



Leading healthcare
terminology, worldwide

SNOMED CT SQL Practical Guide

Publication date: 2019-11-25

Web version link: <http://snomed.org/sqlpg>

SNOMED CT document library: <http://snomed.org/doc>

This PDF document was generated from the web version on the publication date shown above. Any changes made to the web pages since that date will not appear in the PDF. See the web version of this document for recent updates.

Table of Contents

- 1. Introduction.....4
- 2. Objectives, Audiences and Uses5
- 3. SNOMED CT Example Database.....6
- 4. Database Design.....9
 - 4.1. Essential Reference Information9
 - 4.2. Release Type Options.....9
 - 4.3. Data Type Options.....12
 - 4.4. Database Table Naming.....14
 - 4.5. Database Table Design.....17
 - 4.6. Enabling Versioned Views19
 - 4.7. Enabling Subtype Testing29
 - 4.8. Composite Views30
 - 4.9. Stored Procedures46
- 5. Creating and Populating a SNOMED CT Database.....56
 - 5.1. Creating the Database.....56
 - 5.2. Creating Tables for Components.....56
 - 5.3. Creating Tables for Reference Sets59
 - 5.4. Importing Release Files.....63
- Appendix A: Building the SNOMED CT Example Database.....67**
 - A.1 Download the SNOMED CT Example Database Package67
 - A.2 Download the Release File Package67
 - A.3 Instructions for Mac OS Users.....68
 - A.4 Instructions for Windows Users.....73
 - A.5 Using MySQL Workbench to Query SNOMED CT.....87
 - A.6 Overview of the SNOMED CT MySQL Database92
 - A.7 MySQL Reference Data.....97
- Appendix B: Obtaining SNOMED CT Release Files.....99**
- Appendix C: Release Types and Versioned Views.....100**
 - C.1. Practical Uses for Versioned Views101
 - C.2. Release Type Support for Versioned Views101
 - C.3. Common Mistakes with Snapshot Generation102

The logo for SNOMED International, featuring the word "SNOMED" in a large, bold, white sans-serif font above the word "International" in a smaller, white sans-serif font, both set against a solid blue square background.

Leading healthcare
terminology, worldwide

This guide provides a simple and practical example of how a relational database can be used to enable effective access to the content and features of SNOMED CT. The guide includes tested SQL scripts for loading release files into a relational database, searching the terminology and querying the terminology for concepts that meet simple but useful constraints.

The primary purpose of this document is to enhance understanding of the logical design of SNOMED CT and to provide practical ways to access the key features of SNOMED CT in a widely understood programming language. For more scalable and performant approaches to implementing SNOMED CT please visit SNOMED International's Github repository - e.g. <https://github.com/IHTSDO>.

Web browsable version: <http://snomed.org/sqlpg>

SNOMED CT Document Library: <http://snomed.org/doc>

© Copyright 2019 International Health Terminology Standards Development Organisation, all rights reserved.

This document is a publication of International Health Terminology Standards Development Organisation, trading as SNOMED International. SNOMED International owns and maintains SNOMED CT®.

Any modification of this document (including without limitation the removal or modification of this notice) is prohibited without the express written permission of SNOMED International. This document may be subject to updates. Always use the latest version of this document published by SNOMED International. This can be viewed online and downloaded by following the links on the front page or cover of this document.


SNOMED®, SNOMED CT® and IHTSDO® are registered trademarks of International Health Terminology Standards Development Organisation. SNOMED CT® licensing information is available at <http://snomed.org/licensing>. For more information about SNOMED International and SNOMED International Membership, please refer to <http://www.snomed.org> or contact us at info@snomed.org.

1. Introduction

Summary

This practical guide outlines key requirements for enabling effective access to [SNOMED CT](#) and illustrates some of the options for meeting these requirements using a freely available [SQL](#) database. The rationale for using SQL as an illustrative example is that this is a widely understood way to access structured data that can be readily applied to SNOMED CT release files.

The approaches and options documented in the guide provide working examples of some essential SNOMED CT [terminology services](#). However, the primary purpose of these examples is to enhance understanding and not to recommend SQL as a way to deliver large scale versions of these services. A range of more advanced technologies, including those used in [SNOMED International Tools](#), are able to deliver more scalable implementations of the required services.

 SNOMED International also provides a range of tools that enable access to SNOMED CT content. These include a [SNOMED CT Browser](#) through which to explore the terminology and a range of other open source tools that enable programmatic access to terminology content. For further details see [SNOMED International Tools](#) or visit the [SNOMED International GitHub repository](#) to access open source projects supporting a range of terminology service requirements.

Background

[SNOMED CT](#) is made available to [licensees](#) as a package of tab-delimited text files. The format of these files is specified in the [Release File Specification](#) (Sections [4 Component Release Files Specification](#) and [5. Reference Set Types](#)). These release files provide a standard way to distribute SNOMED CT content and derivatives but they do not offer a direct way to provide user-friendly access to the terminology.

This guide documents a worked SQL example that loads the SNOMED CT release files into a database designed to enable practical access to terminology content. This worked example enables exploration of the types of terminology required for effective delivery of [terminology services](#).

Purpose

The primary purpose of this document is to enhance understanding of the logical design and release file structure of SNOMED CT and to provide practical ways to access the key features of SNOMED CT in a widely understood programming language. For more scalable and performant approaches to implementing SNOMED CT please visit SNOMED International's Github repository - e.g. <https://github.com/IHTSDO>.

For more details on the purpose of the document see Section [2. Objectives, Audiences and Uses](#).

Limitations

1. The programming code that this guide contains and/or refers to is only made available to provide illustrative examples of key points related to the design, accessibility and use of SNOMED CT.
2. It is licensed under the [Apache 2](#) licence and is not certified to be suitable for use in a production system.
3. The code has been developed to work in a freely available open source relational database (MySQL - <https://www.mysql.com>) to make it accessible to all SNOMED CT licensees without imposing additional costs.
 - The code may need modification to work in other database environments.
 - Other database environments may also support additional features which enable alternative approaches not described in this guide.
4. The use of SQL in this example is intended to enhance understanding of the structure and does **not** imply that SQL is recommended for use production systems that deliver terminology services.

2. Objectives, Audiences and Uses

Objectives

The key objectives of this guide are to:

1. Enhance understanding of the logical design of SNOMED CT;
2. Demonstrate the feasibility of effective terminology access;
3. Enable terminology access for practical exercises;
4. Provide a starting point for further development.

Audiences

There are three distinct target audiences for this guide:

1. Designers and developers
 - The guide provides practical examples of the ways in which features in SNOMED CT can be accessed using a widely understood technology;
 - It demonstrates services that designers and developers should consider including in their terminology server developments;
 - It can also be used to inform terminology server requirements for system access to SNOMED CT.
2. People seeking practical ways to access SNOMED CT content and reference sets:
 - The demonstrator documented in this guide provides ways to access SNOMED CT content and reference sets using simple SQL queries.
 - This type of access can complement the use of a SNOMED CT browser or other specific tools by enabling customizable access to specific collections of terminology content.
3. Anyone interested in SNOMED CT who is seeking a practical example of accessing the terminology in ways described in other specifications and guides:
 - Although many SNOMED CT tools deliver user-friendly services for well-established use cases, some use cases require customized services that are not readily available from existing tools.
 - For example, the full version of the demonstration database allows queries to be written that report the history of changes that impact a defined set of SNOMED CT components.

Each section of the guide includes an indication of its applicability to each of these audiences.

Ways of Using this Guide

The guide is designed to be used in any of the following ways to suit the needs of different audiences.

1. As a set of step-by-step instructions for a practical demonstration project
2. As a source of ideas and advice to inform other development approaches
3. As a source of practical examples that illustrate and validate general guidance
4. As background reading when exploring technical options

3. SNOMED CT Example Database

Role of the Example Database

A SNOMED CT example database is used throughout this guide to illustrate ways to meet a various requirements for practical access to SNOMED CT. It is not essential for readers to create or have access to an instance of the example database but for some people access to a practical example of the database will be valuable.

Some readers of the guide may wish to setup and use a working version of this database to explore simply to provide themselves with another way to explore SNOMED CT. Others may use the example database to test out the efficacy and performance of the techniques described in the guide. Those with existing knowledge and experience of SQL may also wish to implement and test alternative approaches while reading this guide. Readers may also prefer to adapt the guidance in this document to another database environment with which they are already more familiar. Most of techniques used in the example database are standard features of SQL and that are supported by many relational database environments. Therefore, it should be fairly straight forward to apply or adapt the SQL examples in the guide to other database environments. On the other hand readers who are familiar with databases with more powerful capabilities than MySQL may see opportunities to use additional features to replace some techniques used in the example database.

Requirements for Creating the Database

- A computer system running Windows or MacOS or a Unix-based environment like Linux or Ubuntu for which MySQL Server 8.0.x if available.
- Installation of MySQL Community Edition Server (which is used to build and provide access to the database) and MySQL Community Edition Workbench (which provides a user interface through which the database can be conveniently explored). Both of these are freely available with standard installation packages for all the most widely used operating system environments include Windows, MacOS and other Unix-based environments such as Linux and Ubuntu.
- Those using Windows will also need to install a Perl processor, but fortunately the Strawberry Perl environment is also freely available, so there is no additional software to be purchased.
- At least 10.5 Gb of disk storage space during the build process. Assuming the release package archive file and the expanded release package folder are deleted the storage requirement reduces to 6.5Gb after the build has completed [1](#).

Creating the Example Database

Those readers wishing to create their own instance of the example database should refer to [Appendix A: Building the SNOMED CT Example Database](#). That appendix provides instructions on the steps required to configure MySQL and then to create, populate and experiment with the example SNOMED CT database. These instructions are supported by installation scripts for Windows and MacOS, which have been tested to work with the standard installations of MySQL 8.0.x Community Server for those environments.

Functionality of the Example Database

Overview

The SNOMED CT example database is designed to provide an effective and flexible way to access SNOMED CT content imported from a [SNOMED CT release package](#). Access is provided through running SQL queries against tables containing the data and predefined database views that allow data to be selected in ways that recognize versioning information and where appropriate combine related data from different tables and views.

Important Note

The example database in its current form should be considered as a read-only resource. Although an SQL databases enables addition, deletion and updating of data, the design of the example database is **not** intended to be used for editing the release data in any way. The reasons for this limitation include the conditions of the SNOMED CT license as well as lack of support for the formal processes required for authoring, change management and component versioning. Those interested in tools that support editing of SNOMED CT content and reference sets should refer to information on [SNOMED International Software and Tools](#).

Feature	Description
Database tables created for data in all full and snapshot release files	Separate database tables are created for the full and snapshot release of each distinct component and reference set type. Consistent design principles are applied to these tables to match the data structure, data type and function requirements.
Import of all full release files	The full release makes it possible to use SQL queries to access to the complete history of all SNOMED CT components and reference sets from the first release of the terminology in January 2002 up to the most recent release.
Import of all snapshot release files	The snapshot release provides rapid access to the current view of the terminology, without the need for queries to explicitly exclude earlier versions.
Examples of computed current snapshot views	As the example database imports the snapshot release files into separate tables, there is no requirement for a computed snapshot views. However, computed current snapshot views derived from some of the full release tables are included in the example database. These views provides practical illustrations of the way to derive a current snapshot from the full release. They also allow comparisons of performance and output between SQL queries run on the snapshot table and the same queries run on the computed snapshot view.
Configurable retrospective snapshot views	The example database includes two configurable snapshot views of every full release database table. This allows SQL queries to be refer to and compare the current snapshot and snapshot and one or two earlier dates. A simple configuration procedure is included to allow the retrospective snapshot dates of each view to be changed.
Configurable delta views	The example database includes three configurable delta views of every full release database table. This allows SQL queries to be refer only to versions of a component added or changed between two specified dates. A simple configuration procedure is included to allow the delta data ranges (start and end times) of each view to be changed.
Language configuration of views	The example database includes a simple configuration procedure to specify the preferred display language or dialect. This is dependent on the languages in the release files, so with the International Edition only the options en-US and en-GB can be used. However, when used with National Editions that include translated descriptions and an appropriate language reference set this feature can be readily configured to support those additional language settings. The language configuration setting can be applied separately to different snapshot views allowing or can be switched as part of an sequence of queries to enable multilingual query results.
Consistent access to all components and reference set members	The same configurable snapshot and delta views are available for all component types and reference set members and these views follow a consistent naming convention. This means individual components and reference sets can be queried in a consistent manner. It also provides a foundation for composite views that bring together data from the same snapshot view of different related tables. The features below illustrate the practical application of this principle.
Language refset dependent views of descriptions and concepts	Built-in views allow concepts to be displayed using either their fully specified name or preferred synonym in a specified language or dialect. Other built-in language dependent descriptions views include: <ul style="list-style-type: none"> • All the active synonyms of each concept (including or excluding the preferred synonym). • All active synonyms associated with active concepts (this view is particularly useful for text searches).
Integrated views of relationships and descriptions	The database also features built-in views that provide access to information about relationships between concepts. These views include synonym and fully specified name variants for each of the following: <ul style="list-style-type: none"> • The <i>id and term</i> of all subtype children of specified concept. • The <i>id and term</i> of all supertype parents of a specified concept. • Defining relationships as the <i>id and term</i> of the source, type and destination concepts followed by the relationship group number. <p>All references to <i>id and term</i> refer to a pair of columns in the output of the view. The <i>id</i> is the concept identifier, depending on the view the <i>term</i> is either the fully specified name or the preferred synonym.</p>

Full text term searches	The database uses a full-text index to allow searches by words within a term independent of the order in which those terms appear. Example queries demonstrate this search facility as well as a simple way to display the closest matches first. Other more technical search approaches using complete, pattern and regular expression matching can also be readily applied.
Rapid subtype testing using a transitive closure table	The database import process includes a step that builds a transitive closure file and loads this into an indexed database table. This enables rapid testing of whether one concept is a subtype descendant or supertype ancestor of any other concept ² .
Procedure demonstrating selection of concepts based on expression constraints	This illustrates an approach to testing expression constraints. The current version of the procedure is limited to constraints including one focus concept with one or two attribute value constraints ² .
Procedure demonstrating term search limited to a specified hierarchy	This illustrates an approach to searching using full text search with the returned results limited to concepts that are subtypes of a specified concept. For example, there are more than 600 synonyms that contains both "mitral" and "valve". However if this search is limited to subtypes of a concept (e.g. procedure, body structure, observable, physical object etc.) it returns substantially fewer matches ³ .

⚠ Note

This summary of the functionality the database should be interpreted in the scope and context of the its intended use as a demonstrator and learning tool. It is possible that the database may have other practical applications for some use cases, but it is not regarded as a robust or high performance solution suitable for large scale use. Rather it is designed to confirm the feasibility implementing some of the feature of SNOMED CT and to stimulate others to use and improve upon the ideas and approaches described in the guide.

-
- ¹ These storage requirements are for the International Edition Release for 2019-07-31. More storage may be required by other Editions with substantial additional national or local extension content.
- ² Only the current snapshot view of the transitive closure table is available. Therefore this feature is not available for retrospective snapshots. [a b]
- ³ Only the current snapshot view of the transitive closure table is available. Therefore subtype testing is not available when searching retrospective snapshots.

4. Database Design

This section discusses some of the points that should be considered when designing a relational database to provide access to SNOMED CT. It explains the rationale for decisions taken while designing the SNOMED CT example database. It also identifies some other options which, while not implemented in the example database, are worth considering.

4.1. Essential Reference Information

Before designing a database to accommodate SNOMED CT, you should refer to the [SNOMED CT Release File Specification](#). This provides the authoritative documentation about the way that SNOMED CT content is distributed to licensees. The list below picks out the key sections of this guide for those developing solutions that provide access to SNOMED CT.

Section 3 [Release Types, Packages and Files](#) contains the follow chapters that provide essentially information about [SNOMED CT Release Packages](#)

Section 4 [Component Release Files Specification](#) contains the authoritative documentation about the structure of release files containing data that represents SNOMED CT components ([concepts](#), [descriptions](#) and [relationships](#)).

Section 5 [Reference Set Release Files Specification](#) contains the authoritative documentation about the structure of release files that represent [SNOMED CT reference sets](#).


4.2. Release Type Options

Using Full and Snapshot Releases

Meeting Version Access Requirements

A key decision when designing any solution intended to provide to SNOMED CT is whether it should support import files from the [full release](#), the current [snapshot release](#) or both. The determining factor when considering these options is the range of SNOMED CT versions that to which the solution needs to provide access. [Table 4.2-1](#) identifies the release type import options that can be used to meet particular requirements to for access to different sets of SNOMED CT versions. The release type options are summarized in the sections following this table.

Table 4.2-1: Release Type Import Options to Support Version Access Requirements

Requirements for access to SNOMED CT versions	Release Type Import Options	Notes
Access to the current version of SNOMED CT only	<ul style="list-style-type: none"> Recommended: Import Snapshot Release Only 	
Access to the current version of SNOMED CT and full details of changes since the previous version	<ul style="list-style-type: none"> Recommended: Import Full and Current Snapshot Release 	Full details of changes since the previous version requires access to the previous versions of all components that were changed in the current release. As a result the same release type import options apply to both these sets of requirements  .
Access to the current version of SNOMED CT and one previous version	<ul style="list-style-type: none"> Minimal: Import Current and Previous Snapshot Release Alternative: Import Full Release Only 	

Access to the current version of SNOMED CT and more than one previous version	<ul style="list-style-type: none"> Recommended: Import Full and Current Snapshot Release Alternative: Import Full Release Only 	If access to more than two versions is required, it is easier to support access to all versions. As a result the same release type import options apply to both these sets of requirements ² .
Access to the current version of SNOMED CT and all previous versions		

Import Current Snapshot Release Only

Description

- A single set of tables is created for all the release files that need to be imported.
- Data is imported into these table from the current snapshot release.
- The database is optimized by appropriate additional indexes.
- Access to common combinations of data from multiple tables may be facilitated by creating database views, procedures and functions.

Advantages

- Simple solution which performs well.
- Supports to access the current snapshot release and current delta view.

Disadvantages

- No access to previous snapshot views.
- Cannot access to the previous state of components in the current delta view.

Import Current and Previous Snapshot Releases

Description

- Two sets of tables are created for all the release files that need to be imported.
- Data from the current snapshot release is imported into one of these sets of tables
- Data from the previous snapshot release is imported into the other set of tables.
- The database is optimized by appropriate additional indexes.
- Access to common combinations of data from multiple tables may be facilitated by creating a sets of database views, procedures and functions applicable to each set of tables.
- Access to information about changes to data between the two snapshot release may be facilitated by additional views, procedures and functions that combine or compare data from the two versions.

Advantages

- Simple duplication of the single snapshot view which performs well.
- Supports to access the current and previous snapshot releases and current and previous delta views.
- Supports access to previous state of components in the current delta view.

Disadvantages

- No access to snapshot views prior to the previous version.
- Heavy use of disk space as a result of duplication of rows that are identical in both snapshots. Each snapshot takes more than 80% of the space required by the full release.
- Not extensible because this approach is not realistic for multiple versions due to a linear increase in redundant use of disk space.

Import Full Release Only

Description

- A single set of tables is created for all the release files that need to be imported.
- Data is imported into these table from the full release.
- The database is optimized by appropriate additional indexes.
- Access to data in specific versions is facilitated by virtual [snapshot views](#) that can be accessed in the same way as database tables³.
- Access to common combinations of data from snapshot views of tables may be facilitated by creating database views, procedures and functions.
- Access to information about changes to data between any two snapshot release may be facilitated by additional views, procedures and functions that combine or compare data from those versions.

Advantages

- Access to the complete release history of all versions of a [SNOMED CT Edition](#).
- Efficient use of disk space requiring only 20% more disk space than importing a single snapshot release.
- Able to access snapshot views for any date.
- Able to access delta views for any date range.

Disadvantages

- Virtual views performs less well than a native database tables because the content of a view is the result of query on a database table.

Import Full and Current Snapshot Releases

- Two set of tables are created for all the release files that need to be imported.
- Data is imported into these table from the full release and current snapshot release.
- The database is optimized by appropriate additional indexes.
- Access to data in specific versions is facilitated by virtual [snapshot views](#) that can be accessed in the same way as database tables³.
- Access to common combinations of data from the snapshot tables or from the snapshot views of full release tables may be facilitated by creating database views, procedures and functions.
- Access to information about changes to data between any two snapshot release may be facilitated by additional views, procedures and functions that combine or compare data from those versions.

Advantages

- Access to the complete release history of all versions of a [SNOMED CT Edition](#).
- High performance access to the current snapshot view using the snapshot release table rather than a virtual snapshot view.
 - This is a significant advantage as the current snapshot is the most commonly used view.
- Able to access snapshot views for any date.
- Able to access delta views for any date range.

Disadvantages

- Requires approximately 80% more disk space than only importing the full release.
 - However, unlike the use of multiple snapshot tables. this disk space increase with each new release is determined by numbers of additions and changes rather and the only redundancy is a single version snapshot.

i The example SNOMED CT database is an example of the "Import Full and Snapshot Release" option. It includes a current snapshot table and a full release table for each of the release files. The current snapshot is accessed directly through the snapshot tables, while all snapshot views for any date between 31 January 2002 and the current release date are accessed as dynamic views. It also provides access to delta views showing changes between any two dates since the first release of SNOMED CT.

Importing the Delta Release

There is no need to import the delta release when importing either the full or snapshot releases. This is because rows that have changed in a release can be identified using their *effectiveTimes*. So it is possible to extract the most recent [delta view](#) from a snapshot release and all [delta views](#) can be extracted from a full release.

There is one potential use case for importing a delta release and that is to update a database containing the previous release to current release. A full release table for a specified previous release date can be updated by simply inserting all the rows from a the relevant delta release file for the period starting immediately after the previous release date up to and including a new release date.

In practice, it may be more efficient to start with an empty database and import the latest full release. However, the delta release update option may be useful where there is requirement to retain the operational integrity of the database during the update process. Future plans for more frequent or "continuous" updates to the SNOMED CT International Edition may also benefit from updates using delta releases of small number of changes or additions.

-
- 1** In theory, the first requirement for full details of changes could be met by a trimmed version of the previous snapshot from which rows that are unchanged in the current snapshot have been removed. However, this would complicate both the import process and the process of querying the data.
 - 2** In theory, requirements for a limited set of previous version views could also be met by importing multiple snapshot releases. However, importing a snapshot uses roughly 80% of the disk space. Performance advantages may in some cases make this approach worthwhile for two versions but it is not a scalable approach.
 - 3** For further information on snapshot views see [4.6. Enabling Versioned Views](#). [[a](#) [b](#)]


4.3. Data Type Options

All data in SNOMED CT release files conforms to one of a set of data types defined in the Release File Specification ([3.1.2 Release File Data Types](#)). The database tables that will hold release file data should be designed in ways that consistently map the general data type characteristics in the SNOMED CT specification to appropriate data types in the database. The data should be represented consistently using an appropriate data type supported by the database.



[Table 4.3-1](#) shows how each release file data type is represented in the MySQL example database. Possible alternative data type mappings are shown where appropriate with notes explaining the rationale for the preferred choice **1**

Table 4.3-1: Mapping from Release Files Data Types to MySQL Data Types

Release File Data Type	Description	DB Data Type	Possible Alternatives	Notes
------------------------	-------------	--------------	-----------------------	-------

SCTID	A SNOMED CT identifier, between 6 and 18 digits long, as described in 6.2 SCTID Representation.	BIGINT	VARCHAR(18) CHAR(18)	<p>Performance tests comparing BIGINT (a 64-bit Integer) and VARCHAR(18) show that BIGINT consistently performs better for almost all types of required access to the SNOMED CT database. A test set of queries took 66% of the time to complete with some tasks completed in less than half the time.</p> <p>CHAR(18) would also be a possible storage form but requires 2 bytes more than storage per SCTID than BIGINT. Whereas on average VARCHAR uses less storage because most SCTIDs are significantly shorter than the maximum length of 18 characters.</p>
UUID	<p>A Universally Unique Identifier is a 128-bit unsigned generated using a standard algorithm.</p> <ul style="list-style-type: none"> • UUIDs are represented as strings of hexadecimal characters split by - characters as points specified by the UUID standard. 	CHAR(36)	BINARY(16) CHAR(32)	<p>Performance tests comparing CHAR(36) and BINARY(16) indicated that CHAR(36) consistently performs significantly better than BINARY in the types of queries used in the SNOMED CT database.</p> <p>BINARY only uses 16 bytes compared with 36 required for CHAR(36).</p> <p>CHAR(36) enables UUIDs to be read and rendered without requiring additional processing to match the standard string representation of UUIDs.</p> <p>CHAR(32) would also be a possible storage form but requires processing to and from the standard string representation of UUIDs.</p>
Integer	A 32-bit signed integer.	INT		<p>Some integer columns may use far fewer than 32-bits (4 bytes) and in future some might require more. However, currently no integer values used in release files exceed the range of a 32-bit signed integer (except SCTIDs which are treated separately). Therefore, for consistency and simplicity the INT options is applied to all integer columns.</p>
String	UTF-8 text of a specified length.			<div style="border: 1px solid gray; padding: 5px; margin-bottom: 5px;">  All strings values use utf8mb4 character set. </div> <p>VARCHAR uses less storage for strings of limited length.</p> <p>TEXT offers flexible solution for longer strings.</p> <p>CHAR could be used for strings of known length.</p>
	If <i>length</i> is specified as no more than 200 characters	VARCHAR(<i>length</i>)	CHAR(<i>length</i>)	
	If <i>length</i> is unspecified and could potentially be more than that 200 characters	TEXT		
Boolean	A Boolean value, represented as one of two possible integer values (1 = true, 0 = false).	TINYINT	CHAR(1)	<p>TINYINT uses only a single byte and it thus the most economic way to store a 0 or 1 value.</p> <p>Although CHAR(1) could be used it offers no advantages.</p>

Time	A date and time format expressed as a text string in line the basic representation specified in the ISO 8601 standard . (i.e. <code>YYYYMMDD</code> or <code>YYYYMMDDTHHMMSSZ</code>)	DATETIME	TIMESTAMP VARCHAR(14) CHAR(8)	DATETIME allows date (and or time) storage in most compact form (5 bytes in MySQL 8.x). Comparison, differences, and flexible output formatting are also supported. TIMESTAMP also supports date and time, only using 4-bytes. However, this does not permit dates after 2038 and the extra byte in DATETIME completely removes the limitation. VARCHAR(14) would allow date and time in ISO format <code>YYYYMMDDhhmmdd</code> but requires more storage and performs less well than DATETIME. CHAR(8) would be sufficient for dates without times. This is all that is currently required for the <i>effectiveTime</i> field but potential uses of the Time data type would not be covered. It also uses 3 bytes per date for storage when compared with DATETIME.
-------------	--	----------	-------------------------------------	---

 If you are applying this guide to a different SQL implementation, you may need to modify some or all of these data type mappings based on assessment of the performance and storage characteristics of the available data types .


 The process for determining the data types used in the example database was as follows:

1. Identify the range of data types capable of representing all possible values based on the characteristics of the general data types defined in the SNOMED CT specifications.
2. For the SNOMED CT data types used in primary keys (**SCTID**, **UUID** and **Time**), assess candidate datatypes based on the performance of views and queries that use these keys for retrieval and database joins.
3. For the string data type, consider different options for columns of different lengths and where relevant indexing requirements differ.
4. Where the above factors do not distinguish between options, choose the data type that uses the least storage space.

[a b]

4.4. Database Table Naming

Introduction

The International Edition of SNOMED CT contains three release types (Full, Snapshot and Delta) and each of these release types include 20 files (2019-07-31 release). Each of those files has a distinct structure representing either a type of component (e.g. concept, relationship or description) or a type of reference set. SNOMED CT Extensions contain additional files and most of these conform to the same structure as one of the International Edition release files .

When designing a database to accommodate SNOMED CT release files, decisions need to be made about the names to give to each of the database tables. One option is to give the tables exactly the same names as the release files they represent. However, analysis of the release file naming conventions indicates that these conventions are not directly applicable to table names.

SNOMED CT release file naming conventions include some elements that represent information about the provenance, language and release date of a specific file. This information is useful and in some cases essential as a way of distinguishing releases files. However, this information is neither essential nor helpful when naming tables that may contain data from different SNOMED CT versions, editions and extensions.

The release file naming conventions do however include some essential elements that relate directly to the specification of the nature and structure of the data they contain. The following sections provide a summary of the

release file naming conventions, identify the elements in release file names that are relevant to database table naming and describe a set rules that can be applied to derive consistent table names from release file names.

Release File Naming

All SNOMED CT release file are named in accordance with the [3.3.2 Release File Naming Convention](#). The naming conventions result in names that can be decomposed into parts as illustrated by examples with color coding in [Table 4.4-1](#).

Table 4.4-1: Illustrations of the Release File Naming Conventions

Description of the pattern or file illustrated	Example release file names
General pattern ²	prefix_ [refsetPattern] componentType_ [refsetType] [extensionName] releaseType [-language]_country_releaseDate.txt
International edition full release concepts file for 2019-07-31	sct2_Concept_Full_INT_20190731.txt
International edition snapshot release english descriptions file	sct2_Description_Snapshot-en_INT_20190731.txt
Spanish extension full release spanish descriptions file	sct2_Description_SpanishExtensionFull-es_INT_20190430.txt
International edition snapshot release extended maps reference set file	der2_iissccRefset_ExtendedMapSnapshot_INT_20190731.txt
Spanish extension full release spanish language reference set file	der2_cRefset_LanguageSpanishExtensionFull-es_INT_20190430.txt
International edition snapshot release english language reference set file	der2_cRefset_LanguageSnapshot-en_INT_20190731.txt

File Name Element Relevance to Table Names

[Table 4.4-2](#) identifies the elements of the release file naming pattern that are relevant to the naming of the database tables containing content from those files. It also outlines the reasons why some elements that form an important part of the release file names can or should be omitted from the relevant database table names.

Table 4.4-2: Relevance of File Name Pattern Elements to Database Table Names

Filename Element	Relevant to Table Name	Explanation
prefix	No	The prefix sct2 or der2 distinguishes components from derivatives (refsets). This information is present in the componentType and refsetType.
refsetPattern	No	This information relates to the datatypes of additional columns in the file and the table. The table structure includes the required columns so there is no reason to include this in the table name.
componentType	Yes	This is essential as it indicates either the type of components represented in the table or that this is a reference set
refsetType	Yes	This is essential to distinguish the tables representing different reference set types (and not present in other file names).
extensionName	No	This is not required as data from extensions files should be included in the same tables as the equivalent data from the international release. Individual records maintained in extensions can be distinguished by moduleId
releaseType	Yes	This is essential if importing data from both the full and snapshot release. However, since this is a fundamental grouping, it is probably sensible for this to be a prefix to the table name. Otherwise with long table names this key distinction may be easier to miss. A short prefix denoting release types with a convention that also allows database views to be named in a similar consistent manner is recommended.
language	No	This is not applicable to the description table name. All descriptions should be accommodated in a single table with the languageCode column indicating the language of the associated term. Similarly it is not applied to a language reference set table name. All language reference sets should be accommodated in a single table with the refsetId column indicating the language and dialect of each language preference.

Filename Element	Relevant to Table Name	Explanation
country	No	This is not required in the table name as the country or other point of origin of the components and reference set members is indicated by the moduleId.
releaseDate	No	This is not required as data from many releases is included in the full release file tables. In the case of the snapshot it would be possible to include the date of the snapshot in the table name. However this is not recommended because, as noted in 4.2. Release Type Options multiple sets of tables representing different snapshot releases multiply the required storage capacity required.

Deriving Table Names from Release File Names

The analysis in Table 4.4-2, identifies three elements in the release file name that are relevant to table names. There are various ways in which table names could be derived by combining these elements and one of these is shown in Table 4.4-3. The end result (shown in Table 4.4-4) is a set of table names that:

- Are as short as possible while clearly identifying:
 - The release type from which they are derived
 - The component or reference set type specification to which they conform
- Are not specific to a particular SNOMED CT release or edition.

Note

The rules shown here are those applied to the example SNOMED CT database. Alternative table naming patterns may be preferred by those developing their own SNOMED CT database. However, it is important to ensure that the table naming pattern should be consistently applicable to all release files. Furthermore, it also should be readily applicable to any additional reference set types that may be added to future releases of the International Edition (or included in other SNOMED CT editions and or extensions).

Table 4.4-3: Rules Applied to Release File Names to Generate Table Name for the Example Database

Start with file name pattern	prefix_[refsetPattern]componentType_[refsetType][extensionName]releaseType[-language]_country_releaseDate.txt
Remove element that are not required	componentType_[refsetType]releaseType
Make release type the prefix	releaseType_componentType_[refsetType]
Abbreviate the prefix to 4 characters (full or snap)	rtyp_componentType_[refsetType]

Table 4.4-4: Results of Mapping Release File Names to Example Database Table Names

List of Release File Name	List of Corresponding Table Names in the Example Database
---------------------------	---

sct2_Concept_Full_INT_20190731.txt	▶ full_concept
sct2_Description_Full-en_INT_20190731.txt	▶ full_description
der2_cRefset_AssociationFull_INT_20190731.txt	▶ full_refset_Association
der2_cRefset_AttributeValueFull_INT_20190731.txt	▶ full_refset_AttributeValue
der2_ciRefset_DescriptionTypeFull_INT_20190731.txt	▶ full_refset_DescriptionType
der2_iisssccRefset_ExtendedMapFull_INT_20190731.txt	▶ full_refset_ExtendedMap
der2_cRefset_LanguageFull-en_INT_20190731.txt	▶ full_refset_Language
der2_ssRefset_ModuleDependencyFull_INT_20190731.txt	▶ full_refset_ModuleDependency
der2_cisssccRefset_MRCMAAttributeDomainFull_INT_20190731.txt	▶ full_refset_MRCMAAttributeDomain
der2_sssccRefset_MRCMAAttributeRangeFull_INT_20190731.txt	▶ full_refset_MRCMAAttributeRange
der2_sssssssRefset_MRCMDomainFull_INT_20190731.txt	▶ full_refset_MRCMDomain
der2_cRefset_MRCModuleScopeFull_INT_20190731.txt	▶ full_refset_MRCModuleScope
sct2_sRefset_OWLEExpressionFull_INT_20190731.txt	▶ full_refset_OWLEExpression
der2_cciRefset_RefsetDescriptorFull_INT_20190731.txt	▶ full_refset_RefsetDescriptor
der2_Refset_SimpleFull_INT_20190731.txt	▶ full_refset_Simple
der2_sRefset_SimpleMapFull_INT_20190731.txt	▶ full_refset_SimpleMap
sct2_Relationship_Full_INT_20190731.txt	▶ full_relationship
sct2_StatedRelationship_Full_INT_20190731.txt	▶ full_statedRelationship
sct2_TextDefinition_Full-en_INT_20190731.txt	▶ full_textDefinition
sct2_Concept_Snapshot_INT_20190731.txt	▶ snap_concept
sct2_Description_Snapshot-en_INT_20190731.txt	▶ snap_description
... list continues for all Snapshot release files	... list continues for all the snap_ tables

- 1 A few files in an extension may conform to a reference set that has been defined by the organization responsible for that extension.
- 2 Pattern elements in square brackets [] are optional depending on file type.

4.5. Database Table Design

Introduction

Every SNOMED CT release file conforms to a formal specification which defines the names and data types of each of the columns in the file. These specifications are subsections of the [Release File Specification](#).

- [4.2 File Format Specifications](#)
- [5.2 Reference Set Types](#)

The preceding chapters addressed the representation of SNOMED CT data types in a database and naming the individual tables into which release data will be loaded. This section considers two remaining design decisions related to database table design, names to be applied to the columns in the tables and keys and indexes that should be added to support speedy access to the data.

Column Names

The column names used in the release files are the formally specified names and these should be used as the column names in the relevant database tables.

Column Data Types

Each column in a database table should be assigned a data type by consistently by applying defined mappings between data types defined in SNOMED CT specifications [1](#) and those available in your database environment [2](#).

Additional Column Options

In most cases there should be no need to add additional columns to the release tables. However, as discussed in [4.4 .3. Snapshot View Options](#), some approaches to optimization of snapshot views may require an additional column. If any additional columns are added they should comply with the following good practice guidelines.

Additional columns must:

1. only be added for technical purposes such as optimizing database performance;
2. only be added after the columns that represent standard SNOMED CT release files data;
3. be given names that clearly distinguish them from the columns that represent data from the release files;
4. not be presented to a user of the database in ways that suggest they are part of the terminology.

Primary Keys

All database tables representing SNOMED CT release data should use a primary key that combines *id* and *effectiveTime*.

- This combined primary key is:
 - Essential for full release tables as *id* alone is not unique.
 - Recommended for tables representing data from a snapshot release for overall consistency [3](#).

Additional Indexes

Additional indexes are required to support rapid access to interrelated data (for example the descriptions and relationships associated with an identified concept). [Table 4.5-1](#) outlines the rationale for each of the additional indexes used to improve performance of specific tables in the SNOMED CT example database [4](#). Further indexes or revisions of these indexes may also be useful to further enhance performance. [Table 4.5-1](#) is also intended as a starting point for the developing SNOMED CT solutions in other database environments. However, the benefits of adding particular indexes will depend on the characteristics of the database server. Therefore, some of these indexes may not be required and other indexing strategies may be more effective at improving performance.

Table 4.5-1: Additional Indexes for Specific Tables

Database Table	Index Name	Index Columns	Rationale for this Index
(full or snap)_description	description_concept	conceptId	Find descriptions for concept.
	description_lang	conceptId,languageCode	Find descriptions with specific language code for concept.
	description_term	term (fulltext) 5	Search for terms.
(full or snap)_relationship	relationship_destination	destinationId,typeId,sourceId	Find concepts with relationships of a specified type of which a specified concept is the destinationId (value or supertype) or find relationships with a specific combination of destination, type and source.
	relationship_source	sourceId,typeId,destinationId	Find concepts with relationships of a specified type of which a specified concept is the destinationId (defined concept or subtype) or find relationships with a specific combination of source, type and destination.
(full or snap)_statedRelationship	statedRelationship_dest	destinationId,typeId,sourceId	Find concepts with stated relationships of a specified type of which a specified concept is the destinationId (value or supertype) or find relationships with a specific combination of destination, type and source.

	statedRelationship_source	sourceId,typeId,destinationId	Find concepts with relationships of a specified type of which a specified concept is the destinationId (defined concept or subtype) or find relationships with a specific combination of source, type and destination.
(full or snap)_textDefinition	textDefinition_concept	conceptId	Find text definitions for concept.
	textDefinition_language	conceptId,languageCode	Find text definitions with specific language code for concept.
	textDefinition_term	term (fulltext)	Search for terms in text definitions.
(full or snap)_refset_ [REFSETTYPE]	[REFSETTYPE]_c	referencedComponentId	Find rows in any reference set of type [REFSETTYPE] that refer to a specified referenced component.
	[REFSETTYPE]_rc	refsetId,referencedComponentId	Find rows in an identified reference set of type [REFSETTYPE] that refer to a specified referenced component.
(full or snap)_refset_Extended Map	ExtendedMap_map	refsetId,mapTarget	Find map records in a specified mapping reference set for a particular mapTarget. Find all concepts that have a map to a particular mapTarget in a specified mapping reference set.
(full or snap)_refset_SimpleMap	SimpleMap_map	refsetId,mapTarget	Find map records in a specified mapping reference set for a particular mapTarget. Find all concepts that have a map to a particular mapTarget in a specified mapping reference set.
(full or snap)_refset_MRCMAtributeDomain	MRCMAtributeDomain_domain	domainId	Find attribute domain information for a specified domain.

- 1 SNOMED CT data types are defined in section 3.1.2 Release File Data Types of the SNOMED CT Release File Specifications.
- 2 See recommendation on the approach to data type mapping in section 4.3. Data Type Options.
- 3 A primary key consisting only of the *id* is a potential alternative for tables representing data from a snapshot release.
- 4 To avoid slowing the data import process additional indexes are not added to the database tables until after all text files have been imported.
- 5 Full text indexes for terms allow effective searching. However, unless the database is correctly configured, short words, abbreviations and stop words may prevent effective indexing of common clinical terms. For further details refer to A.7.1 Required MySQL Configuration Settings.

4.6. Enabling Versioned Views

As noted in 4.2. Release Type Options, SNOMED CT data can be imported into database tables from snapshot release files, from full release files or from both these sets of release files. The objective of importing the data into the database is to provide effective access to useful views of that data. In practical terms this means facilitating access to some or all of the views summarized in Table 4.6-1.

Table 4.6-1: Summary of Versioned Views of SNOMED CT Components and Reference Set Members

View	Content Description	Use Cases	Value
Current Snapshot	The most recent version of each SNOMED CT component and reference set member.	<ul style="list-style-type: none"> All practical uses of the current version of SNOMED CT 	Essential for any use of SNOMED CT

Retrospective Snapshot	The most recent version of each SNOMED CT component and reference set member released prior to a specified earlier snapshot time.	<ul style="list-style-type: none"> • A baseline against which to review of changes to SNOMED CT during or after installing a new release. • Review of health records taking account of the version of SNOMED CT at the time the data was recorded. • Comparative analysis of health record data collected at different times taking account of changes to SNOMED CT. 	Valuable
Most Recent Delta View	The latest version of each SNOMED CT component and reference set member added, changed or inactivated in the most recent release. Typically, all these items with have an effectiveTime equal to the most recent release date. However, in cases where interim releases are made available between releases, the most recent delta view may be specified as including all items with an effectiveTime after the previous major release date.	<ul style="list-style-type: none"> • Identification of changes to SNOMED CT arising from the most recent release. 	Valuable as an indicator of recent changes
Other Delta Views	The versions of each SNOMED CT component and reference set member added, changed or inactivated after a specified delta start time and at or before a specified delta end time.	<ul style="list-style-type: none"> • Identification of changes to SNOMED CT over a period of time. 	Useful for longer term monitoring of changes.
Delta Views with Details of Changes	The content of a specified delta view combined with the retrospective snapshot view of SNOMED CT components and reference set members in the delta view at the specified delta start time.	<ul style="list-style-type: none"> • Reviewing full details of changes to SNOMED CT between two releases or over a period of time. • Assessing and managing the impact of updates to SNOMED CT. 	Required for effective change management

Table 4.6-2 summarizes the way in which different release type options affect the ability to access particular snapshot and delta views of SNOMED CT data. Importing the snapshot release supports direct access to the current snapshot view and query access to the most recent delta view. The full release provides access to all snapshot and delta views but is likely to perform slightly less well with the current snapshot view. Importing both full and current snapshot releases offers all the advantages of importing the full release and also provides direct access to the current snapshot view. This combined option requires more storage capacity but may be worthwhile because the current snapshot is likely to be the most commonly used view.

Table 4.6-2: Summary of Versioned View Access Capabilities Depending on Release Types Imported

<i>Views Supported</i>	<i>Release Types Imported</i>		
	Current Snapshot	Full	Full & Current Snapshot
Current Snapshot View	Direct	Query	Direct
Retrospective Snapshot Views	Not supported	Query	Query
Most Recent Delta View	Query	Query	Query
Other Delta Views	Not supported	Query	Query
Delta Views with Details of Changes	Not supported	Query	Query

The following subsections explore specific mechanisms that can be used to deliver these views in a relational database.

See also [Appendix C: Release Types and Versioned Views](#)

4.6.1. Versioned View Queries

Current Snapshot Queries

Extracting the current snapshot from a full release table requires a query that can identify the rows with the most recent effectiveTime for each uniquely identified component or reference set member. The general form of a current snapshot view query is shown below.

General Snapshot Query - Replace full_tableName with Name of a Full Release Table

```
select * from full_tableName tbl
       where tbl.effectiveTime = (select max(sub.effectiveTime) from full_tableName
sub
                                where sub.id =
tbl.id);
```

Including Other Constraints in a Snapshot Query

It may be tempting to write queries that add criteria specific to a particular query within the structure of a general snapshot query. However if this is done, it must be done with care because additional conditions may cause incorrect results.

Additional criteria to be applied to snapshot view must be added to the outer query to deliver the expected results. Otherwise they may inadvertently exclude the most recent component from the snapshot and thus leading to a misleading result.

- ❗ For example, in the following query the check for the active status is included in the nested query. This will lead to *the most recent active version of each component* being selected. The result of this query will therefore include an earlier active version of any component that is now inactive. A similar issue may also occur with other criteria [1](#).

General Snapshot Query - With Error Due to Added Condition in Nested Query

```
select * from full_tableName tbl
       where tbl.effectiveTime = (select max(sub.effectiveTime) from
full_tableName sub
                                where sub.id =
tbl.id and sub.active=1);
```

- ✅ The query below corrects the error in the shown above. This query will return components that are active in the current snapshot view. It will not return any components that are inactive in the current snapshot.

General Snapshot Query for Active Components in the Snapshot

```
select * from full_tableName tbl
       where tbl.active=1 and tbl.effectiveTime = (select max(sub.effectiveTi
me) from full_tableName sub
                                where sub.id =
tbl.id);
```

Retrospective Snapshot Queries

Queries for earlier snapshot views require an additional condition so that only versions with an effectiveTime that is equal to or earlier than (i.e. less than) the date of the snapshot. In this case, it is correct to include this condition in the nested query because the objective is to constrain the maximum effectiveTime to that the subquery returns. This ensures that the outer query does not return component versions added after the specified snapshot time.

Retrospective Snapshot Query for Snapshot as at 2019-01-31

```
select * from full_tableName tbl
      where tbl.effectiveTime = (select max(sub.effectiveTime) from full_tableName
                                sub
                                where sub.id = tbl.id
                                and sub.effectiveTime<='20190131');
```

Most Recent Delta View Query

The most recent delta view is usually considered to be all the rows in a table with an effectiveTime equal to the release date.² The general form of a query for the most recent delta view is shown below. Note that this query can be applied either to full release tables or the current snapshot release tables.

Most Recent Delta View Query for 2019-07-31 release. Replace tableName with Name of a Full or Snapshot Release Table for 2019-07-31 Release

```
select * from tableName tbl
      where tbl.effectiveTime = '20190731';
```

Specified Period Delta View Query

A more general purpose approach to delta views is to include all changes between two specified dates (or times). This can be applied to the period between two releases or to a defined period during which multiple changes may have occurred to the same component. The general form of a delta view query for a specified period is shown below.

General Most Recent Delta View Query - for Change After 2019-01-31 and On or Before 2019-07-31

```
select * from full_tableName tbl
      where tbl.effectiveTime > '20190131' and tbl.effectiveTime <= '20190731' ;
```

Note

This query requires the effectiveTime to be **greater than** the startDate and **less than or equal** to the endDate. This avoids double counting items in two consecutive periods. This means that ranges can be specified to start on one release date and end on another release date without counting changes that occurred on the first release date.

Delta View with Details of Changes

It is also possible to create an enhanced delta view that not only shows which components have changed but allows the pre-change state of that component to be seen. From a practical perspective this simply combines a delta view for a range of dates with a retrospective snapshot view for the delta view start date.

Delta View with Details of the Component Prior to the Changes

```

select * from full_tableName tbl
       where tbl.effectiveTime > '20190131' and tbl.effectiveTime <= '20190731'
union
select * from full_tableName tbl
       where tbl.effectiveTime = (select max(sub.effectiveTime) from full_tableName
sub
                                where sub.id = tbl.id
and sub.effectiveTime<='20190131')
and tbl.id IN (select id from full_tableName
               where effectiveTime > '20190131' and
               effectiveTime <= '20190731')
order by id;

```

- 1 For more details about potential snapshot view query errors see [C.3. Common Mistakes with Snapshot Generation](#).
- 2 This interpretation of "the most recent delta" depends on the practice of periodic releases with all rows added since the last release assigned the effectiveTime of the release. However, in cases where frequent interim releases are made, it may be more accurate to consider the "the most recent delta" to consist of all rows with an effectiveTime greater than the previous release date and less than or equal to the current release date. In this case, all delta view queries would to follow the form of the *Specified Date Range Delta View Query* with a start and end date.

4.6.2. Versioned Database Table Views

The previous section described queries that can be applied to database tables to access snapshot and delta views of individual database tables. However, meaningful access to SNOMED CT requires querying of interrelated data in multiple database tables including concepts, descriptions, relationships and reference sets. In most cases, the data from each of those tables needs to be from a snapshot view for the same date. This implies these composite need to derive information from snapshot queries applied to the table rather than directly accessing the table. Consequently, what should be relatively simple queries rapidly become complex, making them hard to understand and prone to errors.

Fortunately, relational databases provide solutions that reduce this complexity by allowing a query to be used to define a database view. These defined views can then be queried in exactly the same way as a database table. The result of this is that if all the required snapshot and delta views are defined for every table in a SNOMED CT database, it is possible to write relatively simple queries that return useful combinations of data from snapshot views of different tables.

Representing Snapshots as Database Views

The examples below two snapshot database views of a full release table. The first is current snapshot view and the second a is snapshot for 2019-01-31.

Creating a Current Snapshot View from the Full Concept Table

```
CREATE VIEW snap_concept as (select * from full_concept tbl
    where tbl.effectiveTime = (select max(sub.effectiveTime) from full_concept
    sub
    where sub.id =
tbl.id));
```

Creating a Retrospective Snapshot View for 2019-01-31 from the Full Concept Table

```
CREATE VIEW snap20190131_concept as (select * from full_concept tbl
    where tbl.effectiveTime = (select max(sub.effectiveTime) from full_concept
    sub
    where sub.id = tbl.id
and sub.effectiveTime<='20190131');
```

Having created those two views it is then possible to write queries like the examples below to display data from these snapshot views.

Creating a Current Snapshot View from the Full Concept Table

```
select * from snap_concept where id in (3859001,3704008);
```

Creating a Retrospective Snapshot View for 2019-01-31 from the Full Concept Table

```
select * from snap20190131_concept where id in (3859001,3704008);
```

Comparing the results of these two queries shows that in January 2019 both concepts changed in the 2019-07-31 release. The concept with id 3859001 was made inactive and the definitionStatusId of concept with id 3704008 was changed from 900000000000074008 (primitive) to 900000000000073002 (defined).

Creating Configurable Snapshot Views

It is possible to create snapshot views of every release file for every SNOMED CT release of SNOMED CT. However, this would result in over 600 distinct snapshot table views, most of which would rarely be used. A more practical solution is to create views for the current snapshot and two or three retrospective views with a configurable snapshot date. The SNOMED CT example database includes two retrospective snapshot views with a snapshotTime value in an identified row in a configuration table specifying the snapshot date.

The definition of one of these views is shown here. The other view (snap2_concept) has the same definition except that it specifies `cfg`.`id` = 2 rather than `cfg`.`id` = 1 so it refers to a different row in the configuration table. Having two independently configurable snapshots, allows queries to be written than compare different snapshots. Because the same configurable views are provided for all full release tables, changes to the snapshotTime for a view apply simultaneously to all those components and reference set members.

Example of Configurable Retrospective View Definition

```
CREATE VIEW `snap1_concept` AS select `tbl`.`id` AS `id`,`tbl`.`effectiveTime` AS
`effectiveTime`,`tbl`.`active` AS `active`,`tbl`.`moduleId` AS
`moduleId`,`tbl`.`definitionStatusId` AS `definitionStatusId` from `full_concept`
`tbl` where (`tbl`.`effectiveTime` = (select max(`sub`.`effectiveTime`) from
(`full_concept` `sub` join `config_settings` `cfg`) where ((`sub`.`id` = `tbl`.`id`)
and (`cfg`.`id` = 1) and (`sub`.`effectiveTime` <= `cfg`.`snapshotTime`))));
```

Creating Configurable Delta Views

The SNOMED CT example database also includes two configurable delta views. These are created and configured in the same way as the configurable snapshot views. However in this case, deltaStartTime and deltaEndTime values in the identified configuration table row specify the delta period.

Example of Configurable Retrospective View Definition


```
CREATE VIEW `delta1_concept` AS select `tbl`.`id` AS `id`,`tbl`.`effectiveTime` AS
`effectiveTime`,`tbl`.`active` AS `active`,`tbl`.`moduleId` AS
`moduleId`,`tbl`.`definitionStatusId` AS `definitionStatusId` from (`full_concept`
`tbl` join `config_settings` `cfg`) where ((`cfg`.`id` = 1) and
(`tbl`.`effectiveTime` <= `cfg`.`deltaEndTime`) and (`tbl`.`effectiveTime` >
`cfg`.`deltaStartTime`));
```

4.6.3. Optimizing Versioned Table Views

Using Separate Snapshot Tables

One of the points identified in the previous section is that snapshot queries and database views are less likely to perform as well as direct access to database tables. As shown in [Table 4.6.3-1](#) on the SNOMED CT example database confirm that there is a substantial performance difference between these two approaches.

Table 4.6.3-1: Testing Performance of Queries on Snapshot Tables and Snapshot Views

	Snapshot Table seconds	Snapshot Views  seconds	Performance Ratio
Read 1 million rows from relationship snapshot	1.52	11.06	15%
Read 1 million rows from description snapshot	3.57	12.73	28%
Read all rows from concept snapshot	0.66	2.45	26%
Total time for all operations above	5.75	25.74	22%
Advanced test reading 10,000 relationships and with joins to descriptions and language reference set for the fully specified names of source, type and target concept	2.34	4.70	50%

Based on these findings the most effective way to optimize access to a snapshot view, is to replace the use of database views with snapshot tables. Representing the a snapshot with tables, rather than using a database view, adds roughly 2.6 Gb to the storage requirements for the example database.

The current snapshot view is essential and is used for most interactions with the database. Therefore, the performance enhancements justify use of the additional disk space required to store the current snapshot in separate tables. If there are specific reasons for extensive access to one or two retrospective snapshots, it might

also be worthwhile representing those snapshots in separate tables. However, it would not be worthwhile to apply the same approach to the full range of less frequently used retrospective snapshots. Therefore, if the snapshot views defined in 4.6.2 do not perform sufficiently well, it is worth considering ways to optimize snapshot access.

Unlike the views described earlier, the optimization methods described below require additional columns to be added to each of the full release tables. After importing release files, the tables are processed to generate values for the additional columns and this data is used to simplify the snapshot view queries by avoiding the need for nested queries.

Snapshot Flags Optimization Method

Overview

A column is added to each full release table. This column is used to represent flags that indicate which snapshot view each row is included in.

Practical Example

1. A single 64-bit integer column called *flag* is added to all full release tables with a default value of 0 (zero).
2. A distinct number which is a power of 2 between 2^0 and 2^{63} is assigned to each required retrospective snapshot time.
3. The *flag* column in each row is set to the sum of the values of all the snapshots in which that row appears.
 - To be precise this means that a specific bit in the *flag* value is set if the row is part of a particular snapshot and is not set if it is not part of that snapshot.
4. Once this process is complete, it is possible to select the rows of a retrospective snapshot with a simple query that tests the relevant bit in the *flag* column value.
 - This avoids the need for the nested query required to identify rows that are part of a snapshot.

The example SQL below illustrates the process of flag setting. In practice, while this query works it is not very efficient for several reasons. A more efficient approach would be to use a stored procedure that computes the full set of flags applicable to each row. This approach would allow the flag column in each row to be updated once rather than requiring a separate update for each snapshot view.

A configurable snapshot view can then be created tests the appropriate bit rather than requiring a nested query.

Illustrative Example of a Flag Setting Query

```

SET SQL_SAFE_UPDATES=0;
update full_tableName tbl
  set flag=flag | 1
  where tbl.effectiveTime=(select max(sub.effectiveTime) from copy_full_tableName sub
  where sub.id=tbl.id and sub.effectiveTime<='20190731');
update full_tableName tbl
  set flag=flag | 2
  where tbl.effectiveTime=(select max(sub.effectiveTime) from copy_full_tableName sub
  where sub.id=tbl.id and sub.effectiveTime<='20190131');
update full_tableName tbl
  set flag=flag | 4
  where tbl.effectiveTime=(select max(sub.effectiveTime) from copy_full_tableName sub
  where sub.id=tbl.id and sub.effectiveTime<='20180731');
update full_tableName tbl
  set flag=flag | 8
  where tbl.effectiveTime=(select max(sub.effectiveTime) from copy_full_tableName sub
  where sub.id=tbl.id and sub.effectiveTime<='20180131');
SET SQL_SAFE_UPDATES=1;

```

Once the flags are set, a query such as the one below can be used to return component versions that are part of a particular snapshot view. The example query returns row in which the flag the second bit (value 2) is set. Based on the settings in the query above this means it would include rows that are part of the 2019-01-31 snapshot view of this table

Example of a Snapshot Query Using the Flag Method

```
select * from full_tableName where flag & 2;
```

Performance

Limited testing of this optimization approach indicates that it is between 2 and 3 times faster than the unoptimized snapshot views. Direct access to a snapshot table is still twice as fast as this optimized approach.

Storage Requirements

The full release files in the 2019-07-31 release contain a total of approximately 16 million rows. If flags are added each of these rows will require a further 8 bytes of storage. No additional indexes are required to support this optimization. As a result the overall increase in storage requirements to support this optimization is less than 150 Mb.

As described here the approach is limited to 64 snapshot times. This is probably more than sufficient for most practical requirements. However, it and could be extended by adding an another flag column or by changing the data type of the flag column to binary.

Disadvantages

- The process of setting the flags required for this approach adds significantly to the time taken to build the database.
- Adding an additional column to every table means that queries using "SELECT * FROM ..." will return a *flag* column that is not part of the original SNOMED CT data.
- The flag values are technically essential to the process but this may not be apparent to anyone exploring the database.
- Significantly slower than direct access to snapshot tables. Optimum current snapshot performance still requires the snapshot table.

Advantages

- A significant improvement in retrospective snapshot performance compared with unoptimized tables.
- Minimal impact on disk capacity (adds less than 5% to the size of the full release tables).
- Provides a fallback option for the current snapshot view if storage capacity is limited.

Superseded Time Optimization Method

Overview

An additional datetime column is added to each row. This column is used to represent the time at when a row was replaced by the next version of that component or reference set member.

Practical Example

1. A single datetime column called *supersededTime* is added to all full release tables with a default value of a long distant future date (e.g. 9999-12-31).
2. Each full table is queried to establish the sequence of versions of each component or reference set member in effectiveTime order.

3. The supersededTime of a component that have been updated is set to the *effectiveTime* of the immediately following version of that component
4. Once this process is complete each component is part of all snapshot views with a snapshot time greater than or equal to its *effectiveTime* and less than its *supersededTime*.
 - This can be tested without the need for a nested query.

The example SQL below illustrates the process of flag setting. In practice, while this query works it is not very efficient for several reasons. A more efficient approach would be to use a stored procedure that computes the full set of flags applicable to each row. This approach would allow the flag column in each row to be updated once rather than requiring a separate update for each snapshot view.

Illustrative Example of Queries Setting the supersededTime Value

```
-- Create temporary table for the supersededTime values
CREATE TEMPORARY TABLE tmp (id CHAR(36) NOT NULL,effectiveTime
DATETIME,supersededTime DATETIME, PRIMARY KEY (id,effectiveTime));

-- Compute the supersededTime values for each combination of id+effectiveTime and add
these to the temporary file
INSERT INTO tmp SELECT tbl.id, tbl.effectiveTime, (SELECT IFNULL(MIN(sub.effectiveTim
e),DATE "99991231") FROM full_tableName sub
WHERE tbl.id=sub.id AND tbl.effectiveTime<sub.effectiveTime) supersededTime FROM
full_tableName tbl;

-- Apply the appropriate supersededTime values to each row in the full table
UPDATE full_tableName tbl
JOIN tmp
SET tbl.supersededTime=tmp.supersededTime
WHERE tmp.id=tbl.id AND tmp.effectiveTime=tbl.effectiveTime;
```

Once the superseded time values are set, a query such as the one below can be used to return component versions that are part of a particular snapshot view. The example query returns row in which the flag the second bit (value 2) is set. Based on the settings in the query above this means it would include rows that are part of the 2019-01-31 snapshot view of this table

Example of a Current Snapshot Query Using the supersededTime Method

```
-- This query assumes the default supersededTime is 9999-12-31
SELECT *
FROM full_tableName
WHERE supersededTime = DATE '99991231';
```

Example of a Retrospective Snapshot Query Using the supersededTime Method

```
-- This query assumes that [snapshotTime] is replaced by the required snapshotTime
SELECT *
FROM full_tableName
WHERE [snapshotTime] >= effectiveTime AND [snapshotTime] < supersededTime;
```

Performance

Past experience indicates that this approach is 2 times faster than the unoptimized snapshot views². However, this figure varies depending on the complexity of the queries it is used in. Direct access to a snapshot table is between 2 and 3 times as fast as this optimized approach.

Storage Requirements

The full release files in the 2019-07-31 release contain a total of approximately 16 million rows. If a datetime column is added, each of these rows will require a further 5 bytes of storage. Database designs using this additional column also included additional indexes including `supersededTime`³. As a result, the storage to fully support this approach required an additional 750 Mb.

Disadvantages

- The process of setting the `supersededTime` required for this approach adds significantly to the time taken to build the database.
- Adding an additional column to every table means that queries using "SELECT * FROM ..." will return the `supersededTime` column that is not part of the original SNOMED CT data.
- The `supersededTime` values are technically essential to the process but this may not be apparent to anyone exploring the database.
- The use of `supersededTime` together with associated indexes increases the storage capacity required for the full release tables by approximately 20%.
- Significantly slower than direct access to snapshot tables and since the introduction of MySQL 8.0 are also slower than the snapshot flagging method.

Advantages

- An improvement in retrospective snapshot performance compared with unoptimized tables but is outperformed by the snapshot flagging method.
- A fallback option for the current snapshot view if storage capacity is limited but requires more storage than the snapshot flagging method.
- Supports an unlimited number of snapshot times.

¹ These views are defined using the general form described in 4.6.2.

² Previous tests in MySQL 5.7 were 3 times faster than unoptimized snapshot views. However, MySQL 8.0 seems to have enhanced the performance of the unoptimized queries without significantly improving the results of this approach to optimization.

³ The performance impact of removing some of these indexes was not tested, so it is unclear if benefits could still be delivered by this approach without these indexes.

4.7. Enabling Subtype Testing

Requirement

The SNOMED CT subtype polyhierarchy is an important resource which allows retrieval of all the concepts that are subtypes of a specified concept. The full hierarchy is represented by subtype (is a) relationships between each concept and its proximal supertypes. Therefore, to determine whether a concept is a subtype of another concept a chain of these subtypes must be followed. It is possible to follow these chain in a relational database environment but the queries to achieve this are complex and often perform poorly.

Solution

Fortunately, there is a well-understood way to simplify and speed up the testing process. This requires the creation of a resource known as a transitive closure table. A transitive closure table includes direct relationships between every concept and all of its subtypes and supertypes. This makes it possible to test whether a concept is a subtype of another concept looking for a single row in that table. Similarly it makes it easy to access all the subtypes (or supertypes) of a concept with a query on a single table.

Creating a Transitive Closure File

SNOMED International provides a Perl script that reads the snapshot relationships file. By processing this file, it rapidly generates the transitive closure of all the subtype relationships and then saves this as a file. The result is a two-column file (subtypeld and supertypeld) containing more than 6.5 million rows. This transitive closure file can then be read into a database table in the same way as the release files.

Using a Transitive Closure File

By using the transitive closure table very simple queries can:

- Test if concept A is a subtype of concept B
- List all supertypes of concept A
- List all subtypes of concept A

Computing Proximal Primitive Supertypes

The transitive closure can also be used to determine which supertype concept or concepts are essential to the definition of a concept. These concepts are known as the [proximal primitive supertypes](#) of a concept. *Proximal primitive supertypes* can be computed as follows.

1. Use the transitive closure table to get the set of all concepts that are supertypes of concept **A**
 - Call this set: Set-B
2. Create a subset of Set-B containing only those concepts with definitionStatusId= 900000000000074008 | Primitive
 - Call this set: Set-C
3. Create a subset of Set-C containing only those concepts that have no *subtypes* that are also in Set-C¹
 - The resulting set represents the primitive supertypes of concept **A**.

¹ Concepts in Set-C that also have subtypes in Set-C are primitive supertypes of concept **A** but they are not **proximal primitive supertypes** because these subtype(s) are more specific concepts that are also primitive supertypes of concept **A**.

4.8. Composite Views

Section 4.6. [Enabling Versioned Views](#) considered the value of views of individual database tables. The objective of these views was to facilitate access to a range of useful snapshot and delta views. This section introduces the idea of defining composite views that access interrelated sets of data from different release files. The objective of composite views is to make it easier to access appropriate sets of data.

Example

Apart from the concept identifier, the concept table contains no useful human-readable data about a concept. Therefore, simply selecting data from a snapshot view of the concept table is unlikely to be useful.

To display appropriate human-readable information about a concept information is required from two other tables:

- The human-readable information about a concept is in the description table or view (e.g. `snap_description`)
- Information about which descriptions are preferred or acceptable in given language data is in a language refset table or view (e.g. `snap_refset_language`).

To display human-readable information about the way a concept is defined data is required from three other tables:

- The defining relationships are in the relationships table or view (e.g. `snap_relationship`)
- Human-readable display of the type and value specified in the relationship requires data from the description table or view (e.g. `snap_description`)
- Information about which descriptions are preferred or acceptable in given language data is in a language refset table or view (e.g. `snap_refset_language`).

The following subsections describe some general characteristics of composite views and introduce some of the composite views included in the SNOMED CT example database.

4.8.1. General Characteristics of Composite Views

Data Sources for Composite Views

Composite views need to get data from snapshot views. In most cases the requirements will be met by use of the current snapshot view¹ of each of the relevant tables. However, when reviewing past data the relevant retrospective views will need to be accessed. For this reason, consideration should be given to creating similar composite views for each supported snapshot view².

Most composite views should to gather all the required data from the table views for the same snapshot as illustrated in [Table 4.8.1-1](#).

Table 4.8.1-1: Source Views for Data in Composite Views of Different Snapshots

Composite View	snap_pref	snap1_pref	snap2_pref
Description View	snap_description	snap1_description	snap2_description
Language Refset View	snap_refset_language	snap1_refset_language	snap2_refset_language

Composite views may themselves gather data from other composite views. For example as shown in [Table 4.8.1-2](#) gets preferred term data from the preferred term composite views shown above.

Table 4.8.1-2: Composite View Including Data from Another Composite View

Composite View	snap_rel_pref	snap1_rel_pref	snap2_rel_pref
Concept View	snap_concept	snap1_concept	snap2_concept
Relationship View	snap_relationship	snap1_relationship	snap2_relationship
Preferred Term View	snap_pref	snap1_pref	snap2_pref

Composite views designed to support review of changes may gather data from different views as illustrated in [Table 4.8.1-3](#).

Table 4.8.1-1: Source Views for Data in Historical Composite Views of Different Delta Views

Composite View	delta_inactive_concepts	delta1_inactive_concepts	delta2_inactive_concepts
Concept View	delta_concept	delta1_concept	delta2_concept

Association Refset View	snap_refset_association	snap_refset_association	snap_refset_association
Attribute Value Refset View	snap_refset_attributevalue	snap_refset_attributevalue	snap_refset_attributevalue
Preferred Term View	snap_pref	snap_pref	snap_pref
Fully Specified Name View ³	snap_fsn	snap_fsn	snap_fsn

Representation of Composite Views

Composite views should be represented as database views rather than a physical database tables. Composite views denormalize data by combining the same data in different views therefore attempts to represent composite views as database tables is likely to rapidly multiply the size of the database. The example below is just one of many cases where creating concrete database tables to accommodate composite views might seem an attractive idea. However, pursuing this would create redundant data with few benefits, a major impact on storage requirements and a significantly more complex maintenance process when reviewing and installing future release packages. In contrast, representing composite views as database views, ensures the data is derived in real-time from tables representing the authoritative content of the full and/or snapshot release files.

i Example

Most English language descriptions are either preferred or acceptable in both US and GB english. Therefore instantiating tables that represent the sets of preferred and acceptable terms in either or both dialects would not only duplicate much of the data in that table but would require even more space to duplicate the relevant indexes. In addition to the impact of disk space, data duplicated in these composite tables would need updating to take account of new releases.

- ¹ As noted in [4.6.3. Optimizing Versioned Table Views](#) the current snapshot may be represented as tables or database views. While this may make a difference to performance it does not make any difference to the design of composite views.
- ² In the SNOMED CT example database, most composite views have been created for the current snapshot (snap) and for both of the configurable retrospective snapshot views (snap1 and snap2). However, composite views that access either the transitive closure (snap_transclose) or proximal primitives (snap_proxprim) are not supported for the retrospective snapshots. This is because those tables are at present not available for the current snapshot view.
- ³ The views snap_fsn, snap1_fsn and snap2_fsn are composite views similar to snap_pref but return the fully specified name rather than the preferred term.

4.8.2. Composite Description Views



Composite description views enable access to appropriate individual descriptions or sets of descriptions in a specific language or dialect. There are two sets of composite description views. Views in the first of these sets are designed to facilitate selecting descriptions associated with one or more identified concepts. Views in the other set are designed to enable searching descriptions to find concepts.

Views that Facilitate Selecting Concept Descriptions

For each snapshot view the SNOMED CT example database includes four views designed for selecting specific sets of descriptions for one or more specified concepts. The characteristics of each of these views are shown in [Table 4.8.2-1](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.2-1](#). To create each of the views named in the table, the placeholders for {typeid} and {acceptabilityId} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.2-1](#) demonstrates the use of these views to show all the active descriptions of a specified concept that are acceptable or preferred according to the language reference set referenced by the configuration file.

Table 4.8.2-1: Composite Description Views for Selecting Concept Descriptions

Name 	Description	Specific Settings	
		{typeId}	{acceptabilityId}
snap_fsn	This view selects the fully specified name of a concept (identified by conceptId)	= 9000000000000000000003001	= 9000000000000548007
snap_pref	This view selects the preferred synonym of a concept (identified by conceptId)	= 9000000000000013009	= 9000000000000548007
snap_syn	This view selects other acceptable synonyms of a concept (identified by conceptId)	= 9000000000000013009	= 9000000000000549004
snap_synall	This view selects all synonyms of a concept (identified by conceptId)	= 9000000000000013009	IN (9000000000000548007, 9000000000000549004) 

Template 4.8.2-1: Description Selection Composite View Template

```

CREATE VIEW `snap_{name}` AS
(SELECT `d`.* FROM `snap_description` `d`
  JOIN `snap_refset_Language` `rs` ON `d`.`id` = `rs`.`referencedComponentId`
  JOIN `config_settings` `cfg` ON `rs`.`refSetId` = `cfg`.`languageId`
WHERE `d`.`active` = 1 AND `d`.`typeId` {typeId}
  AND `rs`.`active` = 1 AND `rs`.`acceptabilityId` {acceptabilityId}
  AND `cfg`.`id`=0);

```

Example 4.8.2-1: Selecting Description Data for a Concept

SQL Query			
<pre> Select conceptId,'FSN',id,term from snap_fsn where conceptId=95570007 UNION Select conceptId,'Pref',id,term from snap_pref where conceptId=95570007 UNION Select conceptId,'Syn',id,term from snap_syn where conceptId=95570007; </pre>			
Result			
conceptId	FSN	id	term
95570007	FSN	839752010	Kidney stone (disorder)
95570007	Pref	158296018	Kidney stone
95570007	Syn	158297010	Renal stone
95570007	Syn	158298017	Nephrolith
95570007	Syn	158299013	Renal calculus
95570007	Syn	512193015	Calculus of kidney
95570007	Syn	512194014	Nephrolithiasis
95570007	Syn	512195010	Kidney calculus

Views that Facilitate Searching for Concepts

For each snapshot view the SNOMED CT example database includes two views that are designed to facilitate searches for concepts associated with terms that match specified criteria. These views require both the description and the associated concept to be active in the chosen snapshot. This avoids the need to filter out inactive concepts from the search results³. The specific characteristics of these views is shown in [Table 4.8.2-2](#) and a general template for the SQL definitions of these views is shown in [Template 2](#). To create the each of the views named in the table, the placeholder for {typeId} needs to be replaced with values in the *Specific Settings* column of the table.

[Example 2](#) show a query that searches the snap_syn_search_active view using MySQL's boolean full text search. This search method requires all the words preceded by + (plus) to be included and excludes all words preceded by - (minus). The results are shown together with the fully specified name of the concept (looked up using the snap_fsn view). Although this is not a user-friendly way to specify a searches, the example SQL code illustrates the technical power of this search technique. Other search techniques can also be applied to the search views and additional options for enhancing searches are discussed in [4.9.4. Search Procedures](#).

Table 4.8.2-2: Composite Description Views that Facilitate Searching for Concepts

Name ¹	Description	Specific Settings
		{typeId}
snap_syn_search_active	This view includes active preferred and acceptable synonyms of active concepts. It excludes fully specified names and also excludes all descriptions associated with concepts that are inactive in the specified snapshot.	= 900000000000013009
snap_term_search_active	This view includes active preferred and acceptable synonyms of active concepts.	IN (90000000000003001, 900000000000013009) ⁴

Template 4.8.2-2: Description Search Composite View Template

```
CREATE VIEW `snap_syn_search_active` AS
(SELECT `d`.*,`rs`.`acceptabilityId` FROM `snap_description` `d`
JOIN `snap_refset_Language` `rs` ON `d`.`id` = `rs`.`referencedComponentId`
JOIN `snap_concept` `c` ON `c`.`id` = `d`.`conceptId`
JOIN `config_settings` `cfg` ON `rs`.`refSetId` = `cfg`.`languageId`
WHERE `d`.`active` = 1 AND `d`.`typeId` {typeId}
AND `rs`.`active` = 1
AND `c`.`active` = 1
AND `cfg`.`id`=0);
```

Usage Example

Example 4.8.2-2: Searching Terms to Find Concepts

SQL Query

```
SELECT `s`.`conceptId`,`s`.`term` 'matching term`,`f`.`term` `FSN` FROM
`snap_syn_search_active` `s`
JOIN `snap_fsn` `f` ON `f`.`conceptId`=`s`.`conceptId`
WHERE MATCH (`s`.`term`)
AGAINST ('+acute +anterior +myocardial +infarction -ecg
-old -ekg' IN BOOLEAN MODE) ORDER BY length(`f`.`term`),length(`s`.`term`);
```

Result		
conceptId	matching term	FSN
54329005	Acute anterior myocardial infarction	Acute myocardial infarction of anterior wall (disorder)
54329005	Acute myocardial infarction of anterior wall	Acute myocardial infarction of anterior wall (disorder)
703164000	Acute anterior ST segment elevation myocardial infarction	Acute ST segment elevation myocardial infarction of anterior wall (disorder)
703164000	Acute STEMI (ST elevation myocardial infarction) of anterior wall	Acute ST segment elevation myocardial infarction of anterior wall (disorder)
703164000	Acute ST segment elevation myocardial infarction of anterior wall	Acute ST segment elevation myocardial infarction of anterior wall (disorder)
703252002	Acute myocardial infarction of anterior wall involving right ventricle	Acute myocardial infarction of anterior wall involving right ventricle (disorder)
703252002	Acute myocardial infarction of anterior wall with right ventricular involvement	Acute myocardial infarction of anterior wall involving right ventricle (disorder)
703165004	Acute ST segment elevation myocardial infarction of anterior wall involving right ventricle	Acute ST segment elevation myocardial infarction of anterior wall involving right ventricle (disorder)
703165004	Acute anterior ST segment elevation myocardial infarction with right ventricular involvement	Acute ST segment elevation myocardial infarction of anterior wall involving right ventricle (disorder)
703165004	Acute STEMI (ST elevation myocardial infarction) of anterior wall with right ventricular involvement	Acute ST segment elevation myocardial infarction of anterior wall involving right ventricle (disorder)
285981000119103	Acute ST segment elevation myocardial infarction involving left anterior descending coronary artery	Acute ST segment elevation myocardial infarction involving left anterior descending coronary artery (disorder)

- 1 The prefix snap is replaced by snap1 or snap2 for retrospective views. [a b]
- 2 An alternative way to represent snap_synall is to remove the acceptability condition. The link to the language refset and the test for the `rs`.`active` condition must be retained to ensure only descriptions in the relevant language refset are returned.
- 3 Requirements for searches that need to include inactive concepts can be run against the concept description selection views.
- 4 Alternatively remove the typeId condition to permit all types to be searched.

4.8.3. Composite Subtype Hierarchy Views

Composite subtype hierarchy views enable selection of supertypes and subtypes of specified *concepts* accompanied by human-readable terms for each of the selected concepts. The hierarchy views include [supertype parents](#), [subtype children](#), [supertype ancestors](#) and [subtype descendants](#) of specified *concepts*. More specialized views are also included to list the [proximal primitive parents](#) of a specified *concepts* and to list *concepts* that share a specified *proximal primitive parent*. All these views use include a human-readable term (either the fully specified name or the preferred synonym) for each concept listed in the output [1](#).

Supertype Parent and Subtype Child Views

For each snapshot view the SNOMED CT example database includes two views that select the [supertype parents](#) of a specified concept and two views that select the [subtype children](#) of a specified concept. These views select the id and either the fully specified name or preferred synonym each parent or child concept. The characteristics of each of these views are shown in [Table 4.8.3-1](#) and a general template for the SQL definitions of these views is shown

in [Template 4.8.3-1](#). To create each of the views named in the table, the placeholders for {termtype} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.3-1](#) demonstrates the use of these views to select the id and preferred term for a specified concept and all of its supertype parents and subtype children.

Table 4.8.3-1: Composite Views of Supertype Parents and Subtype Children

Name ²	Description	Specific Settings
		{termtype}
snap_rel_parent_fsn	Selects the id and fully specified name of each supertype parent of a concept specified by conceptId.	fsn
snap_rel_parent_pref	Selects the id and preferred synonym of each supertype parent of a concept specified by conceptId.	pref
snap_rel_child_fsn	Selects the id and fully specified name of each subtype child of a concept specified by conceptId.	fsn
snap_rel_child_pref	Selects the id and preferred synonym of each subtype child of a concept specified by conceptId.	pref

Template 4.8.3-1: SQL Templates for Composite Views of Supertype Parents and Subtype Children

```

-- Supertype Parent View
CREATE VIEW `snap_rel_parent_{termtype}` AS
  SELECT `r`.`targetId` AS `id`, `d`.`term` AS `term`, `r`.`sourceId` AS
  `conceptId`
    FROM `snap_relationship` `r` JOIN `snap_{termtype}` `d` ON `r`.`targetId` =
  `d`.`conceptId`
    WHERE `r`.`active` = 1 AND `r`.`typeId` = 116680003;
-- Subtype Child View: Differences are `sourceId` changed to `targetId` and
`targetId` changed to `sourceId` as shown below
CREATE VIEW `snap_rel_child_{termtype}` AS
  SELECT `r`.`sourceId` AS `id`, `d`.`term` AS `term`, `r`.`destinationId` AS
  `conceptId`
    FROM `snap_relationship` `r` JOIN `snap_{termtype}` `d` ON `r`.`sourceId` =
  `d`.`conceptId`
    WHERE `r`.`active` = 1 AND `r`.`typeId` = 116680003;

```

Example 4.8.3-1: Selecting Supertype Parents and Subtype Children

SQL Query		
<pre> Select "Concept", conceptid, term from snap_pref where conceptId=6025007 UNION Select "Supertype Parent", id, term from snap_rel_parent_pref where conceptId=6025007 UNION Select "Subtype Child", id, term from snap_rel_child_pref where conceptId=6025007; </pre>		
Result		
Concept	id	term
Concept	6025007	Laparoscopic appendectomy
Supertype Parent	51316009	Laparoscopic procedure

Supertype Parent	80146 002	Appendectomy
Supertype Parent	26427 4002	Endoscopic operation
Supertype Parent	44058 8003	Endoscopic procedure on appendix
Subtype Child	17404 1007	Laparoscopic emergency appendectomy
Subtype Child	30758 1005	Laparoscopic interval appendectomy
Subtype Child	70887 6004	Robot assisted laparoscopic appendectomy

Transitive Closure Views of Supertype Ancestors and Subtype Descendants

For each snapshot view the SNOMED CT example database includes two views that select the [supertype ancestors](#) of a specified concept and two views that select the [subtype descendants](#) of a specified concept. These views select the id and either the fully specified name or preferred synonym each ancestor or descendant concept. The characteristics of each of these views are shown in [Table 4.8.3-2](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.3-2](#). To create each of the views named in the table, the placeholders for {termtype} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.3-2](#) demonstrates the use of these views to select the id and preferred term for a specified concept and all of its supertype ancestors and subtype descendants.

Table 4.8.3-2: Transitive Closure Views of Supertype Ancestors and Subtype Descendants

Name	Description	Specific Settings
		{myph}
snap_tc_ancestor_fsn	Selects the id and fully specified name of each supertype ancestor of a concept specified by conceptId.	fsn
snap_tc_ancestor_pref	Selects the id and preferred synonym of each supertype ancestor of a concept specified by conceptId.	pref
snap_tc_descendant_fsn	Selects the id and fully specified name of each subtype descendant of a concept specified by conceptId.	fsn
snap_tc_descendant_pref	Selects the id and preferred synonym of each subtype descendant of a concept specified by conceptId.	pref

Template 4.8.3-2: SQL Templates for Composite Views of Supertype Ancestors and Subtype Descendants

```

-- Supertype Ancestor View: Differences are `sourceId` changed to `targetId` and
`targetId` changed to `sourceId` as shown below
CREATE VIEW `snap_tc_ancestor_{termtyp}` AS
(SELECT `r`.`supertypeId` `id`,`d`.`term` `term`,`r`.`subTypeId` `conceptId`
FROM `snap_transclose` `r`
JOIN `snap_{termtyp}` `d` ON (`r`.`supertypeId` = `d`.`conceptId`));

--- Subtype Descendant View
CREATE VIEW `snap_tc_descendant_{termtyp}` AS
(SELECT `r`.`subTypeId` `id`,`d`.`term` `term`,`r`.`supertypeId` `conceptId`
FROM `snap_transclose` `r`
JOIN `snap_{termtyp}` `d` ON (`r`.`subTypeId` = `d`.`conceptId`));

```

Example 4.8.3-2: Selecting Supertype Ancestors and Subtype Descendants

SQL Query

```

Select "Concept", conceptid, term from snap_pref where conceptId=6025007
UNION
Select "Ancestor", id, term from snap_tc_ancestor_pref where conceptId=6025007
UNION
Select "Descendant", id, term from snap_tc_descendant_pref where conceptId=6025007;

```

Result

Concept	conceptId	term
Concept	16001004	Otalgia
Ancestor	22253000	Pain
Ancestor	102957003	Neurological finding
Ancestor	106147001	Sensory nervous system finding
Ancestor	118234003	Finding by site
Ancestor	118236001	Ear and auditory finding
Ancestor	118254002	Finding of head and neck region
Ancestor	138875005	SNOMED CT Concept
Ancestor	247234006	Ear finding
Ancestor	276435006	Pain / sensation finding
Ancestor	279001004	Pain finding at anatomical site
Ancestor	297268004	Ear, nose and throat finding
Ancestor	301354004	Pain of ear structure
Ancestor	301857004	Finding of body region
Ancestor	404684003	Clinical finding
Ancestor	406122000	Head finding
Ancestor	699697007	Finding of sensation by site
Descendant	12336008	Referred otalgia
Descendant	74123003	Otogenic otalgia
Descendant	162356005	Earache symptoms
Descendant	162359003	Bilateral earache
Descendant	430879002	Posterior auricular pain

Descendant	1084561000119106	Bilateral referred otalgia of ears
Descendant	1089561000119107	Referred otalgia of left ear
Descendant	1092171000119100	Referred otalgia of right ear

Proximal Primitive Parent Views

For each snapshot view the SNOMED CT example database includes two views that select the **proximal primitive parents** of a specified concept and two views that select the **subtype children** of a specified concept. These views select the id and either the fully specified name or preferred synonym each parent or child concept. The characteristics of each of these views are shown in [Table 4.8.3-1](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.3-1](#). To create each of the views named in the table, the placeholders for {termtype} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.3-1](#) and [Example 4.8.3-4](#) demonstrate the use of these views to select the id and preferred term for a specified concept and all of its supertype parents and subtype children.

For each snapshot view the SNOMED CT example database includes The characteristics of each of these views are shown in [Table 4.8.3-3](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.3-3](#). To create each of the views named in the table, the placeholders for {myph} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.3-3](#) demonstrates the use of these views to select the id and preferred term for a specified concept and its *proximal primitive parents*. [Example 4.8.3-4](#) demonstrates the use of these views to select the id and preferred term for a specified concept and all the concepts that have this concept as a *proximal primitive parent*.

Table 4.8.3-3: Views of Proximal Primitive Supertype Ancestors and Concepts with a Specific Proximal Primitive Ancestor

Name ³	Description	Specific Settings
		{viewtype}
snap_pp_parent_fsn	Selects the id and fully specified name of each <i>proximal primitive parent</i> of a concept specified by conceptId.	fsn
snap_pp_parent_pref	Selects the id and preferred synonym of each <i>proximal primitive parent</i> of a concept specified by conceptId.	pref
snap_pp_child_fsn	Selects the id and fully specified name of each concept with a <i>proximal primitive parent</i> specified by conceptId.	fsn
snap_pp_child_pref	Selects the id and preferred synonym of each concept with a <i>proximal primitive parent</i> specified by conceptId.	pref

Template 4.8.3-3: SQL Templates for Proximal Primitive Supertype Views

```

-- Proximal primitive parents of a specified concept
CREATE VIEW `snap_pp_parent_{viewtype}` AS
(SELECT `r`.`supertypeId` `id`,`d`.`term` `term`,`r`.`subTypeId` `conceptId`
  FROM `snap_proximal_primitives` `r`
  JOIN `snap_{viewtype}` `d` ON (`r`.`supertypeId` = `d`.`conceptId`));

-- Concepts with a specified proximal primitive parent concept
CREATE VIEW `snap_pp_child_{viewtype}` AS
(SELECT `r`.`subTypeId` `id`,`d`.`term` `term`,`r`.`supertypeId` `conceptId`
  FROM `snap_proximal_primitives` `r`
  JOIN `snap_{viewtype}` `d` ON (`r`.`subTypeId` = `d`.`conceptId`));

```

Example 4.8.3-3: Selecting Proximal Primitive Parents of a Concept

SQL Query

```
--
Select "Concept", conceptid, term from snap_pref where conceptId=21522001
UNION
Select "Proximal Primitive Parent", id, term from snap_pp_parent_pref where
conceptId=21522001;
```

Result

Concept	conceptid	term
Concept	21522001	Abdominal pain
Proximal Primitive Parent	22253000	Pain

Example 4.8.3-4: Selecting Concepts with a Specified Proximal Primitive Parent

```
Select "Concept", conceptid, term from snap_pref where conceptId=22253000
UNION
Select "Concept with PP-Parent: 21522001|Pain|", id, term from snap_pp_child_pref
where conceptId=22253000;
```

Result

Concept	conceptid	term
Concept	22253000	Pain
Concept with PP-Parent: 21522001 Pain	4448006	Allergic headache
Concept with PP-Parent: 21522001 Pain	4568003	Retrosternal pain
Concept with PP-Parent: 21522001 Pain	6561007	Pain in urethra
Concept with PP-Parent: 21522001 Pain	10601006	Pain in lower limb
Concept with PP-Parent: 21522001 Pain	12584003	Bone pain
Concept with PP-Parent: 21522001 Pain	15803009	Bladder pain
Concept with PP-Parent: 21522001 Pain	16513000	Postcordotomy pain
Concept with PP-Parent: 21522001 Pain	18876004	Pain in finger
Concept with PP-Parent: 21522001 Pain	20793008	Scapulalgia
Concept with PP-Parent: 21522001 Pain	21522001	Abdominal pain
Concept with PP-Parent: 21522001 Pain	21545007	Tenalgia
Concept with PP-Parent: 21522001 Pain	29857009	Chest pain
Concept with PP-Parent: 21522001 Pain	30473006	Pain in pelvis

Concept with PP-Parent: 21522001 Pain	30989 003	Knee pain
... total of 240 rows returned ...		

- 1 The terms are displayed by using the [Composite Description Views](#) described in the previous section.
- 2 The prefix snap is replaced by snap1 or snap2 for retrospective views.
- 3 Transitive closure and proximal primitive views are only available for the current snapshot. [a b]

4.8.4. Composite Relationship Views

The composite relationship views display active, inferred defining relationships of specified *concepts* accompanied by human-readable terms for each the selected concepts [1](#).

Defining Relationship View

For each snapshot view the SNOMED CT example database there are two view of the active, inferred defining relationships. The views select the id and either the fully specified name or preferred synonym for the concept identified in each of the defining columns (sourceId, typeId and destinationId). The characteristics of these views are shown in [Table 4.8.4-1](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.4-1](#). To create each of the views named in the table, the placeholders for {termtype} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.4-1](#) demonstrates the use of these views to select the active defining relationships of a specified concept with the id and preferred term for each of the referenced concepts.

Table 4.8.4-1: Composite Views of Supertype Parents and Subtype Children

Name 2	Description	Specific Settings
		{termtype}
snap_rel_def_fsn	This view includes all active, inferred relationships of a concept specified by sourceId 3 . It selects the id and fully specified name for each of the concept identifiers (sourceId, typeId and destinationId) and the relationshipGroup number.	fsn
snap_rel_def_pref	This view includes all active, inferred relationships of a concept specified by sourceId 3 . It selects the id and fully specified name for each of the concept identifiers (sourceId, typeId and destinationId) and the relationshipGroup number.	pref

Template 4.8.4-1: SQL Templates for Composite Views of Defining Relationships

```
CREATE VIEW `snap_rel_def_{viewtype}` AS
(SELECT `r`.`sourceId` `sourceId`, `src`.`Term` `sourceTerm`, `r`.`typeId`
`typeId`, `typ`.`Term` `typeTerm`, `r`.`destinationId` `destinationId`, `dest`.`Term`
`destinationTerm`, `r`.`relationshipGroup` `relationshipGroup`
FROM (((`snap_relationship` `r`
JOIN `snap_{viewtype}` `src` ON ((`r`.`sourceId` = `src`.`conceptId`))) JOIN
`snap_{viewtype}` `typ` ON ((`r`.`typeId` = `typ`.`conceptId`))) JOIN
`snap_{viewtype}` `dest` ON ((`r`.`destinationId` = `dest`.`conceptId`))) WHERE
((`r`.`active` = 1) AND (`r`.`characteristicTypeId` = 900000000000011006)));
```

Example 4.8.4-1: Selecting Supertype Parents and Subtype Children

SQL Query

```
Select * from snap_rel_def_pref where sourceId=6025007;
```

Result

sourceId	sourceTerm	typeId	typeTerm	destinationId	destinationTerm	relationshipGroup
6025007	Laparoscopic appendectomy	11668003	Is a	51316009	Laparoscopic procedure	0
6025007	Laparoscopic appendectomy	11668003	Is a	80146002	Appendectomy	0
6025007	Laparoscopic appendectomy	11668003	Is a	264274002	Endoscopic operation	0
6025007	Laparoscopic appendectomy	11668003	Is a	440588003	Endoscopic procedure on appendix	0
6025007	Laparoscopic appendectomy	26068604	Method	129304002	Excision - action	1
6025007	Laparoscopic appendectomy	40581307	Procedure site - Direct	66754008	Appendix structure	1
6025007	Laparoscopic appendectomy	42539105	Using access device	86174004	Laparoscope	1

- 1 The terms are displayed by using the [Composite Description Views](#) described in an earlier section of this guide.
- 2 The prefix snap is replaced by snap1 or snap2 for retrospective views.
- 3 The selection criteria for any of these relationship views can also be specified by destinationId, typeId or by a combination of these identifiers. However, to see all the defining relationships of a specified concept, the sourceId should be used as this refers to the concept defined by the relationships. [a b]

4.8.5. Composite Historical Views

Inactive Concept Views

For each delta and snapshot view the SNOMED CT example database includes a view of inactive concepts. The characteristics of each of these views are shown in [Table 4.8.5-1](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.5-1](#). To create each of the views named in the table, the placeholders for {myph} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.5-1](#) demonstrates the use of these views to show all the active descriptions of a specified concept that are acceptable or preferred according to the language reference set referenced by the configuration file.

Table 4.8.5-1: Composite Views of Inactive Concepts with Related Concept Inactivation and Historical Association Refset Data

Name ¹	Description
delta_inactive_concepts	This view selects details of concepts that are inactive in the chosen delta or snapshot view. In addition to the concept id the fully specified name of the inactive concept is selected. The output of this view also includes the reason for activation and any historical associations between this inactive concept and an active concept. The reason for inactivation is shown as the preferred synonym for the concept representing the reason for inactivation in the concept inactivation reference set. The historical association is represented by the preferred synonym of the association reference set(s) and the fully specified name of the associated target concept. Where a concept has multiple active associations each of these reported as a separate row (the inactive concept and inactivation reason data is duplicated on each of these rows).

Template 4.8.5-1: SQL Definition of the Inactive Concepts View

```

CREATE VIEW delta_inactive_concepts AS
select `c`.`id`, `c`.`effectiveTime`, `c`.`active`, `c`.`definitionStatusId`,
`cf`.`term` 'FSN',
      `vp`.`term` 'reason', `arp`.`term` 'assoc_type', `atf`.`id` 'ref_conceptId',
`atf`.`term` 'ref_concept_FSN'
from `delta_concept` `c`
left join `snap_fsn` `cf` ON `cf`.`conceptid`=`c`.`id`
left outer join `snap_refset_attributevalue` `v` on
  `v`.`referencedComponentId`=`c`.`id`
      and `v`.`refsetId`=900000000000489007 and `v`.`active`=1
left outer join `snap_pref` `vp` on `vp`.`conceptid`=`v`.`valueid`
left outer join `snap_refset_association` `a` on `a`.`referencedComponentId`=`c`.`id`
and `a`.`refsetId` IN
  (900000000000528000, 900000000000523009, 900000000000527005,
900000000000526001,
  900000000000525002, 900000000000531004, 900000000000524003,
900000000000530003) and `a`.`active`=1
left outer join `snap_pref` `arp` on `arp`.`conceptid`=`a`.`refsetId`
left outer join `snap_fsn` `atf` on `atf`.`conceptid`=`a`.`targetComponentId`
where `c`.`active`=0
order by `c`.`id`;

```

Example 4.8.5-1: Selecting Inactive Concepts with Related Concepts Inactivation and Historical Association Refset Data

SQL Query								
<pre>SELECT * FROM delta_inactive_concepts;</pre>								
Result (example rows only)								
id	effectiveTime	active	definitionStatusId	FSN	reason	assoc_type	ref_conceptId	ref_concept_FSN
1192004	20190731	0	90000000000074008	Familial amyloid neuropathy, Finnish type (disorder)	Outdated	REPLACED BY	3757892013	Hereditary gelsolin amyloidosis (disorder)
1230003	20190731	0	90000000000074008	No diagnosis on Axis I (finding)	Outdated	REPLACED BY	677781011	Psychological finding (finding)
1427008	20190731	0	90000000000074008	Intraspinal abscess (disorder)	Duplicate	SAME AS	743297013	Spinal cord abscess (disorder)
2461007	20190731	0	90000000000074008	Tennis elbow test (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	3777085015	Lateral epicondylitis test (procedure)
2900003	20190731	0	90000000000074008	Hyperplasia of renal artery (disorder)	Ambiguous	POSSIBLY EQUIVALENT TO	3760067011	Fibromuscular dysplasia of wall of renal artery (disorder)
3105002	20190731	0	90000000000074008	Intron (finding)	Outdated	REPLACED BY	697643016	Finding related to molecular sequence data (finding)
3221003	20190731	0	90000000000074008	Ringer's solution (product)	Nonconformance to editorial policy component			

3734003	20190731	0	900000000000074008	Split thickness skin graft (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	3758568011	Split thickness graft of skin to skin (procedure)
4101004	20190731	0	900000000000074008	Revision of spinal pleurothecal shunt (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	618942015	Revision of spinal subarachnoid shunt (procedure)
4101004	20190731	0	900000000000074008	Revision of spinal pleurothecal shunt (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	618681017	Revision of subdural-pleural shunt (procedure)
4131005	20190731	0	900000000000074008	Implantation into pelvic region (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	2968044014	Procedure on pelvic region of trunk (procedure)
4131005	20190731	0	900000000000074008	Implantation into pelvic region (procedure)	Ambiguous	POSSIBLY EQUIVALENT TO	3756616019	Implantation procedure (procedure)
4518006	20190731	0	900000000000074008	Buthenal (substance)	Ambiguous	POSSIBLY EQUIVALENT TO	796984014	Crotonaldehyde (substance)
4919007	20190731	0	900000000000074008	Congenital protrusion (morphologic abnormality)	Duplicate	SAME AS	642112018	Protrusion (morphologic abnormality)
5034009	20190731	0	900000000000074008	Graft to hair-bearing skin (procedure)	Duplicate	SAME AS	3757739014	Hair bearing graft of skin to skin (procedure)

Inactive Descriptions

For each delta and snapshot view the SNOMED CT example database includes a view of inactive descriptions. The characteristics of each of these views are shown in [Table 4.8.5-2](#) and a general template for the SQL definitions of these views is shown in [Template 4.8.5-2](#). To create each of the views named in the table, the placeholders for {myph} need to be replaced with values in the *Specific Settings* column of the table.

[Example 4.8.5-2](#) demonstrates the use of these views to show all the active descriptions of a specified concept that are acceptable or preferred according to the language reference set referenced by the configuration file.

Table 4.8.5-2: Composite Views of Inactive Descriptions with Related Description Inactivation and Historical Association Refset Data

Name ²	Description
delta_inactive_descriptions	This view selects details of all descriptions that are inactive in the chosen delta or snapshot view. In addition to selecting the description data it also includes the active fully specified name of the related concept and the reason for activation. The reason for inactivation is shown as the preferred synonym for the concept representing the reason for inactivation in the description inactivation reference set.

Template 4.8.5-2: SQL Definition of the Inactive Descriptions View

```

CREATE VIEW delta_inactive_descriptions AS
select `d`.`id`, `d`.`effectiveTime`, `d`.`active`, `d`.`conceptid`, `d`.`term`
`term`,
    `df`.`term` `concept_fsn`, `c`.`active` `concept_active`, `vp`.`term`
`reason`
from `delta_description` `d`
left outer join `snap_fsn` `df` ON `df`.`conceptid`=`d`.`conceptid`
join `snap_concept` `c` ON `c`.`id`=`d`.`conceptid`
left outer join `snap_refset_attributevalue` `v` on
    `v`.`referencedComponentId`=`d`.`id`
    and `v`.`refsetId`=900000000000490003 and `v`.`active`=1
left outer join `snap_pref` `vp` on `vp`.`conceptid`=`v`.`valueid`
where `d`.`active`=0
order by `d`.`id`;
  
```

Example 4.8.5-2: Selecting Inactive Descriptions with Related Description Inactivation Refset Data

SQL Query							
<pre>SELECT * FROM delta_inactive_descriptions;</pre>							
Result (example rows only)							
id	effectiveTime	active	conceptid	term	concept_fsn	concept_active	reason
14132019	20190731	0	7938006	D-Arabinitol dehydrogenase	D-arabinitol 4-dehydrogenase (substance)	1	Nonconformance to editorial policy component
16101018	20190731	0	9156001	Embryo stage 1	Structure of embryo at stage 1 (body structure)	1	Nonconformance to editorial policy component
16837014	20190731	0	9631008	Rheumatoid spondylitis	Ankylosing spondylitis (disorder)	1	Not semantically equivalent component
17234017	20190731	0	9871000	D-Amino-acid acetyltransferase	D-amino-acid N-acetyltransferase (substance)	1	Nonconformance to editorial policy component
17525014	20190731	0	10043003	D-Alanine-alanyl-poly(glycerolphosphate) ligase	D-alanine-alanyl-poly(glycerolphosphate) ligase (substance)	1	Nonconformance to editorial policy component
17526010	20190731	0	10043003	D-Alanyl-alanyl-poly(glycerolphosphate)synthetase	D-alanine-alanyl-poly(glycerolphosphate) ligase (substance)	1	Nonconformance to editorial policy component
17527018	20190731	0	10043003	D-Alanine:membrane-acceptor ligase	D-alanine-alanyl-poly(glycerolphosphate) ligase (substance)	1	Nonconformance to editorial policy component
17615010	20190731	0	10093004	Anisakiasis due to Anisakis simplex	Anisakiasis caused by larva of Anisakis simplex (disorder)	1	Erroneous
20220015	20190731	0	11702002	bis-(p-Chlorophenyl) ethanol	Bis-(p-chlorophenyl) ethanol (substance)	1	Nonconformance to editorial policy component
20469015	20190731	0	1186003	Nannizzia	Genus Arthroderma (organism)	1	Not semantically equivalent component

4.9. Stored Procedures

4.9.1. General Characteristics of Stored Procedures

As noted in 4.6. [Enabling Versioned Views](#), database views allow useful queries to be saved and reused as though they were database tables. The SQL queries used in a view can be complex and can include data from other views such as those described in 4.8. [Composite Views](#). However, the definition of each view is defined by a single SQL query.

Stored procedures and functions provide another way to define reusable resources in a database. The key difference between these database views, stored procedures and functions are summarized in [Table 4.9.1-1](#)¹. From a practical perspective these differences enable stored procedures to facilitate some types of access to a SNOMED CT data that cannot be supported by using database views. The following subsections describe a few examples of stored procedures that are included in the SNOMED CT example database.

Table 4.9.1-1: Features of Views, Stored Procedures and Functions (in MySQL)

Feature	Database View	Stored Procedure	Stored Function
Enable definition of reusable resources that facilitate commonly required processes that access data without	✓	✓	✓
Defined by a single query	✓	✗	✗
Produce output that can queried in the same way as a database table	✓	✗	✗
Can be defined to output the results of a single SQL query	✓	✓	✗
Can be defined to output the results of one or more SQL queries	✗	✓	✗
Can be defined to add, delete or alter data in a table ²	✗	✓	✓
Can be defined to include transactional SQL statements ²	✗	✓	✗
Can create, alter or delete database tables, views, procedures or functions ²	✗	✓	✓
Can be defined with input parameters to be set when invoked with values that affect the results	✗	✓	✓
Can be defined to set values the values of one or more output parameters	✗	✓	✗
Can be defined to return a single value of a specified datatype	✗	✗	✓

¹ The features of stored procedures and functions shown in the table are those that apply to MySQL. Some of these features may differ in other database environments.

² Access to features that make changes to data or database resources may be limited by database security settings. [a b c]

4.9.2. Configuration Procedures

The SNOMED CT example database, has includes configuration settings that control configurable aspects of versioned table views (see 4.6.2. [Versioned Database Table Views](#)). The configuration settings also affect queries, composite views and procedures that refer to configurable versioned table views.

The configuration settings are represented by a database table called `config_settings` and each of the procedures described in this section either selects data from that table or updates data in the table. The details and default settings of the `config_settings` table are shown in [Table 4.9.2-1](#).

Table 4.9.2-1: Configuration Table Specification and initial settings

Column	Description	Datatype	Permitted Values	Default Values
id	Identifies the configuration setting and links to directly to view name prefixes <ul style="list-style-type: none"> id=0 refers to snap and delta views (the time values are fixed for these views) id=1 refers to snap1 and delta1 views id=2 refers to snap2 and delta2 views 	TINYINT	Integer in range 0-255	0, 1, 2
language Id	The refsetId of a language reference set.	BIGINT	Any language refsetId (e.g. 9000000000000509007, 9000000000000508004)	9000000000000509007
language Name	The name of the language represented by the language reference set identified by <code>languageId</code> .	VARCHAR (255)	The name of any language or dialect represented by a language refset (e.g. US English, GB English).	US English
snapshot Time	The <code>snapshotTime</code> for views with the relevant view name prefix [1] .	DATETIME	<ul style="list-style-type: none"> Id=0: Release date (fixed) Id>1: Any valid date. 	<ul style="list-style-type: none"> Id=0: Release date Id=1: Six months before release date Id=2: One year before release date.
deltaStartTime	The <code>effectiveTime</code> of a component or refset member row must be greater than <code>deltaStartTime</code> to be included in the delta view with the relevant prefix [1] .	DATETIME	<ul style="list-style-type: none"> Id=0: Six months before release date (fixed) Id>1: Any valid date before the release date. 	<ul style="list-style-type: none"> Id=0: Six months before release date Id=1: One year before release date Id=2: Eighteen months before release date.
deltaEndTime	The <code>effectiveTime</code> of a component or refset member row must be less than or equal to the <code>deltaStartTime</code> to be included in the delta view with the relevant prefix [1] .	DATETIME	<ul style="list-style-type: none"> Id=0: Release date (fixed) Id>1: Any valid date after the <code>deltaStartTime</code>. 	<ul style="list-style-type: none"> Release DateTime (fixed) Id=1: Six months before release date Id=2: One year before release date.

Show Configuration Procedure

The **showConfig** procedure selects and displays the configuration files data.

Example 4.9.2-1: Using the showConfig Procedure

SQL Call to Procedure


```
call showConfig();
```

Result ²

i d	languageId	language Name	refsetName ³	snapshotTime	deltaStartTime	deltaEndTime
0	9000000000000509007	US English	United States of America English language reference set (foundation metadata concept)	2019-07-31	2019-01-31	2019-07-31
1	9000000000000509007	US English	United States of America English language reference set (foundation metadata concept)	2019-01-31	2018-07-31	2019-01-31
2	9000000000000509007	US English	United States of America English language reference set (foundation metadata concept)	2018-07-31	2018-01-31	2018-07-31


Set Language

The **setLanguage** procedure sets the languageId and languageName for a configuration row specified by its identifier value.

-  Languages can only be set if the following conditions apply.
1. The concept identifying the language reference set is available in the database.
 2. The language abbreviation, language name and the identifier of the language reference set are in the config_language table.
 3. The identified language reference set is available in the snap_refset_language reference set table or view.

SQL Call to Procedure

```
call setDeltaRange(p_id,p_deltaStartTime,p_deltaEndTime);
```

Parameter	Description	Data type	Valid values	Example
p_id	The identifier of the configuration table view (also the number of the snapshot or delta view number to which the setting applies).	TINYINT	0, 1 or 2  The language setting can be changed for the id=0 row.	1
p_lang_code	The date after which changes will be included in the delta view.	VARCHAR(5)	Any value that MySQL recognizes as a date or date-time.	'en-GB'

Example Procedure Call

```
call setLanguage(1, 'en-GB');
```

Result

If p_lang_code does not refer to a language code in the config_language file or if no valid refset or refset members are found the procedure reports an error.

If the procedure succeeds, the language setting is changed but there is no output data. To check the result of the change, call showConfig() after resetConfig.

Set Snapshot Time

The **setDeltaRange** procedure sets the *snapshotTime* for a configuration row specified by its id value.

SQL Call to Procedure

```
call setDeltaRange(p_id,p_snapshotTime,p_deltaEndTime);
```


Parameter	Description	Data type	Valid values	Example
p_id	The identifier of the configuration table view (also the number of the snapshot or delta view number to which the setting applies).	TINYINT	1 or 2 (Note: Values less than 1 or greater than 2 will be treated as referring to row 2)	1
p_snapshotTime	The date for which the identified snapshot view will be computed.	DATE TIME	Any value that MySQL recognizes as a date or date-time.	'2017-07-31'

Example Procedure Call

```
call setSnapshotTime(1, '2017-07-31');
```

Result

The snapshotTime setting is changed but there is no output data. To check the result of the change, call showConfig() after resetConfig.

Set Delta Range

The **setDeltaRange** procedure sets the *deltaStartTime* and *deltaEndTime* for a configuration row specified by its id value.

SQL Call to Procedure

```
call setDeltaRange(p_id,p_deltaStartTime,p_deltaEndTime);
```

Parameter	Description	Data type	Valid values	Example
p_id	The identifier of the configuration table view (also the number of the snapshot or delta view number to which the setting applies).	TINYINT	1 or 2 (Note: Values less than 1 or greater than 2 will be treated as referring to row 2)	1
p_deltaStartTime	The date after which changes will be included in the delta view.	DATE TIME	Any value that MySQL recognizes as a date or date-time.	'2016-07-31'
p_deltaEndTime	The date on or before which changes with be included in the delta view.	DATE TIME	Any value that MySQL recognizes as a date or date-time.	'2019-01-31'

Example Procedure Call


```
call setDeltaRange(1, '2016-07-31', '2017-07-31');
```

Result

The deltaStartTime and deltaEndTime settings are changed but there is no output data. To check the result of the change, call showConfig() after resetConfig.

Reset Configuration

The **resetConfig** procedure resets all the configuration settings to the default values shown in [Table 4.9.2-1](#).

 The reset depends on the date times config_settings on the row with id=0 being unchanged. In particular, it assumes the snapshotTime of that row as the releaseDate. The other procedures described in this section do not change those values. However, if those values are changed by update queries the resetConfig procedure will not correctly reset the snapshot and delta times.

SQL Call to Procedure

```
call resetConfig();
```

Result

The reset is performed but there is no output data. To check the result of the reset, call showConfig() after resetConfig.


- 1 Internally all these configuration dates are stored as the time 23:59:59 on the stated date. This ensure all changes on the end date are included in snapshot and delta views while all changes on the start date are excluded from a delta view. [a b c]
- 2 The results shown here are those the initial default settings for the 2019-07-31 release.
- 3 The refsetName is selected by looking up the languageld in the snap_fsn view.

4.9.3. Language and Dialect Comparison Procedures

Terms in Languages

The **termsInLanguages** procedure displays the terms associated with a list of selected concepts in one or more specified languages or dialects. The concepts to be selected are specified by a comma-separated list of conceptIds and the languages in which they are to be displayed are specified by a comma-separated list of language codes.

Example 4.9.3-1: Example Use of termsInLanguages Procedure

SQL Call to Procedure			
<pre>call snap_termsInLanguages(p_conceptIds,p_langCodes);</pre>			
Parameter	Description	Data type	Examples
p_conceptIds	A string containing a comma separated list of concept identifiers.	text	'80146002,49438003'
p_langCodes	A string containing a comma separated list of language codes. <div style="border: 1px solid yellow; padding: 5px; margin: 5px 0;">  Only languages and dialects in the release files can be selected. With the International Release can be tested with p_langCode set to 'en-GB,en-US'. If other description files with terms in other languages are imported along with the relevant language refsets then these languages will also be accessible. </div>	text	'en-GB,en-US'
Example Procedure Call			
<pre>call snap_ShowLanguages('80146002,49438003','en-GB,en-US');</pre>			
Result			
conceptId	type_and_lang	term	

80146002	FSN en-GB	Excision of appendix (procedure)
80146002	Preferred en-GB	Appendectomy
80146002	Synonyms en-GB	Excision of appendix
80146002	FSN en-US	Excision of appendix (procedure)
80146002	Preferred en-US	Appendectomy
80146002	Synonyms en-US	Excision of appendix
49438003	FSN en-GB	Appendectomy with drainage (procedure)
49438003	Preferred en-GB	Appendectomy with drainage
49438003	Synonyms en-GB	Appendectomy and drainage
49438003	FSN en-US	Appendectomy with drainage (procedure)
49438003	Preferred en-US	Appendectomy with drainage
49438003	Synonyms en-US	Appendectomy and drainage

4.9.4. Search Procedures

Search Plus Procedure

The **searchPlus** procedure lists the conceptId and terms that match a search pattern. The search matches can also be filtered using a simple focus concept expression constraint or a regular expression pattern.



Note

This procedure is intended to demonstrate some of the search options available in MySQL and to illustrate the value of restricting searches to subtypes of concepts that in particular hierarchies or sub-hierarchies. Additional work would be required to make a more user-friendly interface to these search facility and this is beyond the scope of the SNOMED CT example database.

Example 4.9.4-1: Example Use of searchPlus Procedure

SQL Call to Procedure			
call snap_searchPlus(p_search, p_filter);			
Parameter	Description	Data type	Examples
p_search	<p>A search term string.</p> <p>These searches use MySQL's fulltext index employing one of the two supported search modes depending on the string supplied.</p> <p>If p_search includes plus "+" or minus "-" symbols, the search is performed using boolean mode. This means matches are only returned if all words preceded by "+" are present and none of the words "-". Word not preceded by either plus or minus, are not required to match but will contribute to the search. The results of this search are sorted by with the shortest matching terms first.</p> <p>If p_search does not include "+" or "-" symbols, the MySQL natural language search is used. It also orders the returned results by 'relevance' and is intended to assist contextual searching through literature. However, our testing indicate that the boolean mode is usually more effective for SNOMED CT searches.</p> <p>For more information about full text searches in MySQL please see Full-Text Search Functions in the MySQL Reference Manual.</p>	text	'+fundus +stomach' '+lung +disease +chronic' 'appendix' 'hemoglobin' 'infection'

p_filter	<p>A filter that will be applied to selectively include concepts that are subtype descendants of a specified concept:</p> <ul style="list-style-type: none"> • a simple focus concept subtype constraint. <ul style="list-style-type: none"> ▪ Starting with a less than sign < followed either by either <ul style="list-style-type: none"> ◦ a conceptId; or ◦ a shortcut abbreviation for a commonly used concept. To see the current set of shortcuts run the following query "SELECT * FROM config_shortcutPlus;" <p>A regular expression pattern to be used to filter terms returned by the search string</p> <ul style="list-style-type: none"> • a regular expression to exclude matching terms <ul style="list-style-type: none"> ▪ Starting this with an exclamation mark ! followed by the regular expression you want to exclude for the terms found by p_search. • a regular expression required for inclusion <ul style="list-style-type: none"> ▪ The text of the regular expression that must be matched for inclusion. 	text	'<find' '<proc' '<123037004' '!lung' 'heart'
----------	--	------	--

Example Procedure Call

```
call snap_searchPlus('+viral +infection','< 19829001 |disorder of lung|');
```

Result

conceptid	term
75570004	Viral pneumonia
276692000	Congenital viral pneumonia
75570004	Viral pneumonia (disorder)
421508002	Viral pneumonia associated with AIDS
276692000	Congenital viral pneumonia (disorder)
421508002	Viral pneumonia associated with acquired immunodeficiency syndrome
421508002	Viral pneumonia associated with acquired immunodeficiency syndrome (disorder)

4.9.5. Expression Constraint Procedures

ECL Query Procedure

The **eclQuery** lists the conceptId and preferred term for each concept that conforms to a specified **SNOMED CT expression constraint**.

Notes

1. The expression constraints supported do not cover the full ECL specification but are restricted as described below.
2. This procedure will only run in MySQL version 8.0 or later. It uses some function which are not available in earlier versions (including the widely used MySQL version 5.7).

Example 4.9.5-1: Example Use of eclQuery Procedure

SQL Call to Procedure			
<code>call snap_eclQuery(p_ecl);</code>			
Parameter	Description	Data type	Examples
p_ecl	The ECL query text	text	(< 19829001 disorder of lung) OR (< 301867009 edema of trunk) '(< 19829001 disorder of lung) MINUS (< 301867009 edema of trunk)' '>> 40541001 Acute pulmonary edema ' '>39057004 pulmonary valve ' '>!39057004 pulmonary valve '
Example Procedure Call			
<pre>call snap_eclQuery('< 19829001 disorder of lung :116676008 Associated morphology = 40829002 Acute edema ');</pre>			
Result			
conceptId	term		
10519008	Acute pulmonary oedema due to fumes AND/OR vapours		
40541001	Acute pulmonary oedema		
61233003	Silo-fillers' disease		
233706004	Drug-induced acute pulmonary oedema		
233709006	Toxic pulmonary oedema		
233710001	Chemical-induced pulmonary oedema		
233711002	Oxygen-induced pulmonary oedema		
360371003	Acute cardiac pulmonary oedema		
10674871000119105	Pulmonary oedema caused by chemical fumes		

Expression Constraint Feature Support and Limitations

The following notes provide a brief summary of the extent to which this procedure supports evaluation of expression constraints. For a full and detailed understanding of SNOMED CT expression constraints see the specification of the [SNOMED CT Expression Constraint Language](#).

ECL Operators and Examples

ECL Operator		Summary	Example
Symbol	Name		
<	Descendant of	The set of all subtypes of the given concept	< 404684003 Clinical finding
<<	Descendant or self of	The set of all subtypes of the given concept plus the concept itself	<< 73211009 Diabetes mellitus
>	Ancestor of	The set of all supertypes of the given concept	> 40541001 Acute pulmonary edema
>>	Ancestor or self of	The set of all supertypes of the given concept plus the concept itself	>> 40541001 Acute pulmonary edema
<!	Child of	The set of all children of the given concept	<! 195967001 Asthma
>!	Parent of	The set of all parents of the given concept	>! 195967001 Asthma
^	Member of	The set of referenced components in the given reference set	^ 733990004 Nursing activities reference set
*	Any	Any concept in the given SNOMED CT edition	*
:	Refinement	Only those concepts whose defining relationships match the given attribute value pairs	< 404684003 clinical finding : 116676008 associated morphology = *
AND	Conjunction	Only those concepts in both sets	(< 19829001 disorder of lung) AND (< 301867009 edema of trunk)
OR	Disjunction	Any concept that belongs to either set	(< 19829001 disorder of lung) OR (< 301867009 edema of trunk)
MINUS	Exclusion	Concepts in the first set that do not belong to the second set	(< 19829001 disorder of lung) MINUS (< 301867009 edema of trunk)

Additional Notes on Limitations of the ECL Query Procedure

The following notes outline the extent to which this procedure supports the SNOMED CT expression constraints and highlights some of the most significant limitations of this procedure.

One or more constraints can be specified.

- Each constraint must start with a focus concept constraint expressed as one of the following:
 - A single conceptId specifying that concept as the focus concept.
 - A single conceptId preceded by < (specifying subtypes only) or << (specifying self or subtypes)
 - A conceptId that identifies a reference set preceded by a ^ indicating members of that reference set.
 - An asterisk * (indicating any concept)
- In all cases a concept Id may be followed by a term surrounded by pipe characters.
 - The term between pipes will be ignored for processing.
- Spaces between any elements in the expression will be ignored
- The focus constraint may optionally be followed by a refinement constraint separated from the focus constraint by a colon :

If present the refinement constraint must consist of one or more attribute-value-constraint pairs.

- The pair consists of an attribute-constraint and a value-constraint:

- The attribute-constraint must be separated from the value constraint by an = (equals) sign
- Both the constraints may be specified using any of the forms permitted for the focus concept constraint (see 1)
- Attribute attribute-value-constraint pairs must be separated by a comma from any following attribute-value-constraint pair
- **NOTE:** The procedure **does not** support:
 - Nested refinement constraints
 - Role grouping constraints
 - Cardinality constraints

If more than one constraint is specified:

- Each constraint must be enclosed in brackets
- One of the following logical operators must be present between adjacent constraints
 - OR The resulting set is the set of concepts that conform to **either** the constraint to the left or the constraint to the right (or both constraint).
 - AND The resulting set is the set of concepts that conform to **both** the constraint to the left and the constraint on the right.
 - MINUS The resulting set is the set of concepts that both conform to the constraint to the left and do **not** conform to the constraint on the right.
- **NOTE:** The procedure **does not** support the use of brackets to alter the order of evaluation of constraints in a set.
 The set of constraints is evaluated from left to right and, as illustrated below, this is likely to affect the results.

i. **(A) OR (B) AND (C) MINUS (D)**

1. Concepts in either (A) or (B) form temporary set (aT1)
2. Concepts in (aT1) and also in (C) form temporary set (aT2)
3. Concepts in (aT2) that are NOT in (D) form the final result set (aR)
4. *One outcome of this order is that concepts in (B) that are **not** in (C) or are in (D) will not appear in the result set.*

ii. **(A) MINUS (D) AND (C) OR (B)**

1. Concepts in (A) and NOT in (D) form temporary set (bT1)
2. Concepts in (bT1) and also in (C) form temporary set (bT2)
3. Concepts in either (bT2) or (B) for the final result set (bR)
4. *In this order all concepts that are in (B) will be in the result set.*

Summary of Limitations of ECL Support for this Procedure

The Procedure **does not** support:

- Nested constraints
- Dotted attributes
- Nested refinement constraints
- Attribute group constraints
- Cardinality constraints

The Procedure also requires that:

- Even simple expression constraints must be enclosed by brackets when multiple constraints are combined.

5. Creating and Populating a SNOMED CT Database



Tip

This chapter contains excerpts from the SQL script used to build and populate the SNOMED CT example database. The full SQL script can be found in the `sct_mysql_loader_VP_latest.sql` file. This is in the `mysql_load` subfolder of the [SnomedRfsMySql.zip](#) package.

5.1. Creating the Database

The first step is to create the database schema. The code below is an excerpt from the SNOMED CT example database MySQL import script which carries out the following steps:

- Creates a database called "snomedct"
- Sets the default character set of the database as "utf8mb4"
- Sets snomedct as the default database for the following script.
- Sets timeouts appropriate to the import process.
- Clears `sql_mode` settings some of which, if present, may cause import issues.

Create SNOMED CT Database

```

DELIMITER ;
SELECT Now() `--`,`Create Database and Initialize` '---';
-- CREATE DATABASE
DROP DATABASE IF EXISTS `snomedct`;
CREATE DATABASE `snomedct` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
USE `snomedct`;
-- INITIALIZE SETTINGS
SET GLOBAL net_write_timeout = 60;
SET GLOBAL net_read_timeout=120;
SET GLOBAL sql_mode = '';
SET SESSION sql_mode = '';


```

5.2. Creating Tables for Components

Introduction

This section contains SQL statements that create database tables to accommodate the data in each of the main component files. Each table creation is accompanied by a summary of the relevant release file specification.

Notes

1. The  symbol in the top right of each file specification summary table is a link to the full file specification.
2. The table names used on this page are prefixed with **full_** as these are the tables into which the full SNOMED CT release will be imported. The loader script also create identically structured tables with the prefix **snap_** and the latest snapshot view is loaded into those tables.
3. The SQL code on this page creates the primary keys for each table (id, effective time) but omits creation of any other indexes. The loader script creates additional indexes after importing data into the table. This enables faster importing of data from the text files as the additional indexes do not need to be updated while importing.
4. The effectiveTime is set as a DATETIME data type. This supports a specific time in hours, minutes or seconds. In practice, effectiveTime values are formally restricted to YYYYMMDD but we are aware of at least one [SNOMED CT extension](#) that includes time units in the effectiveTime field or its release files. The effectiveTime

is set by default to a 2000-01-31, a date which predates any SNOMED CT effectiveTime value. In practice, the effectiveTime will always be set by the imported data so the default has no material effect.

5. Tables are also created for the full_textDefinition table and its snapshot version. As these tables have the same structure as the description tables, the data from the textDefinition release files could be imported into those tables instead. The text definitions would still be distinguishable from the descriptions as they have a different typeId.

Creating a Concept Table

Create Concept Table


```

DROP TABLE IF EXISTS `full_concept`;

CREATE TABLE `full_concept` (
  `id` BIGINT NOT NULL DEFAULT 0,
  `effectiveTime` DATETIME NOT NULL DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `definitionStatusId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`))
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.2-3: Concept File Specification Summary

Field	Data type	
id	SCTID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
definitionStatusId	SCTID	

Creating a Description Table

```


Create Description Table

DROP TABLE IF EXISTS `full_description`;

CREATE TABLE `full_description` (
  `id` BIGINT NOT NULL DEFAULT 0,
  `effectiveTime` DATETIME NOT NULL DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `conceptId` BIGINT NOT NULL DEFAULT 0,
  `languageCode` VARCHAR (3) NOT NULL DEFAULT '',
  `typeId` BIGINT NOT NULL DEFAULT 0,
  `term` TEXT NOT NULL,
  `caseSignificanceId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`)
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.2-3: Description File Specification Summary

Field	Data type	
id	SCTID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
conceptId	SCTID	
languageCode	String	
typeId	SCTID	
term	String	
caseSignificanceId	SCTID	

Creating a Relationship Table

```


Create Relationship Table

DROP TABLE IF EXISTS `full_relationship`;

CREATE TABLE `full_textDefinition` (
  `id` BIGINT NOT NULL DEFAULT 0,
  `effectiveTime` DATETIME NOT NULL DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `conceptId` BIGINT NOT NULL DEFAULT 0,
  `languageCode` VARCHAR (3) NOT NULL DEFAULT '',
  `typeId` BIGINT NOT NULL DEFAULT 0,
  `term` TEXT NOT NULL,
  `caseSignificanceId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`)
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.2-3: Relationship File Specification Summary


Field	Data type	
id	SCTID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
sourceId	SCTID	
destinationId	SCTID	
relationshipGroup	Integer	
typeId	SCTID	
characteristicTypeId	SCTID	
modifierId	SCTID	

5.3. Creating Tables for Reference Sets

Introduction

This section contains SQL statements that create database tables to accommodate the data in some of the reference set release files. Each table creation is accompanied by a summary of the relevant release file specification. The selection of reference set types shown on this page is incomplete but includes at least one example reference set that includes an additional column of each of the permitted types (componentId, string and integer).

Notes

1. The  symbol in the top right of each file specification summary table is a link to the full file specification.
2. The table names used on this page are prefixed with **full_** as these are the tables into which the full SNOMED CT release will be imported. The loader script also create identically structured tables with the prefix **snap_** and the latest snapshot view is loaded into those tables.
3. The SQL code on this page creates the primary keys for each table (id, effective time) but omits creation of any other indexes. The loader script creates additional indexes after importing data into the table. This enables faster importing of data from the text files as the additional indexes do not need to be updated while importing.
4. The SQL code used on this page does not include any additional optimizations for generating alternative snapshot views. Optimizations discussed in this guide can be added to the tables if required. However, this loader script creates and populates tables for both the full release and the current snapshot views. Therefore, additional optimizations would only deliver performance benefits when querying retrospective snapshot views. Even in this case the performance benefits for most types of query are often limited when compared to the use of unoptimized dynamic views.
5. The effectiveTime is set as a DATETIME data type. This supports a specific time in hours, minutes or seconds. In practice, effectiveTime values are formally restricted to YYYYMMDD but we are aware of at least one [SNOMED CT extension](#) that includes time units in the effectiveTime field or its release files. The effectiveTime is set by default to a 2000-01-31, a date which predates any SNOMED CT effectiveTime value. In previous versions of the script defaults were set to 0000-00-00 but some SQL settings treat these as invalid dates. In practice, the effectiveTime will always be set by the imported data so the default has no material effect.

Creating a Simple Refset Table

Create Concept Table


```

DROP TABLE IF EXISTS `full_refset_Simple`;

CREATE TABLE `full_refset_Simple` (
  `id` char(36) NOT NULL DEFAULT '',
  `effectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `refsetId` BIGINT NOT NULL DEFAULT 0,
  `referencedComponentId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`)
  ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.3-4: Simple Refset File Specification Summary

Field	Data type	
id	UUID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	

refsetId	SCTID
referencedComponentId	SCTID

Creating a Language Refset Table

Create Description Table


```

DROP TABLE IF EXISTS `full_refset_Language`;

CREATE TABLE `full_refset_Language` (
  `id` char(36) NOT NULL DEFAULT '',
  `effectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `refsetId` BIGINT NOT NULL DEFAULT 0,
  `referencedComponentId` BIGINT NOT NULL DEFAULT 0,
  `acceptabilityId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`))
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.3-4: Language Refset File Specification Summary

Field	Data type	
id	UUID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
refsetId	SCTID	
referencedComponentId	SCTID	
acceptabilityId	SCTID	

Creating an Extended Map Refset Table

Create Relationship Table


```

DROP TABLE IF EXISTS `full_refset_ExtendedMap`;

CREATE TABLE `full_refset_ExtendedMap` (
  `id` char(36) NOT NULL DEFAULT '',
  `effectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `refsetId` BIGINT NOT NULL DEFAULT 0,
  `referencedComponentId` BIGINT NOT NULL DEFAULT 0,
  `mapGroup` INT NOT NULL DEFAULT 0,
  `mapPriority` INT NOT NULL DEFAULT 0,
  `mapRule` TEXT NOT NULL,
  `mapAdvice` TEXT NOT NULL,
  `mapTarget` VARCHAR (200) NOT NULL DEFAULT '',
  `correlationId` BIGINT NOT NULL DEFAULT 0,
  `mapCategoryId` BIGINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`,`effectiveTime`))
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.3-4: Extended Map Refset File Specification Summary

Field	Data type	
id	UUID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
refsetId	SCTID	
referencedComponentId	SCTID	
mapGroup	Integer	
mapPriority	Integer	
mapRule	String	
mapAdvice	String	
mapTarget	String	
correlationId	SCTID	
mapCategoryId	SCTID	

Creating a Module Dependency Refset Table

```


Create Relationship Table

DROP TABLE IF EXISTS `full_refset_ModuleDependency`;

CREATE TABLE `full_refset_ModuleDependency` (
  `id` char(36) NOT NULL DEFAULT '',
  `effectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  `active` TINYINT NOT NULL DEFAULT 0,
  `moduleId` BIGINT NOT NULL DEFAULT 0,
  `refsetId` BIGINT NOT NULL DEFAULT 0,
  `referencedComponentId` BIGINT NOT NULL DEFAULT 0,
  `sourceEffectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  `targetEffectiveTime` DATETIME NOT NULL
    DEFAULT '2000-01-31 00:00:00',
  PRIMARY KEY (`id`,`effectiveTime`))
ENGINE=MyISAM DEFAULT CHARSET=utf8mb4;

```

Table 5.3-4: Module Dependency Refset File Specification Summary

Field	Data type	
id	UUID	
effectiveTime	Time	
active	Boolean	
moduleId	SCTID	
refsetId	SCTID	
referencedComponentId	SCTID	
sourceEffectiveTime	Time	
targetEffectiveTime	Time	

5.4. Importing Release Files

This section contains examples of the SQL statements used to imports data from a component release files in the appropriate database table. Before importing the tables must be created (see 5.2. [Creating Tables for Components](#) and 5.3. [Creating Tables for Reference Sets](#)).

Sample SQL Code for Importing from Full Release

 **Note**

The code shown below provides illustrative examples only. For full details download an review the import script. For details see [A.1 Download the SNOMED CT Example Database Package](#).

Import Concepts from Full Release

Import Concept File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Terminology/sct2_Concept_Full_INT_[RELEASE-DATE].txt'
INTO TABLE `full_concept`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`definitionStatusId`);
```

Import Descriptions from Full Release

Import Description File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Terminology/sct2_Description_Full-en_INT_[RELEASE-DATE].txt'
INTO TABLE `full_description`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`definitionStatusId`);
```

Import Relationships from Full Release

Import Relationship File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Terminology/sct2_Relationship_Full_INT_[RELEASE-DATE].txt'
INTO TABLE `full_relationship`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`sourceId`,`destinationId`,`relationshipGro
up`,`typeId`,`characteristicTypeId`,`modifierId`);
```

Import Simple Refsets from Full Release

Import a Simple Refset File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Refset/Content/der2_Refset_SimpleFull_INT_[RELEASE-DATE].txt'
INTO TABLE `full_refset_simple`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`refSetId`,`referencedComponentId` );
```


Import Language Refsets from Full Release

Import a Language Refset File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Refset/Language/der2_cRefset_LanguageFull-en_INT_$(RELDATE).txt'
INTO TABLE `full_refset_Language`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`refsetId`,`referencedComponentId`,`accepta
bilityId`);
```

Import Extended Map Refsets from Full Release

Import an Extended Map Refset File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Full/
Refset/Map/der2_iissccRefset_ExtendedMapFull_INT_$(RELDATE).txt'
INTO TABLE `full_refset_ExtendedMap`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`refsetId`,`referencedComponentId`,`mapGrou
p`,`mapPriority`,`mapRule`,`mapAdvice`,`mapTarget`,`correlationId`,`mapCategoryId`);
```

Sample SQL Code for Importing from a Snapshot Release

Note

The code shown below provides illustrative examples only to show the minor difference between the code for importing the snapshot compared to the full release. For full details download and review the import script. For details see [A.1 Download the SNOMED CT Example Database Package](#).

Import Concepts from Snapshot Release

Import Concept File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Snapshot/
Terminology/sct2_Concept_Snapshot_INT_[RELEASE-DATE].txt'
INTO TABLE `snap_concept`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`definitionStatusId`);
```

Import Simple Refsets from Snapshot Release

Import a Simple Refset File

```
LOAD DATA LOCAL INFILE '[RELEASE-FILE-PATH]/[RELEASE-PACKAGE-VERSION-NAME]/Snapshot/
Refset/Content/der2_Refset_SimpleSnapshot_INT_[RELEASE-DATE].txt'
INTO TABLE `snap_refset_simple`
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(`id`,`effectiveTime`,`active`,`moduleId`,`refsetId`,`referencedComponentId`);
```

Appendix A: Building the SNOMED CT Example Database

A.1 Download the SNOMED CT Example Database Package

Info


The *SnomedRfsMySQL* package includes tools that are used to:


- Apply the MySQL settings required by the SNOMED CT load process
- Run the SNOMED CT load process.

Download and Unzip the *SnomedRfsMySQL* Package

- The package can be downloaded as a zip archive file [SnomedRfsMySQL](#).
- Unzip this file to create a folder with called SnomedRfsMySQL.

Move the *SnomedRfsMySQL* Folder to Your Home Folder

- Use Finder (or File Explorer) to move the SnomedRfsMySQL folder so it becomes a subfolder of you home folder. 
 - On MacOS or other Unix based systems: **/Users/your-username/SnomedRfsMySQL**
 - On Windows systems: **C:\SnomedRfsMySQL**

 If you prefer, you can move the SnomedRfsMySQL folder to a different location. If you do this then references to "\$HOME/SnomedRfsMySQL" (or "C:\SnomedRfsMySQL") in the following instructions should be replaced by references to the path to the **SnomedRfsMySQL** folder on your system.

File


Modified


 [SnomedRfsMySQL.zip](#)

5 minutes ago by [David Markwell](#)

A.2 Download the Release File Package


SNOMED CT Release Packages can be obtained from the [Member Licensing & Distribution Service \(MLDS\)](#).

- Use Finder (or File Explorer) to create a folder in your home folder called **SnomedCT_ReleaseFiles** 
 - On MacOS or other Unix based systems: **/Users/your-username/SnomedCT_ReleaseFiles**
 - On Windows systems: **C:\SnomedCT_ReleaseFiles**
- Download the latest SNOMED CT International Release Package (a zip archive) into your **SnomedCT_ReleaseFiles** folder.

 If you prefer, you can download the SNOMED CT release package to a different location. If you do this then you will need to specify the full path to the release package when you run the SNOMED CT load process.


A.3 Instructions for Mac OS Users


The instructions in this section are specific to user of Mac OS systems.

-  Users of other Unix based system such as Linux or Ubuntu may find some of the instructions in this section applicable in their environment. However, the location of MySQL configuration files may differ and as a result some aspects of the configuration process may need to be altered.

Users of Window systems should skip to [A.4 Instructions for Windows Users](#).

A.3.1 MySQL Installation (MacOS)

-  **Info**
 The instruction in this section assume you are installing MySQL for the first time or have fully uninstalled an earlier installation.
 If MySQL is already installed you may choose to skip this section.

-  **Note**
 if MySQL was installed using a different installation package, some of the configuration steps described in later sections may need to be modified.
 Please refer to <https://www.mysql.com/products/community/> for detailed information about MySQL.

Install MySQL Community Server

- Download and install the **DMG Archive** version of the [MySQL Community Server](#).
- During the installation process you will be prompted to provide a password for the MySQL server root account.
 - **Make a note of the root password** - you will need to use it to load the SNOMED CT release package.

Install MySQL Workbench

- Download and install the **DMG Archive** version of the [MySQL Workbench](#).

A.3.2 Set Required MySQL Configuration (MacOS)

Open the **Terminal** Application



Type the command lines shown below into the terminal window.

Change Directories to the *SnomedRfsMySql* Folder



```
cd "$HOME/SnomedRfsMySql"
```

Make the Scripts in the *bash* Subfolder Executable

```

> chmod u+x bash/*

```



Tip

If this command does reports an error, please try the following modified command, which may prompt for your password to confirm the action:

```
sudo chmod u+x bash/*
```

Run *snomed_config_mysql* Script to Configure MySQL



This script requires you to have administrator rights to access your computer and may prompt you to enter your login password. If you do not have administrator rights to access your computer, you will need to ask someone who does have those rights to run this part of the process.

```

> sudo bash/snomed_config_mysql

```

Close the Terminal Application Window

- You have now completed the configuration process.

Start or Restart the MySQL Server



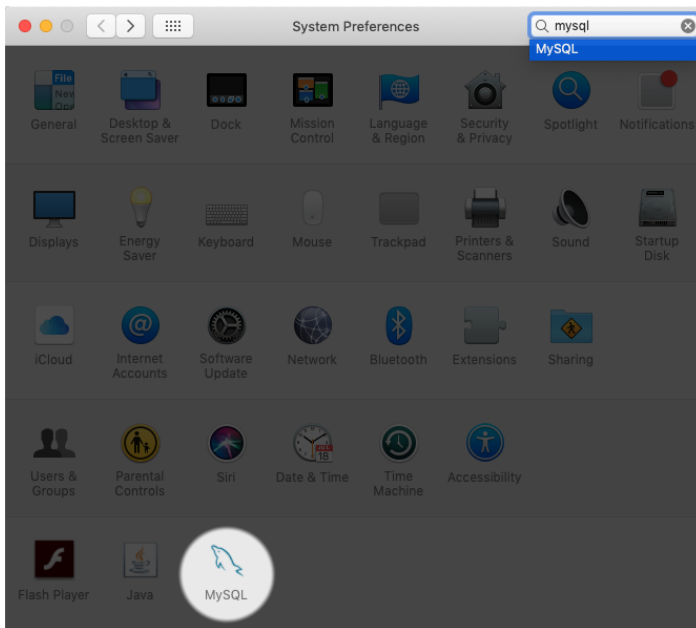
Info

The MySQL Server must be started (or stopped and restarted) to apply the required configuration settings.

Open the Mac OS **Settings** Application



In System Preferences Search for MySQL



Click to Open the MySQL Dialog



Stop MySQL Server (if it is running)

If MySQL is running the dots are green (as shown above).

- Click **Stop MySQL Server** and wait for the dots to turn red (as shown below).
 - You will be prompted for your Mac password to confirm this action.



Start MySQL Server

When the dots are red (as above) MySQL is not running

- Click **Start MySQL Server** and wait for the dots to turn green.
 - You will be prompted for your Mac password to confirm this action.

A.3.3 Load Release Package into MySQL (MacOS)



Disk Space

Before running the SNOMED CT load script, ensure you have at least 10Gb of free disk space. Once the load is completed 4Gb can be released by deleting the Release package folder and zip file.

The following figures apply to the SNOMED CT International Release package 2019-07-31.

- Release Package zip file: 0.5 Gb
- Release Package folder: 3.5 Gb
- Installed database up to: 5.5 Gb

Open the Terminal Application



Change Directories to the SnomedRfsMySQL Folder

You must change directories to the SnomedRfsMySQL folder before starting the loader script.




Start the SNOMED CT Loaded Script for MySQL

As shown above you must be in the **SnomedRfsMySQL** folder to run the loader script. Additionally as shown below you must include the name of the subfolder (**bash**) when running the loader script. The script may not run correctly if called from a different current folder or without the including the subfolder name.

```
bash/snomed_load_mysql
```

Respond to Prompts from the Script

You will be prompted to enter the following data when the script is run.

Prompt	Response options	Notes
Loader script identifying tag	Leave blank to accept the default	Recommended option Uses the default create_latest script in the SnomedRfsMySQL package.
	Enter the name of one of the available scripts (these are listed above the prompt)	Uses the specified script to control the process of loading the SNOMED CT data into the database. Scripts keys have two parts a prefix and suffix separated by an underscore. The prefix indicates the kind of action: <ul style="list-style-type: none"> • create : Create a new database and load the data from the specified version. • update : Update the views and procedures in the database without recreating the database or reloading the the tables. • extend : Extend the database by loading data from another package to the existing database. The suffix indicates the edition and version to which this applies: <ul style="list-style-type: none"> • latest : The most recent International Edition package • yyymmdd: The International Edition for the stated year month and day • packageyyymmdd: The identified Edition or Package for the stated year month and day.
Release package path	You must enter the full path of the release package folder or release package zip archive. <div style="border: 1px solid gray; padding: 5px; width: fit-content;">  You will not be prompted for this if you select an update script option. </div>	The path specified must point to a release package zip archive or an unzipped release package folder. <ul style="list-style-type: none"> • There is no need to include the .zip extension when referring to a release package archive or folder. <ul style="list-style-type: none"> ▪ The script first looks for an unzipped release folder with the relevant name. ▪ If the folder is not found the script looks for a zip archive with the same name plus the .zip extension. ▪ If the zip file is found it is unzipped and the resulting folder is used.
Database name	Leave blank to accept the default	Default option <ul style="list-style-type: none"> • Uses the database name snomedct. • If the snomedct database already exists, it will be dropped (deleted) and recreated.
	Specify a name (must begin with the letter s followed by lowercase letters and/or digits)	Uses the database name provided. <ul style="list-style-type: none"> • Using different names allows several SNOMED CT databases to co-exist (e.g. for different Editions) • Each SNOMED CT database will use about 5Gb of disk space ... so using different names may fill your available disk space! • If the named database already exists, it will be dropped (deleted) and recreated.

MySQL username	Leave blank to accept the default	The default is root .
	Enter your MySQL username	The username chosen must be an account with administrator access rights enabling database creation.

Wait for the Database Password Prompt

Before the creating the database the script generates an additional release file containing the transitive closure (this is used to optimize testing and listing of subtypes). This may take between 2 and 5 minutes to complete.

- If you rerun the loader script again on the same release, the script will reuse the existing transitive closure. If you accept the options, there will be no delay while the rebuild occurs.

Respond to the Database Password Prompt

When the script starts to access MySQL you will then be prompted for your database password.

- Note that the required password here is the password associated with your MySQL account.
- As noted earlier the account used for the SNOMED CT load process must have appropriate access permissions and its username should match your Mac login name.

Wait for the Load Process to Complete

- Depending on system performance the process may take between 20 and 45 minutes to complete. It may take longer with National Editions that contain additional content.
 - As the MySQL script runs it will report progress on the screen. Some steps take much longer than others. For example, loading data into the database tables and adding or building indexes take much longer than any of the other steps. So if the message about these steps are showing for a long time don't worry. Let the process continue.
- When the script completes, scroll back up the command window to check for any ERROR reports for MySQL ... there should not be any!
- Now it is time to open MySQL Workbench to view your SNOMED CT database.

A.4 Instructions for Windows Users

The instructions in this section are specific to users of Windows systems.

Users of Mac OS systems should refer [A.3 Instructions for Mac OS Users](#).

- ✓ Users of other Unix based system such as Linux or Ubuntu may also find [A.3 Instructions for Mac OS Users](#) useful. However, the location of MySQL configuration files may differ and as a result some aspects of the configuration process may need to be altered.

A.4.1 MySQL Installation (Windows)



Info

The instruction in this section assume you are installing MySQL for the first time or have fully uninstalled an earlier installation.
If MySQL is already installed you may choose to skip this section.



Note

if MySQL was installed using a different installation package, some of the configuration steps described in later sections may need to be modified.
Please refer to <https://www.mysql.com/products/community/> for detailed information about MySQL.

Install the MySQL Windows MSI installer

This is available as a free download from: <https://dev.mysql.com/downloads/installer/>

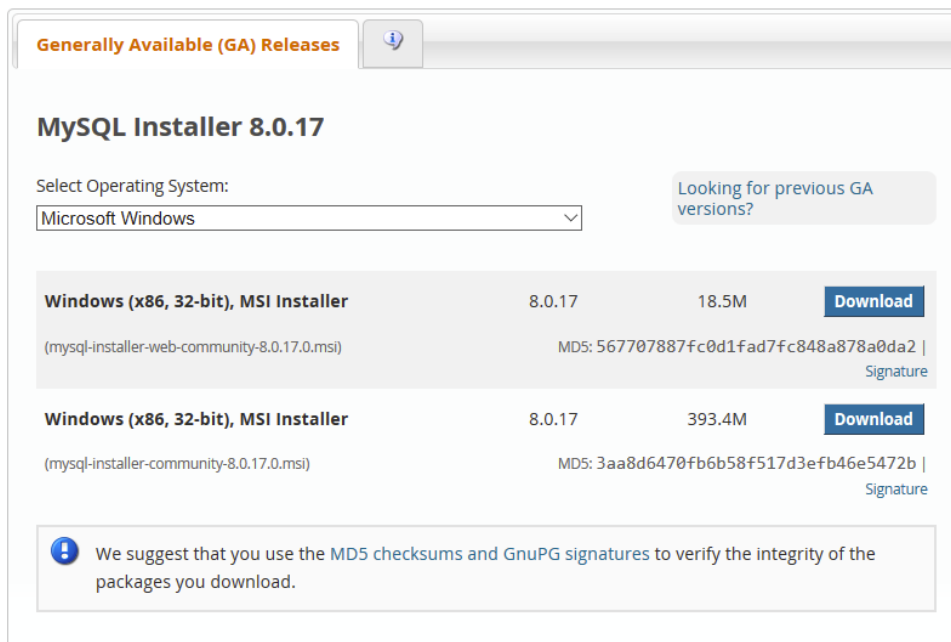


Note

Although the installers are 32-bit software, they will install the 64-bit version of MySQL if you are using a 64-bit system.

MySQL Community Downloads

← MySQL Installer



Generally Available (GA) Releases

MySQL Installer 8.0.17

Select Operating System:
Microsoft Windows

[Looking for previous GA versions?](#)

Windows (x86, 32-bit), MSI Installer	8.0.17	18.5M	Download
(mysql-installer-web-community-8.0.17.0.msi)	MD5: 567707887fc0d1fad7fc848a878a0da2		Signature
Windows (x86, 32-bit), MSI Installer	8.0.17	393.4M	Download
(mysql-installer-community-8.0.17.0.msi)	MD5: 3aa8d6470fb6b58f517d3efb46e5472b		Signature

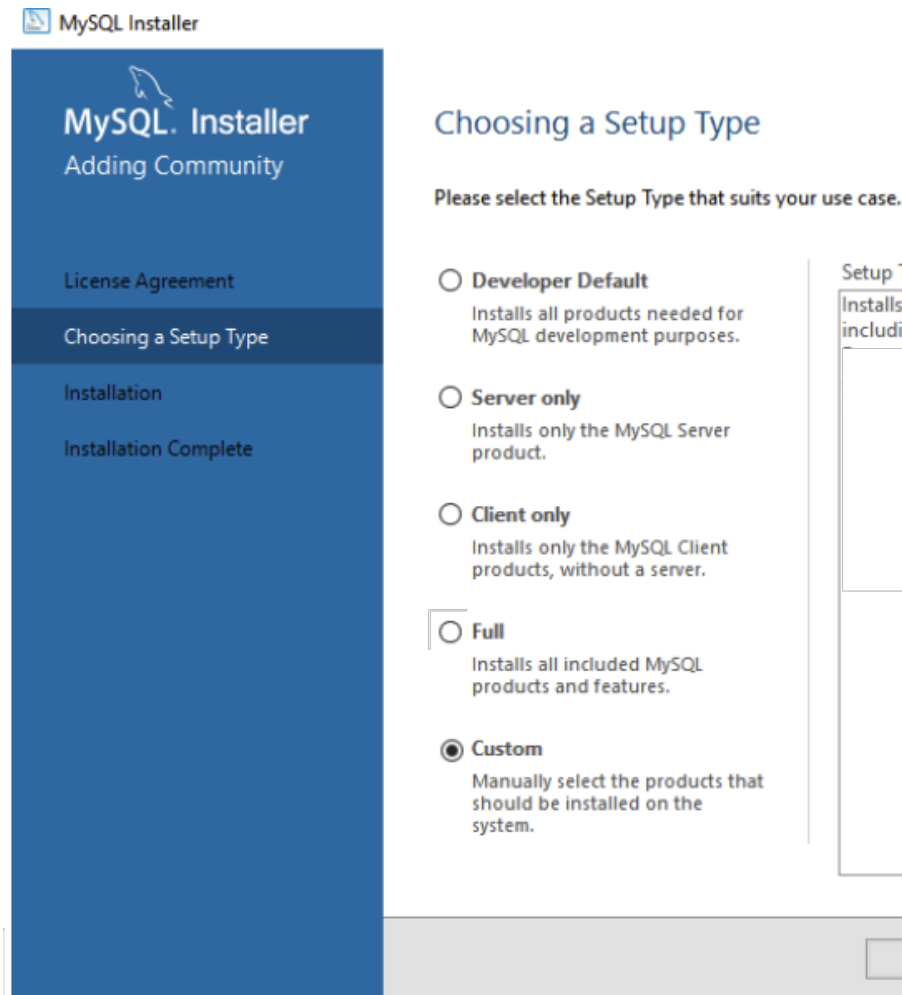
! We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

You can use either of the installers listed here:

- The first installer option downloads a small package and then gets the other packages during the installation process.
- The second installer downloads all the packages and can then continue the installation without an internet connection.

Download and Run your Chosen MySQL Installer

During the installation process you will be prompted to choose the setup type. The minimum recommended installation for the SNOMED CT database only required MySQL Server and MySQL Workbench so (unless you have additional requirements) select the Custom option as shown below.



Continue the process by clicking the next button.

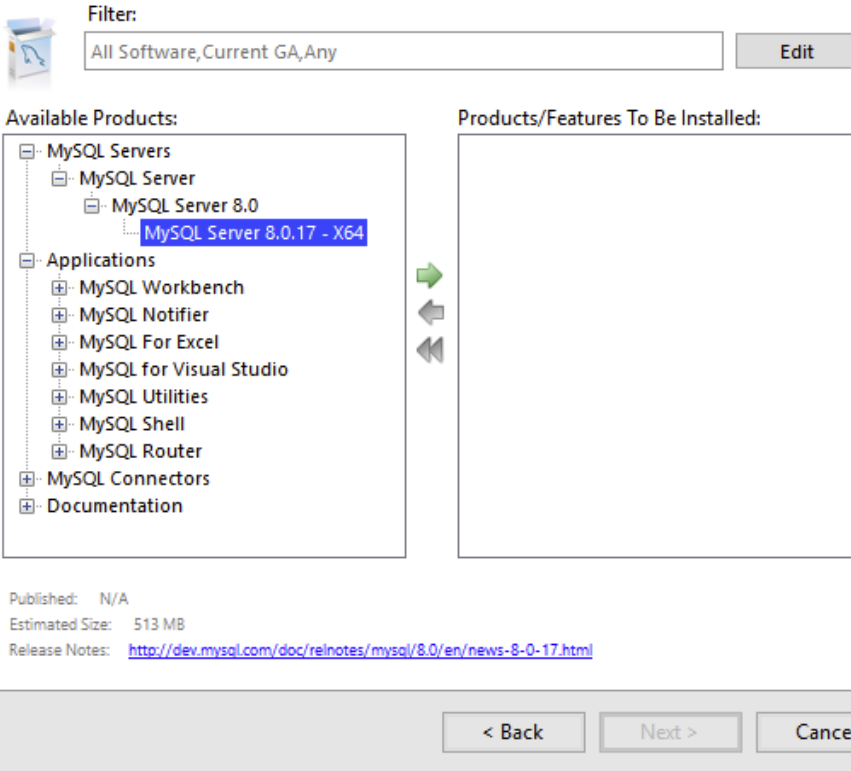
- You will then be prompted to specify the products and features to be installed.

Select MySQL Server for Installation

- Expand the nested list under MySQL Servers
- Select the MySQL Server 8.0.17 (or a higher version if one is shown)
- Click the arrow pointing to the right
- This will add the MySQL Server to the list of Products/Features To Be Installed.

Select Products and Features

Please select the products and features you would like to install on this machine.



Filter: All Software, Current GA, Any Edit

Available Products:

- MySQL Servers
 - MySQL Server
 - MySQL Server 8.0
 - MySQL Server 8.0.17 - X64
- Applications
 - MySQL Workbench
 - MySQL Notifier
 - MySQL For Excel
 - MySQL for Visual Studio
 - MySQL Utilities
 - MySQL Shell
 - MySQL Router
 - MySQL Connectors
 - Documentation

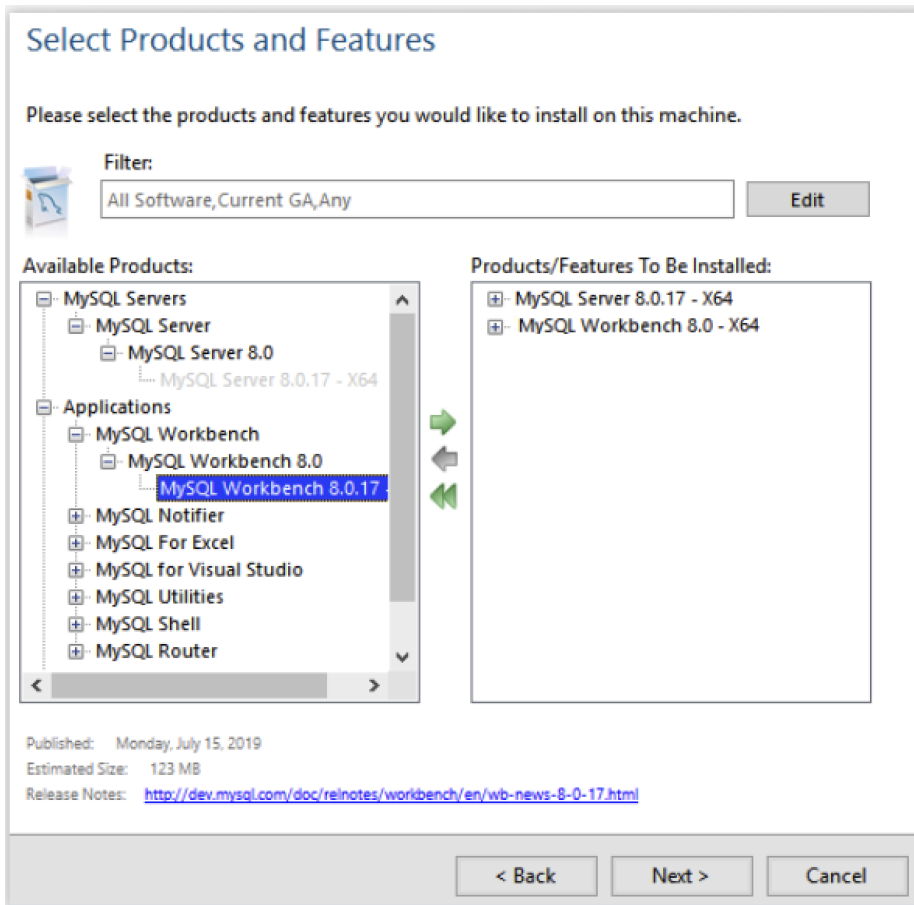
Products/Features To Be Installed:

Published: N/A
Estimated Size: 513 MB
Release Notes: <http://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-17.html>

< Back Next > Cancel

Select MySQL Workbench for Installation

- Expand the nested list under Applications / MySQL Workbench
- Select the MySQL Workbench 8.0.17 (or a higher version if one is shown)
- Click the green arrow pointing to the right
- This will add the MySQL Workbench to the list of "Products/Features To Be Installed".



- When both Server and Workbench are in the right-hand list, click the Next button.

Checking Requirements

At this point you may see a message indicating a requirement for a Visual C++ Redistributable package as shown below.

Check Requirements

The following products have failing requirements. MySQL Installer will attempt to resolve them automatically. Requirements marked as manual cannot be resolved automatically. Click on each item to try and resolve it manually.

For Product	Requirement	Status
<input type="radio"/> MySQL Workbench 8.0	Visual C++ 2015 Redistributable	

If you see this message you should download and install the required package from <https://www.microsoft.com/en-us/download/details.aspx?id=48145> before proceeding.

- Further notes on this process are provided on [Meeting Requirements for MySQL Installation \(Windows\)](#).

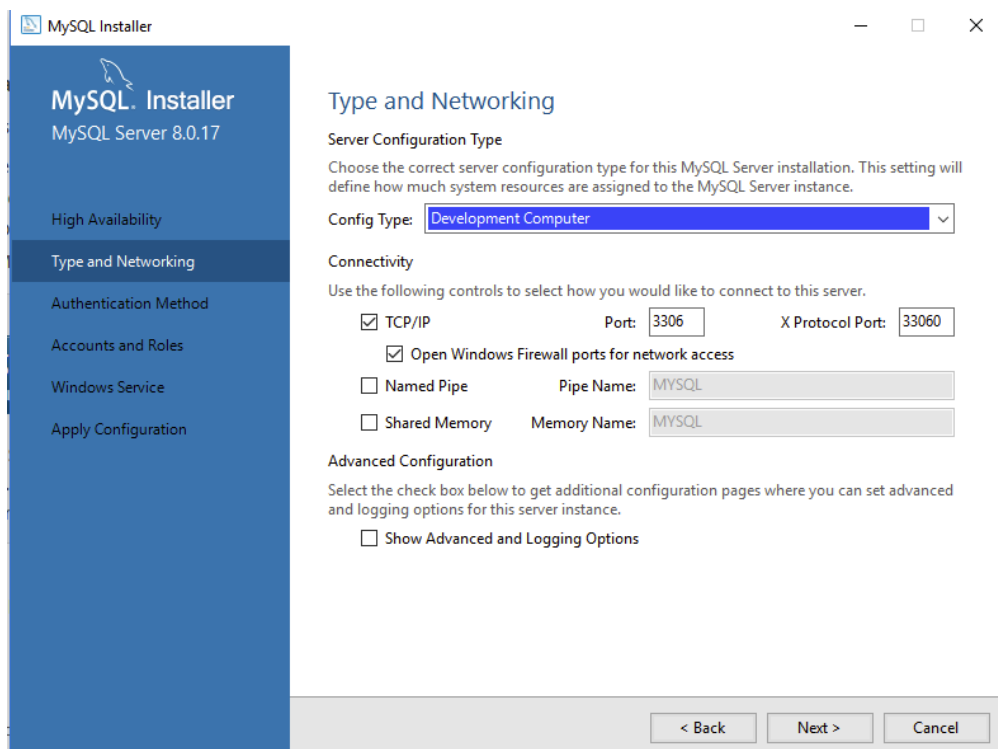
Continue with the Installation Process

- Select the **Standalone MySQL Server option**.

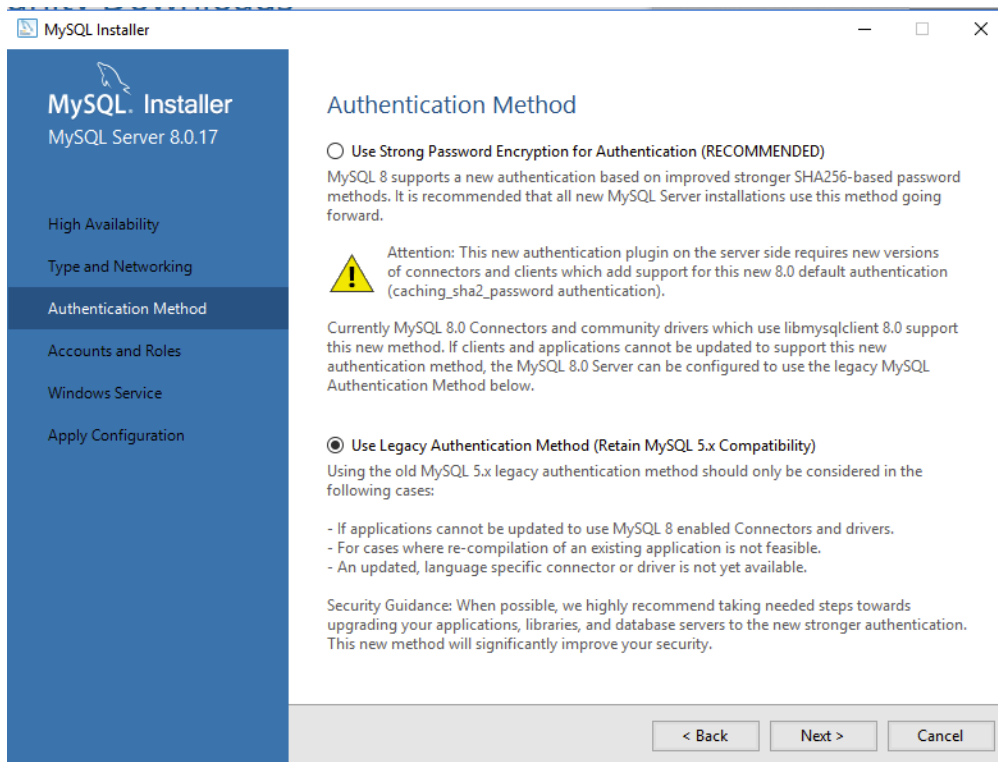
High Availability

- Standalone MySQL Server / Classic MySQL Replication**
 Choose this option to run the MySQL instance as a standalone database server with the opportunity to configure classic replication later. With this option, you can provide your own high-availability solution, if required.
- InnoDB Cluster**
 The InnoDB cluster technology provides an out-of-the-box high availability (HA) solution for MySQL using Group Replication.

- Select the **Development Computer** option



- Select the **Legacy Authentication Method** unless you will be using the database for other purposes that require greater security.

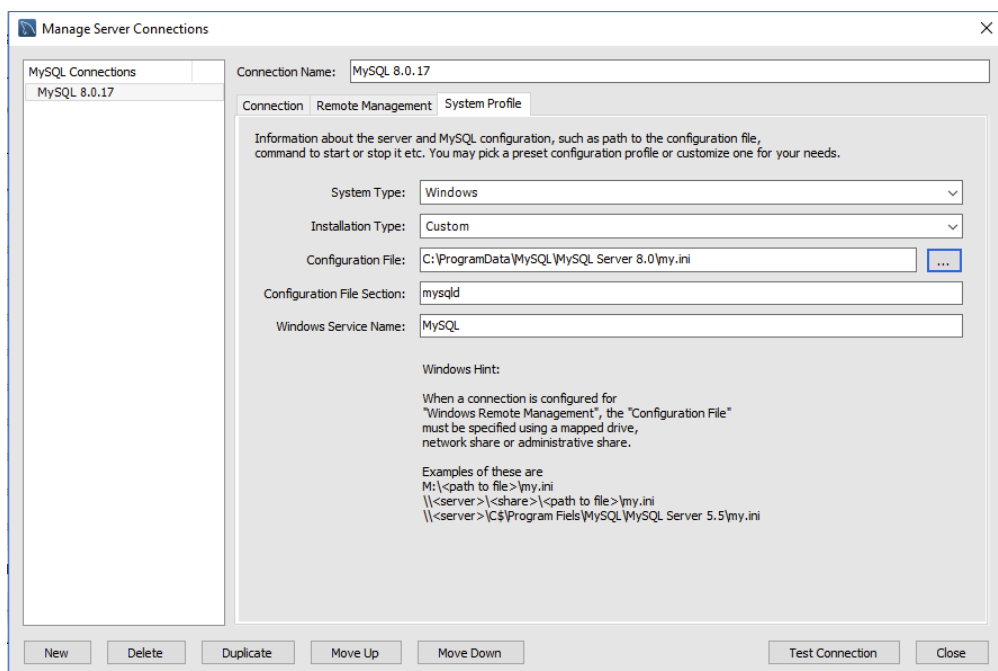


- Accept or apply the following Server Connection settings

Note

The Configuration File setting must refer to: **C:\ProgramData\MySQL\MySQL Server 8.0\my.ini**

- This is a file that will be modified in the to configure the server so that it will correctly load the SNOMED CT release files.



- When prompted, set a **root** password for access to the database.

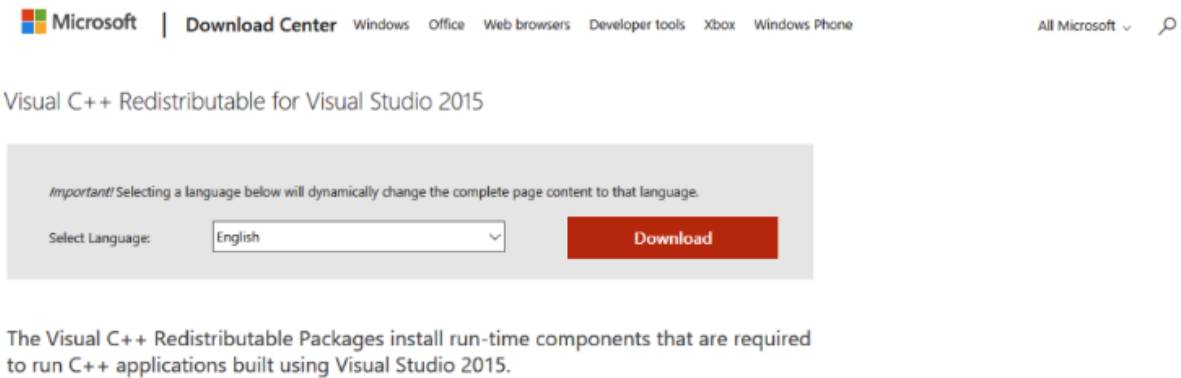
Warning
Be sure to remember your root password this as you will need it for access to the database.

Meeting Requirements for MySQL Installation (Windows)

Installing or Updating Microsoft Visual C++

During the installation of MySQL 8.0.x on Windows you may see a message indicating that you need to install or update to the Microsoft Visual C++ 2015 Redistributable package.

- If required this can be obtained as a free download from <https://www.microsoft.com/en-us/download/details.aspx?id=48145>.



- Click the Download button.

Choose the download you want

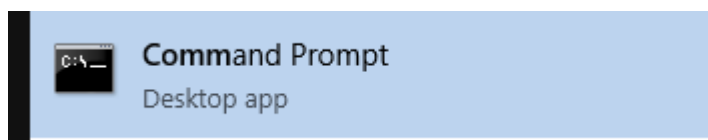
<input type="checkbox"/> File Name	Size
<input checked="" type="checkbox"/> vc_redist.x64.exe	13.9 MB
<input type="checkbox"/> vc_redist.x86.exe	13.1 MB

- Choose the x64 version if you have a 64-bit computer or the x86 if you have an older 32-bit computer.
- Continue an install the package.
- The return to the MySQL installation process.

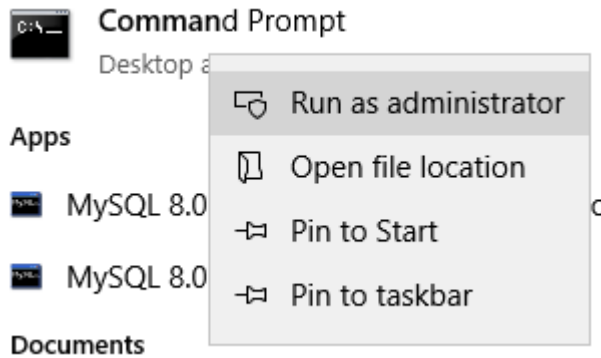
A.4.2 Set Required MySQL Configuration (Windows)

Open the Command Prompt in Administrator Mode

- In the main Window menu locate the **Command Prompt** Desktop app.



- Right-click on this item to show the drop down menu



- Select the **Run as administrator** option.
 - This is necessary as some steps below require administrator status.

Run the Following Commands from the Command Prompt

Type the following command to change directories to the SnomedRfsMySQL.

- Adjust the path as necessary if you have not installed the SnomedRfsMySQL folder in the root folder on drive C.

```
cd C:\SnomedRfsMySQL
```

- Type the following command to stop the MySQL server (if it is running).

```
sc stop MySQL80
```

- Type the following command to run the configuration process. This command runs a script that updates the MySQL server configuration.

```
win\snomed_wconfig_mysql
```

- Type the following command to restart the MySQL server

```
sc start MySQL80
```

A.4.3 Install a Perl Processor

i Before importing the SNOMED CT data the import process creates a "transitive closure file". This allows this file to be imported to create a transitive closure table that supports rapid subtype testing. The process that creates the transitive closure file from other release files uses the Perl language. Windows does not include a Perl language interpreter. Strawberry Perl is a widely used free Perl environment for Windows and this should be installed on your computer before running the import script.

Download Strawberry Perl

- Go to <http://strawberryperl.com>
- Download either the appropriate version for your system (i.e. either 32-bit or 64-bit)

Install Strawberry Perl

- Open and run the downloaded file
 - This will have a name like "strawberry-perl-5.30.0.1-64bit.msi" (version numbers may differ).
- Go through the installation process

- Read and accept the license.
- We recommend that you install in the default installation path (e.g. "C:\Strawberry")
 - You can choose a different path if preferred as the installation script will attempt to locate the file wherever it is installed on the main drive.

⚠ Warnings

Do **not** install Strawberry Perl within the SnomedRfsMySQL folder as the installation will be removed by future updates to the SnomedRfsMySQL package. Ideally you should install Strawberry Perl on drive C.

If you need to install Perl on another disk or network drive (click here) ...

If Perl is installed on a network or a secondary drive (i.e. not drive C), you need to specify that location. To do this create (or edit) a plain text file called **perlPath.cfg** file in the **SnomedRfsMySQL\win folder**. This file must contain a single line of text specifying the full path of the perl.exe file (for example the file might contain the text "D:\AddedSoftware\Strawberry\perl\bin\perl.exe")

If the import script cannot find perl.exe at the location specified in the perlPath.cfg file, that configuration file will be deleted and will need to be recreated with the correct full path to perl.exe.

Check the Installation

- When the installation is complete, you can test the installation by running the following command in a **newly opened** Command Prompt.

```
perl C:\SnomedRfsMySQL\lib\test.pl
```

- If Strawberry Perl is correctly installed the result should be the following message.

```
Perl is correctly installed
```

✓ Tip

If the message above does not appear, check the following:

1. The command line above assumes that SnomedRfsMySQL is installed in the C:\ root folder. If this is not the case, repeat the test with the correct path to SnomedRfsMySQL.
2. Ensure that you have closed all open command line windows. Then open a new command line window and run the test command again.
3. If neither of the above steps corrects the issue, it suggests a problem with the installation. Consider uninstalling and reinstalling Strawberry Perl to correct the issue.

A.4.4 Load Release Package into MySQL (Windows)

⚠ Disk Space

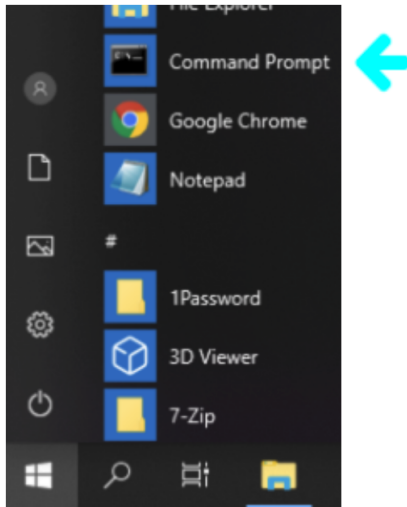
Before running the SNOMED CT load script, ensure you have at least 10Gb of free disk space. Once the load is completed 4Gb can be released by deleting the Release package folder and zip file.

The following figures apply to the SNOMED CT International Release package 2019-07-31.

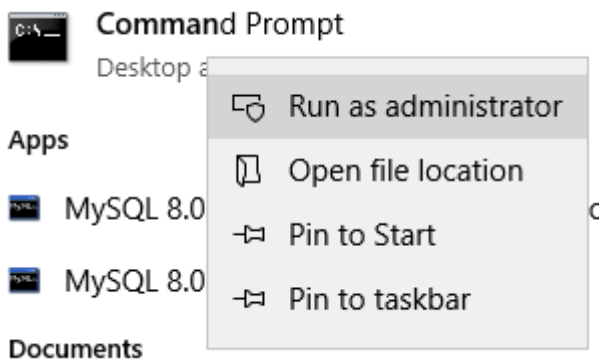
- Release Package zip file: 0.5 Gb
- Release Package folder: 3.5 Gb
- Installed database up to: 5.5 Gb

Open the Command Prompt in Administrator Mode

- In the main Window menu locate the **Command Prompt** Desktop app.



- Right-click on this item to show the drop down menu



- Select the **Run as administrator** option.
 - This is necessary as some steps below require administrator status.

Change Directories to the SnomedRfsMySql Folder

You must change directories to the SnomedRfsMySql folder before starting the loader script.




Start the SNOMED CT Loader Script for MySQL

As shown above you must be in the **SnomedRfsMySql** folder to run the loader script. Additionally as shown below you must include the name of the subfolder (**win**) when running the loader script. The script may not run correctly if called from a different current folder or without the including the subfolder name.



Respond to Prompts from the Script

You will be prompted to enter the following data when the script is run.

Prompt	Response options	Notes
Loader script identifying tag	Leave blank to accept the default	Recommended option Uses the default create_latest script in the SnomedRfsMySQL package.
	Enter the name of one of the available scripts (these are listed above the prompt)	Uses the specified script to control the process of loading the SNOMED CT data into the database. Scripts keys have two parts a prefix and suffix separated by an underscore. The prefix indicates the kind of action: <ul style="list-style-type: none"> • create : Create a new database and load the data from the specified version. • update : Update the views and procedures in the database without recreating the database or reloading the the tables. • extend : Extend the database by loading data from another package to the existing database. The suffix indicates the edition and version to which this applies: <ul style="list-style-type: none"> • latest : The most recent International Edition package • yyyymmdd: The International Edition for the stated year month and day • packageyyyymmdd: The identified Edition or Package for the stated year month and day.
Release package path	You must enter the full path of the release package folder or release package zip archive. <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p> You will not be prompted for this if you select an update script option.</p> </div>	The path specified must point to a release package zip archive or an unzipped release package folder. <ul style="list-style-type: none"> • There is no need to include the .zip extension when referring to a release package archive or folder. <ul style="list-style-type: none"> ▪ The script first looks for an unzipped release folder with the relevant name. ▪ If the folder is not found the script looks for a zip archive with the same name plus the .zip extension. ▪ If the zip file is found it is unzipped and the resulting folder is used.
Database name	Leave blank to accept the default	Default option <ul style="list-style-type: none"> • Uses the database name snomedct. • If the snomedct database already exists, it will be dropped (deleted) and recreated.
	Specify a name (must begin with the letter s followed by lowercase letters and/or digits)	Uses the database name provided. <ul style="list-style-type: none"> • Using different names allows several SNOMED CT databases to co-exist (e.g. for different Editions) • Each SNOMED CT database will use about 5Gb of disk space ... so using different names may fill your available disk space! • If the named database already exists, it will be dropped (deleted) and recreated.
MySQL username	Leave blank to accept the default	The default is root .
	Enter your MySQL username	The username chosen must be an account with administrator access rights enabling database creation.

Wait for the Database Password Prompt

Before the creating the database the script generates an additional release file containing the transitive closure (this is used to optimize testing and listing of subtypes). This may take between 2 and 5 minutes to complete.

- If you rerun the loader script again on the same release, you will be offered the option to reuse the existing transitive closure. If you accept the options, there will be no delay while the rebuild occurs.

Respond to the Database Password Prompt


When the script starts to access MySQL you will then be prompted for your database password.

- Note that the required password here is the password associated with your MySQL account.
- As noted earlier the account used for the SNOMED CT load process must have appropriate access permissions and its username should match your Mac login name.

Wait for the Load Process to Complete

- Depending on system performance the process may take between 20 and 90 minutes to complete. It may take longer with National Editions that contain additional content.
 - As the MySQL script runs it will report progress on the screen. Some steps take much longer than others. For example, loading data into the database tables and adding or building indexes take much longer than any of the other steps. So if the message about these steps are showing for a long time don't worry. Let the process continue.
- When the script completes, scroll back up the command window to check for any ERROR reports for MySQL ... there should not be any!
- Now it is time to open MySQL Workbench to view your SNOMED CT database.

A.4.5 Troubleshooting (Windows)

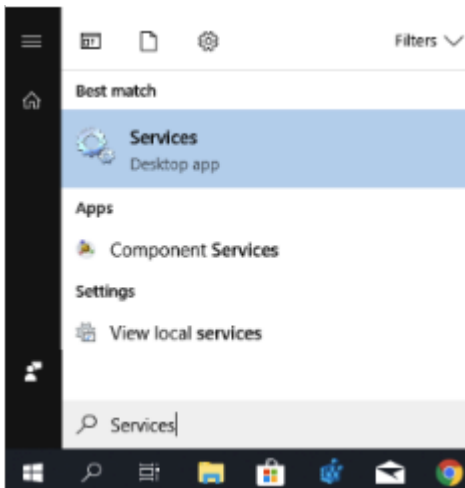
 This section contains notes on some known issues with configuration of MySQL for use with Windows and the actions required to resolve them.

MySQL Workbench Unable to Start or Stop MySQL Server (Windows)

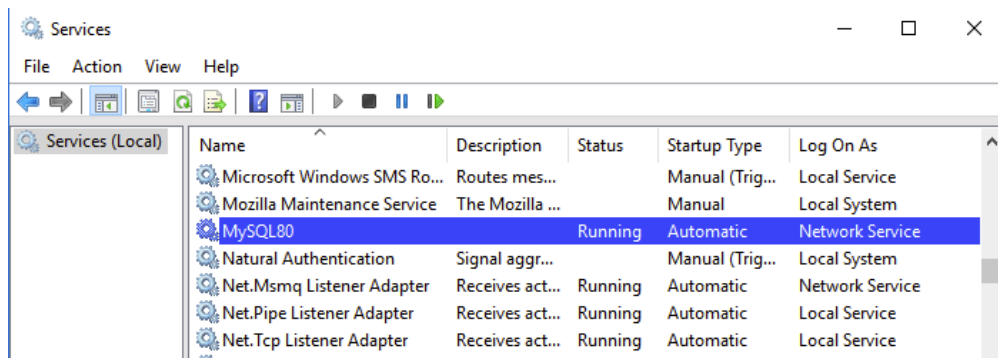
This issue usually results from an incorrect service name reference. This arises when the MySQL Workbench contains a reference to an older service name (e.g. **mysql**), whereas MySQL Server 8.0.x uses the service name **MySQL80**.

Check the Name of the Required Service

- Open the **Services** Desktop app from the Windows menu.

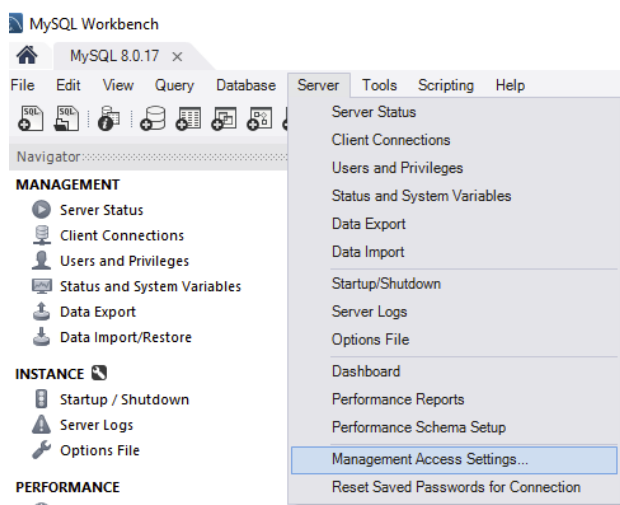


- Then search the list of services to identify the name of the MySQL service.

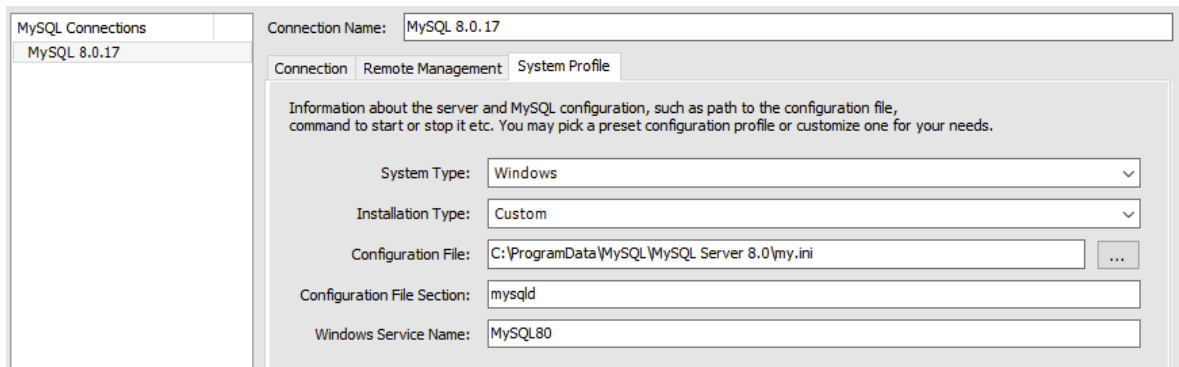


Correct the Reference in MySQL Workbench

- Open MySQL Workbench.
- In the Server menu select Management Access Settings.



- In the **Manage Server Connections** dialog update the Window Service Name to match the service name identified earlier.



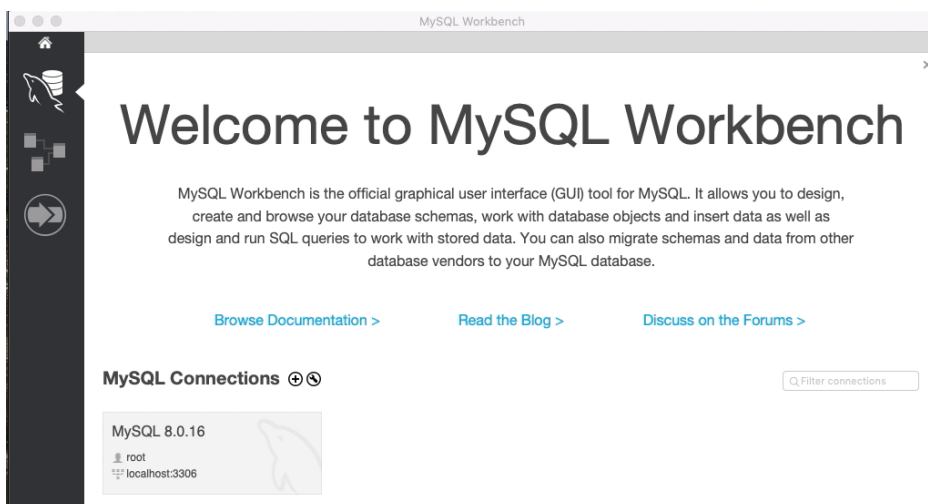
A.5 Using MySQL Workbench to Query SNOMED CT

Creating and Configuring a SNOMED CT Connection

Open MySQL Workbench Application



The following dialog should be displayed.



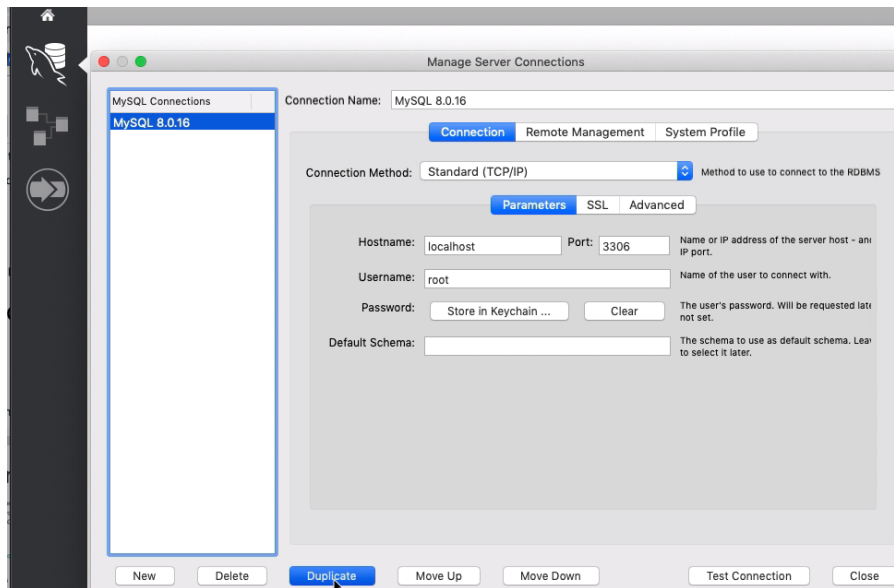
Create a SNOMEDCT Connection

- Click the spanner symbol to the right of the **MySQL Connections** prompt.



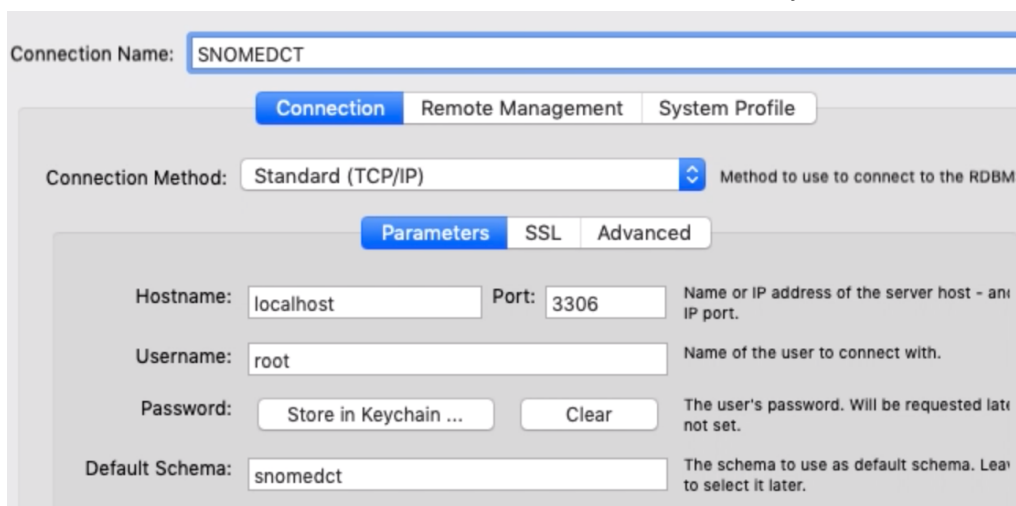
When the dialog below opens:

- Select the default MySQL connection
- Then click the **Duplicate** button

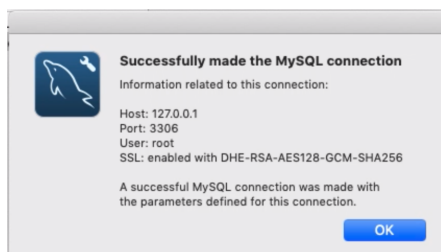


With the newly created connection selected:

- Change the Connection Name to **SNOMEDCT**
- Enter a Default Schema name as **snomedct** (i.e. the name of the newly created database)

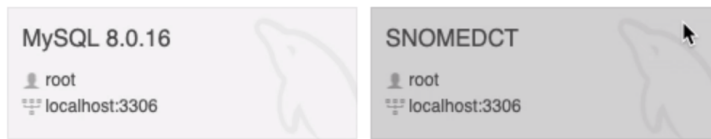


- Click the **Test Connection** button.
- If the connection is successful you will see the following dialog.




In future when you open MySQL Workbench you will see the option to open the SNOMEDCT connection.

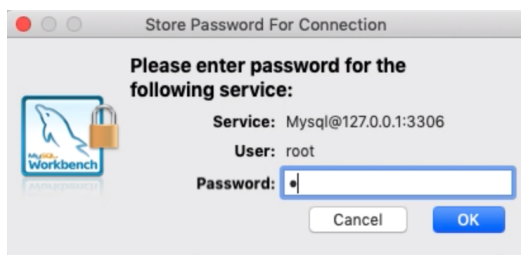
MySQL Connections



Save Your MySQL Password in Keychain

To avoid future prompts for the MySQL Password you can save the password in the Keychain.

- Click the  button.
- Enter the MySQL password to be stored.

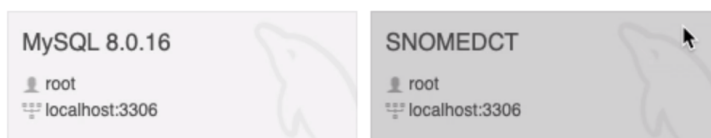


Testing and Using the SNOMED CT Database

Open the SNOMEDCT Connection

- Open MySQL Workbench
- Select the SNOMED CT Connection


MySQL Connections

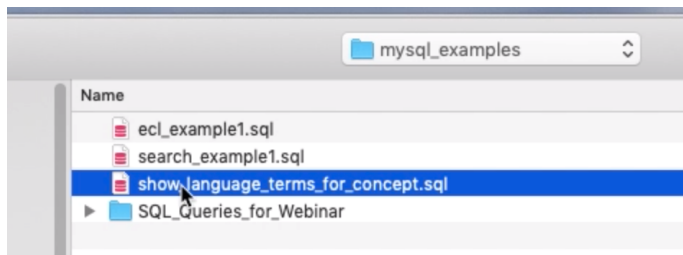


Test the SNOMED CT Database

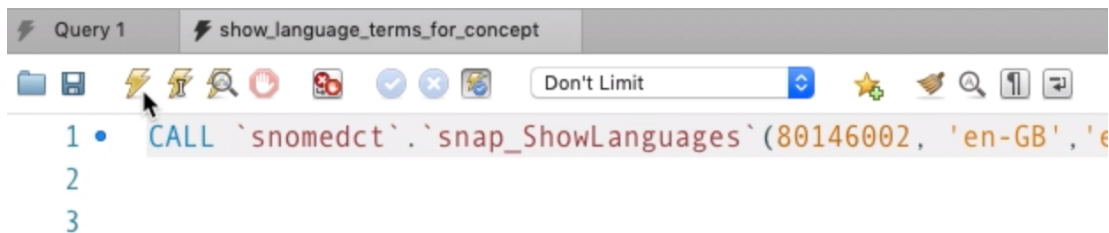
To test the loaded SNOMED CT database open one of the example queries in the **SnomedCtRfsMySQL/mysql_examples** folder.




- Click open SQL button  in the toolbar in the toolbar
- Find the **mysql_examples** folder in the SnomedCtRfsMySQL subfolder of your home directory.
- Select and open one of the queries

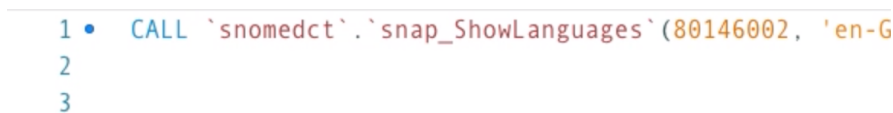


The query should be displayed as shown below



- Click the lightning icon  to run the query

The results of running the query should be displayed as below



conceptId	type_and_lang	term
▶ 80146002	FSN en-GB	Excision of appendix (procedure)
80146002	Preferred en-GB	Appendicectomy
80146002	Synonyms en-GB	Excision of appendix
80146002	FSN en-US	Excision of appendix (procedure)
80146002	Preferred en-US	Appendicectomy
80146002	Synonyms en-US	Excision of appendix

Using the SNOMED CT Database

The SNOMED CT database can be used in different ways:

- Running example SQL queries in the SnomedRfsMySql package.
- Running queries provided as part of a SNOMED CT E-Learning assignment.
- Using example SQL queries as templates for your own queries.
- Running SQL queries you have written yourself from scratch.

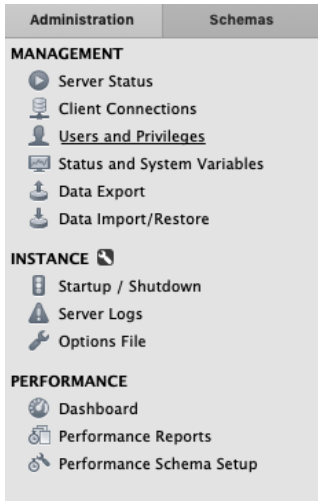
You can run your SQL queries in MySQL Workbench as described in the previous section. Additionally these queries can be run using the MySQL command line interface or through a MySQL connector for one of the programming languages listed at: <https://dev.mysql.com/downloads/>.

- ❗ If you are using example queries as templates for your own queries always copy the query first so that you do not overwrite the original example query.
 - If you accidentally overwrite an example query, you can download the SnomedRfsMySql.zip file again and extract the example queries folder.

Managing MySQL Accounts

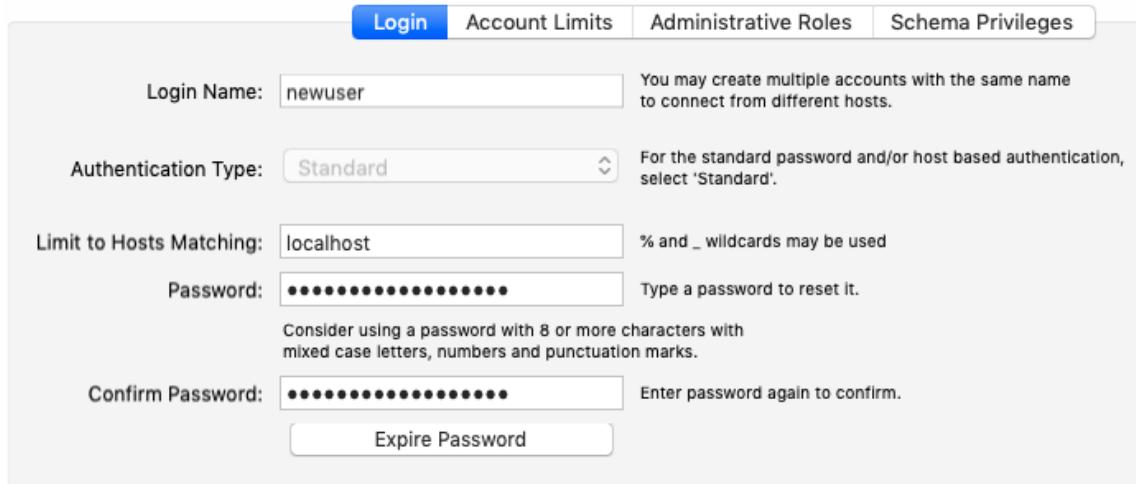
The "root" user account is typically used for importing the SNOMED CT release files. However, you should create additional MySQL accounts for database users that do not have administrator rights. Ideally the name of these MySQL accounts should match the Mac username of the user as this simplifies call the mysql command line tool and the use of MySQL connectors.

To create user accounts in the MySQL Workbench select the **Administration** tab (see below) and select **Users and Privileges** (see below).

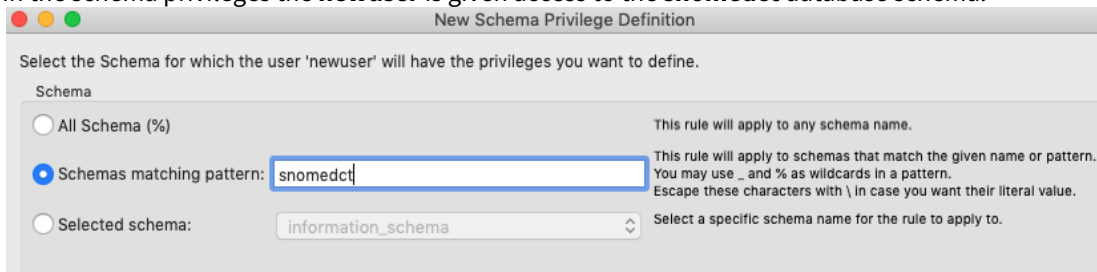


Set up the account details and access permissions.

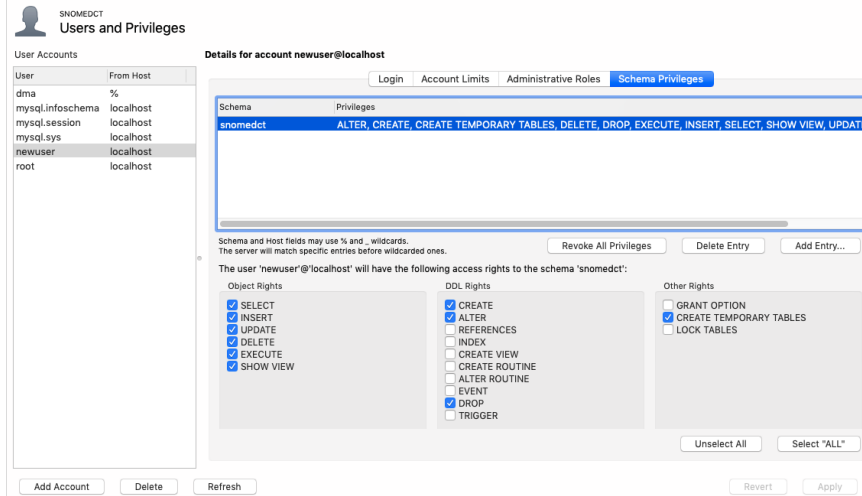
- In this example an account with the username **newuser** account is being created as shown below:



- In the schema privileges the **newuser** is given access to the **snomedct** database schema:



- Within the **snomedct** database the rights of this **newuser** are limited to the following actions.



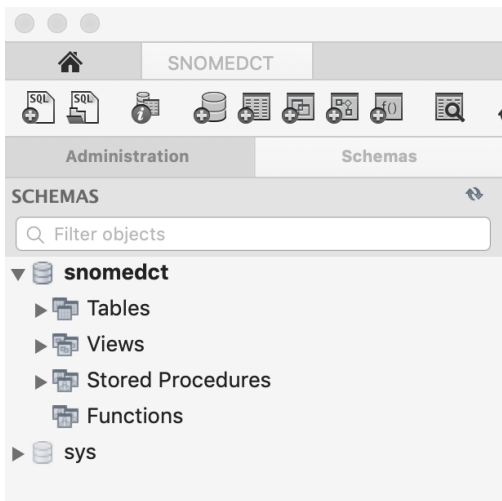
A.6 Overview of the SNOMED CT MySQL Database

Open the MySQL Workbench Schema Tab

Open MySQL Workbench and then select the Schema Tab.

You should see something like the image below.

- If you gave your database a name other than **snomedct** you will see the name you chose listed as a schema.



Make SNOMED CT the Default Database Schema

If the name of you SNOMED CT database schema is displayed in bold this means it is the default database.

- If your SNOMED CT database schema name is not shown in bold, double-click on the schema name. It will become the default and will be displayed in bold.

Tables in the Database

Expand the **Tables** Item under the SNOMED CT Database Schema name. This will reveal a list of table names.

All the listed table names have an initial prefix followed by an underscore character. The meaning of the prefixes used and the names and content of specific tables with these prefixes are summarized in the table below.

Prefix	Tables Using this Prefix	Table Names	Table Content
full	One table with the full prefix is created for each component.	These tables are named full_[component-type] (e.g. full_concept, full_description, full_relationship)	Each of these tables is populated with all the rows from the file (or files) representing this type component in the Full release subfolders.
	One table with the full prefix is also created for each reference set type present in the release.	The tables are named full_refset_[refset-type] (e.g. full_refset_Simple, full_refset_Language, full_refset_Association).	Each of these tables is populated with all the rows from the file (or files) representing reference sets of this type in the Full release subfolders.
snap	One table with the snap prefix is created for each component.	These tables are named snap_[component-type] (e.g. snap_concept, snap_description, snap_relationship)	Each of these tables is populated with all the rows from the file (or files) representing this type component in the Snapshot release subfolders.
	One table with the snap prefix is also created for each reference set type present in the release.	The tables are named snap_refset_[refset-type] (e.g. snap_refset_Simple, snap_refset_Language, snap_refset_Association).	Each of these tables is populated with all the rows from the file (or files) representing reference sets of this type in the Snapshot release subfolders.
	Additional tables with snap prefix are created to represent the transitive closure and proximal primitive supertype relationships	The table are named snap_transclose and snap_proximal_primitives .	The snap_transclose table is populated with all the rows from the transitive closure files generated during the SnomedRfsMySQL import process. The snap_proximal_primitives table is populated with proximal primitive relationships derived by processing the snap_transclose table .
config	Lookup and configuration files used by views and stored procedures. <ul style="list-style-type: none"> See sections below in Views and Procedures for further information. 	config_language	A table linking ISO language codes (e.g. en-US, en-GB, es) to the identifier of the relevant language reference set.
		config_settings	A table storing configuration settings that determine: <ol style="list-style-type: none"> The language reference set used to select synonyms and fully specified names The effectiveTime of two configurable retrospective snapshot views (snap1 and snap2) The effectiveTime range for two configurable delta views (delta1 and delta2)
		config_shortcuts	A table linking a short text keys to commonly used concept ids. This is used to facilitate constraining searches to concepts within these hierarchies without requiring the query to specify the full SNOMED CT identifier for the concept. <p>This is currently only used by the procedures snap_search_plus, snap_search1_plus and snap2_search_plus. In future it may also be used to support procedures with a common requirement for</p>

Views

SQL database views are in effect virtual tables. They can be queried in the same way as a table but they do not store data. The data that appears to be stored in a view is in fact defined by a stored query applied to the data stored in one or more tables.

The SNOMED CT import process creates two distinct types of views. Filtered views of a single table and composite views that bring together related data from different tables.

Filtered Table Views

The import process creates six distinct sets of table views. Five of these are applied to every Full release table. The naming conventions and characteristics of each of these filtered views summarized in the table below.

Pre fix	View Names	View Content
sn ap 1	snap1 <i>[component-type]</i> (e.g. snap1_concept, snap1_description) snap1_refset <i>[refset-type]</i> (e.g. snap1_refset_Simple)	These table views enable access to retrospective snapshots of the Full release data. The most recent version of every component in the table with an effectiveTime less than or equal to the snapshot date When the database is imported the snapshot dates are set as follows:
sn ap 2	snap2 <i>[component-type]</i> (e.g. snap2_concept, snap2_description) snap2_refset <i>[refset-type]</i> (e.g. snap2_refset_Simple)	<ul style="list-style-type: none"> • sn ap 1 views are set as a snapshot date 6 months before the current release • sn ap 2 views are set as a snapshot date 12 months before the current release These snapshot times can be changed by calling the stored procedure setSnapshotTime (viewNumber, dateTime). For example, to set the sn ap 1 date to 31 January 2017 <ul style="list-style-type: none"> • CALL setSnapshotTime(1, "20170131"); and to set the sn ap 2 date to 1 May 2016 <ul style="list-style-type: none"> • CALL setSnapshotTime(2, "20160501");
del ta	delta <i>[component-type]</i> (e.g. delta_concept, delta_description) delta_refset <i>[refset-type]</i> (e.g. delta_refset_Simple)	The delta table views enable access to delta views between any two dates. Only rows in the table with an effectiveTime greater than the start time and less than end time will be included in these views. When the database is imported the delta date ranges are set as follows:
del ta 1	delta1 <i>[component-type]</i> (e.g. delta1_concept, delta1_description) delta1_refset <i>[refset-type]</i> (e.g. delta1_refset_Simple)	<ul style="list-style-type: none"> • del ta views are set with a start date 6 months before the current release and an end date matching the current release date (this matches the current Delta release file content). • del ta 1 views start 12 months before the current release with an end date 6 months before the current release date. • del ta 2 views start 18 months before the current release with an end date 12 months before the current release date.
del ta 2	delta2 <i>[component-type]</i> (e.g. delta2_concept, delta2_description) delta2_refset <i>[refset-type]</i> (e.g. delta2_refset_Simple)	Delta date ranges can be changed by calling the stored procedure setDeltaRange (viewNumber, startDateTime, endDateTime). For example, to set the del ta view range to start on 31 July 2018 and end a year later <ul style="list-style-type: none"> • CALL setDeltaRange(0, "20180731", "20190731"); The del ta 1 and del ta 2 ranges can also be set in the same way <ul style="list-style-type: none"> • CALL setDeltaRange(1, "20170731", "20190731"); • CALL setDeltaRange(2, "20020131", "20070731");

An additional table view (with the prefix **sn ap asview**) provides a current snapshot view derived from the Full release. This is redundant in this database, because the import process imports the Snapshot release files as well as the Full release files. However, a few **sn ap asview** examples are included to provide examples of a views that could be used to avoid the need to import the Snapshot tables.

Composite Views

The table below summarizes the composite views supported by the database. Many of these composite views have variants that access specific snapshot views. These variants are indicated by the view prefixes **snap**, **snap1** and **snap2**. Note that the **snap** variants use the **snap** tables, while **snap1** and **snap2** variants use the relevant snapshot table views. Composite views that require access to the transitive closure table can only access the current snapshot (i.e. the **snap** tables). A few specific composite views are also relevant to the delta views and these have **delta**, **delta1** and **delta2** variants.

Composite View	Purpose	Snap Table and Views	Delta Views
fsn	Display of fully specified name for a specified <i>conceptid</i> .	All snapshot views	-
pref	Display of preferred synonym for a specified <i>conceptid</i> .	All snapshot views	-
syn	Display of acceptable synonyms for a specified <i>conceptid</i> .	All snapshot views	-
synall	Display of all valid synonyms (preferred and acceptable) for a specified <i>conceptid</i> .	All snapshot views	-
syn_search_active	All valid synonyms of active concepts. This is used as the substrate for searches.	All snapshot views	-
term_search_active	Fully specified name and all valid synonyms of active concepts. This can be used as an extended substrate for searches including fully specified names.	All snapshot views	-
rel_fsn	All relationships with fully specified names returned for sourceid (src_id, src_term), typeid (type_id, type_term) and destinationid (dest_id, dest_term) and relationshipGroup.	All snapshot views	-
rel_pref	All relationships with preferred synonyms returned for sourceid (src_id, src_term) typeid (type_id, type_term) and destinationid (dest_id, dest_term) and relationshipGroup.	All snapshot views	-
rel_def_fsn	All defining attribute relationships with fully specified names returned for sourceid (src_id, src_term), typeid (type_id, type_term) and destinationid (dest_id, dest_term) and relationshipGroup.	All snapshot views	-
rel_def_pref	All defining attribute relationships with preferred synonyms returned for sourceid (src_id, src_term) typeid (type_id, type_term) and destinationid (dest_id, dest_term) and relationshipGroup.	All snapshot views	-
rel_child_fsn	All direct subtypes of a concept (conceptId) returned using the id and fully specified name (id, term) of the child concept.	All snapshot views	-
rel_child_pref	All direct subtypes of a concept (conceptId) returned using the id and preferred synonym (id, term) of the child concept.	All snapshot views	-
rel_parent_fsn	All direct supertypes of a concept (conceptId) returned using the id and fully specified name (id, term) of the parent concept.	All snapshot views	-
rel_parent_pref	All direct supertypes of a concept (conceptId) returned using the id and preferred synonym (id, term) of the parent concept.	All snapshot views	-
transclose_pref	Transitive closure table view returned with subtype and supertype returned with id and preferred term.	Only snap table	-
proxprim_pref	Proximal primitive relationships closure table view returned with subtype and supertype returned with id and preferred term.	Only snap table	-
inactive_concepts	Returns all inactive concepts in a specified snapshot or delta view. The returned data includes the fully specified name of the concept, the reason for inactivation (from the concept inactivation reference set) and the associations with active concepts shown in the historical association reference sets.	All snapshot views	All delta views
inactive_descriptions	Returns all inactive descriptions in a specified snapshot or delta view. The returned data includes the fully specified name and active status of the described concept, and the reason for inactivation (from the description inactivation reference set),	All snapshot views	All delta views

Stored Procedures

Procedure	Description	View Prefix Support
snap_SearchPlus (<i>searchWords,filter</i>)	<p>Searches for acceptable synonyms of active concepts using a MySQL fulltext boolean search for the specified word or words. Word prefixed by "+" must be present, words prefixed by "-" but be absent and words with neither prefix will also be searched for but their absence from a term will not prevent a match.</p> <p>The filter can be used as follows to filter the search:</p> <ul style="list-style-type: none"> • Left blank: no filtering • < <i>conceptId</i> : only terms of concepts that are subtypes of the concept identified by <i>conceptId</i> will be included in the search results. • < <i>shortcutTerm</i> : only terms of concepts that are subtypes of the concept identified by looking up the <i>shortcutTerm</i> in the <i>config_shortcuts</i> table will be included in the search results • <i>regular-expression</i> : only terms that match the regular expression will be included in the search results • <i>!regular-expression</i> : only terms that do NOT match the regular expression will be included in the search results <p><u>Examples:</u></p> <pre>CALL snap_SearchPlus('fundus stomach', ''); CALL snap_SearchPlus('+fundus +stomach',''); CALL snap_SearchPlus('+lung +disease -chronic',''); CALL snap_SearchPlus('appendix','<proc'); CALL snap_SearchPlus('hemoglobin','<lab'); CALL snap_SearchPlus('infection','<19829001'); CALL snap_SearchPlus('+fundus', 'ch'); CALL snap_SearchPlus('fundus','!(eye ocul uter)'); CALL snap_SearchPlus('+lung +disease +chronic','oe?dema');</pre>	All snapshot views.
snap_ShowLanguages (<i>conceptId, languageCodeA, languageCodeB</i>)	<p>Shows the terms associated with a specified <i>conceptId</i> in two languages specified by the language codes.</p> <p><u>Example:</u></p> <pre>CALL `snap_ShowLanguages` (80146002, 'en-GB','en-US');</pre>	All snapshot views.
eclSimple (<i>expression-constraint</i>)	<p>Allows a fairly simple ECL expression to be processed. Maximum of one focus concept constraint optionally refined by up to two attribute value constraints.</p> <p><u>Example:</u></p> <pre>CALL `eclSimple` ('<404684003:363698007=<<39057004,116676008=<<415582006');</pre>	Only current snapshot
setLanguage (<i>viewNumber, languageCode</i>)	<p>Sets the language reference set that determines the terms to be displayed by composite views with names ending <i>_fsn</i>, <i>_pref</i>, <i>_syn</i>, <i>_synall</i>, <i>term</i>. The language and dialect code is used to specify the language (e.g. en-US, en-GB).</p> <p>If other values are supported by the SNOMED Edition, these will need to be added to the <i>config_languages</i> to provide the <i>refsetId</i> lookup from the language code.</p> <p><u>Example:</u></p> <pre>CALL `setLanguage` (0, "en-GB")</pre>	-
setDeltaRange (<i>viewNumber, startDateTime, endDateTime</i>)	<p>Sets the date range for a specified delta view (<i>viewNumber</i> 0=delta, 1=delta1, 2=delta2)</p> <p><u>Examples:</u></p> <pre>CALL setDeltaRange(0,"20180731","20190731"); CALL setDeltaRange(2,"20020131","20070731");</pre>	-

setSnapshotTime (<i>viewNumber, dateTime</i>)	Sets the date on which a specified snapshot view is based (viewNumber 1= snap1 , 2= snap2) <u>Example:</u> CALL setSnapshotTime(1,"20170131"); CALL setSnapshotTime(2,"20120131");	-
resetConfig ()	Resets the configuration file to the default initial starting snapshot time and delta range.	-
showConfig ()	Displays the configuration table settings for language, snapshot dates and delta ranges.	-

A.7 MySQL Reference Data

A.7.1 Required MySQL Configuration Settings

This page describes and explains the required additional MySQL configuration settings for the SNOMED CT example database. For instructions on how to apply these settings see [A.3.2 Set Required MySQL Configuration \(MacOS\)](#) or [A.4.2 Set Required MySQL Configuration \(Windows\)](#).

The following MySQL settings are required for loading and using the SNOMED CT MySQL database.

Setting	Explanation	Applies to
local-infile=1	Required to ensure that data can be loaded into the database from local files (e.g. SNOMED CT Release Files)	mysqld, mysql, client
ft_stopword_file = "" ft_min_word_len = 2	The two settings improve the full text search capabilities of the database. <ul style="list-style-type: none"> The first one removes the stop word list (which contains many words that are significant in clinical terms). An alternative approach would be a smaller stop word list but tests with SNOMED CT seem to suggest that this would not result in a significant improvement in performance. The second setting allows words that are 2 letters long to be indexed (the default setting is 4 which means terms like leg, arm, eye, ear ... are not found in searches). Reducing this to 3 resolves this issue but still means that common clinical abbreviations like MI, FH, RA as not indexed. 	mysqld
disable-log-bin skip-log-bin	These two settings stop the MySQL server from creating binary log files. Creation of these log files during the import process not only results in substantially slowing of the process but can also generate huge log files that more than double the space required for installation.	mysqld








The additional configuration settings required are specified in the following way in a file provided in the **SnomedRf sMySQL/cnf** folder.

Content of additional configuration file: my_snomedserver.cnf

```
[mysqld]
local-infile=1
ft_stopword_file = ''
ft_min_word_len = 2
disable-log-bin
skip-log-bin
default-authentication-plugin=mysql_native_password
[mysql]
local-infile=1
[client]
local-infile=1
protocol=tcp
host=localhost
port=3306
```

Appendix B: Obtaining SNOMED CT Release Files

To download SNOMED CT release files you must have a SNOMED CT Affiliate License. The table below provides step-by-step advice on how to obtain a license and register your license to enable you to download SNOMED CT release files.

Step 1	Do you already have a SNOMED CT Affiliate License? <ul style="list-style-type: none"> If you already have a license please skip to Step 6. If you do not have a license the steps below will help you to obtain a license. 	 If you are in a SNOMED International Member territory there will be no charge for licensing provided that you are not using SNOMED CT outside Member territories. However, you still need to register for a license. To learn more about SNOMED CT licensing please go to http://snomed.org/licensing .
Step 2	Are in a SNOMED International Member territory? <ul style="list-style-type: none"> If you are not in a Member territory skip to Step 6. 	 To check if you are based in a Member territory see the Members page on the SNOMED International website . On that page you will see a table listing all the Member territories and providing contacts details for each Member.
Step 3	Does your Member use the Member Licensing and Distribution Service (MLDS)? <ul style="list-style-type: none"> If your Member uses MLDS skip to Step 5. 	 Some SNOMED International Members use the SNOMED International MLDS service while others host their own licensing and download services. To find out if your Member uses MDLS service either: <ul style="list-style-type: none"> Look at the releases available for licensing and download from MLDS; or Follow the contact links for your Member on the SNOMED International Members webpage.
Step 4	Register with the licensing service provided by your Member. <ul style="list-style-type: none"> Finally please check Step 6 below. 	 Follow the contact links for your Member on the Members webpage . These links should take you to information about the licensing and distribution service used by the Member. <ul style="list-style-type: none"> If you are unable to find this information please use the email contact address for the Member as listed on the Members page.¹ In the unlikely event that you are still unable to find the necessary information please email info@snomed.org. <p>To find out more about licensing and downloading in these please follow the contact links on the Member</p>
Step 5	Register with MDLS.	 Register on the SNOMED International Member Licensing and Distribution Service
Step 6	Do you intend to use SNOMED CT in a non-Member Territory? <ul style="list-style-type: none"> If so you will also need to register this use on MDLS. 	 Note If you are intending to use SNOMED CT in a non-Member territory you must register this use on MDLS. This is true even for licensees who have registered their license using a service provided by a SNOMED International Member.  Register on the SNOMED International Member Licensing and Distribution Service

Appendix C: Release Types and Versioned Views

Release Types

SNOMED CT [release packages](#) include the following three distinct representations of the terminology content.

- A **full release**, which is a release type in which the release files contain every version of every component and reference set member ever released.
- A **snapshot release**, which is a release type in which the release files contain only the most recent version of every component and reference set member released, as at the release date.
- A **delta release**, which is a release type in which the release files contain only rows that represent component versions and reference set member versions created since the previous release date.

-
- See also [C.2. Release Type Support for Versioned Views](#).
-

Versioned Views


<p>A versioned view is formally defined as:</p> <ul style="list-style-type: none"> • A set of component versions and reference set member versions defined by characteristics of their effectiveTimes. <p>From a practical perspective, a versioned view is the result of filtering the full release files based on criteria that return a consistent representation of the content of these files as it was (or would have been) at a particular point in time.</p>	<p>Notes</p> <ul style="list-style-type: none"> • <i>Versioned views</i> and release types are closely related. A <i>release type</i> is a physical representation of a particular <i>versioned view</i>. • Some <i>versioned views</i> are not instantiated as <i>release types</i> but all valid <i>versioned views</i> of a SNOMED CT edition can be generated from a full release of that <i>edition</i>.
<p>The three main versioned views are:</p>	
<ul style="list-style-type: none"> • The full view, which is a view of SNOMED CT that includes all versions of all components and reference set members in a full release. 	
<ul style="list-style-type: none"> • A snapshot view which is a view of SNOMED CT that includes the most recent version of all components and reference set members at a specified point in time. 	<p>There are two distinct types of snapshot view:</p> <ul style="list-style-type: none"> • A current snapshot view, which is a snapshot view for the date of the most recent release. • A retrospective snapshot view, which is a snapshot view for a specified snapshot date.
<ul style="list-style-type: none"> • A delta view, which is a view of SNOMED CT that contains only rows that represent changes to components and reference set members since a specified date or between two specified dates in the past. 	<p>There are two distinct types of delta view:</p> <ul style="list-style-type: none"> • The current delta view, which is a delta view for the date range between the most recent release date and the immediately preceding release date. • A retrospective delta view, which is a delta view for a specified date range.

See also [4.6. Enabling Versioned Views](#)

C.1. Practical Uses for Versioned Views

The practical uses of these *views* are outlined in [Table C.1-1](#).

Table C.1-1: Practical Uses for Different Versioned Views

View	Uses
Full view	This view contains all versions of all released components and reference set members.  <ul style="list-style-type: none"> It can support all the uses identified in the following rows of this table.
Current snapshot view	This view contains only the most recent version of all released components and reference set members. <ul style="list-style-type: none"> It should be used for data entry. It can also be used to support for most types of retrieval and analysis. Detailed analysis of past use of concept that have since been made inactive may benefit from access to previous snapshots. It can also be used to generate the <i>current delta view</i> and thus also enables the uses identified for that view.
Retrospective snapshot view	A retrospective snapshot view has the same features as a current snapshot view but excludes changes made after a certain point in time. <ul style="list-style-type: none"> During and immediately after upgrading to a new release version, the previous snapshot view may be of particular value for determining details of changes. Earlier snapshot views can also be valuable when comparing and evaluating results of analyses which may have been affected by terminology changes.
Current delta view	This view only includes the post change state of components that have changed since the previous release. <ul style="list-style-type: none"> It can be used to identify components that have been added or inactivated since the previous release and this information can be used to check whether updates are needed. For example, updating, implementation resources such as local subsets, data entry picklists and queries. These updates may include removing inactive concepts or descriptions as well as adding relevant newly added concepts and descriptions. <ul style="list-style-type: none"> Note that some aspects of this review process also require access to the previous current snapshot view.
Retrospective delta view	A retrospective delta has the same features as a current delta view but relates to changes between two earlier dates. <ul style="list-style-type: none"> It can be used in conjunction with snapshot views

C.2. Release Type Support for Versioned Views

A key factor when deciding which *release type(s)* to import is the extent to which each release by supports required views of the terminology content. [Table C.2-1](#) lists the views that may be required and indicates which *release types* support each of these views.

Table C.2-1: Views Supported by Different Release Types

Supported Views		Release Types		
		Full	Snapshot	Delta
Full View	A view of SNOMED CT that includes all versions of all components and reference set members in a full release.	✓	✗	✗
Snapshot Views				

	Current Snapshot	A snapshot view for the date of the most recent release.	✓	✓	✗
	Retrospective Snapshot	A snapshot view for a specified snapshot date.	✓	✗	✗
Delta Views					
	Current Delta	A delta view for the date range between the most recent release date and the immediately preceding release date.	✓	✓	✓
	Retrospective Delta	A delta view for a specified date range.	✓	✗	✗

C.3. Common Mistakes with Snapshot Generation

The most common type of error when generating a snapshot is to ignore rule 3 and/or 4. For example, it is reasonable to want to filter a view so that only active components are included. However, if you apply that filter in a way that restricts the substrate from which the snapshot is generated you will get incorrect results in which all components appear to be active even after they have been inactivated. An example of this error is shown on

Table C.3-1: Full Release as at 2019-01-31

id	effectiveTime	active	value
A	20170131	1	Red
B	20170131	1	Amber
C	20170131	1	Yellow
A	20180131	0	Red
B	20180131	1	Orange
D	20180131	1	Green
B	20190131	0	Orange
E	20190131	1	Blue

The following query when applied to the full release data shown in [Table C.3-1](#) generates the erroneous snapshot view shown in [Table C.3-2](#). This query excludes inactive rows at the same time as identifying the most recent effectiveTime. The result is that components A and B are shown as active because the later rows that inactivated those components were excluded.

```
Select * from `component` `c` where
  `c`.`effectiveTime`=(Select max(effectiveTime) from `component` where `id`=`c`.`id`
and active=1)
```

Table C.3-2: Erroneous Snapshot view as at 2019-01-31

id	effectiveTime	active	value
A	20170131	1	Red
C	20170131	1	Yellow
B	20180131	1	Orange
D	20180131	1	Green
E	20190131	1	Blue

The following query when applied to the full release data shown in [Table C.3-1](#) generates the correct snapshot view shown in [Table C.3-3](#). The most recent versions of all the components A-E are included. In the case of components A and B these are both inactive.

```

Select * from `component` `c`
  where `c`.`effectiveTime`=(Select max(effectiveTime) from `component` where
`id`=`c`.`id`)
  
```

Table C.3-3: Erroneous Snapshot view as at 2019-01-31

id	effectiveTime	active	value
C	20170131	1	Yellow
A	20180131	0	Red
D	20180131	1	Green
B	20190131	0	Orange
E	20190131	1	Blue

The following query when applied to the full release data shown in [Table C.3-1](#) generates the correct snapshot view shown in [Table C.3-4](#) with inactive rows filtered out after generating the snapshot view. Only components C-E are included as the most recent versions of components A and B are inactive.

```

Select * from `component` `c`
  where `c`.`active`=1
  and `c`.`effectiveTime`=(Select max(effectiveTime) from `component` where
`id`=`c`.`id`)
  
```

Table C.3-3: Correct Snapshot view as at 2019-01-31 with inactive rows excluded

id	effectiveTime	active	value
C	20170131	1	Yellow
D	20180131	1	Green
E	20190131	1	Blue