

The Pyrrho Database Management System

Malcolm Crowe, University of the West of Scotland
www.pyrrhodb.com



Version 7.0 (November 2020)

Contents

| | |
|--|----|
| 1. Introducing Pyrrho | 8 |
| 1.1 Features of Pyrrho..... | 8 |
| 1.2 Pyrrho of Elis..... | 8 |
| 1.4 How to read this manual | 9 |
| 1.5 About this version..... | 9 |
| 2. Obtaining Pyrrho..... | 11 |
| 2.1 Downloading the package..... | 11 |
| 2.2 System requirements..... | 11 |
| 2.3 Licensing and Copyright..... | 11 |
| 2.4 Importing existing data | 11 |
| 2.5 Converting existing database applications | 12 |
| 3. Installing and starting the server | 13 |
| 3.1 Command line options | 13 |
| 3.2 Server account | 14 |
| 3.3 Database folder | 14 |
| 3.4 Security considerations | 14 |
| 3.4.1 Sensitive data..... | 15 |
| 3.4.2 Mandatory access control | 15 |
| 3.5 Forensic investigation of a database | 17 |
| 3.6 Role-based Data Models | 17 |
| 3.7 Virtual Data Warehousing | 19 |
| 3.8 The Pyrrho REST service | 20 |
| 3.9 Localisation and Collations..... | 21 |
| 3.10 Pyrrho DBMS architecture | 21 |
| 4. Pyrrho client utilities..... | 23 |
| 4.1 The Pyrrho Connection library | 23 |
| 4.1.2 Localisation..... | 23 |
| 4.2 Installing the client utilities..... | 23 |
| 4.3 PyrrhoCmd..... | 23 |
| 4.3.1 Checking it works | 24 |
| 4.3.2 Accessing a server on another machine | 24 |
| 4.3.3 Connecting to databases on the server | 25 |
| 4.3.4 The SQL> prompt..... | 25 |
| 4.3.5 Multiline SQL statements | 25 |
| 4.3.6 Adding data and blobs to a table | 25 |
| 4.3.7 Retrieving data and blobs from the server..... | 26 |
| 4.3.8 Command Line synopsis | 26 |
| 4.3.9 Transactions and PyrrhoCmd | 26 |
| 4.4 PyrrhoSQL..... | 27 |
| 4.5 RESTClient..... | 27 |
| 4.6 The RestIfD service | 28 |
| 4.7 The Profile Viewer..... | 29 |
| 5. Database design and creation..... | 30 |
| 5.1 Creating a Database | 30 |
| 5.2 Creating database objects..... | 30 |
| 5.2.1 Pyrrho's data type system | 31 |
| 5.2.2 Indexes, Identity etc | 32 |
| 5.2.3 Row versions..... | 32 |
| 5.3 Altering tables..... | 33 |
| 5.4 Sharing a database with other users | 34 |
| 5.5 Roles | 34 |
| 5.6 Stored Procedures and Functions..... | 35 |
| 5.6.1 Table-valued functions..... | 35 |
| 5.6.2 Simple statements | 36 |
| 5.6.3 Decision Statements..... | 36 |
| 5.6.4 Iterative statements | 36 |
| 5.6.5 Condition handling statements..... | 36 |

| | |
|--|----|
| 5.6.6 Examples..... | 37 |
| 5.7 Structured Types | 38 |
| 5.8 Triggers..... | 38 |
| 5.9 Provenance and extended subtype semantics..... | 39 |
| 5.9.1 IRI references and subtypes | 39 |
| 5.9.2 Row and table subtypes..... | 39 |
| 5.10 Generated Columns..... | 40 |
| 6. Pyrrho application development | 42 |
| 6.1 Getting Started | 42 |
| 6.2 Opening and closing a connection | 42 |
| 6.3 The connection string..... | 43 |
| 6.4 REST and POCO | 44 |
| 6.5 DataReaders | 45 |
| 6.6 LINQ..... | 46 |
| 6.7 Using PHP | 47 |
| 6.8 Python..... | 48 |
| 6.8.1 DatabaseError | 48 |
| 6.8.2 (Date) | 49 |
| 6.8.3 DocArray..... | 49 |
| 6.8.4 Document..... | 49 |
| 6.8.5 DocumentException..... | 49 |
| 6.8.6 (ExcludeAttribute) | 49 |
| 6.8.7 (Field Attribute) | 49 |
| 6.8.8 (KeyAttribute)..... | 49 |
| 6.8.9 PyrrhoArray | 49 |
| 6.8.10 PyrrhoColumn..... | 49 |
| 6.8.11 PyrrhoCommand | 50 |
| 6.8.12 PyrrhoConnect | 50 |
| 6.8.13 PyrrhoDbType..... | 50 |
| 6.8.14 PyrrhoInterval | 51 |
| 6.8.15 (PyrrhoParameter)..... | 51 |
| 6.8.16 (PyrrhoParameterCollection) | 51 |
| 6.8.17 PyrrhoReader | 51 |
| 6.8.18 PyrrhoRow | 51 |
| 6.8.19 PyrrhoTable..... | 51 |
| 6.8.20 PyrrhoTransaction | 52 |
| 6.8.21 (SchemaAttribute)..... | 52 |
| 6.8.22 Versioned | 52 |
| 6.8.23 WebCtrl..... | 52 |
| 6.8.24 WebSvc | 52 |
| 6.8.25 WebSvr | 53 |
| 6.9 The Java Library | 53 |
| 6.9.1 CallableStatement | 54 |
| 6.9.2 Column..... | 54 |
| 6.9.3 Connection | 54 |
| 6.9.4 DBNull..... | 55 |
| 6.9.5 DataType..... | 55 |
| 6.9.6 DatabaseMetaData | 56 |
| 6.9.7 DatabaseException..... | 56 |
| 6.9.8 Date | 56 |
| 6.9.9 DocArray..... | 56 |
| 6.9.10 Document | 56 |
| 6.9.11 DocumentException..... | 57 |
| 6.9.12 DriverManager | 57 |
| 6.9.13 Exclude | 57 |
| 6.9.14 FieldType | 57 |
| 6.9.15 Interval | 57 |
| 6.9.16 Key..... | 57 |
| 6.9.17 NoResultException | 57 |
| 6.9.18 Numeric..... | 58 |

| | |
|--|-----------|
| 6.9.19 Parameter | 58 |
| 6.9.20 PreparedStatement | 58 |
| 6.9.21 Procedure | 58 |
| 6.9.22 ResultSet | 58 |
| 6.9.22 ResultSetMetaData..... | 59 |
| 6.9.23 SQLException..... | 59 |
| 6.9.24 SQLWarning | 60 |
| 6.9.25 Schema..... | 60 |
| 6.9.26 Statement..... | 60 |
| 6.9.27 Time | 60 |
| 6.9.28 TimeSpan | 60 |
| 6.9.29 Timestamp | 60 |
| 6.8.22 Versioned | 60 |
| 6.10 SWI-Prolog | 60 |
| 7. SQL Syntax for Pyrrho | 62 |
| 7.1 Statements..... | 62 |
| 7.2 Data Definition | 63 |
| 7.3 Access Control..... | 68 |
| 7.4 Type | 69 |
| 7.5 Data Manipulation | 70 |
| 7.6 Scalar Expressions | 71 |
| 7.7 Query Expressions | 73 |
| 7.8 Predicates..... | 75 |
| 7.9 SQL Functions | 76 |
| 7.10 SQL Statements | 77 |
| 7.11 XML Support..... | 79 |
| 7.12 Proposed simplification of the SQL2016 security model | 79 |
| General Rules..... | 79 |
| 8. Pyrrho Reference | 81 |
| 8.1 Diagnostics | 81 |
| 8.1.1 SQLSTATE..... | 81 |
| 8.1.2 Get Diagnostics | 90 |
| 8.2 Sys\$ table collection | 91 |
| 8.2.1 Sys\$Audit..... | 91 |
| 8.2.2 Sys\$AuditKey | 91 |
| 8.2.3 Sys\$Classification..... | 91 |
| 8.2.4 Sys\$ClassifiedColumnData..... | 91 |
| 8.2.5 Sys\$Enforcement | 92 |
| 8.2.6 Sys\$Role | 92 |
| 8.2.7 Sys\$RoleUser..... | 92 |
| 8.2.8 Sys\$ServerConfiguration | 92 |
| 8.2.9 Sys\$User | 92 |
| 8.3 Role\$ table collection | 92 |
| 8.3.1 Role\$Class | 92 |
| 8.3.2 Role\$Column | 93 |
| 8.3.3 Role\$ColumnCheck | 93 |
| 8.3.4 Role\$ColumnPrivilege..... | 93 |
| 8.3.5 Role\$Domain | 93 |
| 8.3.6 Role\$DomainCheck | 93 |
| 8.3.7 Role\$Index | 94 |
| 8.3.8 Role\$IndexKey | 94 |
| 8.3.9 Role\$Java..... | 94 |
| 8.3.10 Role\$Method..... | 94 |
| 8.3.11 Role\$Object | 95 |
| 8.3.12 Role\$Parameter..... | 95 |
| 8.3.13 Role\$PrimaryKey..... | 95 |
| 8.3.14 Role\$Privilege..... | 95 |
| 8.3.15 Role\$Procedure..... | 95 |
| 8.3.16 Role\$Python..... | 96 |
| 8.3.17 Role\$Subobject..... | 96 |

| | |
|--|-----|
| 8.3.18 Role\$Table | 96 |
| 8.3.19 Role\$TableCheck | 96 |
| 8.3.20 Role\$TablePeriod | 96 |
| 8.3.21 Role\$Trigger | 97 |
| 8.3.22 Role\$TriggerUpdateColumn | 97 |
| 8.3.23 Role\$Type | 97 |
| 8.3.24 Role\$View | 97 |
| 8.4 Log\$ table collection | 97 |
| 8.4.1 Log\$ | 98 |
| 8.4.2 Log\$Alter | 99 |
| 8.4.3 Log\$Change | 99 |
| 8.4.4 Log\$Check | 99 |
| 8.4.5 Log\$Classification | 99 |
| 8.4.6 Log\$Clearance | 99 |
| 8.4.7 Log\$Column | 99 |
| 8.4.8 Log\$DateType | 100 |
| 8.4.9 Log\$Delete | 100 |
| 8.4.10 Log\$Domain | 100 |
| 8.4.11 Log\$Drop | 100 |
| 8.4.12 Log\$Edit | 100 |
| 8.4.13 Log\$Enforcement | 101 |
| 8.4.14 Log\$Grant | 101 |
| 8.4.15 Log\$Index | 101 |
| 8.4.16 Log\$IndexKey | 101 |
| 8.4.17 Log\$Insert | 101 |
| 8.4.18 Log\$InsertField | 101 |
| 8.4.19 Log\$Metadata | 101 |
| 8.4.20 Log\$Modify | 102 |
| 8.4.21 Log\$Ordering | 102 |
| 8.4.22 Log\$Procedure | 102 |
| 8.4.23 Log\$Revoke | 102 |
| 8.4.24 Log\$Role | 102 |
| 8.4.25 Log\$Table | 102 |
| 8.4.26 Log\$TablePeriod | 103 |
| 8.4.27 Log\$Transaction | 103 |
| 8.4.28 Log\$TransactionParticipant | 103 |
| 8.4.29 Log\$Trigger | 103 |
| 8.4.30 Log\$TriggerUpdateColumn | 103 |
| 8.4.31 Log\$TriggeredAction | 103 |
| 8.4.32 Log\$Type | 104 |
| 8.4.33 Log\$TypeMethod | 104 |
| 8.4.34 Log\$Update | 104 |
| 8.4.35 Log\$User | 104 |
| 8.4.36 Log\$View | 104 |
| 8.5 Table and Cell Logs | 105 |
| 8.5.1 A Table Log | 105 |
| 8.5.2 A Cell Log | 105 |
| 8.6 Transaction Profiling | 105 |
| 8.6.1 Profile\$ | 106 |
| 8.6.2 Profile\$ReadConstraint | 106 |
| 8.6.3 Profile\$Record | 106 |
| 8.6.4 Profile\$RecordColumn | 106 |
| 8.6.5 Profile\$Table | 106 |
| 8.7 Pyrrho Class Library Reference | 107 |
| 8.7.1 DatabaseError | 107 |
| 8.7.2 Date | 108 |
| 8.7.3 DocArray | 108 |
| 8.7.4 Document | 108 |
| 8.7.5 DocumentException | 108 |
| 8.7.6 ExcludeAttribute | 108 |

| | |
|--|-----|
| 8.7.7 FieldAttribute | 108 |
| 8.7.8 KeyAttribute | 109 |
| 8.7.9 PyrrhoArray | 109 |
| 8.7.10 PyrrhoColumn | 109 |
| 8.7.11 PyrrhoCommand | 109 |
| 8.7.12 PyrrhoConnect | 110 |
| 8.7.13 PyrrhoDbType | 111 |
| 8.7.14 PyrrhoInterval | 111 |
| 8.7.15 PyrrhoParameter | 112 |
| 8.7.16 PyrrhoParameterCollection | 112 |
| 8.7.17 PyrrhoReader | 112 |
| 8.7.18 PyrrhoRow | 112 |
| 8.7.19 PyrrhoTable | 112 |
| 8.7.20 PyrrhoTransaction | 112 |
| 8.7.21 SchemaAttribute | 112 |
| 8.7.22 Versioned | 113 |
| 8.7.23 WebCtrl | 113 |
| 8.7.24 WebSvc | 113 |
| 8.7.25 WebSvr | 114 |
| 8.8 The Pyrrho protocol | 114 |
| 8.8.1 Low level-communication | 114 |
| 8.8.2 Sending the connection string | 115 |
| 8.8.3 Protocol details | 116 |
| 8.8.4 Schema | 117 |
| 8.8.5 Column | 117 |
| 8.8.6 Cell | 117 |
| 8.8.7 Type | 118 |
| 8.8.8 Exceptions | 118 |
| 9. Pyrrho Database File Format | 119 |
| 9.1 Data Formats | 119 |
| 9.1.1 Integer format | 119 |
| 9.1.2 Numeric and Real format | 120 |
| 9.1.3 Variant format | 120 |
| 9.1.4 Array and Multiset format | 120 |
| 9.1.5 Row and User Defined Type format | 120 |
| 9.1.6 Blob format | 120 |
| 9.1.7 Boolean format | 120 |
| 9.1.8 Char and XML format | 120 |
| 9.1.9 Date and TimeSpan formats | 120 |
| 9.1.10 Interval format | 120 |
| 9.2 Record formats | 120 |
| 9.2.1 DataType | 122 |
| 9.2.2 Drop Action | 123 |
| 9.2.3 Fields information | 123 |
| 9.2.4 Update information | 123 |
| 9.2.5 Index flags | 123 |
| 9.2.6 Method type | 123 |
| 9.2.7 Privilege flags | 123 |
| 9.2.8 Trigger type | 124 |
| 9.2.9 Ordering type | 124 |
| 9.2.10 Interval fields | 124 |
| 9.2.11 Metadata flags | 124 |
| 9.2.12 GenerationRule | 125 |
| 9.2.13 Mandatory Access Control Label | 125 |
| 10. Troubleshooting | 126 |
| 10.1 Destruction and restoration | 126 |
| 10.2 Hardware failure during commit | 126 |
| 10.3 Alternative names for a database file | 126 |
| 10.4 User identity and database migration | 127 |
| 10.5 API Dependency on database history | 127 |

| | |
|--------------------------------------|-----|
| 11. End User License Agreement | 128 |
| References | 129 |
| Index to Syntax | 130 |

1. Introducing Pyrrho

Pyrrho is a compact and efficient relational database management system for the .NET framework.

Pyrrho implements much of the ISO 9075 SQL standard and also includes other features including support for semantic web services, and data model integration techniques. Databases created by Pyrrho are platform independent, location-independent, and culture-independent.

Since version 4.0 all versions of Pyrrho are freeware: from version 7.0 the professional and open source editions are merged. There is an Embedded edition intended for the situation that a database is available to just one local application, so that a separate database server is not needed.

1.1 Features of Pyrrho

Pyrrho is a rigorously developed relational database management system that can run on small computers but can also scale up to large enterprise uses. It is built for .NET, which is available on Windows, and on Linux systems with Mono. For best results, the server's main memory should be at least eight times the size of the database. Instead of encouraging large single-database systems, Pyrrho supports integration of data from heterogeneous servers in a loose federation.

Pyrrho has strong transactions, designed for business uses. It is most suited to data that includes a regular stream of new information that is to be kept indefinitely, for example, customer data, orders or accounting transactions¹.

Pyrrho supports the SQL database language, largely compatible with the SQL2016 standard². It is stricter than SQL2016 in some areas: for example, integrity constraints can be deferred to the end of a transaction but cannot be disabled, and the only possible transaction isolation level is `SERIALIZABLE`. In Pyrrho the default is that data types are variable-length and independent of platform and locale. There are practical limits, e.g. integers can be up to 2040 bits, and data uids are limited to 60 bits. For division of non-integer quantities Pyrrho sets a default precision of 13 decimal digits, but higher precision is used if specified. If the specified precision of reals or actual values of integers are sufficiently small, hardware arithmetic is used.

The client-server configuration uses a robust TCP-based protocol for communication with clients. The client library is designed as a thread-safe version of the ADO.NET architecture, Pyrrho supplies its own ADO.NET-like classes such as `PyrrhoReader`, `PyrrhoCommand` etc.

Optimistic execution is used as this is more suitable for wide-area operations. Pyrrho supports role, user and timestamp recording for all changes to the database. Transaction log information, including the above details, is recorded permanently in the database file so that deleted or modified data can always be recovered if required. In fact, the physical database files consist exactly of the transaction log and an optional cryptographic endmarker; so that without the endmarker Pyrrho uses "append storage".

The implementation of Pyrrho is in the C# language. Because the database file is the transaction log, Pyrrho typically writes to the disk just once per transaction, and performs well in standard benchmark tests.

1.2 Pyrrho of Elis

This database management system is named after an ancient Greek philosopher, Pyrrho of Elis (360-272BC), who founded the school of Scepticism. We know of this school from writers such as Diogenes Laertius and Sextus Empiricus, and several books about Pyrrhonism (e.g. by Floridi) have recently appeared.

And their philosophy was called investigatory, from their investigating or seeking the truth on all sides.

(Diogenes Laertius p 405)

Pyrrho's approach was to support investigation rather than mere acceptance of dogmatic or oracular utterance.

¹ This is because the transaction log is persistent and contains the complete history of the database (see note later in this section 1.1).

² Throughout this manual, SQL2016 denotes the most recent full version of the SQL standard at the time of writing, including later updates of individual volumes of the standard.

Accordingly in this database management system, care is taken to preserve any supporting evidence for data that can be gathered automatically, such as the record of who entered the data, when and (if possible) why; and to maintain a complete record of subsequent alterations to the data on the same basis. The fact and circumstances of such data entry and maintenance provide some evidence for the truthfulness of the data, and, conversely, makes any unusual activity or data easier to investigate. This additional information is available, normally only to the database owner, via SQL queries to system tables, as described in Chapter 8.2 of this manual. It is of course possible to use such automatically-recorded data in databases and applications.

In other ways Pyrrho supports investigation. For example, in SQL2016, renaming of objects requires copying of its data to a new object. In Pyrrho, by contrast, tables and other database objects can be renamed, so that the history of their data can be preserved. Object naming is role-based (see section 3.6).

The logo on the front cover of this manual combines the ancient “Greek key” design, used traditionally in architecture, with the initial letters of Pyrrho, and suggests security in its interlocking elements.

1.4 How to read this manual

Each chapter begins with a “getting started” section, and most will have sections towards the end intended for developers or advanced users. The reader is advised to skip over the later sections of chapters on a first reading.

The typographical conventions are as follows: Courier New font is used to indicate computer input or output. Bold face type is used for input, and normal for output, and italic font to indicate items that vary depending on user choices, as in

```
PyrrhoCmd -h:host database
SQL> select * from table
```

The current version of the .NET framework on Linux requires the above command to be given as

```
mono PyrrhoCmd.exe -h:host database
```

Similar incantations are needed at present for every .NET executable under Linux. This will not be mentioned every time in this manual, which will generally give the short (Windows) version of commands. Some versions of Linux can be configured with add-ins so that the “mono” prefix is not required.

1.5 About this version

All databases developed under previous versions of Pyrrho should still work with the latest version of the server³. However, when versions change, client applications should be recompiled so that their version of PyrrhoLink matches the server. There is now just one version of the Pyrrho engine: PyrrhoSvr.exe: the former open source versions have been merged with these, and applications that formerly used Pyrrho’s embedded edition can simply spawn their own local server⁴.

A number of features that Pyrrho once offered have been removed over the years. These have included support for Microsoft technology such as DataAdapters and the Entity Framework, for Java Persistence, SPARQL, OWL, RDF and even Mongo. Some previous editions were linked for use in mobile phones and web servers and allowed multi-database connections.

Version 7 of Pyrrho is a major re-implementation of the database engine, and the architecture modifications are described in the SourceIntro document. However, there are some small changes to this manual for Pyrrho v7, of which the first two affect the SQL language:

- The qualified asterisk and all-columns options for SELECT are available.
- Queries that do not access table data can omit the FROM clause.
- There are some slight changes to the system tables (chapter 8).
- A simple kind of prepared statement mechanism has been added to the API (see 8.7.12).

³ The assumed process is one of migration, with an automatic compatibility mode for migrated databases. Databases created or modified with the latest server version generally cannot be used with previous versions.

⁴ PyrrhoSvr can run on the localhost without administrator privileges if network services and HTTP are not required.

- In Windows, database files no longer need a file extension such as .osp or .pfl.

2. Obtaining Pyrrho

Pyrrho is available as a free and very small download for the .NET framework. Later sections of this chapter discuss issues associated with moving an existing database to Pyrrho.

2.1 Downloading the package

The source and binary code of Pyrrho is available from <https://pyrrhodbms.uws.ac.uk> in a single download⁵. Provided the .NET framework (mono for Linux) has been installed, it is possible to extract all of the files in the distribution to a single folder and start to use Pyrrho in this folder without making any system or registry changes. Pyrrho targets the latest version of .NET, and will upgrade to .NET 5.0 in November 2020.

You are allowed to view and test the code, and redistribute any of the files available on the Pyrrho website in their entirety. With suitable acknowledgement, you may embed the dlls or re-use any of the source code in any application. Any uses other than those described here requires a license from the University of the West of Scotland (see below).

2.2 System requirements

The .NET Framework version 4.0 or greater (available from www.microsoft.com for Windows or www.go-mono.com for Linux) is required. Database files are machine-independent and can be transferred between Windows and Linux or between different machine architectures, provided only that the .NET framework or Mono is installed first⁶.

PyrrhoSvr.exe itself is currently just over 1MB, and a minimum of 12MB of memory is required for the server process. However if the database holds xMB of data then at least 8xMB of main memory is recommended.

2.3 Licensing and Copyright

Pyrrho is intellectual property of Malcolm Crowe and the University of the West of Scotland, United Kingdom. The associated documentation and source code, where available, are copyright of Malcolm Crowe and the University of the West of Scotland. Your use of this intellectual property is governed by a standard end-user license agreement, which permits the uses described above without charges. All other use requires a license from the University of the West of Scotland.

Pyrrho depends on the .NET class libraries, which are royalty-free. Pyrrho conforms to the extent described herein to the SQL2016 standard, which is available from the standards bodies (ISO and national bodies).

2.4 Importing existing data

When importing tables from an existing database, it is good to take the opportunity for some minor redesign. For example, additional integrity constraints can be added, or data types can be simplified, for example by relaxing field size constraints. Keywords that imply such sizes, e.g. DOUBLE PRECISION, BIGINT etc are not supported in Pyrrho, which provides maximum precision by default. National character sets are deprecated since they make data locale-specific: universal character sets are used by default.

A more important area for attention is Pyrrho's security model, described fully in chapter 5. This offers an opportunity for improving the security of the business process. For simplicity during migration, the current user should initially use the server's account, as this will generally allow all desired operations to be performed with system privileges.

The first thing to note is that Pyrrho expects the operating system to handle user authentication so that there is no way for a user to pretend to be someone else: a custom encryption of the connection string is used to ensure this. There is an implicit business requirement to know which staff took the responsibility

⁵ At the time of writing, and Pyrrho v7 (alpha) is available at <https://github.com/MalcolmCrowe/ShareableDataStructures> .

⁶ Databases that contain user identities are obviously less portable: see section 10.4.

for data changes (corresponding to initials in former paper-based systems), and Pyrrho's approach is that it is undesirable for the database management system to force anonymity on such operations by disguising the staff responsible behind a faked-up application identity.

This means that users of the database should be identified and granted permissions (normally to use one or more database roles). Where the number of authorised staff is large, mechanisms for authorising new users can be automated. It is useful to use the role mechanism to simplify the granting of groups of permissions to the users.

Existing users and roles can be imported from the existing database: assuming users are identified in the existing database by their login identities. Where applications have been given user identities in the legacy system, this should generally be replaced by roles. Ideally each business process should have a role to enable associated database changes to be tracked. Each connection to Pyrrho is for a role, and this can enable a good record of the reasons for changes to data.

2.5 Converting existing database applications

Stored procedures and view definitions will need to be converted in general since Pyrrho uses the SQL2016 convention whereby identifiers are converted to upper case (not case-sensitive) unless they are enclosed in double quotes. Double-quoted identifiers can include layout and special characters. The use of square brackets instead of double quotes is not supported. Stored procedures must conform to the syntax specified in "SQL2016 – Persistent stored modules", and are detailed in Chapter 7.

Pyrrho supports the SQL language for coding stored procedures, and a simple version of the ADO.NET application programming interface described in Chapter 6: this allows SQL statements to be used as parameters. Pyrrho v7 has a prepared statement API, described in section 8.7.12. Other ways of embedding SQL into program coding are not supported.

The biggest conceptual hurdle in developing applications for Pyrrho is the use of optimistic transactions. It is very important for programmers to accept this approach as a fact of life, explained in the following paragraphs, and not try to imitate a locking model.

All good database architectures today support the ACID properties of transactions (atomicity, consistency, isolation and durability). Database products that use pessimistic locking (such as SQL Server or Oracle) acquire these locks on behalf of transactions by default, and it is not usually necessary for an application to deal with these issues directly. In a pessimistic locking product, transactions can be delayed (blocked) while waiting for the required locks to become available.

A transaction can fail because it conflicts with another transaction. For example, with pessimistic locking, the server may detect that two (or more) transactions have become deadlocked, that is, all of the transactions in the group is waiting for a lock that is held by another transaction in the group. In these circumstances, the server will abort one of the transactions, and reclaim its locks, so that other transactions in the group can proceed.

With pessimistic locking, if a transaction reaches its commit point, the commit will generally succeed. If it does not complete, it retains locks on database resources until it is rolled back. With SQL Server, for example, once a transaction T begins, it acquires locks on data that it accesses. If it updates any data, it acquires an exclusive lock on the data. Until T commits or is rolled back, no other transaction can access any data written by T or make any change to data read by T.

With optimistic locking, the first sign of failure may well be when the transaction tries to commit. A transaction will fail if it tries to make a change that conflicts with a change made by another transaction and if any data it has read has been changed. Except for syntax errors, any exception will abort the current transaction, unless the exception occurs inside a stored procedure that handles the signal.

In the classic transaction example of withdrawing money from a bank account, a transaction for making a transfer might include an SQL statement of the form "update myaccount set balance=balance-100" or "update myaccount set balance=3456". Writing SQL statements in the first form makes them apparently easier to restart, but the point being made here is that it should be the client application's responsibility to decide if the statements should simply be replayed on restart. The server should not simply make assumptions about the business logic of the transaction. Pyrrho transaction checking includes checking that data read by the transaction has not been changed by another transaction.

3. Installing and starting the server

The server `PyrrhoSvr.exe` is normally placed in the folder that will also contain the database files. Then `PyrrhoSvr` can be started from the command line, by the user who owns this folder. It is a good idea to run the server in a command window, because occasionally this window is used for diagnostic output.

After you start the server, it echoes the command line arguments for you to confirm startup (with the Enter key). If there are no arguments, you should then get confirmation that Pyrrho has started its services:

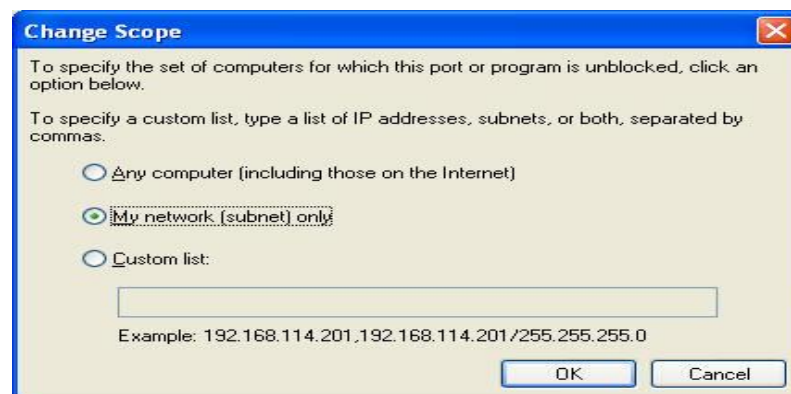


```

Command Prompt - PyrrhoSvr
E:\DATA>PyrrhoSvr
Enter to start up
Pyrrho DBMS (c) 2020 Malcolm Crowe and University of the West of Scotland
7.0 alpha (24 September 2020) www.pyrrhodb.com https://pyrrhodb.uws.ac.uk
PyrrhoDBMS protocol on ::1:5433

```

If Windows announces that it is blocking this program as a precaution, you will need to click the “Unblock” button on this security dialogue if you want to use the server. However, you should configure your firewall to make this service local to your subnet or local machine. The following dialogue box is from Windows XP:



See detailed instructions for Windows Firewall at <http://www.pyrrhodb.com/firewall.htm> .

In Windows 10 there are generally options such as Unblock or Show more to allow full operation of the software.

Under Linux, the command is **mono PyrrhoSvr.exe** .

You can stop the server by closing the window, since all committed transactions are already saved to persistent storage.

3.1 Command line options

The command line syntax is as follows:

```

PyrrhoSvr [-d:path] [-h:host] [-p:port] [-r:port] [+s[:port]]
[+S[:port]] [-M:port] [-R] [-E] [-H] [-T] [-V]

```

The `-h` and `-p` arguments are used to set the TCP host name and port number to something other than `::1` and `5433`⁷ respectively. This can be a useful and simple security precaution, as any client access must specify the same host address. The host argument should be a valid IP address (IPv4 or IPv6), not a

⁷ Port 5433 belongs to PyrrhoDBMS: see <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt> (accessed on 25 September 2020)

computer name (see section 3.4)⁸. The `-d` flag can be used to specify the server's database folder, if the server is placed in another location.

The `+s` and `+S` flags are for starting Pyrrho's HTTP service (see section 3.8). On Windows 7 systems and later, if you get Access denied, you can either run the program as administrator, or you can fix the http url reservations. To do this open a command prompt as administrator and issue the following commands (with your full user name where shown):

```
netsh http add urlacl http://127.0.0.1:8180/ user=DOMAIN\user
netsh http add urlacl https://127.0.0.1:8133/ user=DOMAIN\user
```

If you get other error messages, try using different ports.

Other flags are for instructional use and troubleshooting. The `-T` flag (Tutorial mode) is useful for demonstrating the steps taken by the server for distributed transactions and is less useful in the default situation where this feature is disabled. The `-E` flag can be used to display the per-command execution strategy after optimisation, and the `-V` flag can be used to display the syntax transformations applied to support renaming of database objects. The `-H` flag gives some feedback on the number of rows returned by HTTP requests in the RESTView system.

3.2 Server account

PyrrhoSvr.exe, the folder that contains it, and all the database files in this folder are normally owned by the same user, called the server account in the following notes. Note that the logical "database owner" is different – as described in this section.

The server account can always be used to create new databases. Other users who can access the server over the network can generally create databases (but see section 5.1), and naturally become the owner of any databases they create.

From version 7, if the client account matches the server account, the database will not initially contain user or role identities and can be accessed locally by the server account. The first user to be defined in the database becomes the owner, who then can access the database (e.g. over the network). This facility is useful in an educational context where a tutor wishes to create a database for students to copy to their own servers.

In enterprise contexts it is good practice under Linux for the server account to be a server identity such as `S_PYRRHO`, ie. a user identity created on the system, whose only system privileges are to be able to create, delete, read and write files in the server folder, and provide a TCP service on the Pyrrho port (see section 3.1). Things should be set up so that PyrrhoSvr.exe runs under this account, and no other account should have access to the database folder.

The server operates its own security policies (controlled in the usual SQL way by GRANT and REVOKE) on who is allowed to create and access database files. On Windows the client library uses the Windows.Security package to identify the client user ID (Windows login name) and construct an encrypted connection string to pass this to the server.

3.3 Database folder

By default, the server will create databases in a folder specified on server startup. You can inspect the database folder from time to time to check everything is in order. A database file path can be used if the server account is able to create and access the given path.

Normal file copying utilities can be used for the database: for example, the server account can copy a database created on another machine into its folder. There is one file per database which is the transaction log. Database files are all owned by the server account⁹.

3.4 Security considerations

Pyrrho is a TCP server, and the Internet is generally not a secure place. The Pyrrho DBMS server should be configured behind a firewall, and then accessed from within the firewall by web servers and possibly

⁸ The corresponding client-side argument can be a computer name (see chapters 4-8).

⁹ The server account can be changed provided file and server accounts continue to match.

local users. This precaution guards against denial-of-service and other attacks. Further instructions for firewall configuration are given at the very start of this chapter.

Within such a firewall, the client-server usage of Pyrrho as described in this booklet should conform to the following levels of security.

1. The security of the database file itself. Naturally, access to the database folder (section 3.3) should be limited to the server and operations staff, and strong password policies should be in place.

To protect against loss, copies of this file should be taken periodically and placed in a secure location. It is good practice to compare successive copies of the database: the database should always match the backup copy over the entire length of the latter. These features facilitate the creation of very secure systems.

2. The security of communication with the server. For all editions of Pyrrho, the connection string is encrypted using a custom encryption technique¹⁰. In a secure environment, access to the ports would be limited to authorised users, and the port numbers could be changed periodically.

3. The security of user identity for each transaction. The client library obtains the user identity from the operating system and encrypts it in the connection string for secure transmission to the server. Web applications should be configured so that the remote user's identity is correctly passed through using headers (anonymous access should be discouraged).

Within the database, all objects have owners, initially the user that created them (the definer). There are two predefined roles for a database: the default role, with the same name as the database, initially with all privileges, and the guest (public) role, initially allowed to use the default role. The normal SQL grant/revoke mechanisms can be used to modify these permissions (see also section 3.6 and 7.12).

See also section 5.1 on the question of permissions for users to create new databases. (This is not really a question of database security.)

3.4.1 Sensitive data

Inspired by the EU's General Data Protection Regulations, Pyrrho now supports the concept of sensitive data, for which any access is auditable. Columns, domains and types can be declared SENSITIVE¹¹. Sensitive values are not assignment-compatible with anything that is not sensitive, and there is a sensitive property inherited by any object that contains a sensitive data type. This means for example that the sum of sensitive data is still sensitive. The transaction log will contain a record of every access to sensitive values (apart from by the database owner), even if the transaction is rolled back. Auditing uses the Sys\$Audit system table (see section 8.3.1-2).

3.4.2 Mandatory access control

From December 2018 the DBMS also offers a simulation of Bell-LaPadula security based on clearance and classification levels D to A: the database owner is the security administrator (see section 7). The support is quite extensive, so this section includes some sample discussion. Some aspects of the Bell-LaPadula system are found in current DBMS: essentially the idea that rows of a table can have hidden multi-level security labels that control who can access the rows (and different rows in a table can have different labels).

The access control system is based on the concept of security levels, which are conventionally labelled D to A. Level D is the default and corresponds to no access control beyond the permissions described in the above sections. In the US Department of Defense Orange Book, Levels B and C have subdivisions based on the level of auditing available: since Pyrrho always audits levels above D, its levels C and B roughly equate to levels C2 and B3. Level A requires mathematical proof, which would probably be possible, but is not further discussed here. In addition a security label can contain two lists of identifiers here called groups and references, that are visible only to the security administrator (SA), for the purpose of fine-tuning the authorisations of individual users in individual tables.

¹⁰ After connection, further client-server communication is not encrypted by Pyrrho. The use of transport-layer security or alternative ports should be considered if security is an issue.

¹¹ SENSITIVE is a reserved word in SQL that normally applies to cursor sensitivity. The usage in Pyrrho described here is quite different, and the keyword comes at the end of a type clause (see section 7.4).

A user can be assigned a range of levels¹² called *clearance*, and tables and data records in the database can be assigned a level called *classification*. Initially all users have clearance level D (to D). As mentioned above, both clearance and classification can have lists of groups and references (see syntax below). The clearance and classification labels include the level and two sets of identifiers called here groups and references.

The database owner plays the role of security administrator SA for all objects and users of the database. The database owner has special privileges: to consult all system tables including logs, to access and modify the clearance and classification of users and tables and data records, and to specify the enforcement of these rules for tables in the database. By default, all operations on a table are enforced, but these can be limited to some combination of read/insert/update/delete.

The access rules for users other than the database owner are as follows (where the levels are ordered so that D is the lowest and A the highest). Subject to the normal SQL permissions and the enforcement policy

- A user with clearance x can access data with classification y iff $x \geq y$
- A user with clearance x can change or create data only with classification x

In addition, the list of references in the user's clearance must include all the references mentioned in the object's classification (if any); and the list of groups in the user's clearance must include at least one of the groups mentioned in the object's classification (if any). The second bullet point above means for example that some users will be able to see objects they are not allowed to modify. If a user inserts a new record in a table where insert is subject to enforcement, the new record will have a classification with the user's minimum level, the subset of the user's groups that are in the table's classification, and all the table's references (which must be a subset of the user's references).

The database owner (as security administrator) is exempt from these access rules. The database owner can specify the classification label for a new table or record. By default a new row will have the same classification as the table that receives it. When called directly or indirectly by the SA, triggers and stored procedures follow the usual (definer's) rules. The SA can also determine for each table whether to apply the access rules just for some combination of read, update, insert and delete operations (by default they are applied for all operations).

The SA can use syntax for level and enforcement descriptors: (as usual [] indicate optional, {} a sequence).

Level = LEVEL id ['-id] [GROUPS {id}] [REFERENCES {id}] .

Enforcement = SCOPE [READ] [UPDATE] [INSERT] [DELETE]

where the level id is one of the letters D to A.

The SA can add Level and Enforcement to a CREATE or ALTER for tables, specify Level in an INSERT statement or when defining columns, and use SECURITY as a pseudo column in SELECT, UPDATE and DELETE statements.

The SA can assign a clearance level to a user with the following extension to the GRANT statement:

GRANT Level TO user_id

where the user id normally requires to be enclosed in double quotes. The clearance level takes effect immediately on commit, but because of Pyrrho's approach to transaction isolation ongoing transactions will not be affected.

Where a user is unable to access some data because of classification, such data is silently excluded from any direct or indirect computation by that user. If specifically requested information is thus hidden, the requestor will be told that the objects are undefined or that the data is not found. Other exceptions raised by the operation of these rules contain only the information "access denied" (e.g. if a user has been prevented from updating something they have successfully accessed).

There are several system tables that allow the SA to monitor the operation of the above mechanisms. Actions by the SA are visible in the Log\$ table and there are separate tables (Log\$Clearance,

¹² A range of levels as a user clearance means that the user is free to read material at a high level and trusted to create at a lower level of security (the minimum they can access), and they can update an object whose classification is in their range (its classification does not change).

Log\$Classify and Log\$Enforcement) that allow SQL access to details of the direct and indirect actions taken by the SA to alter clearance or classification. The current status of all clearances, classified rows, classified columns, and enforcement is available to the SA in the Sys\$Clearance, Sys\$Classification, Sys\$ClassifiedColumnData and Sys\$Enforcement table, respectively, where such status is different from the default.

3.5 Forensic investigation of a database

Pyrrho supports two kinds of investigation of a database.

First, full log tables are maintained. These are accessible to the owner of the database. The log files allow tracing back to discover the full history of any object: when it was created, what changes to it were made, and when it was dropped. In each case, full transaction details are recorded: user, role and timestamp. Since objects can be renamed, logs use numeric identifiers to refer to objects in the database. Full details of the log tables are given in chapter 8. Using these tables, it is always possible to obtain details of when and by whom entries were made in the database. The system log refers to columns and tables by their uniquely identifying number rather than by name.

One extension to SQL2016 syntax which assists with forensic investigation is the pseudo-table ROWS(n) where n is the “Pos” attribute of the table concerned in “Sys\$Table” (see section 8.3). For example, suppose we want a complete history of all insert, update and delete operations on table BOOK. We first lookup BOOK in Role\$Table:

```
select "Pos" from "Role$Table" where "Name"='BOOK'
```

If this yields 149, then the required history is

```
select * from rows(149)
```

These can of course be combined:

```
select * from rows((select "Pos" from "Role$Table" where "Name"='BOOK'))
```

```

Command Prompt - pyrrhocmd ab

SQL> select "Pos" from "Role$Table" where "Name"='BOOK'
---|
|Pos|
---|
|149|
---|

SQL> select * from rows(149)
---|-----|-----|-----|-----|
|Pos|Action|DefPos|Transaction|Timestamp|
---|-----|-----|-----|-----|
|365|Insert|365|347|04/10/2020 11:19:16|
|430|Insert|430|412|04/10/2020 11:19:40|
|483|Insert|483|465|04/10/2020 11:20:13|
|540|Update|483|522|06/10/2020 10:41:07|
|605|Delete|605|587|06/10/2020 10:41:30|
---|-----|-----|-----|-----|

SQL> select * from rows((select "Pos" from "Role$Table" where "Name"='BOOK'))
---|-----|-----|-----|-----|
|Pos|Action|DefPos|Transaction|Timestamp|
---|-----|-----|-----|-----|
|365|Insert|365|347|04/10/2020 11:19:16|
|430|Insert|430|412|04/10/2020 11:19:40|
|483|Insert|483|465|04/10/2020 11:20:13|
|540|Update|483|522|06/10/2020 10:41:07|
|605|Delete|605|587|06/10/2020 10:41:30|
---|-----|-----|-----|-----|

```

The second set of parentheses is needed in SQL2016 here to force a scalar subquery. The Log\$ table gives a semi-readable account of all transactions:, and Log\$RecordField enables programmatic access to the data values of Insert and Update records.. Most of the System and log tables have a column called “Pos” which gives the defining position of the relevant entry.¹³

```
SQL> table "Log$"
```

| Pos | Desc | Type | Affects |
|-----|---|--------------|---------|
| 5 | PTransaction for 4 Role=-291 User=-292 Time=10/04/2020 11:17:22 | PTransaction | 5 |
| 23 | PTable AUTHOR | PTable | 23 |
| 34 | Domain INTEGER | PDomain | 34 |
| 48 | PColumn3 AID for 23(0)[34] | PColumn3 | 48 |
| 71 | PIndex AUTHOR on 23(48) PrimaryKey | PIndex | 71 |
| 92 | Domain CHAR | PDomain | 92 |
| 105 | PColumn3 ANAME for 23(1)[92] | PColumn3 | 105 |
| 131 | PTransaction for 6 Role=-291 User=-292 Time=10/04/2020 11:18:07 | PTransaction | 131 |
| 149 | PTable BOOK | PTable | 149 |
| 159 | PColumn3 BID for 149(0)[34] | PColumn3 | 159 |
| 184 | PIndex BOOK on 149(159) PrimaryKey | PIndex | 184 |
| 206 | PColumn3 AUTH for 149(1)[34] | PColumn3 | 206 |
| 233 | PIndex2 on 149(206) ForeignKey refers to [71] | PIndex2 | 233 |
| 252 | PColumn3 TITLE for 149(2)[92] | PColumn3 | 252 |
| 280 | PTransaction for 2 Role=-291 User=-292 Time=10/04/2020 11:18:34 | PTransaction | 280 |
| 298 | Record 298[23]: 48=1,105=Dickens | Record | 298 |
| 323 | Record 323[23]: 48=2,105=Conrad | Record | 323 |
| 347 | PTransaction for 1 Role=-291 User=-292 Time=10/04/2020 11:19:16 | PTransaction | 347 |
| 365 | Record 365[149]: 159=10,206=1,252=A Tale of Two Cities | Record | 365 |
| 412 | PTransaction for 1 Role=-291 User=-292 Time=10/04/2020 11:19:40 | PTransaction | 412 |
| 430 | Record 430[149]: 159=11,206=2,252=Nostromo | Record | 430 |
| 465 | PTransaction for 1 Role=-291 User=-292 Time=10/04/2020 11:20:13 | PTransaction | 465 |
| 483 | Record 483[149]: 159=12,206=1,252=Domby & Son | Record | 483 |
| 522 | PTransaction for 1 Role=-291 User=-292 Time=10/06/2020 10:41:07 | PTransaction | 522 |
| 540 | Update 483[149]: 159=12,206=1,252=Domby and Son Prev:483 | Update | 483 |
| 587 | PTransaction for 1 Role=-291 User=-292 Time=10/06/2020 10:41:30 | PTransaction | 587 |
| 605 | Delete Record 365[149] | Delete1 | 365 |

```
SQL>
```

The normal way for ownership of a Pyrrho database to be changed is for the database owner to invoke the Pyrrho-specific GRANT OWNER statement. This is implemented as part of the normal database service, and it is good practice to ensure that owners of database objects (see section 7.13) are user identities that are still available in the operating system.

3.6 Role-based Data Models

At any time, a database connection in Pyrrho has a user id and a role. On Windows systems, the user is obtained from Windows, and the default role has the same name as the database. Another role that the user is allowed to use can be specified in the connection string, or specified by the SET ROLE statement. Pyrrho allows database objects to be renamed or altered by holders of the appropriate permissions: but from Pyrrho 4.5 such renaming and alteration applies to the current role, so that a database object can have different names in different roles.¹⁴

By default, all roles in a Pyrrho database have a default data model based on the base tables, their columns, and using foreign keys as navigable properties. Composite keys use the list notation for values e.g. (3,4) and the name is the reserved word key, which can be suffixed by the property name of the key component. The default data model can be modified on a per-role basis to provide more user-friendly entity and

¹³ “Pos” and many other columns in the system tables have the integer subtype Position, which is specially handled in Pyrrho. See sec 8.1.3.

¹⁴ In Pyrrho versions 4.5 to 6.3, this mechanism was implemented by modifying source SQL contained in view, trigger and procedure definitions to contain defining positions instead of object names before storing the definition in the database. This behaviour was detectable in system tables such as Log\$. In version 7 and later, the source SQL is stored unchanged. For reasons of compatibility, databases created by previous versions will continue to use the database format of the older version, even for new objects.

column names, and user-friendly descriptions of these entities and properties. Tables and columns can be flagged as entities and attributes as desired.

For example, roles could be defined for users in different countries, using entity names, property names and descriptions appropriate to the language of the country, giving access to localised columns or views. The localisation of columns is facilitated by the Pyrrho-specific UPDATE clause for generated columns which can perform lookups or casts behind the scenes. These defined views or generated columns could even have specific data types targeting specific roles, since they impose no overhead unless they are explicitly used.

Roles that are granted usage of an object will not see any subsequent name changes applied in the parent role, but the role administrator can define new names. Stored procedures, view definitions, generation rules etc use the definer's permissions for execution.

Apart from object names, only the owner of an object can modify objects. This includes changes to object constraints and triggers, and inevitably such modifications can disrupt the use of the object by other roles, procedures etc. References in code in other roles can introduce restrictions on dropping of objects, but as usual, cascades override restrictions, and in Pyrrho, revoking privileges always causes a cascade. Granting select on a table must include at least one non-null column. Granting insert privileges for a role must include any non-null columns that do not have default values, and cannot include generated columns.

Metadata is an added feature in Pyrrho. Role administrators can modify object metadata as viewed from their role, and this is useful primarily for data output over HTTP. However, the iri associated with a database object can only be changed by its owner.

3.7 Virtual Data Warehousing

Normally, data warehousing involves creating central data repositories (using extract-transform-load technologies) to enable analytic processing of a combined data set. There are several situations where this is undesirable, for example where the resulting data protection responsibility at the central repository is excessive, where the data is volatile and it becomes expensive to maintain all of the centrally-held data in real time, or where it is better to leave the data at its sources where the responsibility lies. With database technology, a View (if defined but not materialised) allows access to data defined in other places. The virtual data warehouse concept exploits this notion, and endeavours to avoid the central accumulation of data. Pyrrho uses HTTP to collect data from the remote DBMS using a simple REST interface, and so the resulting technology here is called RESTView.

Thus, with RESTView, a Pyrrho database allows definition of views where the data is held on remote DBMS(s), and is accessible via SQL statements sent over HTTP with Json responses. Pyrrho itself provides such an HTTP service (see the next section) and the distribution includes suitable interface servers (RestIf, see sec 4.6) to provide such a service for remote MySQL and SqlServer DBMS.

The HTTP access provides the user/password combinations set up for this purpose within MySQL by the owners of contributor databases. In the use cases considered here, where a query Q references a RESTView V , we assume that (a) materialising V by Extract-transform-load is undesirable for some legal reason, and (b) we know nothing of the internal details of contributor databases. A single remote select statement defines each RESTView: the agreement with a contributor does not provide any complex protocols, so that for any given Q , we want at most one query to any contributor, compatible with the permissions granted to us by the contributor, namely grant select on the RESTView columns.

Crucially, though, for any given Q , we want to minimise the volume D of data transferred. We can consider how much data Q needs to compute its results, and we rewrite the query to keep D as low as possible. Obviously many such queries (such as the obvious select * from V) would need all of the data. At the other extreme, if Q only refers to local data (no RESTViews) D is always zero, so that all of this analysis is specific to the RESTView technology.

Pyrrho has a set of query-rewriting rules that aim to reduce D by recursive analysis of Q and the views and tables it references. As the later sections of this document explain, some of these rules can be very simple, such as filtering by rows or columns of V , while others involve performing some aggregations remotely (extreme cases such as select count(*) from V needs only one row to be returned). In particular, we will study the interactions between grouped aggregations and joins. The analysis will in general be recursive, since views may be defined using aggregations and joins of other views and local tables.

Any given Q might not be susceptible to such a reduction, or at least we may find that none of our rules help, so that a possible outcome of any stage in the analysis might be to decide not to make further

changes. Since this is Pyrrho, its immutable data structures can retain previous successful stages of query rewriting, if the next stage in the recursion is unable to make further progress.

There are two types of RESTView syntax (see section 7.2): corresponding to whether the view has one single contributor or multiple remote databases, as we will now see.

```
ViewDefinition = [ViewSpec] AS (QueryExpression | GET [USING Table_id]) {Metadata} .
```

The QueryExpression option here is the normal syntax for defining a view. The REST options both contain the GET keyword. The simplest kind of RESTView is defined as GET from a url defined in the Metadata. The types of the columns need to be specified in a slightly extended ViewSpec syntax (see sec 7.2). If there are multiple remote databases, the GET USING table_id option is available. The rows of this table describe the remote contributions: the last column provides a url, and data in the other columns (if any) is simply copied into the view. There are simple examples of this mechanism in the Pyrrho blog and website.

Depending on how the remote contributions are defined, RESTViews may be updatable, and may support insert and delete operations. However, these options require caution as it is impossible to guarantee correct transactional behaviour.

3.8 The Pyrrho REST service

Clients can use a RESTful interface provided by the PyrrhoConnect class as described in section 8.8.8 and 8.3.4. In addition, if the HTTP namespace has been configured on the server machine, the use of the command line +s and +S flags (sec 3.1), will cause Pyrrho to set up a REST service on ports http 8180 and https 8133, using Basic authentication. (You can supply your own server certificate for transport layer security and/or specify different ports.) In order to allow a user to access Pyrrho using basic authentication, the database owner must grant the PASSWORD privilege to the user. If the GRANT PASSWORD does not specify a password to use, the password will be set from the credentials of the next transacted HTTP request for this user.

Pyrrho supplies ETag¹⁵ information with responses, and one or more of these can be submitted in an If-Match header for conditional HTTP processing. Using this approach ACID behaviour can be guaranteed for a sequence of HTTP requests where all except the last are GETs. (As described below a set of changes can be made by posting an SqlStatement to the Role.)

The URL syntax for this service is as follows:

```
proto://[host[:port/]]database{/Selector}/{/Processing}
```

Selector matches¹⁶:

```
[table ]Table_id
[procedure ]Procedure_id [ '(' Parameters ') ' ]
[where ]Column_id=string
[select ]Column_id{,Column_id}
[key ]string
```

Appending another selector is used to restrict a list of data to match a given primary key value or named column values, or to navigate to another list by following a foreign key, or supply the current result as the parameters of a named procedure, function, or method.

Processing matches:

```
distinct [Column_id{, Column_id}]
ascending Column_id{, Column_id}
descending Column_id{, Column_id}
skip Int_string
count Int_string
```

The Http/https Accept and Content-Type headers control the formatting used. At present the supported formats are JSON (application/json), XML (text/xml), HTML (text/html, only for responses) and SQL

¹⁵ See RFC 7232.

¹⁶ The optional keywords here are less restrictive than might appear: In this syntax views and tables can be used interchangeably, so that the keyword **table** if present may be followed by a view. Similarly, the keyword **procedure** if present may be followed by a function call.

(text/plain). The Pyrrho distribution includes a REST client which makes it easier to use PUT, POST and DELETE. A URL can be used to GET a single item, a list of rows or single items, PUT an update to a list of items, POST one or more new rows for a table, or DELETE a list of rows. Thus GET and POST are very different operations: for example, POST does not even return data. All tables referenced by selectors must have primary keys. See section 4.5.

For the string-form selector (**key**), since the parser knows the datatype of the table's key, it is quite flexible about the format. If the key has several components, they should be separated by commas, and in that case it is easiest to single-quote any components that are strings.

For example with an obvious data model, GET `http://Sales/Orders/1234` returns a single row from the Orders table, GET `http://Sales/Orders/Total>50.0/OrderDate/distinct` returns a list of dates when large orders were placed, GET `http://Sales/Orders/OrderDate,Total` returns just the dates and totals, GET `http://Sales/Orders/1234/of OrderItem` returns a list of rows from the OrderItem table, and GET `http://Sales/Orders/1234/CUST/Customer/NAME` returns the name of the customer who placed order 1234. The response will contain a list of rows: if HTML has been requested it will display as a table (or a chart if specified by the Metadata flags, sec 7.2). Using HTML will also localise the output for dates etc for the client.

PUT `http://Sales/Orders/1234/DeliveryDate` with text/plain content of `((date'2011-07-20'))` will update the DeliveryDate cell in a row of the Orders table. PUT content consists of an array of rows, whose type must match the rowset returned by the URL. If the array has more than one row, commas can be used as separators. JSON format is also supported. XML format can also be used, which should match the data format returned by the URL.

POST `http://Sales/Orders` will create one or more new rows in the Orders table. In Pyrrho an integer primary key can be left unspecified. In SQL (text/plain) format, column names can be included in the row format, e.g. `(NAME:'Fred','DoB':date'2007-10-22')`: if no names are provided, all columns are expected. Remember that the REST service is case-sensitive for database object names. JSON can be used with the obvious format. XML format can also be used, in which case column values for the new row can be supplied either as attributes or child nodes irrespective of the data model. A mime type of text/csv has been added to facilitate import from Excel spreadsheets.

If no Selector or Processing components are provided, the target is the Role itself. For this target a POST request can consist of a single SqlStatement (for example a CompoundStatement), and GET returns a set of C# class definitions for POCO use (see sec 6.4).

Pyrrho's HTTP service complies with RFC 7232 and returns ETags with every GET response.

See also sections 4.5 and 4.6.

3.9 Localisation and Collations

Pyrrho's database files are intended to be locale-neutral: they use universal time and UTF-8 encoding with standard case-sensitive collation. Localisation and regional settings are applied in the API library (PyrrhoLink.dll or OSPLink.dll) and by default use the regional settings of the client (see chapter 4). This design makes it easier for databases to be accessed from or copied to different locales, and is consistent with the locale-neutral SQL language.

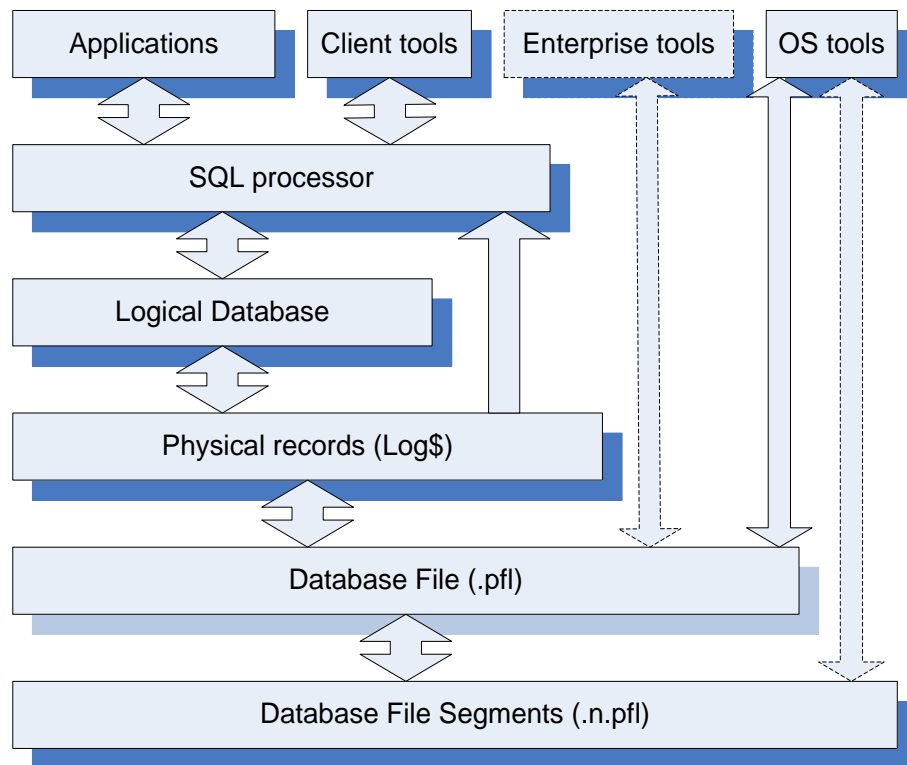
Pyrrho also supports most localisation facilities available in the SQL standard. For example, it uses the character set names as specified in SQL2016. Specifying a character set restricts the values that can be used, not the format of those values. By default, the UCS character set is used. By default, the UNICODE collation is used, and all collation names supported by the .NET framework are supported by Pyrrho. CHAR uses standard culture-independent Unicode. NCHAR is no longer supported and is silently converted to CHAR. UCS_BINARY is supported.

The SQL2016 standard specifies a locale-neutral interface language to the server, notably for dates and times.

In addition, views and updatable generated columns provide opportunities for localisation, which can be targeted by defining roles for specific locales.

3.10 Pyrrho DBMS architecture

The structure of the Pyrrho DBMS is shown in the drawing below



File segments are used for databases larger than 4GB.
Enterprise tools provide facilities for secure backup and mobile checkpoints

4. Pyrrho client utilities

There are three client utilities at present: a traditional command-line interpreter PyrrhoCmd, PyrrhoStudio, and a Windows client called PyrrhoSQL. As with all Pyrrho clients, the PyrrhoLink.dll assembly is also required. We discuss these first. The distribution also contains a REST client and a transaction profiling utility.

PyrrhoStudio is a version of PyrrhoCmd that embeds the database engine, for convenience when developing the database for an application based on EmbeddedPyrrho.dll. Databases developed with these clients do not contain the user identity.

4.1 The Pyrrho Connection library

PyrrhoLink.dll (or the Java package org.pyrrhodb.*) is used by any application that wishes to use the Pyrrho DBMS. This library includes support for client applications. The simplest possible approach is simply to place PyrrhoLink.dll in the same folder as the application that is using it.

In Chapter 6 we will see that PyrrhoLink.dll (or OSPLink.dll) is also needed to be at hand when compiling applications.

Since version 5.4 thread-safety is enforced in client-side programming. Connections can be shared among threads. But there can be at most one transaction or command active in a connection, and transactions, commands and readers cannot be shared between threads.

4.1.2 Localisation

In the current version, PyrrhoLink supplies error messages in English. Localisation to other languages was provided in previous editions and the mechanism to do this is still available in the code.

Locale-independent data from the database, such as dates and times, can be rendered by PyrrhoLink.dll according to the regional settings on the client machine. The database may be in a different country or timezone from the client.

However, SQL2016 itself is invariant (details of data formats are given in section 7). Thus the following behaviour is correct for a client machine in the UK:

```

SQL> insert into a(a2) values(date '2004-10-31')
SQL> table a
+----+-----+-----+-----+
|A1|A2|A3|A4|
+----+-----+-----+-----+
|1|25/12/1872|00:00:00| |
|2|31/10/2004|00:00:00| |
+----+-----+-----+-----+
SQL>

```

4.2 Installing the client utilities

The distribution currently contains PyrrhoCmd.exe, and PyrrhoSQL.exe, and the PyrrhoLink module PyrrhoLink.dll. These can be placed anywhere in the file system so long as the dll is in the same folder as the executable.

Since the client executables are so small (currently 140KB including the DLL) it is generally easier to copy them where they are required rather than using load-paths or registry entries.

It is usually convenient for database administration to install them on the server (in addition to client machines if any), but the client utility do not have to be on the server machine. If the server is not on 127.0.0.1 the **-h:** command line option can be used to specify a different host (e.g. **-h:fred** , or **-h:192.168.1.3**).

4.3 PyrrhoCmd

PyrrhoCmd is a console application for simple interaction with the Pyrrho server. Basically it allows SQL statements to be issued at the command prompt and displays the results of SELECT statements in

a simple form. Insert, update, and delete operations will generally cause a response indicating the number of rows affected¹⁷.

It has some additional features. It supports upload and download of blobs (binary large objects) through use of the escape character ~. It also supports the sort of multi-database connection described in section 2.7. See section 4.1 for locale issues.

4.3.1 Checking it works

On the same machine as the server, open a command window and use `cd` to navigate to the same folder as the client executable.

PyrrhoCmd

```
SQL> table "Sys$Table"
```

In SQL2016 `table id` is the same as `select * from id` for base tables and system tables.

```
C:\PyrrhoDB\OSP\OSP>pyrrhocmd
SQL> table "Sys$Table"
+-----+-----+-----+-----+-----+-----+-----+-----+
|Pos|Name|Columns|Rows|Triggers|CheckConstraints|References|RowIri|
+-----+-----+-----+-----+-----+-----+-----+-----+
|1|Sys$Table|1|1|1| | | |
+-----+-----+-----+-----+-----+-----+-----+-----+
SQL>
```

PyrrhoCmd will respond with the list of ables in the current database. The default database Temp is created if necessary by the command processor. To create or use another database, specify it on the command line. The above response from the server merely gives information about the tables in the database that are accessible from the current role (you are the database owner in this case, but it contains no tables). If you look in the folder: you will sce a file called Temp (it was not there before).

You can use control-C, or close the window, to exit from PyrrhoCmd.

Note that database names in Pyrrho are case sensitive. (Windows is relatively careless.)

4.3.2 Accessing a server on another machine

If the client is running on a different machine from the server, you will need to specify the host in the command line, as in:

```
PyrrhoCmd -h:address
```

Normally, PyrrhoCmd is used to connect to a particular database, specified as an argument in the command line. If no argument is supplied, then as indicated above, the Temp database is used.

If the database is new, and the client account is not the same as the server account, a default role is created with the same name as the database, the client account is granted use of it, and becomes the database owner. If the client account is the same as the server account, no role or user information is initially placed in the database, allowing roles and users to be defined later. This makes it easier to create databases for students. In both cases, the first role to be created becomes the default role for the database, and the first user to be granted this role becomes the database owner, and these can access any database objects created up to that point.

¹⁷ For operations involving table constraints that specify cascade or other side effects, the response will be simply OK.

4.3.3 Connecting to databases on the server

For example, if there is a database called Book, use

PyrrhoCmd Book

to connect to it. Note that case is significant in database names (since these are parts of actual file names). If more than one database name is given on the command line, a connection is established that opens a list of databases in the order given. See section 2.7.

4.3.4 The SQL> prompt

PyrrhoCmd is normally used interactively. At the SQL> prompt you can give a single SQL statement. There is no need to add a semicolon at the end. There is no maximum line length either, so if the command wraps around in PyrrhoCmd's window this is okay.

```
SQL> set role ranking
```

Be careful not to use the return key in the middle of an SQL statement as the end of line is interpreted by PyrrhoCmd as EOF for the SQL statement. If you want to use multiline SQL statements, see section 4.3.5.

At the SQL command prompt, instead of giving an SQL statement, you can specify a command file using *@filename*. Command files are ordinary text files containing an SQL statement on each line.

4.3.5 Multiline SQL statements

If wraparound annoys you, then you can enclose multi-line SQL statements in [] . [and] must then enclose the input, i.e. be the first and last non-blank characters in the input.

```
SQL> [ create table directors ( id int primary key,
> surname char,
> firstname char, pic blob ) ]
```

Note that continuation lines are prompted for with > . It is okay to enclose a one-line statement in [] .

Note that Pyrrho creates variable length data fields if the length information is missing, as here. This seems strange at first: a field defined as CHAR is actually a string.

4.3.6 Adding data and blobs to a table

Binary data is actually stored inside the database table, and in SQL such data is inserted using hex encoding. But PyrrhoCmd supports a special syntax that uses a filename as a value:

```
SQL> [ insert into directors (id, surname, firstname) values (1,
'Spielberg', 'Steven', ~spielberg.gif) ]
```

The above example shows how PyrrhoCmd allows the syntax *~source* as an alternative to the SQL2016 binary large object syntax X'474946...' . PyrrhoCmd searches for the file in the current folder, and embeds the data into the SQL statement before the statement is sent to the server.

As this behaviour may not be what users expect, the first time Pyrrho uploads or downloads a blob, a message is written to the console, e.g.:

Note: the contents of *source* is being copied as a blob to the server
source can be enclosed in single or double quotes, and may be a URL, i.e. *~source* can be *~"http://something"*.

A textfile containing rows for a table can similarly be added using a command such as

```
insert into directors values ~rowsfile
```

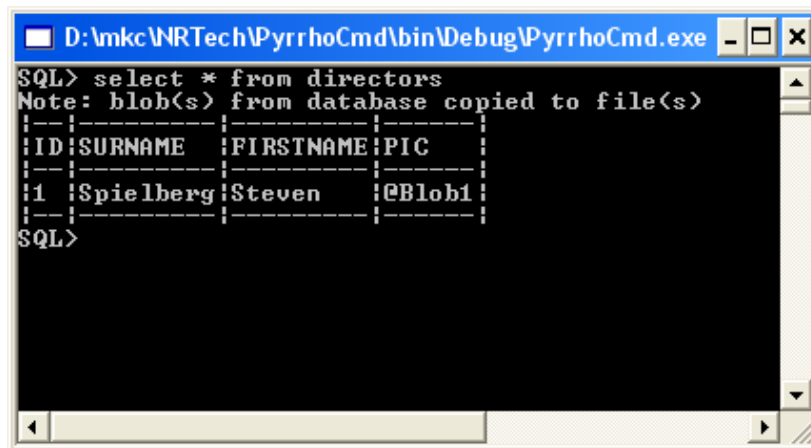
Simple data can be provided in a csv or similar file. The first line containing column headings and exposed spaces in the file are ignored. Data items in the given file are separated by exposed commas or tabs. Rows are parenthesized groups (optionally separated by commas), or provided without parentheses but separated by exposed semicolons or newlines. Characters such as commas etc are not considered to

be separators if they are within a quoted string or a structure enclosed in braces, parentheses, brackets, or pointy brackets.

4.3.7 Retrieving data and blobs from the server

Data is retrieved from the database using TABLE or SELECT statements, as indicated in 4.2.1.

If data returned from the server includes blobs, by default PyrrhoCmd puts these into files with new names of form `Blobnn`. Again PyrrhoCmd will alert the user to this process on the first occasion (unless `-s` flag has been set, see section 4.3.8, or the above message has been shown). To prevent downloads, use the `-b` flag, see section 4.3.8.



Blobs retrieved to the client side by this method end up in PyrrhoCmd's working directory (which is usually different from the database folder). To view them it is usually necessary to change the file extension, e.g. to `Blob1.gif`.

For ways to retrieve data and blobs using an application, see Chapter 6.

4.3.8 Command Line synopsis

When starting up PyrrhoCmd, the following command line arguments are supported:

| | |
|---------------------------|---|
| <code>database ...</code> | One or more database names on the server. The default is Temp. See section 2.7. |
| <code>-h:hostname</code> | Contact a server on another machine. The default is 127.0.0.1 |
| <code>-p:nnnn</code> | Contact the server listening on this port number. The default is 5433 |
| <code>-s</code> | Silent: suppress warnings about uploads and downloads |
| <code>-e:command</code> | Use the given command instead of taking console input. (Then the SQL> prompt is not used.) |
| <code>-f:file</code> | Take SQL statements from the given file instead of from the console. |
| <code>-c:locale</code> | Specify a language for the user interface, overriding .NET defaults. Localised versions of the error messages will be used if available. See section 4.1.2. |
| <code>-b</code> | No downloads of Blobs |
| <code>-v</code> | Show version and readCheck information for each row of data |
| <code>-?</code> | Show this information and exit. |

Whether the command prompt (console) window is able to display the localised output will depend on system installation details that are outside Pyrrho's control. Localisation is more effective with Windows Forms or Web Forms applications.

4.3.9 Transactions and PyrrhoCmd

Transactions in Pyrrho are mandatory, and are always serializable. By default, each command is committed immediately unless an error occurs. Alternatively, you can start an explicit transaction at the SQL> prompt:

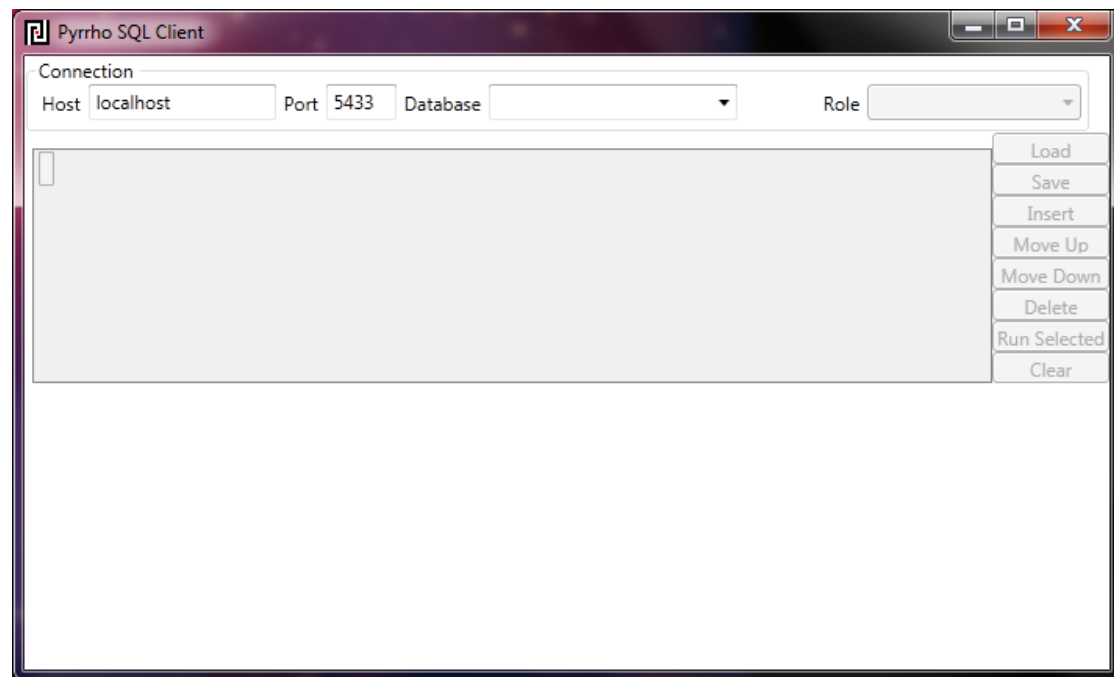
```
SQL> begin transaction
```

Then the command line prompt changes to SQL-T> to remind you that a transaction is in progress. This will continue until you issue a **rollback** or **commit** command at the SQL-T> prompt. If an error is reported by the database engine during an explicit transaction, you may get an additional message saying that the transaction has been rolled back followed by a normal SQL> prompt, or another SQL-T> prompt as an invitation to try to continue the transaction by means of another SQL command.

This continue behaviour is similar to the support offered by SQL's CONTINUE handler. The PyrrhoCmd client examines the TRANSACTION_ACTIVE diagnostic after an exception to see if the transaction can continue.

4.4 PyrrhoSQL

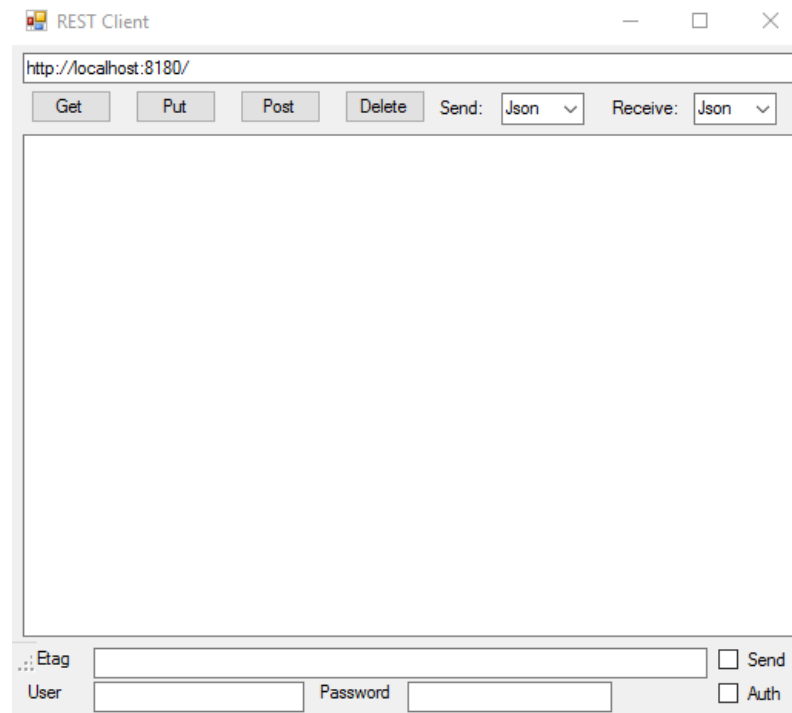
PyrrhoSQL (OSPSQL in the Open Source edition) is a more modern shell interface, based on Windows Presentation Foundation:



It allows selection of a database and role, and scripts can be created, loaded, modified and saved. Lines of SQL scripts are executed one at a time.

4.5 RESTClient

This Windows Forms program is useful for using the REST interface described in section 3.7. It is not Pyrrho-specific and uses Windows authentication to the server:



It offers a choice of send and receive formats (SQL, XML and HTML). It is important to remember the role must be specified in addition to the database name, and URLs are case-sensitive.

The drop-down lists offer alternative formats for request and response: Json, XML, SQL, and String,

If an ETag is returned by the service, it is displayed. The Send checkbox is used to send the contents of the ETag box as an If-Match condition with the HTTP request.

The Auth checkbox is used to supply the given User and Password as credentials to the service.

4.6 The RestIfD service

RestIfD is a simple web server based on TAWQT.com's AWebSvr architecture. It provides a simple SQL and Json interface to a MySQL database, and is intended to run on the same server as a running MySQL instance. Start it up and leave it running: it reports the URL it is listening on, normally <http://localhost:8078/>. Since source code is provided, it is a simple matter to add controllers for other DBMS to this service.

RESTIfD expects credentials using the normal HTTP Authentication header. These credentials are supplied directly to MySQL on each request. ETag and If-Match headers are supported.

The service offers the following interfaces.

| | |
|---|---|
| GET http://localhost:8078/ | This returns a list of databases in Json format, as reported by SHOW DATABASES for the given credentials. |
| GET http://localhost:8078/db | This returns a list of tables in database db, as reported by SHOW TABLES for the given credentials. |
| POST http://localhost:8078/db | The posted data should be plain text comprising one or more SQL statements (ending with semicolons). These are executed as a single transaction by MySQL. |
| GET http://localhost:8078/db/tb | This returns all of the rows of table tb in database db in Json format, and an ETag. |
| POST http://localhost:8078/db/tb | The posted data should be a single row for the given table in JSon format. |
| PUT http://localhost:8078/db/tb | For this request tb should have a primary key. The posted data is used to update a single row matvhing the key supplied in the row. |
| GET http://localhost:8078/db/tb/w | As above but w is a where condition (a string, or a Json document), used to filter the returned rows. |

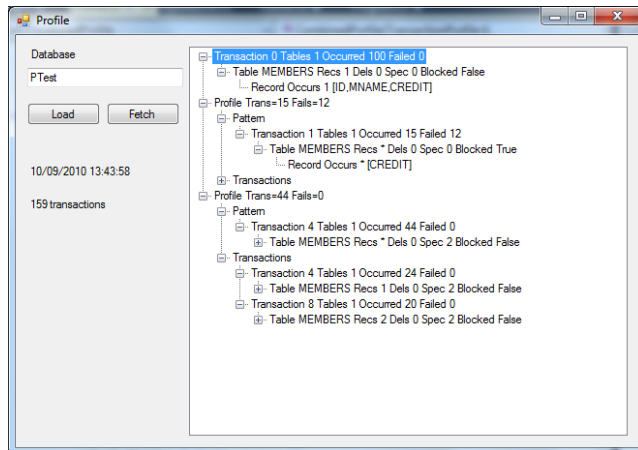
| | |
|--|--|
| DELETE http://localhost:8078/db/tb/w | This request will delete the specified rows. |
|--|--|

For PUT, POST and DELETE the returned data shows the number of rows affected.

4.7 The Profile Viewer

If profiling is turned on for a database (see section 8.4), Pyrrho maintains a transaction profile, which is persisted not in the database itself, but in an XML file: this is because it is a record not of the entire database activity, but just the periods for which profiling is enabled. Profiles can be deleted without harming the database in any way.

There is a convenience utility called ProfileViewer which displays the profile in a readable tree-view format. The profile can either be “fetched” from the server (assuming profiling is enabled), or “loaded” from the XML file (in which case ProfileViewer expects to find the xml file in its working folder).



5. Database design and creation

This chapter assumes that the reader is familiar with the general principles of relational databases including normal form and integrity constraints. For simplicity, we will document the use of the command line utility to carry out the steps discussed in this chapter.

Many activities could of course be automated using command scripts or application programs. For the latter, see Chapter 6.

5.1 Creating a Database

As mentioned in the last section, by default Pyrrho will create a database if necessary. To create a database, simply issue the command

PyrrhoCmd *dbname*

The file *dbname.osp* will be created in the database folder, and owned by the server account. The database will not be completely empty: it will have two initial records. The first of these will be a User record identifying the client account as the owner of the database. The second will be a default Role (with the same name as the database) which permits all actions on the database. The User will be the client's login ID. These two records specify the database owner and the schema role for the database.

The remainder of this subsection can be skipped on a first reading.

For example, suppose the Pyrrho service account on VANCOUVER is PYR_USR, and user LONDON\Fred issues the command

```
PyrrhoCmd -h:VANCOUVER MyLibrary
```

This command assumes that Fred has access to the client program, and to port 5433 on VANCOUVER where PyrrhoSvr is already running. If database MyLibrary already exists on host VANCOUVER, and LONDON\Fred is allowed to access it, the command line utility will start up on the client computer with a connection to this database. If MyLibrary does not exist on VANCOUVER, the PyrrhoSvr will create a new database file MyLibrary.pfl in the database folder, which will be owned by PYR_USR. MyLibrary.pfl will have an initial User record for user 'LONDON\Fred' of type owner, and a Role called 'MyLibrary'. In both cases, the PyrrhoCmd utility will now give the command prompt

SQL>

for SQL commands such as creation of the first few objects in this new database.

It is entirely reasonable for administrators to wish to limit the ability to create databases in the database folder. A better solution on a corporate network will be for databases to be initially created by their owners on their local machines but using their network login, and then copied by an administrator to the database folder on the server host. On the server host, the database folder should have permissions such that the server account cannot create new files. This approach would have the added advantage that the database file would actually continue to be owned by the client user.

5.2 Creating database objects

When using CREATE TABLE and other SQL statements at the command prompt, bear the following points in mind:

- SQL2016 syntax is somewhat different from many legacy systems. In particular:
- Identifiers are not case-sensitive unless they are enclosed in double quotes
- Double-quoted identifiers can be used to avoid confusion with reserved words and for identifiers that contain special characters
- By default, variable length data types can be used, e.g. CHAR instead of CHAR(16). If size and precision are specified, values are truncated. Precision specification for numeric types, if specified, is in decimal digits
- Single quotes are still used for string literals.

- Date, time, timestamp, and interval literals have a fixed syntax (e.g. DATE '2005-07-20') and the formats are not locale-sensitive.

In the current version the SQL2016 Timezone feature is not implemented (since it impedes moving a database between timezones), so time and timestamp are displayed for the local time zone on the computer in question, but are stored in the database in universal time.

For example the SQL statement

```
Insert into Winner ("YEAR",Rep) values (2005,'Fred')
```

will create a new record in an already-existing table WINNER(YEAR,REP) of form

| YEAR | REP |
|------|------|
| 2005 | Fred |

The double quotes are needed since YEAR is a reserved word in SQL2016. The single quotes are needed since Fred would otherwise be interpreted as an identifier (e.g. a column name). These requirements come from SQL2016.

```

C:\mkc\PyrrhoCmd\bin\Debug\PyrrhoCmd.exe
SQL> table "Sys$Column"
+-----+-----+-----+-----+-----+-----+-----+-----+
|Pos|Table|Name|Seq|Unique|Domain|Default|NotNull|Generate|
+-----+-----+-----+-----+-----+-----+-----+-----+
|102|A|A1|0|0|INTEGER|False|False|
+-----+-----+-----+-----+-----+-----+-----+-----+
SQL> table a
+-----+
|A1|
+-----+
|12345678901234567890|
+-----+
SQL> _

```

The illustration above shows an integer value (larger than “long”) in an “ordinary” integer field. The following example shows precision greater than double precision:

```

Shortcut to PyrrhoCmd.exe
SQL> select sum(c1) from c
+-----+
|Expr$1|
+-----+
|14691.34890123456789|
+-----+
SQL> table c
+-----+
|C1|
+-----+
|12345.678901234567890|
|2345.67|
+-----+
SQL> table "Sys$Column"
+-----+-----+-----+-----+-----+-----+-----+-----+
|Pos|Table|Name|Seq|Unique|Domain|Default|NotNull|Genera|
+-----+-----+-----+-----+-----+-----+-----+-----+
|107|C|C1|0|0|NUMERIC(0,0)|False|False|
+-----+-----+-----+-----+-----+-----+-----+-----+
SQL> _

```

5.2.1 Pyrrho's data type system

SQL2016 differs from older DBMS's by having a stronger system of data types. For example, user defined types have methods and constructors, and ordering functions can be declared.

Pyrrho's type system is as in SQL2016 with the following changes (a) char and char(0) indicate unbounded strings; int, int(0), integer and integer(0) indicate 2048-bit integers (see example in the section above); and real has a mantissa of 2048 bits by default; (b) size-specific standard types such as bigint or double are not supported; (c) persisted data is not changed by subsequent changes to column datatypes as long as it is coercible to the new type.

To explain the last point, suppose a table has a column of type numeric, and contains values with (say) up to 5 significant digits. Suppose now the table is altered so that the column is numeric(3). At this point all new values will be truncated on insertion, and all existing values will be truncated on retrieval, so that the table appears to contain values with a maximum of 3 significant digits. Now suppose the data type is changed back to numeric. Now the old data with 5 significant digits is visible once more, but of course the data inserted when the data type was numeric(3) will only have 3 significant digits. A case could be made that this behaviour is incorrect. But it helps to avoid accidental loss of data.

5.2.2 Indexes, Identity etc

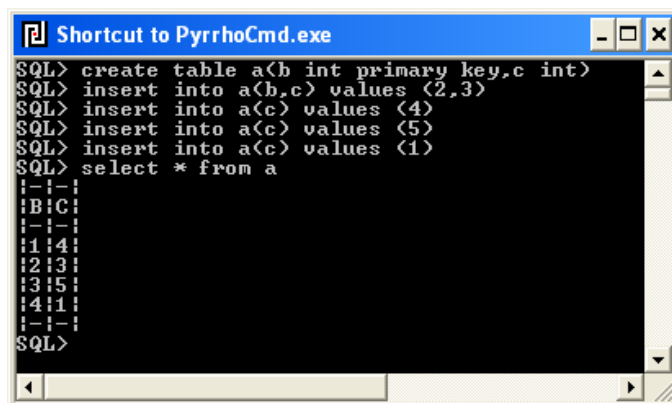
Indexes are not database objects in standards SQL. Integrity, uniqueness and referential constraints imply their use within the database engine, and behind the scenes Pyrrho uses indexes built in this way for automatic query optimisation.¹⁸

Pyrrho extends the notion of referential constraint to allow adapter functions: this behaviour is helpful when working with semantic inference systems. To illustrate the concept of an adapter function, consider this example:

foreign key(rdate, regionid) references t using (extract (year from rdate), regionid)

If the USING value is non-null, this should be a key in the referenced table. When this extended behaviour is used, the value of the computed foreign key is recorded along with the transaction.

Pyrrho does not have “identity”, “autonumber”, “sequence”, or “generator” features found in other databases. Instead, it has the following automatic feature, which it claims is better. If insert is proposed and a key component is missing, Pyrrho will find a suitable value: this behaviour is similar to POST in REST-based systems.



```

Shortcut to PyrrhoCmd.exe
SQL> create table a(b int primary key,c int)
SQL> insert into a(b,c) values (2,3)
SQL> insert into a(c) values (4)
SQL> insert into a(c) values (5)
SQL> insert into a(c) values (1)
SQL> select * from a
+---+
|B|C|
+---+
|1|4|
|2|3|
|3|5|
|4|1|
+---+
SQL>

```

If multiple rows are used in a single INSERT, as in “insert into a(c) values (4),(5),(1)”, the actual order of insertion will not necessarily seem to be the obvious one.

5.2.3 Row versions

Pyrrho supplies pseudocolumns in all base tables for row versioning purposes and security. There are four of these: SECURITY, CHECK, and its components POSITION and VERSIONING. CHECK is a string (actually an ETag), and POSITION and VERSIONING are integers.

As the name implies, SECURITY is reserved for use by the database owner as security administrator, and has a special type called Level. It is assignable to a value of type Level For further information see section 3.4.2..

The information in CHECK includes the transaction log name (for the partition containing the row), defining position of the row and current offset of the row version. When retrieved it refers to the version valid at the start of the transaction, but it can be used at any time subsequently (inside or outside the transaction) to see if the row has been updated by this or any other transaction (this is the only violation of transaction isolation in Pyrrho).

¹⁸ A syntax for CREATE INDEX has however been added to Pyrrho so support the MongoDB service.

The normal use of this data in application programming is to check that information previously read by the application is still valid. For example, if the application reads a row of data including the VERSIONING pseudocolumn and saves the version in a local variable called (say) *rvv*, a subsequent UPDATE of this row could specify WHERE VERSIONING=*rvv*, so that the application could check the number of rows affected.

Transaction behaviour complicates this picture considerably, as clients can retrieve rows during a transaction that updates them. The versioning information needs to be updated when the transaction commits, and it is (alas) the client's responsibility to include a set of versioned objects to have their versioning updated when calling Commit.

The Check() method for PyrrhoConnect allows the client to check whether a rowversion is still valid. See section 8.8.8.

5.3 Altering tables

SQL2016 allows for tables to be altered by adding, altering or dropping columns, and adding and dropping constraints.

Tables can also be dropped. Pyrrho supports the renaming of objects, with the following syntax for renaming tables:

```
alter table oldname to newname
```

and similar syntax for renaming other objects. Renaming columns is a special case:

```
alter table tname alter [ column ] oldname to newname
```

The position of a column can also be changed. (Column positions have little semantic value but it is convenient to have a known ordering of columns in select * results.)

```
alter table tname alter [ column ] cname position n
```

Renaming of database objects is role-specific: renaming applies to the current role (see sec 5.5 below), and requires appropriate privileges. The database file (transaction log) uses numeric identifiers instead of names. The Log\$... system tables show these, while the Role\$... system tables show the current names in the current Role. The following screenshot shows these numeric identifiers in the log:

```

SQL> create table a(a1 int)
SQL> create view vw as select a1 from a
SQL> alter table a alter a1 to id
SQL> select * from "Role$View"

```

| Pos | View | Select | Struct | Using | Definer |
|-----|------|----------------------|--------|-------|---------|
| 126 | VW | select "ID" from "A" | | | t53 |

```

SQL> select * from "Log$"

```

| Pos | Desc | Type | Affects | Transaction |
|-----|---|--------------|---------|-------------|
| 5 | PRole t53 | PRole | 0 | -1 |
| 32 | PUser | PUser | 32 | -1 |
| 55 | PTransaction for 3 Role=5 User=32 Time=11/02/2018 08:26:37 | PTransaction | 0 | 0 |
| 71 | PTable A | PTable | 71 | 55 |
| 77 | PDomain INTEGER | PDomain | 77 | 55 |
| 91 | PColumn A1 for 71(0)[77] | PColumn3 | 91 | 55 |
| 110 | PTransaction for 1 Role=5 User=32 Time=11/02/2018 08:26:48 | PTransaction | 0 | 0 |
| 126 | View VW as select "91" from "71" | PView | 126 | 110 |
| 156 | PTransaction for 1 Role=5 User=32 Time=11/02/2018 08:27:04 | PTransaction | 0 | 0 |
| 172 | Change PColumn3 [91] from A1 [PColumn A1 for 71(0)[77]] to ID | Change | 91 | 156 |

Pyrrho reconstructs compiled representations of database objects as required to reflect schema changes. The -V flag for the server allows this compilation process to be verified. Internally, objects not yet written to the database are given temporary numeric identifiers starting with 0x400000000001. For convenience these are abbreviated to '1', '2' etc.

Pyrrho does not modify database data when column types are changed: however, it does check that the database data can be coerced into the new column type.

5.4 Sharing a database with other users

One of the first uses for the client utilities should be to create the base tables of the database and grant permissions on them to users. The best ways of doing this are explained in the next section.

The database creator initially is the only user known to the database. Other users must be granted some specific privileges (so that they have a valid user id in the database) before they are allowed to make any changes to the database. The simplest (worst) way of sharing the database is to give all such named users permission to do anything, and all anonymous users permission to read anything:

Thus, under Windows, if database MyDb has no security settings on it, the creator of the database can share it with user mary on computer (or domain) JOE by the following GRANT statement:

```
grant "MyDb" to "JOE\mary"
```

This allows mary to access or alter the data in any way. (To let mary alter the schema she will need to be granted the admin option too.) The double quotes are needed because of case-sensitivity for database and user names.

```
grant "MyDb" to public
```

This allows any user to access or modify the database MyDb. Other grant statements can be used to apply specific privileges to specific database objects. Privileges not granted to PUBLIC can be revoked using the REVOKE statement.

When users are granted permissions later, they are of course able to access current data as determined by their current privileges. There are some special cases: the database owner is able to access all of the logs and profiles. For best results use Roles: these are described next.

Pyrrho allows the loading of a database as it was at some past time. If user permissions have changed since the “stop time” an administrator may need to recreate the user id of some user who had access permissions at the time in question. The required names can be found in the log. Note that Pyrrho user ids are user names (on Windows these have form "*DOMAIN\user*"), not the UIDs or SIDs used by the operating system. (See section 3.5)

5.5 Roles

For example, suppose a small sporting club (such as squash or tennis) wishes to allow members to record their matches for ranking purposes:

```
Members: (id int primary key, firstname char)
```

```
Played: (id int primary key, winner int references members, loser int references members,
agreed boolean)
```

For simplicity we give everyone select access to both these tables.

```
Grant select on members to public
Grant select on played to public
```

Although Pyrrho records which user makes changes, it will save time if users are not allowed to make arbitrary changes to the Played table. Instead we will have procedure Claim(won,beat) and Agree(id), so that the Agree procedure is effective only when executed by the loser. With some simple assumptions on user names, the two procedures could be as simple as:

```
Create procedure claim(won int,beat int)
    insert into played(winner,loser) values(claim.won,claim.beat)

Create procedure agree(p int)
    update played set agreed=true
    where winner=agree.p and
    loser in (select m.id from members m
             where current_user like '%'||firstname escape '!')
```

We want all members of the club to be able to execute these procedures. We could simply grant execute on these procedures to public. However, it is better practice to grant these permissions instead to a role (say, membergames) and allow any member to use this role:

```
Create role membergames 'Matches between members for ranking purposes'
Grant execute on procedure claim(int,int) to role membergames
Grant execute on procedure agree(int) to role membergames
Grant membergames to public
```

This example could be extended by considering the actual use made of the Played table in calculating the current rankings, etc.

In SQL2016, although a user may be entitled to use roles, he/she can only use one at a time, and the current role determines the permissions available. This is established in the connection string or using SET ROLE, and can be referred to as SESSION_ROLE.

Apart from the owner privilege (which can be held by just one user), granting privileges directly to users is deprecated. It is recommended to grant roles to users instead. Similarly, attempting to create a hierarchy of roles is also deprecated, and in Pyrrho the grant of role A to role B has the effect only of granting role A to all users authorised to use role B at the time of the grant: it does not create a permanent relationship between the roles; revoking a role from a role does nothing, and all roles are in the root namespace. This behaviour appears to be a departure from SQL2016 (see section 7.11 below).

Similarly, a grant of privileges does not create any permanent relationship between roles. For example, granting Select on a Table implies granting select on all of the *current* columns. The grant can be repeated later if new columns are added, or the new columns can be granted. Similarly in Pyrrho, access to a column can be revoked even though the role was previously granted access to the whole table (again see section 7.11).

A user who has been granted the admin option for a role can define new tables, procedures, constraints, types, etc in that role, and can grant privileges on these objects to other roles. All SQL code, if it is executable by the current role, executes with the permissions of the owner of the code (definer's rights). A user entitled to administer a role can modify metadata (including the object name, but excluding the iri) of objects visible from their role: other defining properties of the object can only be changed by the owner or schema role. All standard types are PUBLIC and all roles remain in the root namespace. Other objects can be prefixed with the name of the role if this is helpful for disambiguation.

On creation a database has a default role with the same name as the database, and the owner of the database can use this "schema" role to create the starting set of objects for the database.

The System tables can be used to ascertain the privileges held at any time: from v4.5 these are accessible by the database owner, or by using the schema role.

5.6 Stored Procedures and Functions

Pyrrho supports stored procedures and functions following the SQL2016 syntax (volumes 2 and 4). The programming model offered in this way is computationally complete, so the use of external code written in other programming languages is not supported.

Following SQL2016 the syntax :v is not supported for variable references, and instead variables are identified by qualified identifier chains of form a.b.v. The syntax ? for parameters is not supported either.

Following SQL2016-2-11.60, procedures never have a returns clause (functions should be used if a value is to be returned), and procedure parameters can be declared IN, OUT or INOUT and can be RESULT parameters. Variables can be ROW types and collection types. For functions, TABLE is a valid RETURNS type (it is strictly speaking a "multiset" in SQL2016 terminology). From SQL2016-2-6.45 we see that RETURN TABLE (QueryExpression) is valid syntax for a return statement.

The operation of the security model for routines in SQL2016 is rather subtle. All routines operate with definer's rights by default, but access to them is controlled according to the current role.

Pyrrho allows some metadata properties to be set for functions. MONOTONIC (order-preserving) functions used in join conditions can allow Pyrrho to speed up joins by sorting the table operands provided USING syntax specifies the use of an adapter function. The INVERTS metadata property establishes a pair of mutually inverse functions and this information means that views and joins defined USING such functions can be updatable depending on the availability of keys.

The next few sections include some outlines of procedure statements specified in SQL2016-4 and supported in Pyrrho. Complete syntax summaries for Pyrrho are given in chapter 7.

5.6.1 Table-valued functions

```
create table author(id int primary key, aname char)
create table book(id int primary key, authid int, title char)
...
```

```
create function booksby(auth char) returns table(title char)
  return table(select title from author a inner join book b on
    a.id = b.authid where aname = booksby.auth )
```

This example also shows that a routine body is a single procedure statement (possibly a compound BEGIN..END statement). If you use the command line utility PyrrhoCmd (section 4.2), very long SQL statements such as the last one above can be enclosed in square brackets and supplied on several lines as described in section 4.2.5.

The above function can be referenced by statements such as

```
select * from table(booksby('Charles Dickens'))
```

The keyword `table` in this example is required by SQL2016-2(7.6).

5.6.2 Simple statements

Semicolons are used as separators in statements lists, and are not part of any statement syntax. Declarations can appear anywhere in a statements list (which defines the scope of the declaration).

BEGIN statements END

DECLARE varnames type

SET id = value

SET (ids) = value

SQL statements such as CREATE, GRANT, INSERT, DELETE, REVOKE, DROP are also allowed here, as are SELECT INTO , which is basically queryexpression INTO ids .

RETURN value

CALL procedure (values)

5.6.3 Decision Statements

CASE value { WHEN value THEN statements } [ELSE statements] END CASE

CASE WHEN searchcondition THEN statements [ELSE statements] END CASE

IF condition THEN statements { ELSEIF condition THEN statements } [ELSE statements] END IF

5.6.4 Iterative statements

Iterative statements can be labelled (with an identifier followed by a colon) and LEAVE and ITERATE statements can refer to these labels, to break out of nested loops or skip to the next iteration of a loop. Variable references to variables declared inside these constructs can be of form label.name .

FOR queryexpression DO statements END FOR

LOOP statements END LOOP

WHILE searchcondition DO statements END WHILE

REPEAT statements UNTIL searchcondition END REPEAT

LEAVE label

BREAK

ITERATE label

5.6.5 Condition handling statements

The following condition handling apparatus (as specified in SQL2016) is also supported. The predefined conditions are SQLSTATE string, SQLEXCEPTION, SQLWARNING and NOT FOUND, but any identifier can be used . All of the following can appear where statements are expected, and handlers apply anywhere in the scope where they are declared.

DECLARE CONTINUE HANDLER FOR conditions statement

DECLARE EXIT HANDLER FOR conditions statement

DECLARE UNDO HANDLER FOR conditions statement

UNDO is defined in the SQL standard (04-4.8): it offers more fine-grained behaviour than rollback, as it merely removes any changes made in the scope of the handler.

SIGNAL condition setlist

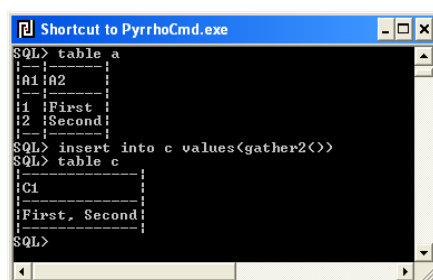
Here the options for condition are SQLSTATE string or any identifier. The setlist allows a set of keywords defined in the SQL standard, corresponding to items in the diagnostics area. For example, you can pass a reason in the diagnostic area using the MESSAGE_TEXT keyword.

5.6.6 Examples

The following functions perform the same task. The first uses a handler, while the second uses a for statement.

```
create function gather1() returns char
begin
  declare c cursor for select a2 from a;
  declare done Boolean default false;
  declare continue handler for sqlstate '02000' set done=true;
  declare a char default '';
  declare p char;
  open c;
  repeat
    fetch c into p;
    if not done then
      if a = '' then
        set a = p
      else
        set a = a || ', ' || p
      end if
    end if
  until done end repeat;
  close c;
  return a
end
```

```
create function gather2() returns char
begin
  declare b char default '';
  for select a2 from a do
    if b='' then
      set b = a2
    else
      set b = b || ', ' || a2
    end if
  end for;
  return b
end
```



5.7 Structured Types

SQL2016 supports structured types. Structured types, multisets and arrays can be stored in tables. There is a difference between (say) a table with certain columns, a multiset of rows with similarly named fields and a multiset of a structured type with similarly named attributes, even though in an element of each of these the value of a column, field or attribute respectively is referenced by syntax of the form *a.b*. Some constructs within SQL2016 overcome these differences: for example the INSERT statement uses a set of values of a compatible row type to insert data into a table, and TABLE *v* constructs a table out of a multiset *v*. The type model in Pyrrho allows user-defined types to be simple or structured, they can define XML data types (e.g. for RDF/OWL use) have an associated URI and constraints.

To use structured types, it is necessary to CREATE TYPE for the structured type: this indicates the attributes and methods that instances of the type will have. Then a table (for example) can be defined that has a column whose values belong to this type. At this stage the table could even be populated since (there is an implicit constructor for any structured type); but before any methods can be invoked they need to be given bodies using the CREATE METHOD construct. Note that you cannot have a type with the same name as a table or a domain (since a type has features of both).

Values of a structured type can be created (using NEW), assigned to variables, used as parameters to suitably declared routines, used as the source of methods, and placed in suitably declared fields or columns.

```

PyrrhoCmd
SQL> Create type point as (x int, y int)
SQL> Create type size as (w int, h int)
SQL> Create type line as (strt point, en point)
SQL> Create type rect as (t1 point, sz size) constructor method rect(x1 int, y1 int, x2 int, y2 int), method centre() returns point
SQL> Create table Figure (id int primary key, title char)
SQL> Create table FigureLine (id int primary key, fig int references figure, what line)
SQL> Create table FigureRect (id int primary key, fig int references figure, what rect)
SQL> create constructor method rect(x1 int, y1 int, x2 int, y2 int) begin t1 = point(x1, y1); sz = size(x2-x1, y2-y1) end
SQL> create method centre() returns point for rect return point(t1.x+sz.w/2, t1.y+sz.h/2)
SQL> create function centrerect(a int) returns point return (select what.centre() from figurerect where id=a)
SQL> insert into figure values(1, 'Diagram')
SQL> insert into figurerect values(1, 1, rect(point(1, 2), size(3, 4)))
SQL> insert into figurerect values(2, 1, rect(4, 5, 6, 7))
SQL> insert into figureline values(1, 1, line(centrerect(1), centrerect(2)))
SQL> table figureline
+----+-----+
| ID | FIG | WHAT |
+----+-----+
| 1  | 1   | LINE<POINT<2, 4>, POINT<5, 6>> |
+----+-----+
SQL>

```

Notes: 1) The coordinates have been declared as int, so the first point here is not (2.5, 4)). 2) In v7, the last method here must refer to a as *id=centrerect.a*, not *id=a*. This is because in the SQL standard (2011, sec 6.6), unqualified names need to lie in the context of a range variable or table name, to which they refer, and so an identifier chain is required in this example.

Arrays and multisets of known types do not need explicit type declaration. Their use can be specified by the use of the keyword ARRAY or MULTiset following the type definition of a column or domain.

5.8 Triggers

SQL2016 supports triggers.

Pyrrho has built-in facilities to do activity logging (see section 3.5 and 8.2). However, triggers allow for a more customizable approach as the following example shows:

```

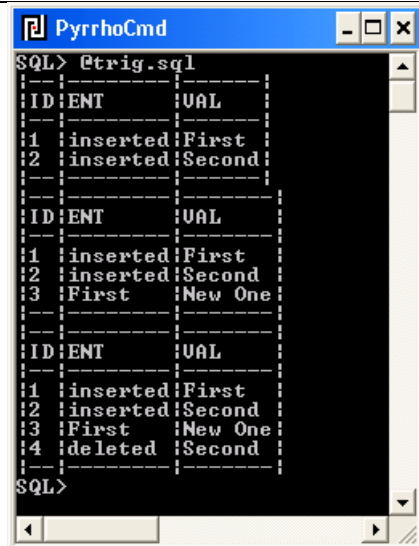
create table test1(id int primary key, val char)
create table test2(id int primary key, ent char, val char)
create procedure log(g char, h char) insert into test2(ent, val)
values(g, h)
create trigger logininsert after insert on test1 referencing new row as
a for each row call log('inserted', a.val)
create trigger logupdate before update on test1 referencing old row as
a new row as b for each row call log(a.val, b.val)

```

```

create trigger logdelete before delete on test1 referencing old row as
  a for each row call log('deleted',a.val)
insert into test1 values(1,'First'),(2,'Second')
table test2
update test1 set val='New One' where id=1
table test2
delete from test1 where id=2
table test2

```



5.9 Provenance and extended subtype semantics

Many changes have been made to Pyrrho especially since version 4.4, to allow much stronger internal data types. These are useful for semantic inference. However, Pyrrho itself does not support semantic inference. Mechanisms are being developed to allow semantic tools to take advantage of this extra information.

5.9.1 IRI references and subtypes

Semantic information (from provenance or some other source) can be used to define a subtype, for example, a Pyrrho extension to SQL2016 allows declarations such as:

```
CREATE DOMAIN ukregno 'uri://carregs.org/uk'
```

The final string constant is stored by the database as metadata. Subtype information can be examined using the standard SQL2016 type predicate, for example

```
SELECT * FROM cars WHERE reg IS OF (ukregno)
```

5.9.2 Row and table subtypes

In an INSERT operation, whole rows or tables can be assigned a subtype using the TREAT function. A structured type can be declared with additional metadata uri information, such as

```
CREATE TYPE t AS (c CHAR, b INT) 'http://a.com'
```

Then supposing table A had been created with a compatible row type, such as CREATE TABLE(c CHAR,b INT), we could write

```
INSERT INTO A TREAT (VALUES ('Ex',1)) AS t
```

The type of a row can be tested using the ROW keyword, e.g. ROW IS OF(*type*) . Following the SQL standard, the row type for a table or view T is REF(T).

5.10 Generated Columns

From version 4.5, Pyrrho provides support for conceptual modelling, based on roles (see section 3.6). Three additional features have been added to Pyrrho to enrich this process.

The first is the notion of an updatable generated property, which can be implemented at the conceptual (role) level and creates no overhead on the physical database. The idea here is that some generation rules are one-to-one, so that there may be a suitable action to take on the source data if someone tries to modify the generated property.

In SQL2016 there is already a way of defining a “generated always” column in the physical database. This behaves like a get property in C#: it appears to be a column and is retrieved using `select *` etc, but (unlike the “generated by default” column type) is not separately stored in the database or separately updatable, e.g.

```
create table mystats (salesmonth date(year to month), home numeric,
overseas numeric, total generated always as (home+overseas))
```

Now such a generated column feature can also be used to help make data compatible, by performing type conversions or combining fields, and this aspect is expected to become more important with the use of semantically-significant subtypes as described above. This notion is at the bottom of the introduction of updatable generated columns, which are described below.

The second implements the type of navigable two-way relationships found in entity-relationship diagrams. In previous versions of Pyrrho this feature was only provided in the Java Persistence library (see section 6.14). The example given in the Java Enterprise Edition 5 tutorial was given in terms of a player table and a team table. By setting up an infrastructure of annotations, the Java Persistence Query Language allowed retrieved items to include lists of teams that a given player was in. For REST, Pyrrho offers a very simple solution using the “of” syntax (sec 3.7). A proposal for “reflection” was implemented for v4.5, but its columns, containing arrays of rows, are rather unmanageable in SQL: in the meantime the REST facilities are also available in SQL, using the HTTP syntax extension (see sec 7).

The third feature in this section can be used to specify a column with a restricted set of values by referring to a lookup table. Again the syntax is an alternative for `ColumnDefinition`:

```
| id Table_id '.' Column_id
```

The domain is implicitly defined to be that of the referenced column, and the allowed values are the current distinct values of the specified column in the lookup table. Again, this syntax can be used to define a property in a role-based data model, and in that context entity and property identifiers can be used for getting the list of referenced values.

The rest of this section focuses on generated columns. The generated always clause has the following syntax in SQL2016 (section 11.4)

```
<column definition> ::=
<column name> [ <data type or domain name> ]
[ <default clause> | <identity column specification> | <generation clause> ]
[ <column constraint definition>... ] [ <collate clause> ]

<generation clause> ::=
<generation rule> AS <generation expression>

<generation rule> ::=
GENERATED ALWAYS

<generation expression> ::=
<left paren> <value expression> <right paren>
```

Annoyingly, the SQL2016 standard then restricts the usefulness of the generation expression by requiring that “10) If <generation clause> GC is specified, then: a) Let GE be the <generation expression> contained in GC. b) C is a generated column. c) Every <column reference> contained in GE shall reference a base column of T. d) GE shall not be possibly deterministic. e) GE shall not contain a <routine invocation> whose subject routine possibly reads SQL-data. f) GE shall not contain a <query expression>.”

In Pyrrho requirements c), e) and f) are dropped, and the syntax is extended to include an update rule:

```
<generation clause> ::=
```



```
<generation rule> AS <generation expression> [ <update rule> ]  
<update rule> ::=  
UPDATE <left paren> <set clause list> <right paren>
```

where <set clause list> is defined in section 14.15 of the standard. In Pyrrho's syntax description in section 7.2 this is an Assignment .

With this syntax we can write definitions such as

```
create table mystats (salesyear int, salesmonth int,  
    salesperiod date(year to month) generated always as  
        (cast (salesyear as date(year))+  
         cast(salesmonth as interval(month)))  
    update (salesyear=extract(year from salesperiod),  
        salesmonth=extract(month from salesperiod)))
```

or include them in property definitions in data models. It can be seen that such a definition, though tedious in itself, could save a lot of effort to users of this data.

As a result of the changes in this section, generated columns may not be used in a table constraint.

6. Pyrrho application development

This section contains technical information required by database application programmers. For many purposes the first few subsections are sufficient.

For simplicity, it is assumed in sections 6.1-6.5 that the application programmer is writing in C# under Windows or Linux/Mono. Later sections discuss the APIs available for Python, Java, PHP, and SWI-Prolog, all available on Windows and Linux.

6.1 Getting Started

For C#, the programming model is ADO.NET, which is accessible in the common language runtime by

```
using System.Data;
```

Pyrrho provides a small dll (PyrrhoLink.dll or OSPLink.dll) for making the initial TCP/IP connection to the PyrrhoServer.

```
using Pyrrho;
```

and extends the facilities of ADO.NET to handle SQL additional data types such as Multiset and Array.

On platforms such as Mono v1, where System.Data is not available, the Pyrrho namespace is extended to work without System.Data.

In either case, the resulting effective API is documented in section 8.7 of this manual. Note that from v7 the Pyrrho API supports a simple kind of prepared statements, see sec. 8.7.12, but in a different way from the SQL standard, as the prepared statements are stored in the current connection and not in the database.

Unless the dll is installed in the global assembly cache, it should be copied to the same folder as the application executable. If you are using a tool such as Visual Studio to develop your application, ensure that the project references PyrrhoLink.dll or OSPLink.dll. You may need to browse to the location where Pyrrho has been installed. Visual Studio will then make information from PyrrhoLink.dll available during compilation and place a copy of PyrrhoLink.dll or OSPLink.dll in the same folder as the executable.

If you are using the .NET SDK instead of Visual Studio, then when your application is compiled, you will need to mention the reference to PyrrhoLink.dll or OSPLink.dll in the compilation command line:

```
csc -r:PyrrhoLink.dll test.cs
```

assuming that a copy of PyrrhoLink.dll is in the folder where compilation takes place. It will also be needed for execution.

6.2 Opening and closing a connection

The database connection is provided using an extension to the standard ADO.NET IDbConnection interface:

```
var db = new PyrrhoConnect(connectionstring);
```

See section 6.4 for details of the connection string. A sample is provided below.

The connection must be opened before it can be used:

```
db.Open();
```

Connections should be closed when no longer required:

```
db.Close();
```

An application may use this cycle many times during its operation, as connections may be opened for different databases, groups of databases, or using different roles. By default, the connection operates in autocommit mode where every command is immediately committed. If explicit transactions are used, any uncommitted transactions are silently rolled back when a connection is closed (see section 8.8.20). Two functions in this interface described below are CreateCommand (section 6.5) and BeginTransaction (section 6.7).

As usual with ADO.NET, at most one IDataReader can be open for any connection. Remember to close the IDataReader before calling another ExecuteReader.

For example, the following console program connects to a database **Movies** on the local server, and lists the **TITLEs** found in table **MOVIE**:

```
using Pyrrho;
class Test
{
    public static void Main(string[] args)
    {
        var db = new PyrrhoConnect("Files=Movies");
        db.Open();
        var cmd = db.CreateCommand();
        cmd.CommandText = "select title from Movie";
        var rdr = cmd.ExecuteReader();
        while (rdr.Read())
            Console.WriteLine((string)rdr["TITLE"]);
        rdr.Close();
        db.Close();
    }
}
```

Note that SQL is not normally case sensitive: see section 5.2. If you want SQL identifiers to be case sensitive, you will need to double-quote them, and in C# strings, the double-quote will need to be escaped. For more details of the ADO.NET and similar functionality, see section 8.6.

POCO technology is also available. Pyrrho will supply class definitions to paste into your application program, either using the REST interface, or from the Role\$Class system table. If this has been done for the **MOVIE** class here, the above code can be simplified to:

```
using Pyrrho;
class Test
{
    public static void Main(string[] args)
    {
        var db = new PyrrhoConnect("Files=Movies");
        db.Open();
        var obs = db.Get("/MOVIE");
        foreach(MOVIE m in obs)
            Console.WriteLine(m.TITLE);
        db.Close();
    }
}
```

As suggested by format of the Get parameter, this mechanism uses the new role-based REST features. See section 6.5 below.

6.3 The connection string

ConnectionString = filename {';'Setting} .

Setting = id='val'{'val'} .

If the connection string begins with Files=, this portion is ignored for reasons of backward compatibility for single-database connections. Note that a database file name cannot contain = or ; .

The possible fields in Settings are as follows:

| Field | Default value | Explanation |
|--------------------|---------------|--|
| <i>Base</i> | | <i>Used by server-server communication to create a new partition remotely. Not for client-server use.</i> |
| <i>Coordinator</i> | | <i>Used in server-server communications: the transaction coordinator server</i> |
| Host | 127.0.0.1 | The name of the machine providing the service. |
| <i>Length</i> | | <i>Used in server-server communication to notify clients of a new master file length. Not for client-server use. The connection is closed immediately.</i> |
| Locale | | The name of the locale to be used for error reporting. The default is supplied by the .NET framework. |

| | | |
|----------|---------------------|--|
| Modify | | The default value is true for the first file in the connection, and false for others. If the value true is specified then it applies to all of the Files in the current connection string. |
| Port | 5433 | The port on which the server is listening |
| Provider | PyrrhoDBMS | |
| Role | <i>databasename</i> | A role name selected as the session role. If this field is not specified, the session role will be the default database role if the user is the database owner or has been granted this role (it has the same name as the database), or else the guest role, which can access only PUBLIC objects. |
| Stop | | If a value is specified, this means that Pyrrho is to load the database as it was at some past time. |
| User | | <i>This field is supplied by infrastructure</i> |

6.4 REST and POCO

POCO stands for Plain Old CLR Object. In addition to the HTTP REST service in section 3.7, Pyrrho has a RESTful API that supports row-versioning (Laiho and Laux, 2010). The Role\$Class system table (see sec 8.4.1) supplies a set of class definitions that can be pasted into a C# application¹⁹. Similar tables Role\$Java (8.4.9) and Role\$Python (8.4.16) provide class definitions for Java and Python.

For example Pyrrho will generate a C# class definition similar to the following:

```
[Schema(█)]
/// <summary>
/// Class AUTHOR from Database haikus, Role haikus
/// </summary>
public class AUTHOR : Versioned
{
    [Key(0)]
    public String ID;
    public String NAME;
    public Byte[] PIC;
}
```

The Versioned base class contains the following data, which is used by the database to check for transaction conflicts when any of the functions below are called²⁰. Do not modify these values (in C# and Java they are internal to the library).

```
public class Versioned
{
    public string version = "";
    public string readCheck = "";
}
```

The Schema attribute gives the version of the schema for this object (the schema key). Additional data annotations are added for fields with specified precision or scale, or that require special processing (e.g. keys, Date). The readCheck field is similar to the concept of ETags for HTTP (see RFC 7232).

A database connection is then used to access the database. PyrrhoConnect conforms to normal ADO.NET/ODBC rules: it is opened for a database, may have a current transaction that can be committed or rolled back.

This leads to a very tidy RESTful API, consisting of the following methods for the PyrrhoConnect (or Connection) class, where E is Versioned or a subclass of Versioned defined by code obtained from the Role\$Class (or Role\$Java or Role\$Python) system table in Pyrrho. Classically, REST uses the HTTP 1.1

¹⁹ It is important to note that these class definitions should always be generated from the database and not copied from definitions used in another database, not even a database with the same structure and objects. This is because there is a dependency on the length of the user name of the database owner.

²⁰ The versioning is remembered and will be checked even in a later Connection. Explicit transactions should be kept as short as possible since they must run exclusively in one thread.

verbs of GET, POST, PUT, and DELETE, and the strongly typed Get and FindXXX methods below are recommended over Get(..).

| Method | Explanation |
|--|---|
| C#: void Delete(E ob) Java: void Delete(E ob) Python*: delete(ob) | Delete the given entity from the database table E. An exception will be thrown if the object is out of date. |
| C#: E[] FindAll<E>() Java: Versioned[] FindAll(E.class) Python: E[] findAll(E) | Retrieve all entities of the given Versioned type. |
| C#: E FindOne<E>(params IComparable[] w) Java: FindOne(E.class, Object[] w) Python: E findOne(E, w) | Retrieve a single entity of a given Versioned type E with key fields w. |
| C#: E[] FindWith<E>(string w) Java: Versioned[] FindWith(E.class, String w) Python: E[] findWith(E, w) | Retrieve a set of Versioned entities satisfying a given condition. w is a comma-separated set of conditions of form <i>field=value</i> . Field names are case sensitive and values are in SQL format (single quotes on strings are optional in the absence of ambiguity). |
| C#: E[] Get<E>(string rurl) Java: Versioned[] Get(E.class, String w) Python: E[] get(E, rurl) | The relative url provided should be compatible with the Versioned subclass E. |
| C#: void Post(E ob) Java: void Post(E ob) Python*: post(ob) | The object should be a new entity. An integer key field will be autopopulated with a suitable value ²¹ . |
| C#: void Put(E ob) Java: void Put(E ob) Python*: put(ob) | The object should be an updated version of an entity that is in the database, identifiable by key and/or version. An exception will be thrown if the object is out of date. |

* Make sure that ob has been created by the constructor of a subclass of Versioned, or by one of the methods in this table.

Triggers defined in the database are called for POST, PUT and DELETE as if the operations were INSERT, UPDATE and DELETE SQL operations. In the respects noted, the Versioned base class departs from normal POCO behaviour.

As Laiho and Laux observed, excessive caching of database objects in middleware can complicate transaction processing and lead to data coming from a mixture of database states.

6.5 DataReaders

The IDataReader interface is fully described in the ADO.NET documentation. To get an IDataReader, call the ExecuteReader() method of IDbCommand, e.g.:

```
IDataReader rdr = cmd.ExecuteReader();
```

The columns that will be returned in the rows of the DataReader can be accessed using the following IDataReader methods (extracted from the IDataRecord part of the IDataReader interface):

| Property or Method signature | Explanation |
|------------------------------|--|
| int FieldCount | Gets the number of fields returned per row |
| string GetName(int i) | Returns the name of the ith field (the first field is field 0) |
| Type GetFieldType(i) | Returns the System.Type of the ith field |

Before an IDataReader can access any data, the Read() method must be called. Each time it is called, it moves on to the next row of the results if there is one. This function returns a Boolean value: which is true if Read() has succeeded in moving to the next row of data, and false if there is no more data.

Assuming that Read() has returned true, the fields in the returned row can be obtained by indexing the DataReader object. Fields can be indexed by ordinal position or by name. The value returned is a

²¹ If the table has some other kind of primary key, use INSERT instead of POST to insert new entities, or use a BEFORE INSERT trigger to create suitable key fields.

System.Object. If the corresponding value might be a null value, then it can be checked against DBNull.Value (or for being DBNull) before being cast to the expected System.Type.

For example:

```
if (!(rdr[1] is DBNull)) then Console.WriteLine((string)rdr[1]);
```

For languages where casting to different types is awkward, the DataReader interface has a range of functions of form GetByte(i), GetInt64(i) etc. For integers and numerics whose precision cannot fit into the standard types, Pyrrho returns a string representation. If this is expected, then you should test if the value is string.

| SQL basic type | .NET data type |
|--------------------------------|-----------------|
| Boolean | System.Boolean |
| Int, integer | System.Int64 |
| Real, Float | System.Double |
| Char, CLOB | System.String |
| BLOB | System.Byte[] |
| Date, Timestamp | System.DateTime |
| Row, Interval, Array, Multiset | See section 8.5 |

If indexing by name is used, remember that strings in the programming language are case-sensitive, even though SQL (unquoted) identifiers are not, so you will probably need to ensure your field names are in upper case letters.

The client library uses the DataReader interface with as few added classes as possible. The only added classes are PyrrhoRow, PyrrhoArray, and PyrrhoInterval. Dates and Timestamps use the DateTime class in the common language runtime, Times use the TimeSpan class for a simple time of day, but Intervals are handled using PyrrhoInterval. The three new classes are documented in section 6.8.

The routines ExecuteReaderCrypt and ExecuteNonQueryCrypt send the SQL string to the server using Pyrrho's encryption algorithm.

6.6 LINQ

Language-Integrated Query was an innovation in C# 3.0. Linq allows queries of the sort

```
var query1 = from t in things where t.Cost > 300 select new { t.Owner.Name, t.Cost };
```

to be written directly in C#.

The Pyrrho support for Linq is therefore inspired by the idea of supporting queries to simple small databases, and avoiding declarations and annotations wherever possible. The client-side objects can be modified using the methods in sec 8.8.8. The Linq support is only for single-component primary keys (they can be any scalar type and do not have to be called "Id").

The following complete program works with a database called home, which contains two tables with the following structure:

```
create table "Person" ("Id" int primary key, "Name" char, "City" char, "Age" int)
```

```
create table "Thing" ("Id" int primary key, "Owner" int references "Person", "Cost" int, "Descr" char)
```

Then the Role\$Class system table (see sec 8.4.1) provides text for the two class definitions as below. The PyrrhoConnect connects to the database as usual, and the database is opened. Two PyrrhoTable<> declarations form a link between client side data and data in the home database. Then the LINQ machinery is available. (For the program to produce output, there needs to be some data in the tables.)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Pyrrho;

namespace ConsoleApplication1
{
    /// <summary>
    /// Class Person from Database home, Role home
    /// </summary>
```

```

public class Person {
    public System.Int64 Id; // primary key
    public System.String Name;
    public System.String City;
    public System.Int64 Age;
}

/// <summary>
/// Class Thing from Database home, Role home
/// </summary>
public class Thing {
    public System.Int64 Id; // primary key
    public Person Owner;
    public System.Int64 Cost;
    public System.String Descr;
}

class Program
{
    static void Main(String[] args)
    {
        // Data source.
        PyrrhoConnect db = new PyrrhoConnect("Files=home");
        db.Open();
        // constructs an index for looking up t.Owner as a side effect
        var people = new PyrrhoTable<Person>(db);
        var things = new PyrrhoTable<Thing>(db);
        // Query creation.
        var query1 = from t in things where t.Cost > 300 select new {
t.Owner.Name, t.Cost };

        // Query execution.
        foreach (var t in query1)
            Console.WriteLine(t.ToString());

        var query2 = from p in people
            select new { p.Name, Things=from t in things
            where t.Owner.Id == p.Id select t};

        foreach (var t in query2)
        {
            Console.WriteLine(t.Name + ":" );
            foreach (var u in t.Things)
                Console.WriteLine("  " + u.Descr);
        }
        db.Close();
    }
}

```

6.7 Using PHP

There is an extra class ScriptConnect in PyrrhoLink.dll which is very useful for use with the scripting language PHP.

PHP can be used for building web applications, and then the same considerations as in the last section apply for the user identity of the web server and ownership of the databases. Unfortunately there does not yet seem to be a good way for PHP to work with Mono as a web server extension.

To enable PHP support for Pyrrho under Windows, an administrator needs to issue the following two commands from the folder that contains PyrrhoLink.dll:

```

gacutil -i PyrrhoLink.dll
regasm PyrrhoLink.dll -tlb:PyrrhoLink.tlb

```

(For OSP use OSPLink instead of course.) You need to ensure that your PHP installation is 32-bit and has the php_com_dotnet extension.

The following steps can be used to access Pyrrho databases from PHP:

To create a connection to a Pyrrho database:

```

$conn = new COM("OSPLink"); // "PyrrhoLink" for the Pro version
$conn->ConnectionString = ...;

```

```
$conn->Open();
```

Once a connection is open as above, an SQL statement can be sent to the database as follows

```
$rdr = $conn->Execute(...);
```

The result returned will be a `ScriptReader` in the case that the SQL statement returns data.

Then

```
$row = $rdr->Read();
```

can be used to return successive rows of the data as variant arrays. If there are no more rows then the value returned is -1, which can be tested using `is_int($row)`:

```
$row = $rdr->Read();
while(!is_int($row))
{
    print($row[0].': '.$row[1].'\n'); // or "\r\n"
    $row = $rdr->Read();
}
```

`$rdr->Close();` should be called when the reader is no longer required.

`$conn->Execute(...)` can also be used for other types of SQL statements.

6.8 Python

OSPLink.py is available in the distribution and enables the open-source Pyrrho server OSPSvr to be accessed from Python 3.4 clients. The API has similarities to Pyrrho's version of ADO.NET as documented in section 8.8, and the following subsections are numbered similarly to those of section 8.8 in a conscious attempt to show the relationship.

Since version 5.4 of Pyrrho, thread-safety is enforced by the OSPLink.py library. The connection object can be shared between threads. But a connection can have at most one transaction and/or command active at any time, and these cannot be shared between threads. As a result, the methods noted below will block until the connection is available.

To use OSPLink.py, place it in the same folder as your Python script.

For example:

```
from OSPLink import *
from builtins import print

conn = PyrrhoConnect("Files=Temp;User=Fred")
conn.open()
try:
    conn.act("create table a(b date)")
except DatabaseError as e:
    print(e.message)
conn.act("insert into a values(current_date)")
com = conn.createCommand()
com.commandText = 'select * from a'
rdr = com.executeReader()
while rdr.read():
    print(rdr.val(0))
rdr.close()
print("Done")
```

6.8.1 DatabaseError

| Attribute | Explanation |
|--------------------|--|
| <i>dict</i> info | Information placed in the error: see section 8.1.2 |
| <i>str</i> message | The message text: see section 8.1.1 |
| <i>str</i> sig | The SQLSTATE |

6.8.2 (Date)

OSPLink.py uses the Python *date* class.

6.8.3 DocArray

| Attribute | Explanation |
|----------------------------------|---|
| <code>build(ob)</code> | Append the attributes of <code>ob</code> not starting with <code>_</code> to this document; the process recursively builds embedded Documents and DocArrays for structured values |
| <code>bytes()</code> | Create the Bson representation of this DocArray |
| <code>cls[] _extract(cls)</code> | Construct an array of <code>cls</code> objects from this |
| <code>fromBson(bytes)</code> | Append the given Bson data to an empty DocArray |
| <code>list items</code> | The items of the DocArray |
| <code>parse(str)</code> | Append items from the given string to this DocArray |
| <code>str()</code> | Create the Json representation of this DocArray |

6.8.4 Document

| Attribute | Explanation |
|--------------------------------|---|
| <code>build(ob)</code> | Append the attributes of <code>ob</code> not starting with <code>_</code> to this document; the process recursively builds embedded Documents and DocArrays for structured values |
| <code>bytes()</code> | Create the Bson representation of this document |
| <code>cls _extract(cls)</code> | Construct an object of type <code>cls</code> from this |
| <code>fromBson(bytes)</code> | Append the given Bson data to this document |
| <code>list fields</code> | Each field is a pair (key,value) |
| <code>parse(str)</code> | Append fields from the given string into this document |
| <code>str()</code> | Create the Json representation of this document |

6.8.5 DocumentException

This subclass of Exception is used to report parsing errors in the `Document.parse` method.

6.8.6 (ExcludeAttribute)

There is no analogue to C# attributes/Java annotations in Python.

6.8.7 (Field Attribute)

There is no analogue to C# attributes/Java annotations in Python.

6.8.8 (KeyAttribute)

There is no analogue to C# attributes/Java annotations in Python. But a model class declaration can specify the list of primaryKey fields.

6.8.9 PyrrhoArray

| Attribute | Explanation |
|------------------------|----------------------------|
| <code>str kind</code> | The domain name if defined |
| <code>list data</code> | The items in the array |

6.8.10 PyrrhoColumn

| Attribute | Explanation |
|-------------------------------|---|
| <code>str columnName</code> | The name of the column |
| <code>str caption</code> | The name of the column |
| <code>str datatypeName</code> | The domain or type name of the column |
| <code>int type</code> | The PyrrhoDbType of the column (see sec 6.8.13) |

6.8.11 PyrrhoCommand

| Attribute | Explanation |
|------------------------------|---|
| <i>str</i> commandText | The SQL statement for the Command |
| PyrrhoConnect conn | The connection |
| PyrrhoReader executeReader() | Initiates a database SELECT and returns a reader for the returned data (as in IDataReader). Will block until the connection is available. |
| int executeNonQuery() | Initiates some other sort of Sql statement and returns the number of rows affected. Will block until the connection is available. |

6.8.12 PyrrhoConnect

| Attribute | Explanation |
|--|---|
| <i>int</i> Act(sql) | Convenient shortcut to construct a PyrrhoCommand and call ExecuteNonQuery on it. Will block until the connection is available. |
| PyrrhoTransaction BeginTransaction() | Start a new isolated transaction (like IDbTransaction). Will block until the connection is available. |
| <i>bool</i> Check(ch) <i>bool</i> Check(ch, rc) | Check to see if a given Versioned rowCheck string is still current, i.e. the row has not been modified by a later transaction. (See sec 5.2.3 and 8.8.21). The second version shown also tests the readCheck. (There is no need to perform a check unless the Versioned data is from a previous transaction.) |
| Close() | Close the channel to the database engine |
| <i>str</i> connectionString | Get the connection string for the connection |
| PyrrhoCommand CreateCommand() | Create an object for carrying out an Sql command (as in IDbCommand). |
| Delete(ob) | Delete (drop) a Versioned object from the database. Will block until the connection is available. |
| <i>list</i> FindAll(cls) | Retrieve all of the instances of the given Versioned class. Will block until the connection is available. |
| <i>object</i> FindOne(cls,key) | Retrieve the single instance of the given Versioned class with the given key (key is a list) Will block until the connection is available. |
| <i>list</i> FindWith(cls,cond) | Retrieve a list of instances of the given Versioned class that satisfy the given SQL condition. Will block until the connection is available. |
| <i>list</i> Get(cls,rurl) | The rurl should be the portion of a REST url following the Role component, targeting class cls in the client application. Will block until the connection is available. |
| void Open() | Open the channel to the database engine |
| Post(ob) | The object should be a new Versioned object to be entered in a base table. If autoKey is set key field(s) containing default values (0,"" etc) in ob are overwritten with suitable new value(s). Will block until the connection is available. |
| Put(ob) | The given object is an updated Versioned object that should be used to update the database. Will block until the connection is available. |
| PyrrhoConnect(cs) | Create a new PyrrhoConnect with the given connection string. Documentation about the connection string is in section 6.3, except that for Python you should supply the User field. |
| <i>list</i> Update(cls,w,u) | Specifies a Document update operation on a Versioned class containing documents. Documents matching w are updated according to the operations in u, and the set of modified objects is returned. Will block until the connection is available. |

6.8.13 PyrrhoDbType

| member | int |
|---------|-----|
| DBNull | 0 |
| Integer | 1 |
| Decimal | 2 |
| String | 3 |

| | |
|-----------|----|
| Timestamp | 4 |
| Blob | 5 |
| Row | 6 |
| Array | 7 |
| Real | 8 |
| Bool | 9 |
| Interval | 10 |
| Time | 11 |
| Date | 12 |
| UDType | 13 |
| Multiset | 14 |
| Xml | 15 |
| Document | 16 |

6.8.14 PyrrhoInterval

| Attribute | Explanation |
|-------------------|--------------------------------------|
| <i>int</i> years | The years part of the time interval |
| <i>int</i> months | The months part of the time interval |
| <i>long</i> ticks | The ticks part of the time interval |

6.8.15 (PyrrhoParameter)

Not implemented.

6.8.16 (PyrrhoParameterCollection)

Not implemented.

6.8.17 PyrrhoReader

| Attribute | Explanation |
|------------------------------------|---|
| <code>close()</code> | Close the reader |
| <i>object</i> <code>col(nm)</code> | Get the value in the column with name nm in the current row |
| <i>bool</i> <code>read()</code> | Get the next row of data into the reader. Return False if none. |
| PyrrhoRow row | Get the current row |
| PyrrhoTable schema | Get the schema for the rows |
| <i>str</i> <code>type(i)</code> | Get the subtype name of val(i) |
| <i>object</i> <code>val(i)</code> | Get the value in the ith column of the current row |

6.8.18 PyrrhoRow

| Attribute | Explanation |
|------------------------------------|---|
| <i>object</i> <code>col(nm)</code> | Get the value in the column with name nm |
| <i>str</i> <code>check</code> | Get the check string if any |
| <i>int</i> <code>version</code> | Get the row version if any |
| <i>str</i> <code>type(i)</code> | Get the subtype name of the value in the ith column |
| <i>object</i> <code>val(i)</code> | Get the value in the ith column |

6.8.19 PyrrhoTable

| Attribute | Explanation |
|--|--|
| PyrrhoColumn[] columns | A set of columns |
| <i>dict</i> cols | Maps column names to column positions |
| <i>str</i> <code>connectionString</code> | The connection string |
| PyrrhoReader <code>getReader()</code> | Used for structured values |
| PyrrhoColumn[] <code>primaryKey</code> | The columns that form the primary key if any |
| <i>str</i> <code>selectString</code> | The select string that retrieved the table |
| <i>str</i> <code>tableName</code> | The name of the table |

6.8.20 PyrrhoTransaction

| Attribute | Explanation |
|------------|---------------------------|
| commit() | Commit the transaction |
| rollback() | Roll back the transaction |

6.8.21 (SchemaAttribute)

There is no analogue to C# attributes/Java annotations in Python. But a model class definition can specify a schemaKey value.

6.8.22 Versioned

| Attribute | Explanation |
|---------------|--|
| str rowCheck | A string giving the server's row version validator. For Pyrrho this is a comma-separated list of form <i>dbname:deftpos:lasttrans</i> |
| str readCheck | A validator to check that the query used to retrieve the data would still return the same results. This is conservative: the validation will fail if the server is unable to provide this guarantee. The server takes account of all data read during the transaction that gave the validator. |

6.8.23 WebCtrl

This class is similar to WebCtrl in the AWebSvr library. Your controllers will derive from this class. The base class implementations of get, post, put, and delete do nothing and return an empty string.

| Attribute | Explanation |
|-----------------------|--|
| bool allowAnonymous() | The base implementation returns false, but anonymous logins are always allowed if no login page is supplied (Pages/Login.htm or Pages/Login.html). |
| str delete(ws, ps) | Do a Delete for the given WebSvc and parameters |
| str get(ws, ps) | Do a Get for the given WebSvc and parameters |
| str post(ws, ps) | Do a Post for the given WebSvc and parameters ([0] is the posted data) |
| str put(ws, ps) | Do a Put for the given WebSvc and parameters ([0] is the posted data) |

6.8.24 WebSvc

This class is similar to WebSvc in the AWebSvr library. In this library it is a subclass of BaseHTTPHandler. Your custom web server/service instance(s) will indirectly be subclasses of this class, so will have access to its protected fields and methods documented here. Your subclass will typically organise connection(s) to the DBMS being used. The connection can be for the service or for the request, and so should be set up in an override of the open method, using server or client credentials respectively. (The normal case with the AWebSvr library is to use an embedded DBMS, but this Python API currently supports only OSPSvr, the server edition of Pyrrho.)

| Field | Explanation |
|--------------------------|--|
| bool authenticated() | Is called to enforce authentication, if there is a login page and there is no controller for the request or the controller's allowAnonymous() returns false. The default implementation populates the WebSvc's user and password and your override can look up the credentials supplied. |
| close() | Can be overridden to release request-specific resources. |
| str getData() | Extracts the HTTP data supplied with the request: a URL component beginning with { will be converted to a Document. |
| log(verb, url, postData) | Write a log entry for the current controller method. The default implementation appends this information to Log.txt together with the user identity and timestamp. |

| | |
|---------------------------|---|
| <code>open ()</code> | Can be overridden by a subclass, e.g. to choose a database connection for the current request. The default implementation does nothing. |
| <code>str password</code> | The client's claimed credentials. See <code>authenticated()</code> |
| <code>serve()</code> | <i>Calls the requested method using the above templates. Don't call this method directly.</i> |
| <code>str user</code> | The client's claimed credentials. See <code>authenticated()</code> |

6.8.25 WebSvr

This class is similar to `WebSvr` in the `AWebSvr` library. Your custom web server should be a subclass of `WebSvr`, and `WebSvr` is a subclass of `WebSvc` and hence of `BaseHTTPHandler`. It defines the URL address (hostname and port number) for the service. If your service is multi-threaded, you can override the `Factory` method to return a new instance of your `WebSvc` subclass. Finally, call the `Server` method to start the service loop.

| Field | Explanation |
|------------------------------------|---|
| <code>WebSvc factory ()</code> | Can be overridden by a subclass to create a new service instance. The default implementation returns self (for a single-threaded server). |
| <code>server(address,port)</code> | Starts the server listening on the given address and port. |

6.9 The Java Library

The Pyrrho Java Connector `PyrrhoJC` and the `org.pyrrhodb.*` package have been significantly modified as of September 2018 as a replacement for `java.sql.*`. Work will continue to implement less-used parts of the Java SE API. There is no intention to implement any parts of Java EE, because the data model and transactions are defined by the database, not by middleware components.

`PyrrhoJC.jar` is contained in `PyrrhoJC\dist` in the Open Source Distribution of Pyrrho. It is best to extract this file where your Java project is and compile and execute with

```
javac -cp . xxxx.java
```

```
java -cp . xxxx
```

For example, assuming the OpenSource Pyrrho server is running on the local machine, and permits guest access to a database called `def`, `JCTest.java` could contain:

```
import org.pyrrhodb.*;
public class JCTest
{
    static Connection conn;
    public static void main(String args[]) throws Exception
    {
        conn = DriverManager.getConnection ("def","Student","password");
        CreateTable();
        AddData();
        ShowTable();
        conn.close();
    }

    static void CreateTable() throws Exception
    {
        Statement stmt = conn.createStatement();
        try {
            stmt.executeUpdate("drop table a");
        } catch (Exception e) {}
        stmt.executeUpdate("create table a(b int,c char)");
    }

    static void AddData() throws Exception
```

```

{
    Statement stmt = conn.createStatement();
    stmt.executeUpdate("insert into a values(1,'One'),(2,'Two')");
}
static void ShowTable()
{
    try {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("select * from a");
        while (rs.next())
        {
            System.out.println(""+rs.getInt("B")+"; "+rs.getString("C"));
        }
        rs.close();
    }
    catch(Exception e)
    {
        System.out.println(e.getMessage());
    }
}
}

```

For security reasons, the Windows version of the C# and Python client libraries supplies the current Windows account name as the user name for a session and this is generally of form DOMAIN\username. The Java API does the same (and ignores the supplied username and password supplied by the API), but unfortunately only gets the username part. This affects the ownership of the database and objects created, and limits access to the Log\$ tables, although, if the database security settings have not been defined, any login will be allowed to use the default role of the database. Currently Java clients can only use the default role.

The API currently has 42 public classes: *Bson*, *CallableStatement*, *CellValue*, *Column*, *Connection*, *Crypt*, *DBNull*, *DataType*, *DatabaseException*, *DatabaseMetaData*, *Date*, *DocArray*, *DocBase*, *Document*, *DocumentException*, *DriverManager*, *Exclude*, *FieldInfo*, *FieldType*, *Interval*, *Key*, *NoResultException*, *Numeric*, *Parameter*, *PreparedStatement*, *Procedure*, *PyrrhoArray*, *PyrrhoInputStream*, *PyrrhoInteger*, *PyrrhoOutputStream*, *PyrrhoReader*, *PyrrhoRow*, *PyrrhoTable*, *ResultSet*, *ResultSetMetaData*, *SQLException*, *SQLWarning*, *Schema*, *Statement*, *Time*, *TimeSpan*, *Timestamp*, *Versioned*. Those in *italic* are simply part of the infrastructure and you should not need to use them directly.

The following classes offer a public interface. Many of these are replacements for interfaces in java.sql.*. toString() methods are not documented here.

6.9.1 CallableStatement

In addition to the interface inherited from *PreparedStatement* (see below):

| Method | Explanation |
|--|---|
| <i>CallableStatement</i> (<i>Connection</i> con, <i>ArrayList</i> < <i>Column</i> > outs) | Constructor |
| <i>void</i> registerOutParameter(int parameterIndex,int sqlType) | Registers the Java type of an output parameter from the call.. See Types in Java SE documentation |

6.9.2 Column

This class is used in *ResultSetMetaData*.

| Field | Explanation |
|----------------------------|----------------------------------|
| <i>String</i> name | The caption for the column |
| <i>String</i> dataTypeName | The datatype name for the column |

6.9.3 Connection

For the usual way to get a *Connection*, see *DriverManager* below. Most operations on *Connections* use *Statements*. The italicised methods and fields below are not usually required.

| Constants | Explanation |
|-------------------------------------|---|
| <i>TRANSACTION_NONE</i> | <i>Ignored: All transactions are serialisable</i> |
| <i>TRANSACTION_READ_UNCOMMITTED</i> | <i>Ignored: All transactions are serialisable</i> |
| <i>TRANSACTION_READ_COMMITTED</i> | <i>Ignored: All transactions are serialisable</i> |
| <i>TRANSACTION_REPEATABLE_READ</i> | <i>Ignored: All transactions are serialisable</i> |
| <i>TRANSACTION_SERIALIZABLE</i> | (All transactions are serializable) |

| Field | Explanation |
|---|--|
| <i>PyrrhoReader rdr</i> | <i>A Connection has at most one reader</i> |
| <i>HashMap<String,DataType> dataTypes</i> | <i>Known named Pyrrho datatypes</i> |
| <i>HashMap<String,Procedure> procedures</i> | <i>Known named stored procedures/functions</i> |

| Method | Explanation |
|---|---|
| <code>void commit()</code> | Completes the current transaction. See <code>setAutoCommit</code> below. |
| <code>Connection(HashMap<String,String>)</code> | A list of properties such as Host, Port, Files, etc |
| <code>DatabaseMetaData getMetaData()</code> | See <code>DatabaseMetaData</code> below |
| <code>Statement createStatement()</code> | Statements are used for database operations. See below. |
| <code>void Delete(Versioned ob)</code> | <i>See section 6.4</i> |
| <code>Versioned[] FindAll(Class tp)</code> | <i>See section 6.4</i> |
| <code>Versioned FindOne(Class tp, Object[] w)</code> | <i>See section 6.4</i> |
| <code>Versioned[] FinalWith(Class tp, String w)</code> | <i>See section 6.4</i> |
| <code>PreparedStatement prepareStatement(String sql)</code> | The sql string can contain placeholders for parameters |
| <code>CallableStatement prepareCall(String sql)</code> | As above, but the statement can be used to call an sql stored procedure. |
| <code>void Post(Versioned ob)</code> | <i>See section 6.4</i> |
| <code>void Put(Versioned ob)</code> | <i>See section 6.4</i> |
| <code>void rollback()</code> | Abandons the current transaction. See <code>set AutoCommit</code> below. |
| <code>void setAutoCommit(boolean)</code> | Transaction control. If false, ensure that every transaction ends with <code>commit()</code> or <code>rollback()</code> . (If a transaction is aborted because of an exception, you must still call <code>rollback()</code> .) If true, transactions automatically commit after each <code>Statement</code> , and automatically rollback on exceptions. Default true. |
| <code>void setTransactionIsolation(int level)</code> | Ignored. All transactions are isolated and serializable. |

6.9.4 DBNull

| Field | Explanation |
|---------------------------|-------------------------------|
| <code>DBNull value</code> | Corresponds to SQL null value |

6.9.5 DataType

Pyrrho has its own set of abstract primitive data types, independent of operating system, hardware or locale. (See section 8.9.9)

| Constants | Value |
|------------------------|-------|
| <code>NULL</code> | 0 |
| <code>INTEGER</code> | 1 |
| <code>NUMERIC</code> | 2 |
| <code>STRING</code> | 3 |
| <code>TIMESTAMP</code> | 4 |
| <code>BLOB</code> | 5 |
| <code>ROW</code> | 6 |
| <code>ARRAY</code> | 7 |
| <code>REAL</code> | 8 |
| <code>BOOLEAN</code> | 9 |

| | |
|----------|----|
| INTERVAL | 10 |
| TIMESPAN | 11 |
| UDT | 12 |
| DATE | 13 |
| PASSWORD | 14 |
| MULTISET | 15 |

| Method | Explanation |
|---|---------------------------------------|
| <code>short fromSqlType(int sqlType)</code> | Maps Java sqlType to Pyrrho datatype. |

6.9.6 DatabaseMetaData

The many constant values defined for this class are not described here. They are not needed, since Pyrrho implements databases in a different way, and comprehensive database metadata can be obtained from the system tables using SQL.

| Method | Explanation |
|---|---|
| <code>String getDatabaseProductName()</code> | "Pyrrho DBMS" |
| <code>ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)</code> | All parameters are ignored except tableNamePattern. If it is the empty string, the same information is returned as from the query 'table "Role\$Table"'. The pattern supplied filters the table names using LIKE semantics. |

6.9.7 DatabaseException

A subclass of SQLException, see below. The method getMessage() (inherited from Throwable) contains a readable explanation of the exception.

6.9.8 Date

This wraps java.util.Date: toString() gives the date in the short form for the current locale.²²

| Field | Explanation |
|----------------------------------|-------------|
| <code>Java.util.Date date</code> | |

| Method | Explanation |
|-------------------------------------|-------------|
| <code>Date(java.util.Date d)</code> | Constructor |

6.9.9 DocArray

| Field | Explanation |
|--|-------------|
| <code>ArrayList<Object> items</code> | |

| Method | Explanation |
|---|---|
| <code>DocArray()</code> | Constructor: [] |
| <code>DocArray(String s)</code> | Constructor: The string contains [item..] |
| <code>DocArray(byte[],int pos,int end)</code> | Constructor: From Bson format |

6.9.10 Document

| Field | Explanation |
|---|---|
| <code>ArrayList<SimpleEntry<String,Object>> fields</code> | The ordering is significant so it can't be a hashmap. |

| Method | Explanation |
|---------------------------------|---|
| <code>Document()</code> | Constructor: {} |
| <code>Document(String s)</code> | Constructor: the string contains { "key": value,... }. Throws DocumentException |

²² For correctness, dates with the same short format should be considered equal.

| | |
|-----------------------------------|--|
| Document(byte[],int pos, int end) | Constructor: from Bson format. Throws DocumentException. |
|-----------------------------------|--|

6.9.11 DocumentException

This subclass of Exception is used to report parsing errors in a Document.constructor.

| Field | Explanation |
|---------|---|
| int pos | The position in the constructor parameter where the error was found |

6.9.12 DriverManager

| Method | Explanation |
|--|---|
| Connection getConnection(String url, String user, String password) | The url should be a valid Pyrrho connection string, such as "Files=myDb" (see section 6.3). The user and password parameters are ignored. The user for the connection will be obtained from the operating system (on Windows this excludes the domain identifier). The role for the connection will be the name of the first database in the connection (this is the Pyrrho default). Throws IOException. |

6.9.13 Exclude

@Exclude is an annotation for fields in a Versioned class. If you add a field to a class definition supplied by Pyrrho (see the Role\$Java system table), you *must* flag it with this annotation.

6.9.14 FieldType

@FieldType is an annotation for fields in a Versioned class. Can be omitted for standard types (see DataType). These are supplied in the results of the "Role\$Java" system table. You should not change them or use them in your own code.

| Attribute | Explanation |
|-----------------|--|
| int type() | The base DataType number (n.b. Pyrrho data type) |
| int maxLength() | The maximum length (e.g. for Char) |
| int scale() | The scale for numeric types (number of decimal places to right of decimal point) |
| String udt() | The name of a user-defined type. |

6.9.15 Interval

This type is the difference between two dates or timestamps.

| Field | Explanation |
|----------------------------|------------------------|
| int years | The number of years |
| int months | The number of months |
| long ticks | The number of ticks |
| static long TicksPerSecond | 10000000 (ten million) |

| Method | Explanation |
|---------------|--|
| String Format | Returns the Interval in ISO-9075 format. |

6.9.16 Key

This is an annotation for a key field in a Versioned class. This annotation is supplied in the Role\$Java system table. Do not modify it or use it in your own code.

| Attribute | Explanation |
|-----------|--|
| int key() | The 1-based position of this field in the primary key. |

6.9.17 NoResultException

You can throw this exception to simulate NO DATA.

6.9.18 Numeric

Not yet developed in the Java API.

| Field | Explanation |
|----------------------------|-------------|
| <i>static Numeric zero</i> | 0 |

6.9.19 Parameter

This class provides information about procedure parameters. See Procedure below.

| Field | Explanation |
|----------------------------|--------------------------------------|
| <i>String name</i> | The name of the parameter |
| <i>DataType dataType</i> | ThePyrrho data type of the parameter |
| <i>short mode</i> | One of the values below: |
| <i>static short IN</i> | 0 |
| <i>static short OUT</i> | 1 |
| <i>static short INOUT</i> | 2 |
| <i>static short RESULT</i> | 3 |

6.9.20 PreparedStatement

This is a subclass of Statement, and inherits its fields and methods. You can create a PreparedStatement with the Connection class, and provide a template for the Statement's command. This can have numeric and symbolic placeholders, called parameters. (This API is currently incomplete. You can use the Statement class directly for any given SQL string.)

| Method | Explanation |
|---|--|
| <i>void setInt(int parameterIndex, int x)</i> | Places an int value at the 1-based position shown. |
| <i>ResultSet executeQuery()</i> | Evaluates the template with the parameters that have been supplied, and calles Statement.executeQuery |
| <i>int executeUpdate()</i> | Evaluates the template with the parameters that have been supplied, and calles Statement.executeUpdate |
| <i>void setQueryTimeout(int t)</i> | Ignored. |

6.9.21 Procedure

Connection can have a list of known Procedures and Functions for the current database (obtained from the Role\$Procedure system table).

| Field | Explanation |
|--|---|
| <i>String name</i> | The name of the procedure |
| <i>ArrayList<Parameter> parameters</i> | The list of formal parameters |
| <i>DataType returns</i> | The value returned from the Function. NULL for Procedure. |

6.9.22 ResultSet

| Field | Explanation |
|-----------------------|---|
| <i>Statement stmt</i> | The statement that built this ResultSet |

| Method | Explanation |
|---|---|
| <i>void close()</i> | Must be called when you have finished with the ResultSet. |
| <i>int findColumn(String columnLabel)</i> | Returns the 1-based position of a given named column in the ResultSet |
| <i>Date getDate(int columnIndex)</i> | Returns the Date in the current row at the given 1-based column index. |
| <i>Date getDate(String columnLabel)</i> | Returns the Date in the current row in the column with the given label. |
| <i>double getDouble(int columnIndex)</i> | Returns the double in the current row at the given 1-based column index. |
| <i>double getDouble(String columnLabel)</i> | Returns the double in the current row in the column with the given label. |

| | |
|---|---|
| <i>float</i> getFloat(int columnIndex) | Returns the float in the current row at the given 1-based column index. |
| <i>float</i> getFloat(String columnLabel) | Returns the float in the current row in the column with the given label. |
| <i>int</i> getInt(int columnIndex) | Returns the int in the current row at the given 1-based column index. |
| <i>int</i> getInt(String columnLabel) | Returns the int in the current row in the column with the given label. |
| <i>Numeric</i> getNumeric(int columnIndex) | Returns the Numeric in the current row at the given 1-based column index. |
| <i>Numeric</i> getNumeric(String columnLabel) | Returns the Numeric in the current row in the column with the given label. |
| <i>Object</i> getObject(int columnIndex) | Returns the Object in the current row at the given 1-based column index. |
| <i>Object</i> getObject(String columnLabel) | Returns the Object in the current row in the column with the given label. |
| <i>ResultSetMetaData</i> getResultSetMetaData() | Get information about the ResultSet |
| <i>short</i> getShort(int columnIndex) | Returns the short in the current row at the given 1-based column index. |
| <i>short</i> getShort(String columnLabel) | Returns the short in the current row in the column with the given label. |
| <i>String</i> getString(int columnIndex) | Returns the String in the current row at the given 1-based column index. |
| <i>String</i> getString(String columnLabel) | Returns the String in the current row in the column with the given label. |
| <i>Time</i> getTime(int columnIndex) | Returns the Time in the current row at the given 1-based column index. |
| <i>Time</i> getTime(String columnLabel) | Returns the Time in the current row in the column with the given label. |
| <i>Timestamp</i> getTimestamp(int columnIndex) | Returns the Timestamp in the current row at the given 1-based column index. |
| <i>Timestamp</i> getTimespamp(String columnLabel) | Returns the Timestamp in the current row in the column with the given label. |
| <i>boolean</i> wasNull() | Returns true if the last column read had a value of NULL. (The API cannot return a null value directly) |

6.9.22 ResultSetMetaData

| Field | Explanation |
|-----------------------------------|---|
| <i>List<Column></i> columns | The columns of the ResultSet |
| <i>int[]</i> key | An array of 1-based positions of the key columns of the ResultSet |

| Method | Explanation |
|--|--|
| <i>int</i> getColumnCount() | Returns the number of columns in the ResultSet |
| <i>String</i> getColumnName(int i) | Get the name of the 1-based i-th column. |
| <i>String</i> getColumnType(int i) | Get the dataTypeName of the 1-based i-th column. |
| <i>int</i> getColumnDisplaySize(int i) | Get the display size of the 1-based i-th column (deprecated) |

6.9.23 SQLException

| Method | Explanation |
|--|--|
| <i>String</i> getErrorCode() | Gets the SQL condition code of the exception. (This is useful if you want to write a condition handler.) |
| <i>SQLException</i> getNextException() | Returns null. (There can only be one.) |
| <i>String</i> getSQLState() | Gives the same information as getErrorCode() |
| <i>str</i> connectionString | The connection string |
| <i>PyrrhoReader</i> getReader() | Used for structured values |
| <i>PyrrhoColumn[]</i> primaryKey | The columns that form the primary key if any |

| | |
|-------------------------|--|
| <i>str</i> selectString | The select string that retrieved the table |
| <i>str</i> tableName | The name of the table |

6.9.24 SQLWarning

| Method | Explanation |
|------------------------------------|--|
| <i>SQLWarning</i> getNextWarning() | Returns null. Use Statement.getWarnings instead. |
| <i>String</i> getSQLState() | Gets the SQL condition code of the warning. |

6.9.25 Schema

The @Schema annotation is added to a table, view or structured type definition in the Role\$Java system table.

| Field | Explanation |
|------------------|--|
| <i>int</i> key() | The last change made to the schema affecting this structure. An exception will be raised if the database detects use of an incorrect Schema key. |

6.9.26 Statement

| Field | Explanation |
|------------------------|---|
| <i>Connection</i> conn | The connection that created the Statement |

| Method | Explanation |
|---|---|
| <i>void</i> cancel() | Not implemented |
| <i>void</i> close() | Closes any reader associated with the Statement |
| <i>boolean</i> execute(String sql) | Deprecated |
| <i>ResultSet</i> executeQuery(String sql) | Creates a ResultSet for a given SELECT or TABLE statement.. Throws Exception |
| <i>int</i> executeUpdate(String sql) | Returns the number of rows directly affected by execution of some other sort of SQL statement (does not include effects of triggers). Throws Exception. |
| <i>ResultSet</i> getResultSet() | Returns the ResultSet |
| <i>int</i> getUpdateCount() | Returns the number of rows directly affected. |
| <i>SQLWarning</i> getWarnings() | Get and remove the first warning for this Statement |
| <i>void</i> setCursorName(String name) | Not implemented |

6.9.27 Time

A wrapper for java.util.Date.

6.9.28 TimeSpan

A time interval measured in miniticks (ten-millionths of a second).

6.9.29 Timestamp

Currently a wrapper for java.sql.Timestamp.

6.8.22 Versioned

The base class for classes known to the database for the object-oriented API (See section 6.4). The Versioned class has no public fields or methods, but is used for transaction safety. If the database finds a version mismatch, it will raise a DatabaseException that explains the problem and mark the transaction as non-committable. The application then needs to decide how to proceed.

6.10 SWI-Prolog

The Open Source Edition of Pyrrho also comes with some support for SWI-Prolog. This is contained in a module pyrrho.pl which is part of the distribution. The code is at an early stage, so comments are welcome. The following documentation uses the conventions of the SWI-Prolog project.

The interface with SWI-Prolog is implemented by providing SWI-Prolog support for the Pyrrho protocol (section 8.9). The following publicly-visible functions are currently supported:

| | |
|---|---|
| connect (<i>-Conn</i> , <i>+ConnectionString</i>) | Establish a connection to the Open Source Pyrrho server. Conn has the form conn (<i>InStream</i> , <i>InBuffer</i> , <i>OutStream</i> , <i>OutBuffer</i>). Codes in <i>OutBuffer</i> are held in reverse order. |
| sql_reader (<i>+Conn0</i> , <i>-Conn1</i> , <i>+SQLString</i> , <i>-Columns</i>) | Like ExecuteReader on the connection. Conn0. Conn1 is the updated connection. Columns is a list of entries of form column (<i>Name</i> , <i>Type</i>) . |
| read_row (<i>+Conn0</i> , <i>Conn1</i> , <i>+Columns</i> , <i>-Row</i>) | Reads the next row (fails if there is no next row) from the connection Conn0. Conn1 is the updated connection. Columns is the column list as returned from sql_reader. Row is a list of corresponding values for the current row. |
| close_reader (<i>+Conn</i>) | Closes the reader on connection Conn. |
| field (<i>+Columns</i> , <i>+Row</i> , <i>+Name</i> , <i>-Value</i>) | Extracts a named value from a row. The atom null is used for null values. |

7. SQL Syntax for Pyrrho

The following details are provided here for convenience. The syntax shown is merely suggestive in relation to semantics. Full details may be found in SQL2016, but not all of the details in SQL2016 are relevant to Pyrrho. In addition, GRANT OWNER, ALTER .. TO, and SET statements are Pyrrho specific.

In this section capital letters indicate key words: those that are reserved words are shown in a sans-serif font. Tokens such as id, int, string are shown as all lower case words. Mixed case is used for grammar symbols defined in the following productions. The characters = . [] { } are part of the production syntax. Characters that appear in the input are enclosed in single quotes, thus ' '. Where an identifier representing an object name is required, and the type of object is not obvious from the context, locutions such as *Role_id* are used.

There are three tokens: xmlname, iri and xml, which are used in XPath below, which are extensions to SQL2016. These tokens are not enclosed in single or double quotes, but may contain string literals that are enclosed in quotes. xmlname represents a case-sensitive sequence of letters and digits, iri is an IRI enclosed in <> and xml represents any Xml content not including an exposed ,] or). In SQL all string literals are enclosed in single quotes, case-sensitive identifiers or containing special characters are enclosed in double quotes.

7.1 Statements

Sql = SqlStatement [';'] .

```
SqlStatement =  Alter
               |  BEGIN TRANSACTION [WITH PROVENANCE string ]
               |  Call
               |  COMMIT
               |  CreateClause
               |  CursorSpecification
               |  DeleteSearched
               |  DropStatement
               |  Grant
               |  Insert
               |  Rename
               |  Revoke
               |  ROLLBACK
               |  SET AUTHORIZATION '=' CURATED
               |  SET PROFILING '=' Value
               |  SET ROLE id [FOR DATABASE id]
               |  SET TIMEOUT '=' int
               |  UpdateSearched
               |  HTTP HttpRest .
```

The above statements can be issued at command level. You SELECT multiple rows from tables using the CursorSpecification. Inside procedures and functions there is a different set. (Note that “direct SQL” statements are in both lists.)

Apart from SET ROLE, the above SET statements are available only to the database owner. SET AUTHORIZATION = CURATED makes all further transaction log information PUBLIC (it is not reversible).

```
Statement =   Assignment
               |  Call
               |  CaseStatement
               |  Close
               |  CompoundStatement
               |  BREAK
               |  Declaration
               |  DeletePositioned
```

```

|      DeleteSearched
|      Fetch
|      ForStatement
|      GetDiagnostics
|      IfStatement
|      Insert
|      ITERATE label
|      LEAVE label
|      LoopStatement
|      Open
|      Repeat
|      RETURN Value
|      ROLLBACK23
|      SelectSingle
|      Signal
|      UpdatePositioned
|      UpdateSearched
|      While
|      Http HttpRest .

```

Http = HTTP [*user_Value* [': ' *password_Value*]] .

The optional details are used for basic authentication. HTTPS may be specified in the url, and the default authentication is that of the current or defining user.

```

HttpRest =      (ADD|UPDATE) url_Value data_Value [AS mime_string] [WhereClause]
|      DELETE url_Value [data_Value [AS mime_string]] [WhereClause] .

```

Here ADD and UPDATE are used as the SQL analogues of POST and PUT. The WhereClause in this syntax is an alternative to using an expression for the uri, but depending on the web service being accessed it may be limited to a conjunction of equality conditions. (For Http GET see Value.)

7.2 Data Definition

As is usual for a practical DBMS Pyrrho's Alter statements are richer than SQL2016.

```

Alter =      ALTER DOMAIN id AlterDomain
|      ALTER FUNCTION id '(' Parameters ') RETURNS Type AlterBody
|      ALTER PROCEDURE id '(' Parameters ') AlterBody
|      ALTER Method AlterBody
|      ALTER TABLE id AlterTable
|      ALTER TYPE id AlterType
|      ALTER VIEW id AlterView .

```

Procedures, functions and methods are distinguished by their name and arity (number of parameters)²⁴.

Method = MethodType METHOD *id* '(' Parameters ')' [RETURNS Type] [FOR *id*].

Parameters = Parameter {', ' Parameter } .

Parameter = *id* Type .

²³ By design in Pyrrho, the execution of ROLLBACK causes immediate exit of the current transaction with SQLSTATE 40000, and premature COMMIT is not supported, so that while ROLLBACK is in both lists above, COMMIT is only in one.

²⁴ See SQL2003-02 11.50 Syntax Rules 16.

The specification of IN, OUT, INOUT and RESULT is not (yet) supported.

MethodType = [OVERRIDING | INSTANCE | STATIC | CONSTRUCTOR] .

The default method type is INSTANCE. All OVERRIDING methods are instance methods.

Classification = SECURITY Level .

Level = LEVEL *level_id* ['-' *level_id*] [GROUPS { *id* }] [REFERENCES { *id* }] .

Classification is added to SQL2013 (*level_id* = D,C,B or A) and can only be specified by the database owner: D is the default. See section 3.4.2.

AlterDomain = SET DEFAULT Default

| DROP DEFAULT
| TYPE Type
| AlterCheck .

AlterBody = AlterOp { ',' AlterOp } .

AlterOp = TO id
| Statement
| [ADD|DROP] { Metadata } .

Default = Literal | DateTimeFunction | CURRENT_USER | CURRENT_ROLE | NULL |
ARRAY('') | MULTISSET('').

AlterCheck = (ADD|DROP) CheckConstraint

| [ADD|DROP] { Metadata }
| DROP CONSTRAINT id .

Note that anonymous constraints can be dropped by finding the system-generated id in the Sys\$TableCheck, Sys\$ColumnCheck or Sys\$DomainCheck table (see section 8.1).

CheckConstraint = [CONSTRAINT id] CHECK '(' [XMLOption] SearchCondition ')' .

XMLOption = WITH XMLNAMESPACES '(' XMLNDec { ',' XMLNDec } ')'

XMLNDec = (string AS id) | (DEFAULT string) | (NO DEFAULT) .

The following standard namespaces and prefixes are predefined:

'http://www.w3.org/1999/02/22-rdf-syntax-ns#' AS rdf
'http://www.w3.org/2000/01/rdf-schema#' AS rdfs
'http://www.w3.org/2001/XMLSchema#' AS xsd
'http://www.w3.org/2002/07/owl#' AS owl

AlterTable = TO id
| ADD ColumnDefinition
| ALTER [COLUMN] id AlterColumn
| DROP [COLUMN] id DropAction
| (ADD|DROP) (TableConstraintDef | VersioningClause)
| SET Cols REFERENCES id [Cols] [USING (id|('Values'))] ReferentialAction
| ALTER PERIOD id TO id
| DROP TablePeriodDefinition DropAction
| Classification | Enforcement
| AlterCheck
| [ADD|DROP] Metadata { Metadata } .


```

AlterColumn =  TO id
               |      POSITION int
               |      SET ((NOT NULL)|ColumnConstraint )
               |      DROP ((NOT NULL)|ColumnConstraint ) DropAction
               |      AlterDomain
               |      Classification
               |      GenerationRule
               |      [ADD|DROP DropAction] { Metadata } .

```

When columns are renamed, Pyrrho cascades the change to SQL referring to the columns.

```

AlterType =    TO id
               |      ADD ( Field | Method )
               |      DROP ( Field_id | Routine) DropAction
               |      Classification
               |      Representation
               |      [DROP] { Metadata }
               |      ALTER Field_id AlterField .

```

Other details of a Method can be changed with the ALTER METHOD statement (see Alter above). A sensitive type cannot be altered to a non-sensitive type.

Field = id Type [DEFAULT Value] Collate .

```

AlterField =   TO id
               |      [DROP] DropAction { Metadata }
               |      TYPE Type
               |      SET DEFAULT Value
               |      DROP DEFAULT .
AlterView =    SET SOURCE TO QueryExpression
               |      TO id
               |      [ADD|DROP] { Metadata } .

```

```

Metadata =     ATTRIBUTE | CAPTION | ENTITY | HISTOGRAM | LEGEND | LINE | POINTS |
               PIE | X | Y | JSON | CSV | INVERTS | MONOTONIC | string | iri | id.

```

This syntax is a Pyrrho extension. The iri can only be set or changed by the object owner, but most of the options affect output for a role in Pyrrho's Web service, so can be added or modified by the role administrator. By default the output is an HTML table. Attribute and Entity if present set a preference for XML output for structured data. Histogram, Legend, Line, Points, Pie (for table, view or function metadata), Caption, X and Y (for column or subobject metadata) specify JavaScript added to HTML output to draw the data visualisations specified. The string is usually for a description, and for X and Y columns is used to label the axes of charts. For RestViews, the iri is the url for the view²⁵. For INVERTS the id should be the name of the function being inverted²⁶.

```

AddPeriodColumnList =  ADD  [COLUMN]  Start_ColumnDefinition  ADD  [COLUMN]
End_ColumnDefinition .

```

```

Create = CREATE ROLE id [Description_string]
|  CREATE DOMAIN id [AS] DomainDefinition [Classification]
|  CREATE FUNCTION id '('Parameters')' RETURNS Type {Metadata} Statement27

```

²⁵ The definer of the view may supply this in a string (for backwards compatibility).

²⁶ Pyrrho uses such information automatically in the implementation of updatable views and joins.

²⁷ Functions that return tables have an explicit row type, so the table value returned by the Statement should explicitly alias columns to match the returns clause, in case table columns are changed later.

```
| CREATE ORDERING FOR UDType_id (EQUALS ONLY|ORDER FULL) BY Ordering
| CREATE PROCEDURE id '(' Parameters ')' Statement
| CREATE Method Statement
| CREATE TABLE id TableContents [Classification][Enforcement] {Metadata}
| CREATE TRIGGER id (BEFORE|AFTER) Event ON id [ RefObj ] Trigger
| CREATE TYPE id ((UNDER id )|AS Representation) [Classification] [ Method {',' Method} ]
| CREATE VIEW id ViewDefinition
| CREATE XMLNAMESPACES XMLNDec { ',' XMLNDec } .
```

Method bodies in SQL2016 are specified by CREATE METHOD once the type has been created...In Pyrrho types UNDER or Representation must be specified (not both). CREATE XMLNAMESPACES is for creating a persistent association of namespace uris with identifiers. Classification and Enforcement can only be set by the database owner (see section 3.4.2).

Enforcement = SCOPE [READ] [INSERT] [UPDATE] [DELETE] .

Representation = (StandardType|Table_id|(' Field {',' Field }')) {CheckConstraint} .

DomainDefinition = StandardType [DEFAULT Default] { CheckConstraint } Collate .

Ordering = (RELATIVE|MAP) WITH Routine
| STATE .

TableContents = '(' TableClause {',' TableClause } ')' { VersioningClause }
| OF *Type_id* ['(' TypedTableElement {',' TypedTableElement} ')']
| AS Subquery .

VersioningClause = WITH SYSTEM VERSIONING .

TableClause = ColumnDefinition {Metadata} | TableConstraint | TablePeriodDefinition .

ColumnDefinition = id Type [DEFAULT Default] {ColumnConstraint|CheckConstraint} Collate
| id GenerationRule
| id *Table_id* '.' *Column_id*.

The last version is a convenience form for lookup tables, e.g. if a.b has domain int then a.b is a shorthand for int check (value in (select b from a)).

GenerationRule = GENERATED ALWAYS AS '(' Value ')' [UPDATE '(' Assignments ')']
| GENERATED ALWAYS AS ROW (START|END) .

The update option here is an innovation in Pyrrho.

ColumnConstraint = [CONSTRAINT id] ColumnConstraintDef .

ColumnConstraintDef = NOT NULL
| PRIMARY KEY
| REFERENCES id [Cols] [USING (id|('Values'))] { ReferentialAction }
| UNIQUE
| DEFAULT Value
| Classification .

The Using expression here is an extension to SQL2016 behaviour, allowing a row expression or the name of an adapter function. See section 5.2.2. A column default value overrides a domain default value.

TableConstraint = [CONSTRAINT id] TableConstraintDef .

TableConstraintDef= UNIQUE Cols
| PRIMARY KEY Cols

| FOREIGN KEY Cols REFERENCES id [Cols] [USING (id|('Values'))]
{ ReferentialAction } .

The Using expression here is an extension to SQL2016 behaviour allowing a row expression or the name of an adapter function. See section 5.2.2.

TablePeriodDefinition= PERIOD FOR PeriodName '(' *Column_id* ',' *Column_id* ')'

PeriodName = SYSTEM_TIME | id .

TypedTableElement = ColumnOptionsPart | TableCnstraint .

ColumnOptionsPart = id WITH OPTIONS '(' ColumnOption {' , ' ColumnOption } ')'

ColumnOption = (SCOPE *Table_id*) | (DEFAULT Value) | ColumnConstraint .

Values = Value {' , ' Value } .

Cols = '(' ColRef {' , ' ColRef } [' , ' PERIOD *ApplicationTime_id*] ')'

The period syntax here can only be used in a foreign key constraint declaration where both tables have application time period definitions, and allows them to be matched up.

ColRef = *Column_id* { '.' *Field_id* [AS Type]}.

The *Field_id* syntax is Pyrrho specific and can be used to reference fields of structured types or documents.

ReferentialAction = ON (DELETE|UPDATE) (CASCADE| SET (DEFAULT|NULL)|RESTRICT) .

The default ReferentialAction is RESTRICT.²⁸

ViewDefinition = [ViewSpec] AS (QueryExpression | GET [USING *Table_id*]) {Metadata} .

The resulting view may be updatable using UPDATE, DELETE and INSERT statements. The GET syntax here is for the RestView feature of Pyrrho²⁹.

ViewSpec = Cols | OF *Type_id* | OF Representation.

The third syntax here is to define the row type for RESTViews.

TriggerDefinition = TRIGGER id (BEFORE|(INSTEAD OF)| AFTER) Event ON id [RefObj] Trigger .

Event = INSERT | DELETE | (UPDATE [OF id {' , ' id }]) .

RefObj = REFERENCING { (OLD|NEW)[ROW|TABLE][AS id] } .

In this syntax, the default is ROW; TABLE cannot be specified for a BEFORE trigger; OLD cannot be specified for an INSERT trigger; NEW cannot be specified for a DELETE trigger.

Trigger = FOR EACH (ROW|STATEMENT [DEFERRED]) [TriggerCond] (Statement | (BEGIN ATOMIC Statements END)) .

TriggerCond = WHEN '(' SearchCondition ')'

DropStatement = DROP DropObject DropAction .

²⁸ The SQL standard erroneously specifies that the default should be NO ACTION. This option is not available in Pyrrho.

²⁹ For AS GET url, the url string is supplied in the Metadata syntax. Explicit column names can be specified using the extended ViewSpec in this section. For AS GET USING id the specified USING table gives some data identifying contributing servers including a primary key and the URL of the contribution as the last column. The row type of the Representation should consist of the columns of the USING table (except the last), and the remaining columns must match the contributed data.

```
DropObject =      ObjectName
                |      ORDERING FOR id
                |      ROLE id
                |      TRIGGER id
                |      XMLNAMESPACES (id|DEFAULT) {',' (id|DEFAULT) }
                |      INDEX id30 .
```

```
DropAction = | RESTRICT | CASCADE .
```

The default DropAction is RESTRICT.

```
Rename =SET ObjectName TO id .
```

7.3 Access Control

```
Grant =          GRANT Privileges TO GranteeList [ WITH GRANT OPTION ]
                |          GRANT Role_id { ',' Role_id } TO GranteeList [ WITH ADMIN OPTION ]
                |          GRANT Level TO user_id .
```

Grant can only be used in single-database connections (section 3.4). For roles see section 5.5. Clearance levels (D to A) can only be applied to users by the database owner (D is the default).

```
Revoke =         REVOKE [GRANT OPTION FOR] Privileges FROM GranteeList
                |         REVOKE [ADMIN OPTION FOR] Role_id { ',' Role_id } FROM GranteeList .
```

Revoke can only be used in single-database connections. Revoke withdraws the specified privileges in a cascade, irrespective of the origin of any privileges held by the affected grantees: this is a change to SQL2016 behaviour. (See also sections 5.5 and 7.13.)

```
Privileges = ObjectPrivileges ON ObjectName
                |          PASSWORD [id] [FOR Role_id] .
```

The Password privilege (Pyrrho specific) is for access to the database using HTTP, and can only be granted by the database owner. If the password field is blank it will be set by the next request from this user. The optional role identifier provides an initial role for access and implies a grant of the role to the user.

```
ObjectPrivileges = ALL PRIVILEGES | Action { ',' Action } .
```

```
Action =        SELECT [ '(' id { ',' id } ')' ]
                |        DELETE
                |        INSERT [ '(' id { ',' id } ')' ]
                |        UPDATE [ '(' id { ',' id } ')' ]
                |        REFERENCES [ '(' id { ',' id } ')' ]
                |        USAGE
                |        TRIGGER
                |        EXECUTE
                |        OWNER .
```

The owner privilege (Pyrrho-specific) can only be granted by the owner of the object (or the database) and results in a transfer of ownership of that object to a single user or role (not PUBLIC).. Ownership always implies grant option for the owner privilege. References here can be to columns, methods, fields or properties depending on the type of object referenced by the objectname (usage is for domains).

³⁰ Non-SQL, for supporting the MongoDB service.

```

ObjectName =  TABLE id
              |      DOMAIN id
              |      TYPE id
              |      Routine
              |      VIEW id .

```

```
GranteeList = PUBLIC | Grantee { ‘,’ Grantee } .
```

```
Grantee =      [USER] id
              |      ROLE id .
```

See section 5.5 for the use of roles in Pyrrho.

```

Routine =      PROCEDURE id [DataTypeList]
              |      FUNCTION id [DataTypeList]
              |      [ MethodType ] METHOD id [DataTypeList] [FOR id ]
              |      TRIGGER id .

```

```
DataTypeList = ‘(‘Type, {‘,’ Type }’)’ .
```

7.4 Type

```
Type =          (StandardType | DefinedType | Domain_id | Type_id | REF’(‘TableReference’))
                [SENSITIVE] [Level].
```

Any type can be declared sensitive: this property is silently inherited by values, columns, table and views. A non-sensitive object cannot receive a sensitive value.

```
StandardType =  BOOLEAN | CharacterType | FloatType | IntegerType | LobType | NumericType |
DateTimeType | IntervalType | XML | PASSWORD | DOCUMENT | DOCARRAY.
```

The last three types are Pyrrho-specific: Password values show as `*****`, Document is as in <http://bsonspec.org>, DocArray is for the array variant used in Bson. Documents and DocArrays are transmitted to clients as subtypes of `byte[]` data, using Bson format. All three types have automatic conversion from strings: Json to Bson for Document and DocArray. Documents are considered equal if corresponding fields match³¹.

```
CharacterType = (([NATIONAL] CHARACTER) | CHAR | NCHAR | VARCHAR) [VARYING] [‘(‘int
‘)’] [CHARACTER SET id ] Collate .
```

All of these are Unicode in Pyrrho³².

```
Collate  =      [ COLLATE id ] .
```

There is no need to specify COLLATE UNICODE, since this is the default collation. COLLATE UCS_BASIC is supported but deprecated. Other CultureInfo strings (in double quotes) are supported depending on the current version of the .NET libraries: since Windows 10 any valid BCP-47 language tag can be used. This determines comparison of strings and conversion from dates etc.

```
FloatType =      (FLOAT|REAL|DOUBLE PRECISION) [‘(‘int’,‘int’)’] .
```

The names here are regarded as equivalent in Pyrrho³³.

³¹ This is extremely useful though counter-intuitive, as the empty document is “equal” to every other document!

³² The SQL standard says that the national character set is implementation-defined. Pyrrho uses Unicode strings of arbitrary length. The length integer is a constraint during query processing, but strings in the physical database are not truncated. Note that the SQL standard does not include keywords TEXT or NVARCHAR found in some systems.

³³ The SQL standard states at sec 6.1 SR28-30 that the maximum precision of these types is implementation-defined. Internally Pyrrho’s maximum precision for a mantissa is 2040 bits and for an exponent, 63 bits.

IntegerType = INT | INTEGER | BIGINT | SMALLINT .

All these integer types are regarded as equivalent in Pyrrho³⁴.

LobType = ([NATIONAL] CHARACTER |BINARY) LARGE OBJECT | BLOB | CLOB | NCLOB .

National is ignored, the character large object types are regarded as equivalent to CHAR since they represent unbounded character strings, and of course BINARY LARGE OBJECT is the same as BLOB.

NumericType = (NUMERIC|DECIMAL|DEC) [‘(‘int’,’int’)] .

The names here are regarded as equivalent in Pyrrho³⁵.

DateTimeType = (DATE | TIME | TIMESTAMP) ([IntervalField [TO IntervalField]] | [‘(‘ int ’)’]).

The use of IntervalFields when declaring DateTimeType is an addition to the SQL standard.

IntervalType = INTERVAL IntervalField [TO IntervalField] .

IntervalField = YEAR | MONTH | DAY | HOUR | MINUTE | SECOND [‘(‘ int ’)’] .

DefinedType = (ROW|TABLE) Representation

| DataTypeList
| Type ARRAY
| Type MULTISSET .

The TABLE alternative here is a Pyrrho extension to SQL2016, but currently there is no difference between ROW and TABLE. DataTypeList is an anonymous row type (no column names), also specific to Pyrrho.

7.5 Data Manipulation

Insert = INSERT [WITH PROVENANCE string] [XMLOption] INTO *Table_id* [Cols] (TableValue | DEFAULT VALUES) [Classification] .

The VALUES keyword is mandatory if you are providing an explicit TableValue (see section 7.7). For example INSERT INTO t VALUES (4,5) , or INSERT INTO t (SELECT c,d FROM e). Provenance is a sort of row metadata that cannot be updated: it can be viewed by anyone with select permission for the table. As in 7.2, only the database owner is permitted to provide a classification: otherwise, if the insert succeeds, the classification of the row is determined by the clearance of the current user, and may differ from the classification of other rows in the table. The column list if present names the columns from the table for which values are provided: otherwise values must be provided for all columns.

UpdatePositioned = UPDATE [XMLOption] *Target_id* Assignment WHERE CURRENT OF *Cursor_id* .

UpdateSearched = UPDATE [XMLOption] *Target_id* Assignment [WhereClause] .

DeletePositioned = DELETE [XMLOption] FROM *Target_id* WHERE CURRENT OF *Cursor_id* .

DeleteSearched = DELETE [XMLOption] FROM *Target_id* [WhereClause] .

³⁴ The SQL standard states at . sec 6.1 SR28 “The precision of SMALLINT shall be less than or equal to the precision of INTEGER, and the precision of BIGINT shall be greater than or equal to the precision of INTEGER” Internally Pyrrho’s precision is 2040 bits. In clients integer values are generally represented as 64-bit integers. The Pyrrho protocol supplies larger values as strings.

³⁵ The SQL standard states at 6.1 SR25 that the maximum precision of these types is implementation-defined. Internally Pyrrho’s maximum precision is 2040 bits with a 64-bit maximum for the scale.

In these four definitions *Target* can be a table or view.

CursorSpecification = [XMLOption] QueryExpression .

QueryExpression = QueryTerm {(UNION|EXCEPT)[ALL|DISTINCT] QueryTerm} [OrderByClause]
[FetchFirstClause] .

DISTINCT is the default and discards duplicates from both operands.

QueryTerm = QueryPrimary { INTERSECT [ALL | DISTINCT] QueryPrimary } .

DISTINCT is the default.

QueryPrimary = SimpleTable | '(' QueryExpression ')' .

SimpleTable = QuerySpec | TableValue | TABLE id .

QuerySpec = SELECT [ALL | DISTINCT] SelectList TableExpression [FOR UPDATE] .

FOR UPDATE is ignored by Pyrrho, and is allowed in the syntax only for compatibility with other DBMS.

SelectList = SelectItem { ',' SelectItem } .

SelectItem = [Col '.' '*' | Scalar [AS id] | RowValue '.' '*' [AS Cols].

Alias = [[AS] id [Cols]] .

The id is an alias for the referenced table, and the column list if present selects columns from it.

TimePeriodSpecification = AS OF Scalar
| BETWEEN [ASYMMETRIC|SYMMETRIC] Scalar AND Scalar
| FROM Scalar TO Scalar .

This syntax is slightly more general than in SQL2016.

Subquery = '(' QueryExpression ')' .

Subqueries return different sorts of values depending on the context, including simple values (scalars, structures, arrays, multisets, etc), rows and tables.

JoinedTable = TableReference CROSS JOIN TableFactor
| TableReference NATURAL [JoinType] JOIN TableFactor
| TableReference[JoinType]JOIN TableReference ((USING '('Cols'))(ON SearchCondition)) .

JoinType = INNER | (LEFT | RIGHT | FULL) [OUTER] .

SearchCondition = BooleanExpr .

OrderByClause = ORDER BY OrderSpec { ',' OrderSpec } .

OrderSpec = Scalar [ASC | DESC] [NULLS (FIRST | LAST)] .

The default order is ascending, nulls first.

FetchFirstClause = FETCH FIRST [int] (ROW|ROWS) ONLY .

XmlColumns = COLUMNS XmlColumn { ',' XmlColumn } .

XmlColumn = id Type [DEFAULT Scalar] [PATH str] .

7.6 Scalar Expressions

Value = Scalar | RowValue | TableValue .

Scalar = Literal

```

|      Scalar BinaryOp Scalar
|      '- Scalar
|      '(' Scalar ')'
|      Scalar Collate
|      Scalar '[' Scalar ']'
|      Scalar AS Type
|      ColumnRef
|      VariableRef
|      Scalar_Subquery
|      (SYSTEM_TIME|Period_id|(PERIOD'('Scalar','Scalar'))
|      VALUE
|      Scalar '.' Field_id
|      Scalar_MethodCall
|      NEW Constructor_MethodCall
|      Scalar_FunctionCall
|      Document
|      DocArray
|      Xml
|      (MULTISET |ARRAY) (('['Value { ',' Value } ']| Table_Subquery)
|      TREAT '(' Scalar AS sub_Type ')'
|      CURRENT_USER
|      CURRENT_ROLE .

```

The VALUE keyword is used in Check Constraints, A scalar subquery must have exactly one column and return a single value. The explicit list option for multiset and array cannot directly contain table expressions. A scalar MethodCall or FunctionCall does not return a table. Collate if specified applies to an immediately preceding Boolean expression, affecting comparison operands etc. The AS syntax in Scalar AS Type is allowed only in parameter lists and methodcalls.

Document = '{ [keyname ':' DocValue { ',' keyname ':' DocValue }] }'.

Keynames are case-sensitive and should be enclosed in single or double quotes.

DocArray = '[' [DocValue { ',' DocValue }]]' .

DocValue = Scalar | doublequotedstring .

To avoid being parsed as a doublequotedstring, in a DocValue a double-quoted identifier needs to be part of a larger expression such as a dotted identifier chain.³⁶

Xml = '<' XmlName { XmlAttr } '>' Scalar '</' same_XmlName '>'
 | '<' XmlName { XmlAttr } '/>' .

XmlAttr = XmlName '=' DocValue .

XmlName = [keyname ':']keyname .

This exposed Xml syntax is a Pyrrho extension (and is different from the XML support specified in SQL2016). As above keynames are case-sensitive but for XML they should not be enclosed in double-quotes: the character set is specified in XML standards³⁷.

BinaryOp = '+' | '-' | '*' | '/' | '||' | MultisetOp .

|| is used in array and string concatenation.

VariableRef = { Scope_id '.' } Variable_id .

ColumnRef = [TableOrAlias_id '.'] ColRef

³⁶ In Mongo, keynames and strings starting with \$ have special meanings and can be used to refer to values in the current context (e.g. "\$a.b"). Mongo is available as an option in the source code.

³⁷ <http://www.w3.org/TR/REC-xml/>


```

|      TableOrAlias_id '.' (PROVENANCE| CHECK)
|
|      SECURITY .

```

The use of the SECURITY, PROVENANCE and CHECK pseudo-columns is a change to SQL2016 behaviour. CHECK is a row versioning cookie derived from a string type, which is columns are read-only and accessible by anyone with select permission for the table.. SECURITY is reserved to the database owner (security administrator) and can be set to a value of type Level (see below).

MultisetOp = MULTiset (UNION | INTERSECT | EXCEPT) (ALL | DISTINCT) .

```

Literal =      int
|             float
|             string
|             TRUE | FALSE
|             'X' '' { hexit } ''
|             DATE date_string
|             TIME time_string
|             TIMESTAMP timestamp_string
|             INTERVAL ['-'] interval_string IntervalQualifier
|             Level .

```

Strings are enclosed in single quotes. Two single quotes in a string represent one single quote. Hexits are hexadecimal digits 0-9, A-F, a-f and are used for binary objects. Level literal can only be used by the database owner.

Dates, times and intervals use string (single quoted) values and are not locale-dependent. For full details see SQL2016: e.g.

- a date has format like DATE 'yyyy-mm-dd' ,
- a time has format like TIME 'hh:mm:ss' or TIME 'hh:mm:ss.sss' ,
- a timestamp is like TIMESTAMP 'yyyy-mm-dd hh:mm:ss.ss',
- an interval is like e.g.
 - INTERVAL 'yyy' YEAR,
 - INTERVAL 'yy-mm' YEAR TO MONTH,
 - INTERVAL 'm' MONTH,
 - INTERVAL 'd hh:mm:ss' DAY(1) TO SECOND,
 - INTERVAL 'sss.ss' SECOND(3,2) etc.

The SQL20111 specifies that intervals cannot have a mixture of year-month and date-second fields.

IntervalQualifier = StartField TO EndField

```

|      DateTimeField .

```

StartField = IntervalField ['(' int ')'] .

EndField = IntervalField | SECOND ['(' int ')'] .

DateTimeField = StartField | SECOND ['(' int [',' int] ')'] .

The ints here represent precision for the leading field and optionally for seconds the fraction part.

IntervalField = YEAR | MONTH | DAY | HOUR | MINUTE .

7.7 Query Expressions

```

TableValue =  VALUES '(' Scalar { ',' Scalar } ') ' { ',' '(' Scalar { ',' Scalar } ') ' }
|             QueryExpression

```

```

|      Table_Subquery
|      Http GET url_Scalar [ AS mime_string ] [WhereClause].

```

The mime string is used for retrieval of a particular content type from the server (Json is the default). The WhereClause in the Http GET syntax is an alternative to using an expression for the uri, but depending on the web service being accessed may be limited to a conjunction of equality conditions.

```

RowValue =      [ROW] '(' Scalar { ',' Scalar } ')'
|
|      Scalar .

```

The Scalar option here constructs a row with a single column whose value is the given scalar value.³⁸

```

TableExpression = [FromClause] [WhereClause] [GroupByClause] [HavingClause] [WindowClause] .

```

GroupByClause and HavingClause are used with aggregate functions. WindowClause is used with window functions. From v7 the FromClause can be omitted.

```

FromClause =      FROM TableReference { ',' TableReference } .

```

```

WhereClause = WHERE BooleanExpr .

```

```

GroupByClause = GROUP BY [DISTINCT|ALL] GroupingSet { ',' GroupingSet } .

```

```

GroupingSet = OrdinaryGroup | GroupingSpec | '(' ')'.

```

```

OrdinaryGroup = ColumnRef [Collate] | '(' ColumnRef [Collate] { ',' ColumnRef [Collate] } ')' .

```

```

GroupingSpec = GROUPING SETS '(' GroupingSet { ',' GroupingSet } ')' .

```

```

HavingClause = HAVING BooleanExpr .

```

```

WindowClause = WINDOW WindowDef { ',' WindowDef } .

```

Window clauses are only useful with window functions, which are discussed in section 7.7.

```

WindowDef = id AS '(' WindowDetails ')' .

```

```

WindowDetails = [Window_id] [ PartitionClause] [ OrderByClause ] [ WindowFrame ] .

```

```

PartitionClause = PARTITION BY OrdinaryGroup .

```

```

WindowFrame = (ROWS|RANGE) (WindowStart|WindowBetween) [ Exclusion ] .

```

```

WindowStart = ((Scalar | UNBOUNDED) PRECEDING) | (CURRENT ROW) .

```

```

WindowBetween = BETWEEN WindowBound AND WindowBound .

```

```

WindowBound = WindowStart | ((Scalar | UNBOUNDED) FOLLOWING ) .

```

```

Exclusion = EXCLUDE ((CURRENT ROW)|GROUP|TIES|(NO OTHERS)) .

```

```

TableReference = TableFactor Alias | JoinedTable .

```

```

TableFactor =      Table_id [FOR SYSTEM_TIME [TimePeriodSpecification ]]
|
|      View_id
|      ROWS '(' int [ ',' int ] ')'
|      Table_FunctionCall
|      Table_Subquery
|      '(' TableReference ')'
|      TABLE '(' Scalar ')'
|      UNNEST '(' Scalar ')'

```

³⁸ This syntax is called <row value constructor> in the SQL standard (section 7.1).

XMLTABLE ‘(‘ [XMLOption] xml [PASSING NamedValue {‘,’ NamedValue}]
 XmlColumns ‘)’
 | DocArray .

ROWS(..) is a Pyrrho extension (for table and cell logs), and the last option above is also Pyrrho-specific and allows a specific list of documents to be supplied. The value in UNNEST is normally an array of rows, but DocArray or Xml values are interpreted in the obvious way.

7.8 Predicates

BooleanExpr = BooleanTerm | BooleanExpr OR BooleanTerm .

BooleanTerm = BooleanFactor | BooleanTerm AND BooleanFactor .

BooleanFactor = [NOT] BooleanTest .

BooleanTest = Predicate | ‘(‘ BooleanExpr ‘)’ | *Boolean_Value* .

Predicate = Any | Between | Comparison | Contains | Every | Exists | In | Like | Member | Null | Of |
 PeriodBinary | Some | Unique | [ColumnRef ‘.’] *Document_Value* .

The use of a Document as a predicate is considered to be an equality condition consisting of a conjunction of equality conditions for its field names and values.

Any = ANY ‘(‘ [DISTINCT|ALL] Value) ‘)’ FuncOpt .

Between = Value [NOT] BETWEEN [SYMMETRIC|ASYMMETRIC] Value AND Value .

Comparison = Scalar CompOp Scalar .

CompOp = ‘=’ | ‘<>’ | ‘<’ | ‘>’ | ‘<=’ | ‘>=’ .

Contains = PeriodPredicand CONTAINS (PeriodPredicand | *DateTime_Value*) .

Every = EVERY ‘(‘ [DISTINCT|ALL] Value) ‘)’ FuncOpt .

Exists = EXISTS *Table_Subquery* | XMLEXISTS ‘(‘ *XmlQuery* ‘)’ .

FuncOpt = [FILTER ‘(‘ WHERE SearchCondition ‘)’] [OVER WindowSpec] .

The presence of the OVER keyword makes a *window function*. In accordance with SQL2016-02 section 6.10 and 4.16.3. Window functions can only be used in the select list of a QuerySpec or SelectSingle or the order by clause of a simple table query. Thus window functions cannot be used within expressions or as function arguments.

In = RowValue [NOT] IN ‘(‘ *Table_Subquery* | (Scalar { ‘,’ Scalar }) ‘)’ .

Like = Scalar [NOT] LIKE *Char_Scalar* [ESCAPE *Char_Scalar*].

LIKE_REGEX and SIMILAR can be supported using directives in the source code.

Member = RowValue [NOT] MEMBER OF *Multiset_Scalar* .

Null = Scalar IS [NOT] NULL .

Of = RowValue IS [NOT] (OF ‘(‘ [ONLY] Type { ‘,’ [ONLY] Type } ‘)’ | (CONTENT | DOCUMENT | VALID)) .

Some = SOME ‘(‘ [DISTINCT|ALL] TableValue) ‘)’ FuncOpt .

Unique = UNIQUE *Table_Subquery* .

PeriodBinary = PeriodPredicand (OVERLAPS | EQUALS | [IMMEDIATELY] (PRECEDES | SUCCEEDS)) PeriodPredicand .

See also Contains above.

PeriodPredicand = { id '.' } id | PERIOD '(' Scalar ',' Scalar ') '.

7.9 SQL Functions

FunctionCall = NumericValueFunction | StringValueFunction | DateTimeFunction | SetFunctions | TypeCast | XMLFunction | UserFunctionCall | MethodCall .

All FunctionCalls are considered Scalars unless the returned type is TABLE.

NumericValueFunction = AbsoluteValue | Avg | Ceiling | Coalesce | Count | Exponential | Extract | Floor | Grouping | Last | LengthExpression | Maximum | Minimum | Modulus | NaturalLogarithm | Next | Nullif | Position | PowerFunction | RowNumber | Schema | SquareRoot | Sum .

AbsoluteValue = ABS '(' Scalar ') '.

Avg = AVG '(' [DISTINCT|ALL] Scalar ') ' FuncOpt .

Ceiling = (CEIL|CEILING) '(' Scalar ') '.

Coalesce = COALESCE '(' Scalar {',' Scalar } ') ' Count = COUNT '(' '*' ') '
| COUNT '(' [DISTINCT|ALL] Scalar ') ' FuncOpt .

Exponential = EXP '(' Scalar ') '.

Extract = EXTRACT '(' ExtractField FROM Value ') '.

ExtractField = YEAR | MONTH | DAY | HOUR | MINUTE | SECOND.

Floor = FLOOR '(' Scalar ') '.

Grouping = GROUPING '(' ColumnRef {',' ColumnRef } ') '.

Last = LAST ['(' ColumnRef ') ' OVER WindowSpec] .

LengthExpression = (CHAR_LENGTH|CHARACTER_LENGTH|OCTET_LENGTH) '(' Scalar ') '.

Maximum = MAX '(' [DISTINCT|ALL] Scalar ') ' FuncOpt .

Minimum = MIN '(' [DISTINCT|ALL] Scalar ') ' FuncOpt .

Modulus = MOD '(' Scalar ',' Scalar ') '.

NaturalLogarithm = LN '(' Scalar ') '.

Next = NEXT ['(' ColumnRef ') ' OVER WindowSpec] .

Nullif = NULLIF '(' Scalar ',' Scalar ') '.

WindowSpec = Window_id | '(' WindowDetails ') '.

WithinGroup = WITHIN GROUP '(' OrderByClause ') '.

Position = POSITION ['(' Scalar IN TableValue ') '].

Without parameters POSITION gives a Pyrrho log entry (see section 3.5).

PowerFunction = POWER '(' Scalar ',' Scalar ') '.

RowNumber = ROW_NUMBER '(' ')' OVER WindowSpec .

Schema = SCHEMA '(' ObjectName [COLUMN id] ') '.

Added for Pyrrho: returns a number identifying the most recent schema change affecting the specified object (including any change to this object by another name in another role). Note the syntax of ObjectName given in sec 7.4 above above uses keyword prefixes such as TABLE. The COLUMN syntax shown can only be used with tables.

SquareRoot = SQRT '(' Scalar ')'

Sum = SUM '(' [DISTINCT|ALL] Scalar ')' FuncOpt .

DateTimeFunction = CURRENT_DATE | CURRENT_TIME | LOCALTIME |
CURRENT_TIMESTAMP | LOCALTIMESTAMP .

StringValueFunction = Substring | XmlAgg .

Normalize= NORMALIZE '(' Scalar ')'

Substring = SUBSTRING '(' Scalar FROM Scalar [FOR Scalar] ')'

XmlAgg = XMLAGG '(' Scalar [OrderByClause] ')'

SetFunction = Cardinality | Collect | Element | Fusion | Intersect | Set .

Collect = COLLECT '(' [DISTINCT|ALL] Scalar ')' FuncOpt .

Fusion = FUSION '(' [DISTINCT|ALL] Scalar ')' FuncOpt .

Intersect = INTERSECTION '(' [DISTINCT|ALL] Value ')' FuncOpt .

Cardinality = CARDINALITY '(' Scalar ')'

Element = ELEMENT '(' Scalar ')'

Set = SET '(' Scalar ')'

Typecast = (CAST | XMLCAST) '(' Scalar AS Type ')' | TREAT '(' Scalar AS Sub_Type ')'

7.10 SQL Statements

Assignment = SET Target '=' Scalar { ',' Target '=' Scalar }
| SET '(' Target { ',' Target } ')' '=' Scalar.

For a simple assignment of form Target = Scalar, the keyword SET can be omitted.

Target = id { '.' id } '[' Scalar ']'.

Targets which directly contain parameter lists are not supported in the SQL2016 standard.

Call = CALL Procedure_id '(' [Scalar { ',' Scalar }] ')'
| MethodCall .

Inside a procedure declaration the CALL keyword can be omitted.

CaseStatement = CASE Scalar { WHEN Values THEN Statements } [ELSE Statements] END CASE
| CASE { WHEN SearchCondition THEN Statements } [ELSE Statements] END CASE .

There must be at least one WHEN in the forms shown above.

Close = CLOSE id .

CompoundStatement = Label BEGIN [XMLDec] Statements END .

XMLDec = DECLARE Namespace ';' .

Declaration = DECLARE id { ‘,’ id } Type
 | DECLARE id CURSOR FOR CursorSpecification
 | DECLARE HandlerType HANDLER FOR ConditionList Statement .

Declarations of identifiers, cursors, and handlers are specific to a scope in a SQL routine.

HandlerType = CONTINUE | EXIT | UNDO .

ConditionList = Condition { ‘,’ Condition } .

Condition = ConditionCode | SQLEXCEPTION | SQLWARNING | (NOT FOUND) .

The ConditionCode not_found is acceptable as an alternative to not found.

Signal = SIGNAL ConditionCode [SET CondInfo'= Scalar { ‘,’ CondInfo'= Scalar }]
 | RESIGNAL [ConditionCode] [SET CondInfo'='Value { ‘,’ CondInfo'= Scalar }] .

ConditionCode = *Condition_id* | SQLSTATE string .

CondInfo = CLASS_ORIGIN|SUBCLASS_ORIGIN|CONSTRAINT_CATALOG|
 CONSTRAINT_SCHEMA| CONSTRAINT_NAME|CATALOG_NAME|SCHEMA_NAME|
 TABLE_NAME|COLUMN_NAME|CURSOR_NAME|MESSAGE_TEXT .

GetDiagnostics = GET DIAGNOSTICS Target '=' ItemName { ‘,’ Target '=' ItemName } .

ItemName = NUMBER | MORE | COMMAND_FUNCTION | COMMAND_FUNCTION_CODE |
 DYNAMIC_FUNCTION | DYNAMIC_FUNCTION_CODE | ROW_COUNT |
 TRANSACTIONS_COMMITTED | TRANSACTIONS_ROLLED_BACK |
 TRANSACTION_ACTIVE | CATALOG_NAME | CLASS_ORIGIN | COLUMN_NAME |
 CONDITION_NUMBER | CONNECTION_NAME | CONSTRAINT_CATALOG |
 CONSTRAINT_NAME | CONSTRAINT_SCHEMA | CURSOR_NAME |
 MESSAGE_LENGTH | MESSAGE_OCTET_LENGTH | MESSAGE_TEXT |
 PARAMETER_MODE | PARAMETER_NAME | PARAMETER_ORDINAL_POSITION |
 RETURNED_SQLSTATE | ROUTINE_CATALOG | ROUTINE_NAME |
 ROUTINE_SCHEMA | SCHEMA_NAME | SERVER_NAME | SPECIFIC_NAME |
 SUBCLASS_ORIGIN | TABLE_NAME | TRIGGER_CATALOG | TRIGGER_NAME |
 TRIGGER_SCHEMA .

SQLSTATE strings are 5 characters in length, comprising a 2-character class and a 3 character subclass. See the table in section 8.1.1.

Fetch = FETCH [How] *Cursor_id* INTO VariableRef { ‘,’ VariableRef } .

How = NEXT | PRIOR | FIRST | LAST | ((ABSOLUTE | RELATIVE) Value) .

ForStatement = Label FOR [*For_id* AS][id CURSOR FOR] QueryExpression DO Statements END
 FOR [*Label_id*] .

IfStatement = IF BooleanExpr THEN Statements { ELSEIF BooleanExpr THEN Statements }
 [ELSE Statements] END IF .

Label = [label ':'] .

LoopStatement = Label LOOP Statements END LOOP .

Open = OPEN id .

Repeat = Label REPEAT Statements UNTIL BooleanExpr END REPEAT .

SelectSingle = SELECT [ALL|DISTINCT] SelectItems INTO TargetList TableExpression .

TargetList = VariableRef { ‘,’ VariableRef } .

Statements = Statement { ';' Statement } .

While = Label WHILE SearchCondition DO Statements END WHILE .

UserFunctionCall = Id '(' [Scalar { ',' Scalar }] ')'

MethodCall = Scalar '.' Method_id '(' [Scalar { ',' Scalar }] ')'
 | '(' Scalar AS Type ')' '.' Method_id '(' [Scalar { ',' Scalar }] ')'
 | Type '::' Method_id '(' [Scalar { ',' Scalar }] ')'

7.11 XML Support

XMLFunction = XMLComment | XMLConcat | XMLDocument | XMLElement | XMLForest | XMLParse | XMLProc | XMLQuery | XMLText | XMLValidate.

XMLComment = XMLCOMMENT '(' Value ')'

XMLConcat = XMLCONCAT '(' Value { ',' Value } ')'

XMLDocument = XMLDOCUMENT '(' Value ')'

XMLElement = XMLELEMENT '(' NAME id [',' Namespace] [',' AttributeSpec] { ',' Value } ')'

Namespace = XMLNAMESPACES '(' NamespaceDefault [(string AS id { ',' string AS id })] ')'

NamespaceDefault = (DEFAULT string) | (NO DEFAULT)

AttributeSpec = XMLATTRIBUTES '(' NamedValue { ',' NamedValue } ')'

NamedValue = Value [AS id]

XMLForest = XMLFOREST '(' [Namespace { ',' }] NamedValue { ',' NamedValue } ')'

XMLParse = XMLPARSE '(' CONTENT Value ')'

XMLProc = XMLPI '(' NAME id [',' Value] ')'

XMLQuery = XMLQUERY '(' Value , *xpath_xml* ')'

This syntax seems to be non-standard in Pyrrho but allows extraction from an xml Value using an XPath expression

XMLText = XMLTEXT '(' xml ')'

XMLValidate = XMLVALIDATE '(' (DOCUMENT|CONTENT|SEQUENCE) Value ')'

7.12 Proposed simplification of the SQL2016 security model

The rationale for the following changes is twofold:

- (a) REVOKE should be effective whatever the history, except for privileges that are available to PUBLIC,
- (b) authorised changes should persist even after the authority for them is revoked (for example if an employee leaves, there may be implementation defined ways (as in section 3.5) of recovering tables that have no authorised users).

The following subsection number corresponds to the numbering in the SQL2016 standard.

12.7 <revoke statement>

General Rules

Delete rules 1) to 50) and replace them with the following.

- 1) If the <revoke statement> is a <revoke privilege statement>, then all identified privilege descriptors are destroyed in a cascade that also revokes from the identified user any roles that have the identified privileges. The <revoke statement> may also revoke GRANT and HIERARCHY options.
- 2) If the <revoke statement> is a <revoke role statement>, then the identified grantees are removed from the identified role descriptors, if present.
- 3) If the privilege being revoked will remain granted to PUBLIC following execution, the implementation must issue a warning.

8. Pyrrho Reference

There are five collections of system tables in Pyrrho. The Sys\$ collections contain the current system information set, the Role\$ collection is the schema for the current role, and the Log\$ collection accesses the transaction log. All these collections consist of virtual tables, whose data is constructed as required from the Pyrrho engine's data structures. From version 5.0 it is possible to see uncommitted details in the current transaction, so that "defining positions" in these system tables are no longer Integer but String data: the fields contain the string version of the Integer defining position if it is committed, and otherwise contain a numeric identifier preceded by a single quote.

The fourth kind of system table is for reviewing data operations on an individual table. See section 8.5.

There is a set of six system tables that contains transaction profile information. See section 8.6.

All these tables and their attributes are case-sensitive, and the table-names contain the character \$, so all SQL statements will need to use double-quoted (delimited) identifiers, as in

```
Select * from "Sys$Role" where "Name" like 'Sales%'
```

8.1 Diagnostics

Pyrrho implements basic diagnostics management as defined in SQL2016, with a single diagnostics area. The NOT_FOUND condition is signalled if there is a handler for it (no-data is not regarded as an error in SQL2016).

8.1.1 SQLSTATE

Pyrrho defines the following SQLSTATES, shown here with the message formats for the invariant culture (these can be localised in the client library).

Pyrrho treats many things as errors that appear in the SQL standard as warnings, and imposes fewer restrictions: see comments below. A large number of error messages below (in category 40) relate to transaction conflicts caused by schema changes.

It is permissible to define, raise and handle other condition codes.

| Number | Message Template | ISO | Pyrrho | Comments |
|--------|--|-----|--------|---|
| 00000 | Successful completion | y | y | Not an exception |
| 01000 | Warning | y | n | |
| 01001 | Warning – cursor operation conflict | y | n | Not reported |
| 01002 | Warning – disconnect error | y | n | Condition 2E000 raised instead |
| 01003 | Warning – null value eliminated in set function | y | y | |
| 01004 | Warning – string data, right truncation | y | n | 01004 is used for fixed length binary data: see 22001 instead |
| 01005 | Warning – insufficient item descriptor areas | y | n | |
| 01006 | Warning – privilege not revoked | y | n | Condition 42105 raised instead |
| 01007 | Warning – privilege not granted | y | n | Condition 42105 raised instead |
| 01005 | Warning – insufficient item descriptor areas | y | n | Cannot occur |
| 01009 | Warning – search condition too long for information schema | y | n | Cannot occur |
| 0100A | Warning – query expression too long for information schema | y | n | Cannot occur |
| 0100B | Warning – default value too long for information schema | y | n | Cannot occur |
| 0100C | Warning – result sets returned | y | n | |
| 0100D | Warning – additional result sets returned | y | n | |
| 0100E | Warning – attempt to return too many result parameters | y | y | |
| 0100F | Warning – statement too long for information schema | y | n | Cannot occur |

| | | | | |
|-------|---|---|---|---------------------------------------|
| 01012 | Warning – invalid number of conditions | y | n | Not reported |
| 0102F | Warning – array data, right truncation | y | n | Cannot occur |
| 02000 | No data | y | y | Not an exception |
| 02001 | No additional result sets returned | y | n | |
| 07000 | Dynamic SQL error | y | n | |
| 07001 | Using clause does not match dynamic parameter specifications | y | n | |
| 07002 | Using clause does not match target specifications | y | n | |
| 07003 | Cursor specification cannot be executed | y | n | |
| 07004 | Using clause required for dynamic parameters | y | n | |
| 07005 | Prepared statement not a cursor specification | y | n | |
| 07006 | Restricted data type attribute violation | y | n | |
| 07007 | Using clause required for result fields | y | n | |
| 07008 | Invalid descriptor count | y | n | |
| 07009 | Invalid descriptor index | y | n | |
| 0700B | Data type transform function violation | y | n | |
| 0700C | Undefined DATA value | y | n | |
| 0700D | Invalid DATA target | y | n | |
| 0700E | Invalid LEVEL value | y | n | |
| 0700F | Invalid DATETIME_INTERVAL_CODE | y | n | |
| 0700G | Invalid pass-through surrogate value | y | n | |
| 0700H | PIPE ROW not during PTF execution | y | n | |
| 08000 | Connection exception | y | n | See 2E |
| 08001 | SQL-client unable to establish SQL-connection | y | y | |
| 08002 | Connection name in use | y | y | |
| 08003 | Connection does not exist | y | y | |
| 08004 | SQL-Server rejected establishment of SQL-connection | y | y | |
| 08006 | Connection failure | y | y | |
| 08007 | Connection exception – transaction resolution unknown | y | y | |
| 08C00 | Client-side threading violation for reader | n | y | |
| 08C01 | Client-side threading violation for command | n | y | |
| 08C02 | Client-side threading violation for a transaction | n | y | |
| 08C03 | An explicit transaction is already active in this thread and connection | n | y | |
| 08C04 | A reader is already open in this thread and connection | n | y | |
| 08C05 | Conflict with an open reader in this thread and connection | n | y | |
| 08C06 | Cannot change connection properties during a transaction | n | y | |
| 09000 | Triggered action exception | y | n | Pyrrho uses a single diagnostics area |
| 0A000 | Feature not supported | y | n | |
| 0A001 | Feature not supported – multiple server transactions | y | n | Pyrrho supports multiple servers |
| 0D000 | Invalid target type specification | y | y | |
| 0E000 | Invalid schema name list specification | y | n | S071 is not supported |
| 0F000 | Locator exception | y | n | T561 is not supported |

| | | | | |
|-------|--|---|---|---------------------------|
| 0F001 | Locator exception – invalid specification | y | n | T561 is not supported |
| 0L000 | Invalid grantor | y | n | Condition 42105 is raised |
| 0M000 | Invalid SQL-invoked procedure reference | y | n | T471 is not supported |
| 0P000 | Invalid role specification | y | n | Condition 42105 is raised |
| 0S000 | Invalid transform group specification | y | n | S241 is not supported |
| 0T000 | Target table disagrees with cursor specification | y | y | |
| 0U000 | Attempt to assign to non-updatable column | y | y | |
| 0V000 | Attempt to assign to ordering column | y | n | B031 is not supported |
| 0W000 | Prohibited statement encountered during trigger execution | y | n | See 27000 |
| 0W001 | Trigger error - modify table modified by data change delta table | y | n | See 27001 |
| 0Z000 | Diagnostics exception | y | n | |
| 0Z001 | Maximum number of stacked diagnostics areas exceeded | y | n | Cannot occur |
| 11000 | Prohibited column reference encountered during trigger execution | y | n | |
| 21000 | Cardinality violation | y | y | |
| 22000 | Data exception | y | y | |
| 22001 | String data, right truncation | y | y | |
| 22002 | Null value, no indicator parameter | y | n | |
| 22003 | Numeric value out of range | y | y | |
| 22004 | Null value not allowed | y | y | |
| 22005 | Error in assignment | y | y | |
| 22006 | Invalid interval format | y | n | |
| 22007 | Invalid datetime format: ? | y | y | Diagnostic info added |
| 22008 | Datetime field overflow: ? | y | y | Diagnostic info added |
| 22009 | Invalid time zone displacement value | y | n | |
| 2200B | Escape character conflict | y | n | |
| 2200C | Invalid use of escape character | y | n | |
| 2200D | Invalid escape octet | y | n | |
| 2200E | Null value in array target | y | n | |
| 2200F | Zero-length character string | y | n | |
| 2200G | Most specific type mismatch | y | y | |
| 2200H | Sequence generator limit exceeded | y | n | |
| 2200J | Nonidentical notations with the same name | y | n | |
| 2200K | Nonidentical unparsed entities with the same name | y | n | |
| 2200N | Invalid XML content | y | y | |
| 2200P | Interval value out of range | y | n | |
| 2200Q | Multiset value overflow | y | n | Cannot occur |
| 2200S | Invalid XML comment | y | n | |
| 22010 | Invalid indicator parameter value | y | n | |
| 22011 | Substring error | y | n | |
| 22012 | Division by zero | y | y | |
| 22013 | Invalid preceding or following size in window function | y | n | |
| 22014 | Invalid argument for NTILE function | y | n | |
| 22015 | Interval field overflow | y | n | |
| 22016 | Invalid argument for NTH_VALUE function | y | n | |
| 22018 | Invalid character value for cast | y | n | |
| 22019 | Invalid escape character | y | y | |
| 2201B | Invalid regular expression | y | y | |
| 2201C | Null row not permitted in value | y | n | |

| | | | |
|-------|---|---|---|
| 2201E | Invalid argument for natural logarithm | y | n |
| 2201F | Invalid argument for power function | y | n |
| 2201G | Invalid argument for width bucket function | y | n |
| 2201H | Invalid row version | y | n |
| 2201M | Namespace ? not defined | y | y |
| 2201S | Invalid XQuery regular expression | y | n |
| 2201T | Invalid XQuery option flag | y | n |
| 2201U | Attempt to replace a zero-length string | y | n |
| 2201V | Invalid XQuery replacement string | y | n |
| 2201W | Invalid row count in a fetch first clause | y | n |
| 2201X | Invalid row count in result offset clause | y | n |
| 2201Y | Zero-length binary string | y | n |
| 22020 | Invalid period value | y | n |
| 22021 | Character not in repertoire | y | n |
| 22022 | Indicator overflow | y | n |
| 22023 | Invalid parameter value | y | n |
| 22024 | Unterminated C string | y | n |
| 22025 | Invalid escape sequence | y | y |
| 22026 | String data length mismatch | y | n |
| 22027 | Trim error | y | n |
| 22029 | Noncharacter in UCS string | y | n |
| 2202D | Null value substituted for mutator subject parameter | y | n |
| 2202E | Array element error | y | n |
| 2202F | Array data, right truncation | y | n |
| 2202G | Invalid repeat argument in a sample clause | y | n |
| 2202H | Invalid sample size | y | n |
| 2202J | Invalid argument for row pattern navigation operation | y | b |
| 2202K | Skip to non-existent row | y | n |
| 2202L | Skip to first row of match | y | n |
| 22030 | Duplicate JSON object key value | y | n |
| 22031 | Invalid argument for SQL/JSON datetime function | y | n |
| 22032 | Invalid JSON text | y | n |
| 22033 | Invalid SQL/JSON subscript | y | n |
| 22034 | More than one SQL/JSON item | | |
| 22035 | No SQL/JSON item | y | n |
| 22036 | Non-numeric SQL/JSON item | y | n |
| 22037 | Non-unique keys in JSON object | y | n |
| 22038 | Singleton SQL/JSON item required | y | n |
| 22039 | SQL/JSON array not found | y | n |
| 2203A | SQL/JSON member not found | y | n |
| 2203B | SQL/JSON number not found | y | n |
| 2203C | SQL/JSON object not found | y | n |
| 2203D | Too many JSON array elements | y | n |
| 2203E | Too many JSON object members | y | n |
| 2203F | SQL/JSON scalar required | y | n |
| 22041 | Invalid RDF format | n | y |
| 22102 | Type mismatch on concatenate | n | y |
| 22103 | Multiset element not found | n | y |
| 22104 | Incompatible multisets for union | n | y |
| 22105 | Incompatible multisets for intersection | n | y |
| 22106 | Incompatible multisets for except | n | y |
| 22107 | Exponent expected | n | y |
| 22108 | Type error in aggregation operation | n | y |
| 22109 | Too few arguments | n | y |
| 22110 | Too many arguments | n | y |

OWL type extension to SQL

| | | | | |
|-------|--|---|---|---------------------------------------|
| 22111 | Circular dependency found | n | y | |
| 22201 | Unexpected type ? for comparison with Decimal | n | y | |
| 22202 | Incomparable types | n | y | |
| 22203 | Loss of precision on conversion | n | y | |
| 22204 | Query expected | n | y | |
| 22205 | Null value found in table ? | n | y | |
| 22206 | Null value not allowed in column ? | n | y | |
| 22207 | Row has incorrect length | n | y | |
| 22208 | Mixing named and unnamed columns is not supported | n | y | |
| 22209 | AutoKey is not available for ? | n | y | |
| 22210 | Illegal assignment of sensitive value | n | y | |
| 22211 | Domain ? Check constraint fails | n | y | |
| 22212 | Column ? Check constraint fails | n | y | |
| 22300 | Bad document format | n | y | Document Extension to SQL |
| 23000 | Integrity constraint violation | y | y | |
| 23001 | RESTRICT: ? referenced in ? | y | y | A referenced object cannot be deleted |
| 23103 | This record cannot be updated | n | y | usually integrity violation |
| 24000 | Invalid cursor state | y | y | |
| 24101 | Cursor is not open | n | y | |
| 25000 | Invalid transaction state | y | y | |
| 25001 | Active SQL-transaction | y | y | |
| 25002 | Branch transaction already active | y | n | |
| 25003 | Inappropriate access mode for branch transaction | y | n | |
| 25004 | Inappropriate isolation level for branch transaction | y | n | |
| 25005 | No active SQL-transaction for branch transaction | y | n | |
| 25006 | Read-only SQL-transaction | y | n | |
| 25007 | Schema and data statement mixing not supported | y | n | |
| 25008 | Held cursor requires same isolation level | y | n | |
| 26000 | Invalid SQL statement name | y | y | |
| 27000 | Triggered data change violation | y | n | |
| 27001 | Trigger exception – modify table modified by data change delta table | y | n | |
| 28000 | Invalid authorization specification | y | y | No role ? in database ? |
| 28101 | Unknown grantee kind | n | y | |
| 28102 | Unknown grantee ? | n | y | |
| 28104 | Users can only be added to roles | n | y | |
| 28105 | Grant of select: entire row is nullable | n | y | |
| 28106 | Grant of insert must include all notnull columns | n | y | |
| 28107 | Grant of insert cannot include generated column ? | n | y | |
| 28108 | Grant of update : column ? is not updatable | n | y | |
| 2B000 | Dependent privilege descriptors still exist | y | n | |
| 2C000 | Invalid character set name | y | n | |
| 2C001 | Cannot drop SQL-session default character vset | y | n | |
| 2D000 | Invalid transaction termination | y | y | |
| 2E000 | Invalid connection name | y | y | |
| 2E104 | Database is read-only | n | y | |
| 2E105 | Invalid user for database ? | n | y | |

| | | | | |
|-------|---|---|---|----------------|
| 2E106 | This operation requires a single-database session | n | y | |
| 2E108 | Stop time was specified, so database is read-only | n | y | |
| 2E110 | Unauthorized HTTP access | n | y | |
| 2E111 | User ? can access no columns of table ? | n | y | |
| 2E201 | Connection is not open | n | y | See also 080nn |
| 2E202 | A reader is already open | n | y | |
| 2E203 | Unexpected reply | n | y | |
| 2E204 | Bad data type ? (internal) | n | y | |
| 2E205 | Stream closed | n | y | |
| 2E206 | Internal error: ? | n | y | |
| 2E208 | Badly formatted connection string ? | n | y | |
| 2E209 | Unexpected element ? in connection string | n | y | |
| 2E210 | LOCAL database server does not support distributed or partitioned operation | n | y | |
| 2E300 | <i>The calling assembly does not have type ?</i> | n | y | |
| 2E301 | <i>Type ? doesn't have a default constructor</i> | n | y | |
| 2E302 | <i>Type ? doesn't define field ?</i> | n | y | |
| 2E303 | Types ? and ? do not match | n | y | |
| 2E304 | Get rurl should begin with / | n | y | REST service |
| 2E305 | No data returned by rurl ? | n | y | REST service |
| 2E307 | Obtain an up-to-date schema for ? from Role\$Class | n | y | |
| 2F000 | SQL routine exception | y | n | |
| 2F002 | Modifying SQL-data not permitted | y | n | |
| 2F003 | Prohibited SQL-statement attempted | y | y | |
| 2F004 | Reading SQL-data not permitted | y | n | |
| 2F005 | Function executed no return statement | y | n | |
| 2H000 | Invalid collation name | y | y | |
| 30000 | Invalid SQL statement identifier | y | n | |
| 33000 | Invalid SQL descriptor name | y | y | |
| 33001 | Error in prepared statement parameters | n | y | |
| 34000 | Invalid cursor name | y | y | |
| 35000 | Invalid condition number | y | n | |
| 36000 | Cursor sensitivity exception | y | n | |
| 36001 | Cursor sensitivity exception – request rejected | y | n | |
| 36002 | Cursor sensitivity exception – request failed | y | n | |
| 38000 | External routine exception | y | n | |
| 38001 | External routine – containing SQL not permitted | y | n | |
| 38002 | External routine – modifying SQL-data not permitted | y | n | |
| 38003 | External routine – prohibited SQL-statement attempted | y | n | |
| 38004 | External routine – reading SQL-data not permitted | y | n | |
| 39000 | External routine invocation exception | y | n | |
| 39004 | External routine invocation – null value not allowed | y | n | |
| 3B000 | Savepoint exception | y | n | |
| 3B001 | Savepoint exception – invalid specification | y | n | |
| 3B002 | Too many savepoints | y | n | |

| | | | | |
|-------|---|---|---|----------------------------------|
| 3C000 | Ambiguous cursor name | y | n | |
| 3D000 | Invalid catalog specification | y | y | |
| 3D001 | Database ? not open | n | y | |
| 3D005 | Requested operation not supported by this edition of Pyrrho | n | y | |
| 3D006 | Database ? incorrectly terminated or damaged | n | y | |
| 3D007 | Database is not append storage | n | y | Server is append storage version |
| 3D008 | Database is append storage | n | y | Server is not for append storage |
| 3D010 | Invalid Password | n | y | |
| 3F000 | Invalid schema name | y | n | |
| 40000 | Transaction rollback | y | y | |
| 40001 | Transaction Serialisation Failure | y | y | |
| 40002 | Transaction rollback – integrity constraint violation | y | n | |
| 40003 | Transaction rollback – statement completion unknown | y | y | |
| 40004 | Transaction rollback – triggered action exception | y | n | |
| 40005 | Transaction rollback – new key conflict with empty query | n | y | |
| 40006 | Transaction conflict: Read constraint for ? | n | y | |
| 40007 | Transaction conflict: Read conflict for ? | n | y | |
| 40008 | Transaction conflict: Read conflict for table ? | n | y | |
| 40009 | Transaction conflict: Read conflict for record ? | n | y | |
| 40010 | Object ? has just been dropped | n | y | |
| 40011 | Supertype ? has just been dropped | n | y | |
| 40012 | Table ? has just been dropped | n | y | |
| 40013 | Column ? has just been dropped | n | y | |
| 40014 | Record ? has just been deleted | n | y | |
| 40015 | Type ? has just been dropped | n | y | |
| 40016 | Domain ? has just been dropped | n | y | |
| 40017 | Index ? has just been dropped | n | y | |
| 40021 | Supertype ? has just been changed | n | y | |
| 40022 | Another domain ? has just been defined | n | y | |
| 40023 | Period ? has just been changed | n | y | |
| 40024 | Versioning has just been defined | n | y | |
| 40025 | Table ? has just been altered | n | y | |
| 40026 | Integrity constraint: ? has just been added | n | y | |
| 40027 | Integrity constraint: ? has just been referenced | n | y | |
| 40029 | Record ? has just been updated | n | y | |
| 40030 | A conflicting table ? has just been defined | n | y | |
| 40031 | A conflicting view ? has just been defined | n | y | |
| 40032 | A conflicting object ? has just been defined | n | y | |
| 40033 | A conflicting trigger for ? has just been defined | n | y | |
| 40034 | Table ? has just been renamed | n | y | |
| 40035 | A conflicting role ? has just been defined | n | y | |
| 40036 | A conflicting routine ? has just been defined | n | y | |
| 40037 | An ordering now uses function ? | n | y | |

| | | | |
|-------|--|---|---|
| 40038 | Type ? has just been renamed | n | y |
| 40039 | A conflicting method ? for ? has just been defined | n | y |
| 40040 | A conflicting period for ? has just been defined | n | y |
| 40041 | Conflicting metadata for ? has just been defined | n | y |
| 40042 | A conflicting index for ? has just been defined | n | y |
| 40043 | Columns of table ? have just been changed | n | y |
| 40044 | Column ? has just been altered | n | y |
| 40045 | A conflicting column ? has just been defined | n | y |
| 40046 | A conflicting check ? has just been defined | n | y |
| 40047 | Target object ? has just been renamed | n | y |
| 40048 | A conflicting ordering for ? has just been defined | n | y |
| 40049 | Ordering definition conflicts with drop of ? | n | y |
| 40050 | A conflicting namespace change has occurred | n | y |
| 40051 | Conflict with grant/revoke on ? | n | y |
| 40052 | Conflicting routine modify for ? | n | y |
| 40053 | Domain ? has just been used for insert | n | y |
| 40054 | Domain ? has just been used for update | n | y |
| 40055 | An insert conflicts with drop of ? | n | y |
| 40056 | An update conflicts with drop of ? | n | y |
| 40057 | A delete conflicts with drop of ? | n | y |
| 40058 | An index change conflicts with drop of ? | n | y |
| 40059 | A constraint change conflicts with drop of ? | n | y |
| 40060 | A method change conflicts with drop of type ? | n | y |
| 40068 | Domain ? has just been altered, conflicts with drop | n | y |
| 40069 | Method ? has just been changed, conflicts with drop | n | y |
| 40070 | A new ordering conflicts with drop of type ? | n | y |
| 40071 | A period definition conflicts with drop of ? | n | y |
| 40072 | A versioning change conflicts with drop of period ? | n | y |
| 40073 | A read conflicts with drop of ? | n | y |
| 40074 | A delete conflicts with update of ? | n | y |
| 40075 | A new reference conflicts with deletion of ? | n | y |
| 40076 | A conflicting domain or type ? has just been defined | n | y |
| 40077 | A conflicting change on ? has just been done | n | y |
| 40078 | Read conflict with alter of ? | n | y |
| 40079 | Insert conflict with alter of ? | n | y |
| 40080 | Update conflict with alter of ? | n | y |
| 40081 | Alter conflicts with drop of ? | n | y |
| 40082 | ETag validation failure | n | y |

| | | | | | | |
|-------|---|---|---|------------------|------------|-----------|
| 40083 | Secondary connection conflict on ? | n | y | Remote differ | connection | snapshots |
| 42000 | Syntax error or access rule violation at ? | y | y | | | |
| 42101 | Illegal character ? | n | y | | | |
| 42102 | Name cannot be null | n | y | | | |
| 42103 | Key must have at least one column | n | y | | | |
| 42104 | Proposed name conflicts with existing database object (e.g. table already exists) | n | y | | | |
| 42105 | Access denied ? | n | y | | | |
| 42107 | Table ? undefined | n | y | | | |
| 42108 | Procedure ? not found | n | y | | | |
| 42109 | Assignment target ? not found | n | y | | | |
| 42111 | The given key is not found in the referenced table | n | y | | | |
| 42112 | Column ? not found | n | y | | | |
| 42113 | Multiset operand required, not ? | n | y | | | |
| 42115 | Unexpected object type ? ? for GRANT | n | y | | | |
| 42116 | Role revoke has ADMIN option not GRANT | n | y | | | |
| 42117 | Privilege revoke has GRANT option not ADMIN | n | y | | | |
| 42118 | Unsupported CREATE ? | n | y | | | |
| 42119 | Domain ? not found in database ? | n | y | | | |
| 4211A | Unknown privilege ? | n | y | | | |
| 42120 | Domain or type must be specified for base column ? | n | y | | | |
| 42123 | NO ACTION is not supported | n | y | | | |
| 42124 | Colon expected .. | n | y | | | |
| 42125 | Unknown Alter type ? | n | y | | | |
| 42126 | Unknown SET operation | n | y | | | |
| 42127 | Table expected | n | y | | | |
| 42128 | Illegal aggregation operation | n | y | | | |
| 42129 | WHEN expected | n | y | | | |
| 42131 | Invalid POSITION ? | n | y | | | |
| 42132 | Method ? not found in type ? | n | y | | | |
| 42133 | Type ? not found | n | y | | | |
| 42134 | FOR phrase is required | n | y | | | |
| 42135 | Object ? not found | n | y | | | |
| 42138 | Field selector ? not defined for ? | n | y | | | |
| 42139 | :: on non-type | n | y | | | |
| 42140 | :: requires a static method | n | y | | | |
| 42142 | NEW requires a user-defined type constructor | n | y | | | |
| 42143 | ? specified more than once | n | y | | | |
| 42146 | OLD specified on insert trigger or NEW specified on delete trigger | n | y | | | |
| 42147 | Cannot have two primary keys for table ? | n | y | | | |
| 42148 | FOR EACH ROW not specified | n | y | | | |
| 42149 | Cannot specify OLD/NEW TABLE for before trigger | n | y | | | |
| 42150 | Malformed SQL input (non-terminated string) | n | y | | | |
| 42151 | Bad join condition | n | y | | | |
| 42152 | Non-distributable where condition for update/delete | n | y | | | |
| 42153 | Table ? already exists | n | y | | | |
| 42154 | Unimplemented or illegal function ? | n | y | | | |
| 42156 | Column ? is already in table ? | n | y | | | |

| | | | |
|-------|--|---|---|
| 42157 | END label ? does not match start label ? | n | y |
| 42158 | ? is not the primary key for ? | n | y |
| 42159 | ? is not a foreign key for ? | n | y |
| 42160 | ? has no unique constraint | n | y |
| 42161 | ? expected at ? | n | y |
| 42162 | Table period definition for ? has not been defined | n | y |
| 42163 | Generated column ? cannot be used in a constraint | n | y |
| 42164 | Table ? has no primary key | n | y |
| 42166 | Domain ? already exists | n | y |
| 42167 | A routine with name ? and arity ? already exists | n | y |
| 42168 | AS GET needs a schema definition | n | y |
| 42169 | Ambiguous column name ? needs alias | n | y |
| 42170 | Column ? must be aggregated or grouped | n | y |
| 42171 | A table cannot be placed in a column | n | y |
| 42172 | Identifier ? already declared in this block | n | y |
| 42173 | Method ? not defined | n | y |
| 44000 | With check option violation | y | y |
| 44001 | Domain check ? fails for column ? in table ? | n | y |
| 44002 | Table check ? fails for table ? | n | y |
| 44003 | Column check ? fails for column ? in table ? | n | y |
| 44004 | Column ? in Table ? contains null values, not null cannot be set | n | y |
| 44005 | Column ? in Table ? contains values, generation rule cannot be set | n | y |
| HZ000 | Remote Database Access error | y | n |

8.1.2 Get Diagnostics

From version 4.8, Pyrrho supports the GET DIAGNOSTICS statement, giving useful information for the following keys. When an exception condition is handled in an SQL routine or reported to the client, information from this collection is included in the DatabaseError.

| | |
|------------------------|--|
| CATALOG_NAME | |
| CLASS_ORIGIN | This is ISO 9075 for conditions whose class is defined in SQL2016 |
| COLUMN_NAME | |
| COMMAND_FUNCTION | From Table 32 of the SQL standard |
| COMMAND_FUNCTION_CODE | From Table 32 of the SQL standard |
| CONDITION_NUMBER | |
| CONNECTION_NAME | This is the Files part of the connection string |
| CONSTRAINT_NAME | |
| CURSOR_NAME | |
| MESSAGE_LENGTH | Computed from MESSAGE_TEXT |
| MESSAGE_OCTET_LENGTH | Computed from MESSAGE_TEXT |
| MESSAGE_TEXT | By default, this is formatted when an exception occurs |
| RETURNED_SQLSTATE | The condition code |
| ROUTINE_NAME | |
| ROW_COUNT | |
| SERVER_NAME | The host part of the connection string |
| SUBCLASS_ORIGIN | This is ISO 9075 if the whole condition code is defined in SQL2016 |
| TABLE_NAME | |
| TRANSACTIONS_COMMITTED | The number of transactions committed for this connection |

| | |
|--------------------------|--|
| TRANSACTIONS_ROLLED_BACK | The number of rollbacks for this connection |
| TRIGGER_NAME | |
| TYPE* | The target type |
| VALUE* | The value type |
| WITH* | Additional information for transaction conflicts (version 5.4) |

*Pyrrho specific.

8.2 Sys\$ table collection

Sys\$Audit, Sys\$Role, and Sys\$User list all of the corresponding objects in the current database. The Sys\$ tables are read-only and available only to the database owner: the only way to change anything in a database is by means of the APIs provided e.g. SQL or REST.

8.2.1 Sys\$Audit

| Field | Data Type | Description |
|-----------|-----------|---|
| Pos | Char | The location of this access record in the transaction log |
| User | Char | The name of the accessing user |
| Table | Char | The defining position of the sensitive or classified object |
| Timestamp | Int | The time of the access in ticks |

Audit records are only for committed sensitive data. Entries come from physical Audit records, and are added immediately on access (do not wait for transaction commit).

8.2.2 Sys\$AuditKey

| Field | Data Type | Description |
|-------|-----------|---|
| Pos | Char | The location of the access record in the transaction log |
| Seq | Int | The ordinal position of the key (0 based) |
| Col | Char | The defining position of the key column |
| Key | Char | A string representation of the key value at this position |

Key information for audit records comes from the filters used to access a sensitive object. For example, if a record is inserted in a table, there is no applicable filter, the audit record will apply to the whole table, and there will be no key information here.

8.2.3 Sys\$Classification

| Field | Data Type | Description |
|-----------------|-----------|---|
| Pos | Char | The defining position of this record in the transaction log |
| Type | Char | The object type |
| Classification | Char | Readable version of Level as in 7.2 |
| LastTransaction | Char | The most recent transaction for this object |

This table contains information for all current objects and data records with classification different from D. The order is not specified. Rows are not included unless the whole row is classified (see Sys\$ClassifiedColumnData). The key in this table is Pos.

8.2.4 Sys\$ClassifiedColumnData

| Field | Data Type | Description |
|-----------------|-----------|--|
| Pos | Char | The defining position of the record in the transaction log |
| Col | Char | The Column's defining position |
| Classification | Char | Readable version of Level as in 7.2 for the contents |
| LastTransaction | Char | The most recent transaction for this record |

This table contains information for current records affecting columns whose classification is different from D, excluding records contained in Sys\$Classification. The order is not specified. The key in this table is (Pos,Col).

8.2.5 Sys\$Enforcement

| Field | DataType | Description |
|-------|----------|-------------------|
| Name | char | The Table name |
| Scope | char | Enforcement flags |

By default classification is enforced for all operations: there will be entries in this table only for tables with specified enforcement levels. There may also be an entry for the table in Sys\$Classification.

8.2.6 Sys\$Role

| Field | DataType | Description |
|---------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Name | char | The Role identifier |
| Details | char | A readable description of the intended use of the role |

(Pos) and (Name) are keys in this table.

8.2.7 Sys\$RoleUser

| Field | DataType | Description |
|-------|----------|--|
| Role | Char | The Role identifier |
| User | Char | A User identifier allowed to use this role |

(Role,User) is the key in this table.

8.2.8 Sys\$ServerConfiguration

| Field | DataType | Description |
|----------|----------|--|
| Property | Char | Currently one of AllowDatabaseCreation (true), SegmentationBits (35), ValueRowSetLimit (0=no limit), IndexLimit (0=no limit) |
| Value | Char | The value of this configuration setting. |

Currently ValueRowSetLimit and IndexLimit must be the same.

8.2.9 Sys\$User

| Field | DataType | Description |
|--------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Name | Char | The User identifier |
| SetPassword | Bool | Password will be set on next login. For HTTP authentication this field must be True or False (not null). |
| Initial Role | Char | Defines an initial Role for the user (for HTTP only) |
| Clearance | Char | Readable version of Level as in 7.2 |

Users are created in the database the first time they are assigned privileges. (There is no CREATE USER in SQL2016.) Users cannot be renamed. (Pos) and (Name) are keys in this table.

8.3 Role\$ table collection

Objects owned by other roles may be prefixed by the role name in order to preserve the key(s) noted for each table.

The Role\$ tables are read-only: the only way to change anything in a database is by means of the APIs provided e.g. SQL or REST.

8.3.1 Role\$Class

| Field | DataType | Description |
|------------|----------|--|
| Name | Char | The name of a base table or view, with the same name as the class |
| Key | Char | A comma separated list of the key columns of this object if any |
| Definition | Char | A C# class definition suitable for receiving rows of this object. See also Role\$Java below. |

Dots in top-level column names coming from views are automatically replaced by underscores.

8.3.2 Role\$Column

| Field | DataType | Description |
|--------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Table | Char | The current name of the Table or View |
| Name | Char | The current name of the Column |
| Seq | Int | The current position in the row (there may be gaps in the sequence here due to columns inaccessible from the current role) |
| Domain | Char | The data type for the Column |
| DefaultValue | Char | String representation of the default value |
| NotNull | Boolean | Whether the column has been defined NOT NULL |
| Generated | Boolean | Whether the column is GENERATED ALWAYS |
| Update | Char | The update statement for a generated column |

(Pos) and (Table,Name) are keys.

8.3.3 Role\$ColumnCheck

| Field | DataType | Description |
|-----------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Table | Char | The current name of the Table |
| Name | Char | The current name of the Column |
| CheckName | Char | The current identifier for the CHECK (unique per domain) |
| Select | Char | The QueryExpression used to check the VALUE |

(Pos) and (Table,Name,CheckName) are keys.

8.3.4 Role\$ColumnPrivilege

| Field | DataType | Description |
|-----------|----------|--|
| Table | Char | The current name of the table in the current role |
| Name | Char | The current name of the column in the current role |
| Grantee | Char | The Grantee name (a Role) |
| Privilege | Char | The privilege granted |

(Table,Name,Grantee) is the key.

8.3.5 Role\$Domain

| Field | DataType | Description |
|--------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Name | Char | The current identifier for the DOMAIN. May have forms such as CHAR(6), U(5005) if not user-defined (see note below). |
| DataType | Char | The data type |
| DataLength | Int | The data length (precision for DECIMAL, REAL, INTEGER) |
| Scale | Int | The scale (for Numeric type) |
| StartField | Char | The start field (for Interval type) |
| EndField | Char | The end field (for Interval type) |
| DefaultValue | Char | String representation of the default value |
| Struct | Char | Type string for MULTiset or ARRAY or ROW element |
| Definer | Char | The owning role |

Pyrrho creates a new domain for each new type in the database (e.g. CHAR(6)), and makes a special domain for evaluating generated columns. (Pos) and (Name) are keys.

8.3.6 Role\$DomainCheck

| Field | DataType | Description |
|------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| DomainName | char | The current identifier for the DOMAIN |
| CheckName | Char | The current identifier for the CHECK (unique per domain) |
| Select | Char | The QueryExpression used to check the VALUE |

(Pos) and (DomainName, CheckName) are keys in this table

8.3.7 Role\$Index

| Field | DataType | Description |
|----------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Table | Char | The current name of the table |
| Name | Char | The name of the index (see note) |
| Flags | Int | Sum of values in table below |
| RefTable | Char | Name of referenced table (or null) |
| RefIndex | Char | Name of referenced index (or null) |
| Distinct | Int | Number of distinct values |
| Adapter | Char | Name of adapter function or method (or null) |
| Rows | Int | The number of rows in the index |

User indexes are not supported in SQL2016. Pyrrho builds indexes automatically for all primary, unique, and foreign keys (there is no CREATE INDEX) in order to enforce integrity and referential constraints. They have names like U(67). (Pos) is the key for this table

| Flag | Meaning |
|------|--------------------|
| 1 | Primary Key |
| 2 | Foreign Key |
| 4 | Unique |
| 8 | Descending |
| 16 | Restrict Update |
| 32 | Cascade Update |
| 64 | Set Default Update |
| 128 | Set Null Update |
| 256 | Restrict Delete |
| 512 | Cascade Delete |
| 1024 | Set Default Delete |
| 2048 | Set Null Delete |

The Restrict flags are currently unused, since RESTRICT is the default and is only overridden if CASCADE or SET NULL has been set.

8.3.8 Role\$IndexKey

| Field | DataType | Description |
|-------------|----------|---|
| IndexName | Char | The name of the index |
| TableColumn | Char | The current name of the column |
| Position | Int | Zero-based column position in the index |
| Flags | Char | Blank except for Mongo |

(IndexName, TableColumn) and (IndexName, Position) are keys in this table.

8.3.9 Role\$Java

| Field | DataType | Description |
|------------|----------|--|
| Name | Char | The name of a base table or view, with the same name as the class |
| Key | Char | A comma separated list of the key columns of this object if any |
| Definition | Char | A Java class definition suitable for receiving rows of this object. See also Role\$Class |

Dots in top-level column names coming from views are automatically replaced by underscores.

8.3.10 Role\$Method

| Field | DataType | Description |
|------------|----------|--|
| Name | Char | The identifier for the type |
| Method | Char | The name of the method |
| Arity | Int | The number of parameters |
| MethodType | Char | Instance, Constructor, Static, or Overriding |
| Definition | Char | The method body |

| | | |
|---------|------|-----------------|
| Definer | Char | The owning role |
|---------|------|-----------------|

(Name, Method, Arity) is the key in this table.

8.3.11 Role\$Object

| Field | DataType | Description |
|-------------|----------|--|
| Type | Char | The type of database object, e.g. Table, Role etc |
| Name | Char | The current name of the database object |
| Source | Char | The transaction provenance at the time of creation |
| Output | Char | Metadata |
| Description | Char | Metadata |
| Reference | Char | Metadata |
| Iri | Char | Metadata |

(Type, Name) are keys in this table. For the available Metadata flags see section 7.2 (page 51 at the last count)

8.3.12 Role\$Parameter

| Field | DataType | Description |
|-------|----------|---|
| Pos | Char | System key (position information) for the procedure or method |
| Seq | Int | The ordinal of the parameter |
| Name | Char | The name of the parameter |
| Type | Char | The name of the parameter's type |
| Mode | Char | In or None, Out, InOut, Result |

(Pos, Seq) is the key in this table

8.3.13 Role\$PrimaryKey

| Field | DataType | Description |
|---------|----------|---|
| Table | Char | The current name of the table |
| Ordinal | Int | The position of the column in the primary key |
| Column | Char | The current name of the column |

(Table, Ordinal) is the key in this table

8.3.14 Role\$Privilege

| Field | DataType | Description |
|------------|----------|--|
| ObjectType | Char | The kind of object for which the privilege is granted |
| Name | Char | The name of the object for which the privilege is granted: for columns, methods etc this may have form id.id.. |
| Grantee | Char | The Grantee name |
| Privilege | Char | The privilege granted |
| Definer | Char | The owning role of the granted object |

(ObjectType, Name, Grantee) is the key in this table. Tables can have delete permission in this table, but Select, Insert and Update apply to columns.

8.3.15 Role\$Procedure

| Field | DataType | Description |
|------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Name | Char | The current name of the Procedure or Function |
| Arity | Int | The number of parameters. |
| Returns | Char | The return type (Null for a Procedure) |
| Definition | Char | The string containing the procedure or function definition |
| Inverse | Char | The name of the inverse function if any |
| Monotonic | Boolean | Whether the function has been declared monotonic |
| Definer | Char | The owning role (body will run as this role) |

The Definition starts from the beginning of the parameter list. (Pos) and (Name, Arity) are keys in this table.

8.3.16 Role\$Python

| Field | DataType | Description |
|------------|----------|---|
| Name | Char | The name of a base table or view, with the same name as the class |
| Key | Char | A comma separated list of the key columns of this object if any |
| Definition | Char | A Python class definition suitable for receiving rows of this object. See also Role\$Class |

8.3.17 Role\$Subobject

| Field | DataType | Description |
|-------------|----------|---|
| Type | Char | The type of database object, e.g. Table, Role etc |
| Name | Char | The current name of the database object |
| Seq | Int | The ordinal position of the column |
| Column | Char | The name of the column |
| Output | Char | Metadata |
| Description | Char | Metadata |
| Iri | Char | Metadata |

The primary key in this table is (Type, Name, Seq). For the available Metadata flags see section 7.2 (page 51 or thereabouts). TableColumns are found in the Role\$Object table: the Role\$Subobject table is for columns in views and the tables returned from functions.

8.3.18 Role\$Table

| Field | DataType | Description |
|------------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| Name | Char | The name of the Table |
| Columns | Int | The number of columns |
| Rows | Int | The number of rows |
| Triggers | Int | The number of triggers |
| CheckConstraints | Int | The number of Table check constraints |
| References | Int | The number of references |
| RowIri | Char | The Iri type constraint for rows if defined |

Base tables are entered in this table. Entries are made in Sys\$Table also for anonymous row types, with names such as "ROW(F INT,G CHAR)". (Pos) and (Name) are keys in this table.

8.3.19 Role\$TableCheck

| Field | DataType | Description |
|-----------|----------|--|
| Pos | Char | System key (position information) for the current object |
| TableName | Char | The current identifier for the Table |
| CheckName | Char | The identifier for the CHECK (unique per table) |
| Select | Char | The QueryExpression used to check the VALUE |

(Pos) and (TableName,CheckName) are keys in this table

8.3.20 Role\$TablePeriod

| Field | DataType | Description |
|-------------------|----------|--|
| Pos | Char | System key (position information) for the current object |
| TableName | Char | The current name of the table |
| Period Name | Char | The name of the Period (e.g. SYSTEM_TIME) |
| PeriodStartColumn | Char | The name of the system time period start column |
| PeriodEndColumn | Char | The name of the system time period end column |
| Versioning | Boolean | Whether period versioning has been specified |

8.3.21 Role\$Trigger

| Field | DataType | Description |
|-----------|----------|---|
| Pos | Char | System key (position information) for the trigger |
| Name | Char | The name of the Trigger |
| Flags | Char | Before/After, Insert/Delete/Update |
| TableName | Char | The current name of the table concerned |
| Definer | Char | The definer role for the Trigger |

(Pos) and (Name) are keys in this table. Use Log\$Trigger to see the defining code.

8.3.22 Role\$TriggerUpdateColumn

| Field | DataType | Description |
|------------|----------|---|
| Pos | Char | System key (position information) for the trigger |
| Name | Char | The name of the Trigger |
| ColumnName | Char | Column for Update |

(Pos) and (Name,ColumnName) are keys in this table

8.3.23 Role\$Type

| Field | DataType | Description |
|---------------|----------|---|
| Pos | Char | System key (position information) for the current object |
| Name | Char | The identifier for the type |
| Supertype | Char | The name of the supertype |
| OrderFunc | Char | The name of the ordering function if specified |
| OrderCategory | Char | The string representation of the order category (see 9.2.8) |
| WithURI | Char | Uri specified in With field |
| Definer | Char | The owning role |

Other details are given in the Sys\$Domain table. (Pos) and (Name) are keys in this table.

8.3.24 Role\$View

| Field | DataType | Description |
|---------|----------|--|
| Pos | Char | System key (position information) for the current object |
| View | Char | The current VIEW identifier |
| Select | Char | The current corresponding query expression |
| Struct | Char | The name of the structure type (OF) if any |
| Using | Char | The name of the GET USING table if any |
| Definer | Char | The owning role |

(Pos) and (View) are keys in this table.

8.4 Log\$ table collection

The Log\$ tables generally identify all objects by (long) integer values, shown in the Sys\$ tables as Pos, and in the Log\$ tables as DefPos (the defining position of the object, i.e. the log entry which records the creation of the object).

Exceptions to this rule are view, check and procedure definitions, where the actual string used in the definition contains the names of referenced objects at the time the definition was made³⁹. The current state of the definition can be obtained from the system tables (definitions are automatically updated if tables and columns are renamed).

Tables in this collection are read-only. They are publicly available in the personal edition (with the recommended firewall configuration, see section 3, this means available on the local subnet or local machine). They are always available to the database owner.

³⁹ See also the footnote on this topic to section 3.6.

8.4.1 Log\$

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Desc | char | A semi-readable version of the log information |
| Type | Char | The type of log entry (see table below) |
| Affects | Char | The object affected* |
| Transaction | Char | The transaction this log entry belongs to |

*Affects is a single defining position added to this table for convenience (it is not actually in the log file). For log entries that cause cascading changes, there is no attempt to replay the action to obtain a full list of affected objects.

For some log records (e.g. Index), the Description field may be incomplete because dependent objects have been dropped from the database.

The Pos key enables machine-readable versions of the Log\$ to be obtained from the tables described in the following sections.

| Type | Further information in | Comments |
|--------------------|-------------------------------|----------------------------|
| Alter | Log\$Alter | Alter column properties |
| AlterRowIri | | |
| Authenticate | Log\$Authenticate | |
| Change | Log\$Change | Rename object |
| Check | Log\$Check | |
| Checkpoint | | |
| Column | Log\$Column | |
| ColumnPath | | |
| Curated | | |
| Delete | Log\$Delete | |
| DeleteReference1 | | |
| Domain | Log\$Domain | |
| Drop | Log\$Drop | |
| Edit | Log\$Edit | Alter domain properties |
| Grant | Log\$Grant | |
| Index | Log\$Index, Log\$IndexKey | |
| Metadata | Log\$Metadata | |
| Method | Log\$TypeMethod | |
| Method | Log\$TypeMethod | |
| Modify | Log\$Modify | Alter proc/func/method |
| Namespace | | |
| Ordering | Log\$Ordering | |
| Partition | | |
| Partitioned | | |
| PDateType | Log\$DateType | |
| PeriodDef | Log\$PeriodDef | |
| PImportTransaction | Log\$Transaction | <i>no longer supported</i> |
| PPeriodDef | Log\$TablePeriod | |
| PProcedure | Log\$Procedure | |
| PRecord1 | Log\$Insert, Log\$InsertField | |
| Procedure/Function | Log\$Procedure | |
| PTemporalView | Log\$TemporalView | <i>no longer supported</i> |
| PTransaction2 | Log\$TransactionParticipant | See Transaction |
| PType1 | Log\$Type | |
| Record | Log\$Insert, Log\$InsertField | |
| Reference1 | | |
| RestView | | See View |
| Revoke | Log\$Revoke | |
| Role | Log\$Role | |
| Table | Log\$Table, Log\$Column | |

| | | |
|-----------------|---|-----------------|
| Transaction | Log\$Transaction | |
| Trigger | Log\$Trigger, Log\$TriggerUpdateColumn | |
| TriggeredAction | | |
| Type | Log\$Type | See also Domain |
| Update | Log\$Update, Log\$InsertField | |
| User | Log\$User | |
| Versioning | Log\$Versioning | |
| View | Log\$View | |

8.4.2 Log\$Alter

| Field | Data Type | Description |
|-------------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| DefPos | Char | The defining position of the object |
| Transaction | Char | The transaction this log entry belongs to |

The new column information is recorded in the Log\$Column table (for the same Pos).

8.4.3 Log\$Change

| Field | Data Type | Description |
|-------------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| Previous | Char | The previous log entry for the affected object |
| Name | Char | The new name for the object |
| Transaction | Char | The transaction this log entry belongs to |

8.4.4 Log\$Check

| Field | Data Type | Description |
|-------------|-----------|--|
| Pos | Char | System key (position information) for the log entry |
| Ref | Char | The database table or domain referred to |
| ColRef | Char | The column referred to (if not -1) |
| Name | Char | The original name of the constraint (possibly system supplied e.g. U(nnn)) |
| Check | Char | The source code for the check condition |
| Transaction | Char | The transaction this log entry belongs to |

8.4.5 Log\$Classification

This table contains all log entries for database objects that change the classification. For Records see Log\$Update.

| Field | Data Type | Description |
|----------------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| Obj | Char | The defining position of the object affected |
| Classification | Char | D to A |
| Transaction | Char | The transaction this log entry belongs to |

8.4.6 Log\$Clearance

This table contains all log entries that change the clearance of users.

| Field | Data Type | Description |
|-------------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| User | Char | The defining position of the user affected |
| Clearance | Char | D to A |
| Transaction | Char | The transaction this log entry belongs to |

8.4.7 Log\$Column

| Field | Data Type | Description |
|-------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| Table | Char | The defining position of the table |

| | | |
|------------------|-------------|---|
| Name | Char | The original name of the column |
| Seq | Int | The ordinal position of the column (used in select *) |
| Domain | Char | The associated domain (usually system supplied) |
| Default | Char | Source code for generating a default value |
| NotNull | Boolean | Whether the column must have a non-null value |
| Generated | Boolean | Whether GENERATED ALWAYS |
| Update | Char | The update assignment rule for a generated column |
| <i>RefTable</i> | <i>Char</i> | <i>The reflection table defining position</i> |
| <i>RefIndex</i> | <i>Char</i> | <i>The referencing index for reflection</i> |
| <i>RefIndex2</i> | <i>Char</i> | <i>The secondary index for many-many reflection</i> |
| Transaction | Char | The transaction this log entry belongs to |

8.4.8 Log\$DateType

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The original name for the date type domain |
| Kind | Char | The base type of the date type (e.g. INTERVAL) |
| StartField | Char | The start field for the date type |
| EndField | Char | The end field for the date type |
| Transaction | Char | The transaction this log belongs to |

8.4.9 Log\$Delete

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the delete operation |
| DelPos | Char | The defining Pos for the record |
| Transaction | Char | The transaction this log belongs to |

8.4.10 Log\$Domain

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the log entry |
| Kind | Char | Domain, Edit, or Type |
| Name | Char | The name of the domain or type |
| DataType | Int | Describes the data type |
| DataLength | Int | Length of the data type |
| Scale | Int | Scale factor for numerics |
| Charset | Char | Character set identifier |
| Collate | Char | The collation identifier |
| Default | Char | String representation of default value |
| StructDef | Char | Domain reference for MULTiset or ARRAY element, or Table reference for ROW or TYPE element |
| Transaction | Char | The transaction this log entry belongs to |

8.4.11 Log\$Drop

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| DelPos | Char | The defining position of the object being deleted |
| Transaction | Char | The transaction this log entry belongs to |

8.4.12 Log\$Edit

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the Alter Domain operation |
| Prev | Char | The previous log record for the domain |
| Transaction | Char | The transaction this log belongs to |

8.4.13 Log\$Enforcement

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the Alter Domain operation |
| Table | Char | The defining position of the table |
| Flags | Int | Enforecemnt flags (read,insert,update,delete) see 9.2.7 |
| Transaction | Char | The transaction this log belongs to |

8.4.14 Log\$Grant

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Privilege | Int | Describes the privilege granted |
| Object | Char | The object for which the grant is made |
| Grantee | Char | The object gaining the privilege |
| Transaction | Char | The transaction this log entry belongs to |

8.4.15 Log\$Index

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The name of the index (system generated, e.g. U(nnn)) |
| Table | Char | The table on which this index is defined |
| Flags | Int | Describes this index, see 8.1.8 |
| Reference | Char | Identifies the referenced index |
| Adapter | Char | Name of adapter function or method (or null) |
| Transaction | Char | The transaction this log entry belongs to |

8.4.16 Log\$IndexKey

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| ColNo | Int | The ordinal position of the column in the key |
| Column | Char | Identifies the key column* |
| Transaction | Char | The transaction this log entry belongs to |

*Key information may not be available if the the table or column has been dropped.

8.4.17 Log\$Insert

| Field | DataType | Description |
|----------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Table | Char | The defining position of the table for the insert |
| SubType | Char | The defining position of the subtype if specified |
| Classification | Char | D to A |
| Transaction | Char | The transaction this log entry belongs to |

8.4.18 Log\$InsertField

| Field | DataType | Description |
|--------|----------|---|
| Pos | Char | System key (position information) for the current log entry |
| ColRef | Char | Identifies the column |
| Data | Char | String version of the data |

8.4.19 Log\$Metadata

| Field | DataType | Description |
|--------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| DefPos | Char | The defining position of the database object |
| Name | Char | The new name of the object as viewed from this role |

| | | |
|-------------|------|--|
| Description | Char | The object description for this role |
| Output | Char | Output flags (e.g. Attribute, Entity) |
| RefPos | Char | The defining position referred to (if any) |
| Iri | Char | Web metadata for this role |
| Transaction | Char | The transaction this log entry belongs to |

8.4.20 Log\$Modify

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the log entry |
| DefPos | Char | The defining position of the proc/func/method being modified |
| Name | Char | The new name of the object; or update assignments for Column; for View, one of Name,Query,Update,Insert,Delete |
| Body | Char | The modified source code of the proc/func; for View Name, the new name |
| Transaction | Char | The transaction this log entry belongs to |

8.4.21 Log\$Ordering

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| TypeDefPos | Char | The defining position of the type being ordered |
| FuncDefPos | Char | The defining position of the function or method |
| OrderFlags | Int | The ordering category flags (see 9.2.8) |
| Transaction | Char | The transaction this log entry belongs to |

8.4.22 Log\$Procedure

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The original name of the procedure |
| Arity | Int | Number of parameters |
| RetDefPos | Char | The defining position of the return type |
| Proc | Char | The original source code of the proc/func (including the formal params) |
| Transaction | Char | The transaction this log entry belongs to |

8.4.23 Log\$Revoke

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for the log entry |
| Privilege | Int | Identifies the privilege being revoked |
| Object | Char | The object to which the privilege relates |
| Grantee | Char | The grantee from whom the privilege is being withdrawn |
| Transaction | Char | The transaction this log entry belongs to |

8.4.24 Log\$Role

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The Name of the role |
| Details | Char | The description of the intended use of the role |
| Transaction | Char | The transaction this log entry belongs to |

8.4.25 Log\$Table

| Field | DataType | Description |
|--------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The original name of the table |
| Defpos | Char | The previous def for this table (or null) |

| | | |
|-------------|------|---|
| Iri | Char | The iri for the table |
| Output | Char | Entity, Lookup, or null |
| Details | Char | The description of the table |
| Iri | Char | Web metaadata |
| Transaction | Char | The transaction this log entry belongs to |

A table record is made for table and row type declarations and for modifications to this metdata.

8.4.26 Log\$TablePeriod

This table records details of period type definitions.

| Field | DataType | Description |
|-------------|----------|--|
| Pos | Char | System key (position information) for this log entry |
| Table | Char | The defining position of the table |
| PeriodName | Char | Thye original name of the period (or SYSTEM_TIME) |
| Versioning | Boolean | Whether system versioning is specified |
| StartColumn | Char | The defining position of the system time period start column |
| EndColumn | Char | The defining position of the system time period end column |

8.4.27 Log\$Transaction

| Field | DataType | Description |
|-------|-----------|---|
| Pos | Char | System key (position information) for the log entry |
| NRecs | Int | The number of log entries following |
| Time | TimeStamp | A timestamp |
| User | Char | Identifies the current user |
| Role | Char | Identifies the current role |

8.4.28 Log\$TransactionParticipant

| Field | DataType | Description |
|-------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Path | Char | A participating database |
| PPos | Char | The system key on the participating database |

8.4.29 Log\$Trigger

The table records trigger definitions.

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The original name of the trigger |
| Flags | Char | Before/After, Insert/Delete/Update |
| Table | Char | The identifier of the table concerned |
| OldRow | Char | Referencing identifier for old row |
| NewRow | Char | Referencing identifier for new row |
| OldTable | Char | Referencing identifier for old table |
| NewTable | Char | Referencing identifier from new table |
| Def | Char | The original code for the trigger including WHEN if defined |
| Transaction | Char | The transaction this log entry belongs to |

8.4.30 Log\$TriggerUpdateColumn

This table provides details for trigger definitions.

| Field | DataType | Description |
|--------|----------|---|
| Pos | Char | System key (position information) for the trigger |
| Column | Char | Column for Update |

8.4.31 Log\$TriggeredAction

| Field | DataType | Description |
|-------|----------|-------------|
|-------|----------|-------------|

| | | |
|-------------|------|--|
| Pos | Char | System key (position information) for the log entry |
| Trigger | Char | Identifies the defining position of the trigger that is starting |
| Transaction | Char | The transaction in which this action occurs |

Entries of this type in the log show a change of responsibility from the user and role starting the transaction to the defining user and owning role of the trigger.

8.4.32 Log\$Type

| Field | DataType | Description |
|-----------|----------|--|
| Pos | Char | System key (position information) for the Domain log entry |
| SuperType | Char | Identifies the defining log entry for the supertype |
| WithUri | Char | The Uri provided with the representation if any |

The type name is given in the Log\$Domain table. The list of methods is in the Log\$TypeMethod table. The list of members is in the Log\$Table table. The method bodies are in the Log\$Modify table.

8.4.33 Log\$TypeMethod

This table records method declarations.

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) identifying this method |
| Type | Char | Identifies the defining log entry for the type |
| MethodType | Int | See coding below |
| Name | Char | The original name of the method |
| Transaction | Char | The transaction this log entry belongs to |

| Value | MethodType |
|-------|-------------|
| 0 | Instance |
| 1 | Overriding |
| 2 | Static |
| 3 | Constructor |

Method bodies are given in the Log\$Modify table.

8.4.34 Log\$update

| Field | DataType | Description |
|----------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| DefPos | Char | Identifies the defining log entry for the record |
| Table | Char | Identifies the table for the update |
| SubType | Char | Identifies the subtype if any |
| Classification | Char | D to A |
| Transaction | Char | The transaction this log entry belongs to |

8.4.35 Log\$User

This table records the first occurrence of a user identity in the database.

| Field | DataType | Description |
|-------------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The name of the user |
| Transaction | Char | The transaction this log entry belongs to |

8.4.36 Log\$View

This table records view definitions.

| Field | DataType | Description |
|--------|----------|---|
| Pos | Char | System key (position information) for the log entry |
| Name | Char | The original name of the View |
| Select | Char | The original query expression defining the view |
| Struct | Char | The defining position of the structure type (OF) if any |

| | | |
|-------------|------|---|
| Using | Char | The defining position of the GET USING table if any |
| Transaction | Char | The transaction this log entry belongs to |

8.5 Table and Cell Logs

In auditing databases (section 3.5), it is convenient to be able to review all insert, update, and delete operations for a specific table, or for a specific cell. Pyrrho provides table and cell log facilities to do this, provisionally referred to as ROWS(nnnn) and ROWS(rrr,ccc) where nnnn is the numeric identifier of the table in question, rrr the defining position of the desired row, and ccc that of the desired column.

8.5.1 A Table Log

Pyrrho provides a table log facility, provisionally referred to as ROWS(nnnn) where nnnn is the numeric identifier of the table in question. ROWS(nnnn) is a table with the following fields:

| Field | Data Type | Description |
|-------------|-----------|--|
| Pos | Int | System key (position information) for the log entry |
| Action | Char | “Insert”, “Update”, or “Delete”* |
| Transaction | Int | The transaction this log entry belongs to |
| cccc | Cell | The value specified for the column with identifier ccccc |

*Entries from cascading updates and deletes are not included in this table.

This feature allows data to be recovered even where columns have been removed (by ALTER TABLE or even DROP TABLE).

8.5.2 A Cell Log

Pyrrho provides a cell log facility, provisionally referred to as ROWS(rrr,ccc) where rrr is the defining position of the row containing the cell, and ccc the defining position of the column in question. ROWS(rrr,ccc) is a table with the following fields:

| Field | Data Type | Description |
|------------------|-----------|--|
| Pos | Int | System key (position information) for the log entry |
| Value | Cell | The value |
| StartTransaction | Int | The transaction responsible for placing this value |
| StartTimestamp | Timestamp | The timestamp for the StartTransaction |
| EndTransaction | Int | The transaction responsible for replacing this value |
| EndTimestamp | Timestamp | The timestamp for the EndTransaction |

This feature allows data to be recovered even where the row and/or even the column or table has been removed (by DELETE, or ALTER TABLE, or DROP TABLE).

8.6 Transaction Profiling

From version 4.2, Pyrrho can profile transactions, and this is useful for troubleshooting transaction conflicts. Profiling has a negligible effect on performance and memory use. Profiling can be enabled for all databases, or in the configuration of individual databases.

The purpose of gathering or storing profile information is to understand and monitor the causes of transaction conflicts. Performance tuning and database design should seek to minimise failed transactions during normal operation. It is inevitable that an unusual operation, such as changing the schema or making an update affecting all rows of a table, will be hard to commit during heavy traffic, because a conflicting transaction will probably occur in the meantime.

Except for the Silverlight edition, when profiling is turned off or on for a database called *name* profiling information is destructively saved as or if available loaded from an XML document with name *name.xml*. Thus a database administrator can carefully take a database offline by throttling, and then turning off profiling to record a snapshot before shutting down a server, and in this way a full profile of normal operations can be maintained. This level of completeness for profile information will not be achieved if the database server is simply killed.

If profiling is enabled, any transaction conflict exception will report its profile id (see section 8.7.1). The system profile table will contain the number of successful and failed transactions recorded for this profile

based on the available information from recorded periods of full profiling (or since the time profiling was enabled for the server).

If profiling is turned on, the tables described in this section enable inspection of the real-time state of the profile information, always excluding any information about transactions in progress. The profile viewer described in section 4.6 obtains profile information from these tables or from the XML document, and also groups profiles with similar pattern (for example where everything is the same apart from the number of affected rows).

8.6.1 Profile\$

This table records the transaction profiles for the database.

| | | |
|-------------|---------|--|
| Id | Int | The transaction profile identity |
| Occurrences | Int | The number of times this profile has occurred |
| Fails | Int | The number of failures recorded for this profile |
| Schema | Boolean | Whether this transaction includes schema changes |

Further details for this profile are contained in the following tables.

8.6.2 Profile\$ReadConstraint

| Field | DataType | Description |
|--------------|----------|--|
| Id | Int | The transaction profile identity |
| Table | String | The current name of the table |
| ColPos | Char | The defining position of a read column whose update is blocked |
| ReadCol | String | The current name of a read column whose update is blocked |

8.6.3 Profile\$Record

| Field | DataType | Description |
|--------------|----------|---|
| Id | Int | The transaction profile identity |
| Table | String | The current name of the table |
| Rid | Char | The record profile identity |
| Recs | Int | The number of records altered with this profile |

8.6.4 Profile\$RecordColumn

This table records the columns containing added or updated data in a record profile

| Field | DataType | Description |
|--------------|----------|---|
| Id | Int | The transaction profile identity |
| Table | String | The current name of the table |
| Rid | Char | The record profile identity |
| ColPos | Char | The defining position of an affected column |
| RecCol | String | The current name of an affected column |

8.6.5 Profile\$Table

This table records the profile of delete operations for a specific table as well as providing information about update blocking.

| | | |
|--------------|---------|---|
| Id | Int | The transaction profile identity |
| Table | String | The current name of the table |
| BlockAny | Boolean | This profile blocks on any concurrent update of the table |
| Dels | Int | The number of deletions in a transaction |
| Index | Char | The defining position of an index with specific records |
| Pos | Char | The defining position of the table |
| ReadRecs | Int | The number of specific records whose update is blocked |
| Schema | Boolean | Whether the profile changes the table schema |

If BlockAny is true, Index and ReadRecs will be 0; and if there are Profile\$ReadColumn entries blocking is limited to these columns.

8.7 Pyrrho Class Library Reference

Any application using Pyrrho should include just one of the following dlls: PyrrhoLink.dll, OSPLink.dll, EmbeddedPyrrho.dll, OSP.dll, and SilverlightOSP.dll. Apart from Silverlight, the client-side API is derived from ADO.NET. Although ADO.NET is not available in Silverlight, the SilverlightOSP.dll API is designed to be similar.

Except where noted, all of these dlls define (export) the following classes, which are described in the following subsections:

SQL2016 API:

| Class | Subclass of | Description |
|----------------|---------------------------|--|
| Date | | Data type used for dates. |
| PyrrhoArray | | Data type used for ARRAY and MULTISSET if a column of type ARRAY or MULTISSET has been added to the table. |
| PyrrhoColumn | | Helps to describe the columns of a table or structured type |
| PyrrhoConnect | System.Data.IDbConnection | Establishes a connection with a Pyrrho DBMS server, and provides additional methods and properties. |
| PyrrhoDocument | | This class allows editing of embedded Documents (in the sense of MongoDB) |
| PyrrhoInterval | | This class is used to represent a time interval |
| PyrrhoRow | | Data type used for ROW fields in a database table, a column of type ROW can be added to the table. (SQL2016) |

Exceptions:

| Class | Subclass of | Description |
|---------------------|------------------|---|
| DatabaseError | System.Exception | Used for “user” exceptions, e.g. a specified table or column does not exist, an attempt is made to create a table or column that already exists, incorrect SQL etc. The message property gives a readable explanation. see section 8.1. |
| TransactionConflict | DatabaseError | The action attempted has conflicted with a concurrent transaction, e.g. two users have attempted to update the same cell in a table. The changes proposed by the current transaction have been rolled back, because the database contents have been changed by the other transaction. |

| Class | Subclass of | Description |
|----------------|-------------|-------------|
| PyrrhoTable | | |
| PyrrhoTable<T> | PyrrhoTable | |

PHP support:

| Class | Subclass of | Description |
|---------------|-------------|--|
| ScriptConnect | | Provided for PHP support (section 6.7) |
| ScriptReader | | Provided for PHP support (section 6.7) |

8.7.1 DatabaseError

The methods and properties of DatabaseError are:

| Method or Property | Explanation |
|--------------------------------|---|
| Dictionary<string,string> info | Information placed in the error: the keys specified in the SQL standard are CLASS_ORIGIN, SUBCLASS_ORIGIN, CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CATALOG_NAME, SCHEMA_NAME, |

| | |
|-----------------|---|
| | TABLE_NAME, COLUMN_NAME, CURSOR_NAME, MESSAGE_TEXT. Pyrrho adds PROFILE_ID if profiling is enabled. |
| String Message | The reason for the exception (inherited from Exception): this can be localised as described in section 3.8. |
| String SQLSTATE | The signal sent from the DBMS: usually a five character string beginning with a digit such as “2N000”. Many of these codes are defined in the SQL standard. |

8.7.2 Date

The methods and properties of Date are:

| Method or Property | Explanation |
|--------------------|--|
| DateTime date | The underlying DateTime value |
| Date(DateTime d) | Constructor. |
| string ToString() | Overridden: Formats the date using DateTime.ToShortDate() which is locale-specific |

For the embedded editions, Pyrrho.Common.Date is equivalent.

8.7.3 DocArray

| Property | Explanation |
|-----------------------------------|--|
| DocArray(string s) | Create a DocArray from JSON. |
| C[] Extract<C>(params string[] p) | Extract instances of C from a DocArray. C must have a public parameterless constructor. P is a path of fields in the documents of the array. |
| List<object> fields | A document array consists of an array of documents |

8.7.4 Document

PyrrhoConnect.Get/Post/Put/Delete can be used for whole Documents and BSON, Json and XML formats are supported. This class can be used to access fields within Documents and to convert to and from Json and XML Note: this class remembers the connection to the database if any, and all these changes are transacted in the database unless the Document is detached or the connection is closed.

| Method or Property | Explanation |
|---|---|
| bool Contains(string k) | Tests if there is a field k in the top level of the document |
| Document() | Constructor: a new empty Document |
| Document(object) | Constructor: reflection is used to build a Document based on the public fields of the given parameter |
| Document(string) | Constructor: the string should be JSON. |
| C[] Extract<C>(params string[]) | Reflection using class C is used recursively to extract instances of C from this document, starting at a place indicated by the given path of keys. |
| List<KeyValuePair<string,object> fields | The content of the Document (accessed using this[]) |
| object this[string] | Access a field of the document. |
| string ToString() | Convert a document to Json |

8.7.5 DocumentException

This subclass of Exception is used to report parsing errors in Document parameters.

8.7.6 ExcludeAttribute

Mark a public field of a Versioned class with the [Exclude] attribute to avoid its use in Put/Post.

8.7.7 FieldAttribute

Class definitions obtained from Role\$Class have some fields marked [Field..] if necessary for validation purposes. In many cases, including UDTs, arrays, multisets etc the attribute is not required.

| Attribute form | Explanation |
|-------------------------------------|--|
| [Field(PyrrhoDbType t)] | Pyrrho's data type is t |
| [Field(PyrrhoDbType t,int n)] | Pyrrho's data type is t, maximum length/precision is n |
| [Field(PyrrhoDbType t,int n,int s)] | Pyrrho's data type is t, with precision n, scale s |

see 8.7.9.

8.7.8 KeyAttribute

Class definitions obtained from Role\$Class have key fields marked [Key(0)] etc.

| Attribute form | Explanation |
|----------------|---|
| [Key(int n)] | The field is the nth component of the base table key (0 is the first) |

8.7.9 PyrrhoArray

| Method or Property | Explanation |
|-----------------------|--|
| PyrrhoArray(object[]) | |
| string kind | "ARRAY" or "MULTISET" |
| object[] data | The values of the array or multiset. Note that the ordering of multiset values is non-deterministic and not significant. |

8.7.10 PyrrhoColumn

The methods and properties of PyrrhoColumn are:

| Method or Property | Explanation |
|--------------------|---|
| bool AllowDBNull | Whether the column can contain a null value |
| string Caption | The name of the column |
| string DataType | The domain or type name of the column |
| bool ReadOnly | Whether the column is read-only |

8.7.11 PyrrhoCommand

PyrrhoCommand implements IDbCommand or imitates it.

From version 5.4, thread-safety is enforced for client-side programming. PyrrhoCommand cannot be shared among threads because methods of the IDbCommand class might be used in another thread to modify the command. PyrrhoConnect can be shared among threads, but there can be at most one command active at any time per connection. As a result, methods such as ExecuteReader will block until the connection is available.

| Method or Property | Explanation |
|---|---|
| string CommandText | The SQL statement for the Command |
| IDbDataParameter CreateParameter() | The returned object is a PyrrhoParameter. |
| PyrrhoReader ExecuteReader() | Initiates a database SELECT and returns a reader for the returned data (as in IDataReader). Will block until the connection is available. |
| PyrrhoReader ExecuteReaderCrypt() | Initiates a database SELECT and returns a reader for the returned data (as in IDataReader). The results are not encrypted. Will block until the connection is available. |
| object ExecuteScalar() | Initiates a database SELECT for a single value. Will block until the connection is available. |
| object ExecuteScalarCrypt() | Initiates a database SELECT for a single value. Will block until the connection is available. |
| int ExecuteNonQuery(params Versioned[]) | Initiates some other sort of Sql statement and returns the number of rows affected. If the transaction automcommits, the given versioned objects have versions updated if affected. Will block until the connection is available. |
| Int ExecuteNonQueryCrypt() | Initiates some other sort of Sql statement and returns the number of rows affected. Will block until the connection is available. |

8.7.12 PyrrhoConnect

Depending on the Pyrrho version, PyrrhoConnect implements or imitates the IDbConnection interface and supplies some additional functionality. The following methods described here provide a RESTful interface*: Get, Post, Put and Delete.

From version 5.4, thread-safety is enforced for client-side programming. Although the PyrrhoConnect can be shared among threads, there can be at most one transaction and/or command active at any time per connection, and transactions, commands, and readers cannot be shared with other threads. As a result, methods such as BeginTransaction will block until the connection is available.

From v7 the Prepare and Execute methods of the connection manage and use a set of named SQL statements with ? as placeholders for zero or more Literal values. The Execute method provides these SQL literals as actual parameters for each such placeholder. The named statements are available in the PyrrhoConnect instance that defined them.

| Method or Property | Explanation |
|---|---|
| int Act(string sql) | Convenient shortcut to construct a PyrrhoCommand and call ExecuteNonQuery on it. Will block until the connection is available. |
| Activity activity | (AndroidOSP) Set only. Set the Activity into PyrrhoConnect. This must be done before the connection is opened. E.g. in Activity.OnCreate(bundle) use code such as <pre>conn = new PyrrhoConnect("Files=mydb"); conn.activity = this; conn.Open();</pre> Note that mydb (without the osp extension) needs to be an AndroidAsset to be copied to the device. |
| PyrrhoTransaction BeginTransaction() | Start a new isolated transaction (like IDbTransaction). Will block until the connection is available. [In Java, PyrrhoJC.Connection does this automatically if autoCommit has been set false.] |
| bool Check(string ch) bool Check(string ch, string rc) | Check to see if a given Versioned check string is still current, i.e. the row has not been modified by a later transaction. (See sec 5.2.3 and 8.8.21). The second version shown also tests the readCheck. (There is no need to perform a check unless the Versioned data is from a previous transaction.) |
| void Close() | Close the channel to the database engine |
| string ConnectionString | Get the connection string for the connection |
| PyrrhoCommand CreateCommand() | Create an object for carrying out an Sql command (as in IDbCommand). |
| void Delete(Versioned ob) | Delete the row corresponding to this object.* Will block until the connection is available. |
| int Execute (string name, params string[] actuals) | Execute the named prepared statement with the given actual parameters (given as SQL Literals). (For the more familiar ExecuteReader and ExecuteNonQuery, see PyrrhoCommand, sec 8.7.11). See Prepare() below. |
| PyrrhoReader ExecuteReader (string name, params string[] actuals) | |
| E[] FindAll<E>() | Retrieve all Versioned entities of a given type.* Will block until the connection is available. |
| E FindOne<E>(params IComparable[] w) | Retrieve a single entity of a given Versioned type E with key fields w.* |
| E[] FindWith<E>(string w) | Retrieve a set of Versioned entities satisfying a given condition. w is a comma-separated set of conditions of form <i>field=value</i> . Field names are case sensitive and values are in SQL format (single quotes on strings are optional in the absence of ambiguity).* Will block until the connection is available. |
| E[] Get<E>(string rurl) | The rurl should be a partial REST url (the portion following the Role component), that targets a class E in the client application.* Will block until the connection is available. |
| string[] GetFileNames | Returns the names of accessible databases. |
| PyrrhoColumn[] GetInfo(string dataType) | Get information about a datatype: the string must exactly match the datatypename of a table or type. |

| | |
|--|--|
| void Open() | Open the channel to the database engine |
| void Post(Versioned ob) | The object should be a new row for a base table.* If autoKey is set key field(s) containing default values (0,"" etc) in ob are overwritten with suitable new value(s). Will block until the connection is available. |
| void Prepare(string name, string sql) | Prepare a named statement. The sql can contain ? placeholders for actual parameters, which are supplied as SQL fragments in Execute. Prepared statements are local to the current PyrrhoConnect. The Prepare function should not be called within a transaction. |
| void Put(Versioned ob) | The object should be an updated version of an entity retrieved from or committed to the database.* Will block until the connection is available. |
| PyrrhoConnect(string cs) | Create a new PyrrhoConnect with the given connection string. Documentation about the connection string is in section 6.3. |
| void ResetReader() | Repositions the IDataReader to just before the start of the data |
| void SetRole(string s) | Set the role for the connection |
| E[] Update<E> (Document w, Document u) | Specifies a Document update operation on a Versioned class containing documents. Documents matching w are updated according to the operations in u, and the set of modified objects is returned. (See 8.8.4)* Will block until the connection is available. |
| DatabaseError[] Warnings | Warnings for the most recent operation on the connection |

* The Find., Get, Put, Post, Delete and Update methods assume that the Version subclasses corresponding to the relevant database tables have been installed in the application, for example using the sources provided by the Role\$Class system table (sec 8.4.1), so that the base table name matches the class name. These methods use .NET Reflection machinery to access public fields in the supplied object. If you add other public fields and properties to these classes, consider marking them with the [Exclude] attribute.

8.7.13 PyrrhoDbType

DbType in System.Data is used for DbParameters and is rather specific for SQL Server. Pyrrho's version of this is as follows:

| Value |
|-----------|
| DBNull |
| Integer |
| Decimal |
| String |
| Timestamp |
| Blob |
| Row |
| Array |
| Real |
| Bool |
| Interval |
| Time |
| Date |
| UDType |
| Multiset |
| Xml |
| Document |

8.7.14 PyrrhoInterval

The methods and properties of PyrrhoInterval are:

| Method or Property | Explanation |
|--------------------|--------------------------------------|
| int years | The years part of the time interval |
| int months | The months part of the time interval |
| long ticks | The ticks part of the time interval |

| | |
|----------------------------|---|
| static long TicksPerSecond | Gets the constant number of ticks per second |
| static string ToString() | Formats the above data as e.g. (0yr,3mo,567493820000ti) |

8.7.15 PyrrhoParameter

This class is Pyrrho's implementation of IDbDataParameter and IDataParameter. The only change introduced is that the native field type is publicly accessible. See PyrrhoDbType in 8.8.9 above.

8.7.16 PyrrhoParameterCollection

This is Pyrrho's implementation of DbParameterCollection.

8.7.17 PyrrhoReader

This class is Pyrrho's implementation of IDataReader. The only additional members of PyrrhoReader are:

| Method or Property | Explanation |
|-------------------------------|---|
| string DataSubtypeName(int i) | Returns the domain or type name of the actual type of the ith column in the current row. (Usually this will be the same as DataTypeName.) |
| string Description(int i) | Returns the description metadata of the ith column |
| T GetEntity<T>() | Used in strongly-typed PyrrhoReaders (as in ExecuteTable<T>) |
| string Output(int i) | Returns the output flag of the ith column |
| string Url(int i) | Returns the web metadata url of the ith column |

8.7.18 PyrrhoRow

PyrrhoRow is used only when required for values of structured types. The methods and properties of PyrrhoRow are:

| Method or Property | Explanation |
|----------------------------|--|
| List<PyrrhoColumn> columns | The names of the fields of the row |
| object[] data | The values of the fields (may be null). This is the indexer for PyrrhoRow (indexed by column number or column name). |
| string[] subTypes | The names of the actual types for the current row |

8.7.19 PyrrhoTable

A PyrrhoTable is constructed internally by every invocation of ExecuteReader. As in ADO.NET DataTable there are properties called Rows and Columns, and an array of PrimaryKey columns.

8.7.20 PyrrhoTransaction

This class imitates IDbTransaction, but provides an extra method: CommitAndReport()

| Method or Property | Explanation |
|---------------------------------|---|
| int Commit (params Versioned[]) | Commit the transaction and optionally fill in version information for a set of objects. Returns the number of records affected by deferred triggers. |
| bool Conflict | Gets whether a conflicting transaction has been committed since the start of this transaction. (Requires a round trip to the transaction master server.) If Conflict is true, a subsequent Commit will fail, but the transaction is not closed. |
| void Rollback() | Roll back the transaction |

8.7.21 SchemaAttribute

This attribute is for validation of class definitions, and is generated by Role\$Class.

| Attribute form | Explanation |
|--------------------|---|
| [Schema(long nnn)] | The value is the latest change for the entity type in the specified role. |

8.7.22 Versioned

Versioned is the base class for Pyrrho's entities as generated by Role\$Class. See the Check() function in PyrrhoConnect (sec 8.8.12).

| Field | Explanation |
|----------------------|---|
| string version | The value is the latest row version validator for the entity, which is a string returned by the server. For Pyrrho, the format is a comma-separated list of form <i>file:dfpos:version</i> . |
| <i>str</i> readCheck | A validator to check that the query used to retrieve the data would still return the same results. This test is conservative: validation will fail if the server cannot guarantee the same results. The server takes account of all data read during the transaction that gave the validator. |

8.7.23 WebCtrl

This class is from the AWebSvr library. Derived classes (e.g. XXController) should provide one of more of the standard HTTP methods GetXX, PutXX, PostXX, DeleteXX according to one or both of the following templates:

```
public static string VERBXX(WebSvc ws, Document d)
public static string VERBXX(WebSvc ws, params object data)
```

The value returned should be the response string for sending to the client.

| Field | Explanation |
|-------------------------------|--|
| virtual bool AllowAnonymous() | Can be overridden by a subclass. The default implementation returns false, but anonymous logins are always allowed if no login page is supplied (Pages/Login.htm or Pages/Login.html). |

8.7.24 WebSvc

This class is from the AWebSvr library. Your custom web server/service instance(s) will indirectly be subclasses of this class, so will have access to its protected fields and methods documented here.

Controllers should be added in a static method, e.g. in Main()

Derived classes typically organise a connection to the DBMS being used. The connection can be for the service or for the request, and so should be set up in an override of the Open method.

| Field | Explanation |
|---|--|
| static void Add(WebCtrl wc) | Install a controller for the service. |
| virtual bool Authenticated() | Override this to discriminate between users. By default the request will be allowed to proceed if AllowAnonymous is set on the controller or there is no login page. Get user identities etc from the context. |
| virtual void Close() | Can be overridden to release request-specific resources. |
| System.Net.HttpListenerContext context | Gives access to the current request details. |
| <i>dict</i> controllers | The controllers for the service. Make sure you add controller to this dictionary. |
| static System.Collections.Generic.Dictionary<string, WebCtrl> controllers | The controllers defined for the service. |
| string GetData() | Extracts the HTTP data supplied with the request: a URL component beginning with { will be converted to a Document. |
| virtual void Log(string verb, System.Uri u, string postData) | Write a log entry for the current controller method. The default implementation appends this information to Log.txt together with the user identity and timestamp. |
| virtual void Open (System.Net.HttpListenerContext cx) | Can be overridden by a subclass, e.g. to choose a database connection for the current request. The default implementation does nothing. |
| <i>Serve()</i> | <i>Calls the requested method using the above templates. Don't call this method directly.</i> |

8.7.25 WebSvr

This class is from the AWebSvr library. Your custom web server should be a subclass of WebSvr, and WebSvr is a subclass of WebSvc. It defines the URL prefixes (including hostnames and port numbers) for the service. If your service is multi-threaded, you can override the Factory method to returning a new instance of your WebSvc subclass. Finally, call either of the two Server methods to start the service loop.

| Field | Explanation |
|--|---|
| virtual WebSvc Factory () | Can be overridden by a subclass to create a new service instance. The default implementation returns this (for a single-threaded server). |
| void Server(params string[] prefixes) | Starts the server listening of a set of HTTP prefixes (up to the appName), with anonymous authentication. |
| void Server(System.Net.AuthenticationSchemes au, params string[] prefixes) | Starts the server listening of a set of HTTP prefixes (up to the appName), with the given authentication scheme(s). |

8.8 The Pyrrho protocol

The "Pyrrho protocol" defines the binary traffic between the client and server. (Note that this is different from the "PyrrhoDb protocol" mentioned in section 6.13, which is actually implemented on the client side by class PyrrhoWebRequest in file PyrrhoDbClient.cs).

In the following discussion, ints are coded in 4 octets as signed 32-bit quantities, most significant octet first, and longs are 8 octets. A String is always coded in UTF8 invariant-culture Unicode, prefixed by an int giving the number of octets in the string data.

Localisation is handled by the client library.

8.8.1 Low level-communication

As soon as the TCP connection to the server is established, the server sends a long to the client. This is a nonce used for encrypting the connection string.

Client replies with octet 0x0 .

Since version 1.0, the low-level communication uses asynchronous buffering, with the help of the class AsyncStream. All communication between client and server uses 2048-octet buffers, which normally contain in the first two octets (octets 0 and 1) the count of valid bytes that follow in the buffer (i.e. this count is in range 0..2046.)

Since version 2.0, this mechanism has been modified to provide better support for exceptions reported by the server during transmission of data (e.g. during PutRow()). If the count appears to be 2047, the buffer contains an exception record instead, in which the next two octets (octets 2 and 3) the count of octets used to transmit the exception details. On the server side, this exception mechanism is supported by AsyncStream.StartException(). On the client side, there is a corresponding AsyncStream.GetException().

The following protocol bytes are supported (enumeration PyrrhoBase.Protocol).

| Protocol Name | Byte |
|-----------------------|------|
| Authority | 13 |
| BeginTransaction | 6 |
| Check | 42 |
| CheckConflict | 32 |
| CloseConnection | 9 |
| CloseReader | 5 |
| Commit | 7 |
| CommitAndReport | 43 |
| CommitAndReport1 | 77 |
| CommitAndReportTrace | 75 |
| CommitAndReportTrace1 | 78 |
| CommitTrace | 74 |
| Delete | 48 |

| | |
|----------------------|----|
| DetachDatabase | 15 |
| Execute | 55 |
| ExecuteNonQuery | 2 |
| ExecuteNonQueryCrypt | 39 |
| ExecuteNonQueryTrace | 73 |
| ExecuteReader | 21 |
| ExecuteReaderCrypt | 28 |
| Get | 33 |
| Get1 | 47 |
| Get2 | 56 |
| GetFileNames | 10 |
| GetInfo | 54 |
| Mark | 23 |
| Prepare | 11 |
| Post | 45 |
| Put | 46 |
| ReaderData | 16 |
| ResetReader | 14 |
| Rollback | 8 |
| TypeInfo | 19 |
| Update | 49 |

The following response bytes are defined (enumeration PyrrhoBase.Responses)

| | |
|------------------------|----|
| Acknowledged | 0 |
| ReaderData | 10 |
| Done | 11 |
| Schema | 13 |
| CellData | 14 |
| NoData | 15 |
| Files | 18 |
| Prepare | 55 |
| Primary | 60 |
| Begin | 62 |
| TransactionReport | 65 |
| Warning | 67 |
| PostReport | 68 |
| Columns | 71 |
| Schema1 | 72 |
| DoneTrace | 76 |
| TransactionReportTrace | 77 |

8.8.2 Sending the connection string

For .NET implementation, the user name is supplied by the operating system (not by the user). Not all fields in the connection string are sent to the server: provider: host and port are already used in establishing the connection to the server, and the server is locale-independent, so locale is not sent either. For the reference for the connection string, see section 6.3.

All traffic in this section is encrypted including the protocol octets. Recall that the encryption algorithms in PyrrhoLink.dll and OSPLink.dll are different. Note that Locale is handled by these DLLs and not sent to the server.

| Connecting Octet | Octet value | Further data | Description |
|------------------|-------------|--------------|---|
| Files | 22 | String | a comma-separated list of databases* |
| Role | 23 | String | the Role for the connection |
| Done | 24 | | signals end of the connection string data |
| Stop | 25 | String | the stop time |
| Key | 27 | String | the DBMS access key |

| | | | |
|--------|----|----------------------|--|
| Modify | 32 | true <i>or</i> false | Allow modification (default true for the first database in the connection) |
|--------|----|----------------------|--|

*The Files keyword is now a misnomer as only one database is permitted per connection.

On successful completion of this phase, non-encrypted communication resumes, and the server responds as follows:

| Response Octet | Octet value | Further data | Description |
|----------------|-------------|--------------|------------------------------------|
| Primary | 60 | | <i>Preserved for compatibility</i> |

8.8.3 Protocol details

Normal traffic consists of client requests and server replies, using formats described in the following subsections (braces { } indicate repetition prefixed by an int count):

| Protocol Octet | Further data | Description | Response Octet† | Further data | Description |
|--------------------|--------------|---|--------------------|--------------|---|
| Authority | String | Session role name | Done | | |
| BeginTransaction | | | | | |
| Check | String | | Valid | | |
| | | | Invalid | | |
| CheckConflict | | | | int | 1 if transaction conflict has occurred |
| CloseConnection | | | | | |
| CloseReader | | | | | |
| Commit | | | Done | | |
| CommitAndReport | {check} | | Transaction Report | int, {check} | Updates the check information |
| Delete | int, sql | Schema key, Delete statement single row | Done | | |
| DetachDatabase | String | Database name | Done | | |
| Execute | {String} | Prepared statement name, actual params | Schema | schema, int | Reader opened, number of records affected |
| | | | Done | int | number of records affected |
| ExecuteNonQuery | String | SQL statement | Done | int | number of records affected |
| ExecuteReader | String | | Schema | schema | |
| | | | Done | | if not a select statement |
| ExecuteReaderCrypt | String | Encrypted SQL | Schema | schema | |
| | | | Done | | if not a select statement |
| Get | String | rurl | Schema | schema | |
| | | | Done | | no data |
| Get1 | long, String | Schema key, rurl | Schema | schema | |
| | | | Done | | no data |
| Get2 | String | rurl | Schema | long, schema | Schema key, schema |
| | | | Done | | No data |
| GetFileNames | | | Files | {string} | Names of databases in folder |
| GetInfo | String | typeName | Columns | {column} | Details for a structured type |
| Mark | | | | | Allows error recovery (uses TRANSACTION_ACTIVE) |

| | | | | | |
|-------------|---------------------|---|------------|---|------------------------|
| Prepare | 2 Strings | name, SQL parametrised statement | Done | | |
| Post | int, sql | Schema key, Insert statement single row | Schema | schema, CellData, check [‡] , {cell}, Done | |
| Put | int, sql | Schema Key, update single row | | | |
| ResetReader | | | Done | | |
| Rollback | | | Done | | |
| ReaderData | | | ReaderData | {cell} | cells* |
| Rest | String, url, String | Verb, url, Jsondata | Done | | |
| TypeInfo | String | Data type name | | string | Type definition in xml |

[†] Octet 0x84 (Warning) may precede any reply, followed by string,{string} for signal and parameters.

[‡] For explicit transactions, update the check info using CommitAndReport

* A single large cell may take more than one physical block. Otherwise the ReaderData call returns the number of cells that will fit into a physical block, which may include data from subsequent rows if any.

** As above, the highwater mark excludes the end-of-file marker if present.

8.8.4 Schema

The Schema reply consists of 0xb if the table is empty. Otheriwes it consists of 0xd, following by the number of columns, the name of the table, and then for each column, the caption and type data as described below (sec 8.9.8).

8.8.5 Column

A Columns reply consists of the number of columns, followed by the caption for the column and a type. The caption is a String. The type information consists of a type name followed by an int constructed as follows:

| Mask | Description |
|-------|--|
| 0x00f | Base Data Type (see below) |
| 0x0f0 | 0 if not a primary key column, otherwise primary key ordinal+1 |
| 0x100 | Not Null |
| 0x200 | Generated Always |
| 0x400 | Reverse order (internal) |

8.8.6 Cell

The number of columns was provided beforehand, so a row consists of CellData for each of the columns.

CellData may be optionally preceded by octet 3 and a row version validator string and/or octet 4 and a readCheck string. Them octet 0 if the column contains null, octet 1 followed by the cell value if the value type matches the column's typecode (followed by the value), octet 2 otherwise (followed by subtype name and value).

| Typecode | Data Type | Value format |
|----------|----------------------------|----------------|
| 0 | null | 0 for null |
| 1 | Integer | String |
| 2 | Numeric | String |
| 3 | String (also used for XML) | String |
| 4 | Timestamp | long : ticks |
| 5 | Blob | { Octet } |
| 6 | NestedRow | { Field } |
| 7 | Array or multiset | ARRAY { Cell } |

| | | |
|----|---------------|-------------------------------|
| 8 | Real | String |
| 9 | Boolean | int |
| 10 | Interval | 3 longs: years, months, ticks |
| 11 | Time | long: ticks |
| 12 | Type or Field | String, Cell |
| 13 | Date | long: ticks |
| 14 | Table | Schema { Cell } |
| 15 | Multiset | MULTISET { Cell } |

8.8.7 Type

Type information is given as an XML string.

8.8.8 Exceptions

These are exception replies during the normal traffic sequence. Since version 2.0, these are reported in a special exception block, as follows. If the count appears to be 2047, the buffer contains an exception record instead, in which the next two octets (octets 2 and 3) contain the count of octets used to transmit the exception details.

| Server Octet | Further data | Description |
|--------------|-------------------------------|----------------------|
| 0xc | String, Strings, StringPairs* | Database Exception |
| 0x11 | String | Transaction Conflict |
| 0x10 | String | Other exception |

* added in version 4.8 for diagnostics information.

9. Pyrrho Database File Format

Close to the start of every database file, Pyrrho records the identity of the creator: it generally does not work to use a copy of a database that someone else has created⁴⁰. An exception is where databases have been created using OSPStudio or PyrrhoStudio, as these are for use as embedded databases, generally in environments without user identities (the user name is “Me”). The length of the owner’s user name is generally different and this will affect internal file positions and schema-keys (see 8.4.1,8.4.9,8.4.16).

The Pyrrho database file begins with a key (777) and version number (e.g. 50) encoded using Pyrrho’s integer format 9.1.1. The rest of the file consists of a sequence of variable length records, whose type is given by the opening byte, and whose contents are variable length. The first two of these records are the database default role (this is always at position 5) and the database user name (as discussed in the previous paragraph). Each record is made up of a set of data fields: some have fixed format, and some have variable format. The last record is an EndOfFile (see 9.2) unless the append storage option has been selected: append storage does not use and EndOfFile marker. This chapter of the booklet describes all of these details.

Apart from the EndOfFile marker, once any data has been written to the file it stays unchanged at the position it was written. Database files larger than 32GB are physically divided into 32GB segments. The data is continued logically from one file to the next without any additional formatting.

9.1 Data Formats

Byte and Unicode are the only predefined formats. It is assumed that all data files are dealt with by the operating system as a sequence of bytes. In particular, Pyrrho has its own way of encoding integers, floats etc, which are described below.

Pyrrho constructs a small set of data types from these, as follows:

| Code | Data Type | Format as |
|------|-----------|---|
| 1 | Time | 1 Integer (UTC ticks) |
| 2 | Interval | 3 Integers (year,month, ticks) |
| 3 | Integer | 1 byte (bytelength), bytelength bytes: see 9.1.1 |
| 4 | Numeric | 2 Integers (mantissa, scale: see 9.1.2) |
| 5 | String | 1 Integer (bytelength), bytelength UTF-8 bytes |
| 6 | Date | 1 Integer (UTC ticks) |
| 7 | TimeStamp | 1 Integer (UTC ticks) |
| 8 | Boolean | 1 byte: T=1,F=0 |
| 9 | DomainRef | Structured: 2 Integers (typedefpos,els), els variants: see 9.1.3 Otherwise: 1 Integer (domaindefpos) |
| 10 | Blob | 1 Integer (bytelength), bytelength bytes |
| 11 | Row | 2 Integers (typedefpos,cols), cols pairs(coldefpos,variant: see 9.1.3) |
| 12 | Multiset | 2 Integers (typedefpos,els), els variants: see 9.1.3 |
| 13 | Array | 2 Integers (typedefpos,els), els variants: see 9.1.3 |
| 14 | Password | A more secure type of string (write-only) |

9.1.1 Integer format

Zero is encoded as 0 bytes. An integer that fits in a signed byte is encoded as 1 byte (i.e. -127.. 127). Otherwise integers are encoded in unsigned bytes (radix 256), using as many as are required to ensure the first byte has a sign bit (0x80) if and only if the integer is negative.

Unless otherwise specified, unbounded precision is used for integer arithmetic. A string representation is used if required to return a very large integer value to the client.

⁴⁰ The database owner is initially the same as the creator, but this can be set later using GRANT OWNER TO. Such a change will be recorded at the current end of the database file. There is no way of changing any record in the log, so the creator’s name cannot be changed.

9.1.2 Numeric and Real format

Numeric format has one Integer for the mantissa, and 1 for the scale. If these are m and s respectively, then the value of the decimal is $m \cdot 10^{-s}$. This format is used for both numeric/decimal and real quantities.

Unless constrained by precision specifications, addition and multiplication of numeric quantities uses 2040-bit precision, while division uses a default precision of 13 decimal digits. If greater precision is required for division, it can be specified. It should be obvious that there are resource implications to using very large precision values.

9.1.3 Variant format

This consists of

- a 1-byte code for the data type (the code in the above table 9.1),
- if this byte is 9 (DomainRef), the defining position of the type
- data in the corresponding format.

9.1.4 Array and Multiset format

Two Integers (9.1.1), namely the defining position of the element type, the number of elements n , followed by n items in the specified format.

9.1.5 Row and User Defined Type format

Two Integers (9.1.1), namely the defining position of the row type, the number of non-null fields n , then for each, an Integer (9.1.1) for the defining position of the field (a column), and an element of that type.

9.1.6 Blob format

An Integer (9.1.1), namely the number of bytes n , followed by n bytes.

9.1.7 Boolean format

1 byte (1 for true, 0 for false).

9.1.8 Char and XML format

An Integer (9.1.1), namely the number of bytes n of actual data, followed by n bytes in UTF8 encoding. (The fieldsize is not used).

9.1.9 Date and TimeSpan formats

An Integer (9.1.1) namely the number of ticks in the date or timespan.

9.1.10 Interval format

Three Integers (9.1.1), namely years, months, and ticks.

9.2 Record formats

The record formats are as follows (note that many are now deprecated for all new transaction data as indicated below)::

| Code | Record type | Format as 1 byte for Code and then |
|------|-----------------|---|
| | Physical | 1 integer (transaction id) |
| 0 | EndOfFile | 4 bytes (validation). Not used with append storage. |
| 1 | Table | 1 string (name), Physical |
| 2 | Role | 2 strings (name, details), Physical |
| 3 | Column | 1 integer (table id), 1 string (name), 2 integer (position, domain id), Physical. <i>Deprecated – see Column3</i> |
| 4 | Record (Insert) | 1 integer (table id), Fields (see 9.2.2), Physical |
| 5 | Update | 2 integers (replaced record id, other fields: see 9.2.3), Record |
| 6 | Change | 1 integer (object id), Table |

| | | |
|----|--------------------------|--|
| 7 | <i>Alter</i> | <i>1 integer (prev), Column.Deprecated – see Alter3</i> |
| 8 | Drop | 1 integer (object id), Physical |
| 9 | <i>Checkpoint</i> | <i>(no data), Physical</i> |
| 10 | Delete | 1 integer (record id), Physical <i>Note: deprecated: use Delete1 instead</i> |
| 11 | Edit | 1 integer (replaced domain id), Domain |
| 12 | Index | 1 string (name), 2 integers (table id, ncols), ncols integers (\pm column id), 2 integers (flags, reference, see 9.2.5), Physical. Negative column id indicates reverse ordering |
| 13 | Modify | 1 integer (replaced id), 2 strings (name, body), Physical |
| 14 | Domain | 1 string (name), 3 integers (dataType: see 9.2.1, dataLength, scale), 3 strings (charset, collate, default), 1 integer (element domain or table id), Physical |
| 15 | Check | 1 integer (object id), 2 string (name, check source), Physical |
| 16 | Procedure | 1 string, 1 integer, 1 string (name, arity, proc source), Physical - <i>deprecated: see Procedure2</i> |
| 17 | Trigger | 1 string (name), 3 integers (table id, trigger type, position, see 9.2.8), 1 string (definition), Physical |
| 18 | View | 2 strings (name, view source), Physical |
| 19 | User | 1 string (name), Physical |
| 20 | Transaction | 4 integers (nrecs, role id, user id, time) |
| 21 | Grant | 3 integers (privilege, see 9.2.7, object id, grantee id), Physical |
| 22 | Revoke | Grant |
| 23 | Role1 | 1 string (name), Physical. <i>Deprecated – use Role instead</i> |
| 24 | Column2 | 1 string (default), 1 boolean (notNull), 1 GenerationRule, Column <i>Deprecated</i> |
| 25 | Type | 1 integer (under type id), Domain |
| 26 | Method | 2 integers (type id, method type: see 9.2.6), Procedure - <i>deprecated: see Method2</i> |
| 27 | <i>Transaction2</i> | <i>Participants, Transaction Not supported</i> |
| 28 | Ordering | 3 integers (type def, func def, flags: see 9.2.9), Physical |
| 29 | (not used) | Used internally for an Update variant |
| 30 | DateType | 2 integers (start field, end field, see 9.2.10). Domain (dataLength and scale are for seconds precision), Physical |
| 31 | <i>TemporalView</i> | <i>no longer supported</i> |
| 32 | <i>ImportTransaction</i> | <i>1 string (provenance), Transaction</i> |
| 33 | Record1 | 1 string (provenance), Record – <i>deprecated, see Record2</i> |
| 34 | Type1 | 1 string (with uri), Type |
| 35 | Procedure2 | 1 string, 2 integers, 1 string (name, arity, ret type id, proc source), Physical |
| 36 | Method2 | 2 integers (type id, method type: see 9.2.6), Procedure2 |
| 37 | Index1 | 1 integer (adapter), Index |
| 38 | Reference | 2 integers (index def pos, referrer pos), Fields (see 9.2.2), Physical: only used when a coercion or adapter function creates a reference. The Fields give the computed foreign key. |
| 39 | Record2 | 1 integer (subtype), Record |
| 40 | Curated | Physical. Makes subsequent log entries PUBLIC |
| 41 | <i>Partitioned</i> | <i>No longer supported, see Partition below</i> |
| 42 | <i>Domain1</i> | <i>2 strings(typeiri,abbrev),Domain</i> |
| 43 | <i>Namespace</i> | <i>2 strings(prefix,iri)</i> |
| 44 | Table1 | 1 string(rowiri), Table |
| 45 | <i>Alter2</i> | <i>1 long (prev), Column2</i> |
| 46 | <i>AlterRowIri</i> | <i>1 long (prev), Table1</i> |
| 47 | Column3 | 1 strings (update), 3 ints (reftable, refindex, refindex2), Column2 <i>reftable, refindex and refindex2 are not used</i> |
| 48 | <i>Alter3</i> | <i>1 long (prev), Column3</i> |
| 49 | <i>View1</i> | <i>no longer supported</i> |

| | | |
|----|------------------|--|
| 50 | Metadata | 3 strings (name, details, iri), 3 ints (seq, objid, flags – see 9.2.11), Physical. Seq is nonzero only for view and function columns. Role specific. |
| 51 | PeriodDef | 1 integer (table), 1 string (periodname), 2 integers (start, end) |
| 52 | Versioning | 1 integer (period) only for system versioning |
| 53 | Check2 | 1 integer (column defpos), Check |
| 54 | Partition | 2 integers (base database curpos, schema length), {Octet} (schema info from base database), Physical. |
| 55 | Reference1 | 1 string (referrer partition), Reference: Reference1's are constructed for cross-partition foreign keys, so that the referrer pos is in the named partition. The index may be zero; in which case the entry gives details of a remote User record (see 8.3.4). |
| 56 | ColumnPath | 1 integer (column defpos), 1 string (the path, starting with .), 1 integer (the domain definition). |
| 57 | Metadata2 | 1 int (maxDocuments) 1 long (storageSize), Metadata not used |
| 58 | Index2 | 1 integer (flags), Index1 not used |
| 59 | DeleteReference1 | Reference1 |
| 60 | Authenticate | 1 string (password), 1 int (defrole), User |
| 61 | RestView | 1 integer (struct), View. The URL is provided in metadata as the desc field. |
| 62 | TriggeredAction | 1 integer (trigger defpos) introducing an embedded set of changes |
| 63 | RestView1 | <i>Name, password, RestView deprecated: provide any credentials in URL</i> |
| 64 | Metadata3 | refpos, Metadata2 |
| 65 | RestView2 | usingtablepos, RestView |
| 66 | Audit | 3 integers (user, table, ticks) {integer}{string} (cols, keys), Physical |
| 67 | Clearance | 1 integers (user), Label (clearance, see 9.2.13), Physical |
| 68 | Classify | 2 integers (object), Label (classification, see 9.2.13), Physical |
| 69 | Enforcement | 2 integers (table, flags see 9.2.7 Privilege below), Physical |
| 70 | Record3 | Label (classification, see 9.2.13), Record2 |
| 71 | Update1 | Label (classification, see 9.2.13), Update |
| 72 | Delete1 | 1 integer (table), Delete |
| 73 | Drop1 | 1 integer (dropAction), Drop |
| 74 | RefAction | 2 integers (defpos, flags) Physical |
| 75 | Audit2 | 1 string (user), Audit |

9.2.1 DataType

| Code | DataType |
|----------------|-----------|
| 11 | ARRAY |
| 26 | BLOB |
| 27 | BOOLEAN |
| 37 | CHAR |
| 40 | CLOB |
| 65 | CURSOR |
| 67 | DATE |
| 135 | INTEGER |
| 152 (was 137) | INTERVAL |
| 168 | MULTISET |
| 171 | CHAR |
| 172 | CLOB |
| 177 | NULL |
| 179 | NUMERIC |
| 203 (also 199) | REAL |
| 218 | PASSWORD |
| 257 | TIME |
| 258 | TIMESTAMP |
| 267 | TYPE |
| 356 | XML |

These codes are used only in the PDomain record. The numbers 137 and 199 are supported as an attempt at backward compatibility.

9.2.2 Drop Action

| Code | Drop Action |
|------|--------------------|
| 3 | Cascade |
| 2 | Default |
| 1 | Null |
| 0 | Restrict (default) |

9.2.3 Fields information

The sequence of fields defining a record is formatted as 1 integer (nfields), nfields x (1 integer (column id), 1 variant (value)) see 9.1.3. Fields not defined by a record are not supplied.

9.2.4 Update information

The Update record contains in the base class (Record) part the fields that are updated. The other fields integer identifies the most recent previous Record or Update record with field information that remains current. The replaced record id is the original record that subsequent updates have altered.

9.2.5 Index flags

The reference field is the id of a reference index.

| Flag | Meaning |
|------|--|
| 0 | NoType |
| 1 | Primary Key |
| 2 | Foreign Key |
| 4 | Unique |
| 8 | Descending (all key columns) <i>Deprecated</i> |
| 16 | Restrict Update |
| 32 | Cascade Update |
| 64 | Set Default Update |
| 128 | Set Null Update <i>Deprecated</i> |
| 256 | Restrict Delete |
| 512 | Cascade Delete |
| 1024 | Set Default Delete |
| 2048 | Set Null Delete |
| 4096 | TemporalKey <i>Deprecated</i> |

Not all flags are permitted or required: Restrict is a default, and Set Null is not permitted.

9.2.6 Method type

| Value | Meaning |
|-------|-------------|
| 0 | Instance |
| 1 | Overriding |
| 2 | Static |
| 3 | Constructor |

9.2.7 Privilege flags

| Flag | Meaning | Flag | Meaning |
|------|------------|--------|-----------------------------|
| 0x1 | Select | 0x400 | Grant Option for Select |
| 0x2 | Insert | 0x800 | Grant Option for Insert |
| 0x4 | Delete | 0x1000 | Grant Option for Delete |
| 0x8 | Update | 0x2000 | Grant Option for Update |
| 0x10 | References | 0x4000 | Grant Option for References |
| 0x20 | Execute | 0x8000 | Grant Option for Execute |

| | | | |
|-------|---------|---------|--------------------------|
| 0x40 | Owner | 0x10000 | Grant Option for Owner |
| 0x80 | Role | 0x20000 | Admin Option for Role |
| 0x100 | Usage | 0x40000 | Grant Option for Usage |
| 0x200 | Handler | 0x80000 | Grant Option for Handler |

9.2.8 Trigger type

| Flag | Meaning |
|------|----------------|
| 1 | Insert |
| 2 | Update |
| 4 | Delete |
| 8 | Before |
| 16 | After |
| 32 | Each row |
| 64 | Instead |
| 128 | Each statement |
| 256 | Deferred |

9.2.9 Ordering type

| Flag | Meaning |
|------|----------|
| 0 | None |
| 1 | Equals |
| 2 | Full |
| 4 | Relative |
| 8 | Map |
| 16 | State |

9.2.10 Interval fields

| Flag | Meaning |
|------|---------|
| 0 | SECOND |
| 1 | MINUTE |
| 2 | HOUR |
| 3 | DAY |
| 4 | MONTH |
| 5 | YEAR |

9.2.11 Metadata flags

| Flag | Meaning |
|-------|------------------|
| 0 | Unspecified |
| 1 | ENTITY |
| 2 | ATTRIBUTE |
| 4 | PIE |
| 8 | SERIES |
| 16 | POINTS |
| 32 | X |
| 64 | Y |
| 128 | HISTOGRAM |
| 256 | LINE |
| 512 | CAPTION |
| 1024 | CAPPED |
| 2048 | USEPOWEROF2SIZES |
| 4096 | BACKGROUND |
| 8192 | DROPDUPS |
| 16384 | SPARSE |
| 32768 | LEGEND |

In XML output, column values that are attributes appear as attributes of the row element rather than child elements. In HTML output from a table, a chart is generated if the table is a pie, series, or points, one column has x and at least one column has y, histogram or line. Some of the later entries here are for MongoDB.

9.2.12 GenerationRule

| Flag | Meaning |
|------|-------------------------|
| 0 | No |
| 1 | Generated AS expression |
| 2 | Generated AS ROW START |
| 3 | Generated as ROW NEXT |
| 4 | Generated AS ROW END |

9.2.13 Mandatory Access Control Label

There are two formats depending on whether the label is in the cache. The record begins with an Integer flag, and determines the format of what follows.

| Flag | Rest of Record |
|------|---|
| 0 | 1 Integer (defining position of the Label in the transaction log) |
| 1 | 2 Integers (minLevel, maxLevel) {id} (groups) {id} (references) |

10. Troubleshooting

This section reviews a number of circumstances in which a database can become unusable. The safeguards that cause a database to be marked unusable are there to protect business operations as far as practicable against hardware errors or malicious activity.

Databases should not become unusable during normal operation. Any performance issue of this sort should be notified immediately to malcolm@pyrrhodb.com, so that this issue can be resolved.

Suggested additions to this section will be very welcome. The following checklist is intended for use where a correctly installed Pyrrho installation ceases to work.

| Symptom | Possible causes | Section |
|--|--|---------|
| Application crashes or malfunctions | The PyrrhoLink.dll it uses needs to be updated to match the PyrrhoSvr | 10.7 |
| A database will not load | The database file may have been removed, renamed, or damaged | 10.1-3 |
| An application reports an invalid schema key | A user has updated the database schema and the Role\$Class, Role\$Java or Role\$Python system table should be used to regenerate the database class. | 10.5 |
| A user can no longer access or modify data | The user may be accessing the data from another user's account, or from an environment that reports the user name differently | 10.4 |
| | The user's (or role's) permissions have been modified | 5.5 |

10.1 Destruction and restoration

It is fundamental to database design that transactions are durable once committed, with results that can only be changed by subsequent transactions. There are some interventions at the operating system level that violate this principle, which are possible even with Pyrrho.

- Destruction of the entire database through deletion of the database file, formatting or disposing of the storage media etc.
- Restoration of a database from a backup copy

These actions will result in some or all work recorded in the database to be lost. Restoration from backup can restore transactions up to the time of the backup, but transactions committed after the last backup will be permanently lost.

There are other interventions that can make the database temporarily inaccessible: such as stopping the server, or altering access permissions on the file or the network. These are not regarded as changing the durability of the transaction. The notes in this section assume that such matters can be resolved in the usual ways, such as restoring the accessibility of the database file, restoring network connectivity, etc.

Some hardware failures can cause a single transaction being committed at the time of the failure to be lost (section 10.2).

10.2 Hardware failure during commit

If a hardware failure occurs during the commit phase of a transaction, the client or application will be told that the connection has been broken but may not know whether the transaction commit was completed before communication with the server was broken.

When the database is reloaded, it is very likely that either (a) the transaction will have been forgotten (rolled back) or (b) the transaction will be found in its entirety. If a part of the transaction data was actually written to physical media, then recovery is required.

10.3 Alternative names for a database file

The database name can be the pathname of the file. Databases can be renamed in this version of Pyrrho, provided the connection strings in all applications are modified to reflect the change.

10.4 User identity and database migration

It is deliberately made difficult in Pyrrho for a user to pretend to be someone else: the user's name is supplied by the operating system. If a database file is installed in a new context, or a user's identity is changed, it may be difficult for an application to have the correct user identity for contacting the database.

Unless the database has withdrawn privileges from the system role, the server account can be used to access the database.

If any user identity in the database is still available, and has suitable admin privileges, it can be used to grant permissions to the new user identities.

Otherwise, use investigation of the log files to find out the user identities configured in the database, and temporarily install a user identity that is recognised by the database (preferably that of the database owner) and grant the permissions that the new user identities require.

10.5 API Dependency on database history

Section 6.4 discussed the API for object-oriented access to the database. It is important to remember that the class definitions (for C#, Java, or Python) used by this API must match the database schema. Each class and structured type has a schema key and this must match the position in the database file of the last schema change affecting the class or type.

Following such a change (or reconstruction of the database by another user) the affected schema keys must be updated in the application program.

11. End User License Agreement

You may use and redistribute the client libraries (PyrrhoLink.dll and/or PyrrhoJC.jar) in any product. You may copy and distribute this booklet in its entirety.

You are hereby granted a non-transferable, royalty-free license to use the software described in this manual in accordance with its provisions, and to view and test the source code, including modifications or incorporation in other software. Under no circumstances will Malcolm Crowe or the University of the West of Scotland be liable for any loss or damage however caused.

This software is and remains intellectual property of the University of the West of Scotland, protected by copyright. You are permitted to redistribute and include any of the code in any product, provided its ownership and copyright status is suitably acknowledged.

References

- Ceri S.; Pelagatti, G. (1984): Distributed Database Design: Principles and Systems (McGraw-Hill)
- Codd, E. F. (1985) Does your DBMS run by the rules? *ComputerWorld* Oct 21, 1985.
- Crowe, M. K. (2007): An introduction to the source code of the Pyrrho DBMS. *Computing and Information Systems Technical Reports*, **40**, University of Paisley.
- Crowe, M., Begg, C., Laux, F., Laiho, M (2017): Data Validation for Big Live Data, *DBKDA 2017, The Ninth International Conference on Advances in Databases, Knowledge and Data Application*, Barcelona, Spain, May 21-26 2017. ISBN 978-1-61208-558-6, p. 30-36.
- Floridi, Luciano: Sextus Empiricus: The transmission and recovery of Pyrrhonism (American Philological Association, 2002) ISBN 0195146719
- Laertius, Diogenes (3rd cent): The Lives and Opinions of Eminent Philosophers, trans. C. D. Yonge (London 1895)
- Laiho, M., Laux, F. (2010): Implementing Optimistic Concurrency Control for Persistence Middleware Using Row Version Verification, *Advances in Databases Knowledge and Data Applications (DBKDA)*, 2010 Second International Conference on, IEEE, ISBN 978-1-4244-6081-6 p. 45-50, DOI: 10.1109/DBKDA.2010.25.
- SQL2016: ISO/IEC 9075-2:2016 Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation); ISO/IEC 9075-4:2016: Information Technology – Database Languages – SQL – Part 4: Persistent Stored Modules; (International Standards Organisation, 2016)
- SWI-Prolog: www.swi-prolog.org

Index to Syntax

| | | | |
|-----------------------|---------|-----------------------------|--------|
| AbsoluteValue | 72 | Column2..... | 118 |
| Action | 64 | ColumnConstraint | 62 |
| Adapter function..... | 28 | ColumnConstraintDef | 62 |
| Alias | 67 | ColumnDefinition | 62 |
| Alter..... | 59 | ColumnOption | 63 |
| AlterCheck..... | 60 | ColumnOptionsPart..... | 63 |
| AlterColumn | 61 | ColumnRef..... | 69 |
| AlterDomain | 60 | COMMAND_FUNCTION | 74 |
| AlterField..... | 61 | COMMAND_FUNCTION_CODE | 74 |
| AlterTable..... | 60 | COMMIT | 58 |
| AlterType..... | 61 | Comparison..... | 71 |
| AlterView | 61 | CompOp..... | 71 |
| Any | 71 | CompoundStatement..... | 17, 74 |
| ASC | 68 | CondInfo | 74 |
| Assignment | 73 | CondInfo' | 74 |
| ATTRIBUTE..... | 61 | Condition | 74 |
| AttributeSpec..... | 75 | CONDITION_NUMBER | 74 |
| Authority | 40 | ConditionCode | 74 |
| Avg | 72 | ConditionList | 74 |
| BEGIN..... | 58, 74 | CONNECTION_NAME..... | 74 |
| Between | 71 | CONSTRAINT | 60 |
| BETWEEN..... | 71 | CONSTRAINT_CATALOG | 74 |
| BinaryOp | 69 | CONSTRAINT_NAME | 74 |
| BLOB | 66 | CONSTRAINT_SCHEMA..... | 74 |
| BooleanExpr..... | 71 | CONSTRUCTOR | 60 |
| BooleanFactor | 71 | CONTENT | 72 |
| BooleanTerm | 71 | Count..... | 72 |
| BooleanTest..... | 71 | Create | 61 |
| BREAK..... | 58 | CROSS | 67 |
| Bson..... | 65 | CSV..... | 61 |
| Call | 74 | CURRENT | 73 |
| CAPTION..... | 61 | CURSOR_NAME..... | 74 |
| Cardinality | 73 | CursorSpecification..... | 67 |
| CASCADE | 64 | DatabaseError | 103 |
| Cascade Delete | 90, 120 | DataReader..... | 42 |
| Cascade Update | 90, 120 | Date..... | 104 |
| CaseStatement | 74 | DateTimeField | 70 |
| Cast..... | 73 | DateTimeFunction | 73 |
| CATALOG_NAME | 74 | DateTimeType | 66 |
| Ceiling | 72 | DBNull..... | 42 |
| CHAR_LENGTH..... | 72 | Declaration..... | 74 |
| CharacterType | 65 | DEFAULT | 60 |
| CheckConstraint | 60 | DEFERRED..... | 64 |
| CLASS_ORIGIN..... | 74 | DefinedType | 66 |
| classification | 67 | DeletePositioned | 67 |
| Classification | 60, 61 | DeleteSearched | 67 |
| clearance..... | 67 | DESC | 68 |
| Clearance | 64, 89 | DISTINCT | 67 |
| CLOB | 66 | DOCARRAY..... | 65 |
| Close..... | 74 | DOCUMENT | 65, 72 |
| Coalesce..... | 72 | DomainDefinition | 62 |
| Collate | 66 | DropAction | 64 |
| Collect | 73 | DropObject..... | 64 |
| Cols..... | 63 | DropStatement | 64 |
| COLUMN_NAME | 74 | | |

| | | | |
|-----------------------------|-----|----------------------------|--------|
| DYNAMIC_FUNCTION..... | 74 | INTERSECT | 69 |
| DYNAMIC_FUNCTION_CODE | 74 | INTERVAL | 69 |
| Element..... | 73 | IntervalField..... | 66, 70 |
| EndField | 70 | IntervalQualifier..... | 70 |
| EndTimestamp..... | 102 | IntervalType..... | 66 |
| EndTransaction..... | 102 | INVERTS..... | 61 |
| Enforcement | 62 | ItemName..... | 74 |
| ENTITY..... | 61 | ITERATE..... | 59 |
| ETag | 28 | JoinedTable | 67 |
| Event..... | 63 | JoinType..... | 67 |
| Every | 71 | JSON..... | 61 |
| EXCEPT..... | 67 | Label | 75 |
| Exclusion | 71 | LAST | 68 |
| Execute | 107 | LEAVE | 59 |
| Exists | 71 | LEFT..... | 67 |
| Exponential..... | 72 | LEGEND | 61 |
| Extract | 72 | LengthExpression | 72 |
| ExtractField | 72 | Level | 60 |
| Fetch | 74 | LEVEL..... | 60 |
| FetchFirstClause | 68 | Like | 72 |
| Field..... | 61 | LINE | 61 |
| FieldCount | 41 | Literal..... | 69 |
| FIRST | 68 | LobType..... | 66 |
| FloatType..... | 66 | Locale | 40 |
| Floor | 72 | LOCALTIME..... | 73 |
| Foreign Position..... | 120 | LoopStatement | 75 |
| ForStatement | 75 | Maximum..... | 72 |
| FromClause | 70 | Member | 72 |
| FULL..... | 67 | MESSAGE_LENGTH..... | 74 |
| FuncOpt..... | 71 | MESSAGE_OCTET_LENGTH | 74 |
| FunctionCall | 72 | MESSAGE_TEXT..... | 74 |
| Fusion | 73 | Metadata..... | 61 |
| GenerationRule..... | 62 | Method | 59 |
| GetDiagnostics | 74 | MethodCall | 75 |
| GetFieldType..... | 41 | MethodType..... | 60 |
| GetFileNames | 107 | Minimum | 72 |
| GetName..... | 41 | Modulus | 72 |
| Grant..... | 64 | MongoDB | 65 |
| Grantee | 65 | MORE..... | 74 |
| GranteeList | 65 | MULTISET | 66, 68 |
| GroupByClause | 70 | MultisetOp | 69 |
| Grouping..... | 72 | NamedValue | 75 |
| GroupingSet..... | 70 | Namespace | 75 |
| GroupingSpec | 70 | NATURAL..... | 67 |
| HandlerType..... | 74 | NaturalLogarithm..... | 73 |
| HavingClause | 70 | NCLOB..... | 66 |
| HISTOGRAM | 61 | Normalize..... | 73 |
| Host | 39 | Null | 72 |
| How | 74 | Nullif..... | 73 |
| HTTP GET | 70 | NULLS | 68 |
| IDataReader..... | 41 | NUMBER | 74 |
| IfStatement | 75 | NumericType | 66 |
| In | 71 | NumericValueFunction..... | 72 |
| INNER..... | 67 | ObjectName | 65 |
| Insert..... | 66 | ObjectPrivileges | 64 |
| INSTANCE | 60 | OCTET_LENGTH..... | 72 |
| IntegerType | 66 | Of | 72 |
| Intersect | 73 | | |

| | | | |
|----------------------------------|---------------|--------------------------------|------------|
| Open | 75 | SCHEMA_NAME | 74 |
| OrderByClause | 67 | SearchCondition | 67 |
| Ordering | 62 | SECURITY | 28, 60, 69 |
| OrderSpec | 68 | SelectItem | 67 |
| OrdinaryGroup | 70 | SelectSingle | 75 |
| OUTER | 67 | SERVER_NAME | 74 |
| OVERRIDING | 60 | Set | 73 |
| OWNER | 65 | Set Default Delete | 90, 120 |
| Parameter | 60 | Set Default Update | 90, 120 |
| PARAMETER_MODE | 74 | Set Null Delete | 90, 120 |
| PARAMETER_NAME | 74 | Set Null Update | 90, 120 |
| PARAMETER_ORDINAL_POSITION | 74 | SetAuthority | 107 |
| Parameters | 60 | SetFunction | 73 |
| PartitionClause | 70 | Signal | 74 |
| PASSWORD | 65 | SIGNAL | 74 |
| PeriodName | 63 | SimpleTable | 67 |
| PIE | 61 | Some | 72 |
| POINTS | 61 | SPECIFIC_NAME | 74 |
| Port | 40 | Sql | 58 |
| Position | 73 | SqlStatement | 58 |
| PowerFunction | 73 | SquareRoot | 73 |
| Predicate | 71 | StandardType | 65 |
| Prepare | 107 | StartField | 70 |
| Privileges | 64 | StartTimestamp | 101 |
| Provenance | 35 | StartTransaction | 101 |
| Provider | 40 | Statement | 58 |
| PyrrhoArray | 103, 105, 106 | Statements | 75 |
| PyrrhoConnect | 103 | STATIC | 60 |
| PyrrhoInterval | 103, 108 | StringValueFunction | 73 |
| PyrrhoRow | 103, 108 | SUBCLASS_ORIGIN | 74 |
| QueryExpression | 67 | Subquery | 67 |
| QueryPrimary | 67 | Substring | 73 |
| QuerySpecification | 67 | Sum | 73 |
| QueryTerm | 67 | System.Type | 41 |
| REAL | 66 | TABLE_NAME | 74 |
| REF | 65 | TableClause | 62 |
| ReferentialAction | 63 | TableConstraint | 63 |
| RefObj | 63 | TableConstraintDef | 63 |
| Rename | 64 | TableContents | 62 |
| Repeat | 75 | TableExpression | 70 |
| Representation | 62 | TableFactor | 71 |
| ResetReader | 107 | TablePeriodDefinition | 63 |
| RESTRICT | 64 | TableReference | 71 |
| Restrict Delete | 90, 120 | TakeOwnership | 14 |
| Restrict Update | 90, 120 | Target | 74 |
| RETURN | 59 | TargetList | 75 |
| RETURNED_SQLSTATE | 74 | TicksPerSecond | 108 |
| Revoke | 64 | TimePeriodSpecification | 67 |
| RIGHT | 67 | TIMESTAMP | 69 |
| ROLLBACK | 58, 59 | TRANSACTION_ACTIVE | 74 |
| Routine | 65 | Transaction2 | 118 |
| ROUTINE_CATALOG | 74 | TransactionConflict | 104 |
| ROUTINE_NAME | 74 | TRANSACTIONS_COMMITTED | 74 |
| ROUTINE_SCHEMA | 74 | TRANSACTIONS_ROLLED_BACK | 74 |
| ROW_COUNT | 74 | TREAT | 68, 73 |
| RowNumber | 73 | Trigger | 64 |
| Scalar | 68 | TRIGGER_CATALOG | 74 |
| | | TRIGGER_NAME | 74 |

| | | | |
|------------------------------|--------|-----------------------|----|
| TRIGGER_SCHEMA | 74 | WindowDetails | 70 |
| TriggerCond | 64 | WindowFrame | 70 |
| TriggerDefinition..... | 63 | WindowSpec | 73 |
| Type..... | 65 | WindowStart | 70 |
| TypedTableElement | 63 | WithinGroup | 73 |
| UNBOUNDED..... | 70 | X | 61 |
| UNICODE..... | 66 | xml | 58 |
| UNION..... | 67, 69 | XML..... | 65 |
| Unique | 72 | XmlAgg | 73 |
| UNNEST..... | 71 | XmlColumn | 68 |
| UpdatePositioned..... | 67 | XmlColumns | 68 |
| UpdateSearched | 67 | XMLComment..... | 75 |
| uri | 58 | XMLConcatenation..... | 75 |
| UserFunctionCall..... | 75 | XMLDocument | 75 |
| VALID..... | 72 | XMLElement | 75 |
| Value | 68 | XMLEXISTS | 71 |
| Values | 63 | XMLForest..... | 75 |
| VALUES | 70 | XMLFunction | 75 |
| VariableRef | 69 | xmlname..... | 58 |
| Versioned..... | 107 | XMLNAMESPACES..... | 60 |
| ViewDefinition | 16, 63 | XMLNDec | 60 |
| ViewSepecification..... | 63 | XMLOption | 60 |
| WhereClause | 70 | XMLParse | 75 |
| While | 75 | XMLProc | 75 |
| <i>window function</i> | 71 | XMLQuery..... | 75 |
| WindowBetween | 71 | XMLTABLE | 71 |
| WindowBound..... | 71 | XMLText | 75 |
| WindowClause | 70 | XMLValidate | 76 |
| WindowDef | 70 | Y | 61 |