Kognitio 8.2 Release Notes

2017/06/16

Table of Contents

Introduction

This document outlines the changes made in the version 8.2 release of Kognitio when compared with the previous standalone release, which was version 8.1.0. Since then we have done a Hadoop only release called 8.1.50 which contained many of the changes below. 8.2 follows on from 8.1.50 and all 8.1.50 features are also present in 8.2. This document will note which features are in 8.1.50 and which are brand new in 8.2.

Upgrade instructions

This version can only be upgraded to from 8.1.0 or 8.1.50 versions.

If you are upgrading from a version earlier than 8.1, you will need to read the release notes for that release, and take the steps indicated to upgrade to 8.1.

Changes in Behaviour

Behaviour changes since 8.1.50:

- Tables and views can no longer be created in the sys schema, which is now intended to be used only for system tables, related objects and small, frequently referenced user tables like process control, status or summary tables. The sys_objects parameter can be set to 1 or 2 to allow the creation of objects in the sys schema. See the Sys Ramstores section below.

Behaviour changes since 8.1.0:

- Previously Kognitio used to allow queries of the following format but this is no longer the case as it is an SQL standard variation: (WITH X AS …)
- The default percentage of RAM store memory for PMAs (i.e. user table and view images) has increased from 70% to 85%, due to improvements in memory handling.
- Security classes used to bypass the check on number of login failures if IPE_ALLLOGIN was not in RAM. Now they give an error for connection attempts for users in such security classes if IPE_ALLLOGIN is not in RAM. So if using this sort of security class, ensure IPE_ALLLOGIN is put into RAM at some point during system restarts (e.g. in an imaging script).
- Privilege changes
    a. GRANT ALL PRIVILEGES ON TABLE now grants every privilege rather than the subset of privileges that happened to be in SQL92. Setting the ci_legacyprivs parameter keeps the old behaviour – those same privileges can also be granted with GRANT BASIC PRIVILEGES ON TABLE.
    b. GRANT ... ON EVERY [TABLE|SCHEMA|USER|...] IN [SYSTEM|SCHEMA ...] TO ... is improved syntax for granting aggregate privileges.
- Spider joins are now turned off by default – they can be enabled by setting ci_nospider to 0. These have been replaced with improved compile-time optimisation which does the job better.
- Larger SYS and LOG slabs by default when recommissioning – new default is 1% of disk resources.
- Faster rollback on startup – rollback of incomplete transactions can complete whilst users are allowed onto the system; locking prevents users accessing objects which are being rolled back, but allows querying on unrelated objects to take place much sooner.
- View images are now created according to dependencies during a create image, making the old ci_allow_revviid parameter obsolete.
- The old periodic trigger mechanism is now deprecated – this functionality became irrelevant many years ago.
- HDFS connector defaults to 64-bit now. So any existing systems which rely on the default being 32-bit will need to specifically set the bitness attribute in the target string of the connector (this can be done prior to upgrading).
- Autocommit is now a server-side operation if you use an up-to-date ODBC driver. Select queries run with server-side autocommit will run using 'AT Now' by default, meaning they will be point in time queries that don't require locks on user-level objects. This allows tables to be queried while they are being modified.
- JDBC bridge changes require client update as well as bridge, and driver class name has changed: it is now com.kognitio.jdbc4.Driver. Also, three jar files are now provided in the distribution: Kognitio_Java6.jar, Kognitio_Java7.jar and Kognitio_Java8.jar, for the various versions of Java. Run "java -version" to determine the version of your JVM.

New Features

New features since 8.1.50:

- The sys schema is now serviced by a separate, dedicated set of ramstores known as 'sys ramstores'. Images of objects in the sys schema will be placed in the sys ramstores by default. This feature provides improved compile times and faster management queries when the server is under heavy load. See the 'Sys Ramstores' section below for more detail.
- Instead of running every query plan step on every ramstore, Kognitio's new asymmetric query processing feature will size each step based on concurrency and the estimated work required by the step and will assign each step to an appropriate number of ramstores. This greatly increases throughput when the server is running a very large number of relatively quick queries. See the Asymmetric Query Processing section below for details.
- The Location Hash Value (LHV) feature from previous releases has been extended to work with multiple LHV values per access step and the compiler has been tuned to properly use this feature when gathering statistics for estimates. This allows the LHV feature to work in cases where it previously didn't, including queries which do joins and queries where the LHV values are obtained from values that come out of another table at runtime. See the Location Hash Values section below for details.

New features since 8.1.0:

- Kognitio's software is now able to run within Hadoop as a YARN application as well as a standalone. This requires a different deliverable, the kodoop.tar.gz package. For more information about running on Hadoop see http://www.kognitio.com/on-hadoop
- Centralised Hadoop and Java configuration. This allows Hadoop and Java specific configuration to be defined in one place in the server's config files rather than repeatedly in every plugin. See the 'Hadoop and Java Configuration section below for details.
- The server is now able to use HDFS for disk storage in addition to type 60 partitions and flat files. This is the default mode of operation when running on Hadoop and is configured automatically. HDFS storage uses a directory in HDFS to store disk blocks as HDFS files. For standalone systems, configure Hadoop and Java in the config file then set the disk resource to 'hdfs:/path/to/folder'. See 'HDFS disk storage' below for details.
- Kognitio can now be configured to authenticate via Kerberos and Active directory. It can be configured to automatically accept users from active directory without pre-creating accounts for that user on the server with the user's permissions being automatically picked up based on the active directory groups the user is in. See 'Active directory integration' below for details.
- Kognitio can now load data in the JSON format. This can be done with wxloader and via external tables. See JSON loading below for details.
- An extended set of geographic functions from the GEOS library is now provided via the 'fullgeometry' plugin which ships with the standard install.
- It is now possible for a user to reset an expired password by connecting to the server using kognitio console, authenticating with the expired password and specifying a new one. This feature is off by default and needs to be enabled by the sys user by setting bit 0 of the global am_pwd_policy parameter.
- The transaction mode behaviour clause ('At XXX' at the end of the query) has been extended to allow extra fuctionality. See the 'at modes' section below for more information.
- Online documentation for the product is now available at http://www.kognitio.com/documentation/latest. This is currently a work in progress and the

full set of documentation will be added to this over time.

- Check on parity recreate - setting parity_c_write to 1 makes parity writes be checked against original contents read first, and only do the write if a difference. Also, log differences to serverdbg. Parameter defaults to 0 (off).

Known Issues

1. When running on Hadoop, if the 8.2 version of the kodoop command is used to create or start up instances running with a release earlier than 8.1.50-rel170105 then attempting to upgrade those instances will not work. In order to use the 8.2 kodoop script to upgrade these older versions, to stop the instance and offline upgrade to 8.1.50-rel170105 before attempting to upgrade to 8.2.

Additional Information

Sys Ramstores

Sys ramstores are a special set of ramstores which are dedicated to servicing system table and metadata queries. The server will create one 512M sys ramstore per node with a 50% PMA percentage. The other ramstores are now known as 'global ramstores'. Memory images for objects in the SYS schema will be put into the sys ramstores instead of the global ramstore set. This should be completely transparent to the user -- you can still query objects in the sys schema in the same way as before. If you can join them to non-sys objects the server will migrate the data to the appropriate ramstore set at runtime.

This feature enables the server to quickly compile SQL and respond to system table/metadata queries even when the server is under very heavy load. Users may want to consider placing small and frequently accessed user tables (Process control/co-ordination tables, etc) in the sys schema to speed up queries on their objects.

Global ramstores will be slightly smaller than they were in 8.1.0/8.1.50 as a result of this feature but this should be offset by the larger PMA percentage for global ramstores (85% instead of 70%) and the memory freed up by not putting replicated system table copies into the global ramstores.

The sys ramstore feature can be disabled by setting '[boot options] sys_ramstores=no' in the global configuration file. A server restart is required to enable or disable sys ramstores and the server will not be able to recover memory images if this setting has been changed.

The sys ramstores are smaller and fewer in number than the global ramstores. It is therefore not recommended to store very large amounts of user data in tables in the 'sys' schema, which is intended for system tables, process control tables, logging tables and the like. We have disabled creation of user objects in the sys schema by default to prevent users from accidentally using the sys ramstores for user data without realising. The behaviour of objects in the sys schema can be controlled with the sys_objects runtime parameter, which can be set in the config file or defined globally, per user or per session at runtime. This parameter can have one of 3 values:

- 0 -- Disable creation of views/tables in the sys schema. This is the default behaviour.
- 1 -- Enable creation of views/tables in sys. Memory images created go into the sys ramstores.
- 2 -- Enable creation of sys objects. Memory images for non-system tables go into the global ramstores and only system tables are put in the sys ramstores.

This parameter only needs to be set while objects are being created and placed into memory. Once

placed objects will stay where they have been put regardless of changes to the sys_objects parameter. It is therefore possible, for example, to create a process control table in the sys schema like this:

```
set current_session parameter sys_objects to 1;
create table sys.process_control (job_id int, state varchar(20));
drop current_session parameter sys_objects;
```

Sys ramstores populate the system management virtual tables in the same way as regular ramstores, so querying ipe_ram_access, for example, will return accesses from both sys and global ramstores. Sys ramstores appear in the ipe_process table with a type of 'srs' (global ramstores are 'rs') and this can be used to filter rows from the other tables.

Asymmetric Query Processing

Previous versions of Kognitio had only two modes for running query plan steps -- either globally or on a single ramstore. Every step in every query had to be run in one of these two ways. Now in 8.2 the server is able to dynamically scale queries to run on differently sized groups of ramstores based on estimates of the work to be performed and the amount of other work being run concurrently. This enables higher query throughput on systems which run a very large number (high tens or hundreds) of relatively short queries concurrently. This feature improves performance because starting and stopping very large numbers of tiny tasks places a significant load on the server.

The formula the server uses to size a query also takes into account the concurrent workload on the server at the time. With a very small number of queries running in parallel most query plan steps will run on all of the ramstores and only the really small steps will be run on reduced sets. As concurrency goes higher and higher the ramstore sets being used will get smaller and smaller.

The asymmetric query processing feature is switched on by default and can be switched off with the ai_auto_asym runtime parameter, which can be set in the config file or globally, per-user or per-session at runtime.

Location hash Values

The Location Hash Value (LHV) feature was introduced into 8.1 to allow certain queries to be run single ramstore. This feature worked by looking at filters applied to the hash column of a table image and using these to deduce that the whole query could be serviced by a single ramstore. For example if you have a hashed table:

```
create table t (f1 int, f2 varchar) hashed (f1);
```

The table uses a hash function on f1 to map rows into ramstores. So if the user runs a query like this:

```
select * from t where f1 = 7;
```

Then we know we only need to consider the ramstore that 7 hashes to. So we hash the number 7 to make the LHV and then use this to direct the query to a single ramstore. This greatly improves query throughput for servers which are running a large number of highly selective queries concurrently by eliminating the need to set up and tear down a lot of operations which aren't going to do anything.

In 8.2 this feature has been extended via the asymmetric query enhancements to allow multiple LHVs per query. This means that it can cope with cases where the filter would map to more than

one ramstore, for example:

```
select * from t where f1 in (1, 2, 3, 4, 5, 6, 7);
```

In this case up to 7 LHVs would be produced (some might be duplicates) and the query would run on up to 7 ramstores.

The LHV feature is also able to infer LHV values from joins to small or highly filtered tables. This has been improved since 8.1 to allow the inference of multiple LHV values and to use these during selectivity gathering and other parts of query compilation, meaning that the server should never perform a full non-LHV scan of a table if there are valid LHV extractions in the query. For example:

```
create table image fact hashed(fk);
select ... from fact, dim
    where fk = dim_pk
     and  dim.name in ('a', 'b', 'c', 'd', 'e', 'f');
```

In this case the server will extract the dim_pk values matching the filters, make LHV values from those and apply those values when selecting from the 'fact' table. Cases where it was necessary to denormalise images in 8.1 to make use of LHV can now be run using normalised images and joins and still get the scale-out and concurrency benefits of LHV.

LHV optimisations are on by default. To make use of these simply hash the image on a column which will be filtered using constants or values derived from a join to a lookup table (the lookup table should be replicated for best performance). The feature can be disabled by setting the runtime parameter ai_auto_lhv to 0 in the config file or at the global, user or session level at runtime.

Hadoop and Java Configuration

Kognitio 8.1.50 and 8.2 have the concept of a system-wide Hadoop and Java configuration. This is a set of configuration values for Java and Hadoop settings which are detected by the server at startup or defined in the server's config files. Plugins, external scripts, script based connectors and any other components can pick these values up and use them, removing the need to specify things like JAVA_HOME over and over again. In all cases it is still possible to specify values directly when creating components, the configuration settings are only used where no other values are specified.

When running on Hadoop, Kognitio should pick up all of the required values from the Yarn task so you shouldn't need to define anything at all. When running standalone the server will try to determine these values by looking at the login environment for the root user. If every node is configured so that root can run 'hadoop fs -ls' from its default login environment and have it work then the server should pick up everything it needs and no further configuration is required.

If necessary the Hadoop and Java configuration settings can be specified manually in the server's local or global config files. These settings are per-node and can be different on each node if necessary. The following settings are available:

```
[hadoop]
java_home=/path/to/java/home

hadoop_conf_dir=/path/to/hadoop/conf/dir
hadoop_prefix=/path/to/hadoop/install/dir

classpath=the:java:classpath:to:use

libhdfs_path=/path/to/libhdfs.so
libjvm_path=/path/to/libjvm.so
```

```
hdfs_host=name.or.ip.of.namenode
hdfs_port=<namenode port>
user=<user ID to use for HDFS>
```

HDFS Disk Storage

Kognitio 8.1.50 and 8.2 have the ability to use HDFS storage as diskstore storage in addition to flat files or type 60 partitions. HDFS volumes work in a similar way to flat files, using HDFS storage to emulate a raw disk volume. Due to the nature of HDFS the server uses a tree of files to emulate the volume instead of a single flat file. Compression is used and zeroed blocks are eliminated to return storage back to hadoop. HDFS volumes can do all the things that other disk types can do, so they have slabs inside them, need to be reclaimed, can be repacked to zero slabs, etc just like regular disk volumes.

When running on Hadoop HDFS storage is the only option available and the server will automatically be configured to use a number of HDFS volumes. If desired, standalone systems can also be configured to run with HDFS storage. When configuring HDFS disk storage for standalone you first need to have a working Hadoop and Java configuration (see above). Then you can set the server to use HDFS storage by setting the following in the [disks] global config file section:

```
[disks]
type=hdfs
size=<number followed by G, e.g. 100G>
diskcount=<number of volumes>
```

This tells the server to use HDFS storage, tells it the size of each emulated disk volume and the number of emulated volumes to create in each specified path. Then you can set the '[system] disks=' setting to be one or more directories in HDFS. Emulated volumes will be placed under these directories. If you give the same HDFS path on multiple nodes (for example if you put the disks setting in the global config file) then multiple nodes will be able to see the same HDFS volumes. At startup time the server will automatically place one diskstore for each volume in a balanced way.

Active Directory Integration

It is now possible to authenticate using Active Directory / Kerberos, or potentially other mechanisms supported by the GSS API. Building on top of this capability allows users to be auto-created, and permissions assigned based on Active Directory group membership.

An already-existing Kognitio user can be associated with a principal name (a Kerberos term referring to someone's identity – effectively a Kerberos username).

In addition, an Active Directory group can be associated with a Kognitio group so that they are considered equivalent, by creating an external group mapping.

Finally, an external group mapping can be configured such that a new user can authenticate with a principal name and connect to Kognitio without specifying a Kognitio username, and the principal's membership of an associated Active Directory group will cause the Kognitio username to be automatically created and associated with the principal concerned.

Kognitio server-side Configuration

Kognitio needs to be able to authenticate clients using Kerberos. To this end, certain libraries need

to be present on all database nodes: * libkrb5.so (package name krb5-32bit-... which may rely on other packages) * libldap.so (package name libldap-...) * cyrus-sasl-gssapi

We recommend using at least SLES11 if using the SLES Linux distribution, to minimise missing dependencies on other libraries.

You also need appropriate realm details in /etc/krb5.conf on all nodes. Typically that file will have entries like the following:

```
[libdefaults]
        default_realm = DEVELOPMENT.LOCAL
        rdns = false
        default_tkt_enctypes = arcfour-hmac-md5 des-cbc-crc des-cbc-md5
        default_tgs_enctypes = arcfour-hmac-md5 des-cbc-crc des-cbc-md5

[realms]
        DEVELOPMENT.LOCAL = {
                kdc = 172.30.0.4
                admin_server = 172.30.0.4
        }
```

[arcfour-hmac-md5 entries are to allow the DB nodes to be able to understand service principal keys generated on Windows]

You need to add lines like the following to the config file as the wxroot user, with the server principal name matching whatever you specified as the principal name when creating the service instance account. To use external groups with Active Directory you need to add the ldap_server_name setting, and this should be a hostname rather than just an IP address (so you might have to add an entry in /etc/hosts on every DB node):

```
[general]
server_principal_name=kognitio/<hostname>@<YOUR.REALM.NAME>
ldap_server_name=DEV-AD_DC.development.local
```

You need to restart the Kognitio system for this change to take effect.

Kognitio client-side Configuration

Add the following to your odbc.ini file in the appropriate DSN section:

```
GSSEnabled=Y
GSSServicePrincipalName=<server principal name>
```

The first line enables GSS authentication in the absence of a password, or if a blank password is provided.

The second line prevents being prompted to verify the server principal name on first connection.

If GSSServicePrincipalName is not set, then if you have a terminal, the first time you connect, you will be prompted to confirm that the service principal name of the server matches the service principal name you intend to connect to. If you say yes, a line will be added to a file called ~/.kogknownservices associating the DSN with that service principal name, and when you connect to that DSN in the future, this file will be consulted and the client will only authenticate if the server's principal name matches this.

This approach to discovering the service name, where the server is asked for its name the first time the client connects, and the user is asked to verify it, is also used by SSH (hence the warnings it gives you when the server's host key changes), and is known as the leap-of-faith approach. It is described in page 15 of http://kerberos.org/software/appskerberos.pdf in the section "Discovering the service name".

If the server's principal name changes later on then when you connect you'll get an error telling you to verify you're connecting to the service you think you're connecting to, and if you are, edit the file accordingly.

Optionally, you can also set these:

```
GSSEncryption=Y           # use GSS to encrypt the session, rather than SSL. Y
is the default.
GSSEncryptionAllowFallback=N   # fail if we can't use encryption for some
reason. Y is the default, which means fallback to plaintext if encryption is not
available at the GSSAPI layer.
```

For Linux clients, you will need to install the Kerberos client tools and add details of the domain to /etc/krb5.conf as shown earlier on the client side. You will also need to run kinit, which will prompt for a password, to add the appropriate ticket to your credential cache so the GSS library can find it:

```
wxadmin wxadmin@hp-rack1-enc5-7:~> kinit testy.mctestface@DEVELOPMENT.LOCAL
Password for testy.mctestface@DEVELOPMENT.LOCAL: <enter the Windows password for
testy.mctestface>
wxadmin wxadmin@hp-rack1-enc1-1:~> wxsubmit -s myserver testy
Kognitio WX2 SQL Submission Tool v8.02.00
(c)Copyright Kognitio Ltd 1992-2016.

Connected to myserver ODBC Version 8.02.00 Server Version 08.02.0000
>
```

The ticket created with kinit will persist for around 10 hours (you can check with klist). When it expires, you need to run kinit and enter your password again.

For Windows clients, the ODBC setup dialogue has an "Authentication and encryption" section; click the "Kerberos authentication using Windows credentials". Then press the "Discover" button to ask the Kognitio nodes what the server principal name is – the name should appear in the text box next to the Discover button. Press "OK" to save changes.

Database new tables

A new system table, IPE_ALLPRINCIPAL_NAME, maps a Kognitio user id to a principal name with a priority setting.

IPE_ALLEXTERNAL_GROUP associates an Active Directory group with a Kognitio group.

IPE_CURSESSION_GROUP shows which groups a user's session is a member of, either via direct inclusion with ALTER GROUP … ADD USER…, or because of equivalence between Active Directory groups the user is in, and Kognitio groups.

Supporting SQL syntax

You need the ADD/DROP PRINCIPAL privilege on a user to change the principal names for that user. These are granted via

```
GRANT { ADD | DROP } PRINCIPAL ON <user1> TO <user2>
```

To modify the principal names associated with a user:

```
ALTER USER { ADD | DROP } PRINCIPAL <principalname> [ PRIORITY <n> ]
```

The priority clause is only valid when adding a principal – the user entry with the lowest priority associated with a principal is used when authenticating without specifying a Kognitio user name.

External groups

An external group has an entry in IPE_ALLEXTERNAL_GROUP mapping an Activity Directory group to a Kognitio group.

A Kognitio user authenticated via Kerberos, and who is a member of the Active Directory Group, will be treated as being in the specified Kognitio group in terms of privileges. Group memberships are considered transitive, so if a user is a member of Active Directory group G1, which is in turn a member of Active Directory group G2, that user will be considered to be a member of both G1 and G2, and will have all the same rights associated with G2 as if it were a member of G2 directly.

A Kognitio user's Active Directory group membership is established by Kognitio at the start of a session, so changing a user's Active Directory group membership will take effect for new sessions for that user, but not existing ones.

To create an external group, the syntax is:

```
CREATE EXTERNAL GROUP <canonical name of AD group>
GROUP <Kognitio group name>
[CAN LOGIN]
[TEMPLATE USER <user>]
[PRIORITY <n>]
```

The final three optional clauses are only relevant for autocreated users (see below).

To disassociate an AD group from a Kognitio group, the syntax is:

```
DROP EXTERNAL GROUP <canonical name of AD group>
```

Automatically created users

If a client connects and authenticates via Kerberos, but does not specify a user name, the principal name they authenticated with is looked up in IPE_ALLPRINCIPAL_NAME and the Kognitio user with the lowest priority value that matches is used.

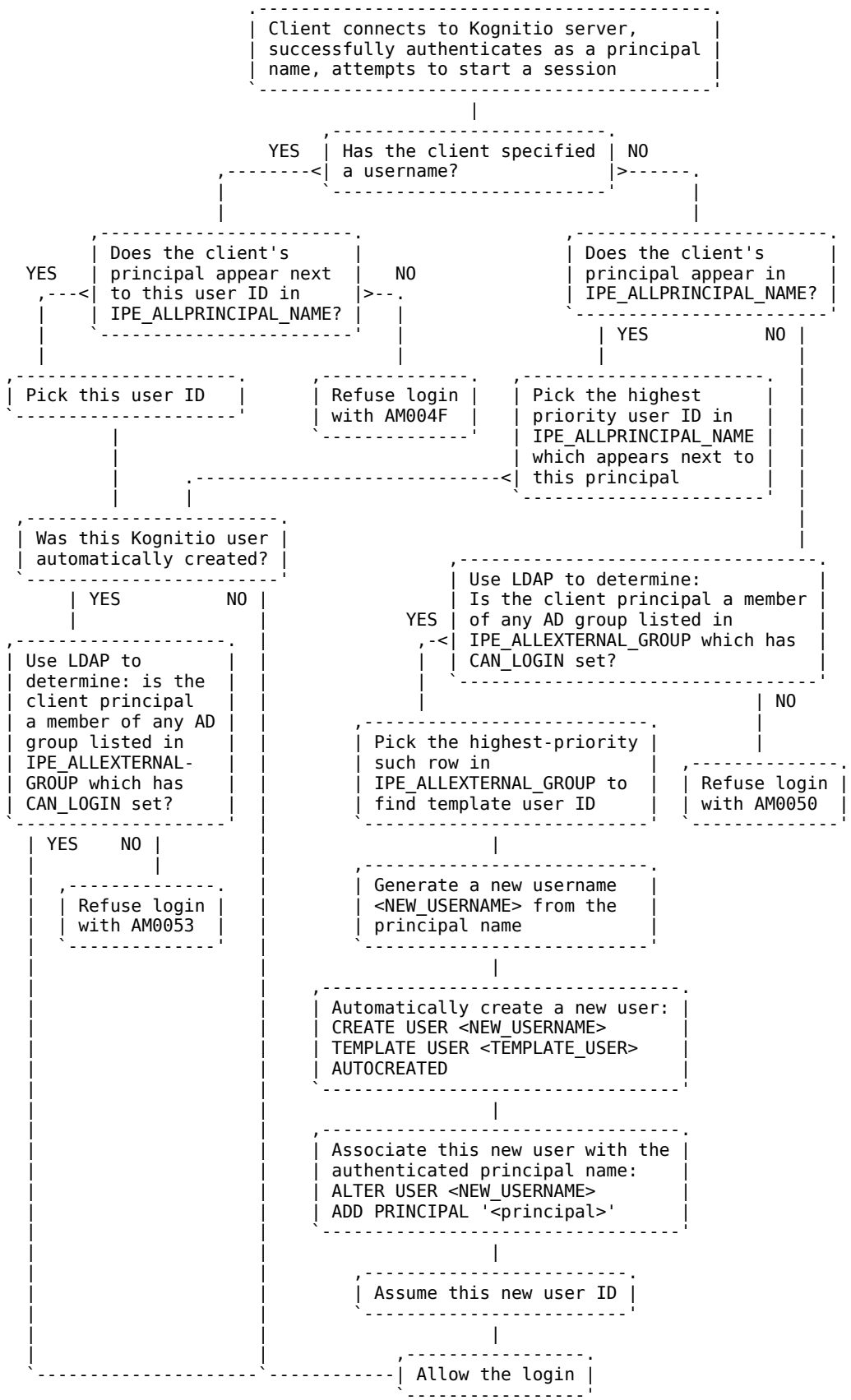
If the client's principal does not appear in IPE_ALLPRINCIPAL_NAME, then if the AD user is a member of at least one group which appears in IPE_ALLEXTERNAL_GROUP with the CAN_LOGIN field set, then a Kognitio username is created automatically for the user, and the client logs in as this user. The user is then associated with the principal name by adding a new row to IPE_ALLPRINCIPAL_NAME. The Kognitio username is derived from the principal name. If there are multiple matching external group entries, the one with the lowest priority value is used.

The new Kognitio user is created from the specified template user (same per-user parameters, queue settings, and privileges). If the template user's default schema matches the template user name, the new user will have their own schema created matching their user name, and that will be their default schema.

Automatically created users have their IPE_ALLUSER.STATUS value set to 16 to distinguish them from manually created users. If their principal name is later removed from all AD groups which gave them CAN_LOGIN status, the user will no longer be allowed to log in.

Summary of authentication process

The following flowchart explains the process used when authenticating with Kerberos:



The error codes referenced above expand as follows:

```
AM004F  Authentication succeeded but you are not authorised to log on as this
user

AM0050  Not authorised to log in without a pre-existing username

AM0053  Automatically created user is not a member of an external group with
permission to log in
```

More background references on Kerberos

Useful links on setting up and using Kerberos:

http://www.tldp.org/HOWTO/html_single/Kerberos-Infrastructure-HOWTO/

http://web.mit.edu/kerberos/krb5-1.3/krb5-1.3.5/doc/krb5-install.html

JSON Loading

Some users have JSON data they would like to query in the database. The JSON needs to be converted into a tabular format so that it can be represented by a table or an external table. How this is done depends on the structure of the data.

wxjson2csv is a program which takes a JSON file, some instructions on how to generate CSV records from it, called the column definition, and the name of the array in which the relevant objects can be found (if applicable), and produces a CSV file. See the end of this section for details on how to use wxloader to load data into KAP without converting into CSV as an interim step.

For example, suppose we have a JSON file containing information about different types of fruit.

```
{
    "fruits" : [
        { "name" : "apple", "colour" : "green", "tastiness" : 8 },
        { "name" : "orange", "colour" : "orange", "tastiness" : 9 },
        { "name" : "banana", "colour" : "yellow", "tastiness" : 4 },
        { "name" : "pear", "colour" : "green", "tastiness" : 6 }
    ]
}
```

If you wanted to represent this information in a FRUIT table, it's easy enough:

```
name    colour  tastiness
apple   green   8
orange  orange  9
banana  yellow  4
pear    green   6
```

The column definition for wxjson2csv to convert this JSON to CSV would therefore be:

```
name, colour, tastiness
```

You would run wxjson2csv like this, with the JSON input in "fruits.json". The -a fruits means that the array of objects we need to look at is called fruits.

```
$ wxjson2csv -j "name, colour, tastiness" -a fruits fruits.json
apple,green,8
orange,orange,9
banana,yellow,4
pear,green,6
```

Alternatively, you can specify the target array name in square brackets at the start of the column definition list, like this:

```
$ wxjson2csv -j "[fruits] name, colour, tastiness" fruits.json
```

The rest of this document will use this notation.

Nested objects

A JSON object representing a single entity is rarely a simple flat structure containing only named values. Objects may contain other objects. For example, here is a possible JSON representation of a list of towns with their locations.

```
{
    "towns": [
        {
            "name" : "Bracknell",
            "location" : {
                "latitude" : 51.408333,
                "longitude" : -0.756666,
                "osgridref" : {
                    "northing" : 168500,
                    "easting" : 486500
                }
            }
        },
        {
            "name" : "Reading",
            "location" : {
                "latitude": 51.455000,
                "longitude": -0.970000,
                "osgridref": {
                    "northing": 173500,
                    "easting": 471500
                }
            }
        },
        {
            "name" : "Bristol",
            "location" : {
                "latitude" : 51.476666,
                "longitude" : -2.568333,
                "osgridref" : {
                    "northing" : 175500,
                    "easting" : 360500
                }
            }
        }
    ]
}
```

To refer to names of members inside other named objects, use the "." character to separate names in the hierarchy. For example, if we specified this column list:

```
[towns] name, location.latitude, location.longitude
```

We would get this output:

```
$ wxjson2csv -j "[towns] name, location.latitude, location.longitude" towns.json
```

```
Bracknell,51.408333,-0.756666
Reading,51.455000,-0.970000
Bristol,51.476666,-2.568333
```

Or, we could include the grid references as well:

```
[towns] name, location.latitude, location.longitude,
location.osgridref.northing, location.osgridref.easting
```

To give this:

```
Bracknell,51.408333,-0.756666,168500,486500
Reading,51.455000,-0.970000,173500,471500
Bristol,51.476666,-2.568333,175500,360500
```

Dimensions

JSON files with nested objects, like the one above, are essentially no different from flat objects, except with some of the key values grouped in a certain way. These are easily dealt with. However, in some JSON files, an object representing a single entity might contain an array of other objects, each of which represent another entity, and you want one output record for each. For example, suppose you have a JSON file which contains an array, where each element of the array represents a person. Each person has many hobbies, which are listed in an array.

```
{
    "people" : [
        {
            "name" : "Alice",
            "hobbies" : ["skiing", "cycling"]
        },
        {
            "name" : "Bob",
            "hobbies" : ["birdwatching", "cycling", "bridge"]
        },
        {
            "name" : "Charlie",
            "hobbies" : ["skydiving", "boxing", "chess"]
        }
    ]
}
```

Our target array, "people", is the array which will be split up and each element converted into a record. But the "hobbies" array means that it isn't normalised. So how do we make a CSV file out of this? We don't want one field to contain a list, and we don't want a variable number of fields per record. Ideally we want one record per name-hobby pair.

The solution is to use the dimension operator, [].

To output one CSV record for every person's hobby, the column definition would look like this:

```
[people] name, hobbies[]
```

The [] on the end of "hobbies" means to iterate over the "hobbies" array and put each value in turn into that column.

The CSV output would look like this:

```
Alice,skiing
Alice,cycling
Bob,birdwatching
Bob,cycling
```

```
Bob,bridge
Charlie,skydiving
Charlie,boxing
Charlie,chess
```

Nested objects underneath dimensions

What if the array inside your entity object contains a series of JSON objects, rather than values? You can specify how to find the value inside each object of the sub-array using the dot-notation, as before. For example, suppose your JSON file is a list of houses and the people who live there:

```
{
    "houses": [
        {
            "address" : "221B Baker Street, London",
            "occupants" : [
                {
                    "name" : "Sherlock Holmes",
                    "occupation" : "Consulting Detective"
                },
                {
                    "name" : "John Watson",
                    "occupation" : "Doctor"
                }
            ]
        },
        {
            "address" : "62 West Wallaby Street",
            "occupants" : [
                {
                    "name" : "Wallace",
                    "occupation" : "Inventor"
                },
                {
                    "name" : "Gromit",
                    "occupation" : "Dog"
                }
            ]
        }
    ]
}
```

A suitable column definition would be as follows:

```
[houses] address, occupants[].name, occupants[].occupation
```

This means "address, the name field in each element of occupants, and the occupation field in each element of occupants".

wxjson2csv will recognise that the occupants[] in the second field and the occupants[] in the third field are the same expression, so it will only iterate through them once - you don't need to worry about accidentally generating the cartesian product of all the names and occupations.

The resulting CSV would be:

```
"221B Baker Street, London",Sherlock Holmes,Consulting Detective
"221B Baker Street, London",John Watson,Doctor
62 West Wallaby Street,Wallace,Inventor
62 West Wallaby Street,Gromit,Dog
```

Sequence numbers and matrices

wxjson2csv can convert arrays of arrays, using dimensions of arbitrary depth as long as what's specified in each column of the column definition corresponds to a scalar value in the object.

A list of two matrices might be specified like this in JSON:

```
{
    "matrices" : [
        {
            "matrixid" : 10001,
            "cells" : [
                [0.1, -0.1, 0.8, 1.3],
                [0.0, 0.4, 0.8, 1.2],
                [-0.3, -0.9, -1.7, -2.5]
            ]
        },
        {
            "matrixid" : 10002,
            "cells" : [
                [9.3, 8.7, 6.0],
                [1.2, 3.4, 5.6]
            ]
        }
    ]
}
```

Your output records need to include the matrix ID, and not only the value of each cell but also its row and column number.

The seq() function takes a dimension expression and returns the current position in that array. So, specifying "matrices" as the target array, a suitable column definition would be this:

```
[matrices] matrixid, seq(cells[]), seq(cells[][]), cells[][]
```

The last column, cells[][] means to iterate over the array cells, and for each (array) element in it, iterate over each element in that and make each one in turn the column's value. The third column, seq(cells[][]) means the position of the current cells[][] value in its array, and the second column, seq(cells[]) means the position of the current cells[] value in its array. In other words, the second column is the row number and the third column is the column number in the matrix.

The result would be this:

```
10001,0,0,0.1
10001,0,1,-0.1
10001,0,2,0.8
10001,0,3,1.3
10001,1,0,0.0
10001,1,1,0.4
10001,1,2,0.8
10001,1,3,1.2
10001,2,0,-0.3
10001,2,1,-0.9
10001,2,2,-1.7
10001,2,3,-2.5
10002,0,0,9.3
10002,0,1,8.7
10002,0,2,6.0
10002,1,0,1.2
10002,1,1,3.4
10002,1,2,5.6
```

The target array

According to the standard, a valid JSON document must be a single object, although that object can be arbitrarily complex, containing any number of arrays and nested objects.

Our JSON conversion code assumes there will be an array somewhere in the JSON object, which can be split up into each of its elements, and each element can be converted into one or more tabular records without reference to any other part of the JSON document.

In some cases, the input file does not contain a single JSON object, but several objects, one after the other. Alternatively, the document might contain only a JSON array, which might be our target array. In both of these cases, specifying no target array at all means that the conversion code will expect either an array of objects (in which case each object is converted individually, and the names in the column definition are in the namespace of that object), or a file containing many objects one after the other (again, each of these objects is given to the column definition to extract records).

Absent fields

Sometimes, not all the fields you want will be present in an object.

Let's say this is your JSON input, containing details of some US Presidents:

```
{
    "presidents" : [
        {
            "firstname" : "Franklin",
            "middleinitial" : "D",
            "surname" : "Roosevelt",
            "start" : "1933-03-04",
            "end" : "1945-04-12"
        },
        {
            "firstname" : "John",
            "middleinitial" : "F",
            "surname" : "Kennedy",
            "start" : "1961-01-20",
            "end" : "1963-11-22"
        },
        {
            "firstname" : "George",
            "middleinitial" : "W",
            "surname" : "Bush",
            "start" : "2001-01-20",
            "end" : "2009-01-20"
        },
        {
            "firstname" : "Barack",
            "surname" : "Obama",
            "start" : "2009-01-20"
        }
    ]
}
```

You might use the following column definition to list the names of these presidents and the dates they entered and left office:

```
[presidents] firstname, middleinitial, surname, start, end
```

However, if you run wxjson2csv, giving the above column definition, you get this error message:

```
wxjson2csv: presidents.json: object 4, ending on line 28: value not present in
object: middleinitial
wxjson2csv: too many errors (more than 0)
```

Barack Obama doesn't use his middle initial so it isn't included in the object that represents him. In addition, his "end" field isn't included because he's still the president (times have changed since this example was created!). By default, if you specify a JSON name in the column definition which does not exist in one of the objects in the target array, this is an error.

If you want a field to be optional, you can use the firstvalid() function. This function takes an unlimited number of arguments, and returns the first argument whose evaluation does not cause a runtime error. In this case, you can use firstvalid(middleinitial, null), which means "the value of the middleinitial property, or null if that doesn't exist".

```
[presidents] firstname, firstvalid(middleinitial, null), surname, start,
firstvalid(end, null)
```

Note that an attribute setting comes after the column name it corresponds to, and consists of a series of name=value pairs separated by whitespace.

This will cause the following records to be generated:

```
Franklin,D,Roosevelt,1933-03-04,1945-04-12
John,F,Kennedy,1961-01-20,1963-11-22
George,W,Bush,2001-01-20,2009-01-20
Barack,,Obama,2009-01-20,
```

Further details on the expressions and functions you can use in the JSON format string are in the wxjson2csv man page.

wxloader accepts the same JSON format string as wxjson2csv (use -j <format string> or -J <file containing format string>), so if you want to load the JSON data straight into the server rather than convert it to CSV and load that, you can.

At Modes

At modes are used to control the transactional behaviour of a query. You can specify an "AT mode" for a SELECT in all versions of Kognitio. Prior to 8.1.50 this could only be one of:

- AT_NOW -- run point in time from the start of the current transaction without getting locks
- AT_FULL_HISTORY -- run from disk and see deleted rows
- AT_CURRENT_TRANSACTION (the default) -- get locks and run normally.

These actually control several behaviours and we decided it would be useful to add flags to achieve these individually. Now the 'at mode' is a combination of comma separated values which can be chosen from the above and the following new modes:

- AT_IGNORE_LOCKS - don't obtain locks on user objects
- AT_FORCE_DISK - get rows from disk, ignoring table/view images
- AT_FORCE_RANDOM - treat ram image as random; thus if it is replicated you will get a copy of each row from each ramstore. wx_ram_proc() and wx_ram_addr() are useful here.
- AT_INCLUDE_DELETED - include rows that were deleted prior to the current transaction (i.e. see deleted rows). wx_create_tno() and wx_update_tno() are useful here; if wx_update_tno() <> 2147483647 then the row has been deleted/updated.
- AT_INCLUDE_UNCOMMITTED - include rows that have been created since the start of

the current transaction (i.e. see uncommitted rows). wx_create_tno() is useful here.

The default mode for select queries with server-side autocommit (requires >= 8.1.50 ODBC driver) has been changed in 8.1.50 and 8.2 to AT NOW so that sessions just doing selects wouldn't get shared locks. The default at mode is controlled with the parameter default_at_now (default 2; set to 0 to get shared locks). Subquery selects within e.g. an update will still get shared locks.

Syntax:

```
[with ...] select ... [from ...] [where ...] [group by ...] [having ...] [at
<at_mode>]
```

Examples:

```
-- Setup create table MY_TABLE (C1 int) replicated; insert into MY_TABLE values
between 0 and 2; delete from MY_TABLE where C1 <> 1; insert into MY_TABLE values
(2);

-- 2 rows returned, values (1), (2) select * from MY_TABLE;

-- 4 rows returned, values e.g. (599, 600, 0), -- (599, 2147483647, 1), -- (599,
600, 2), -- (600, 2147483647, 2) select wx_create_tno(), wx_update_tno(), * from
MY_TABLE at include_deleted;

-- 2 * nrs rows returned, values e.g. (0, 1), (0, 2), ..., (7, 1), (7, 2) select
wx_ram_proc(), * from MY_TABLE at force_random;

-- Access from replicated ("REPL") table picture select * from MY_TABLE;

-- Access from disk ("DISK" via "RAND"/"LOAD") table picture select * from
MY_TABLE at force_disk;
```

The Fullgeometry Plugin

The new fullgeometry plugin is an implementation of the OGC standard, which can be found at http://portal.opengeospatial.org/files/?artifact_id=25354. The implementation uses “Well Known Text” (WKT) as the textual representation, and “Well Known Binary” (WKB) as the internal representation (https://en.wikipedia.org/wiki/Well-known_text) – with one minor caveat. We also support Extended Well Known Binary/Text (EWKB/EWKT), which contain additional information regarding the SRID used (if any), and can contain 3D points.

Inside Kognitio WKB data is stored in as varbinary data using an internal representation. Data will need to be converted to/from Kognitio's WKB representation to be imported/exported from the system. To use WKB produced by an external system, use ST_FromExternalWKB() to convert it into our internal format. To output something in standard WKB (so that it can be used by an external system), use ST_AsBinary(), ST_AsHEXBinary(),ST_AsEWKB(), or ST_AsHEXEWKB().

All the other functions can be passed in either our internal representation of WKB, or WKT – of the two, internal WKB is much more efficient. When they return a geometry it will be in our internal WKB format – use ST_AsText() to convert this to a human readable WKT format. Later on we will use the word geometry to indicate that either WKT or WKB can be passed in to many of our functions.

For example:

```
select ST_Envelope ('MULTILINESTRING ((10 10, 20 20, 10 40),(40 40, 30 30, 40
```

```
20, 30 10))');
```

returns the same answer as

```
select st_envelope (
x'4024000000000000403E000000000000403400000000000040440000000'
'0000403E000000000000403E000000000000404400000000000040440000'
'0000000000000004000000020140440000000000004024000000000000040'
'3400000000000040340000000000004024000000000000402400000000000'
'0000000003000000020100000020000000501');
```

To convert this to human-readable WKT, use ST_AsText:

```
select ST_AsText (ST_Envelope ('MULTILINESTRING ((10 10, 20 20, 10 40),(40 40,
30 30, 40 20, 30 10))'));
```

returns “POLYGON ((10 10, 40 10, 40 40, 10 40, 10 10))”

Note that as our functions are all standard, a web search will bring up documentation on the functions if more information is needed.

Loading the fullgeometry plugin

The fullgeometry plugin ships in the standard Kognitio release package but is not activated by default. To activate the plugin, use this command:

```
CREATE MODULE FULLGEOMETRY MODE ACTIVE;
```

This will load the fullgeometry plugin (named fullgeometry.wxpi) and activate it ready for use.

Supported Functions

```
varbinary ST_Geom (varchar WKT)
varbinary ST_GeomFromEWKT (varchar WKT)
```

These two functions convert WKT to internal WKB. e.g.

```
SELECT ST_GEOM (‘POINT (1, 1)’);
```

```
varbinary ST_WKTToSQL (varchar WKT [,integer SRID])
varbinary ST_GeomFromText(varchar WKT [,integer SRID])
varbinary ST_GeometryFromText (varchar WKT [,integer SRID])
varbinary ST_Point(varchar WKT [,integer SRID])
varbinary ST_Line(varchar WKT [,integer SRID])
varbinary ST_LineString(varchar WKT [,integer SRID])
varbinary ST_Polygon(varchar WKT [,integer SRID])
varbinary ST_Triangle(varchar WKT [,integer SRID])
varbinary ST_GeomCollection(varchar WKT [,integer SRID])
varbinary ST_MultiPoint(varchar WKT [,integer SRID])
varbinary ST_MultiLineString(varchar WKT [,integer SRID])
varbinary ST_MultiPolygon(varchar WKT [,integer SRID])
varbinary ST_PointFromText(varchar WKT [,integer SRID])
varbinary ST_LineFromText(varchar WKT [,integer SRID])
varbinary ST_LinestringFromText(varchar WKT [,integer SRID])
varbinary ST_PolyFromText(varchar WKT [,integer SRID])
varbinary ST_TriFromText(varchar WKT [,integer SRID])
varbinary ST_GeomCollFromText(varchar WKT [, integer SRID]))
varbinary ST_MPointFromText(varchar WKT [, integer SRID])
varbinary ST_MLineFromText(varchar WKT [, integer SRID])
varbinary ST_MPolyFromText(varchar WKT [, integer SRID])
```

These functions convert WKT to internal WKB, and take an optional SRID parameter. e.g.
```
SELECT ST_GEOM ('POINT (1, 1)', 4326);
```

```
varchar ST_AsText (varbinary WKB [, format string])
```

This converts our internal WKB to text. The arguments are a string indicating precision and trim values to use. e.g. `SELECT ST_AsText (geom.wkb, 'precision=4 | trim=2');`

```
varchar ST_AsEWKT (varbinary WKB [, format string])
```

This converts the WKB to EWKT, taking an optional argument string that specifies the precision and trim values to use. e.g.
```
SELECT ST_AsEWKT (geom.wkb, 'precision=4 | trim=2');
```

```
varbinary ST_FromExternalWKB (Varbinary WKB)
```

This converts external WKB to our internal format. It is designed to be used with WKB produced by other systems.

```
varbinary ST_AsBinary (geometry)
varchar ST_AsHEXBinary (geometry)
varbinary ST_AsEWKB (geometry)
varchar ST_AsHEXEWKB (geometry)
```

These all convert either WKB or WKT into standard WKB, and should only be used to allow the data to be used on a different system that uses standard WKB.

```
varbinary ST_MakePoint (x, y [,z])
varbinary ST_Point (x, y [,z])
```

These make a point with the given X, Y [, Z] coordinates. e.g. `SELECT ST_POINT (1, 200);`

```
varbinary ST_MakeLine (x1, y1 [,z1], x2, y2 [,z2])
varbinary ST_MakeLine (point1, point2)
```

This makes a line. In the second case it is passed in two points, e.g.
```
SELECT ST_AsText (ST_MakeLine (ST_Point (1, 1), ST_Point (4, 5)));
```

```
varbinary ST_MakeEnvelope (xmin, ymin, xmax, ymax [,SRID])
```

This creates a bounding envelope with the given coordinates. It is stored as a POLYGON:

```
SELECT st_AsText (ST_MakeEnvelope (10, 20, 30, 40));
POLYGON ((10 20, 10 40, 30 40, 30 20, 10 20))

varchar ST_GeometryType (geometry)
```

This returns a string describing the given geometry

```
SELECT ST_GeometryType (ST_MakeEnvelope (10, 20, 30, 40, 4326));
polygin

integer ST_IsValid (geometry [,flags])
integer ST_IsValidReason (geometry [, flags])
```

The first returns 0 or 1 depending on whether the given geometry is valid, and the second returns a string giving the reason why it is not valid.

```
integer ST_SRID (geometry)
```

Returns the SRID specified in the given geometry.

```
SELECT ST_SRID (ST_Polygon ('SRID=4326; POLYGON ((1 1, 2 2, 2 10, -5 5, 1
1))'));
4326
```

```
double ST_XMin (geometry)
double ST_XMax (geometry)
double ST_YMin (geometry)
double ST_YMax (geometry)
double ST_ZMin (geometry)
double ST_ZMax (geometry)
```

These return the given value from the geometry.

```
double ST_3DLength (geometry)
double ST_3DPerimeter (geometry)
```

These return the total length of all of the lines in the geometry, coping with 3D values.

```
integer ST_Point_Inside_Circle (Point, CX, CY, Radius)
```

This returns 1 or 0 depending on whether the given point is inside the circle of radius Radius centred on (CX, CY).

```
integer ST_Point_Inside_Sphere (Point, CX, CY, CZ, Radius)
```

This returns 1 or 0 depending on whether the given point is inside the sphere of radius Radius centred on (CX, CY, CZ).

```
integer ST_Dimension (geometry)
```

This returns the actual dimension of the given geometry – eg a point returns 0, a line 1, a polygon 2, etc.

```
integer ST_CoordDim (geometry)
integer ST_NDims (geometry)
```

These return the actual dimension of the points contained in the geometry, eg a 2D point would return 2, a 3D point 3, etc.

```
integer ST_Entity (geometry)
integer ST_GeometryTypeCode (geometry)
```

These return the WKB code for the type of the geometry.

```
integer ST_IsEmpty (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is empty.

```
integer ST_IsSimple (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is simple.

```
integer ST_IsRing (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is a ring.

```
integer ST_IsClosed (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is closed.

```
integer ST_IsCollection (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is a geometry collection.

```
integer ST_IsRectangle (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is a rectangle.

```
integer ST_IsValid (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is valid.

```
integer ST_Is3D (geometry)
```

Returns 0 or 1 depending on whether or not the given geometry is 3 dimensional.

```
double ST_X (Point)
```

Returns the x-coordinate of the given point.

```
double ST_Y (Point)
```

Returns the y-coordinate of the given point.

```
double ST_Z (Point)
```

Returns the z-coordinate of the given point, or NULL if the point only has 2 dimensions.

```
integer ST_NumPoints (geometry)
integer ST_NPoints (geometry)
```

The number of points in the geometry.

```
integer ST_NumInteriorRings (polygon)
integer ST_NumInteriorRing (polygon)
```

The number of interior rings in the polygon. If it is passed a collection of polygons it only examines the first.

```
integer ST_NRings (geometry)
```

The total number of rings in the polygon. If it is passed a collection of polygons it finds the total number of rings in the collection.

```
integer ST_NumGeometries (geometry)
```

The total number of geometries in the geometry passed in.

```
double ST_Area (geometry)
```

Returns the total area of the geometry.

```
double ST_Length (geometry)
double ST_2DLength (geometry)
double ST_Perimeter (geometry)
double ST_2DPerimeter (geometry)
```

These all return the total length of the geometry, ignoring any 3D points.

```
double ST_CompactnessRatio (geometry)
```

This is a measure of how compact the geometry is, and is derived from the area and the perimeter.

```
geometry ST_MinimumDiameter (geometry)
```

This returns a line that is the shortest line across the geometry.

```
geometry ST_Affine (geometry, a, b [, c], d, e [, f], g, h [, i], xoff, yoff [, zoff])
```

Performs an affine transformation of the geometry: each point (x, y [, z]) becomes (a*x+b*y+c*z+xoff, d*x+e*y+f*z+yoff [, g*x+h*y+i*z+zoff]).

```
geometry ST_Translate (geometry, xoff, yoff [, zoff])
```

Performs a translation of the geometry: each point (x, y[, z]) becomes (x+xoff, y+yoff [,z+zoff]).

```
geometry ST_Scale (geometry, xf, yf [, zf])
```

Performs a scaling of the geometry: each point (x, y [, z]) becomes (x*xf, y*yf [,z*zf]).

```
geometry ST_TransScale (geometry, xoff, yoff [, zoff], xf, yf [,zf])
```

Performs a scale and translation of the geometry: each point (x, y[, z]) becomes (x*xf+xoff, y*yf+yoff [,z*zf+zoff]).

```
geometry ST_Rotate (geometry, theta [, x0, y0 [,z0]])
```

Rotates the geometry by the angle theta around the point (x0, y0 [,z0]), or the origin if no point is specified.

```
geometry ST_RotateX (geometry, theta)
```

Rotates the geometry by theta around the X axis.

```
geometry ST_RotateY (geometry, theta)
```

Rotates the geometry by theta around the Y axis.

```
geometry ST_RotateZ (geometry, theta)
```

Rotates the geometry by theta around the Z axis.

```
geometry ST_SnapToGrid (geometry [, x0, y0 [,z0]], gx, gy [,gz])
```

Snaps the points in the geometry to the nearest point in the grid centred on (x0, y0 [,z0]) with grid sizes (gx, gy [,gz]). If (x0, y0 [,z0]) is not specified it is centred on the origin.

```
geometry ST_PrecisionReducer (geometry, integer ndecimalplaces)
```

Reduces the precision of the geometry by snapping it to an appropriately sides grid.

```
geometry ST_Segmentize (geometry, length)
geometry ST_Densify (geometry, length)
```

Returns a modified geometry having no line segment longer than the given length.

```
geometry ST_RemoveRepeatedPoints (geometry)
```

Removes immediately repeated points in the given geometry.

```
geometry ST_Force_2D (geometry)
```

Makes the geometry 2D by ignoring all Z coordinates.

```
geometry ST_Force_3D (geometry)
geometry ST_Force_3DZ (geometry)
```

Makes the geometry 3D by adding a z coordinate of 0 to all points that only have two dimension.

```
geometry ST_Force_Collection (geometry)
```

Creates a GeometryCollection containing only the points, linestrings, and polygons in the original geometry.

```
geometry ST_CollectionExtract (geometry, integer type)
```

Returns only the geometries of the given type that are present in the geometry: 1 = POINT, 2 = LINESTRING, 3 = POLYGON.

```
geometry ST_Multi(geometry)
```

Returns the geometry as a MULTI* geometry. If the geometry is already a MULTI*, it is returned unchanged.

```
geometry ST_ForceRHR (polygon)
```

Forces the polygon to obey the right hand rule – all of its rings must be specified in anti-clockwise order.

```
geometry ST_Reverse (geometry)
```

Reverses all of the lines in the geometry.

```
geometry ST_FlipCoordinates (geometry)
```

Swaps the (x, y) values round for every point in the geometry.

```
geometry ST_Shift_Longitude (geometry)
```

Reads every point/vertex in every component of every feature in a geometry, and if the longitude (x) coordinate is <0, adds 360 to it.

```
geometry ST_SetSRID (geometry, integer SRID)
```

Sets the SRID of the geometry to the given value.

```
geometry ST_LinearRing (geometry)
```

Convert the linestring into a linear ring, or if it is not a linestring return it unchanged.

```
geometry ST_Line_Interpolate_Point (geometry, double fraction)
```

Returns a point interpolated along a line. Second argument is between 0 and 1, and represents the fraction of the total length of the linestring at which the point is located.

```
geometry ST_LineString (geometry)
```

Convert the linear ring into a linestring, or if it is not a linear ring return it unchanged.

```
geometry ST_LineFromMultiPoint (geometry)
```

Convert the multipoint into a linestring.

```
geometry ST_Normalize (geometry)
```

Normalize the geometry, ie convert it into its basic canonical form.

```
geometry ST_Boundary(geometry)
```

Returns the combined boundary of the geometry.

```
geometry ST_Centroid (geometry)
```

Returns the point that is the geometric centre of the geometry.

```
geometry ST_Envelope (geometry)
```

Returns the bounding envelope of the geometry.

```
geometry ST_Expand (geometry, xfactor [, yfactor])
```

Returns the envelope of the geometry expanded by the given factors. If yfactor is not given the both X and Y are expanded by xfactor.

```
geometry ST_ConvexHull(geometry)
```

Returns the convex hull of the geometry.

```
geometry ST_Buffer (geometry)
geometry ST_Buffer (geometry, double width, varchar args)
geometry ST_Buffer (geometry, double width, integer quad_segs)
geometry ST_OffsetCurve (geometry, double width)
geometry ST_OffsetCurve (geometry, double width, varchar args)
geometry ST_BufferAndOffsetCurve (geometry, double width, integer quad_segs,
varchar args)
```

Adds the given buffer to the geometry.

```
geometry ST_BoundingCircle (geometry[, integer quad_segs])
```

Returns the bounding circle of the geometry, with the given number of line seqments in each quarter. If quad_segs is not specified it defaults to 48.

```
geometry ST_PointN(geometry, integer N)
```

Returns the N’th point in the first linestring found in the geometry.

```
geometry ST_StartPoint (geometry)
```

Returns the first point in the geometry.

```
geometry ST_EndPoint (geometry)
```

Returns the last point in the geometry.

```
geometry ST_PointOnSurface (geometry)
```

Returns a point guaranteed to be on the surface of the geometry.

```
geometry ST_ExteriorRing (geometry)
```

Returns the exterior ring of the given polygon.

```
geometry ST_InteriorRingN (geometry, integer N)
```

Returns the N'th interior ring of the given polygon.

```
geometry ST_Holes (geometry)
```

Returns the geometry's holes as a geometrycollection.

```
geometry ST_RemoveHoles (geometry)
```

Returns the geometry without its holes.

```
geometry ST_ToMultiPoint (geometry)
```

Converts the geometry to a MULTIPOINT collection.

```
geometry ST_ToMultiLine (geometry)
```

Converts the geometry to a MULTILINESTRING collection.

```
geometry ST_ToMultiSegments (geometry)
```

Converts the geometry to a MULTILINESTRING collection, one per segment.

```
geometry ST_LocateAlong (geometry, double Fraction, double Distance)
```

Places points along the line segments composing the geometry at a distance of Fraction along the segment and at an offset distance of Distance. They are returned as a MULTIPOINT.

```
geometry ST_GeometryN(geometry, integer N)
```

Returns the Nth geometry in the collection.

```
geometry ST_UnaryUnion (geometry)
```

Returns the unary union of the geometry.

```
geometry ST_UnionCascaded (geometry)
```

Returns the union of the geometry, but using a faster algorithm in degenerate cases.

```
geometry ST_Simplify (geometry)
```

Simplifies the geometry.

```
geometry ST_SimplifyPreserveTopology (geometry)
```

Simplifies the geometry while preserving its topology.

```
geometry ST_LineMerge (geometry)
```

Returns a (set of) LineString(s) formed by sewing together the constituent line work of a MULTILINESTRING.

```
geometry ST_DelaunayTriangles (geometry [, double Tolerance [, integer
OnlyEdges]])
```

Returns the geometry's decomposition into Delaunay Triangles.

```
geometry ST_ExtractUniquePoints (geometry)
```

Returns a MULTIPOINT containing all of the unique points in the geometry.

```
geometry ST_Node (geometry)
```

Fully node a set of linestrings using the least possible number of nodes while preserving all of the input ones.

```
geometry ST_BuildArea (geometry)
geometry ST_BdPolyFromText (geometry)
geometry ST_BdMPolyFromText (geometry)
```

Returns a polygon from the given geometries.

```
double ST_Distance_Sphere (point A, point B)
```

Returns the distance between the two long-lat points on a spherical earth of radius 6370986 metres.

```
double ST_LineLocate_point (linestring A, point B)
```

Returns the proportion between 0 and 1 that determines the point that is the closest point on the given linestring to the given point.

```
double ST_MaxDistance (geometry A, geometry B)
```

Returns the distance between the farthest apart points on the two geometries.

```
double ST_Distance (geometry A, geometry B)
```

Returns the distance between the closest points on the two geometries.

```
integer ST_DWithin (geometry A, geometry B, double Distance)
```

Returns 1 if the two geometries are within ‘Distance’ of each other.

```
integer ST_DFullyWithin (geometry A, geometry B, double Distance)
```

Returns 1 if the two geometries are fully within ‘Distance’ of each other, else 0.

```
integer ST_Disjoint (geometry A, geometry B)
```

Returns 1 if the two geometries are disjoint, else 0.

```
integer ST_Intersects (geometry A, geometry B)
```

Returns 1 if the two geometries intersect, else 0.

```
integer ST_Touches (geometry A, geometry B)
```

Returns 1 if the two geometries touch, else 0.

```
integer ST_Crosses (geometry A, geometry B)
```

Returns 1 if the two geometries cross, else 0.

```
integer ST_Overlaps (geometry A, geometry B)
```

Returns 1 if the two geometries overlap, else 0.

```
integer ST_Contains (geometry A, geometry B)
```

Returns 1 if geometry A contains geometry B, else 0.

```
integer ST_Covers (geometry A, geometry B)
```

Returns 1 if geometry A covers geometry B, else 0.

```
integer ST_CoveredBy (geometry A, geometry B)
```

Returns 1 if geometry A is covered by geometry B, else 0.

```
integer ST_ContainsProperly (geometry A, geometry B)
```

Returns 1 if geometry A properly contains geometry B, else 0.

```
integer ST_Within (geometry A, geometry B)
```

Returns 1 if geometry A is completely within geometry B, else 0.

```
double ST _HausdorffDistance (geometry A, geometry B [, DensifyFrac])
```

The Hausdorff Distance between the two geometries.

```
integer ST_Equals (geometry A, geometry B)
```

Returns 1 if the two geometries are equivalent, else 0.

```
integer ST_OrderingEquals (geometry A, geometry B)
```

Returns 1 if the two geometries are equivalent and in the same order, else 0.

```
double ST _Azimuth (geometry A, geometry B)
```

Returns the azimuth of the segment defined by the given Point geometries, or NULL if the two points are coincident. Return value is in radians. Angle is computed clockwise from down-to-up: on the clock: 12=0; 3=PI/2; 6=PI; 9=3PI/2

```
integer ST_EnvIntersects (geometry A, geometry B)
integer ST_EnvelopesIntersects (geometry A, geometry B)
```

Returns 1 if the envelopes of the two geometries intersect.

```
integer ST_EqualsExact (geometry A, geometry B [, Tolerance])
```

Returns 1 if the two geometries are exactly equal to within the given tolerance, else 0.

```
varchar ST_Relate (geometry A, geometry B [, integer BoundaryNodeRule])
```

Returns a string describing the relationship pattern between the two geometries.

```
integer ST_RelatePattern (geometry A, geometry B, varchar Pattern)
```

Returns 1 if the two have the relationship given by the Pattern, else 0.

```
integer ST_LineCrossingDirection (geometry A, geometry B)
```

Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behaviour they exhibit.

```
varbinary ST_AddPoint (geometry LineString, geometry Point [, integer position])
```

Adds the point to the linestring at the given position, or at the end if no position is specified.

```
varbinary ST_SetPoint (geometry LineString, , integer position, geometry Point)
```

Sets the point at the given position in the LineString to the given Point.

```
varbinary ST_RemovePoint (geometry LineString, , integer position)
```

Removes the point at the given position in the LineString.

```
varbinary ST_Intersection (geometry A, geometry B)
```

The intersection of the two geometries.

```
varbinary ST_Union(geometry A, geometry B)
```

The union of the two geometries.

```
varbinary ST_Difference (geometry A, geometry B)
```

The parts of geometry A that do not intersect with geometry B.

```
varbinary ST_SymDifference (geometry A, geometry B)
varbinary ST_SymmetricDiff (geometry A, geometry B)
```

Those parts of the two geometries that do not intersect.

```
varbinary ST_SharedPaths (geometry A, geometry B)
```

For two LINESTRING/MULTILINESTRING geometries, this is the paths that are shared between them both.

```
varbinary ST_Snap (geometry A, geometry B, double Tolerance)
```

Snaps nearby points in the geometry A to the point, within Tolerance, in geometry B.

```
varbinary ST_ClosestPoint (geometry A, geometry B)
```

The point in A that is closest to B.

```
varbinary ST_ShortestLine (geometry A, geometry B)
```

The shortest line between A and B.

```
varbinary ST_LongestLine (geometry A, geometry B)
```

The longest line between A and B.

```
varbinary ST_MakePolygon (linearring Shell [, geometry Holes [, SRID]])
```

Creates a polygon by adding the holes (specified as a LinearRing or a geometrycollection of LinearRing) to the given shell.

```
varbinary ST_Collect (geometry A, geometry B)
```

Creates either a single multi* or a GeometryCollection containing the two geometries.

```
integer ST_RelateMatch (varchar Match, varchar Pattern)
```

Returns 1 if Match implies Pattern, else 0.