Let's begin with a simple SQL query.

In SQL, the **SELECT** clause is used to list which data columns you would like from a specific data table; the table is referenced after the **FROM** clause.

```
select *

--Account_Type, Description


from StateDW.dbo.Account_Type
```

Account_Type	Description
01	Assets
02	Liabilities
03	Equity
20	Pre Encumbrance
21	Encumbrance
22	Expenditure
24	Expense
31	Revenue
90	Assets
99	Misc
42	Budget
51	Estimated Revenue

Columns Filters

Page Size: 20 **1** to **12** of **12** Page **1** of **1**

Note how the SQL statement above brought in two data fields *Account_Type* and *Description*, and this, from the *Account_Type* table in the state datawarehouse.

What if we were only looking to return a subset of rows from that query?

We can actually filter out data using a **WHERE** clause.

```
select *

from Account_Type

where Account_Type='22'
```

Account_Type	Description
22	Expenditure

Columns Filters

Page Size: 20 1 to 1 of 1 Page 1 of 1

Note how the **WHERE** clause must go last. SQL queries have an **order**. The order generally goes, **SELECT...FROM...WHERE**.

We can get a little more precise in the **WHERE** clause concerning the details of the filtering we would like.

This happens mostly using the logic of **AND**/**OR**.

```
select Account_Type, Description

from Account_Type

where Account_Type='22' or Account_Type='02'
```

Account_Type	Description
02	Liabilities
22	Expenditure

Columns Filters

Page Size: 20 1 to 2 of 2 Page 1 of 1

The **OR** indicates that for all the data in the query, any row of data that satisfies **at least one of the conditions** will survive the filter.

The **AND** indicates that for all the data in the query, any row of data that satisfies **both of the conditions** will survive the filter.

Let's use **AND** below.

```
select Account_Type, Description

from Account_Type

where Account_Type='22' and Account_Type='02'
```

Account_Type	Description

Columns

No Rows To Show

Filters

Page Size: 20 **0** to **0** of **0** Page **0** of **0**

No data was returned because no row of data has an account type that is simultaneously both 22 **AND** 02.

There are two more special ways to filter data in the **WHERE** clause: the **LIKE** and the **IN** statements.

```
select Account_Type, Description

from Account_Type

where Description like '%Exp%'
```

Account_Type	Description
22	Expenditure
24	Expense

Columns

Filters

Page Size: 20 **1** to **2** of **2** Page **1** of **1**

Note how the **LIKE** operator enables use to use text matching! We can use the **%** sign as a wild card pull back any data values in the *Description* column that start with *Exp* regardless of what comes after.

Let's try the **IN** statement.

```
select Account_Type, Description

from Account_Type

where Account_Type IN ('22','31')
```

Account_Type	Description
22	Expenditure
31	Revenue

Columns Filters

Note how the **IN** statement allows us to make a list of values that survive the filter. This is like a long list of **OR** statements. If any data values are found in the **IN** statement, that data row survives the filter.

Now let's combine the **IN**, **LIKE**, and lastly, **AND** all in one **WHERE** clause.

```
select Account_Type, Description

from Account_Type

where Account_Type IN ('22','31') AND Description like 'Exp%'
```

Account_Type	Description
22	Expenditure

Columns Filters

Page Size: 20 1 to 1 of 1 Page 1 of 1

Here there were two rounds of filters, and the only data row that survived was the one whose *Account_Type* was **22 OR 31** (either one), **AND** whose *Description* was **LIKE** 'Exp...'

Now let's try to use these skills on the *Accounting_Journal*

```
select aj.Document_Code,aj.Doc_Department,aj.Document_ID,

aj.Debit_Credit,aj.Amount, aj.Object, aj.Account_Type


from Accounting_Journal as aj

where aj.Document_Code='GAX' and aj.Document_ID='74000000001' and aj.Doc_Department='810'
```

Document_Code	Doc_Department	Document_ID	Debit_Credit	Amount
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95

Columns Filters

Page Size: 20 **1** to **2** of **2** Page **1** of **1**

Look at this nice accounting above - do you notice how there is a debit and a credit for one transaction, each with amounts that cancel the other, each of a different account type.

Note how I can give something a nickname with the **AS** statement, I nicknamed the Accounting_Journal as **aj**.

Once you give something a nickname (an alias), you can reference the nickname to note what things you would like to pull from that source. In the query above, we referenced **aj** to pull back some of its data columns.

Nicknames are most useful when you are pulling back many different tables in one SQL query - when using a **JOIN**.

```
select aj.Document_Code,aj.Doc_Department,aj.Document_ID,

aj.Debit_Credit,aj.Amount, aj.Object, aj.Account_Type,


acc.Description


from Accounting_Journal aj
inner join Account_Type as acc

on aj.Account_Type=acc.Account_Type

where aj.Document_Code='GAX' and aj.Document_ID='74000000001' and aj.Doc_Department='810'
```

Document_Code	Doc_Department	Document_ID	Debit_Credit	Amount
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95

Columns

Filters

Page Size: 20 1 to 2 of 2 Page 1 of 1

The **JOIN** clause goes after the **FROM**, right before the **WHERE** clause. When you use a **JOIN** you have to list the new data table (here, it was *Account_Type* nicknamed *acc*) and you have to list how that new table should relate to the original table in the **FROM** clause.

In the **ON** statement, we note how the *Accounting_Journal* and the *Acount_Type* tables sometimes talk about the same thing, specifically, they both talk about the data field *Account_Type*. Using this join, we are saying "When the account type of one table is the same account type of the other table, join the tables' information together". Thus, the *Description* that was part of the *Account_Type* table is now also appearing beside the data pulled from the *Accounting_Journal* table.

Let's try to add one more table into this mix.

Let's pull in more information about the *Object* by joining the *Object* table onto the *Accounting_Journal*'s object code.

```
select aj.Document_Code,aj.Doc_Department,aj.Document_ID,

aj.Debit_Credit,aj.Amount, aj.Object,aj.Account_Type,


acc.Description,

o.Object_Name, o.Fiscal_Year


from Accounting_Journal aj
inner join Account_Type as acc
on aj.Account_Type=acc.Account_Type


inner join Object as o
on aj.Object=o.Object


where aj.Document_Code='GAX' and aj.Document_ID='74000000001' and aj.Doc_Department='810'
```

Document_Code	Doc_Department	Document_ID	Debit_Credit	Amount
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95

Columns

Filters

DBCode

GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95
GAX	810	74000000001	D	34.95

Page Size: 20 | 1 to 20 of 44 | Page 1 of 3

What happened?

We started with only 2 rows of data and after this particular join we have 44 rows!

Here is something to remember about joins...

```
select * from Object as o where o.Object='6177'
```

Fiscal_Year	Object	Object_Name	Object_Class	Object_Class_Name
2011	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2021	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2022	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2012	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2009	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2007	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2015	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2023	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2016	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2013	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2010	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2008	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2018	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2025	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp
2019	6177	6177 Building & Grounds Security	DDC	DDC Operating Supp

Columns Filters

Page Size: 20 | 1 to 20 of 22 | Page 1 of 2

In the *Object* table, there are 22 instances of object code 6177, one per fiscal year.

This means that for the 2 data rows we were expecting in our initial query (the debit and credit of the transaction), the debit row got matched to each of these 22 instances of object code 6177 in the *Object* table; the credit row did the same.

That is why there were 44 (2*22) rows of data! This is called the **join explosion problem**.

We were joining these tables by matching only on object code, without accounting for fiscal year. Let's account for fiscal year in the join so that our **JOIN** only matches when both the *Object* **AND** *Fiscal_Year* match between both tables.

```
select aj.Document_Code,aj.Doc_Department,aj.Document_ID,

aj.Debit_Credit,aj.Amount, aj.Object,aj.Account_Type,


acc.Description,

o.Object_Name


from Accounting_Journal aj

inner join Account_Type as acc on aj.Account_Type=acc.Account_Type
inner join Object as o on aj.Object=o.Object AND aj.Fiscal_Year=o.Fiscal_Year


where aj.Document_Code='GAX' and aj.Document_ID='74000000001' and aj.Doc_Department='810'
```

Document_Code	Doc_Department	Document_ID	Debit_Credit	Amount
GAX	810	74000000001	D	34.95
GAX	810	74000000001	C	-34.95

Columns Filters

That looks right!

We have been using an **INNER JOIN** but there are other types of joins.

Before we look at the others, let's remind ourselves of how the *Account_Type* table looks.

```
select * from Account_Type as acc
```

Account_Type	Description
01	Assets
02	Liabilities
03	Equity
20	Pre Encumbrance
21	Encumbrance
22	Expenditure
24	Expense
31	Revenue
90	Assets
99	Misc
42	Budget
51	Estimated Revenue

Columns Filters

Page Size: 20 **1** to **12** of **12** Page **1** of **1**

Notice how in our **SELECT** statement, we used an asterisk (*) instead of listing the specific data fields. This indicates that we want to return all data columns available.

The second most common join in SQL is the **LEFT JOIN**.

Let's try it below.

```
select *


from Account_Type as acc left join Accounting_Journal as aj


on acc.Account_Type=aj.Account_Type AND aj.Document_Code='GAX' and aj.Document_ID='74000000001' and
aj.Doc_Department='810'
```

Account_Type	Description	Account_Type	Accounting_Line_Number	Activity
01	Assets			
02	Liabilities	02	1	
03	Equity			
20	Pre Encumbrance			
21	Encumbrance			
22	Expenditure	22	1	
24	Expense			
31	Revenue			
90	Assets			
99	Misc			
42	Budget			
51	Estimated Revenue			

Columns Filters

Page Size: 20 **1** to **12** of **12** Page **1** of **1**

Notice how the **LEFT JOIN** brought back **all the data rows and values** from the LEFT table (in the **FROM** clause - *Account_Type*), but only the **data values** from the RIGHT table (in the **JOIN** clause - *Accounting_Journal*) where there was a match.

In the **JOIN**, we also listed other conditions to **pre-filter** data rows from the RIGHT table (the *Accounting_Journal*).

If we didn't do that, we would pull back a query with as many rows as the *Accounting_Journal* itself.

If there were 2 million rows in the *Accounting_Journal* that used account type 22, we would see 2 million matches made just for just that account type.

By pre-filtering the RIGHT table, we are only pulling back the particular GAX we have been looking at from the *Accounting_Journal*.

```
select * from Account_Type as acc

INNER join Accounting_Journal as aj
on acc.Account_Type=aj.Account_Type AND aj.Document_Code='GAX' and aj.Document_ID='74000000001' and
aj.Doc_Department='810'
```

Account_Type	Description	Account_Type	Accounting_Line_Number	Activity
02	Liabilities	02	1	
22	Expenditure	22	1	

Columns Filters

Page Size: 20 **1** to **2** of **2** Page **1** of **1**

An **INNER JOIN** is different than a **LEFT JOIN** in that it keeps **only matches**. The **LEFT JOIN** kept the data from the LEFT table even when it did not match.

Of course, we can do that **LEFT JOIN** *and then use a* ***WHERE*** *filter* to get rid of useless matches. But it is more efficient to get rid of those useless matches in the actual join. Why? The **WHERE** clause filters the data **after** the join does all its matches.

Thus, you want to avoid a **hidden join explosion** - where the join explosion actually happens, but where the WHERE clause cleans up the mess, throwing out 99% of the join matches.

This will make your SQL queries take a long time, even though only few rows are returned.

```
select * from Account_Type as acc

left join Accounting_Journal as aj on acc.Account_Type=aj.Account_Type
Where aj.Document_Code='GAX' and aj.Document_ID='74000000001' and aj.Doc_Department='810'
```

Account_Type	Description	Account_Type	Accounting_Line_Number	Activity
02	Liabilities	02	1	
22	Expenditure	22	1	

Columns Filters

Page Size: 20 **1** to **2** of **2** Page **1** of **1**

Now you know the basics of SQL!