Geomatics
Guidance Note Number 7, part 4

EPSG Geodetic Parameter Relational Database – Developers Guide

Revision history:

Version	Date	Amendments
1.0	October 2004	First release of this document, GN7 part 1. Former GN7 now released as part 2.
1.1	November 2004	Minor editorial corrections to text. Annex E SQL scripts updated.
2.0	April 2006	Amendment to deprecation rules. Updated references to EPSG.
2.1	February 2007	Deprecation rules updated. Policy on code uniqueness clarified. Additional information on user update utility added. Minor editorial corrections to text.
3	July 2007	Use of data conditions amended. Annex F added.
4	April 2009	Major revision to include web registry. Access- and SQL-specific discussion moved to new GN7 part 4.

CONTENTS

(There is no annex D in this document)

PREFACE

The EPSG Geodetic Parameter Dataset, abbreviated to the **EPSG Dataset**, is a repository of parameters required to:

- define a *coordinate reference system* (CRS), which ensures that coordinates describe position unambiguously.
- define transformations and conversions that allow coordinates to be changed from one CRS to another CRS. Transformations and conversions are collectively called *coordinate operations*.

The EPSG Dataset is maintained by the OGP Surveying and Positioning Committee's Geodetic Subcommittee. It conforms to ISO 19111 – *Spatial referencing by coordinates*. It is distributed in three ways:

- the **EPSG Registry**, in full the *EPSG Geodetic Parameter Registry*, a web-based delivery platform in which the data is held in GML using the CRS entities described in ISO 19136.
- the **EPSG Database**, in full *the EPSG Geodetic Parameter Database*, a relational database structure where the entities which form the components of CRSs and coordinate operations are in separate tables, distributed as an MS Access database;
- in a relational data model as **SQL scripts** which enable a user to create an Oracle, MySQL, PostgreSQL or other relational database and populate that database with the EPSG Dataset;

OGP Surveying and Positioning Guidance Note 7 is a multi-part document for users of the EPSG Dataset.

- *Part 0, Quick Start Guide*, gives a basic overview of the Dataset and its use.

- *Part 1, Using the Dataset*, sets out detailed information about the Dataset and its content, maintenance and terms of use.

- *Part 2, Formulas*, provides a detailed explanation of formulas necessary for executing coordinate conversions and transformations using the coordinate operation methods supported in the EPSG dataset. Geodetic parameters in the Dataset are consistent with these formulas.

- *Part 3, Registry Developer Guide*, is primarily intended to assist computer application developers who wish to use the API of the Registry to query and retrieve entities and attributes from the dataset.

- *Part 4, Database Developer Guide*, (this document), is primarily intended to assist computer application developers who wish to use the Database or its relational data model to query and retrieve entities and attributes from the dataset.

The complete text may be found at http://www.epsg.org/guides/index.html. The terms of use of the dataset are also available at http://www.epsg.org/CurrentDB.html.

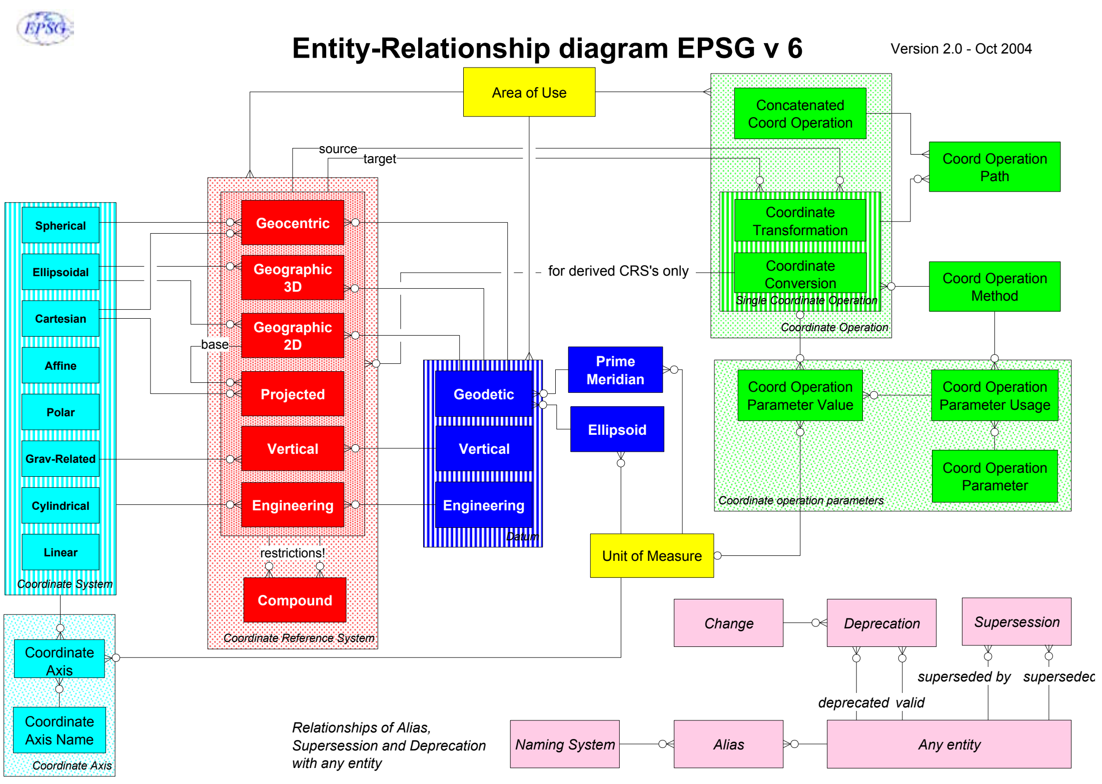
In addition to these documents, the Registry user interface contains online help and the Database user interface includes context-sensitive help accessed by left-clicking on any label

This Part 4 of the multipart Guidance Note is primarily intended to assist computer application developers in using the EPSG geodetic parameter relational database and SQL scripts. It may also be useful to other users of the data. Readers are recommended to have read Part 1 of the guidance note before this part.

1 EPSG DATASET RELATIONAL IMPLEMENTATION

1.1 Overview of relational data structure

An overview of the relational table structure is shown in the diagram below:



There has been no change to the overall structure between dataset versions 6 and 7.

The 21 tables forming the relational dataset are:

Access table name	SQL script table name	Table content	(see note)
Alias	epsg_alias	Aliases for all object types.	S
Area	epsg_area	Text and minimum bounding geographic rectangle descriptions for area of use, referenced by datum, CRS and coordinate operation records.	D
Change	epsg_change	A record of change requests received by EPSG and summary of changes made to records.	
Coordinate Axis	epsg_coordinateaxis	Coordinate axis abbreviation, orientation and order. Also links coordinate axis name with coordinate system.	I
Coordinate Axis Name	epsg_coordinateaxisname	Coordinate axis name and description.	D
Coordinate Reference System	epsg_coordinatereferencesystem	CRS name, type and scope. Also the base CRS for projCRSs and constituent single CRSs for compound CRSs.	D
Coordinate System	epsg_coordinatesystem	Coordinate system name, type and dimension. Referenced from CRS table.	D
Coordinate_Operation	epsg_coordoperation	Map projection, transformation and concatenated operation name, type and scope. Transformation version and accuracy. Codes for transformation source and target CRS.	D

Access table name	**SQL script table name**	**Table content**	(see note)
Coordinate_Operation Method	epsg_coordoperationmethod	Transformation and conversion method name, formula, example and reversibility.	D
Coordinate_Operation Parameter	epsg_coordoperationparam	Transformation and conversion parameter name and description.	D
Coordinate_Operation Parameter Usage	epsg_coordoperationparamusage	Transformation and conversion parameter order and reversibility. Also links coordinate operation parameters with coordinate operation methods.	I
Coordinate_Operation Parameter Value	epsg_coordoperationparamvalue	Transformation and conversion (map projection) parameter values.	I
Coordinate_Operation Path	epsg_coordoperationpath	Concatenated operation step and step sequence information.	I
Datum	epsg_datum	Datum name, type, scope, epoch and origin information. Referenced from CRS table.	D
Deprecation	epsg_deprecation	Information for tracking deprecated records and their replacement (if any).	
Ellipsoid	epsg_ellipsoid	Ellipsoid name and dimensions. Referenced from datum table.	D
Naming System	epsg_namingsystem	Alias naming system names.	D
Prime Meridian	epsg_primemeridian	Prime meridian name, Greenwich longitude.	D
Supersession	epsg_supersession	Information for tracking valid records which have been superseded by more recent data.	
Unit of Measure	epsg_unitofmeasure	Length, angle and scaling unit name and conversion factor. Referenced by ellipsoid and coordinate operation parameters, coordinate axis and prime meridian.	D
Version History	epsg_versionhistory	Dataset version release record.	

Notes:
1. Where the table content column has a D to its right, the table content includes primary object data such as name and code. These tables additionally include metadata to describe information origin. The table includes a field indicating whether records are deprecated. Valid records have deprecation = no = 0. For records where deprecation = yes = 1, the records are invalid, i.e. have been deprecated (flagged as invalid but remain in the dataset). Deprecated records should only be used when documenting or reversing use of the record that was made before it was declared invalid.
2. Where the table content column has an I to its right, the table is an intersection table within the relational data model. It includes essential data fields used by one or more of the tables to which it is joined.
3. Where the table content column has an S to its right, the table content is entirely secondary data which supplements the primary data.
4. Other tables contain supplementary information to facilitate data management or automated computer access to the primary data.

Field names within tables are consistent between the Access and SQL versions except for:

Access table name	**Access field name**	**SQL script table name**	**SQL script field name**
Alias	ALIAS CODE	epsg_alias	alias_code
Change	COMMENT	epsg_change	change_comment
Coordinate Axis	ORDER	epsg_coordinateaxis	coord_axis_order

Access to SQL data type conversions are:

Access type	SQL type	Comment
Yes/No	SMALLINT	1 = Yes / 0 = No
Date/Time	DATE	
Memo	VARCHAR(4000)	This will not work with MySQL, which needs TEXT
Ole object	BLOB	(This type declaration not currently used).
Text	VARCHAR(length)	
Binary	LONG	(This type declaration not currently used).
Double	DOUBLE PRECISION	
Long integer	INTEGER	
Integer	SMALLINT	
Byte	LONG	

The strategy employed for the EPSG dataset is to commit to retaining backward compatibility with the above SQL table and field structure, data type and field names, which will remain unchanged. However in order to facilitate product improvement OGP reserves the right to supplement these by additional fields or tables when necessary. If there is perceived to be a need to include additional capability, this will be done such that the development will retain backward compatibility with the above structure. Note however that new fields might be inserted within existing tables: programmers should never rely on the order of fields in a table, but only on the names of the fields.

1.2 SQL scripts describing the data model and dataset content

From v3.0 the EPSG dataset has been implemented in Microsoft Access, which in addition to being the data repository has acted both as a data entry tool for the OGP Geodesy Subcommittee (until replaced for these purposes by the registry from version 7) and as a searching, browsing and reporting tool for some users.

A further use that the Geodesy Subcommittee hoped the database might be put to was the querying of the dataset by software programs. However, the reliance on a Microsoft Windows database server and the lack of cross platform support for such queries meant that such uses have not been developed. To address this issue, from version 6.5.3 onwards, three additional items are being provided. These are:

1) A Data Description Language (DDL) file called epsg_v[version].mdb_Tables.sql which contains the SQL CREATE statements necessary to create a database equivalent to the Access on an SQL server.
2) A Data Manipulation Language (DML) file called epsg_v[version].mdb_Data.sql which contains the SQL INSERT statements necessary to populate a database created using the EPSG*Tables.SQL DDL file with the current contents of the dataset.
3) A second DDL file epsg_v[version].mdb_fKeys.sql which contains the SQL ALTER statements necessary to enforce Foreign Key Constraints on the tables. Note that this should either not be run (if users are not intending to add supplementary data, there is no need to run this), or that it should be run *after* the tables have been populated using the DML file.

Platforms that have been tested are listed below.

1.2.1 Oracle

The import of the DDL & DML scripts have been tested on an Oracle version 9i release 2 i386 server.

The Oracle implementation is not in any way non-standard, the only constraint, already addressed in the table name translations, is that Oracle table names need be 30 characters or less.

Refer to Oracle documentation for loading instructions.

1.2.2 PostgreSQL

The import of the DDL & DML scripts have been tested on an PostgreSQL version 7.3.4. From the command line, execute the following (make sure your default user has sufficient privileges):

```
afaichney@localhost:~> psql
afaichney=# create database epsg_v653
CREATE DATABASE
afaichney=# \c epsg_v653
You are now connected to database epsg_v653.
epsg_v653=# \i EPSG_v653.mdb_Tables.sql
       <outputs removed>
epsg_v653=# \i EPSG_v653.mdb_Data.sql
       <outputs removed>
       <Next step is optional>
epsg_v653=# \i EPSG_v653.mdb_fKeys.sql
       <outputs removed>
```

1.2.3 MySQL

MySQL has no support for VARCHARs longer than 254 characters, and so implementations of the database on MySQL will need to ensure that the non-standard TEXT type is used to replace long VARCHARs. Also the Foreign Key Constraints in the DDL will fail and should be removed.

With these modifications, import of the scripts into MySQL has been tested with MySQL 4.0.12. From the command line, execute the following:

```
    afaichney@localhost:~> mysql -u root -p
    mysql> create database epsg_v653;
    Query OK, 1 row affected (0.00 sec)
    mysql> \u epsg_v653
    Database changed
    mysql> \. EPSG_v653.mdb_Tables.sql
           <outputs removed>
    mysql> \. EPSG_v653.mdb_Data.sql
           <outputs removed>
           <Next step optional>
    mysql> \. EPSG_v653.mdb_fKeys.sql
<outputs removed>
```

2 SEARCHING THE RELATIONAL DATASET

Information relevant to both the registry and relational database implementations is given in Guidance Note 7 part 1 (GN7-1) and should be read in conjunction with this section.

2.1 MS Access forms and reports

Included within the Access version of the dataset are several forms and reports, underpinned by some queries and macros. These are for human interaction with the dataset. There are four groups:

- ***browse forms*** which allow users to review individual records by selected topic;
- ***reports*** which give to users reports on records by selected topic;
- ***edit/add forms*** which allow for maintenance of the dataset content;
- forms which allow users to navigate the above database forms and reports.

Further detail of these forms and reports is given in annex A.

2.2 MS Access queries

The Access database includes several queries which underpin the forms and reports. These have names beginning with "*qry –* ".

Additionally, from version 6.6, there are queries which retrieve related data. These have names beginning with "*qry_epsg_gn7_1_*". Their use is described in the sections below and in annex E. The sample scripts in annex E are divided into three sections:

1. Data discovery – scripts to identify records or tabulate related data.
2. Coordinate Reference System description – scripts to retrieve data essential to describing a coordinate reference system. This includes parameters necessary for geographical to/from grid coordinate conversions.
3. Coordinate Transformation description – scripts to retrieve data essential to describing a coordinate transformation. This includes parameters necessary for the execution of datum transformations

2.3 Valid data

As described in GN7-1section 5.5 and annex B, the dataset includes records which are invalid. Record validity is indicated by the setting in the DEPRECATED field. In general, searches for data should exclude the invalid records. This may be accomplished by searching for data where the value of the DEPRECATED field = "false" (or "No" or "0"). Most of the example SQL scripts in annex E include this constraint.

There may be occasions when there is a requirement to replicate data used previously, regardless of its current validity. On these rare occasions the search constraint criteria should omit the value of DEPRECATED. In Annex E to this document, example 1.00 gives SQL script to identify the version and date of a dataset.

2.4 Searching by Name

See Guidance note 7 Part 1 section 6.2. An example SQL query to search by name or alias is given in annex E.1.01.

2.5 Searching by Area of Use

See Guidance note 7 Part 1 section 6.3.

2.6 Coordinate Reference Systems

See also Guidance note 7 Part 1 section 6.4.

The primary information for all subtypes of coordinate reference system is stored within the *table "Coordinate Reference System*" (Access) or *epsg_coordinatereferencesystem* (SQL). The field COORD_REF_SYS_KIND indicates the type of CRS. The CRS table includes fields to cross-reference supporting information included within the coordinate system, datum and area tables (COORD_SYS_CODE, DATUM_CODE and AREA_OF_USE_CODE respectively).

A SQL script to identify a CRS code from name or alias is given in annex E example 1.01. SQL scripts to identify CRS type and validity from CRS code are given in annex E examples 1.04 and 1.05.

2.6.1 Geographic and geocentric CRSs
SQL scripts to extract from the dataset the essential data for describing geographic and geocentric CRSs, including datum and ellipsoid information, are given in annex E examples 2.1 and 2.4. For a query which tabulates these related geodetic CRS records see annex E example 1.06.

2.6.2 Projected and other derived CRSs
The CRS table includes a field named SOURCE_GEOGCRS_CODE which contains the code of the base CRS for projected (and other derived) coordinate reference systems. The field is only populated when the CRS is a derived (usually projected) CRS. When this field is populated, so too will be the field PROJECTION_CONV_CODE which cross-references a map projection (conversion) included within the coordinate operation table. Projected CRSs inherit geodetic datum from their base geographic CRS: the DATUM_CODE field is not used for projected CRS records. A record for a projected or other derived CRS cannot exist until there is a record for the base CRS in the CRS table and a record for the conversion in the coordinate operation table.

SQL scripts to extract from the dataset the essential data for describing projected CRSs, including datum, ellipsoid and projection information, are given in annex E examples 2.2 and 2.4.

2.6.3 Vertical and engineering CRSs
SQL scripts to extract from the dataset the essential data for describing a vertical CRS or an engineering CRS are given in annex E examples 2.3 and 2.4.

2.6.4 Compound CRSs
In the EPSG Dataset, a compound CRS is a geographic or projected CRS combined with a vertical CRS. The CRS table includes two fields named CMPD_HORIZCRS_CODE and CMPD_VERTCRS_CODE. These give the codes of the two CRSs which together form the compound CRS. A record for a compound CRS cannot exist until the record for each of the component CRSs is in the CRS table. See annex E example 2.5 for an SQL query to retrieve identifying details (i.e. name and code) for each of the component single CRSs, given the code of the compound CRS. The details for each of the component single CRSs may then be found and described as above.

2.6.5 Coordinate Systems
Coordinate system information is given in three tables: coordinate system, coordinate axis and coordinate axis name. A cryptic description of the CS is given in the coordinate system table's COORD_SYS_NAME field. The CS table also contains the type and dimension (i.e. number of axes) of the CS, together with CS metadata.

The coordinate axis name table contains axis name and description. The coordinate axis table contains axis orientation, axis abbreviation and axis order information. If any of these axis attributes are changed, a new CS is defined which combines with a datum to form a new CRS. Axes for a CS should be presented sorted by ascending value of the field ORDER (in Access, coord_axis_order in SQL).

See annex E example 2.4 for an SQL script to return coordinate system axis information from CRS code. This applies to any geographic, geocentric, projected, vertical or engineering CRS. Annex E example 1.07 tabulates projected CRSs with common datum but differing coordinate system.

2.6.6 Datums
The primary information for all subtypes of datum is stored within the table *Datum* (Access) or *epsg_datum* (SQL). The field DATUM_TYPE indicates the type of datum. The datum table includes fields to cross-reference supporting information included within the area, ellipsoid and prime meridian tables (AREA_OF_USE_CODE, ELLIPSOID_CODE and PRIME_MERIDIAN_CODE respectively). The ellipsoid and prime meridian fields are populated only if the datum type is geodetic.

2.6.7 Ellipsoids
See also GN7-1 section 5.11.4 and 6.4.3

The primary information for ellipsoids is stored within the table *Ellipsoid* (Access) or *epsg_ellipsoid* (SQL). The ellipsoid table includes a UOM_CODE field to cross-reference supporting information included within the unit of measure table.

2.6.8 Prime meridians

See also GN7-1 section 6.4.4

The primary information for prime meridians is stored within the table "Prime meridian" (Access) or epsg_primemeridian (SQL). The Prime meridian table includes a field UOM_CODE to cross-reference supporting information included within the unit of measure table. See section 5.5 above for comment upon the unit where the value of GREENWICH_LONGITUDE field is in degrees.

2.7 Coordinate Operations

See also GN7-1 sections 5.11 and 6.5.

The information for all subtypes of coordinate operation (transformation, conversion including map projection and concatenated operation) is stored in six tables. The primary information for all subtypes is stored within the table *Coordinate_Operation* (Access) or *epsg_coordoperation* (SQL). The field COORD_OP_TYPE indicates the type of coordinate operation. The coordinate operation table includes fields to cross-reference supporting information included within the area and coordinate operation method tables.

The coordinate operation table also includes fields which reference a transformation's source and target CRSs (SOURCE_CRS_CODE and TARGET_CRS_CODE respectively). Neither of these fields is used for map projections; for map projections the base CRS information is included within the CRS table.

The coordinate operation table also includes two fields UOM_CODE_SOURCE_COORD_DIFF and UOM_CODE_TARGET_COORD_DIFF which cross reference to the unit of measure table. These fields are only used for polynomial transformation methods and indicate the unit in which evaluation point coordinates are required to be in for application within the method formula. These fields should not be confused with those that indicate the unit in which parameter values are tabulated, discussed below.

Four additional tables hold supplementary but critical information for single coordinate operations (i.e. transformations and conversions, including map projections). These are described in section 2.7.1 below. A further table holds information for concatenated operations.

Scripts to identify the type of coordinate operation from coordinate operation code and to list all transformations to or from a CRS are given in annex E.1.08 and E.1.09. Scripts for accessing the essential information for describing a map projection are given in annex E.2.2(ii) and for describing a transformation in annex E.3.

2.7.1 Transformation and conversion methods and parameters

Each transformation and conversion is related to a coordinate operation method through the coordinate operation table's COORD_OP_METHOD_CODE field. Primary information for transformation and conversion methods is held in the table "*Coordinate_Operation Method*" (Access) or *epsg_coordoperationmethod* (SQL).

Information on coordinate operation parameters used by a particular method is held in the intersection table "*Coordinate_Operation Parameter Usage*" (Access) or *epsg_coordoperationparamusage* (SQL). This table has a dual key of COORD_OP_METHOD_CODE and PARAMETER_CODE. The record for each combination of method and parameter also includes a field SORT_ORDER.

The name of each parameter, together with a description, is held in the table "*Coordinate_Operation Parameter*" (Access) or *epsg_coordoperationparam* (SQL).

Transformation and conversion parameter values

The values for coordinate operation parameters are held in the intersection table "*Coordinate_Operation Parameter Value*" (Access) or *epsg_coordoperationparamvalue* (SQL). In general these are in the field PARAMETER_VALUE. The coordinate operation parameter value table includes a field UOM_CODE to cross-reference supporting information included within the unit of measure table. See GN7-1 section 5.10 for comment upon the unit for angle values given in sexagesimal degree representation.

Parameter values for methods which use gridded data files

In the relational implementation, for any one record in the parameter value table, either the PARAMETER_VALUE or the PARAM_VALUE_FILE_REF field, but not both, will be populated.

ANNEX A - MS ACCESS DATABASE FORMS AND REPORTS

Included within the Access version of the dataset are several forms and reports, underpinned by some queries and macros. These are for human interaction with the dataset. There are four groups:

- browse forms which allow users to navigate the database forms and reports;
- browse forms which allow users to review individual records by selected topic;
- reports which give to users reports on records by selected topic;
- edit/add forms which allow for maintenance of the dataset content.

Database Forms

There are a total of 31 primary forms in the current version of the database. Within the forms, standard Access *filtering* functions can be used to limit a selection whilst standard Access *sorting* functions can be used to sort data. See the comments under the reports section below for filtering on the area of use field. To summarise, the forms provided in this version of the data base are as follows.

- four "Welcome Forms" to assist users in navigating the database's forms and reports.
- a series of twelve forms that allow browsing of specific data types.
- the browse forms offer options to go to a further series of fifteen forms allowing editing and addition of data.
- supporting the above are numerous subforms that provide integral components for the 31 primary forms, including one subform for user provision of a Company Logo.
- a form to facilitate the importation into Access of an Oracle script file of the dataset content. This is described further in Annex C of this document.

Help "Pop-ups" on Database Forms

Pop-up hot-key **"help"** subforms are currently implemented on all Edit/Add and Browse forms. To utilise this feature, click on a field caption (*e.g.*, the text "*Information Source*") and an appropriate help box will appear. This feature addresses comments on earlier versions to make data entry more intuitive for those unfamiliar with the database design.

Database "Company Logo" Implementation

There is provision for insertion of a user/company logo on forms and reports. To make use of this facility, a user should close the opening Welcome Form and open the first form listed, "1 - Company Logo". Substitute your company logo (sized to fit the subform) for the EPSG dataset logo currently there. Once this is done, the primary welcome form (and others) will open showing EPSG dataset logo on the left and your company logo on the right. If your logo is sized larger than the current company logo subform, part of your logo will be cut off when displayed. The same process should be applied to the first report listed, "1 - Company Logo".

Database Reports

The 54 reports (28 on valid data plus 26 to review deprecated records) include the ability to search the database and obtain reports based on user entry.

- Detailed reports based on an input area of use, name, or alias are available for various coordinate reference systems (geographic, projected, engineering, vertical and compound) and coordinate operations (map projections, transformations and concatenated operations). In detailed geographic coordinate reference system reports, most associated datum information is included.
- In addition, summary reports are available for most coordinate reference system types and for most transformations, with the same search criteria.
- Reports for reference purposes can also be run for all ellipsoids, angle units, length units, scale units, transformation parameters and prime meridians contained in the database. The report on transformation methods includes their formulae; these are given with improved clarity due to the use of multiple fonts in Guidance Note 7 part 2.
- Reports also review the database change requests that have been implemented in this version as well as any change requests that remain outstanding.

Each "area search" is actually a text search that runs on the *area of use*, *name* and *code#* fields in the relevant tables or queries, as well as on any associated *alias* fields in the alias table. Searches can be made by country name (ISO English spelling) and in the case of the projected coordinate reference system report in addition by US, Canada or Australia state/province/territory name, US state 2-character postal code abbreviation, or county name. In all these searches, MS Access requires an exact text string match. The area fields usually contain several country names and/or regional descriptions and a search on one country name will therefore generally fail. Users are recommended to include their search area text string within two asterisks as wild-card characters; for example to search for Oman ent**er *o**man*. This will retrieve all records that include Oman within the area of use or name fields; however it will also retrieve data for Romania. To produce a report with all possible contents simply enter a single wildca**rd (*).

Reports are dependent upon printer drivers in use. Users may need to modify printer setup the first time they use the report feature. For screen viewing, the reports work best at 75 % viewing size (user may set in MS Access) using large fonts (set in Control Panel/Display).

Database Queries

In the Access database there are a number of queries supporting the above forms and reports.

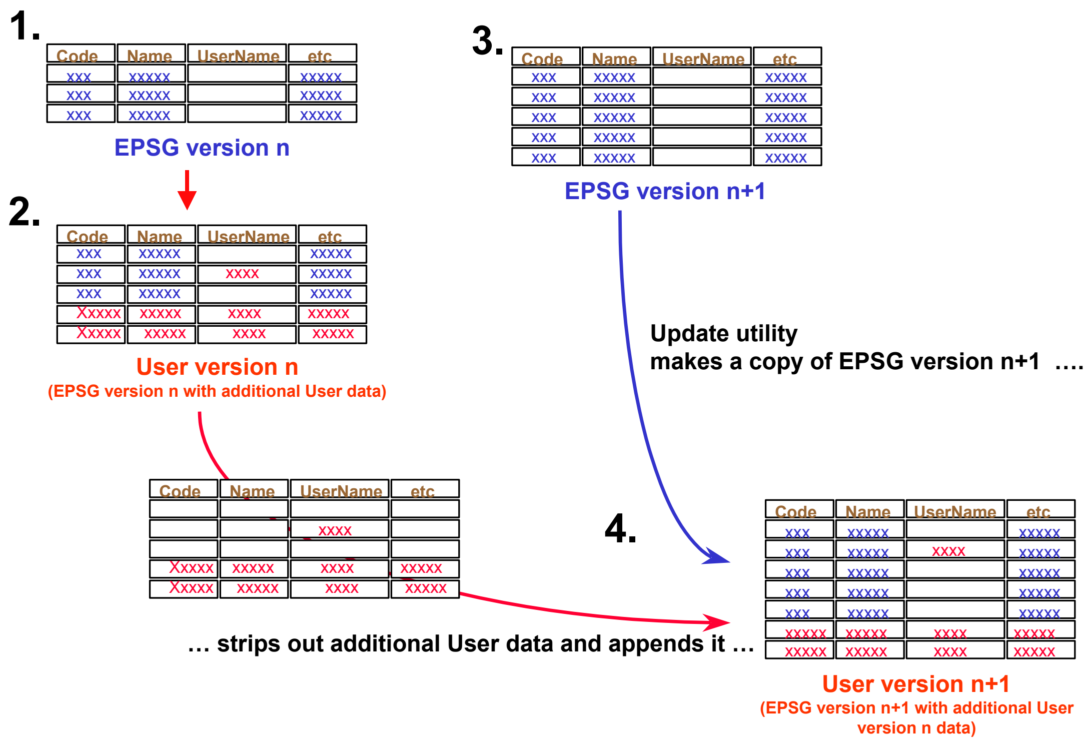
- sixteen queries are used to provide appropriate inputs for various reports and forms associated with datums, coordinate reference systems and coordinate operations (map projections, transformations, and concatenated operations). [Other control queries are incorporated into the specific reports and forms needing them].
- one make-table query is used to generate new coordinate operation (map projection and transformation) values in the edit/add coordinate transformation form.
- one query provides latest database version information for input into various form and report headers.

The Access database also includes five queries for monitoring the length of fields with memo type declaration (limited to 4000 characters in the SQL export).

The database additionally includes a number of queries to demonstrate automated use of the EPSG dataset. These are discussed in section 2 and annex E of this document.

ANNEX B – MS ACCESS USER UPDATE UTILITY

OGP Geodesy Subcommittee makes available a utility for managing updates of version 6.x datasets which hold additional user data. The utility will copy records which users have added to an earlier EPSG v6.x Access database from that database into the new EPSGv6.xx database, creating a version of the 6.xx database supplemented with the user's own records that had been added to v6.x. It is only available for the Access version of the dataset. This is an executable suite designed by Simon Dewing for the OGP Geodesy Subcommittee. It must be installed on a user's computer prior to running the utility. The UpdateEPSG utility for EPSG v6 databases is available for downloading from the EPSG dataset pages of the OGP web site.


1.
Code Name UserName etc
EPSG version n
2.
User version n
(EPSG version n with additional User data)
3.
EPSG version n+1
Update utility
makes a copy of EPSG version n+1 ....
4.
… strips out additional User data and appends it …
User version n+1
(EPSG version n+1 with additional User version n data)

The utility will:

- Expect that all user records have codes outside the EPSG dataset reserved range of 0-32767. This includes user change records which should have real number codes (Change ID) greater than 32767.0.
- Recognise that users may add user names, user aliases and user abbreviations to EPSG dataet records, as well as other aliases based on user-defined Naming Systems.
- Expect that all user records have "Data Source" field not equal to "EPSG" or "OGP".
- Prompt for the filenames and paths of input user-extended v6.x EPSG-format database and the new EPSG database (v6.xx) as well as for the newly created output user-extended database.
- Write a v6.xx-extended output database under the user-selected filename:
- This version of EPSG database in its entirety.
- A copy of records from the input user database v6.x with codes outside the EPSG dataset code range.
- Copy user abbreviations and user aliases found in the input user database v6.x to the equivalent records in the output database.
- Write a report listing the number of user records written to each table of the new database.
- The current version of the utility does not copy any user logo subreport or subform to the output database. They must be added to the output database after running the utility as described previously.

There have been several minor changes in the EPSG database structure since v6.2 was released. The User Update Utility currently has several known limitations, all of which have simple workarounds. These relate to:

- Incorrect Code Assignment in three tables with previously Auto Numbering Codes
- Codes table (present in earlier EPSG v6,x dataset releases)
- Extra Fields problem specific to EPSG_v6.9 (as originally posted to website)
- Moving User Records in Version History table

Detailed procedures for workarounds to each of these issues are given in the separate file, epsg-update-help.pdf, downloadable from the EPSG website along with the epsg-update-exe.zip executable.

Any changes to EPSG dataset records other than the addition of user-assigned names, aliases or abbreviations will not be resolved by the utility. The utility will not resolve duplications between user data and new EPSG dataset data.

ANNEX C – MS ACCESS REGISTRY IMPORT UTILITY

The data release cycle for the EPSG Dataset is described in Guidance Note 7 part 1. In summary, Access and SQL versions of the Dataset are made available only for full releases, nominally about twice per year. Between these full releases additional interim releases of the Dataset may be made in the Registry. This annex describes how users can create Access versions of these interim releases.

Registered Users of the on-line Registry have access to the registry export facility. There are options to export the dataset as GML or Oracle scripts. From v7.1 (May 2009) the published Access database includes a facility to import the Oracle script into Access. The steps of the process are:

- Make a copy of the current Access database.
- Delete all EPSG records from this database.
- Download the Oracle script from the Registry.
- Import the Oracle script into the Access database.

Delete EPSG Records

1. Open the copy of the MS Access database;
2. Open the form 'Import EPSG Oracle SQL';
3. Click 'Delete EPSG Records'. This will delete all records (data) from the database, but keeps the database tables, queries, forms, etc.

Download Oracle script from the on-line Registry

1. Log in to the Registry; (this requires having a registry user account. Prompts to create an account are given on the Registry home page. Details of how to create an account are given in Guidance Note 7 part 3.
2. Follow the 'export registry' link;
3. Choose the 'Export Oracle SQL scripts' link;
4. Save the zip file;
5. Extract 'Oracle_Data_Script.sql' to in the same folder as the MSAccess database to be populated.

Import EPSG Oracle SQL

1. Open the MS Access database to be populated;
2. Close the Welcome screen;
3. Open the form 'Import EPSG Oracle SQL';
4. Click 'Import EPSG Oracle SQL' in the top menu (importing status is displayed in the status bar).

Because of problems experienced by users of Access 2007, we have had to disable the automatic display of dataset version and data. The version displayed will be that copied at the tart of this process. To see the actual version of the Dataset, use the Dataset Version History button on the Forms selection screen and scroll to the last entry.

The displayed version and date can be amended by those confident in using design mode for database forms and reports. The six that need editing are:

- subform browse Version History MainForms
- subform browse Version History
- subform edit/add Version History
- subrpt footer
- subrpt page header logo
- subrpt report header logo

ANNEX D

(This page is intentionally left blank)

ANNEX E – SQL SCRIPTS FOR EXTRACTING DATA

The sample scripts in this annex are divided into three sections:

E.1. Data discovery – scripts to identify record metadata.
E.2. Coordinate Reference System description.
E.3. Coordinate Transformation description.

The "description" examples also provide the information necessary to execute a geographical to geocentric coordinate conversion, a geographical to grid coordinate conversion, or a datum transformation. They return only critical data; no metadata is included.

These simple examples have been constructed to illustrate the dataset content and its extraction. They should not be taken as fulfilling all extraction requirements, for which more extensive queries utilising several of the samples may be appropriate. The examples do not trap erroneous input.

The scripts have been tested against a MySQL version 4.0.18 database. Equivalent scripts are also included within the MS Access database; the Access database query names are given with each example below. The example results are consistent with the v6.6 dataset of October 2004.

E.1 Data Discovery

Example	Function returns	Input argument
E.1.00	version and date of a dataset.	
E.1.01	name, alias and code.	user "name" (name or alias in dataset)
E.1.02	metadata (remarks, information source, data source, change table record id(s) and revision date).	code
E.1.03	coordinate operation code, name, version and type.	area "name" (area of use in dataset)
E.1.04	CRS type.	CRS code
E.1.05	CRS type, validity and, if deprecated, the reason for deprecation and code of replacement (if any).	CRS code
E.1.06	geographic 2D CRS code	geocentric or geographic 3D CRS code
E.1.06a	tabulation of geographic and geocentric CRSs referenced to the same geodetic datum.	
E.1.07	tabulation of projected CRSs which share datum but have differing CS.	
E.1.08	retrieve Coordinate Operation code and type	code for a non-reversible transformation
E.1.09	tabulation of reciprocal non-reversible transformations	code for non-reversible transformation
E.1.10	list of transformations and concatenated operations related to CRS.	CRS code

E.1.00 SQL script to retrieve dataset version and date
(equivalent Access query: *qry Version History*)

This is done in two steps:

(i) Query to find the latest release date.

```
SELECT MAX(epsg_versionhistory.version_date) AS 'Date'
FROM epsg_versionhistory;
```

(ii) Query which uses the date found above to return the dataset's release date and version number.

```
SELECT epsg_versionhistory.version_date AS 'Date', epsg_versionhistory.version_number AS 'Dataset
Version'
FROM epsg_versionhistory
WHERE (epsg_versionhistory.version_date = 'yyyy-mm-dd');
```

E.1.01 SQL script to retrieve name, alias and code given user "name"
(Access query *qry_epsg_gn7_1_e101_nameAlias_crs*)

This example uses the coordinate reference system table. Similar scripts can be applied to other primary tables.

In MySQL, the process is split into two separate queries: (a) for searching for CRS name and (b) for searching for alias name.

(a). This query is used to search for a match between user "name" and dataset CRS name:

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_name, epsg_alias.alias,
epsg_coordinatereferencesystem.coord_ref_sys_code
FROM epsg_coordinatereferencesystem
LEFT JOIN epsg_alias ON (epsg_coordinatereferencesystem.coord_ref_sys_code =
epsg_alias.object_code)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_name LIKE 'abcd') AND
(epsg_coordinatereferencesystem.deprecated = 0))
ORDER BY epsg_alias.alias;
```

(b). This query is used to search for match between user "name" and dataset alias:

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_name, epsg_alias.alias,
epsg_coordinatereferencesystem.coord_ref_sys_code
FROM epsg_coordinatereferencesystem
INNER JOIN epsg_alias ON epsg_coordinatereferencesystem.coord_ref_sys_code =
epsg_alias.object_code
WHERE (epsg_alias.alias LIKE 'abcd') AND (epsg_alias.object_table_name =
'epsg_coordinatereferencesystem')
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_name;
```

Examples:
(i) Input CRS name = "Port Bouet"

CRS name	Alias	CRS code
Abidjan 1987	Port Bouet	4143
Locodjo 1965	Port Bouet	4142

(ii) Input CRS name = "Abidjan 1987"

CRS name	Alias	CRS code
Abidjan 1987	Côte D'Ivoire	4143
Abidjan 1987	Port Bouet	4143

(iii) Input CRS name = "ED50"

CRS name	Alias	CRS code
ED50	(null)	4230

(*Note: no alias or CRS ED50 in dataset.*)

E.1.02 SQL script to retrieve metadata for a record
(equivalent Access query: *qry_epsg_gn7_1_e102_metadata_ellipsoid*)

This example uses the ellipsoid table. Similar scripts can be applied to other primary tables in the dataset.

```
SELECT epsg_ellipsoid.ellipsoid_code, epsg_ellipsoid.remarks, epsg_ellipsoid.information_source,
epsg_ellipsoid.data_source, epsg_ellipsoid.change_id, epsg_ellipsoid.revision_date
FROM epsg_ellipsoid
WHERE ((epsg_ellipsoid.ellipsoid_code = xxxx) AND (epsg_ellipsoid.deprecated  = 0));
```

Example: Input ellipsoid code = 7001

Attribute	Value

Code:	7001
Remarks:	Original definition is a=20923713 and b=20853810 feet of 1796. For the 1936 retriangulation OSGB defines the relationship of feet of 1796 to the International metre through log(1.48401603) exactly [=0.3048007491...]. 1/f is given to 7 decimal places.
Information source:	Ordnance Survey of Great Britain.
Data source:	EPSG
Change ID:	98.321 98.34
Revision date:	1995-06-02

E.1.03. SQL script to retrieve coordinate operation code, name, version and type, given area of use

(equivalent Access query: *qry_epsg_gn7_1_e103_area_coordOp*)

```
SELECT epsg_coordoperation.coord_op_code, epsg_coordoperation.coord_op_name,
epsg_coordoperation.coord_tfm_version, epsg_coordoperation.coord_op_type
FROM epsg_coordoperation
LEFT JOIN epsg_area ON epsg_coordoperation.area_of_use_code = epsg_area.area_code
WHERE ((epsg_coordoperation.coord_op_type = 'transformation') OR
(epsg_coordoperation.coord_op_type = 'concatenated operation')) AND (epsg_area.area_name = 'abcd')
AND (epsg_coordoperation.deprecated = 0)
ORDER BY epsg_coordoperation.coord_op_variant;
```

Example: Input area of use = "Algeria"

Code	Coordinate Operation Name	Version	Coordinate Operation Type
1253	Nord Sahara 1959 to WGS 84 (1)	DMA-Alg	transformation
1882	Nord Sahara 1959 (Paris) to Nord Sahara 1959	IGN-Fra	transformation
8640	Nord Sahara 1959 (Paris) to WGS 84 (1)	EPSG-Dza	concatenated operation

E.1.04. SQL script to retrieve CRS type, given CRS code

(equivalent Access query: *qry_epsg_gn7_1_e104_crs_findType*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_kind
FROM epsg_coordinatereferencesystem
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatereferencesystem.deprecated = 0));
```

Examples:

	CRS code	CRS type
(i) Input CRS code = 7405	7405	compound
(ii) Input CRS code = 2105	2105	projected

E.1.05. SQL scripts to retrieve CRS status, given CRS code

(equivalent Access queries: *qry_epsg_gn7_1_e105a_crs_deprecation* and *qry_epsg_gn7_1_e105_crs_findStatus*)

This search returns CRS type, validity and if deprecated the reason for deprecation and code of replacement record (if any). (In Access it is run in two parts. The first finds all CRS records in the Deprecation table. These results are used by the second part).

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_kind, epsg_coordinatereferencesystem.deprecated,
epsg_deprecation.replaced_by, epsg_deprecation.deprecation_reason
FROM epsg_coordinatereferencesystem
LEFT JOIN epsg_deprecation ON epsg_coordinatereferencesystem.coord_ref_sys_code =
epsg_deprecation.object_code
```

```
WHERE (epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx);
```

Examples:

Attribute	(i) Input CRS code = 4977	(ii) Input CRS code = 2141
CRS code:	4977	2141
CRS type:	geographic 3D	projected
Deprecated?:	0	1
Replaced by (code):	(null)	2946
Deprecation reason:	(null)	Change of base geogCRS name to accord with revised Geomatics Canada practice.

Note: The record is valid if *deprecated* = "No" or "False" or "0".

E.1.06. SQL script to retrieve geographic 2D CRS code, given geocentric CRS or geographic 3D CRS code

(equivalent Access query: *qry_epsg_gn7_1_e106_geocen2geog2D*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_kind,
epsg_coordinatereferencesystem_1.coord_ref_sys_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_kind,
epsg_coordinatereferencesystem.coord_ref_sys_name
FROM epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_1
INNER JOIN epsg_coordinatereferencesystem
ON epsg_coordinatereferencesystem_1.datum_code = epsg_coordinatereferencesystem.datum_code
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx)
AND ((epsg_coordinatereferencesystem.coord_ref_sys_kind = 'geocentric')
OR (epsg_coordinatereferencesystem.coord_ref_sys_kind = 'geographic 3D'))
AND (epsg_coordinatereferencesystem_1.coord_ref_sys_kind = 'geographic 2D')
AND (epsg_coordinatereferencesystem.deprecated = 0)
AND epsg_coordinatereferencesystem_1.deprecated = 0);
```

Examples:

Input CRS code	CRS type	Related geog2D CRS code	Geog2D CRS type	Geog2D CRS name
4976	geocentric	4619	geographic 2D	SWEREF99
4978	geocentric	4326	geographic 2D	WGS 84
4979	geographic 3D	4326	geographic 2D	WGS 84

E.1.06a. SQL script to tabulate geographic and geocentric CRSs referenced to the same geodetic datum

(equivalent Access query: *qry_epsg_gn7_1_e106a_tabulation_geocrs*)

This script provides similar information to the previous example but rather than returning information for a specific input it tabulates all dataset content.

```
SELECT epsg_datum.datum_name, epsg_coordinatereferencesystem.coord_ref_sys_name,
epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_kind,
epsg_coordinatereferencesystem_1.coord_ref_sys_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_kind,
epsg_coordinatereferencesystem_2.coord_ref_sys_code,
epsg_coordinatereferencesystem_2.coord_ref_sys_kind
FROM ((epsg_coordinatereferencesystem
INNER JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_1 ON
epsg_coordinatereferencesystem.datum_code = epsg_coordinatereferencesystem_1.datum_code)
INNER JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_2 ON
epsg_coordinatereferencesystem.datum_code = epsg_coordinatereferencesystem_2.datum_code)
INNER JOIN epsg_datum ON (epsg_datum.datum_code = epsg_coordinatereferencesystem.datum_code)
AND (epsg_coordinatereferencesystem_1.datum_code = epsg_datum.datum_code)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_kind = 'geocentric') AND
(epsg_coordinatereferencesystem_1.coord_ref_sys_kind = 'geographic 3D') AND
(epsg_coordinatereferencesystem_2.coord_ref_sys_kind = 'geographic 2D') AND
(epsg_coordinatereferencesystem.deprecated = 0) AND (epsg_coordinatereferencesystem_1.deprecated =
0) AND (epsg_coordinatereferencesystem_2.deprecated = 0))
ORDER BY epsg_datum.datum_name;
```

E.1.07. SQL script to tabulate projected CRSs which share datum but have differing CS
(equivalent Access query: *qry_epsg_gn7_1_e107_tabulation_projcrsCSchange*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_name, epsg_coordinatesystem.coord_sys_name,
epsg_coordinatereferencesystem_1.coord_ref_sys_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_name, epsg_coordinatesystem_1.coord_sys_name
FROM epsg_coordinatesystem
INNER JOIN (epsg_coordinatesystem AS epsg_coordinatesystem_1
INNER JOIN (epsg_coordinatereferencesystem
INNER JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_1
ON (epsg_coordinatereferencesystem.projection_conv_code =
epsg_coordinatereferencesystem_1.projection_conv_code)
AND (epsg_coordinatereferencesystem.source_geogcrs_code =
epsg_coordinatereferencesystem_1.source_geogcrs_code)) ON epsg_coordinatesystem_1.coord_sys_code
= epsg_coordinatereferencesystem_1.coord_sys_code)
ON epsg_coordinatesystem.coord_sys_code = epsg_coordinatereferencesystem.coord_sys_code
WHERE ((epsg_coordinatereferencesystem.deprecated = 0)
AND (epsg_coordinatereferencesystem_1.deprecated = 0)
AND ((epsg_coordinatereferencesystem_1.coord_sys_code) <>
(epsg_coordinatereferencesystem.coord_sys_code)))
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_code;
```

E.1.08. SQL script to retrieve Coordinate Operation type, given Coordinate Operation code
(equivalent Access query: *qry_epsg_gn7_1_e108_coordOp_findType*)

```
SELECT epsg_coordoperation.coord_op_code, epsg_coordoperation.coord_op_type
FROM epsg_coordoperation
WHERE ((epsg_coordoperation.coord_op_code = xxxx) AND (epsg_coordoperation.deprecated = 0));
```

Examples:

	Coordinate Operation code	Coordinate Operation type
(i) Input code = 1301	1301	transformation
(ii) Input code = 10301	10301	conversion

E.1.09. SQL script to retrieve CRS code for a reciprocal transformation, given CRS code for a non-reversible transformation

```
SELECT epsg_coordoperation.coord_op_code, epsg_coordoperation.coord_op_name,
epsg_coordoperation_2.coord_op_code, epsg_coordoperation_2.coord_op_name
FROM epsg_coordoperation
INNER JOIN epsg_coordoperation AS epsg_coordoperation_2
ON (epsg_coordoperation.coord_op_method_code = epsg_coordoperation_2.coord_op_method_code)
AND (epsg_coordoperation.source_crs_code = epsg_coordoperation_2.target_crs_code)
AND (epsg_coordoperation.target_crs_code = epsg_coordoperation_2.source_crs_code)
AND (epsg_coordoperation.coord_tfm_version = epsg_coordoperation_2.coord_tfm_version)
WHERE ((epsg_coordoperation.coord_op_code = xxxx) AND (epsg_coordoperation_2.deprecated = 0));
```

Examples: Using coordinate operation code 1044 as the input argument should return coordinate operation code 1045. Note that not all transformations using non-reversible methods will have a reciprocal. The above SQL query using coordinate operation code 1028 as the input argument will not return anything.

(Similar Access query of all which tabulates results: *qry_epsg_gn7_1_e109_tabulation_reciprocalOperation*).

E.1.10. SQL script to retrieve data describing available coordinate transformations and concatenated operations, given CRS code

(equivalent Access query: *qry_epsg_gn7_1_e110_tfmStatusByCRS*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_code, epsg_coordoperation.coord_op_code,
epsg_coordoperation.coord_op_name, epsg_coordoperation.coord_tfm_version,
epsg_coordoperation.coord_op_type,
epsg_coordoperationmethod.coord_op_method_name, epsg_coordoperationmethod.reverse_op,
epsg_coordoperation.coord_op_accuracy,
epsg_area.area_north_bound_lat, epsg_area.area_south_bound_lat, epsg_area.area_east_bound_lon,
epsg_area.area_west_bound_lon, epsg_supersession.superseded_by,
epsg_coordoperation_1.coord_op_name, epsg_coordoperation_1.coord_tfm_version
FROM epsg_coordinatereferencesystem, epsg_coordinatereferencesystem AS
epsg_coordinatereferencesystem_1
INNER JOIN epsg_coordoperation ON ((epsg_coordinatereferencesystem.coord_ref_sys_code =
epsg_coordoperation.source_crs_code) AND (epsg_coordinatereferencesystem_1.coord_ref_sys_code =
epsg_coordoperation.target_crs_code))
INNER JOIN epsg_coordoperationmethod ON (epsg_coordoperation.coord_op_method_code =
epsg_coordoperationmethod.coord_op_method_code)
INNER JOIN epsg_area ON (epsg_coordoperation.area_of_use_code = epsg_area.area_code)
LEFT JOIN epsg_supersession ON ((epsg_supersession.object_table_name = 'epsg_coordoperation') AND
(epsg_coordoperation.coord_op_code = epsg_supersession.object_code))
LEFT JOIN epsg_coordoperation AS epsg_coordoperation_1 ON (epsg_coordoperation_1.coord_op_code
= epsg_supersession.superseded_by)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordoperation.deprecated = 0))
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_code;
```

Example: Input CRS code = 4289

	Transformation #1	Transformation #2	Transformation #3	Transformation #4
Source CRS code:	4289	4289	4289	4289
Target CRS code:	4258	4258	4326	4326

Example: Input CRS code = 4289

	Transformation #1	Transformation #2	Transformation #3	Transformation #4
Coord Operation Code:	1751	1066	1112	1672
Coord Operation Name:	Amersfoort to ETRS89 (1)	Amersfoort to ETRS89 (2)	Amersfoort to WGS 84 (1)	Amersfoort to WGS 84 (2)
Coord Tfm Version:	NCG-Nld 2000	NCG-Nld 2000	NCG-Nld 93	EPSG-Nld
Coord Operation Type:	transformation	transformation	transformation	transformation
Coord Op Method:	Coordinate Frame rotation	Molodensky-Badekas 10-parameter transformation	Position Vector 7-param. transformation	Coordinate Frame rotation
Reverse Op?	TRUE (or 1)	TRUE (or 1)	TRUE (or 1)	TRUE (or 1)
Coord Tfm Accuracy:	0.5	0.5	1	1
North latitude:	53.47	53.47	53.47	53.47
South latitude:	50.75	50.75	50.75	50.75
East longitude:	7.21	7.21	7.21	7.21
West longitude:	3.37	3.37	3.37	3.37
Succeeded by code:	(null)	(null)	1672	(null)
Coord Operation name:	(null)	(null)	Amersfoort to WGS 84 (2)	(null)
Coord Tfm Version:	(null)	(null)	EPSG-Nld	(null)

E.2 **CRS Description**

This section exemplifies scripts to return the geodetic parameters essential to describing various types of coordinate reference system. These scripts also return all data required to perform geocentric to/from geographical and geographical to/from grid conversions.

To select the appropriate script it is first necessary to know the type of CRS. (See example 1.04 above to determine CRS type from CRS code, and example 1.01 above to determine CRS code from text search on (any part of) CRS name). Then if the CRS type is not compound, run example script 2.1, 2.2 or 2.3 as appropriate for the CRS type followed by example 2.4. If the CRS type is compound first run example script 2.5 to determine the codes and types for the component CRSs, then run 2.1, 2.2 or 2.3 as appropriate and then 2.4 for each of the two component CRSs.

Example	Function returns	Input argument
E.2.1	geodetic data describing a geographic or geocentric CRS.	CRS code
E.2.2	geodetic data describing a projected CRS.	CRS code
E.2.3	geodetic data describing a vertical or engineering CRS.	CRS code
E.2.4	data describing a coordinate system (CS) for any single CRS.	CRS code
E.2.5	names and codes of CRSs forming a compound CRS	compound CRS code

E.2.1. SQL script to retrieve geodetic parameters describing a geographic or geocentric CRS, given CRS code

(equivalent Access query: *qry_epsg_gn7_1_e21_crsID_geo*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code, SELECT
epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_name,
epsg_coordinatereferencesystem.coord_ref_sys_kind, epsg_datum.datum_name,
ell.ellipsoid_name, ell.semi_major_axis, epsg_unitofmeasure.unit_of_meas_name,
IF (ell.inv_flattening>0, ell.inv_flattening, IF ((ell.semi_major_axis - ell.semi_minor_axis) = 0, '',
(ell.semi_major_axis/(ell.semi_major_axis - ell.semi_minor_axis)))) AS '1/f',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '',
epsg_primemeridian.prime_meridian_name) AS 'Non-Greenwich_Prime_Meridian',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '',
epsg_primemeridian.greenwich_longitude) AS 'PM_Greenwich_Longitude',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '',
epsg_unitofmeasure_1.unit_of_meas_name) AS 'PM_LongitudeUnit'
FROM epsg_unitofmeasure AS epsg_unitofmeasure_1
INNER JOIN (epsg_primemeridian
INNER JOIN (epsg_unitofmeasure
INNER JOIN epsg_ellipsoid AS ell ON epsg_unitofmeasure.uom_code = ell.uom_code
INNER JOIN (epsg_datum
INNER JOIN epsg_coordinatereferencesystem ON epsg_datum.datum_code =
epsg_coordinatereferencesystem.datum_code) ON ell.ellipsoid_code = epsg_datum.ellipsoid_code)
ON epsg_primemeridian.prime_meridian_code = epsg_datum.prime_meridian_code) ON
epsg_unitofmeasure_1.uom_code = epsg_primemeridian.uom_code
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatereferencesystem.coord_ref_sys_kind LIKE 'geo%') AND
(epsg_coordinatereferencesystem.deprecated = 0))
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_name;
```

Notes: 1. If ellipsoid inverse flattening is included in the dataset it is used, else it is calculated from the semi-major and semi-minor axes.
2. Prime meridian details are shown only for CRS's using a prime meridian other than Greenwich.
3. See example E.2.4 for script to retrieve coordinate axis information.

Examples:	(i) Input CRS code = 4230	(ii) Input CRS code = 4807
Attribute		
CRS name:	ED50	NTF (Paris)
CRS type:	geographic 2D	geographic 2D
Datum name:	European Datum 1950	Nouvelle Triangulation Francaise (Paris)
Ellipsoid name:	International 1924	Clarke 1880 (IGN)
Semi-major axis:	6378388	6378249.2
Ellipsoid unit name:	metre	metre
1/f:	297	293.4660213
Prime meridian name:	(null)	Paris
Prime meridian Greenwich longitude:	(null)	2.5969213
Prime meridian longitude unit:	(null)	grad

All of the data necessary for geographical to/from geocentric coordinate conversions are retrived within the above script.

E.2.2. SQL script to retrieve geodetic parameters describing a projected CRS, given CRS code
(Access queries *qry_epsg_gn7_1_e22a_crsID_proj* and *qry_epsg_gn7_1_e22b_crsID_projParam*)

This is done in two steps:

(i) All information except for map projection parameter data is obtained from:
(equivalent Access query: *qry_epsg_gn7_1_e22a_crsID_proj*)

```
SELECT DISTINCT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_name,
epsg_coordinatereferencesystem.coord_ref_sys_kind,
epsg_coordinatereferencesystem.source_geogcrs_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_name, epsg_datum.datum_name,
epsg_ellipsoid.ellipsoid_name,
epsg_ellipsoid.semi_major_axis * (epsg_unitofmeasure.factor_b * epsg_unitofmeasure_1.factor_c) /
(epsg_unitofmeasure.factor_c * epsg_unitofmeasure_1.factor_b) AS 'sma in CS unit',
epsg_unitofmeasure_1.unit_of_meas_name,
IF (epsg_ellipsoid.inv_flattening>0, epsg_ellipsoid.inv_flattening, IF ((epsg_ellipsoid.semi_major_axis -
epsg_ellipsoid.semi_minor_axis) = 0, '',
(epsg_ellipsoid.semi_major_axis/(epsg_ellipsoid.semi_major_axis - epsg_ellipsoid.semi_minor_axis))))
AS '1/f',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '', prime_meridian_name) AS 'Non-
Greenwich_Prime_Meridian',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '', greenwich_longitude) AS
'PM_Greenwich_Longitude',
IF (epsg_primemeridian.prime_meridian_name LIKE 'Greenwich', '',
epsg_unitofmeasure_2.unit_of_meas_name) AS 'PM_LongitudeUnit',
epsg_coordoperationmethod.coord_op_method_name
FROM epsg_coordinatereferencesystem
LEFT JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_1 ON
(epsg_coordinatereferencesystem.source_geogcrs_code =
epsg_coordinatereferencesystem_1.coord_ref_sys_code)
LEFT JOIN epsg_datum ON (epsg_coordinatereferencesystem_1.datum_code = epsg_datum.datum_code)
LEFT JOIN epsg_ellipsoid ON (epsg_datum.ellipsoid_code = epsg_ellipsoid.ellipsoid_code)
LEFT JOIN epsg_unitofmeasure ON (epsg_ellipsoid.uom_code = epsg_unitofmeasure.uom_code)
LEFT JOIN epsg_datum AS epsg_datum_1 ON (epsg_ellipsoid.ellipsoid_code =
epsg_datum_1.ellipsoid_code)
LEFT JOIN epsg_coordinateaxis ON (epsg_coordinatereferencesystem.coord_sys_code =
epsg_coordinateaxis.coord_sys_code)
LEFT JOIN epsg_unitofmeasure AS epsg_unitofmeasure_1 ON (epsg_coordinateaxis.uom_code =
epsg_unitofmeasure_1.uom_code)
LEFT JOIN epsg_primemeridian ON (epsg_datum.prime_meridian_code =
epsg_primemeridian.prime_meridian_code)
LEFT JOIN epsg_unitofmeasure AS epsg_unitofmeasure_2 ON (epsg_primemeridian.uom_code =
epsg_unitofmeasure_2.uom_code)
LEFT JOIN epsg_coordoperation ON (epsg_coordinatereferencesystem.projection_conv_code =
epsg_coordoperation.coord_op_code)
LEFT JOIN epsg_coordoperationmethod ON (epsg_coordoperation.coord_op_method_code =
epsg_coordoperationmethod.coord_op_method_code)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatereferencesystem.coord_ref_sys_kind = 'projected') AND
(epsg_coordinatereferencesystem.deprecated = 0))
ORDER BY epsg_coordinatereferencesystem_1.coord_ref_sys_name;
```

Notes:

1. Ellipsoid semi-major axis is converted to CS units.
2. If ellipsoid inverse flattening is included in the dataset it is used, else it is calculated from the semi-major and semi-minor axes.
3. If the map projection uses spherical (as opposed to ellipsoidal) formulae, ellipsoid inverse flattening is not shown.
4. Prime meridian details are shown only for base geographic CRS's using a prime meridian other than Greenwich.
5. See example E.2.4 for script to retrieve coordinate axis information.

Examples:

Attribute	Input CRS code (i) 23031	(ii) 32040	(iii) 27572
CRS name:	ED50 / UTM zone 31N	NAD27 / Texas South Central	NTF (Paris) / Lambert zone II
CRS type:	projected	Projected	projected
Base geogCRS name:	4230	4267	4807
Base geogCRS name:	ED50	NAD27	NTF (Paris)
Datum name:	European Datum 1950	North American Datum 1927	Nouvelle Triangulation Francaise (Paris)
Ellipsoid name:	International 1924	Clarke 1866	Clarke 1880 (IGN)
Semi-major axis (in CS units):	6378388	20925832.164	6378249.2
Ellipsoid unit name:	metre	US survey foot	metre
1/f:	297	294.9786982	293.4660213
Prime meridian name:	(null)	(null)	Paris
Prime meridian Greenwich longitude:	(null)	(null)	2.5969213
Prime meridian longitude unit:	(null)	(null)	grad
Coordinate Operation method name:	Transverse Mercator	Lambert Conic Conformal (2SP)	Lambert Conic Conformal (1SP)

(ii) Then the map projection parameter names and values are obtained from:
(equivalent Access query: *qry_epsg_gn7_1_e22b_crsID_projParam*)

```
SELECT epsg_coordoperationparam.parameter_name, epsg_coordoperationparamvalue.parameter_value,
epsg_unitofmeasure.unit_of_meas_name
FROM epsg_coordoperationparam
INNER JOIN epsg_coordoperationparamusage ON (epsg_coordoperationparam.parameter_code =
epsg_coordoperationparamusage.parameter_code)
INNER JOIN epsg_coordoperationmethod ON (epsg_coordoperationparamusage.coord_op_method_code
= epsg_coordoperationmethod.coord_op_method_code)
INNER JOIN epsg_coordoperation ON (epsg_coordoperationmethod.coord_op_method_code =
epsg_coordoperation.coord_op_method_code)
INNER JOIN epsg_coordinatereferencesystem ON (epsg_coordoperation.coord_op_code =
epsg_coordinatereferencesystem.projection_conv_code)
INNER JOIN epsg_coordoperationparamvalue ON ((epsg_coordoperation.coord_op_code =
epsg_coordoperationparamvalue.coord_op_code)
AND (epsg_coordoperationmethod.coord_op_method_code =
epsg_coordoperationparamvalue.coord_op_method_code)
AND (epsg_coordoperationparam.parameter_code = epsg_coordoperationparamvalue.parameter_code))
INNER JOIN epsg_unitofmeasure ON (epsg_coordoperationparamvalue.uom_code =
epsg_unitofmeasure.uom_code)
WHERE (epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx)
ORDER BY epsg_coordoperationparamusage.sort_order;
```

Examples:

(i) Input CRS code = 23031

Parameter name	Parameter value	Unit
Latitude of natural origin	0	degree
Longitude of natural origin	3	degree
Scale factor at natural origin	0.9996	unity
False easting	500000	metre
False northing	0	metre

(ii) Input CRS code = 32040

Parameter name	Parameter value	Parameter value unit
Latitude of false origin	27.5	sexagesimal DMS
Longitude of false origin	-99	sexagesimal DMS
Latitude of 1st standard parallel	28.23	sexagesimal DMS
Latitude of 2nd standard parallel	30.17	sexagesimal DMS
Easting at false origin	2000000	US survey foot
Northing at false origin	0	US survey foot

Note the similarity between this script and that given in example E.3.1(ii) below.

All of the data necessary for geographic to/from grid (i.e. projected) coordinate conversions are retrieved within the above two scripts E2.2a and E2.2b.

E.2.3. SQL script to retrieve geodetic parameters describing a vertical or engineering CRS, given CRS code

(equivalent Access query: *qry_epsg_gn7_1_e23_crsID_VertOrEng*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_name,
epsg_coordinatereferencesystem.coord_ref_sys_kind, epsg_datum.datum_name
FROM epsg_datum
INNER JOIN epsg_coordinatereferencesystem
ON epsg_datum.datum_code = epsg_coordinatereferencesystem.datum_code
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatereferencesystem.coord_ref_sys_kind = 'vertical') AND
(epsg_coordinatereferencesystem.deprecated = 0))
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_name;
```

Notes:

1. For engineering CRSs, replace *vertical* with *engineering*.
2. See example E.2.4 for script to retrieve coordinate axis information.

Example: Input CRS code = 5783

CRS name	CRS type	Datum name
DHHN92	vertical	Deutches Haupthohennetz 1992

E.2.4. SQL script to retrieve data describing a coordinate system (CS), given CRS code

(equivalent Access query: *qry_epsg_gn7_1_e24_crsID_singleCSaxes*)

This script is used in conjunction with each of the three examples E.2.1 through E.2.3 above.

```
SELECT epsg_coordinateaxisname.coord_axis_name, epsg_coordinateaxis.coord_axis_abbreviation,
epsg_coordinateaxis.coord_axis_orientation, epsg_unitofmeasure.unit_of_meas_name
FROM epsg_coordinateaxis
INNER JOIN epsg_coordinatesystem ON (epsg_coordinateaxis.coord_sys_code =
epsg_coordinatesystem.coord_sys_code)
INNER JOIN epsg_coordinatereferencesystem ON (epsg_coordinatesystem.coord_sys_code =
epsg_coordinatereferencesystem.coord_sys_code)
LEFT JOIN epsg_coordinateaxisname ON (epsg_coordinateaxis.coord_axis_name_code =
epsg_coordinateaxisname.coord_axis_name_code)
LEFT JOIN epsg_unitofmeasure ON (epsg_coordinateaxis.uom_code = epsg_unitofmeasure.uom_code)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatesystem.deprecated = 0))
ORDER BY epsg_coordinateaxis.coord_axis_order;
```

Example: Input CRS code = 4979

Axis name	Axis abbreviation	Axis orientation	Axis unit
Geodetic latitude	Lat	north	degree (supplier to define representation)
Geodetic longitude	Long	east	degree (supplier to define representation)
Ellipsoidal height	h	up	metre

E.2.5. SQL script to retrieve names, codes and types of CRSs forming a compound CRS, given compound CRS code

(equivalent Access query: *qry_epsg_gn7_1_e25_crsID_compound*)

```
SELECT epsg_coordinatereferencesystem.coord_ref_sys_code,
epsg_coordinatereferencesystem.coord_ref_sys_name,
epsg_coordinatereferencesystem.coord_ref_sys_kind,
epsg_coordinatereferencesystem.cmpd_horizcrs_code,
epsg_coordinatereferencesystem_1.coord_ref_sys_name,
epsg_coordinatereferencesystem_1.coord_ref_sys_kind,
epsg_coordinatereferencesystem.cmpd_vertcrs_code,
epsg_coordinatereferencesystem_2.coord_ref_sys_name,
epsg_coordinatereferencesystem_2.coord_ref_sys_kind
FROM epsg_coordinatereferencesystem
LEFT JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_1 ON
epsg_coordinatereferencesystem.cmpd_horizcrs_code =
epsg_coordinatereferencesystem_1.coord_ref_sys_code
LEFT JOIN epsg_coordinatereferencesystem AS epsg_coordinatereferencesystem_2 ON
epsg_coordinatereferencesystem.cmpd_vertcrs_code =
epsg_coordinatereferencesystem_2.coord_ref_sys_code
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND
(epsg_coordinatereferencesystem.coord_ref_sys_kind = 'compound') AND
(epsg_coordinatereferencesystem.deprecated = 0))
ORDER BY epsg_coordinatereferencesystem.coord_ref_sys_name;
```

Example: Input CRS code = 7404

Attribute	Value
Compound CRS name:	RT90 + RH70
CRS type:	compound
Horizontal CRS code:	4124
Horizontal CRS name:	RT90
Horizontal CRS type:	geographic 2D
Vertical CRS code:	5718
Vertical CRS name:	RH70
Vertical CRS type:	vertical

The details for each of the two component single CRSs may then be found by using the SQL queries in the previous examples E.2.1 through E2.3 (as appropriate for each single CRS type) and E.2.4 (twice, once for each of the two single systems).

E.3 Coordinate Transformation Description

Example	Function returns	Input argument
E.3.1	geodetic data describing a transformation.	Coordinate operation code
E.3.2	ellipsoid parameters.	CRS code
E.3.3	names and codes of transformations forming a concatenated operation.	Concatenated operation code

E.3.1. SQL script to retrieve geodetic parameters describing a transformation, given coordinate operation code

(equivalent Access queries: *qry_epsg_gn7_1_e31a_tfmID* and *qry_epsg_gn7_1_e31b_tfmID_tfmParam*)

This is done in two or more steps:

(i) All general information except for transformation parameter data is obtained from:
(equivalent Access query: *qry_epsg_gn7_1_e31a_tfmID*)

```
SELECT epsg_coordoperation.coord_op_code, epsg_coordoperation.coord_op_name,
epsg_coordoperation.coord_op_type, epsg_coordoperation.source_crs_code,
epsg_coordoperation.target_crs_code,
epsg_coordoperation.coord_op_variant, epsg_coordoperation.coord_tfm_version,
epsg_area.area_north_bound_lat, epsg_area.area_south_bound_lat, epsg_area.area_east_bound_lon,
epsg_area.area_west_bound_lon,
epsg_coordoperation.coord_op_accuracy, epsg_coordoperationmethod.coord_op_method_name,
epsg_unitofmeasure.unit_of_meas_name, epsg_unitofmeasure_1.unit_of_meas_name
FROM epsg_coordoperation
LEFT JOIN epsg_area ON (epsg_coordoperation.area_of_use_code = epsg_area.area_code)
LEFT JOIN epsg_coordoperationmethod ON (epsg_coordoperation.coord_op_method_code =
epsg_coordoperationmethod.coord_op_method_code)
LEFT JOIN epsg_unitofmeasure ON (epsg_coordoperation.uom_code_source_coord_diff =
epsg_unitofmeasure.uom_code)
LEFT JOIN epsg_unitofmeasure AS epsg_unitofmeasure_1 ON
(epsg_coordoperation.uom_code_target_coord_diff = epsg_unitofmeasure_1.uom_code)
WHERE ((epsg_coordoperation.coord_op_code = xxxx) AND (epsg_coordoperation.coord_op_type =
'transformation') AND (epsg_coordoperation.deprecated = 0));
```

Examples:

Attribute	Input Coord Op code = (i) 1086	(ii) 1048
Coord Operation name:	JAD69 to WGS 84 (3)	Belge 72 / Lambert to ED50 / UTM zone 31N (1)
Coord Operation type:	transformation	transformation
Source CRS code:	4242	31300
Target CRS code:	4326	23031
Coordinate transformation variant:	3	1
Coordinate transformation version:	UT-Jam 1m	NCG-Bel
Geog bounding box north latitude:	19.42	51.83
Geog bounding box south latitude:	14.63	49.53
Geog bounding box right longitude:	-74.38	6.4
Geog bounding box left longitude:	-80.74	2.12
Coordinate transformation accuracy:	1	1
Coordinate Operation method name:	Coordinate Frame rotation	Complex polynomial of degree 3
Polynomial source CRS offset UoM	(null)	metre
Polynomial target CRS offset UoM	(null)	metre

(ii) Then the transformation parameter names and values are obtained from:
(equivalent Access query: *qry_epsg_gn7_1_e31b_tfmID_tfmParam*)

```
SELECT epsg_coordoperationparam.parameter_name, epsg_coordoperationparamvalue.parameter_value,
epsg_unitofmeasure.unit_of_meas_name, epsg_coordoperationparamvalue.param_value_file_ref
FROM epsg_coordoperationparam
INNER JOIN epsg_coordoperationparamusage ON (epsg_coordoperationparam.parameter_code =
epsg_coordoperationparamusage.parameter_code)
INNER JOIN epsg_coordoperationmethod ON (epsg_coordoperationparamusage.coord_op_method_code
= epsg_coordoperationmethod.coord_op_method_code)
INNER JOIN epsg_coordoperation ON (epsg_coordoperationmethod.coord_op_method_code =
epsg_coordoperation.coord_op_method_code)
LEFT JOIN epsg_coordoperationparamvalue ON ((epsg_coordoperation.coord_op_code =
epsg_coordoperationparamvalue.coord_op_code)
AND (epsg_coordoperationmethod.coord_op_method_code =
epsg_coordoperationparamvalue.coord_op_method_code)
AND (epsg_coordoperationparam.parameter_code = epsg_coordoperationparamvalue.parameter_code))
LEFT JOIN epsg_unitofmeasure ON (epsg_coordoperationparamvalue.uom_code =
epsg_unitofmeasure.uom_code)
WHERE (epsg_coordoperation.coord_op_code = xxxx)
ORDER BY epsg_coordoperationparamusage.sort_order;
```

Examples:
(i) Input Coordinate Operation code = 1086

Parameter name	Parameter value	Unit	Parameter file name
X-axis translation	-33.722	metre	(null)
Y-axis translation	153.789	metre	(null)
Z-axis translation	94.959	metre	(null)
X-axis rotation	8.581	arc-second	(null)
Y-axis rotation	4.478	arc-second	(null)
Z-axis rotation	-4.54	arc-second	(null)
Scale difference	-8.95	parts per million	(null)

(ii) Input Coordinate Operation code = 1241

Parameter name	Parameter value	Unit	Parameter file name
Latitude difference file	(null)	(null)	ftp://ftp.ngs.noaa.gov/pub/pcsoft/nadcon/conus.las
Longitude difference file	(null)	(null)	ftp://ftp.ngs.noaa.gov/pub/pcsoft/nadcon/conus.los

(iii) Some transformation method formulae require ellipsoid parameters related to the transformation's source and target CRSs. These may be obtained through two applications of the script in E.3.2 below, the argument for which is the CRS code for the source or target CRS, as appropriate. The CRS codes for the source and target CRSs are returned from *qry_epsg_gn7_1_e31a_tfmID* above.

E.3.2. SQL script to retrieve ellipsoid parameters, given CRS code
(Equivalent Access query *qry_epsg_gn7_1_e32_ellipsoidParam*)

```
SELECT DISTINCT epsg_ellipsoid.ellipsoid_name, epsg_ellipsoid.semi_major_axis,
epsg_unitofmeasure.unit_of_meas_name,
IF (epsg_ellipsoid.inv_flattening>0, epsg_ellipsoid.inv_flattening, IF ((epsg_ellipsoid.semi_major_axis -
epsg_ellipsoid.semi_minor_axis) = 0, '',
(epsg_ellipsoid.semi_major_axis/(epsg_ellipsoid.semi_major_axis - epsg_ellipsoid.semi_minor_axis))))
AS '1/f'
FROM epsg_ellipsoid
INNER JOIN epsg_datum ON (epsg_ellipsoid.ellipsoid_code = epsg_datum.ellipsoid_code)
INNER JOIN epsg_coordinatereferencesystem ON (epsg_datum.datum_code =
epsg_coordinatereferencesystem.datum_code)
LEFT JOIN epsg_unitofmeasure ON (epsg_ellipsoid.uom_code = epsg_unitofmeasure.uom_code)
WHERE ((epsg_coordinatereferencesystem.coord_ref_sys_code = xxxx) AND (epsg_ellipsoid.deprecated
= 0))
ORDER BY epsg_ellipsoid.ellipsoid_name;
```

Examples:

Input CRS code	Ellipsoid name	Semi-major axis (a)	Unit	Inverse flattening (1/f)
4314	Bessel 1841	6377397.155	metre	299.1528128
4258	GRS 1980	6378137	metre	298.2572221

If the axes units differ they may require conversion to a unit consistent with other linear parameters required by the transformation method.

E.3.3. SQL script to retrieve names and codes of transformations forming a concatenated operation, given concatenated operation code
(Access query *qry_epsg_gn7_1_e33_tfmID_concat*)

```
SELECT epsg_coordoperationpath.op_path_step, epsg_coordoperation_1.coord_op_code,
epsg_coordoperation_1.coord_op_name, epsg_coordoperationmethod.reverse_op
FROM epsg_coordoperation
LEFT JOIN epsg_coordoperationpath ON (epsg_coordoperation.coord_op_code =
epsg_coordoperationpath.concat_operation_code)
LEFT JOIN epsg_coordoperation AS epsg_coordoperation_1 ON
(epsg_coordoperationpath.single_operation_code = epsg_coordoperation_1.coord_op_code)
LEFT JOIN epsg_coordoperationmethod ON (epsg_coordoperation_1.coord_op_method_code =
epsg_coordoperationmethod.coord_op_method_code)
WHERE ((epsg_coordoperation.coord_op_code = xxxx) AND (epsg_coordoperation.coord_op_type =
'concatenated operation') AND (epsg_coordoperation.deprecated=0))
ORDER BY epsg_coordoperationpath.op_path_step;
```

Example: Input Concatenated Operation code = 8647

Step	Coord Op Code	Coord Op Name	Op reversible?
1	1313	NAD27 to NAD83 (4)	Yes (or 1)
2	1950	NAD83 to NAD83(CSRS) (4)	Yes (or 1)
3	1946	NAD83(CSRS) to WGS 84 (2)	Yes (or 1)

The details describing each of the component single operations may then be found using the SQL queries in the previous example E.3.1 and if necessary also E.3.2.