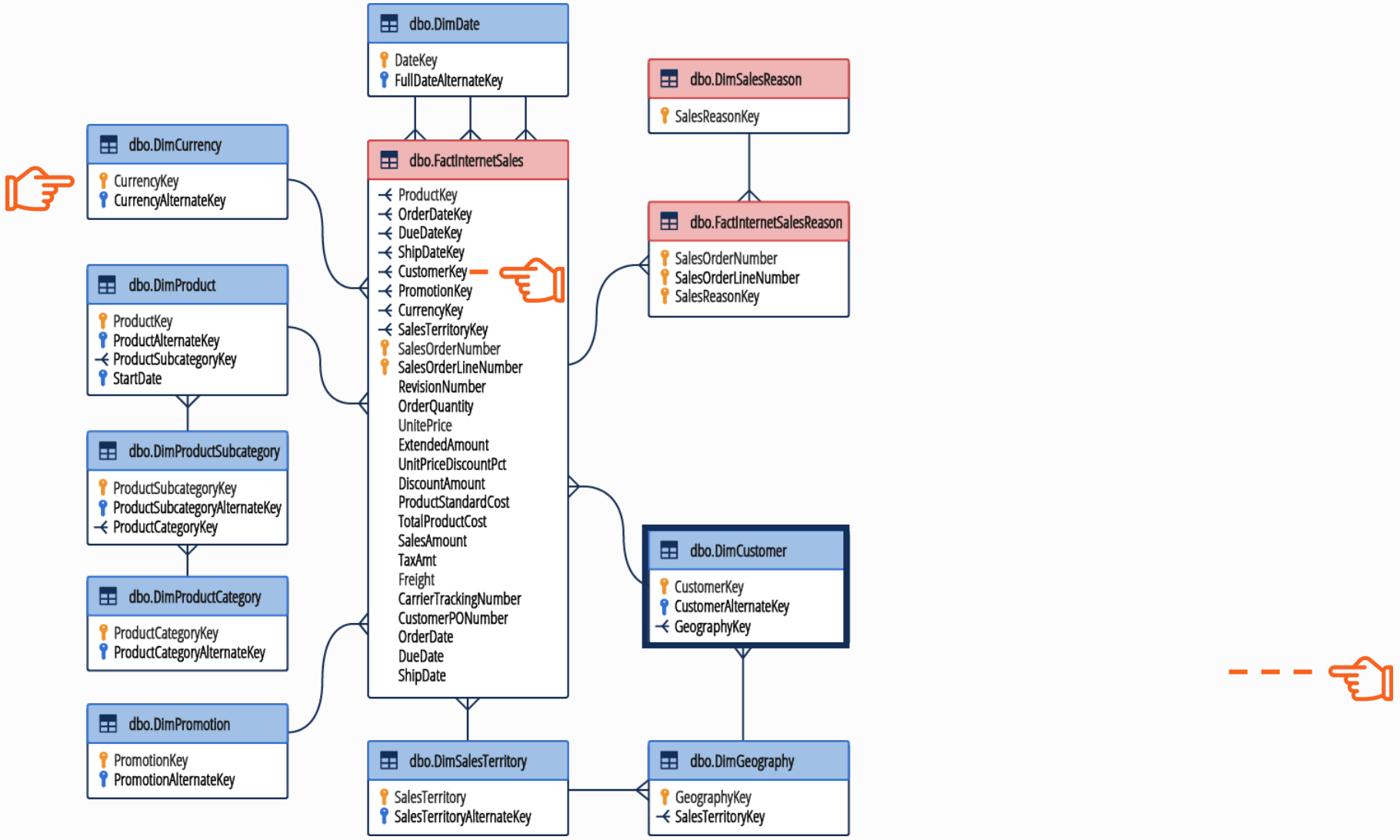
LEFT JOIN vs LEFT JOIN

LEFT JOINS can be **easier to visualize.**

Here we return **all rows from our main table**, and **find matches where available from our JOIN table**.

SQL Code	
Query	**SELECT** dp.EnglishProductName AS ProductName, dp.Color **AS** ProductColor, **ISNULL**(dp.Size,'UNKN') **AS** ProductSize, **ISNULL**(**SUM**(fs.SalesAmount),0) **AS** SalesAmount **FROM** DimProduct **AS** dp **LEFT JOIN** FactInternetSales **AS** fs **ON** dp.ProductKey = fs. ProductKey **WHERE** dp.Status = **N'Current'** **GROUP BY** dp.EnglishProductName, dp.Color, dp.Size **ORDER BY** SalesAmount **DESC**

Internet Sales Schema (Screenshot for video)



Bridge Tables

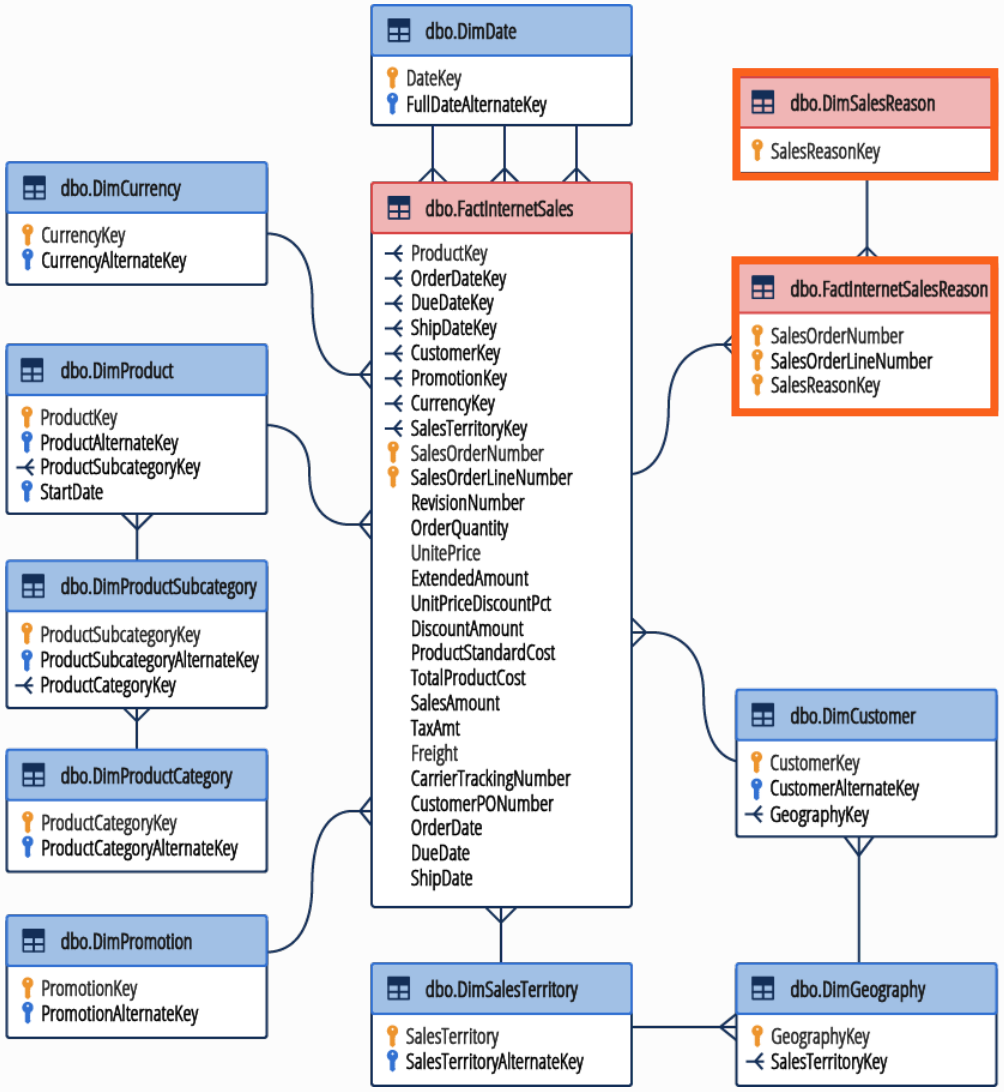
Sales Transaction Table (Fact Table)

SalesOrder Number (PK)	SalesOrder Line (PK)	Revenue
SO51178	1	2319
SO51178	2	9.99
SO51187	1	539.99
SO51187	2	21.5

Sales Reason Table (dimension)

SalesOrderNumber	SalesOrder Line	Reason Id	Sales Reason
SO51178	1	1	Price
SO51178	1	2	On Promotion
SO51178	2	1	Price
SO51178	2	2	On Promotion
SO51187	1	1	Price
SO51187	1	2	On Promotion
SO51187	2	1	Price
SO51187	2	2	On Promotion

Bridge Tables



Bridge Tables

FactInternetSales		
SalesOrder Number	SalesOrder Line	Revenue
SO51178	1	2319
SO51178	2	9.99
SO51187	1	539.99
SO51187	2	21.5

dimSalesReason	
Reason ID	Sales Reason
1	Price
2	Promotion

Bridge Tables

FactInternetSales

SalesOrder Number	SalesOrder Line	Revenue
SO51178	1	2319
SO51178	2	9.99
SO51187	1	539.99
SO51187	2	21.5

Bridge Table

FactInternetSalesReason

SalesOrder Number	SalesOrder Line	Reason ID
SO51178	1	1
SO51178	1	2
SO51178	2	1
SO51178	2	2
SO51187	1	1
SO51187	1	2
SO51187	2	1
SO51187	2	2

dimSalesReason

Reason ID	Sales Reason
1	Price
2	Promotion

Creating JOINS across BRIDGE tables.

The first INNER JOIN creates a link **between our fact table and the bridge table**.

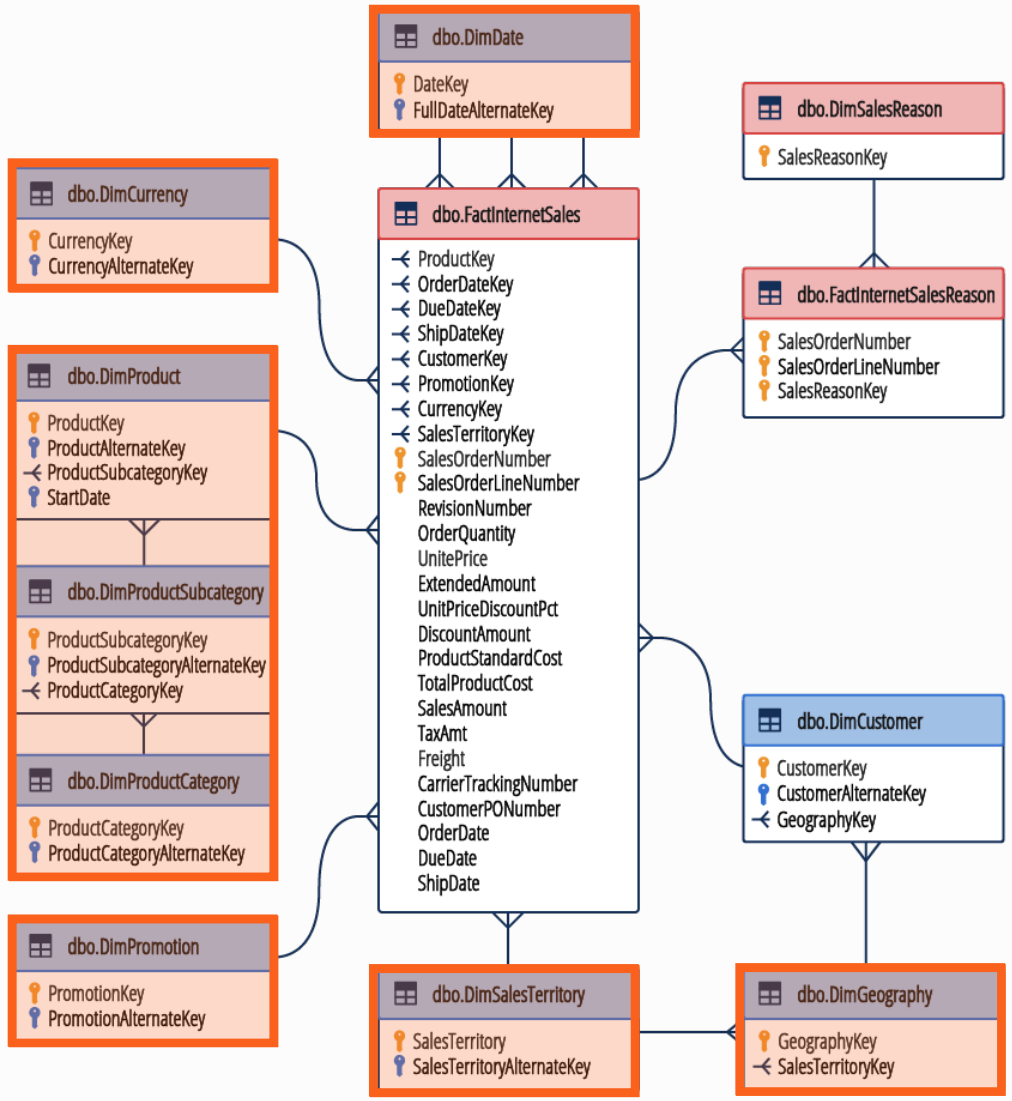
The second INNER JOIN creates a link **between the bridge table and the dimension**.

SQL Code	
Query	**SELECT** --fs.SalesOrderNumber **AS** InvoiceNumber, --fs.SalesOrderLineNumber **AS** InvoiceLineNumber, dsr.SalesReasonReasonType **AS** SalesReason, **SUM**(fs.SalesAmount) **AS** SalesAmount **FROM** FactInternetSales **AS** fs **INNER JOIN** FactInternetSalesReason **AS** fsr **ON** fs.SalesOrderNumber = fsr.SalesOrderNumber AND fs.SalesOrderLineNumber = fsr.SalesOrderLineNumber **INNER JOIN** DimSalesReason dsr **ON** fsr.SalesReasonKey = dsr.SalesReasonKey --WHERE fs.SalesOrderNumber = N'SO51178' **GROUP BY** dsr.SalesReasonReasonType

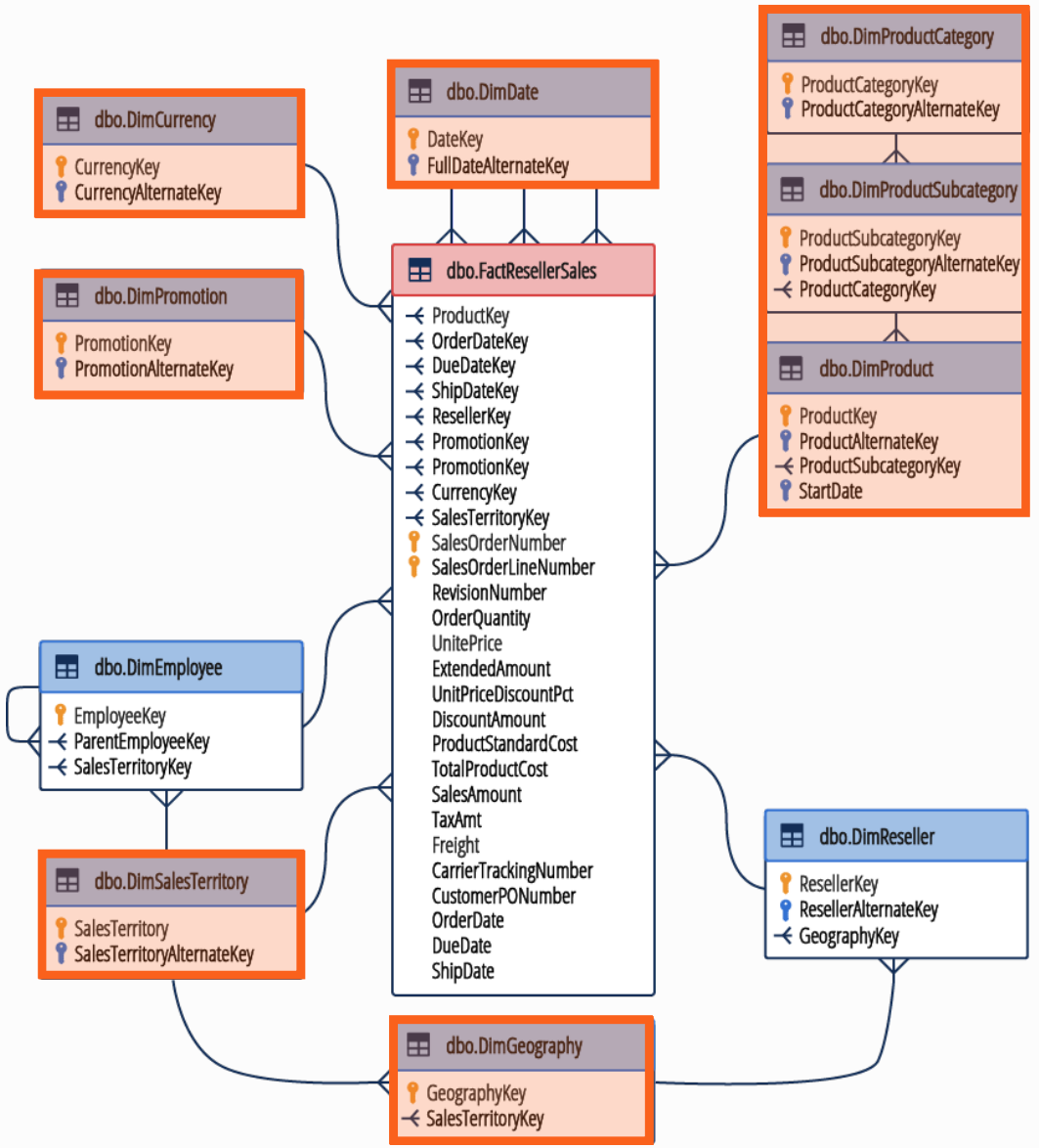
The **MANY to MANY** nature of this connection means we should **use caution when presenting results**.

Adventure Works Schemas

FactInternetSales records of sales from **the website**.



FactResellerSales records of sales from **re-sellers**.



UNION

A UNION combines **2 or more** tables by adding the rows from one table to another.

OrderID	Revenue	CustomerID
1	261.24	1
2	14.62	2
3	957.57	3

OrderID	Revenue	CustomerID
3	957.57	3
4	1706.18	4

=

OrderID	Revenue	CustomerID
1	261.24	1
2	14.62	2
3	957.57	3
4	1706.18	4

A UNION **discards duplicate values.**

UNION ALL

A union combines **2 or more** tables by adding the rows from one table to another.

OrderID	Revenue	CustomerID
1	261.24	1
2	14.62	2
3	957.57	3

OrderID	Revenue	CustomerID
3	957.57	3
4	1706.18	4

=

OrderID	Revenue	CustomerID
1	261.24	1
2	14.62	2
3	957.57	3
3	957.57	3
4	1706.18	4

UNION ALL **keeps** duplicate values.

Creating a UNION between Internet and Reseller Sales

A UNION **combines the rows** from multiple tables (queries).

The UNION should combine two queries that have **the same columns**.

The **ORDER BY** can only be used **after the UNION** of the two queries.

SQL Code	
Query	**-- For illustrative purposes only. For full code refer to Completed Queries folder.** **SELECT** COL1 COL2 COL3 **FROM** FactInternetSales **UNION** **SELECT** COL1 COL2 COL3 **FROM** FactResellerSales **ORDER BY** COL3

SELF JOIN

The SELF JOIN links a table to a copy of itself.

SELF JOINS are particularly common when dealing with hierarchical data, such as manager employee relationships.

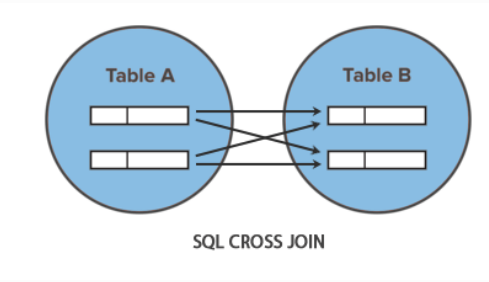
EmployeeID	EmployeeName	ManagerID
1	Jason	2
2	Flo	NULL
3	Rahul	2

EmployeeID	EmployeeName	ManagerID
1	Jason	2
2	Flo	NULL
3	Rahul	2

EmployeeID	EmployeeName	ManagerID	ManagerName
1	Jason	2	Flo
2	Flo	NULL	NULL
3	Rahul	2	Flo

CROSS JOIN

Need Something like this



The Cross Join combines every row in the first table with every row from the second table

CardType
MasterCard
Visa

Color
Blue
Yellow
Red

Cross Join

CardType	Color
MasterCard	Blue
Visa	Blue
MasterCard	Yellow
Visa	Yellow
MasterCard	Red
Visa	Red

Creating a VIEW

Views allow us to **save queries in the database.**

SQL Code	
Query	**-- For illustrative purposes only. For full code refer to Completed Queries folder.** **CREATE VIEW** vwOrdersALL **AS** **--Description of view here to assist future analysts.** **SELECT** COL1 COL2 COL3 **FROM** FactResellerSales **ORDER BY** COL3 **GO - GO is used as best practice at the end of the View creation code.**

Querying a view

We can query a pre-defined view, which means **we don't need to re-do all our hard work**, including JOINS.

SQL Code	
Query	**SELECT *** **FROM** vwOrdersALL

Creating dynamic results using SUBQUERIES

A subquery (inner query) returns data that we can re-use in our main query (outer query).

SQL Code	
Query	**SELECT *** **FROM** vwOrdersALL **SELECT** InvoiceNumber, InvoiceLineNumber, OrderDate, SalesAmount, ProductName, ProductSubcategory **FROM** vwOrdersALL **WHERE** OrderDate = **(SELECT MAX(OrderDate) FROM vwOrdersALL)**

SQL Fundamentals: Student Exercise 4a

Summarize the Internet Sales by Subcategory and return the top 5 subcategories.

1. Write a query that returns the top 5 best-selling subcategories by SalesAmount.

1. We're only interested in sales from our website (internet sales).

1. Finally, the data should only include sales where the country is United States and the currency is US Dollar

1. You are **avoid using the view** we created.

	SubCategory	SalesAmount
1	Road Bikes	4289925.9
2	Mountain Bikes	3417457.74
3	Touring Bikes	1292475.9
4	Tires and Tubes	88762.86
5	Helmets	76663.09

SQL Fundamentals: Student Exercise 4b

It's performance review time. HR Europe need to see sales by sales representative, and by currency.

1. Write a query that will return a list of all current Sales Representatives or Sales Managers in the European territory.

1. For each person, HR need to see sales amounts grouped by currency.

2. Please include the following fields: Full employee name, Employee Title, Currency Name and total sales amount

3. The query should be sorted by Employee Name and Sales Amount.

	EmployeeName	EmployeeTitle	Currency	TotalSalesAmount
1	Amy Alberts	European Sales Manager	United Kingdom Pound	441081.6364
1	Amy Alberts	European Sales Manager	EURO	200960.57
2	Amy Alberts	European Sales Manager	US Dollar	90036.2386
3	José Saraiva	Sales Representative	United Kingdom Pound	3837927.19
4	Rachel Valdez	Sales Representative	EURO	1790640.23
5	Ranjit Varkey Chudukatil	Sales Representative	US Dollar	4026954.02
6	Ranjit Varkey Chudukatil	Sales Representative	EURO	482934.909

SQL Fundamentals

SQL For Reporting

SQL for Reporting - Section Objectives

Tasks

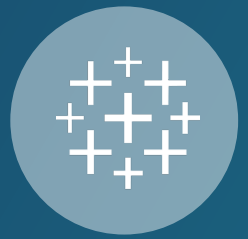
01. Connect to our database and views **from popular BI tools**

02. Create three summary reports that require **more advanced functions**

Skills

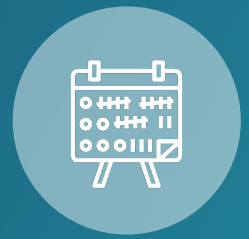
Query SQL in Power BI

Query SQL in Excel



Query SQL in Tableau

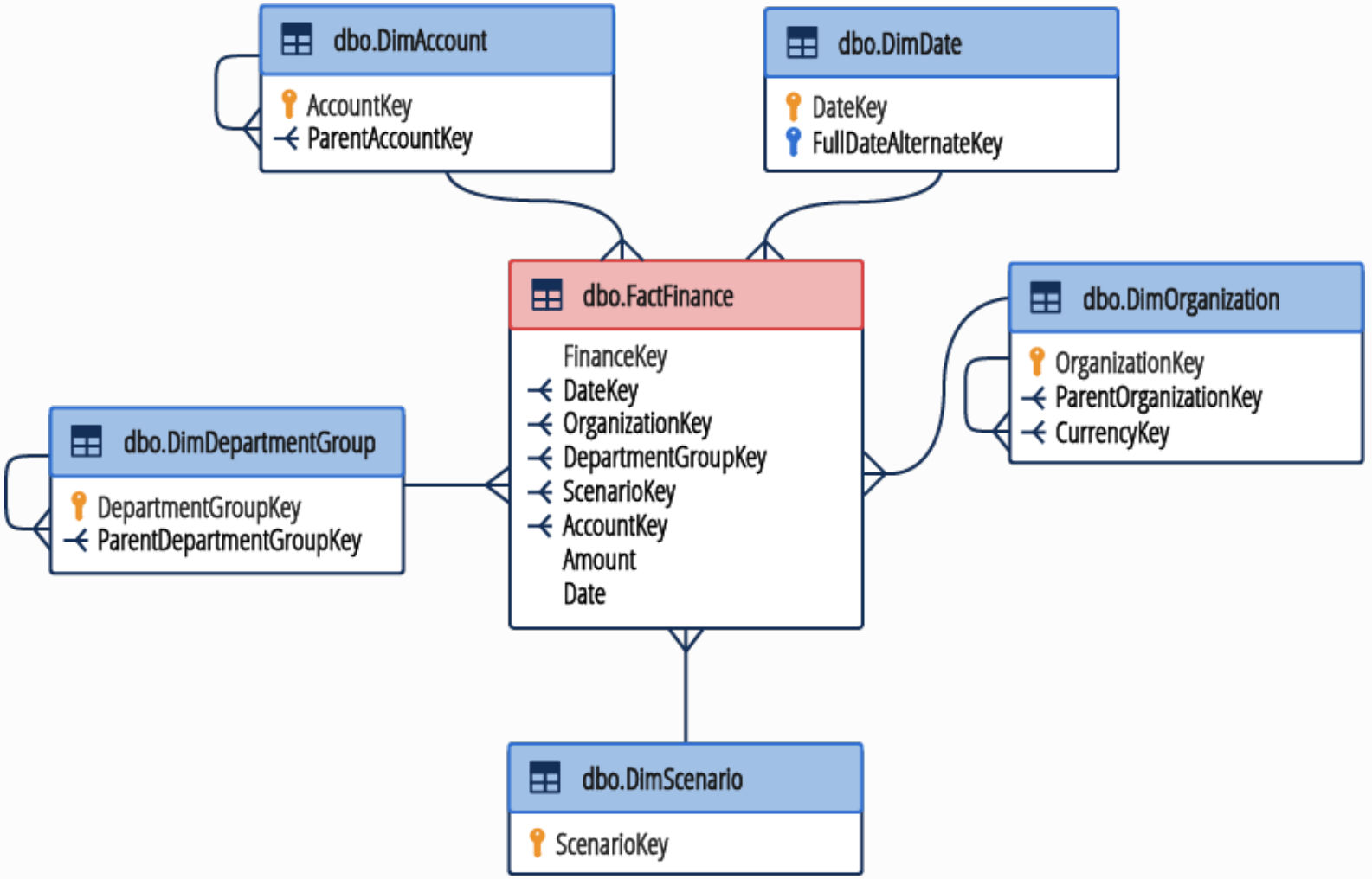
Create a summary with CUBE Function



Create a summary with ROLLUP Function

Calculate % of Total

Finance Schema



Using CUBE to return subtotals and totals

When using GROUP BY across multiple columns, we lose the ability to see subtotals.

CUBE allows us to **see the subtotals and totals** of each row combination.

SQL Code	
Query	**SELECT** Region, ProductSubCategory, **SUM**(SalesAmount) **AS** TotalSales **FROM** vwOrdersALL **WHERE YEAR**(OrderDate) = 2013 **AND** Currency = **N'US Dollar'** **GROUP BY CUBE**(Region, ProductCategory)

Using ROLLUP to return subtotals and totals

ROLLUP gives a similar outcome to CUBE, but **performs better on hierarchical data**.

SQL Code	
Query	**SELECT** Region, ProductSubCategory, **SUM**(SalesAmount) **AS** TotalSales **FROM** vwOrdersALL **WHERE YEAR**(OrderDate) = 2013 **AND** Currency = **N'US Dollar'** **GROUP BY CUBE**(ProductCategory, ProductSubCategory)

Common scenario: Percent of total

We can use a **subquery to calculate the grand total**, and use it in our SELECT statement to **calc % of total**.

SQL Code	
Query	**SELECT** Source **AS** Reseller, **SUM**(SalesAmount) **AS** Sales, **SUM**(SalesAmount) / (SELECT SUM(SalesAmount) FROM vwOrdersALL WHERE Country = N'United States' AND Source <> N'Web') AS PctOfTotal **FROM** vwOrdersALL **WHERE** Country = **N'United States' AND** Source <> **N'Web'** **GROUP BY** Source **ORDER BY** Sales **DESC**

SQL Fundamentals: Student Exercise 5a

Create a summary of expenditure accounts.

1. Write a query that will return the sum of actuals from the FactFinance table.

1. Filter the data to meet the following conditions:
   - January 2011 only
   - Southwest division only
   - Expenditure accounts only

2. For each row, list the Organization, Account Type and Account Name.

1. Group the rows by Organization, Account Type and Account.

	Organization	AccountType	Account	Amt
1	Southwest Division	Expenditures	Standard Cost of Sales	122573
2	Southwest Division	Expenditures	Salaries	39240
3	Southwest Division	Expenditures	Taxes	36299
4	Southwest Division	Expenditures	Salaries	28280
...				
28	Southwest Division	Expenditures	Equipment	43

SQL Fundamentals: Student Exercise 5b

Create a summary of expenditure account totals, and then calculate a PCT of total.

1. Write a query that will return Account Description, and amounts corresponding to ACTUALS.

1. Filter the results to meet the following conditions:
   - ACTUALS only
   - Canadian Division only
   - Calendar year 2013 only
   - Expenditure accounts only

2. Create a subquery to help calculate the total sales that meet the same conditions.

HINT: Sometimes it's easier to create the subquery separately and then add it to your main query.

	AccountDescription	Amount	PctofTotal
1	Standard Cost of Sales	2672904.1	0.3572165
2	Salaries	2163556.3	0.28914543
3	Taxes	665875.37	0.08898998
4	Variances	441498.92	0.0590035
5	Commissions	320161.79	0.04278757
...			
29	Other Travel Related	3053.93	0.00040814