# Migrating your SQL Server Workloads to PostgreSQL

Shameel Ahmed

# Migrating your SQL Server Workloads to PostgreSQL

**Shameel Ahmed Z**

# Complimentary Copy

Originally published in May 2020, the first edition (this book) laid the groundwork for exploring the intricacies of migrating databases from SQL Server to PostgreSQL. It reflects my early insights and findings, serving as a starting point for developers, architects, and database professionals alike. This book is presented to you free of cost to enable you to get started on your database migration journey. While this edition provides a solid foundation, I am confident that you will find the second edition to be an indispensable companion on your journey through the database migration. Should you find the content of this edition insightful and engaging, I invite you to consider purchasing the second edition to gain access to even more comprehensive insights, updated information, and additional resources.

## The Second Edition

Since the initial release of the book, PostgreSQL has evolved significantly, prompting the development of a second edition, released in October 2022. This updated edition not only expands upon the concepts introduced in the first edition but also includes four new chapters:

- Chapter 2 provides an introduction to PostgreSQL.
- Chapter 5 discusses PostgreSQL Cloud/PaaS offerings.
- Chapter 7 discusses Security
- Chapter 8 provides Migration Tips.

If you would like to purchase the second edition, please visit:

India: https://www.amazon.in/dp/B0883CXYVX

US: https://www.amazon.com/dp/B0883CXYVX

UK: https://www.amazon.co.uk/dp/B0883CXYVX

It is also available in other countries. Login to your Amazon account and search for the title of the book.

# Introduction

This concise eBook is intended to be a ready reference and a checklist for migrating your databases, applications and services from Microsoft SQL Server to PostgreSQL and discusses all the important information you need to be aware of before beginning a migration project.

I have had the opportunity of designing and executing a large migration involving several big SQL Server Instances (with over 100 databases in total) to AWS Aurora (PostgreSQL compatible) with a total size of over 20 terabytes. During the migration, many mistakes were made, and a lot of rework done resulting in several learnings for the team. These learnings are shared here.

While I have taken every effort to make sure that the information presented here is accurate; things might have changed, features that were not supported in older versions might be supported in newer versions, etc. If you find something inaccurate, misleading or grossly wrong, please feel free to leave a comment with the details and the links to support your claims, I would be glad to update the eBook with the correct information.

This eBook is divided into four chapters:

## Chapter 1: Getting Started

1. Introduction to Open Source Software, its advantages and disadvantages
2. Moving from Licensed software to Open Source software
3. Basics and the history of Microsoft SQL Server and PostgreSQL
4. Choosing the right PostgreSQL Edition and Version for your needs
5. Production, Development & Testing Workloads
6. On-Premise vs Cloud comparison

## Chapter 2: Platform Comparison

1. In-depth comparison of SQL Server and PostgreSQL
2. Concurrency Control
3. Database Structure comparison
4. Feature comparison
5. Data Type Mapping
6. Built-In Functions and Operators
7. SQL Syntax differences
8. Areas where PostgreSQL has an edge
9. Areas where SQL Server has an edge
10. GUI Client Tools

## Chapter 3: The Actual Migration

1. Schema migration
2. Data migration
3. Code migration
4. ETL/ELT Tools
5. Reporting Platforms
6. Analytics Platforms
7. Scheduling Options
8. Application/Services Migration

## Chapter 4: Maintenance and Monitoring

1. Maintenance and Monitoring
2. High Availability, Load Balancing, and Replication
3. Environment Stabilization via Parallel Deployment

*This book is dedicated to my parents, whose never give-up spirit and tireless efforts to save their special child made me who I am today.*

# About the Author

Shameel is an Enterprise Architect and Programmer who has been programming since 1999. He started his career as a VB5 programmer building apps for the Legal community. During this time, he built a lot of ActiveX/COM components for the developer community and distributed them freely. When .NET was released in early 2001, he switched to C# and Web development. He specializes in building high performance applications and services using .NET, Java, SQL and No-SQL databases. His focus areas are Clean coding, Lean Architecture, Code Reuse and Automation. When not programming, he can be found reading books and pursuing his hobbies which include astrophotography and numismatics.

Here is a photo of moon taken on 18th Nov 2018 at Chennai, India; using a 60AZ telescope and mobile phone camera.

# Contents

# CHAPTER 1

# GETTING STARTED

# Why Open Source?

While there is no question about the capabilities and strengths of licensed and proprietary Databases, their highly restrictive pricing model and a significantly higher Total Cost of Ownership (TCO) for large installation bases has made enterprises and medium businesses look towards less expensive open-source solutions. Enterprises have been shying away from Open Source software primarily because there is no support in case of any major issues. Open Source Databases provide the same or even better functionality at a significantly lower cost. Migrating from commercial to open source databases results in significant cost savings for enterprises in terms of licensing and support.

## Moving from Licensed software to Open Source

Licensed software like SQL Server and Oracle are backed by their respective vendors and provide support as part of their license or separately, depending on the licensing model. They have your back (at least on paper) when your databases are down due to software bugs & vulnerabilities, heavy loads or other hardware and network issues.

On the other hand, open source software is provided "as is" if you host it on your servers and you are on your own when there is any issue. But the good news is that there are third-party companies that provide support to Open source databases. Note that the software itself is still free to use, the

price is for support only. One such company, **EnterpriseDB**, provides support for Open Source PostgreSQL as well as its own fork of PostgreSQL.

Cloud solution providers also provide PostgreSQL as PaaS offerings. One such offering is AWS Aurora (PostgreSQL compatible). This model has high availability and support built-in with a pay-as-you-go pricing model and is best suited for small and medium businesses. Note that AWS also provides PostgreSQL as an RDS service at roughly half the cost but without built-in high availability. You can also run PostgreSQL on Cloud at a much cheaper cost by manually hosting PostgreSQL instances in Linux instances running on the Cloud.

## Why PostgreSQL?

PostgreSQL is an enterprise class, feature-rich free open source database system that is highly reliable and extremely performant and very suited for real-time and mission critical applications. It emphasizes on extensibility and SQL compliance.

Until a decade ago, MySQL used to be the default choice of enterprises for migration to Open Source databases. Ever since its acquisition by Oracle in 2010, its roadmap became uncertain and enterprises started moving away from it.

MariaDB is an open source fork of MySQL that was intended to continue the open source legacy of MySQL, but it did not really take off in the enterprise world. There had to be a

solution to fill the gap and PostgreSQL came in as a viable alternative and has become the choice of enterprises for moving their databases to open source. According to DB Engines Ranking for 2019, PostgreSQL is the fourth most popular database MariaDB is 14th.

# The Contenders

## SQL Server: History, Editions and Versions

SQL Server is a relational database management system developed by Microsoft. It was first released in 1989 and has since been in active development. The latest stable version as of Sep 2019 is SQL Server 2017. According to **DB Engines Ranking**, SQL Server is the third most popular Database in 2019 and also the DBMS of the year 2016.

It comes in various editions such as Enterprise, Standard, Developer, Express, Azure, etc. SQL Server Express Edition is a scaled down, free edition of SQL Server, which includes the core database engine. While there are no limitations on the number of databases or users supported, it is limited to using one processor, 1 GB memory and 10 GB database files.

SQL Server, being a proprietary Microsoft software, has traditionally been available only on the Windows operating system. But starting SQL Server 2017, it is available on Linux too. It is also available as PaaS offerings from Cloud providers like Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP).

For more information, visit **SQL Server Home Page** or the **Wikipedia** page or the **DB Engines** page.

# PostgreSQL: History, Editions and Versions

PostgreSQL is the most advanced open source object-relational database system. It has been in development for over 30 years and is managed by the PostgreSQL Global Development Group. According to **DB Engines Ranking**, PostgreSQL is the fourth most popular Database in 2019 and also the DBMS of the year for 2017 and 2018.

PostgreSQL started as a fork of the Ingres Database in 1982 and was first released in 1996. The latest version is PostgreSQL 11.5 as of Sep 2019. It is a very mature and stable product with an extremely rich set of features and tools. This eBook discusses migrating to PostgreSQL 11.x unless otherwise specifically mentioned.

PostgreSQL is available on Linux, FreeBSD, OpenBSD, macOS and Windows, and as PaaS offerings from Cloud providers like Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP).

To get started with PostgreSQL, visit its **website** or the **Wikipedia** article or the **DB Engines** page.

# Choosing the right PostgreSQL Edition and Version for your needs

PostgreSQL is an open source database and is available in a wide range of varieties. The first thing to do is to decide where to host your PostgreSQL database. The available options are:

1. On-premise hosting on Linux infrastructure
2. On-premise hosting on Windows infrastructure
3. Cloud hosting using IaaS such AWS EC2 VMs running Linux
4. Cloud hosting using IaaS such AWS EC2 VMs running Windows
5. Cloud hosting using PaaS solutions like AWS RDS or Aurora. These solutions almost always run on Linux owing to stability and affordability.

The next thing to do is to choose your PostgreSQL version. For On-prem hosting, the rule of thumb is to always start with the latest stable release (PostgreSQL 12 as of this writing) and then keep upgrading to newly released stable versions (keeping support lifecycle in mind). Cloud providers like AWS allow you to choose the last three or four stable releases and take some time to certify and make available the latest stable release. They also allow you to do in-place upgrade for minor version upgrades and major version upgrades with minimal downtime.

Check out the **Feature Matrix** page for a version-wise comparison of features. Check out the relevant PostgreSQL offering page of the different Cloud providers for features and limitations before signing up.

## Production Workloads

PostgreSQL works best with Linux, so always choose Linux for your production workloads for maximum performance, stability and high-availability. SQL Server traditionally runs on Windows and chances are that if you are planning to reuse your SQL Server hardware, you may want to run PostgreSQL on Windows. You will most probably not be able to do this since you will have to have your PostgreSQL databases instances up and running along with your SQL Server databases in parallel for a period before you can retire your SQL Server databases.

## Development, Testing, Staging or Pilot Workloads

For development and testing, you can either choose Linux or Windows running on on-premise or cloud infrastructure depending on what works best for your organization and the skillset of your team.

Here is a cost-saving tip. You can run all your non-production PostgreSQL databases in Cloud and turn them off when not needed to save on billing. For example, AWS EC2s can be stopped and they will not be billed until you boot-up the instances again. AWS RDS instances can be stopped for up

to a period of seven days, after which the databases will start up again automatically. This restriction is in place to enable the database to be patched automatically for critical security updates.

A neat trick is to take a backup of the database, move the backup to an S3 bucket and then terminate the instance. When development resumes later, launch a new instance and restore from the backup. To avoid changing the database endpoints in code whenever a new database is launched, you can create a DNS entry that points to the current database and use the DNS endpoint in your code and jobs. The only place which will require a change when a new database is launched is the DNS record.

This is cheaper than having physical servers in your data centers throughout the year and spending on their hardware, power and maintenance.

# On-Premise vs Cloud

| | ON-PREMISE | CLOUD IAAS | CLOUD PAAS |
|---|---|---|---|
| **Hosting** | PostgreSQL is installed on servers in your Data Center | You launch Cloud Virtual Machines like AWS EC2 and install PostgreSQL | This is Database As A Service, no installation required. You select the launch parameters and launch the service |
| **Clustering** | Manual clustering setup is required | Manual clustering setup is required | Clustering is built-in depending on the type of service offering |
| **High Availability and Automatic Failover** | High Availability, Load Balancing, and Replication must be set up manually. | High Availability, Load Balancing, and Replication must be set up manually. | High Availability is built-in depending on the service offering. For example, AWS Aurora provides High |

| | ON-PREMISE | CLOUD IAAS | CLOUD PAAS |
|---|---|---|---|
| | | | Availability whereas plain vanilla AWS RDS instance does not. |
| **Tools** | You have control over the hardware and therefore should be able to run all PostgreSQL tools on the servers. | You have control over the hardware and therefore should be able to run all PostgreSQL tools on the servers. | The Cloud Platform may restrict you from running certain tools. For example, pgAgent is not available In AWS Aurora |
| **Scheduling** | You can use pgAgent or other third-party schedulers to schedule data transfer jobs and maintenanc e jobs on the servers. | You can use pgAgent or other third-party schedulers to schedule data transfer jobs and maintenance jobs on the servers. | Cloud providers have their own built-in services for scheduling jobs. For example, AWS has its own scheduling service and does **not** supp |

| | ON-PREMISE | CLOUD IAAS | CLOUD PAAS |
|---|---|---|---|
| | | | ort pgAgent in RDS or Aurora. |
| **Management & Monitoring** | Management & Monitoring should be done manually | Management & Monitoring should be done manually | The Cloud Platform automatically manages the databases for you and provides GUI/portal to monitor the health of your databases. E.g., AWS CloudWatch |
| **Support** | Support for On-Premise installations are usually the organization's infrastructure support team. Some third-party companies provide | Support for On-Premise installations are usually the organization's infrastructure support team. Some third-party companies provide support for Cloud IaaS installations. | Support is provided by the Cloud Provider through automated Support systems. |

| | ON-PREMISE | CLOUD IAAS | CLOUD PAAS |
|---|---|---|---|
| | support for On-Premise installations. | | |
| **Backup & Restore** | Backup and Restore must be done manually or automated jobs need to be set up. | Backup and Restore must be done manually or automated jobs need to be set up. | Automated Backup and Restore feature is available with most Cloud providers. For example, you can set up automated backups and restore from backups through AWS Console. |
| **Cost** | Infrastructure costs only. If you already own the hardware, then this will probably be the cheapest | Clouds VMs like AWS EC2 are comparatively cheaper depending on the configuration you choose. You may have to spend | Cloud PaaS are usually expensive since they have many High Availability and several other features built-in. |

| | ON-PREMISE | CLOUD IAAS | CLOUD PAAS |
|---|---|---|---|
| | option of the three. | additional cost if you choose to use third-party licensed software for monitoring, management, etc. | |

When talking about Cloud providers, I always give examples from AWS because that is only cloud platform I have migrated databases to, so far. It is in no way an endorsement of one provider over another. Other providers have similar offerings, please visit their website to learn more about their offerings.

# CHAPTER 2

# PLATFORM COMPARISON

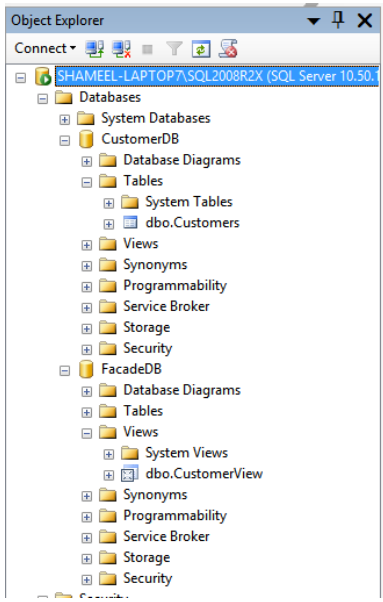
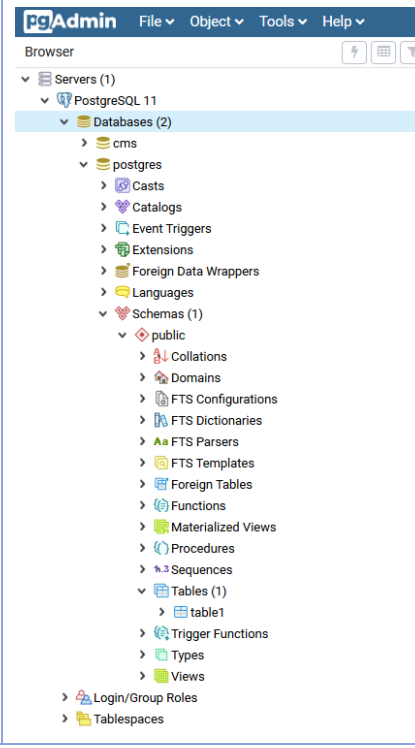
# SQL Server vs PostgreSQL

For a high-level comparison, check this DB Engines page **here**.

## The Database Structure

Like SQL Server, PostgreSQL can contain multiple Databases within a single instance, and each database can contain multiple Schemas. Each of these schemas can contains other database objects like Tables, Views, Stored Procedures, Functions, etc.

When a connection to a database is opened in PostgreSQL, you can only refer to that database within that connection. If your queries on the connection refer to a different database running on the same instance, PostgreSQL will throw an error. You either open a connection for each database separately or use dblink or Foreign Data Wrappers to query tables from multiple databases.

| SQL SERVER | POSTGRESQL |
| --- | --- |
|  |  |

# Concurrency Control

Concurrency control of row data is an important aspect of database systems that determines how efficiently row data is maintained and updated. Different database systems have different concurrency control mechanisms that have a direct impact on their performance.

## SQL Server Concurrency Control

SQL Server versions prior to 2005 traditionally used Row level exclusive locking that locks the entire row during write operations. This prevents other read and write threads from accessing the row and they had to wait till the current write operation on the row completed (either committed or rolled back). This has a significant impact on performance of the database during heavy concurrent writes and reads.

SQL Server 2005 improved this scenario by introducing "row-versioning isolation levels". See **Locking and Row Versioning**. Note that there are two separate MVCC implementations, *read committed isolation using row versioning* (RCSI) and *snapshot isolation* (SI)

SQL Server queries can return dirty (uncommitted) rows depending on the isolation level of the transactions.

## PostgreSQL Concurrency Control

PostgreSQL maintains data consistency using a multiversion model (Multiversion Concurrency Control, MVCC), which

means that each SQL statement sees a snapshot of data (a *database version*) as it was some time ago, regardless of the current state of the underlying data. This prevents statements from viewing inconsistent data produced by concurrent transactions performing updates on the same data rows, providing *transaction isolation* for each database session. MVCC avoids locking rows which minimizes lock contention and improves overall performance.

The main advantage of using the MVCC model of concurrency control rather than locking is that in MVCC locks acquired for querying (reading) data do not conflict with locks acquired for writing data, and so reading never blocks writing and writing never blocks reading. PostgreSQL maintains this guarantee even when providing the strictest level of transaction isolation using a *Serializable Snapshot Isolation* (SSI) level.

PostgreSQL queries never return dirty rows, no matter what the isolation level of the transaction is.

## Procedural Programming: T-SQL vs PL/pgSQL

The SQL Server procedural extension to the standard SQL language is called Transact-SQL or T-SQL in short and provides additional capabilities like procedural programming, looping constructs, conditional constructs, session management, Stored Procedures, etc.

PostgreSQL multiple procedural languages and the default language is called PL/pgSQL.

## PostgreSQL has a more refined SQL syntax

Once you start using PostgreSQL, you will notice that it has a much cleaner and refined language syntax compared to SQL Server. Compare these statements:

### SQL Server

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[customers]') AND type IN (N'U'))

BEGIN

    CREATE TABLE [dbo].[customers]( .... )

END
```

### PostgreSQL

```
CREATE TABLE IF NOT EXISTS customers ( .... )
```

PostgreSQL code is much simpler, cleaner and refined. SQL Server code is a maintenance nightmare.

Few more examples:

### SQL Server

```
-- This throws an error if the table does not exist

DROP TABLE [dbo].[customers]
```

```
-- This is how you do it in SQL Server 2014 and older

IF EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[customers]') AND type IN (N'U'))

BEGIN

    DROP TABLE [dbo].[customers]

END

-- This works in SQL Server 2016 and newer

DROP TABLE IF EXISTS [dbo].[customers]
```

## PostgreSQL

```
DROP TABLE IF EXISTS customers;

-- No error if the table does not exist
```

Similarly, PostgreSQL supports `CREATE OR REPLACE` syntax for Stored Procedures, Functions, Views etc. But in SQL Server 2014 and prior, you had to check if the object exists and then drop it before attempting to create it again. You can use the ALTER statement, but it throws an error if the object does not already exist. SQL Server 2016 SP1 introduced the `CREATE OR ALTER` syntax for Stored Procedures, Functions, Triggers and Views.

SQL Server 2016 and above support `DROP IF EXISTS` for Tables/Views but still does not support the `CREATE IF NOT EXISTS` syntax.

## Case sensitivity of object names

In SQL Server, object names Customers, CUSTOMERS and customers are all the same. SQL Server does not impose case-sensitivity in object names and creates objects with the same case specified in the CREATE statement.

In PostgreSQL, object names are silently converted to lowercase. When a name is enclosed in double quotation marks, the name becomes case sensitive and must be used with quotation marks in queries. In other words, the quote marks become part of the name. Therefore, the above names are converted to customers, but "Customers" (with quotes) is treated as a different object as must the used with quotes in queries that reference the object. This rule holds good for column names in tables as well.

## Database Objects/Features

Migrating database objects from SQL Server to PostgreSQL is straightforward as most of these objects are supported as-is in the target platform.

| OBJECT/ FEATURE | SQL SERVER | POSTGRESQL | COMMENTS |
|---|---|---|---|
| **Table** | Yes | Yes | |
| **Indexes** | Yes | Yes | |
| **Triggers** | Yes | Yes | |
| **Views** | Yes | Yes | |
| **Updatable Views** | Yes | Yes | |
| **Materialized Views** | Yes | Yes, starting v9.3 | |
| **Computed Column** | Yes | Yes, available in v12. Use Views for earlier versions | Computed Columns are called Generated Columns in PostgreSQL |
| **Stored Procedures** | Yes | Yes, starting v11 | Prior to PostgreSQL 11, developers used Functions instead of Stored Procedures |
| **User Defined Functions** | Yes | Yes | |

| | | | |
|---|---|---|---|
| **Overloaded Functions** | No | Yes | In PostgreSQL, you can have multiple functions with the same name but different parameters. |
| **Common Table Expressions (CTE)** | Yes | Yes | PostgreSQL 11 and older: CTEs are always materialized and may not perform well. PostgreSQL 12: Query hints allow CTEs to be either materialized or non-materialized. |
| **Functional Indexes** | No | Yes | |
| **Max Table Size** | Unlimited | 32 Terabytes (32TB) in PostgreSQL 9.6 or earlier 2 Exabytes (2EB) in PostgreSQL 10 | |

| | | | |
|---|---|---|---|
| **Max Columns per table** | 1024 (non-wide) 30000 (wide) | 250 – 1600 depending on column types | |
| **Max Database size** | 524,272 Terabytes | Unlimited | |
| **Max Varchar Length** | Varchar(Max) | Varchar(10485760) Use text for longer strings | |
| **Scheduler** | SQL Server Agent | pgAgent | |

**SQL Server Maximum Capacity Specifications (v2012 and below)**

**SQL Server Maximum Capacity Specifications (v2014)**

# Data Types

Regardless of whether you use tools or do things manually, keep the following data type mappings between SQL Server and PostgreSQL data types while doing the migration. Some tools allow you to play around the data type mappings to fine tune the migration process.

| MICROSOFT SQL SERVER | POSTGRESQL | DESCRIPTION |
|---|---|---|
| **TEXT** | | |
| CHAR(n) | CHAR(n) | Fixed length char string, 1 <= n <= 8000 |
| VARCHAR(n) | VARCHAR(n) | Variable length char string, 1 <= n <= 8000 |
| VARCHAR(max) | TEXT | Variable length char string, <= 2GB |
| NVARCHAR(n) | VARCHAR(n) | Variable length Unicode UCS-2 string |
| NVARCHAR (max) | TEXT | Variable length Unicode UCS-2 data, <= 2GB |
| TEXT | TEXT | Variable length character data, <= 2GB |

| | | |
|---|---|---|
| NTEXT | TEXT | Variable length Unicode UCS-2 data, <= 2GB |
| UNIQUEIDENTIFIER | CHAR(16) | 16 byte GUID(UUID) data |
| **NUMERIC** | | |
| BIGINT | BIGINT | 64-bit integer |
| INTEGER | INTEGER | 32 bit integer |
| TINYINT | SMALLINT | 8 bit unsigned integer, 0 to 255 |
| DOUBLE PRECISION | DOUBLE PRECISION | Double precision floating point number |
| FLOAT(p) | DOUBLE PRECISION | Floating point number |
| NUMERIC(p,s) | NUMERIC(p,s) | Fixed point number |
| SMALLMONEY | MONEY | 32 bit currency amount |
| **DATE** | | |
| DATE | DATE | Date includes year, month and day |

| | | |
|---|---|---|
| DATETIME | TIMESTAMP(3) | Date and Time with fraction |
| DATETIME2(p) | TIMESTAMP(n) | Date and Time with fraction |
| DATETIMEOFFSET(p) | TIMESTAMP(p) WITH TIME ZONE | Date and Time with fraction and time zone |
| SMALLDATETIME | TIMESTAMP(0) | Date and Time |
| **BOOLEAN** | | |
| BIT | BOOLEAN | 1, 0 or NULL |
| **BINARY** | | |
| BINARY(n) | BYTEA | Fixed length byte string |
| VARBINARY(n) | BYTEA | Variable length byte string, 1 <= n <= 8000 |
| VARBINARY (max) | BYTEA | Variable length byte string, <= 2GB |
| ROWVERSION | BYTEA | Automatically updated binary data |
| IMAGE | BYTEA | Variable length binary data, <= 2GB |

This list contains only commonly used data types, PostgreSQL supports a vast variety of data types and supports custom data types. For a comprehensive list of major data types supported by PostgreSQL, check out **this** post.

## Built-In Functions & operators

PostgreSQL has an extensively rich set of operators and built-in functions, way beyond what is offered by SQL Server.

| MICROSOFT SQL SERVER | POSTGRESQL | COMMENTS |
|---|---|---|
| DATEPART | DATE_PART | |
| ISNULL | COALESCE | |
| SPACE | REPEAT | |
| DATEADD | + | |
| + | \|\| | String concatenation |
| CHARINDEX | POSITION | |
| GETDATE | NOW | |
| LTRIM/RTRIM | TRIM | |
| REPLACE | OVERLAY | |
| LEN | OCTET_LENGTH | |
| INT IDENTITY | SERIAL data type | |

For a comprehensive list of all PostgreSQL functions and operators, check out **this** page.

# SQL Language differences

| FEATURE | MICROSOFT SQL SERVER | POSTGRESQL |
|---|---|---|
| Select first N rows | TOP n | LIMIT n |
| Statement terminator | ; (not required) | ; (required) |
| LIKE operator | Case insensitive by default (determined by collation) | Case sensitive, use ILIKE for case insensitive comparison |
| Regular Expressions | LIKE operator | SIMILAR TO operator |

## CTE Performance Differences

In SQL Server, this query:

```
WITH AllUsers AS (SELECT * FROM Users)

SELECT * FROM AllUsers WHERE Id = 100;
```

results in a query plan for the entire query at once, and the WHERE clause filter is passed into the CTE. The resulting query plan is efficient, doing just a single clustered index seek.

In PostgreSQL, CTEs are optimization fences (outer query restrictions are not passed on to CTEs) and the **database evaluates the query inside the CTE and caches the results (i.e., materialized results)** and outer WHERE clauses are applied later when the outer query is processed, which means either a full table scan or a full index seek is performed and results in horrible performance for large tables. To overcome this, you rewrite this query in PostgreSQL as:

```
WITH UserRecord AS (SELECT * FROM Users WHERE Id = 100)

SELECT * FROM UserRecord;
```

The other option is to rewrite the query using a subquery. Keep that in mind when migrating code and queries that involve CTEs.

Note that PostgreSQL 12 addresses this problem by introducing query optimizer hints to enable us to control if the CTE should be materialized or not: MATERIALIZED, NOT

MATERIALIZED. So, the query can be rewritten as follows to achieve better performance.

```
WITH AllUsers AS NOT MATERIALIZED (SELECT * FROM Users)

SELECT * FROM AllUsers WHERE Id = 100;
```

# Collation / Ordering

There are few fundamental differences in the way SQL Server and PostgreSQL store and compare data. In SQL Server, the default collation is case insensitive. In PostgreSQL, it is case sensitive. Therefore, your code/query that searches for text in WHERE clause or ON clause in joins or LIKE statement might fail. PostgreSQL provides an ILIKE statement for case insensitive comparison. For example:

## SQL Server

```
//Returns all names starting with Alic: Alice, Alicia,
alice, alicia

SELECT * FROM AllUsers WHERE UserName LIKE ‘alic%’;
```

## PostgreSQL

```
//Returns all names starting with alic: alice, alicia

SELECT * FROM AllUsers WHERE UserName LIKE ‘alic%’;

//Returns all names starting with Alic: Alice, Alicia,
alice, alicia

SELECT * FROM AllUsers WHERE UserName ILIKE ‘alic%’;
```

## When Delete does not delete

In SQL Server, when a DELETE statement is issued against a table, the rows are permanently deleted from the storage. In PostgreSQL they are just marked for deletion (soft delete) and not immediately removed from storage. The UPDATE statement behaves similarly, the old row is marked for deletion and a new row is inserted with the new row data.

Keep that in mind when migrating tables that undergo heavy deletes and updates regularly, because they will keep growing exponentially until you do a VACUUM. You will have to set up a job that Vacuums the tables in your databases at regular intervals which could be daily, weekly or monthly depending on the growth rate of each table.

The **AutoVacuum** daemon is turned on by default and ensures that Vacuuming is done automatically at regular intervals, so you may not have to worry in most cases. However, it could be turned off during installation or later by the Database Admin. If you host your PostgreSQL databases in AWS Aurora, you need to be careful even when AutoVacuum daemon is running. Check out **this** post for more information.

See **VACUUM** statement for more details.

## Where PostgreSQL has an edge

PostgreSQL provides a rich set of operators and functions to work with different types of data.

## PostgreSQL literally runs anywhere

PostgreSQL runs on Windows, Linux, Unix, etc. whereas SQL Server 2016 and older run only on Windows. SQL Server 2017 runs on Linux as well.

## Inserting Test data into a PostgreSQL table is a breeze

This query inserts one million rows into the customers table with random data.

```
INSERT INTO customers (id, name)

SELECT id, md5(random()::text)

FROM generate_series(1, 1000000) AS id;
```

In SQL Server, you will be able to do this only using procedural code.

## Multiple language support in PostgreSQL

There are currently four procedural languages available in the standard PostgreSQL distribution: PL/pgSQL, PL/Tcl, PL/Perl and PL/Python. PostgreSQL supports many other procedural languages as well, you just need to install the appropriate extension and enable it.

- ***PL/pgSQL***: this PostgreSQL's native procedural language is like SQL Server's T-SQL, but more refined and feature-complete.

- ***PL/Python***: PostgreSQL's support for Python adds the enormous ecosystem of Python libraries to your arsenal. Imagine being able to use Python functions in SQL queries, unbelievable, isn't it?
- ***PL/Perl***: PostgreSQL has full support for Perl as a procedural language.
- ***PL/V8***: This JavaScript engine is stable, feature-packed and extremely fast. The fact that PostgreSQL natively supports JSON data type make it powerful and flexible. This language supports caching data in memory which makes it an ideal language for data intensive row level operations.
- ***PL/R***: PostgreSQL has full support for R, a statistical programming language used extensively in data science that has a robust set of high-quality plugins and add-ons.
- ***PL/Tcl***: PL/Tcl is a loadable procedural language for the PostgreSQL database system that enables the **Tcl language** to be used to write PostgreSQL functions.
- ***C***: Though PostgreSQL supports C, it must be compiled separately. This comes in handy when speed and fine control of memory management, resource usage for tasks and performance are critical.

PostgreSQL also has extensions for other languages like Java, Ruby, PHP, Lua, Tcl, etc.

One can argue that SQL Server also supports multiple languages because it acts as a .NET host and can run code written in any .NET languages like C#, VB.NET, F#, etc. That is correct but involves writing code in a .NET IDE like Visual

Studio, compiling code, deploying the assembly in SQL Server, writing a wrapper T-SQL function for your C# function and then calling the function from your T-SQL code. Not to mention the debugging hell. You cannot write C# code "in-line" within your T-SQL code the way the LANGUAGE keyword in PostgreSQL allows you to write a function in multiple languages.

Having personally done multi-language programming in SQL Server, I can vouch for the fact that the level of multi-language support provided by PostgreSQL is on a totally different plane compared to that of SQL Server.

## PostgreSQL has rich set of Functions and Operators

The GREATEST function is like the MAX function, except that it works across columns. The LEAST is like MIN function, except that it works across columns.

```
SELECT GREATEST(LastLoginDate, LastReportDate,
LastInteractionDate) FROM Users;
```

You cannot do this in SQL Server, except with multiple SELECT statements and procedural constructs. Similarly, there are tons of functions available in PostgreSQL which are not available in SQL Server.

## PostgreSQL has much better support for CSV

People working with data on a day-to-day basis would work with CSV often to convert data from one format to another

and for analytical processing, etc. PostgreSQL's COPY FROM and COPY TO command do a much cleaner job at handling CSVs: no more truncated texts, no more encoding nightmares, no more quoting/escaping issues.

## You can drop an entire Schema in PostgreSQL with a single statement

In PostgreSQL, all you do is execute DROP SCHEMA CASCADE. This is very useful during development but at the same time risky as hell in your Staging and Production environments.

In SQL Server, DROP SCHEMA does not support the CASCADE clause. You must drop all the objects within the schema individually before dropping the schema, which can be a nightmare during development and prototyping.

## Unicode and Character Encoding Basics

Unicode is a character encoding standard and UTF-8 and UTF-16 are different implementations of the standard.

UTF-8: UTF-8 uses one byte for characters codes below 128 and two, three or four bytes for characters beyond that. It is compatible with ASCII.

UTF-16: Always uses two or four bytes, not compatible with ASCII

UCS-2: Always uses two bytes.

## Unicode Support

PostgreSQL has native support for UTF-8 encoding and its CHAR, VARCHAR and TEXT types are UTF-8 by default. String operations and regular expressions are UTF-8 aware. PostgreSQL does not support UTF-16.

SQL Server (prior to 2012) support only UCS-2, a subset of UTF-16. SQL Server 2012 introduced optional support for UTF-16, but you must select an UTF-16 collation for your database for it to work. SQL Server does not support UTF-8.

# Where SQL Server has an edge

## In PostgreSQL, you cannot query from multiple databases directly within a single query

Unlike SQL Server, you cannot reference two databases within a query directly. But there is a workaround, you can use a dblink or a Foreign Data Wrapper.

## You cannot execute procedural code directly in PostgreSQL

In SQL Server, you can declare variables or use conditional logic in scripts directly:

```
DECLARE @MeanAge = 100

IF EXISTS (SELECT * FROM Users WHERE Age > @MeanAge)

    SELECT 'Yay'

ELSE

    SELECT 'Nay';
```

SQL Server will allow you to run this code with a Client tool or in Inline queries in application code. In PostgreSQL, you can run procedural code like this:

```
DO $$

-- declare

BEGIN
```

```
  /* pl/pgsql here */

END $$;
```

However, you cannot return any results to the client. The only way you can run procedural code and return results to the client is by wrapping it within a Stored Procedure or Function and then executing it.

## PostgreSQL does not support Stored Procedures prior to version 11

PostgreSQL 10.x and below does not have Stored Procedures. But this is not a showstopper as you can do everything with a User Defined Function, with the only exception that you cannot have transactions inside a function.

## PostgreSQL does not support Computed Columns prior to version 12

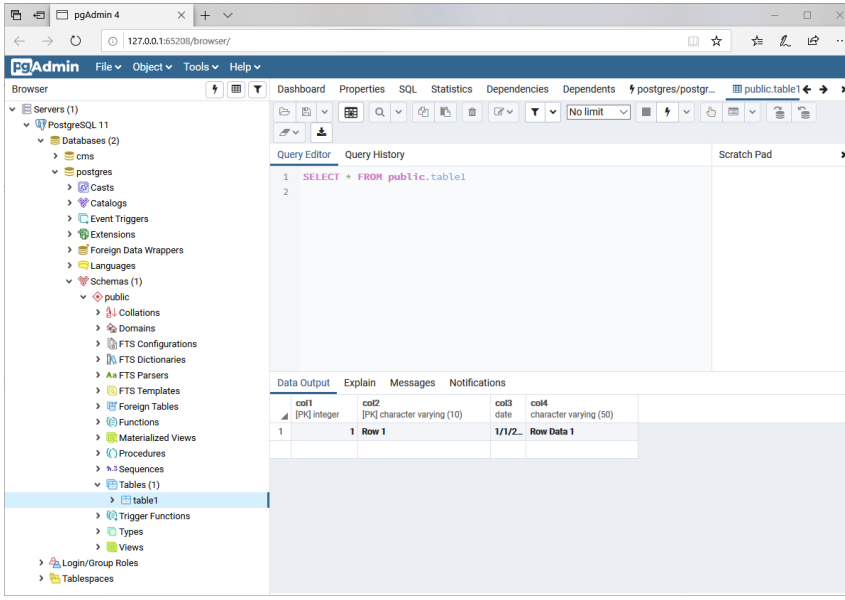
PostgreSQL 11.x and below do not support Computed Columns. However, PostgreSQL 12, scheduled to be released in late 2019 introduces support for 'Generated columns'. Check out **this** article for more information.

# GUI Tools

SQL Server Management Studio provides an excellent GUI to manage and query your SQL Server databases. With PostgreSQL being open source, there are a plethora of open source and commercial tools available for managing and querying PostgreSQL databases. The most common is pgAdmin.

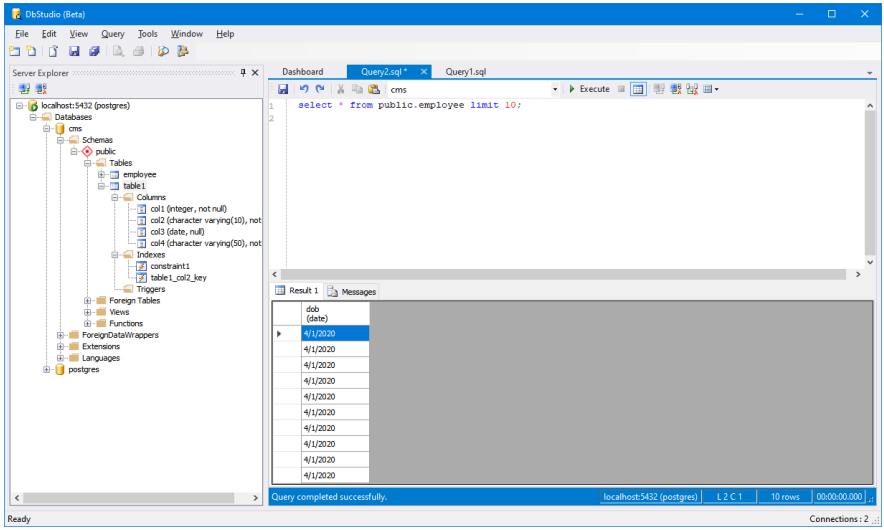
## pgAdmin

**pgAdmin** is a free open source GUI tool and the most popular one for managing PostgreSQL databases. The latest version 4 is web based to enable Cloud deployments.



# DbStudio

**DbStudio** is a tool I am developing for managing and querying PostgreSQL database and many other databases. The primary goal of the project is to provide a consistent look and feel based on SQL Server Management Studio that works across different database platforms like PostgreSQL, SQL Server, Oracle, MySQL, etc. It is still under development and is currently available as a beta to test and evaluate the tool.



# CHAPTER 3

# THE ACTUAL MIGRATION

# Getting Started with the actual migration

The first steps in the migration process is to gather and analyze the following information:

1. The number of SQL Server Instances to be migrated
2. The total number of Databases across all such instances
3. The total size of the data to be migrated in terms of number of tables, number of rows in each table and the total size of the data (e.g. 500 tables, 20 million total rows and 10 TB of data)
4. The number of objects to be moved (Tables, Views, Stored Procedures, Jobs etc.)
5. The capabilities and shortcomings of the source platform and the target platform and their comparison.
6. Workarounds/Alternative solutions for features that are not supported by the target platform (such as ETL, OLAP, etc.)
7. Application code migration
8. Jobs, Queries and Reports migration
9. Data warehousing / OLAP considerations
10. Post migration performance comparison and performance improvement
11. Post migration monitoring and support

# Schema migration

Once you have all the requirements gathered in hand and have already set up your PostgreSQL database instance, the first step is to migrate the database schema and the objects.

## Migration Tools

Migrating Schema from SQL Server to PostgreSQL can be done either manually or using tools. There are free and commercial tools available to migrate schema and data from one platform to another.

Free tools are developed and provided by PostgreSQL community enthusiasts.

### Free/Open Source Tools

**pgloader** migrates SQL Server database to PostgreSQL. It supports automatic discovery of the schema, including build of the indexes, primary key and foreign keys constraints, and provides various casting rules which can convert the SQL Server data type to a PostgreSQL equivalent data type. It also supports loading data.

**Sqlserver2pgsql** is a Perl script that converts SQL Server schema to a PostgreSQL schema. It can also optionally produce a Pentaho Data Integrator (Kettle) job to migrate the Table data from SQL Server to PostgreSQL.

## ETL Tools

Though ETL tools like Talend and Pentaho Kettle provide the option of creating the schema on the go while loading data. This should never ever be done because the indexes, triggers and constraints are not carried over during the migration. Only the table structure with the columns with inferred data types are created. This can be helpful while loading test data but quickly becomes a nightmare if you use it to load production data.

Check out the ETL Considerations section for a detailed discussion on the topic.

## Cloud-specific Vendor Tools

If you are migrating your SQL Server databases to AWS Aurora (PostgreSQL compatible), you can use AWS Schema Conversion Tool (SCT) to migrate your schemas. SCT is a desktop application that analyzes your SQL Server Schema and produces PL/PGSQL scripts to migrate the schemas to PostgreSQL. The SCT does a very good job at migrating your Tables, Views, Indexes, etc. but only partially and incompletely migrates code objects like Stored Procedures and Triggers. You will realize that you must do a lot of rewriting to make it work properly with PostgreSQL. My advice is to not depend on auto-generated code and rewrite all SQL Server Stored Procedures to PostgreSQL functions.

Note that this tool does not migrate the data. To migrate the data, you can either use an ETL tool like Talend or Pentaho or optionally use the AWS Data Migration Service, which is an AWS offering for migrating huge data from source systems to AWS Aurora databases.

Similarly, other Cloud providers have similar tools to migrate schema and data to PostgreSQL databases.

## Manual migration of schema

Manual migration involves:

1. Manually analyze every database object
2. Use the SQL Server Management Studio to generate scripts for the Tables, views, Stored Procedures, etc.
3. Make changes to the script to make it compatible with PostgreSQL syntax.
4. Execute the Scripts on the PostgreSQL instances using pgAdmin or a similar client tool.

This is the most accurate way of migrating the schemas and data, but unfortunately works only for small databases with few tables. This is an impractical approach for large databases with hundreds of tables and Stored Procedures.

# Data Migration

Once the Schema is migrated and the tables have been created, the next logical step is to move the data from the SQL Server database tables to the PostgreSQL database tables.

## Open Source Tools

As already discussed in schema migration section, there are free tools available to migrate data along with schema. pgloader and Sqlserver2pgsql allow you to migrate data.

AWS Data Migration Service (DMS) is specifically meant for migrating data from external sources to AWS Aurora databases. Note that this service is not free and is charged based on the size of data being transferred.

## ETL Tools

ETL tools like Talend and Pentaho are extremely good solutions to move large volume of data between databases. You can create Talend packages or Pentaho Kettle Packages to transfer data between your database tables in an easy, efficient and stable manner.

## Manual migration of data

Manual migration involves the following steps:

1. Use SQL Server Management studio to generate INSERT scripts with data.

2. Modify the scripts to conform to PostgreSQL syntax. The good news is that there is not much difference between the INSERT statement syntax between SQL Server and PostgreSQL.
3. Use pgAdmin or a similar client tool to execute the scripts in the PostgreSQL database.

## Code Migration

The major challenge with any database migration is converting the Stored Procedures, Functions and Queries from the Source Database platform to the target Database Platform syntax. This step could become the most time consuming task in your migration depending on the number of Stored Procedures/Functions and the amount of code to be converted. Most automated migration tools do a pathetic job of migrating code and achieve no more than 10% - 15% accuracy which forces manual review and rewriting of entire code.

## Jobs Migration

Data Load, maintenance and Data Purge jobs that use SSIS packages and run on SQL Agent can be migrated to ETL Packages in Talend or Pentaho and scheduled using the different solutions discussed in the 'Scheduling Options' section.

# ETL/ELT Platforms

ETL refers to Extract, Transform and Load, it is a 3-step process applied to extract the data from various sources (which can exist in various forms), clean-up and transform, and load into a target database. Database systems usually receive data from upstream systems and send data to downstream systems through ETL/ELT processes.

SQL Server Integration Services (SSIS) was the ETL tool of choice of SQL Server databases and came bundled with older versions of SQL Server, but Microsoft has recently separated the two and bundled SSIS under SQL Server Data Tools (SSDT) that is free for use. That means you can literally keep your SSIS packages and make them work with PostgreSQL. I strongly advise you against this approach because SSIS does not have a native driver to connect to PostgreSQL. You will either must use ODBC or use commercial third-party drivers.

Open Source databases usually have a plethora of Open Source solutions for ETL/ELT. Talend and Pentaho are two such tools that started off as Open Source projects but were later acquired by other companies. You can use the Open Source editions without support or use the Commercial editions if you require support.

## Talend

**Talend** is a fast ETL processing machine with in-built support for PostgreSQL. Talend Open Studio is an Eclipse based GUI

tool with drag-drop feature that allows you to create ETL packages and test them. It creates Java code for you behind the scenes and compiles the packages to jar files. Therefore, it can run on any platform that supports Java. Talend is required only to develop the packages and it is not required to be installed on the machines where the jobs are running. It is available both in Open Source and commercial forms. Talend Open Source does not come with a scheduler and scheduling jobs might be a challenge. You will have to resort to platform-provided schedulers like crontab, Windows Task Scheduler or the Cloud schedulers.

## Pentaho

**Pentaho Data Integrator** (Kettle) is another popular ETL tool with in-built support for PostgreSQL. Like Talend, Pentaho Kettle is also a Java based software and can run on any platform with Java support. It started out as an Open Source project and later acquired by Hitachi systems. It is now available as both Open Source and Commercial editions.

# Reporting Platforms

SQL Server's de facto Reporting platform is SQL Server Reporting Services (SSRS). In earlier versions of SQL Server, SSRS came bundled with the database and required a license to use. In recent versions, Microsoft is bundling SSRS as part of SQL Server Data Tools (SSDT) which is royalty free to use. SSRS supports a lot of databases and if you are already on SSRS, you can continue using your Reports by repointing your reports to your PostgreSQL databases.

PostgreSQL does not come with its own Reporting tools, but there are a lot of Open Source Reporting Solutions and Commercial Reporting Solutions that can connect to your PostgreSQL database and generate reports. The list is too long to warrant a discussion on each one of them, however, there few tools that stand out and deserve a mention:

## JasperReports

**JasperReports** Server is a stand-alone and embeddable reporting server. It provides reporting and analytics that can be embedded into a web or mobile application as well as operate as a central information hub for the enterprise by delivering mission critical information on a real-time or scheduled basis to the browser, mobile device, or email inbox in a variety of file formats. JasperReports Server is optimized to share, secure, and centrally manage your Jaspersoft reports and analytic views

## BIRT

The Business Intelligence and Reporting Tools Project **(BIRT**) is an Eclipse based open source technology platform used to create data visualizations and reports that can be embedded into rich client and web applications.

## Pentaho Reporting

**Pentaho Reporting** is a suite of Open Source tools that allows you to create pixel-perfect reports of your data in PDF, Excel, HTML, Text, Rich-Text-File, XML and CSV. These computer-generated reports easily refine data from various sources into a human readable form.

All the above solutions are available in both Open Source and Commercial editions.

# OLAP (Analytics) Platforms

SQL Server Analytics Services (SSAS), the de facto Analytics Platform for SQL Server, is an online analytical processing and data mining tool used by organizations to analyze and make sense of information possibly spread out across multiple databases, or in disparate tables or files

PostgreSQL does not have a built-in OLAP Server, but there are a host of Open Source and commercial OLAP solutions that you can investigate to replace your SSAS Reports and Cubes.

## Pentaho Mondrian

**Mondrian** is an Open Source Online Analytical Processing server (OLAP) solution that allows business users to analyze large and complex amounts of data in real-time.

## Apache Kylin

**Apache Kylin** is an open source distributed analytics engine designed to provide a SQL interface and multi-dimensional analysis on Hadoop and Alluxio supporting extremely large datasets. It was originally developed by eBay, and is now a project of the Apache Software Foundation

# Scheduling Options

SQL Server comes with SQL Agent that allows you to schedule Jobs that run SSIS packages, execute SQL Queries and Stored Procedures and execute OS tasks and programs.

PostgreSQL has **pgAgent**, a job scheduling agent for PostgreSQL databases, capable of running multi-step batch or shell scripts and SQL tasks on complex schedules.

## Cloud

Please note that certain Cloud based PostgreSQL offerings do not support plugins like pgAgent. In those cases, you may want to utilize the cloud specific schedulers like **AWS Batch**. The AWS Batch scheduler evaluates when, where, and how to run jobs that have been submitted to a job queue. Jobs run in approximately the order in which they are submitted if all dependencies on other jobs have been met.

## OS Schedulers

In the absence of any viable alternative, you may choose to use Task Scheduler in Windows and cronjobs in Linux to schedule recurring database activities like data load, monitoring, purging and cleanup activities, etc.

# Application/Services Migration

After the Database migration is completed, you would want to migrate your Applications, Services, Reporting systems, etc. that were pointing to your SQL Server Databases to point to your new PostgreSQL databases.

## Technology Stack / Components / Drivers (Java, .NET drivers)

PostgreSQL has support for ODBC and has native drivers for most programming languages and frameworks:

- .Net
- C (Native Library)
- C++
- Delphi
- Java (JDBC)
- JavaScript (Node.js)
- Perl
- PHP
- Python
- Tcl

## Java Applications

Connecting to PostgreSQL databases from your Java applications and services would primarily be through JDBC drivers. Checkout the **PostgreSQL JDBC** page for more details.

## .NET Applications

**Npgsql** is by far the most widely used open source .NET connector for PostgreSQL. It is a high-performance driver build on top of ADO.NET and provides an easy migration path by using similar object names as SQL Server native driver. For example, to change the connection object, replace `SqlConnection` with `NpgsqlConnection`.

## SQL Server (ADO.NET)

```
public DataTable ExecuteDataTable(string queryText) {
          using (SqlConnection connection = new
                        SqlConnection(connectionString)) {
        connection.Open();

        SqlCommand cm = new SqlCommand(queryText,
connection);
        SqlDataAdapter adapter = new SqlDataAdapter(cm);
        DataTable table = new DataTable();
        adapter.Fill(table);
        return table;
    }

}
```

## PostgreSQL (Npgsql)

```
public DataTable ExecuteDataTable(string queryText) {
          using (NpgsqlConnection connection = new
                    NpgsqlConnection(connectionString)) {
        connection.Open();

        NpgsqlCommand cm = new NpgsqlCommand(queryText,
connection);
        NpgsqlDataAdapter adapter = new
NpgsqlDataAdapter(cm);
        DataTable table = new DataTable();
        adapter.Fill(table);
        return table;
    }

}
```

For a complete list, check out the **drivers and interfaces** page.

# CHAPTER 4

# MAINTENANCE AND MONITORING

# Maintenance and Monitoring

SQL Server comes with built-in maintenance and monitoring tools like Cluster Manager, Profiler, etc. that can be used to manage your SQL Server Clusters, nodes and troubleshoot performance issues.

PostgreSQL on the other hand, does extensive logging and provides many ways of monitoring database performance and errors.

## The Statistics Collector

The Statistics Collector component of PostgreSQL collects and reports Server activity, table and index access information and row count, calls to functions and total time spent on function calls, etc.

## Statistics

PostgreSQL collects several statistics continuously and provided access to these statistics through built-in views which are called dynamic statistics views. You can query these views just like normal views to see various operational metrics related to your PostgreSQL database. The most important of the dynamic statistics views is the **pg_stat_activity** view.

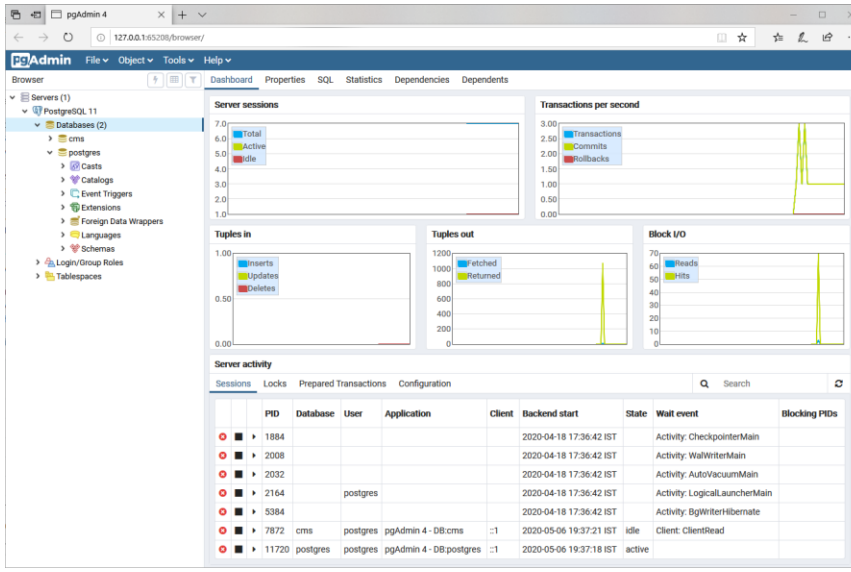
It is important to note that the statistics are not update instantaneously in real-time. Running transactions do not update statistics until they either commit or rollback. When

transactions start, they are provided with a snapshot of the statistics which does not update when the transaction is in progress, thus a running transaction always sees static statistics.

For an extensive list of all monitoring options, please check out **this** page.

## Client Tools

PostgreSQL's most popularly used client tool is **pgAdmin**. pgAdmin is an open source client tool that can connect to many different versions of PostgreSQL, both on-prem and Cloud instances. pgAdmin provides different database health states in its dashboard as graph and it can be used for basic monitoring and troubleshooting.




# Cloud Platforms

## AWS

If you are opting to migrate to AWS Aurora/RDS, it has built-in monitoring for its Aurora and RDS databases. AWS CloudWatch provides additional enhanced monitoring and comes with a cost. AWS also provides Enhanced Monitoring for Aurora databases and must be enabled in each database. Other Cloud providers like Azure have built-in monitoring tools as well to measure PostgreSQL database issues and performance metrics.

# SaaS Platforms

Software as a Service monitoring platforms like **NewRelic** and **DataDog** provide plugins for PostgreSQL monitoring and

automated dashboards for monitoring PostgreSQL databases for errors and performance issues. They have offerings for both Data Center and Cloud infrastructure.

# High Availability, Load Balancing, and Replication

A detailed discussion on the subject is beyond the scope of this concise eBook. Check out these pages for an in-depth discussion on the topics:

**Replication, Clustering, and Connection Pooling**

**High Availability, Load Balancing, and Replication**

**PostgreSQL Replication Solutions** (pdf)

# Environment Stabilization via Parallel Deployment

After your databases are migrated successfully, you would want to run both your SQL Server and PostgreSQL workloads in parallel for few days till you make sure that your new databases are stable and handling your production loads smoothly. The applications and services that write to your databases would have to write to both your SQL Server and PostgreSQL or they need to be kept in sync through replication/CDC solutions till the cut-over date. The applications/reporting systems that only read from your databases can read from either of these databases during the soak-in period and then permanently switch to your PostgreSQL databases after the cut-over date.

If your SQL Server database is an offline database and if it is okay for the database to be down for few hours, then you can just switch over to PostgreSQL database on a pre-determined cut-over date and save yourself the trouble of writing to multiple databases and setting up replications.

# Summary

In this eBook, we discussed migrating your entire SQL Server workloads to PostgreSQL, the different migrations options available and choosing the one that is right for you. We also discussed the comparative strengths and weaknesses of the both the platforms, and post migration considerations as well.

Migrating databases from one platform to another along with their workloads is a mammoth task and requires careful and meticulous planning to make it a success. This eBook touches upon all the topics and tasks that are required for a smooth migration.

# APPENDIX

# Links in this eBook

| Description | Link |
|---|---|
| Original Article on The Developer Space | "http://thedeveloperspace.com/migrating-your-sql-server-workloads-to-postgresql/ |
| EnterpriseDB Home | https://www.enterprisedb.com |
| DB-Engines Ranking | https://db-engines.com/en/ranking |
| Microsoft SQL Server Home | https://www.microsoft.com/en-in/sql-server |
| SQL Server Wikipedia | https://en.wikipedia.org/wiki/Microsoft_SQL_Server |
| DB-Engines SQL Server Page | https://db-engines.com/en/system/Microsoft+SQL+Server |
| PostgreSQL Home | https://www.postgresql.org |
| PostgreSQL Wikipedia | https://en.wikipedia.org/wiki/PostgreSQL |

| | |
|---|---|
| DB-Engines PostgreSQL Page | https://db-engines.com/en/system/PostgreSQL |
| PostgreSQL Feature Matrix | https://www.postgresql.org/about/featurematrix |
| DB-Engines SQL Server vs PostgreSQL | https://db-engines.com/en/system/Microsoft+SQL+Server%3BPostgreSQL |
| SQL Server - Locking and Row Versioning | https://.microsoft.com/en-us/library/ms187101.aspx |
| SQL Server 2017 Maximum Capacity Specifications | https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server?view=sql-server-2017&viewFallbackFrom=sql-server-previousversions |
| SQL Server 2014 Maximum Capacity Specifications | https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server?view=sql-server-2014 |
| PostgreSQL Data Types | https://www.tutorialspoint.com/postgresql/postgresql_data_types.htm |
| PostgreSQL Functions | https://www.postgresql.org/docs/11/functions.html |

| | |
|---|---|
| PostgreSQL AutoVacuum | https://www.postgresql.org/docs/11/runtime-config-autovacuum.html |
| RDS AutoVacuum Case Study | https://aws.amazon.com/blogs/database/a-case-study-of-tuning-autovacuum-in-amazon-rds-for-postgresql |
| PostgreSQL Vacuum | https://www.postgresql.org/docs/11/sql-vacuum.html |
| TCL Language | http://www.tcl.tk |
| PostgreSQL Generated Columns | http://thedeveloperspace.com/generated-columns-in-postgresql/ |
| pgLoader | https://github.com/dimitri/pgloader |
| Sqlserver2pgsql | https://github.com/dalibo/sqlserver2pgsql |
| Talend Open Studio | https://www.talend.com/products/data-integration/data-integration-open-studio |
| Pentaho Data Integration | https://www.hitachivantara.com/en-in/products/big-data-integration-analytics/pentaho-data-integration.html |
| Jasper Reports | https://community.jaspersoft.com/project/jasperreports-server |

| | |
|---|---|
| BIRT | http://www.eclipse.org/birt |
| Pentaho Reporting | https://community.hitachivantara.com/s/article/pentaho-reporting |
| Pentaho Mondrian | https://community.hitachivantara.com/s/article/mondrian |
| Apache Kylin | http://kylin.apache.org |
| pgAgent | https://www.pgadmin.org/docs/pgadmin4/development/pgagent.html |
| AWS Job Scheduler | https://docs.aws.amazon.com/batch/latest/userguide/job_scheduling.html |
| PostgreSQL Java Drivers | https://jdbc.postgresql.org/ |
| Npgsql | https://www.npgsql.org/ |
| PostgreSQL Drivers & Interfaces | https://www.postgresql.org/download/products/2-drivers-and-interfaces/ |
| PostgreSQL Monitoring | https://wiki.postgresql.org/wiki/Monitoring |
| Replication, Clustering, and | https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling |

| Connection Pooling | |
|---|---|
| High Availability, Load Balancing, and Replication | https://www.postgresql.org/docs/current/high-availability.html |
| PostgreSQL Replication Solutions (pdf) | http://momjian.us/main/writings/pgsql/replication.pdf |