



First Look: InterSystems SQL

Version 2019.4
2020-01-28

First Look: InterSystems SQL

InterSystems IRIS Data Platform Version 2019.4 2020-01-28

Copyright © 2020 InterSystems Corporation

All rights reserved.

InterSystems, InterSystems IRIS, InterSystems Caché, InterSystems Ensemble, and InterSystems HealthShare are registered trademarks of InterSystems Corporation.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: support@InterSystems.com

Table of Contents

First Look: InterSystems SQL	1
1 InterSystems SQL: Features and Performance	1
2 Demo: The SQL Shell	1
2.1 Before You Begin	2
2.2 Creating and Populating a Table With a SQL Script File	3
2.3 Running Queries Directly in SQL Shell	4
3 Demo: Using Bitmap Indexing To Maximize Query Performance	5
4 Learn More About InterSystems SQL	8
4.1 Introductory Material	8
4.2 SQL Development	8
4.3 Query Optimization	8
4.4 Sharding and Scalability	8
4.5 SQL Search	8
4.6 JDBC	8

First Look: InterSystems SQL

This First Look will acquaint you with the use of SQL with InterSystems IRIS® data platform: its industry-standard features, its unique capabilities, and how to get up and running with it quickly.

To browse all of the First Looks, including those that can be performed on a [free evaluation instance of InterSystems IRIS](#), see [InterSystems First Looks](#).

1 InterSystems SQL: Features and Performance

InterSystems IRIS provides high-performance, full-featured SQL. You can use SQL with InterSystems IRIS at scales from queries running on a single CPU core, to parallel queries using dozens of cores, to distributed queries across a cluster of InterSystems IRIS servers.

SQL features available in InterSystems IRIS at every scale include:

- Joins
- Flexible, high-performance indexing
- Aggregate functions and grouping
- Stored procedures written in SQL or InterSystems ObjectScript (referred to below as “ObjectScript”)
- JDBC and ODBC connectivity
- Automatic parallel query execution
- Transparently distributed queries

InterSystems SQL offers powerful tools to achieve optimal SQL query performance. One such tool is compressed *bitmap indexing*: using a compact, highly effective structure and vectorized CPU instructions, InterSystems SQL can perform aggregations and check logical conditions for billions of rows per second with just a single core. You’ll see an example of bitmap indexing later in this guide.

Want a quick demo of the SQL capabilities of InterSystems IRIS? Check out the [SQL QuickStart!](#)

2 Demo: The SQL Shell

You can execute SQL with InterSystems IRIS through a variety of APIs, interactive clients, and standard protocols, including:

- The InterSystems IRIS SQL Shell for interactive SQL statement execution
- ODBC and JDBC clients, either interactive (for example, SQuirreL SQL or WinSQL) or embedded in an application via an InterSystems IRIS driver
- The System Explorer in the InterSystems IRIS Management Portal, which offers an interactive web interface for SQL
- Embedded or dynamic SQL in an ObjectScript class

If, after working through this guide, you would like to explore more about any of these topics, see “[Learn More About InterSystems SQL](#)” below.

This demo shows you how to use the SQL Shell to execute SQL statements interactively or from a file.

2.1 Before You Begin

To use the procedure, you will need a running InterSystems IRIS instance. Your choices include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see [Deploying InterSystems IRIS](#) in *InterSystems IRIS Basics: Connecting an IDE*.

You will also need to obtain utility files for this guide from the GitHub repo <https://github.com/interSystems/FirstLook-SQLBasics>. You should clone the repository to download the following files:

- `stock_table_demo_one.sql`, which contains SQL statements to create and load a small (20-row) table of stock data
- `stock_table_demo_two.csv`, which contains a million rows of stock table data
- `Loader.xml`, a class file that contains a utility method to load the data from `stock_table_demo_two.csv` into an InterSystems IRIS table

Note: To download `stock_table_demo_two.csv`, which is very large, you first need to install [Git Large File Storage](#).

The FirstLook-SQLBasics sources must be accessible by the instance. The procedure for downloading the files depends on [the type of instance](#) you are using, as follows:

- If you are using an ICM-deployed instance:

1. Use the `icm ssh` command with the `-machine` and `-interactive` options to open your default shell on the node hosting the instance, for example:

```
icm ssh -machine MYIRIS-AM-TEST-0004 -interactive
```

2. On the Linux command line, use one of the following commands to clone the repo to the [data storage volume](#) for the instance. For a configuration deployed on Azure, for example, the [default mount point](#) for the data volume is `/dev/sdd`, so you would use commands like the following:

```
$ git clone https://github.com/interSystems/FirstLook-SQLBasics /dev/sdd/FirstLook-SQLBasics
OR
$ wget -qO- https://github.com/interSystems/FirstLook-SQLBasics/archive/master.tar.gz | tar xvz
-C /dev/sdd
```

The files are now available to InterSystems IRIS in `/irissys/data/FirstLook-SQLBasics` on the container’s file system.

- If you are using a containerized instance (licensed or Community Edition) that you deployed by other means:
 1. Open a Linux command line on the host. (If you are using Community Edition on a cloud node, [connect to the node using SSH](#), as described in *Getting Started with InterSystems IRIS Community Edition*.)
 2. On the Linux command line, use either the `git clone` or the `wget` command, as described above, to clone the repo to a storage location that is mounted as a volume in the container.
 - For a Community Edition instance, you can clone to the instance’s [durable %SYS directory](#) (where instance-specific configuration data is stored). On the Linux file system, this directory is `/opt/ISC/dur`. This makes the files available to InterSystems IRIS in `/ISC/dur/FirstLook-SQLBasics` on the container’s file system.
 - For a licensed containerized instance, choose any storage location that is mounted as a volume in the container (including the durable %SYS directory if you use it). For example, if your `docker run` command included

the option `-v /home/user1:/external`, and you clone the repo to `/home/user1`, the files are available to InterSystems IRIS in `/external/FirstLook-SQLBasics` on the container's file system.

- If you are using an InterSystems Learning Labs instance:
 1. Open the command-line terminal in the integrated IDE.
 2. Change directories to `/home/project/shared` and use the `git clone` command to clone the repo:

```
$ git clone https://github.com/intersystems/FirstLook-SQLBasics
```

The folder is added to the Explorer panel on the left under **Shared**, and the directory is available to InterSystems IRIS in `/home/project/shared`.

- If you are using an installed instance:
 - If the instance's host is a Windows system with GitHub Desktop and GitHub Large File Storage installed:
 1. Go to <https://github.com/intersystems/FirstLook-SQLBasics> in a web browser on the host.
 2. Select **Clone or download** and then choose **Open in Desktop**.

The files are available to InterSystems IRIS in your GitHub directory, for example in `C:\Users\User1\Documents\GitHub\FirstLook-SQLBasics`.

- If the host is a Linux system, simply use the `git clone` command or the `wget` command on the Linux command line to clone the repo to the location of your choice.

2.2 Creating and Populating a Table With a SQL Script File

For the purposes of this demo, we'll use a SQL script file, `stock_table_demo_one.sql`, to create and load a table with a few rows of sample data.

To create and load the table:

1. Open the InterSystems IRIS Terminal using the [procedure described for your instance](#) in *InterSystems IRIS Basics: Connecting an IDE*. You will see the following interactive prompt:

```
USER>
```

This prompt indicates that you are currently in the `USER` namespace, which is empty by default and reserved for your use. From this prompt, you can execute ObjectScript.

2. Open the SQL Shell by entering

```
DO $SYSTEM.SQL.Shell()
```

at the prompt. This will display the following output:

```
SQL Command Line Shell
-----
The command prefix is currently set to: <<nothing>>.
Enter q to quit, ? for help.
[SQL]USER>>
```

3. Set the current SQL dialect to IRIS:

```
SET DIALECT=IRIS
```

4. To run the statements in `stock_table_demo_one.sql`, enter the command

```
RUN <Path>stock_table_demo_one.sql
```

where *Path* is the location in which you placed the file (see [Before You Begin](#)). You are prompted to specify names for log files containing the statements in the file and their output, how to handle errors, and the statement delimiter. Accept all defaults.

The statements create a table and insert 20 rows. The first few lines of the file are:

```
CREATE TABLE FirstLook.StockTableDemoOne (ClientID INTEGER, BrokerID INTEGER,
      Symbol VARCHAR(10), TransactionType VARCHAR(4), TransactionDate TIMESTAMP,
      Quantity INTEGER, Price DECIMAL(15,2), CommmissionRate DECIMAL(15,2))
GO
INSERT INTO FirstLook.StockTableDemoOne (ClientID, BrokerID, Symbol,
      TransactionType, TransactionDate, Quantity, Price, CommmissionRate)
VALUES (29834783, 3103, 'RTYU', 'SELL', '2016-01-03', 342, 5.05, 3.25)
GO
```

As the script runs, you'll see output after each SQL statement is processed:

```
1. INSERT INTO FirstLook.StockTableDemoOne (ClientID, BrokerID, Symbol,
2.   TransactionType, TransactionDate, Quantity,
3.   Price, CommissionRate)
4.   VALUES (92609349, 3103, 'HWVT', 'BUY', '2017-10-25', 1500, 451.09, 3.25)
1 Row Affected
```

After all statements are processed, the SQL Shell lists the number of statements compiled as well as errors and warnings reported, and reports the elapsed time:

```
Statements
.....compiled: 21
.....with errors reported: 0
...with warnings reported: 0

Elapsed time: .125181 seconds
```

2.3 Running Queries Directly in SQL Shell

Now that you have a populated table, you can run queries against it. You can use single-line or multiline mode to do this, but may find the latter more convenient.

- To enter multiline mode, press Enter at the prompt. You'll see confirmation that you're in multiline mode.
- Enter the following SQL syntax, line by line. The keyword GO instructs the shell to execute the query and exit multiline mode:

```
SELECT BrokerID, TO_CHAR((Quantity * Price), '9,999,999.99') as SubTotal,
      TransactionDate FROM FirstLook.StockTableDemoOne
WHERE TransactionType='SELL'
ORDER BY SubTotal DESC
GO
```

The statement you entered will be echoed to the SQL Shell, and query results will follow.

```
2.      SELECT BrokerID, TO_CHAR((Quantity * Price), '9,999,999.99') as SubTotal,
          TransactionDate FROM FirstLook.StockTableDemoOne
          WHERE TransactionType='SELL'
          ORDER BY SubTotal DESC
```

BrokerID	SubTotal	TransactionDate
5001	302,780.00	2017-11-06 09:51:24.735
5002	92,350.00	2018-01-15 22:21:17.638
3103	57,645.00	2017-09-24 19:36:43.079
3103	45,015.00	2016-10-31 19:21:08.913
5001	23,180.50	2017-07-31 23:05:49.83
5001	13,113.60	2015-11-13 22:13:49.457
5001	12,636.00	2015-10-13 05:50:23.209
3103	1,727.10	2016-01-03 13:59:01.098
1009	1,693.50	2016-01-15 18:18:15.346

After the query results, you'll see information on how long it took to prepare and execute the statements:


```

9 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.0625s/47683/263292/0ms
execute time(s)/globals/cmds/disk: 0.0006s/64/2903/0ms
cached query class: %sqlcq.USER.cls47

```

The preparation step includes the generation of executable code from the syntax of a SQL statement. This code is cached for re-use, so a statement is typically prepared fully only once. Subsequent preparations need only locate the cached code using a hash of the statement's text.

The execution step includes executing the code that was generated for a query and returning its results.

Within each step's listing are the following metrics:

- The time each step took.
- The count of *globals*, which is the number of references that were made to InterSystems IRIS storage to prepare or execute the SQL statement. For more information on globals, see the “[Introduction to Globals](#)” chapter of the *Orientation Guide for Server-Side Programming*.
- The count of ObjectScript *commands* that were executed to prepare or execute the SQL statement.

At the end of the display is the *cached query class*, which is the ObjectScript class that caches the code generated when the statement is first prepared.

- Aggregate functions and GROUP BY are also available. Note that you can order by the alias used for the aggregate function:

```

SELECT BrokerID, TO_CHAR(SUM(Quantity * Price), '9,999,999.99') as SubTotal
FROM FirstLook.StockTableDemoOne
GROUP BY BrokerID
ORDER BY SubTotal DESC
GO

```

```

2.      SELECT BrokerID, TO_CHAR(SUM(Quantity * Price), '9,999,999.99') as SubTotal
        FROM FirstLook.StockTableDemoOne
        GROUP BY BrokerID
        ORDER BY SubTotal DESC

```

BrokerID	SubTotal
3103	868,993.60
1009	808,453.50
5001	593,242.82
5002	187,560.00

```

4 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.1665s/45832/237712/77ms
execute time(s)/globals/cmds/disk: 0.0025s/122/2434/2ms
cached query class: %sqlcq.USER.cls9

```

3 Demo: Using Bitmap Indexing To Maximize Query Performance

If you are working with large data sets, you will need ways to tune query performance. Bitmap indexing is one of several methods available to you.

Bitmap indexing is especially advantageous if a table has one or more fields whose set of possible values is small.

For in-depth information on how bitmap indexing works, see the “[Bitmap Indices](#)” chapter of the *InterSystems SQL Optimization Guide*.

In this demo, you'll see the effects of targeted bitmap index creation on a million-row table of stock transaction data. You'll be using a couple of simple ObjectScript commands along the way; it's easy to access the ObjectScript library seamlessly from within the SQL Shell.

To run the demo:

1. Start a SQL Shell in Terminal as described in “[Creating and Populating a Table With a SQL Script File](#)”.
2. Create the table:

```
CREATE TABLE FirstLook.StockTableDemoTwo (ClientID INTEGER, BrokerID INTEGER,
Symbol VARCHAR(10), TransactionType VARCHAR(4),
TransactionDate TIMESTAMP, Quantity INTEGER,
Price DECIMAL(15,2), CommissionRate DECIMAL(15,2))
```

```
1. CREATE TABLE FirstLook.StockTableDemoTwo (ClientID INTEGER, BrokerID INTEGER,
Symbol VARCHAR(10), TransactionType VARCHAR(4),
TransactionDate TIMESTAMP, Quantity INTEGER,
Price DECIMAL(15,2), CommissionRate DECIMAL(15,2))
```

```
0 Rows Affected
statement prepare time(s)/globals/cmds/disk: 0.0063s/1811/22260/0ms
execute time(s)/globals/cmds/disk: 0.2138s/76495/655985/76ms
cached query class: %sqlcq.USER.cls1
```

3. Import the Loader class (the Loader.xml file). The OBJ prefix instructs the SQL Shell to handle the command that follows as ObjectScript.; the "c" flag instructs InterSystems IRIS to compile the code, and the "k" flag ensures that the source code is stored in the active namespace.

```
OBJ DO $system.OBJ.Load("<Path>Loader.xml", "ck")
```

where *Path* is the location in which you placed the file (see [Before You Begin](#)). You should see output like the following:

```
Load started on 04/19/2018 15:17:53
Loading file C:\Users\user\repos\FirstLook-SQLBasics\Loader.xml as xml
Imported class: FirstLook.Loader
Compiling class FirstLook.Loader
Compiling routine FirstLook.Loader.1
Load finished successfully.
```

4. To load the data in stock_table_demo_two.csv into the table, run the following command in Terminal:

```
OBJ WRITE ##class(FirstLook.Loader).LoadStockTableCSV("<Path>stock_table_demo_two.csv")
```

where *Path* is the location in which you placed the file. The output of this command, 1000000, indicates simply that 1,000,000 rows were loaded.

5. Run the following query:

```
SELECT DISTINCT BrokerID FROM FirstLook.StockTableDemoTwo
```

The output shows that the number of possible broker IDs is very small, making this field a good candidate for bitmap indexing.

```
2. SELECT DISTINCT BrokerID FROM FirstLook.StockTableDemoTwo
```

```
BrokerID
115
107
101
114
119
104
109
108
102
116
110
120
112
106
111
113
```

```
105
118
103
117
```

```
20 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.0645s/43430/197693/9ms
execute time(s)/globals/cmds/disk: 1.2569s/2000039/9001314/0ms
cached query class: %sqlcq.USER.cls10
```

6. To see the performance of a COUNT query involving the BrokerID field before you add a bitmap index, run the following query:

```
SELECT BrokerID, COUNT(*) As Transactions FROM FirstLook.StockTableDemoTwo
GROUP BY BrokerId ORDER BY Transactions DESC
```

```
3. SELECT BrokerID, COUNT(*) As Transactions FROM FirstLook.StockTableDemoTwo
GROUP BY BrokerId ORDER BY Transactions DESC
```

BrokerId	Transactions
103	50386
118	50304
107	50247
112	50207
101	50174
109	50088
115	50088
104	50048
111	50031
105	50008
113	49996
119	49942
114	49919
116	49894
110	49888
108	49882
102	49843
120	49768
106	49742
117	49545

```
20 Rows(s) Affected
```

Observe the query performance statistics that are displayed after the query returns results: the total time elapsed (including both preparation and execution time) is approximately 0.65 seconds.

```
statement prepare time(s)/globals/cmds/disk: 0.0695s/45048/225490/13ms
execute time(s)/globals/cmds/disk: 0.5878s/1000250/11002218/0ms
cached query class: %sqlcq.USER.cls7
```

7. Add a bitmap index on BrokerID:

```
CREATE BITMAP INDEX BrokerIDIdx ON TABLE FirstLook.StockTableDemoTwo (BrokerID)
```

```
4. CREATE BITMAP INDEX BrokerIDIdx ON TABLE FirstLook.StockTableDemoTwo (BrokerID)
```

```
0 Rows Affected
statement prepare time(s)/globals/cmds/disk: 0.0056s/1723/15958/0ms
execute time(s)/globals/cmds/disk: 0.9805s/2071557/18505697/1ms
cached query class: %sqlcq.USER.cls11
```

8. Run the same SELECT query as you did above. Note the improvement in performance: in the example below, the query took approximately 0.35 seconds total, a decrease of nearly 50 percent.

```
SELECT BrokerID, COUNT(*) As Transactions FROM FirstLook.StockTableDemoTwo
GROUP BY BrokerId ORDER BY Transactions DESC
```

```
...
```

```
statement prepare time(s)/globals/cmds/disk: 0.0573s/45585/231374/0ms
execute time(s)/globals/cmds/disk: 0.2926s/622/15004397/0ms
cached query class: %sqlcq.USER.cls1
```

4 Learn More About InterSystems SQL

To learn more about SQL and InterSystems IRIS, see:

4.1 Introductory Material

- [Using InterSystems SQL](#)
- [InterSystems SQL Reference](#)
- [InterSystems SQL Overview](#)

4.2 SQL Development

- [SQL – Things You Should Know](#)
- [Learn InterSystems SQL: Design and Execution](#)
- [Developing with InterSystems Objects and SQL](#)

4.3 Query Optimization

- [First Look: Optimizing SQL Performance with InterSystems IRIS](#)
- [InterSystems SQL Optimization Guide](#)
- [Academy – Optimizing SQL Performance](#)
- [Optimizing SQL Queries](#)
- [Learn InterSystems SQL: Performance](#)

4.4 Sharding and Scalability

- [First Look: Scaling for Data Volume with Sharding](#)
- [Scalability Guide](#)

4.5 SQL Search

- [First Look: SQL Search with InterSystems IRIS](#)
- [Using InterSystems SQL Search](#)
- [Creating iFind Indices for Searching Text Fields](#)

4.6 JDBC

- [First Look: JDBC and InterSystems IRIS](#)
- [Using Java JDBC with InterSystems IRIS \(documentation\)](#)
- [Java Overview](#)

- [Using JDBC with InterSystems IRIS \(online learning\)](#)

