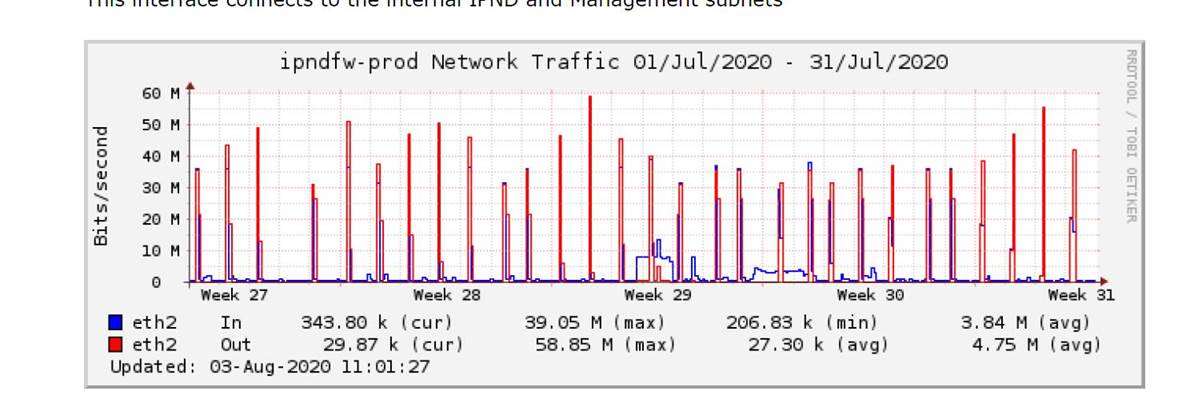
**We should consider several things while choosing database solutions:**

  ·        Read and write needs

**Processor Capacity As Is**

There is enough CPU capacity

**Network usage**



·        **Total storage requirements**

**Storage Usage requirements:**

The total available space in the 6180 is approximately 2.2 TB. Of that total space 1816 GB is allocated leaving approximately 415 GB of unallocated space. Of the allocated space 994 GB is used; all filesystems having adequate provision for growth.

Database tablespace usage is stable. All tablespaces with the exception of Upload Control are set to auto extend.

**·        Typical object size and the nature of access to these objects**

**·        Durability requirements**

**·        Latency requirements**

**Existing Hardware AS IS**

The primary IPND Systems are configured using 3 x Sun Sparc Enterprise T5440, 2 deployed in the production site and 1 at the ARP site. Storage is provided through 2 x StorEDGE 6180 arrays, 1 at the production site and 1 at the ARP site. Backups are to a StorEDGE SL48 tape library in the production site and a LTO4 tape drive in the ARP site.

**IPND system currently is facing the following constraints:**

1. Backup subsystem tape library slot/tape capacity

2. Management network bandwidth

2.1 Backup Subsystem The backup software – Symantec Netbackup 7.1 and Vault 7.1 imposes quite rigid controls on the use and expiration of tapes. This, in conjunction with the limited number of slots in the library and the current tape capacity means that on occasions when the volume of data that is written to tape exceeds about 900GB ( this value is estimated because the actual value is determined by the compressibility of the data ) the data is written to 2 tapes. The backup system already accommodates this for the daily backup policy local and remote tapes but not for the monthly backup policy. Consequently when monthly backups exceed 900GB there is an increased possibility of the backup failing.

2.2 Management Network Bandwidth Network communications between IPND production, ARP and the IPND management site (LT) are based on three way 1MB SHDSL links. Although the capacity of the link between LT and production is adequate for command line management of systems it is proving unsatisfactory when attempting to use the more sophisticated GUIs that several subsystems now rely on; for instance for Oracle database management, RSA authentication management, Disk Storage management etc.

2.3 Recommendations

Rec 1 The backup subsystem and policies are reviewed to determine options.

Rec 2 Management bandwidth requirements are reviewed to determine options.

Rec 3 There is adequate disk capacity. It would be a significant improvement to the value of the IPND is options regarding increasing the age of data retained on disk

were investigated.

**Existing Hardware config.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Loc** | **Units** | **Description** | **CPU** | **RAM (G)** | **Disk (G)** |
| A | 2 | Firewall Appliances | 8 | 8 | 60 |
| B | 2 | Firewall Appliances | 8 | 8 | 60 |
| A | 1 | Firewall management | 1 | 4 | 30 |
| B | 1 | Firewall management | 1 | 4 | 30 |
| A | 1 | Admin and management | 1 | 4 | 80 |
| B | 1 | Admin and management | 1 | 4 | 80 |
| A | 1 | Application - prod | 4 | 16 | 50 |
| B | 1 | Application - prod | 4 | 16 | 50 |
| A | 1 | Application - usertest | 4 | 16 | 50 |
| A | 1 | Application – test | 2 | 16 | 50 |
| A | 1 | Application – dev | 2 | 16 | 50 |
| A | 1 | Database prod | 4 | 32 | 500 |
| B | 1 | Database prod | 4 | 32 | 500 |
| A | 1 | Database usertest | 4 | 32 | 500 |
| A | 1 | Database test | 4 | 32 | 300 |
| B | 1 | Database dev | 4 | 32 | 300 |
| A | 1 | Shared Storage |  |  | 2500 |
| B | 1 | Shared Storage |  |  | 2500 |
| A | 1 | DataBase storage |  |  | 2000 |
| B | 1 | DataBase storage |  |  | 2000 |
| A | 18 | Backup instances |  |  |  |
| B | 18 | Backup instances |  |  |  |
| A | 1 | Backup Storage |  |  | 4500 |
| - | 16 | Redhat licences |  |  |  |

IIS and Web portal

A is prod , B is DR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Zone | Units | Description | CPU | RAM (G) | Disk (G) |
| A | 2 | Firewall Appliances | 8 | 8 | 60 |
| B | 1 | Firewall Appliances | 8 | 8 | 60 |
| A | 1 | Firewall management | 1 | 4 | 30 |
| B | 1 | Firewall management | 1 | 4 | 30 |
| A | 1 | Admin and management | 1 | 4 | 80 |
| B | 1 | Admin and management | 1 | 4 | 80 |
| A | 2 | VPN Servers | 1 | 4 | 30 |
| B | 1 | VPN Server | 1 | 4 | 30 |
| A | 2 | FTS Server prod | 2 | 4 | 50 |
| B | 1 | FTS Server prod | 2 | 4 | 50 |
| A | 1 | FTS Server usertest | 2 | 4 | 50 |
| A | 1 | FTS Server test | 2 | 4 | 50 |
| A | 1 | FTS Server dev | 2 | 4 | 50 |
| A | 2 | Portal WBS | 4 | 16 | 500 |
| B | 1 | Portal WBS | 4 | 16 | 500 |
| A | 2 | Portal APS | 4 | 16 | 500 |
| B | 1 | Portal APS | 4 | 16 | 500 |
| A | 2 | Database prod | 4 | 16 | 500 |
| B | 1 | Database prod | 4 | 16 | 500 |
| A | 1 | Database usertest | 4 | 16 | 500 |
| A | 1 | Database test | 4 | 16 | 300 |
| A | 1 | Database dev | 4 | 16 | 300 |
| A | 1 | Shared Storage |  |  | 2500 |
| B | 1 | Shared Storage |  |  | 2500 |
| A | 1 | DataBase storage |  |  | 2000 |
| B | 1 | DataBase storage |  |  | 2000 |
| A | 18 | Backup instances |  |  |  |
| B | 18 | Backup instances |  |  |  |
| A | 1 | Backup Storage |  |  | 4500 |
| - | 18 | Redhat licences |  |  |  |

·        Maximum concurrent users to support

**Number of Services in database as is**

There are currently about 92 million rows in the Service table comprising about 62 million connected and 30 million disconnected services. The graph also indicates the constant rate of growth over the life of the system of about 5% per annum.

**Projected Users**

No currently projected data is available.

**Actual Users**

Data Providers Current number of Data Providers = 132 \*

**Data Users**

Current number of Data Users (Including EWS, Elipse and IPNDe) = 11

**Current Application Throughput requirements**

**Upload Data Volume as is**

The amount of data uploaded onto the IPND has been fairly constant over the life of the system.

**Records processed per month as is**

This graph shows that apart from several peaks subsequent to the initial load that the usual volume of data processed is between 5 and 10 million rows per month, with a level trend over the last year or two. It is not envisaged that an increase in the number of Data Providers will alter the existing trend.

**Download Records per Month** As Is

Apart from the peaks that represent IPND bulk uploads from Data Providers the volume of IPND records generated per month is about 30 million. This has been steady since about 2009.

·        Nature of queries

**·        Required strength of integrity controls**

The Upload File Processor (UFP) works within the original operational requirements, i.e. 500 000 rows ( as per the original Technical Requirement s) in 2 hours. NB that this is difficult to evaluate because the system is now decoupled into 2 components. Also note that as a consequence of changes introduced with application release 6.5.1 , which facilitates synchronisation of data with the ARP site, throughput has decreased but is still within operational requirements.

In terms of the baseline 500,000 rows, this means that the UFP and DFG combined should be able to process this data in approximately 2 hours. However, given the data sampled (Shown in appendix B and C ) it is estimated that the processing would be more likely take 3 hours.

**·        IPND application availability requirements**

Availability The system will be available for transfer of upload files in the prescribed timeframes 99.5% of the time.

The system will be available for transfer of download files in the prescribed timeframes 99.5% of the time.

Upload error and download data files will be created by the prescribed time 99.5% of the time, where the prescribed time is the start of the file transfer window.

System may be available for transfer of upload and download files outside the prescribed timeframes, but use of this availability is not covered by this availability requirement.

Provide provision for a scheduled maintenance period during a nominated period.

The IPND Manager and IPND Users to be notified, two weeks in advance, of intention to utilize a scheduled maintenance period, with an indication of any expected impact.

Test automatic cut-over to secondary machine on a six monthly cycle.

System availability of 99.5% is defined to not include periods where the scheduled maintenance period is utilized.

**Processing times frames to be considered**

Data uploads and downloads to be scheduled by Data Users from 14:00 until 12:00 the following day, allowing a 22 hour file transfer window.

The maintenance window of 12:00 to 14:00 may be extended to 16:00. Except in exceptional circumstances, users will be given two weeks’ notice of any maintenance that may disrupt the file transfer window.

The system will normally process uploaded files as they arrive. Download files for IPND users will be produced and made available for download once an hour.

There remains a requirement that data processing of files received prior to 11pm to be completed by 1am.

Users should be encouraged to transfer files throughout the day. This will even out processing and enable Data Users to update their systems more frequently and reduce the dependency on a single daily data feed.

·        Backup and recovery needs

BACKUP/RECOVERY AND ARCHIVE The objectives of the system backups are: To ensure that all data stored on disk can be recovered in the event of a system disaster or failure, within a reasonable time frame. To ensure that all data which has been archived to tape and deleted from disk, is stored on another tape to allow recovery of that data in the event of the archive tape failing. To facilitate internal auditing and investigation. The objective of archiving is: To allow access to audit data which is too old to justify storing on disk.

**Challenges When Migrating from Oracle to PostgreSQL—and How to Overcome Them**

**Data types and schema conversion**

Some PostgreSQL data types are much easier to work with than their corresponding Oracle types.

However, there are some important differences to note. The Numeric field in PostgreSQL can be used to map any Number data types. But when it’s used for joins (such as for a foreign key), it is less performant than using an int or bigint.

The PostgreSQL Timestamp with time zone field is slightly different from the Oracle Timestamp with time zone field—it is actually equivalent to Oracle’s Timestamp with local time zone. These small differences can cause either performance issues or subtle application bugs that require thorough testing.

**Migrating data**  
After you convert your schema and tweak the tables to best meet your specific application, it’s time to move the data. For smaller (100 GB or less) databases, this process is fairly straightforward. Using [AWS Data Migration Service (AWS DMS)](https://aws.amazon.com/dms/) or a tool such as HVR, you can create data migration jobs that run on Amazon EC2 instances, connect to your local Oracle database, and pipe the data to an [Amazon RDS for PostgreSQL](https://aws.amazon.com/rds/postgresql/) instance. You will want to verify the data in the target database. Then run the migration a couple of times in dev, test, and finally, your production environment, and resolve any issues that arise.

**Larger data volumes**  
For larger data volumes—more than 1 TB, for example—a purely online migration of data might be too time consuming or take too much of your available bandwidth. In this situation, you should use an export, reload, and sync approach. In this approach, you export your largest tables. Then, you either compress and push them to [Amazon S3](https://aws.amazon.com/s3/) or use [AWS Snowball](https://aws.amazon.com/snowball/) to transport them to an AWS Region and load to Amazon S3. After the data is in Amazon S3, you can bulk load the data files to PostgreSQL.

Using date-based partitions makes it easier to choose and isolate the data that is moved offline. Unchanging historical data that has time stamp–based records can be exported up to a known point in time, with any data after that point migrated using AWS DMS. Or, you can use an Oracle log sequence number (LSN) or application-specific sequential ID as the cutoff point for the export and migration job.

**Architecture clean-up**  
A migration is a great time to clean up certain parts of your architecture and application. For example, if you store files (PDFs, images, etc.) in your database, it’s a great time to break them out into their own reliable Amazon S3 storage bucket. Reduce the size of the database and the time needed for backup and restore operations, and improve your application flexibility by being more flexible in how you work with files.

If you have static historical data, you might be able to either purge it completely if it’s never used by your application, or move it to a separate archive data store. This data store can be either in a lower performance (and less expensive) PostgreSQL database or perhaps in an Amazon S3-backed Amazon Athena or Apache Spark system that is suitable for infrequent queries on cold data.

And, if you’ve mixed online transaction processing (OLTP) and analytics-style data access, moving from a one-tool-for-everything Oracle setup to using a separate warehouse for reporting and analytics can improve both your application responsiveness and your analytics capabilities. There are options to create a dedicated Postgres-XL–based warehouse or use [Amazon Redshift](https://aws.amazon.com/redshift/) as a powerful managed warehouse.

In all, migrating your data takes planning and practice, but is a very solvable challenge.

**Migrating code**  
The most intensive effort during an Oracle-to-PostgreSQL migration is usually porting the code to work with PostgreSQL. This applies to the stored procedures, packages, and functions within the database and the application code that reads and writes to the database.

**Database code**  
PostgreSQL is similar to Oracle in many ways. The main procedural programming language, PL/pgSQL, is similar enough to PL/SQL that most database administrators (DBAs) and developers can easily learn the syntax. Automated tools such as the AWS Schema Conversion Tool (AWS SCT) or the open source Ora2Pg typically can automatically convert more than 70 percent of the database code to work correctly.

The automated conversion includes converting Oracle-specific functions to ANSI standard functions (for example, moving from nvl() to coalesce()), changing legacy syntax such as the plus sign (+) used for outer joins to standard outer join syntax, and adding aliases for subselects that Oracle considers optional, but that PostgreSQL requires. In these cases, the converted code is compatible with both Oracle and PostgreSQL. It could actually be merged into your current app code base and deployed on your existing database before you fully migrate.

The automated tools can also convert code that uses database-specific syntax, such as sequence handling with Oracle’s sequence.nextval vs. PostgreSQL’s nextval(sequence), or the slightly different syntax for executing dynamic SQL within a stored procedure. These changes are not compatible with Oracle and so are only used on your target PostgreSQL system.

The remaining code must be manually converted, either because the SQL and PL/SQL were too complex for the tool to parse and convert perfectly, or because there is not an exact one-for-one conversion that can be automatically applied. Many such cases are trivial for experienced developers and DBAs to convert when they understand both the PostgreSQL approach and the application logic involved.

**features: autonomous transactions, collect all, and BFILEs**  
There are some Oracle features that are either not supported by PostgreSQL or handled through an extension or workaround. For example, PostgreSQL does not directly support autonomous transactions that enable one stored procedure’s changes to be committed inside a larger transaction that is rolled back. The common workaround is to use a DBLink “remote” connection to the same database. You execute the function you want committed as a “remote” call that is treated as a separate connection—and a separate transaction.

Similarly, core PostgreSQL does not support externally organized tables or accessing files on the database file system directly. However, both of these are possible using a Foreign Data Wrapper (FDW).

Some common Oracle packages, such as DBMS\_OUTPUT, are supported by the open source orafce and AWS compatibility extensions that ease the conversion cost. Others, such as UTL\_FILE, UTL\_HTTP, and the SMTP package, could be supported via extensions. But you should review whether this is the right architectural solution, or if your application architecture should be updated as part of the conversion. You need to verify whether the extension you’re considering is [supported in the Amazon RDS-managed environment](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html#PostgreSQL.Concepts.General.FeaturesExtensions), or you might not be able to use Amazon RDS.

**Application code**  
PostgreSQL has broad support in programming languages, and there are drivers available for every mainstream development language, and many others as well. Depending on your application architecture and database connection layer, you might need only a small amount of change, or you might have significant porting changes.

For example, it’s fairly easy to convert a Java-based application that uses generic JDBC classes (not Oracle-specific classes) and always calls stored procedures with no SQL in the code, either dynamically built or hardcoded. If you use an object-relational mapping (ORM) such as Hibernate or JCA, it could be trivially easy to switch the dialect from Oracle to PostgreSQL. There are still little differences. For example, if you use partitioning, the required PostgreSQL triggers change the record count that is returned from an insert and fools Hibernate. So you need to add annotations to the Hibernate queries that update partitioned tables.

For these types of simple cases, you can even use AWS SCT to scan your Java or .NET code to find SQL statements and convert the SQL to PostgreSQL-compatible SQL—similar to the conversion done for PL/SQL.

However, the application code conversion is much more involved if you use embedded SQL such as Oracle’s Pro\*C, link to Oracle-specific libraries such as OCI or the Oracle JDBC classes, or dynamically build SQL based on application conditions. PostgreSQL has support for C/C++ with either embedded SQL (ECPG) or the libpq library. But these are not drop-in compatible with the Oracle equivalents.

Similarly, changing application code that dynamically builds SQL requires a strong understanding of the application logic and robust testing to ensure that the functionality works as expected. We have worked with customers to automate some aspects of these migrations, but it still requires focus from the application maintenance team.

**Transaction control and exception handling**

Every application must ensure proper transaction management and error handling—that’s how we keep the corner cases, run-time failures, and unexpected user input from creating bad data. PostgreSQL has robust transaction handling, supporting full ACID semantics and different isolation levels. PostgreSQL also handles run-time errors gracefully and gives reliable error codes and messages to calling code—PL/pgSQL or applications. But there are a couple of differences in the way PostgreSQL deals with these internally versus how Oracle behaves that require changes to application code or design.

**First,** PostgreSQL does not allow transaction control inside of PL/pgSQL—you cannot commit or roll back a transaction inside a stored procedure. The application that calls the stored procedure must perform the transaction management—starting and committing or rolling back. The stored procedure executes within that calling transaction context. Obviously, if your existing database code has transaction management in procedures, it must be modified.

**Second,** when a run-time exception has occurred in a transaction, you must roll back that transaction before you can execute any statement on the connection. You often see this when you find the following error message in your application log:

**ERROR: current transaction is aborted, commands ignored until end of transaction block.**  
This message indicates that an error occurred, the error was ignored, and another statement (SELECT, INSERT, EXECUTE, anything…) was executed Then the second statement failed because the transaction is already in an error state (aborted). When you see this message, carefully review your database calls and exception handling. Ensure that anywhere an error could occur (any database call), you check errors or have an exception handler set up and ROLLBACK (or ROLLBACK to a savepoint, or close the connection) before trying another database operation.

**Third**, for application logic and to address the above error, you must have exception handling. In PL/pgSQL, using a BEGIN…EXCEPTION…END block lets your code catch an error that occurs. This block automatically creates a savepoint before the block and rolls back to that new savepoint when the exception occurs. You can then determine what logic to execute based on whether there was an error. However, exception blocks, because they create a savepoint, are expensive. If you don’t need to catch an error, or if you are planning to simply raise the error back to the calling application, don’t have the exception block at all. Let the original error flow up to the application.

Similarly, Java, embedded SQL, and other languages have mechanisms for catching exceptions. Review the application to ensure that proper error handling is in place around database calls. If the application currently catches and ignores the exception, you must modify it to roll back the transaction before a new database call can be issued. If the application expects to keep the partial transaction changes before the exception, you might have to add savepoints to your application code and roll back to a savepoint to continue after an exception.

Note that just like the exception block in PL/pgSQL, adding savepoints does affect performance. So use them where needed, but not in every database call. For example, suppose that you are saving a header record along with child records, and you will roll back the entire transaction if you get an exception. In this case, you don’t need to create a savepoint after inserting the header record because if the child records fail, you roll back the insert.

**Finally**, you must map the error codes and exception types that your application expects from Oracle to PostgreSQL. Some error codes (such as the **100 no record found code**) are the same on both, but others are different. Depending on your programming language, if you are catching Oracle-specific JDBC exceptions, you must replace those specific exceptions with either generic cross-database exceptions or switch to PostgreSQL-specific exceptions.

Making sure that your applications handle transactions and errors correctly is a critical part of migrating and usually requires changes to the database and application code.

You also must realize that, although Oracle and PostgreSQL are both relational databases and support most of the same SQL syntax for creating tables and querying data, the internals are different. So they behave differently in some situations.

For example, PostgreSQL’s Multi version Concurrency Control (MVCC) is very different from Oracle’s rollback segments, even though they both provide the foundation for ACID transactions. Developers who are used to designing applications for one, or DBAs who are used to optimizing performance for one, might hit some nasty speed bumps if they use the same techniques on the other.

**Ref :** <https://aws.amazon.com/blogs/database/challenges-when-migrating-from-oracle-to-postgresql-and-how-to-overcome-them/>

**Why PostgreSQL to be considered?**

PostgreSQL is the one trending upwards to its inherited characteristics, that are more suited to today’s dynamic development requirements. Below is a comparison along 5 crucial parameters that cannot be ignored before choosing a solution and making the move.

1. **Value for Money - PostgreSQL**Oracle is a commercial solution that has pretty steep pricing options, with additional payments required for extra features. PostgreSQL easily clinches this comparison since acquisition, installation and support is completely free of charge.
2. **Support – PostgreSQL**Another win for the open-source solution. PostgreSQL has an extremely active community where patches, tweaks, updates and more can be found easily. Even answers to questions that arise during installation or upgrades are to be found easily with minimal delays. This is not the case with Oracle, where support costs money.  
     
   Large organizations that choose to implement PostgreSQL can also opt for paid support professionals, whose services tend to be cheaper than their Oracle counterparts.
3. **Functionality – Oracle**Oracle Database pulls one back thanks to decades of experience and high levels of development expertise. It not only provides more transactions per second than PostgreSQL, but also arguably provides higher levels of security. However, it should be noted that many of Oracle’s security features come at an added cost.  
     
   Oracle Database is secure and ensures that user data is not tampered with through prompt updates. Its better experience with various industries also gives it the upper hand.  
   PostgreSQL is no slouch in the functionality front. It offers three levels of transaction isolation: Read Committed, Repeatable Read and Serializable. Its immune to dirty reads. Requesting a Read Uncommitted transaction isolation level provides read committed instead. PostgreSQL supports full serializability via serializable snapshot isolation (SSI).
4. **Scalability – Draw**While both solutions are quite capable in this category, PostgreSQL possesses the advantage due to its open-source characteristics. Not only is it much lighter than Oracle, you also don’t have to fork out more cash to expand your infrastructure. PostgreSQL is completely capable of accommodating any volume of data.
5. **Compatibility – PostgreSQL**Oracle has a robust language in PL/SQL, however PostgreSQL allows you to write language handlers in multiple languages (Python, R, etc.) directly in the database.  
   PostgreSQL also clearly has the edge when it comes to compatibility with operating systems, which is extremely crucial in today’s diverse and complex development environments. FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix and Windows are all compatible with PostgreSQL , which is a big advantage.

**Existing database Environments to be considered for migration**

**Databases maintained in Production Environment**

**Production** Data Providers submit their changes to the IPND system, these are processed by the IPND Application which in turn add all data to an Oracle Database. The Oracle DB is hosted on a clustered system with shared disk. In the event of a failure with the primary system, the clustered systems will manage and maintain data availability. If a total failure occurs, the ARP system and the disaster recovery procedures will be invoked.

**Production Physical Standby** One (or more) physical standby databases are maintained. This is an optional extra level of redundancy. Production database backups occur from the physical standby located on cluster node

**User Test** A database used for testing by new Data Providers. This database resides on a global area on the 6180 storage area. It is expected that data provider tests would be with a limited set of test data – less than 100,000 rows. In the event of recovery of this database being required, Data Provider test data is not restorable and would need to be resubmitted. Rows created by a data provider may be deleted once the provider has completed their testing.

**Acceptance** A database used acceptance testing. This database will be backed up and restored as required. This database is usually situated in the Disaster recovery environment.

**Volume and Stress Test** A database used for performing volume and stress tests. This database is at least as large as the production database and is periodically refreshed from it. This database is usually situated in the Disaster recovery environment.

**Other** Other databases may be temporarily commissioned as required.