EmberWings

2024/1

IBPhoenix
THE POWER WITHIN

March 2024

www.ibphoenix.com

emberwings@ibphoenix.com

In This Issue:

Setting the Blaze: Introducing EmberWings Magazine

Establishing a computer magazine with a highly specialized focus in today's context may initially seem unconventional. However, we are confident that the professional community centered around Firebird can benefit from such a venture, particularly given the current landscape of rapid and transient information with a limited shelf life. The structured format, consistent style, and varied content found in traditional publications offer attributes not readily available in blogs, podcasts, video channels, discussion forums, or social media. This includes genuine, practical predictability and, crucially, content durability that allows for unrestricted revisiting.

Beyond technical articles, instructions, and tips, we would like regularly bring you interviews with noteworthy individuals, an overview of Firebird's development, information from the Firebird community, evaluations of compelling products, and a touch of thematic humor. All of this presented in a streamlined format with an aesthetically pleasing print-friendly layout.

EmberWings is projected as a quarterly magazine, and although we cannot guarantee the regularity of a new issue on a specific day or week of the month, we can promise four issues each year.

The creation of this initial issue presented its own set of challenges, as we navigated and continue to navigate through a multitude of new practices and technologies. We sincerely hope for your understanding and assistance in overcoming any potential shortcomings we may encounter on this journey.

Your EmberWings editors

In the ever-changing blaze of the digital fire, a devoted young apprentice sought wisdom from the tribal shaman on the sacred ways to elevate the Firebird database server to a new major version.

"O Shaman," the apprentice asked, "how do we traverse the sacred flames of progress, ensuring that our tribal data, like embers of ancient wisdom, continues to burn brightly as we ascend to the realms of a new Firebird version?"

The shaman, surrounded by the soft crackle of digital flames, began to share ancient insights. "Envision the Firebird server as the eternal flame of our tribal knowledge, its essence persisting through cycles of rebirth. Approach the upgrade like the dance of phoenix feathers – with respect, preparation, and a seamless transition from the ashes of the old to the wings of the new.

Inquisitive, the apprentice asked, "But how do we ensure a smooth flight through the realms of version upgrades without leaving the tribal data vulnerable to the winds of instability?"

The shaman, with eyes reflecting the radiant glow of upgrade paths, replied, "Embrace the rituals of preparation. Capture the essence of the tribal flame through thorough backups. Consult the ancient scrolls – the release notes and documentation – for guidance on the journey ahead. Perform tests in a controlled environment, as a cautious rehearsal before the grand dance of the upgrade. Engage the tribal community, sharing the vision of progress and ensuring collective strength through the journey."

As the young apprentice absorbed these teachings, he understood that the upgrade of the Firebird database server was not just a technical task but a sacred passage – a passage into a new era where the tribal data would continue to dance with the fiery spirit of the latest Firebird version

---

This short story may strike some readers as entirely inappropriate for a Firebird magazine. At the same time, it is directly saturated with all essential information on the given topic in a minimal space, layered on several levels. However, its form is completely unusual for today's digital age, and therefore it may seem out of place to some.

Today, the transfer of knowledge, experience and skills is reduced to more or less developed descriptions of states and processes, and ways to achieve the desired states and processes. It is completely free of colorful descriptions and similes, not to mention multiple layers of meaning. It is a natural development at a time when it is necessary to impart rapidly changing knowledge and skills to an ever-increasing number of people. It's basically the only effective way. But we pay a high price for it.

The absence of literalness, imagery and layering of the old ways helps to develop the abilities absolutely necessary to achieve true mastery and the resulting pioneering: creativity fueled by imagination, and the ability to look passionately behind the curtain of the obvious.

That's why we want to present you with not only ready-made solutions, but also possibilities for their further development, including just ideas to think about. And here and there some database koan...

Updating a Database Server: The Basics

Regularly updating a database server is a critical aspect of maintaining a healthy and efficient information system. The database server, serving as the backbone of data storage and retrieval, requires updates to address security vulnerabilities, fix bugs, improve performance, introduce new features, and ensure compatibility with evolving technologies. This article (which is first in the series of articles dedicated to this topic) explores the intricacies surrounding the update process, shedding light on general practices, and highlighting key differences based on whether the upgrade is an integral part of the application development process or performed by a third-party application user.

Why Perform Updates?

- **Security Updates:** One of the primary motivations for updating a database server is to address security vulnerabilities. As technology advances, new threats emerge, and updates often include patches and fixes to safeguard against potential exploits.
- **Bug Fixes:** Database servers, like any software, may have bugs or glitches that impact their functionality. Updates commonly include bug fixes to rectify issues identified through user feedback, testing, or continuous monitoring.
- **Performance Improvements:** Updates frequently bring performance improvements, optimizing query execution times, resource utilization, and overall database responsiveness. These enhancements contribute to a more efficient and streamlined system.
- **Compatibility with Applications:** As applications evolve, database servers must keep pace to maintain compatibility. Updates ensure that the database remains synchronized with the latest software and application versions, preventing compatibility issues.
- **New Features:** Updates can introduce new features and functionalities to the database server, providing users with enhanced capabilities. These additions contribute to the ongoing development and innovation of the information system.

Risks in Database Server Updates

With the benefits come inherent risks that organizations must navigate carefully. Understanding these potential pitfalls is crucial for effective planning and execution of database server updates.

- **Data Loss:** One of the foremost risks associated with database server updates is the potential for data loss. Schema changes, alterations to data structures, or migration processes during updates may lead to unintended data loss if not handled meticulously.

- **Downtime and Service Disruption:** Database updates often require system downtime, impacting the availability of services. Inadequate planning, unexpected issues, or a failure to estimate the required downtime accurately can lead to prolonged service disruptions, affecting critical business operations.
- **Application Compatibility Issues:** Changes introduced during updates may not align seamlessly with existing applications. This can result in compatibility issues, causing errors, unexpected behavior, or functionality problems within applications dependent on the database.
- **Performance Degradation:** While updates aim to enhance performance, there is a risk of unintended consequences. Changes to query execution plans or side effects of code changes impacting operating characteristics can lead to performance degradation, undermining the system's efficiency.
- **Security Vulnerabilities:** Despite efforts to address security vulnerabilities, updates may inadvertently introduce new security risks. Misconfigurations or errors in applying security patches can expose the system to potential threats.
- **Dependency Issues:** Updates may have dependencies on other software components. Ignoring or overlooking these dependencies can result in issues with the overall system, leading to unexpected errors or failures. One such component in the case of Firebird is the ICU library.
- **Rollback Challenges:** In the event of a failed update, rolling back to the previous version can be challenging. Changes made during the update process, such as modifications to the database structure or data, may not be easily reversible, prolonging the recovery process.
- **Inadequate Testing:** Insufficient testing before deploying an update is a significant risk. Bugs, performance bottlenecks, or compatibility problems may only surface in a production environment, potentially causing disruptions that could have been avoided with thorough testing.
- **Unforeseen Compatibility Issues with Hardware or Software:** Changes introduced in an update may not align with the existing hardware or software stack. This can lead to system instability or unexpected behavior, posing risks to the overall system reliability.

- **User Resistance and Training Needs:** Users may encounter difficulties adapting to changes introduced by updates. This can result in resistance, increased support requests, and a need for additional training, affecting the overall user experience.
- **Lack of Vendor Support:** Using outdated database versions, especially those reaching end-of-life, can expose the system to security risks. Vendors may cease to provide support or release updates, leaving the system vulnerable to emerging threats.

Navigating these risks requires a comprehensive approach encompassing thorough planning, rigorous testing, and a robust rollback strategy. Effective communication with stakeholders, including end-users and IT teams, is paramount to managing expectations and addressing challenges that may arise during the update process. Regular backups and a well-defined rollback plan provide a safety net, ensuring resilience in the face of unexpected issues. Ultimately, a balanced approach that considers both risks and benefits is key to a successful database server update.

Differences between updates as part of application development and updates in user deployment

A database server is a means of running a particular application, and there are differences when updating the server if it is done by the application developers (whether it is internal development within the organization or by the application vendor) or by a mere user of the application from a third party vendor.

Development Updates

When a database server update is related to the development of a specific application, the process typically follows these steps:

- **Development Cycle Planning:** The future database server updates should be always part of the overall application development cycle. It is advisable to continuously monitor the development of Firebird and, if possible, continuously test the compatibility of the newly developed version of Firebird with the developed applications. Testing early versions of Firebird also allows early detection of possible pitfalls during migration, and evaluation of the usefulness of new functions for further development of own applications.
- **Testing in Isolated Environment:** The update is initially tested in an isolated development environment, allowing for the identification and resolution of potential issues before deployment to the production environment.
- **Integration with Application Code:** Database server updates may require changes to the application code to maintain compatibility. These changes are integrated with the main application code.

Deployment Updates

In the case where a third-party application user utilizes a database server, the process may differ:

- **Notification from Vendor:** The vendor informs users about the availability of a new database server version, providing information about changes and improvements. In an ideal world, the application vendor also provides assistance with upgrading the database server, at least in the form of documentation on how to do it.
- **Testing in a Test Environment:** Users are encouraged to test the update in a controlled test environment. This phase allows them to evaluate the impact of the update on their specific use cases, ensuring that the new version aligns with their application's requirements.
- **Migration Planning:** Users prepare a migration plan, including data backup, ensuring compatibility with other applications, and planning for rollback if needed.

In the case of developing applications for use within the organization, the procedure for updating Firebird is a synthesis of both approaches.

Database Server Update: General Guidelines

Updating a database server in a production environment is a critical process that requires careful planning and execution. Here are the basic steps and important points to consider:

- **Testing in a test environment:** Try the update first in an isolated test environment that simulates production conditions as closely as possible. This will make it possible to detect potential problems before the actual update in production.

This is important for major version updates, but can be reduced to a bare minimum for minor version updates.

- **Application compatibility verification:** Make sure that all applications that use that database server are compatible with the new major version. This may include updating drivers and/or Firebird client software when used by applications!), changes in SQL queries, etc.
- **Rollback plan:** Prepare a plan in case the update fails (or significant uncorrectable problems will be discovered during subsequent Firebird operation). A rollback of minor version updates is usually a simple swap of Firebird for the original version. The rollback of the major version update is much more complicated, because it is necessary to revert all the changes made during the upgrade (drivers, configuration, ODS, changes in applications etc.).
- **Data Backup:** Before any update, it is crucial to create a complete backup of all data on the database server. This includes both the data itself and configuration files.
- **Schedule an update:** Plan the update to minimize the impact on running operations. This often means doing updates outside of business hours or during low load periods.

- **Informing interested parties:** Before the update itself, inform everyone directly affected by the update about the planned maintenance window and the changes being made.
- **Documentation of changes:** After the update is complete, update the documentation to make it clear what changes have been made and how they should affect the operation and development of the system.
- **Final tests:** Even after the update is complete, it's important to perform final tests to make sure everything is working as it should.
- **Subsequent monitoring:** After the update, regularly monitor system behavior and pay attention to any unexpected problems.

Remember that each system may have specific requirements, so it's important to tailor these basics to the specifics of your environment.

Updating Database Server: Minor vs Major Version

Updating a database server to a new minor version (which mostly includes only fixes) and to a new major version (which usually include new features and significant changes) may share some common steps but also have key differences. Here are the fundamental distinctions and common elements:

Common Elements

- **Planning:** In both cases, careful planning is crucial, considering working hours and minimizing the impact on ongoing operations.
- **Backup:** Before both types of updates, creating a comprehensive backup of data and configuration is essential to facilitate system restoration if needed.
- **Testing:** Although the scope of testing may vary, in both cases, it's critical to conduct tests to ensure that the update doesn't introduce new issues.

- **Documentation:** All changes in configuration should be well-documented, including the changes made to any specific procedures (like backup or replications).

Differences

Updating to a New Minor Version (Fixes):

- **Faster Process:** Upgrading to a new minor version is usually faster because it usually doesn't require configuration changes and database upgrades (due to the ODS change), so the whole upgrade usually consists of uninstalling the original version and installing the new one.
- **Lower Risk of Compatibility Issues:** Upgrading to a minor version carries little risk of compatibility issues with existing applications, so pre-testing can be kept to the bare minimum. However, some fixes may affect server behavior, including performance characteristics, and despite extensive testing by the Firebird QA team, it's always possible that regressions or new bugs could be introduced.

Therefore, it is essential not to neglect monitoring the behavior of the server after the update.

In case of problems with the new version, there is usually no problem to return to the original version.

Updating to a New Major Version (New Features):

- **Comprehensive Testing:** Updating to a new major version requires more extensive testing as it may introduce new features and changes that could impact existing functionality. The most significant migration issues and possible solutions are always documented in Release Notes, but significant changes in Firebird may have unforeseen consequences with impact on performance or behavior characteristics. And even reliability in specific, not so common usage scenarios that may escape Firebird QA.

- **Testing of New Features:** For new features, specific testing is necessary to ensure that the new functionality when used do not interfere with existing processes. For example: various timeouts or READ COMMITTED READ CONSISTENCY introduced in Firebird 4.
- **Careful Planning:** When updating to a new major version, more careful planning is necessary as it typically involves database migration (due to the ODS change), and it may bring more significant changes, for example in security setting etc. Also important is a rollback plan in case the update fails. A special case is a recovery plan executed after a certain time after an update, when the updated data needs to be migrated to a database with an older structure.
- **Consideration for New Configurations:** A new major version may introduce new configuration options, necessitating consideration of their impact and potential adjustments to configurations.

In conclusion, regardless of the update type, meticulous planning, testing, and monitoring are crucial to minimize risks associated with updating the database server.

Database Server Testing: Performance and Reliability

Testing the new database server version before actual update in production deployments is a crucial aspect, particularly focusing on performance and reliability. Below are the fundamental considerations for testing migration to a new major version.

Basics

- **Create a Test Environment:** Establish an isolated testing environment that closely simulates production conditions, including real data and operational scenarios.
- **Performance Benchmark Tests:** Conduct benchmark tests on the existing database server version to establish baseline performance metrics such as query speed and server response times.
- **Performance Metrics Comparison:** After server migration in test environment, rerun benchmark performance tests and compare the new results with pre-migration values to ensure performance has not deteriorated.
- **Functional Verification:** Verify that all system / application functions operate correctly after migration to ensure that the new version does not introduce any new regression errors.
- **Stress Tests:** Conduct stress tests on the new version to understand how the system performs under increased load, identifying potential scalability issues.
- **Rollback Tests:** If possible, perform tests for the rollback option to the previous version, preparing for potential migration failures.

From the above, it follows that ideally there should be three types of automated tests for every enterprise-class database application: performance, functional and stress.

Functional tests

If the database contains stored procedures, functions or triggers, it is convenient to create automatic tests for them. The same applies to frequently used SQL queries.

However, not all the work that the given application does with the database is implemented through stored procedures, functions, triggers and clearly defined queries. Sometimes this functionality may not have a clear expression in SQL (e.g. when using ORM, etc.), or it cannot be easily converted into specific SQL commands (e.g. when automatically generating SQL in the application code according to parameters).

Alternatively, it may be so complex and/or spread across individual system modules that the burden associated with maintaining and synchronizing separate tests with the application may be too great. In such cases, it is better to focus on automated functional testing of the application itself. These tests can be used (at least some of them) also as functional tests of the database.

Stress tests

There is no point in doing standard TPC-C and TPC-H type stress tests because new major versions of Firebird are normally tested using OLTP-EMUL. However, if you have automatic functional tests for the application (see above), then load tests can also be implemented based on them.

Performance tests

If you have automated tests for stored procedures, functions, triggers and the most frequently used SQL queries, they can be successfully used as a basis for performance testing.

In the next installments

Automatic tests are a significant helper (not only) when updating to a new major version of Fierbird. If you don't have such tests, they can be created fairly easily using the Firebird QA tools. In the next two parts of this series, we'll show you how to create and run tests using Firebird's QA tools, and how to generate those tests from the database structure and SQL queries executed by the application.

Because Firebird QA tools are written in Python, this issue also contains an introductory article that explains how to use applicatiuons written in Python for those that are not familiar with this powerful and versatile language.

Interview with Jiří Činčura

If you program for Firebird in .NET and have ever visited an international Firebird conference, then you certainly know Jiří. If not, then at least you are guaranteed to use the .NET Provider for Firebird, which he has been improving for you for many years. That's why we asked him a number of questions for you, not only about upcoming news.

First, can you introduce yourself to our readers who don't already know you? So where do you live and what do you do for a living?

Well, my name is Jiří Činčura, I am a blogger[1], speaker, developer and a geek. I live not far from Brno (second larges city in Czech Republic) and make a living as a software engineer (how else, right?).

What brought you to Firebird, and when?

I can't remember exactly when. But as a young boy, I tried to program and master everything, and because Delphi was hot thing at that time, I went to the database trade, SQL, etc. and through InterBase to Firebird. Well, in connection with .NET, I then started working on the ADO.NET provider and there was no turning back.

You have been developing the Firebird .NET provider (currently version 10.0.0.0) and add-on libraries (Entity Framework, etc.) for many years. How did this project come about, and what led you to embark on such a challenging project?

I don't even know how it came about. I wasn't there when driver was created, but I joined after it had been around for maybe 2 or 3 years. I wanted to do something meaningful and not something for a drawer. I was interested in this because it combined .NET and Firebird, two things that I am still interested in today. Later, I took over the role of main maintainer of the project, and then other parts were added, such as the Entity Framework Core provider. Which is actually the last new part of the project.

I know from my own experience that writing the basic driver code is the "easiest". The real work starts with the first users and their comments, problem reports and requests for improvements. Also, keeping the driver in line with new versions of Firebird isn't always a cakewalk. How do you manage it all these years, and how much time does it cost you?

It is as you say. Some basic functionality is pretty straightforward, but then covering all the variations, fitting into the whole ecosystem, etc. is where the sweat and tears are. In the case of the .NET driver (and perhaps all drivers), there is also "fun" in the fact that, on the one hand, users want support for new functions in Firebird, and on the other hand, they also want support for all the

news in the .NET world.

I don't even want to know how much time it costs. But it's also partly fun for me. Anyone who would like to see how much time a person spends on support for Entity Framework Core 7, for example, can watch the entire process[2]. At the same time, it must be said that the difference is also that I work on the project more or less in my free time, so sometimes the biggest obstacle is not the actual implementation, but having a continuous block of time.

Have you ever encountered any major difficulties while working on the .NET provider? What was the biggest challenge for you?

The biggest challenge is keeping it all together. The project already has something behind it, so there needs to be some minimal backward compatibility. Also, unfortunately, some things in Firebird are built on how the protocol is managed in C++ code, but it's not always the same on the .NET side. So figuring out how to make it meet on both sides is sometimes quite a challenge.

What are your future plans for the .NET provider, and what are you planning for 2024?

I have now started preparing support for Entity Framework Core 8[3]. That will probably be the biggest thing for the first half of 2024. Otherwise, of course, support for new features in Firebird 5 (or 4), depending on which one interests me or what the demand will be. At the same time, I would also like to clean the codebase from legacy things, but I am still debating where exactly to draw the line of what will and will not be supported from the point of view of older environments, especially .NET.

In addition to the .NET provider, you are also a co-author of the open-source `FirebirdDbComparer` library and the `FirebirdMonitorTool` tool. Both projects are interesting, mainly due to their potential. Especially the tree-view visualization of events from the trace log provided by `FirebirdMonitorTool` is very useful. However, both projects are slowly becoming obsolete (with new versions of Firebird, .NET, etc.), and judging by the inactivity on GitHub, it seems that they are doomed to gradually rust. Do you intend to return to any of them in

the future?

I would like to actively maintain them, but unfortunately I don't have that much time, and the time I have I devote mainly to the .NET driver. Both projects were created as a response to the need to have such libraries on a project I participated in. So I could devote more time to it. This is no longer the case, so the projects will probably end up in the abyss of history.

You regularly lectured at international Firebird conferences. Unfortunately, the last one was 2019 (in Berlin). Do you miss attending conferences?

I still actively give lectures, especially at developer conferences (since there are few database ones). So my need to expound wisdom somewhere is satisfied. But of course it's pity that there wasn't a Firebird conference for a long time.

Sometimes you publish videos on YouTube. Have you ever considered making something for the Firebird project channel[4] ? Or participate in a "virtual" Firebird conference on this channel?

Actually, this is the first time I've heard about this channel, and no one has contacted me about it. But if there was interest, I certainly wouldn't resist.

How do you perceive Firebird's position in the Czech Republic?

Actually, I don't even know. The local Firebird community is pretty quiet, and the absence of any happenings on a global level can be seen locally. Maybe this act will awake the local community again?

Thanks for you time!

Resources:

1. tabs over spaces
2. Upgrading provider to Entity Framework Core 7
3. Upgrading provider to Entity Framework Core 8
4. Firebird project Youtube channel

Development update: 2024/Q1

A regular overview of new developments and releases in Firebird Project

Releases:

- Firebird 5.0, released 11.1.2024
- Jaybird 5.0.4, released 10.3.2024
- Firebird ODBC driver 3.0.0.8 BETA, released 21.2.2024
- Python firebird-base 1.7.2, released 20.2.2024
- firebird-qa 0.19.2, released 20.2.2024

Road to Firebird 6.0

Firebird v6.0 is now in active development, with final release planned for Q1/2026.

The Roadmap contains a lot of new features and improvements. As the Project don't want to see yet another 5 years development cycle like it happened with FB v3, this list will be further prioritized, and some features may be postponed.

Highlights from the 6.0 Roadmap

There are some features that are either already completed, or in active development right now. So you may expect that they will appear in v6.

The major new features are:

- 1. SQL-compliant JSON functions
- 2. Support for tablespaces
- 3. Support for SQL schemas. In the planning stage and there is active discussion on the firebird-devel list
- 4. SQL-standard compliant ROW data type
- 5. Shared metadata cache

First two features are finished, but not yet merged into Firebird. Hence no details were published yet.

There is great number of minor features and improvements, from which we would like highlight:

- Named arguments for function call, EXECUTE PROCEDURE and procedure record source
- CALL statement. It is similar to EXECUTE PROCEDURE, but allow the caller to get specific output parameters, or none.
- DROP [IF EXISTS] statement
- EXPLAIN statement and RDB$SQL package
- Allow collation to be a part of data type

- SQL standard FORMAT clause for CAST between string types and datetime types
- Job/task scheduler
- Remove/increase the record size limit
- UNLIST function to decode separated values list into a record set
- TRUNCATE statement
- Parallel statistics collection (GSTAT)

What was discussed

Schemas

SQL Standard 2013 is complex and following does not contain things almost not present in any DBMS, like modules, for example. This text also maps some standard concepts to Firebird similar ones, like EXECUTE STATEMENT and DSQL. There are no details yet about permissions.

```
CREATE SCHEMA statement:

<schema definition> ::=
        create schema <schema name clause>
        [ <schema character set or path> ]
        [ <schema element>... ]

<schema character set or path> ::=
        <schema character set specification>
        | <schema path specification>
        | <schema character set specification> <schema path specification>
        | <schema path specification> <schema character set specification>

<schema name clause> ::=
        <schema name>
        | AUTHORIZATION <schema authorization identifier>
        | <schema name> AUTHORIZATION <schema authorization identifier>

<schema authorization identifier> ::=
        <authorization identifier>
```

```
<schema character set specification> ::=
        DEFAULT CHARACTER SET <character set specification>

<schema path specification> ::=
        PATH <schema name list>

<schema element> ::=
        <table definition>
        | <view definition>
        | <domain definition>
        ...
```

`<schema authorization identifier>` defines the owner of the schema and if omitted it's the current user.

If `<schema name>` is omitted it's the same as `<schema authorization identifier>`.

An interesting thing to note is that is possible to embed DDL statements in the `CREATE SCHEMA`, so it should be something like this:

```
create schema app1
        create table table1 (id integer)
        create table table2 (id integer)
```

`DROP SCHEMA` statement:

```
<drop schema statement> ::=
        DROP SCHEMA <schema name> <drop behavior>

<drop behavior> ::=
        CASCADE
        | RESTRICT
```

There is no `ALTER SCHEMA` statement, hence default character set and schema path cannot be changed. However, given that Firebird supports altering the database default character set, it should do the same for schemas.

A session has a default schema (default unqualified schema name) and a default SQL-path.

The session default schema can be read with `CURRENT_SCHEMA` context variable and can be changed with `SET SCHEMA <name>` statement (it should be supported via DPB too). When a routine is invoked it's implementation-defined if its value is changed during the call.

The session default schema is used to search unqualified object names (except in routine invocation) in `DSQL` or `EXECUTE STATEMENT` queries.

Unqualified object names (except in routine invocation and in `EXECUTE STATEMENT`) present in stored routines always searches the same schema.

SQL-path is a list of one or more schema names and it's present in both a schema descriptor as well in a session. The session SQL-path can be read with `CURRENT_PATH` context variable and can be changed with `SET PATH <names>`.

Session SQL-path is used to search unqualified routine invocation in `DSQL` or `EXECUTE STATEMENT` queries.

SQL-path from schema is used to search unqualified routine invocation directly present (no `EXECUTE STATEMENT`) in stored routines.

Discussion about metadata quickly turned into discussion about object ownership.

> *According to the standard, only schema can have owner, objects in schema belongs to the schema, so schema name naturally must appear in currently existing field RDB$OWNER_NAME. Current ownership of objects is naturally transformed into schemas as in Oracle user = schema. With default schema path including <schema authorization identifier> backward compatibility is guaranteed and the only needed step for migration is creation of schema with name of user.*

However, this matter is more complicated as it seems at first glance. Various use case scenarios and angles (like backward compatibility, SQL standard compliance, implementations in other RDBMS, database restore, SQL SECURITY DEFINER/INVOKER etc.) were discussed, but final agreement wasn't reached at the time of this report.

For example, schemas are used for two different functionalities:

- **Modularity:** multiple "logical databases" for different applications, where sometimes one interact with the other in the database
- **Multiple tenants:** identical (in most cases) "logical databases", one for each customer

Hence possible necessity to support schema-only backup & restore by gbak was raised. However, as the second use case is easily solvable with multiple databases, and the first case is not strong argument, the final decision about this matter was postponed.

Only two final decisions were made so far:

- that GPRE will not be updated to support multiple schemas.
- that Firebird implementation will deviate from the SQL standard that allow a trigger to be created in a different schema than its table. In Firebird, table triggers (and indexes) will always belong to the same schema as the table.

Jim Starkey is still following the Firebird development list and has contributed to this discussion...

> *The only real issue here is standard compliance. And while I have long felt that a multi-level name space is important, I think the SQL standard is asinine and it's approach to access control moronic. It's hard to think of a redeeming value other than being the official standard.*
>
> *The utter joy of Amorphous is that I can invent stuff that actually addresses problems without kowtow to a committee mostly populated by people their colleagues wanted out of the office. But hey, that's just my opinion. [A former secretary of the SQL committee used to work for me. Yeah, I'd have sent him to the committee if it kept him from writing code…]*
>
> *A couple of lifetimes back I was DEC's representative to the CODASYL End User Facility Committee. We got together four times a year to spend a week arguing about the different between an electronic staple and an electronic paper clip (spoiler: the former is resource constrained and nested and the latter is neither).*

> *Ladies and gentlemen: Follow the standard if you feel you must, but if you require that users change existing applications, they just might decide to change them to use a different database system.*
>
> Jim Starkey

Deprecation

Ending support for various features or tools was discussed and several decisions were made:

- `Multifile databases` should be deprecated in v6.0, and support removed in v7.0
- `gsec` will be removed (both tool and service) in v6.0
- `UDF support` will be kept for v6.0. The overhead of UDR calls will be reduced (already done in development tree), and migration to UDR will be encouraged.
- The `SQL Dialect 1` will be retained for v6.0, developers will revisit and address (or declare as won't fix) the remaining issues in v6, and encourage migration to Dialect 3. Then probably Dialect 1 will be removed in v7.

Triggers and standard compliance

During research on statement-level triggers, Adriano discovered that current Firebird triggers are probably not compliant to SQL 2013 Standard.

Firebird executes them in the following order:

```
- before row 1
- after row 1
- before row 2
- after row 2
- before row 3
- after row 3
```

While PostgreSQL:

```
- before row 1
- before row 2
- before row 3
- after row 1
- after row 2
- after row 3
```

The standard says:

> *For each state change SCj in SEC, the BEFORE triggers activated by SCj are executed before any of their triggering events take effect. When those triggering events have taken effect, any AFTER triggers activated by the state changes of SEC are executed.*

The conclusion was that PostgreSQL is correct and Firebird is not. However, no decision was made to address this "issue".

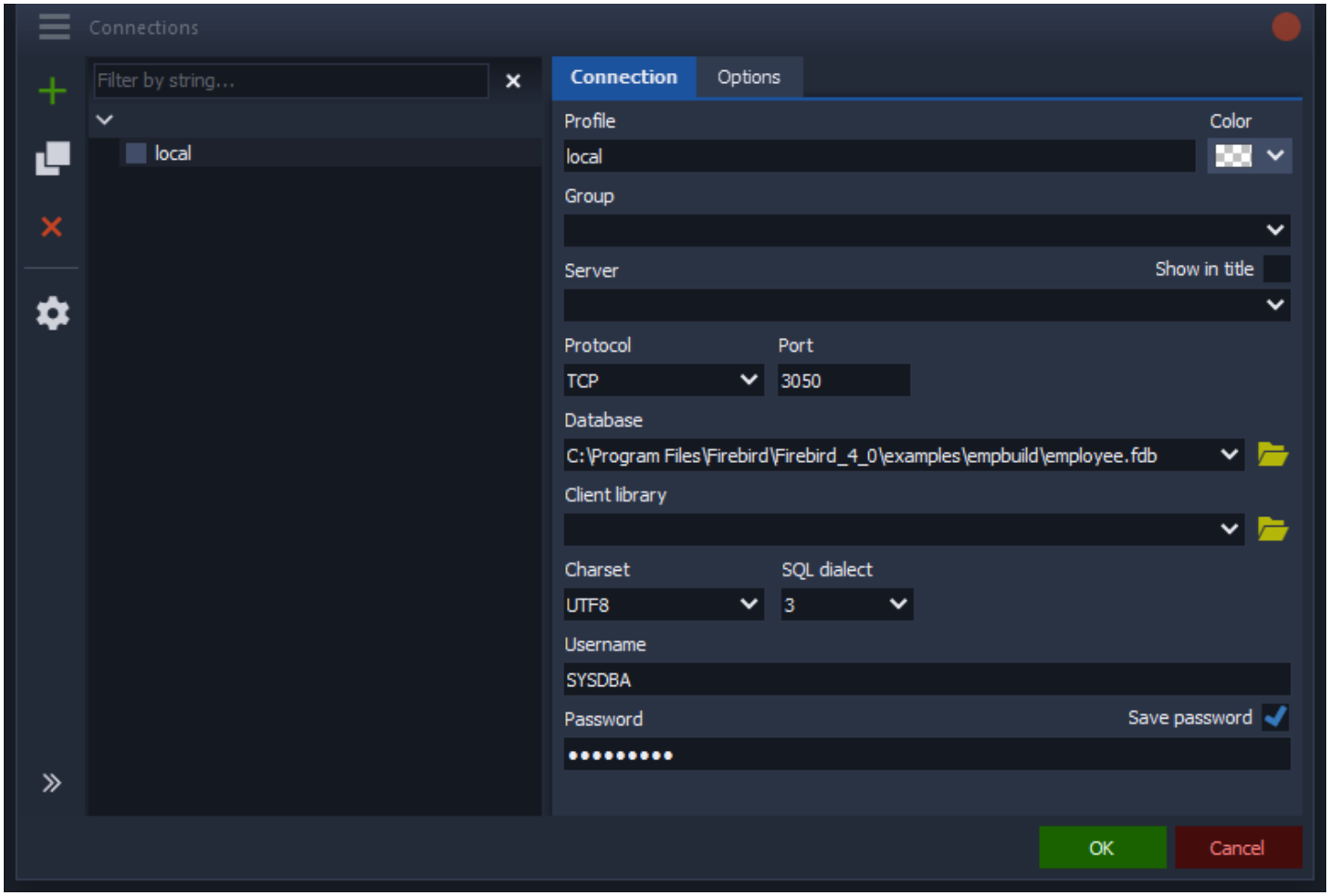
Toolbox: Firebird Editor Pro

Firebird Editor Pro[1] is a simple, elegant **freeware** tool for working with Firebird databases on the Windows platform. It is built in Delphi (RAD Studio) and uses DevArt's IBDAC components to work with Firebird. Is available in both 32-bit and 64-bit versions. In addition to `.exe` and `.msi`, a `.zip` distribution is also available for those who want to carry it with them, for example on a USB key. The entire installation has less than 25MB.

As the name and the size of the distribution suggest, Firebird Editor Pro basically does not offer much more than working with SQL and data stored in the database. However, it provides this functionality stylishly, with a certain amount of thoughtful elegance, and adds a few cherries on top. Its biggest drawback is the absence of any documentation or help, but this is compensated by the simplicity and solid intuitiveness of the controls.

At first glance, the visual style implemented using AlphaSkins[2] components will catch your eye. Firebird Editor Pro offers a total of three light and three dark skins. But you can choose another one in the AlphaSkins gallery[3], and after downloading, copy the `.asz` file to the `Skins` subdirectory of the editor installation so that it appears in the configuration menu.

When starting the editor, you must first select or define the connection you want to work with. Although the list of defined connections is not organized hierarchically, e.g. by server, as with other tools, there is the option of dynamic filtering by name for easier work with a larger number of defined connections.



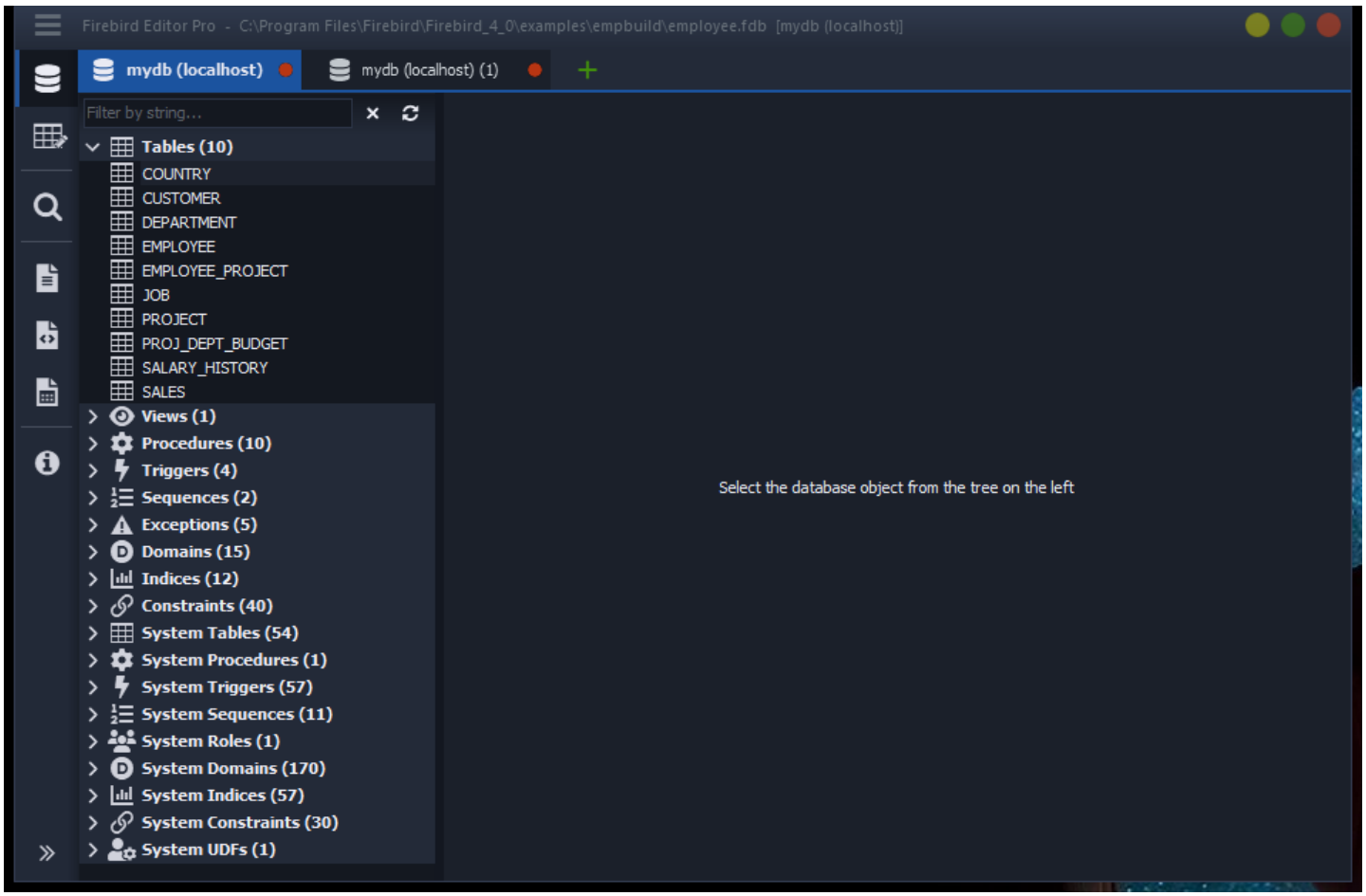
Connection dialog

After creating the first connection and displaying the main window, you will notice the first feature of the editor, which is the way you work with multiple connections at the same time. Individual connections are displayed as tabs in the upper part of the window, and the content of the main window and the status of

the function bar on the left side are tied to the current tab. This arrangement is very practical.

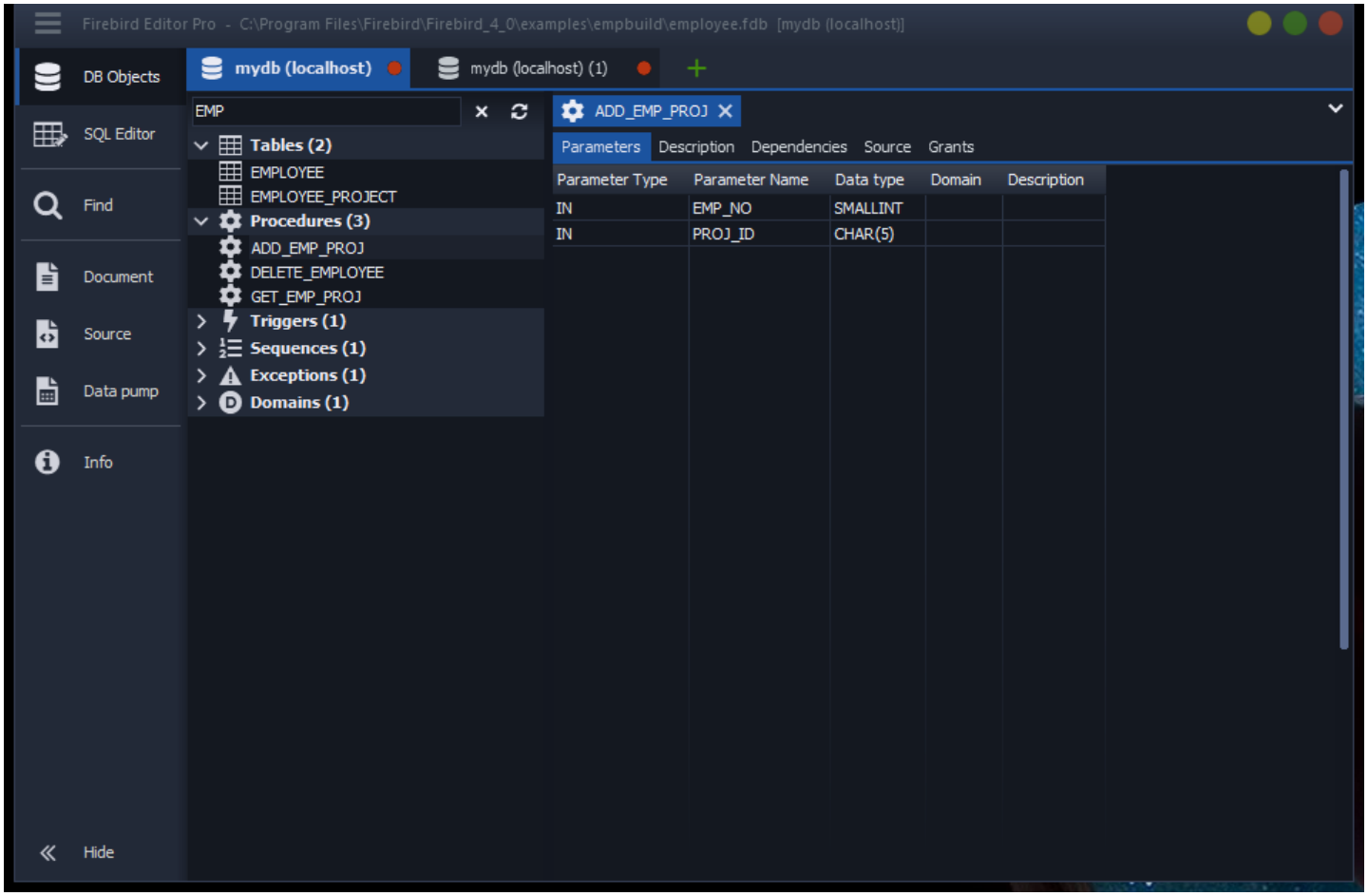
Another nice feature of the editor is the ability to remember active connections when the program is closed, and to open them again when it is started. If you select the Connect prompt option in the Connection Options, you will then be prompted to enter a password at start-up with the option to change the username as well.

The main connection window offers a traditional overview of objects in the database divided into categories. Dynamic filtering by name will help you look for objects even in complex databases.



Main window with list of database objects

After selecting an object, its description is displayed in a structured form according to its type, as is usual with other tools. In this way, multiple objects can be opened, which are then organized into another series of horizontal tabs.

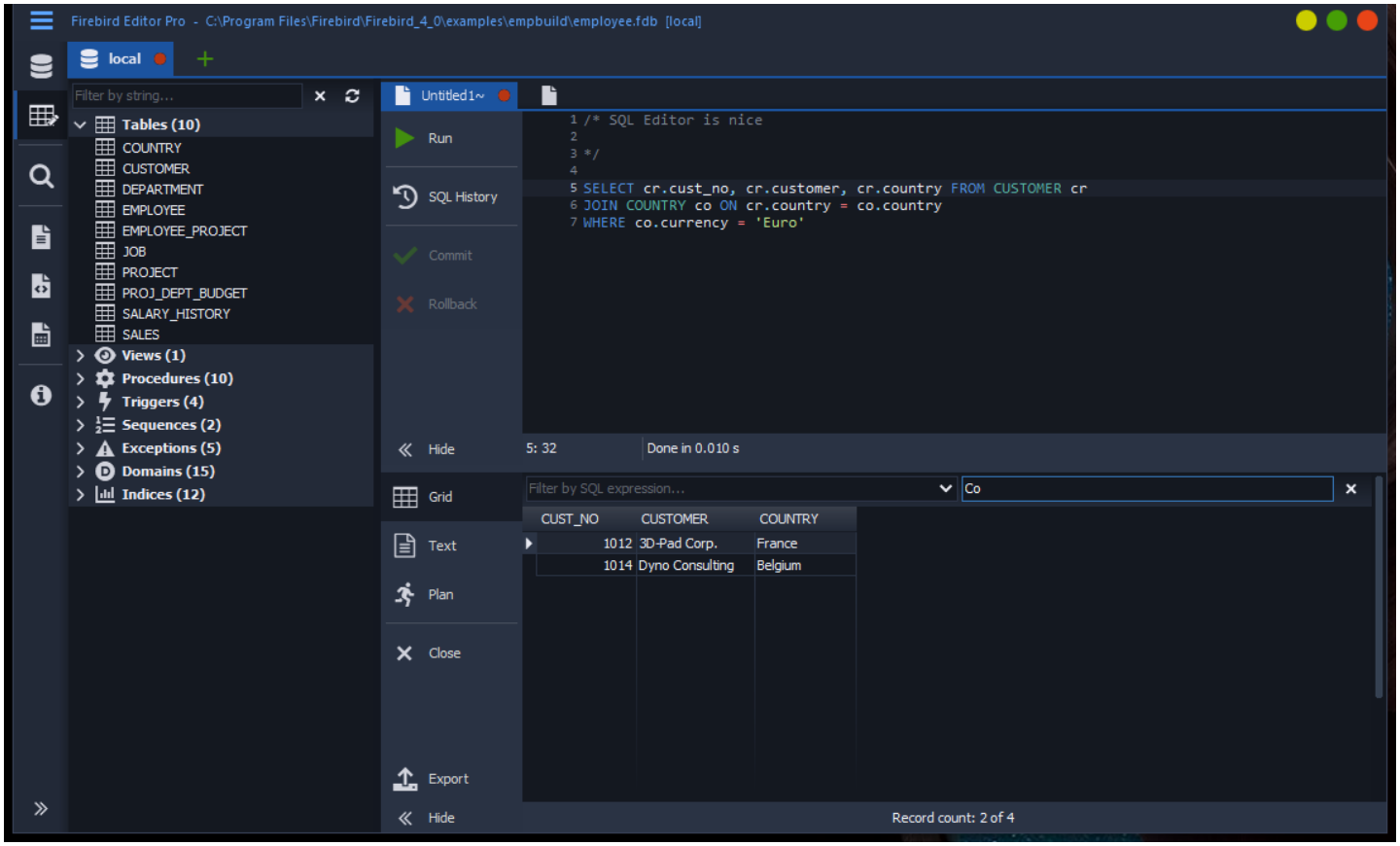


Dynamic filter in object list and object details

The icon bar on the left offers the basic functions of the program. We have already demonstrated the list of database objects. Another feature is the SQL editor. It is built on the open source *TTextEditor* component from the same author, and you can find it on GitHub[4] along with several other components. The editor offers syntax highlighting, folding, command completion, history and other common features. The same component is also used in **Text Editor Pro**[5], which is another freeware product from the same author.

In general, the SQL editor is very pleasant to work with. The only unpleasant surprise is hidden by the command history function, when double-clicking on a selected command (or pressing *Enter*) will add it to the editor without replacing the original content. This can be achieved by pressing *Ctrl+Enter*.

The context menu then contains a whole range of functions, including formatting the command or displaying the execution plan (in extended form) without running it.



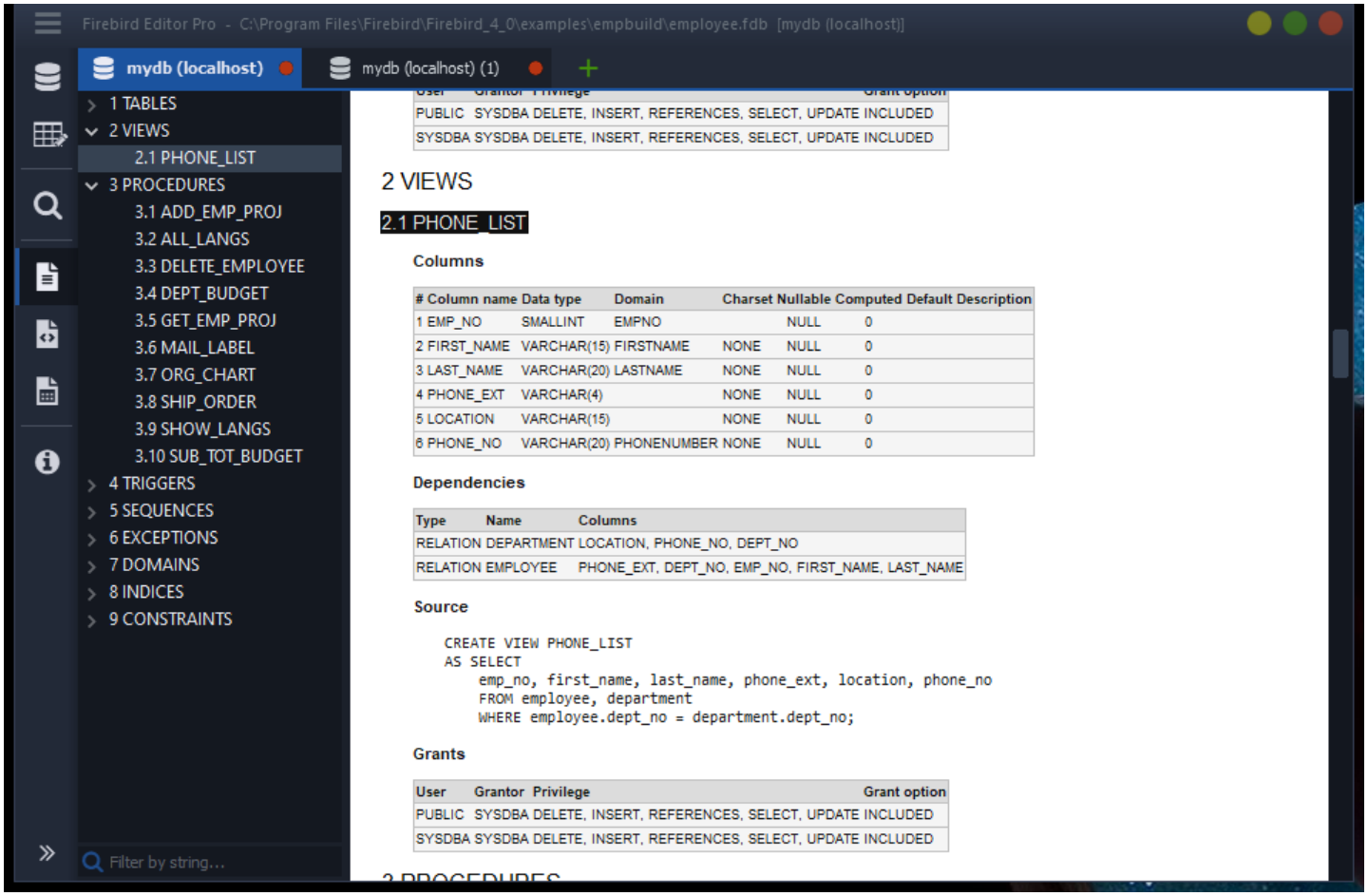
SQL Editor

After running the command, the returned data is displayed in the form of a grid. A text version is also available as a nicely formatted ASCII table for inclusion in documents, e.g. in Markdown format. Data can also be exported in a number of formats including CSV, HTML or SQL script.

The only drawback is the absence of detailed information about the execution of the SQL command. Only information about the total execution time is displayed.

Next function of the editor is a global search in database objects by their name. Basically, it is the same as the already described dynamic filtering in the list of objects, only in a different packaging.

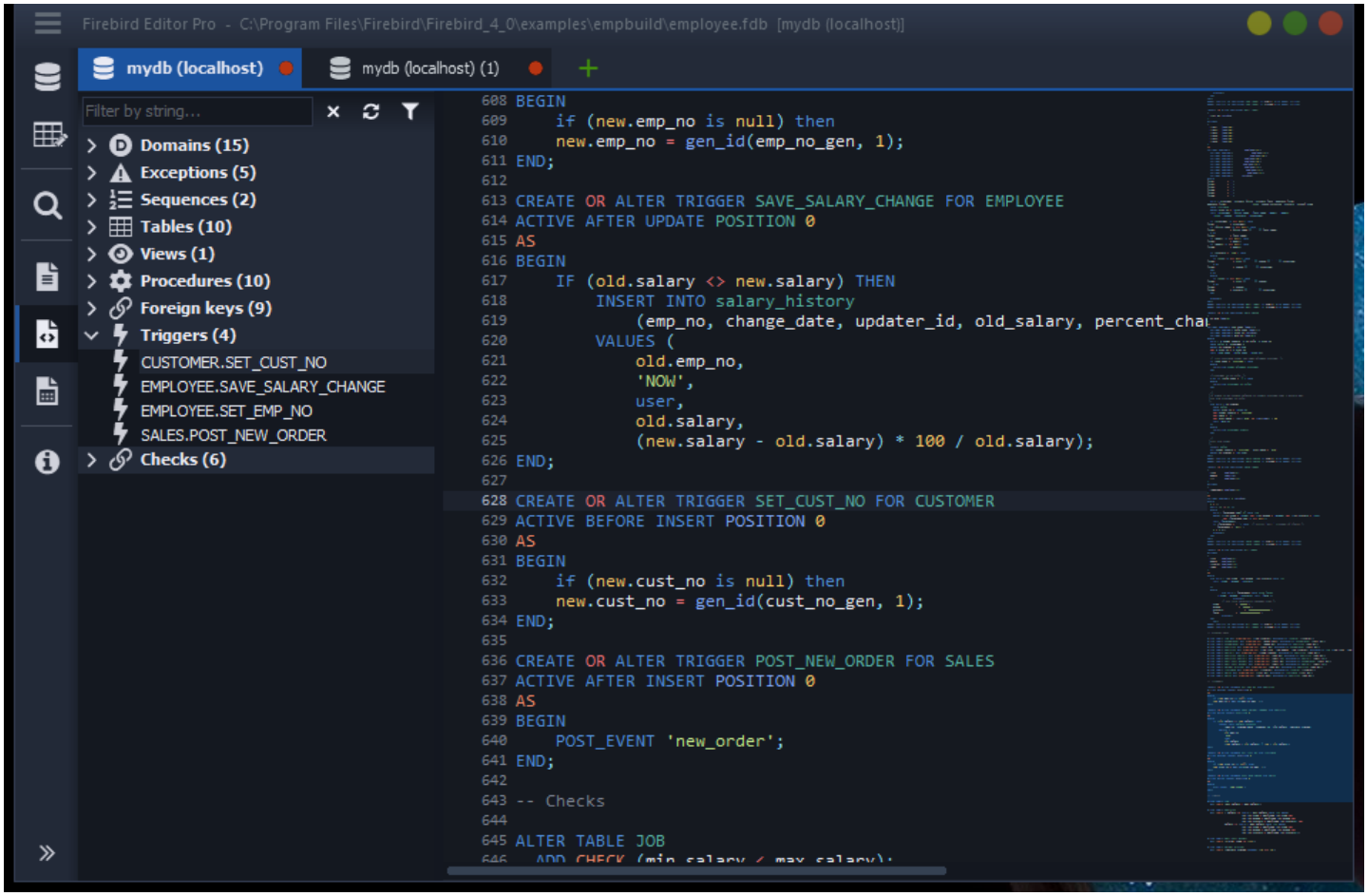
Much more interesting is possibility to generate documentation for the database, which can then be saved in HTML or PDF format. Basically, it is a nicely presented and structured overview of metadata, while the editor also offers quick navigation through this document.



Database documentation

Unfortunately, what should be the content of the created document can only be set in the configuration of the editor, at the level of individual categories. It is therefore not possible to create documentation, for example, only for selected procedures.

Next function in a row is the generation of a script for creating metadata, and this time you have the flexibility to precisely specify the objects you intend to process. The editor facilitates seamless navigation and efficient search functionality, ensuring ease of use even with extensive scripts.

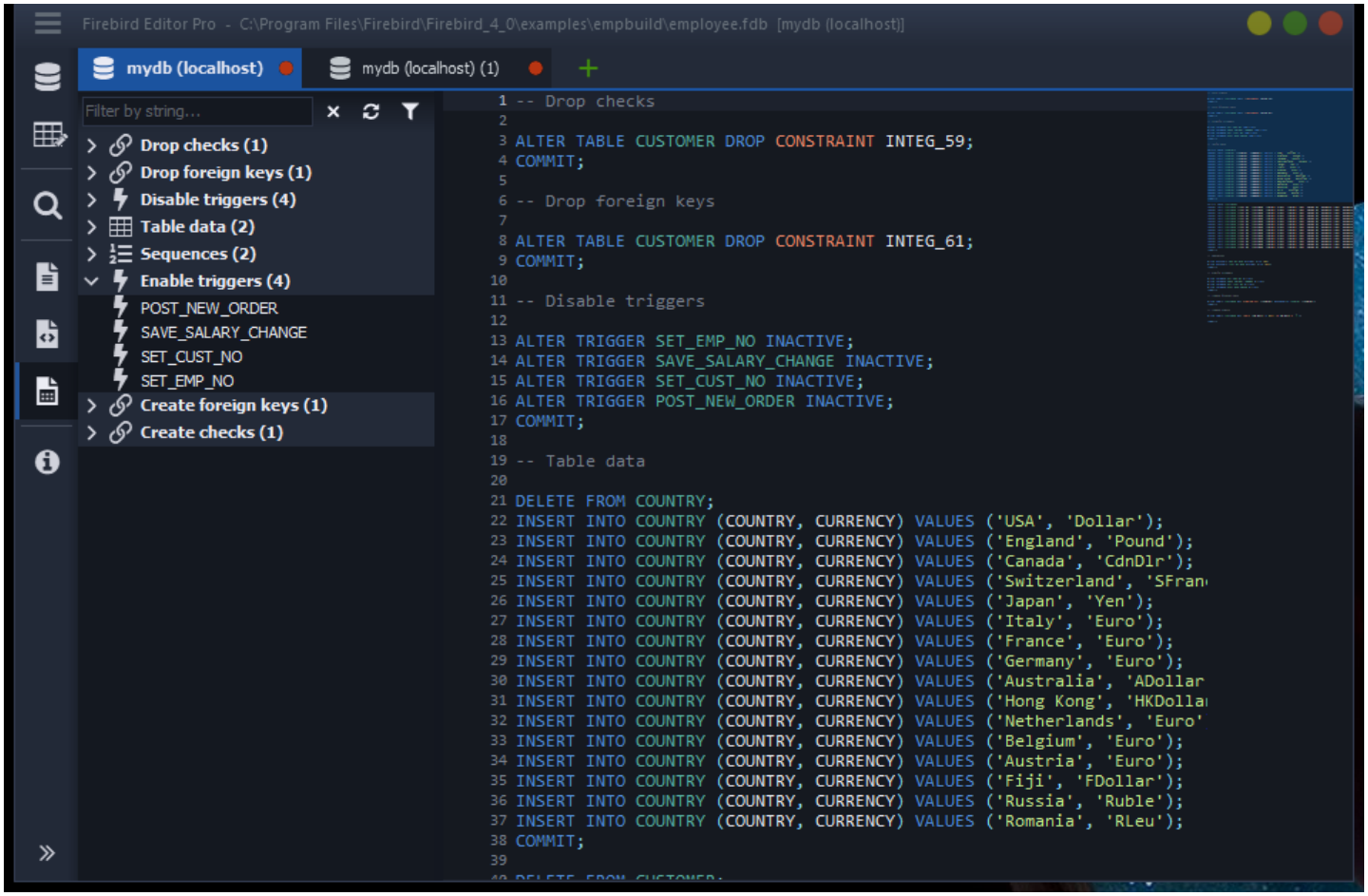


Database source SQL script

The last major function of the editor is the data pump, which generates an SQL script for inserting data from selected tables into another database with the same structure. Optionally, the script may contain commands for turning off referential integrity, checks and triggers for the duration of data insertion.

Summary

Firebird Editor Pro is undoubtedly a quality tool that is pleasant to work with. Although, due to its narrow focus, it cannot be a replacement for any of the big products for working with Firebird, such as SQL Manager or Database Workbench, it will certainly find its users. Thanks to its small footprint, it will be especially appreciated by technicians who have to perform small interventions and modifications in Firebird databases outside their office.



SQL Insert script generator

Also, if you are a vendor of applications built on top of Firebird, then it is definitely worth considering distributing Firebird Editor Pro with your application as an alternative for standard ISQL. Your users and service technicians will surely appreciate it.

Advantages: Undemanding, good SQL editor, elegant design, freeware

Disadvantages: Narrow focus on working with SQL and metadata, absence of documentation or help, for Windows only

Our Rating: 7/10

Answers to your questions

Documentation is said to be a collection of answers to unspoken questions. If you ask a search engine, it will answer you with a link to a document that (hopefully) contains the answer. There are documents, forums and entire systems like Stack Overflow that consisting only of questions and answers. And now an army of AIs is starting to chase us to answer our questions. Questions and answers cannot be avoided, there is no hiding place.

However, amidst the sea of routine questions and responses, there lie truly captivating inquiries and answers, like hidden treasures. Our commitment is to regularly present you with a curated collection of these precious gems.

Does insertion benefit from ordered or sequential primary keys?

Jeff asked:

Is there any advantage to inserting rows with ordered primary key values, or does it not matter?

I intend to use High/Low method for primary keys. With this approach, it is very possible that PKs will not be inserted in sequential order. I'm not sure if this will have an adverse effect on the speed of inserting rows.

Ann W. Harrison answers:

Yes, there are advantages to inserting keys into an index in ascending (for ascending indexes) or descending (for descending indexes) order. The most efficient way to generate primary keys is with a generator - also called a "sequence" in newer versions.

Firebird is designed to handle random inserts into indexes with reasonable efficiency, but building indexes sequentially creates a dense index at low insert cost. When creating a new index on an existing table, Firebird reads the table, sorting by they index key and builds the index from the sorted output.

Why is that fast? Two reasons, neither of which as to do with the (relatively) low cost of moving index entries in memory.

The first is that random index entries tend to land on random pages in the index. Unless you can hold the entire index (and everything else you need) in memory, that means that pages will be written and read multiple times.

> *One of the key misunderstandings in databases is that an index is a good alternative to a sort because the cost of an index look up is K*n, while the cost of a sort is L*nlog(n). It's rarely noticed that the value of K (the cost of a page read) is huge compared with log(n).*

Filling an index page with ordered values and going on to the next page is more efficient that putting one entry on one page, the next on another, and so on, even if all the pages are in memory.

The second reason to store rows in the order of their index (i.e. not just sorted, but sorted ascending for ascending, descending for descending) is prefix compression. Firebird drop the first bytes of a key if they match the previous key value. So if you store `AAAAAA`, `AAAAAAB`, `AAAAAAC`, `AAAAABA`, what goes into the index is `AAAAAA`, `6B`, `6C`, `4BA`. Prefix compression makes the indexes dense, meaning that you read fewer index pages to get to the data you want. Clearly if you store those rows in reverse order, the first key in is `AAAAABA`. When you store `AAAAAAC`, the index contains `AAAAAAC` and the second key value becomes 5BA. Each subsequent entry requires a change to the next older entry.

A comprehensive explanation of the maximum key size limit

---

mpp12310 asked:

When I try to create a table with the Primary Key field being UTF8 VARCHAR of length 255, I get an error saying that the field size is too large. I checked my database page size and it is 4096 (default).

Ann W. Harrison answers:

The maximum length of index keys in bytes is equal to 1/4 of the page size minus nine bytes, i.e. varying from 1006 to 4087. However, some character sets use more than one byte per character, so for UTF8 and 4K page the limit is 253 characters.

Here's a longer explanation of the maximum key size.

The relationship of key size to page size is important because a key that's larger than 1/3 of the page size will cause the index to degenerate from a tree to a list - every new entry causes splits all the way up the tree. Not good. With bad luck, a single insert can turn into an infinite split. Very bad for performance and definitely something to avoid.

Character keys are often bigger than they appear to be, first because of multi-byte character sets and then because of collations. Only the simplest collations generate keys that are the same size as the original data [1].

Several decades ago, the InterBase group decided that the maximum allowable key size would be set at definition time based on the ratio of the largest possibly representation of a key to the page size. In fact, index keys are compressed in two ways: trailing spaces in strings and trailing zeros in numbers are eliminated before concatenation in the case of compound keys, and prefix compression on the whole key which eliminates any bytes at the start of the key which duplicate the previous key. So, generally, keys are smaller than their maximum size. That's even more true now than it was in 1985 when almost all data was ASCII, using a binary collation. The most heavily used characters in UTF8 are one or two bytes, not three or four. Most characters do sort simply on their base identity.

On the other hand, the limits enforced are pretty minimal and an index where every key was actually at the limit would be very deep and inefficient. So the fact that `gstat` shows that you're getting 20 entries per page when the key calculation said you'd get five is really a good thing.

Returning to the question at hand, its not the character length that determines the size of an index key, exactly, but the maximum number of bytes required to represent any character in the character set plus the maximum number of bytes required to represent the levels in the collation, with the sum of those two multiplied by the number of characters.

Using tools written in Python

When dealing with Firebird, Python proves to be a reliable companion that helps not only in the automation of various tasks, but also in the analysis of data from various sources and formats. In addition, there are a plethora of practical tools and applications in Python, which will make the language appear frequently in the pages of this magazine.

Despite Python's popularity, many users still lack experience with the language and its environment. To address this, we've included a brief introduction to running applications and scripts in Python in this first issue. This information will be particularly helpful in the next edition, where you will find an article on creating and running tests for your Firebird databases and applications using Python-based tools commonly used in Firebird's development.

First, you'll need to know a few basic concepts from the world of Python.

Terminal and shell

When working with Python, you cannot completely avoid working in a command line environment. It consists of a *shell* (command processor) and a *terminal* (display), and different shells can be run in different terminals.

While this is not new information for Linux users, it may be different for Windows users. Before the advent of `PowerShell`, the only command line was the `CMD` shell in the standard system console (terminal).

While the shell you use doesn't really matter (use what you're used to), we recommend using a terminal with good ANSI escape sequence support for the best Python experience. On Windows, we definitely recommend using `Windows Terminal` from Microsoft[1].

Python versions

The first thing that is a source of constant problems for new users is the fact that both the Python language itself and its core library are undergoing very dynamic development, and a new version is available practically every year. Individual applications and libraries then require a certain minimum version of the language for their operation. For example, the official driver for Firebird requires Python version 3.8 or later.

If you are new to Python, we recommend installing the latest stable version (currently version `3.12`). If for some reason you already have Python installed, make sure you have at least version `3.8`.

Another important piece of information for you is the fact that **you can easily have multiple versions of Python installed**. We will return to this later.

Python distribution

If you're working on some variant of Linux, you'll find Python in your distribution's repository. Typically, multiple versions are available (see above). The official repository is the recommended source for Python on Linux, but if you can't find the version you want (or higher), try searching the Internet and community repositories. If you fail, you can use the official distribution from the **Python home site**[2]. This distribution is also the recommended source for installing Python on other platforms (Windows, MacOS, etc.).

Distribution of a module or application written in Python

Although the Python standard library is quite comprehensive, you will definitely need to install additional modules, not to mention the need to install the applications themselves. Python code is distributed in the form of source (they have the extension `.tar.gz`) or binary packages (they are called `wheel` and have the extension `.whl`). Typically, both formats are available.

If the given module/application does not contain parts written in other languages, it basically does not matter what type of package you use during installation. Otherwise, when installing from the source package, the compilation of these non-Python parts is also performed, which may require additional elements (compilers, libraries in the given language, etc.). Binary distributions already include these elements compiled for a specific platform, processor, etc., and you can get by with just a Python environment. To begin with, you don't need to worry about this difference, just remember that it exists and why.

An important part of the distribution is the metadata, which, among other things, defines the minimum version of Python, any extension modules needed for running, etc.

Repository of Python modules and applications

The ecosystem of the Python language is huge, and therefore you cannot do without some repository of extension modules and standalone applications. The primary repository is the Python Package Index, or `PyPI`[3]. Although it is not the only existing repository, it is definitely enough to start with.

The pip installer

`pip`[4] is a command-line tool that facilitates the installation, upgrade, and management of Python packages or libraries. Starting from Python 3.4, pip is included by default when you install Python, so you usually don't need to install it separately.

Here are some common pip commands:

Installing a Package:

```
pip install package_name
```

Installing from Requirements File:

```
pip install -r requirements.txt
```

Upgrading a Package:

```
pip install --upgrade package_name
```

Uninstalling a Package:

```
pip uninstall package_name
```

Listing Installed Packages:

```
pip list
```

By default, `pip` fetches packages from the Python Package Index (PyPI), but can also install from other sources such as a local directory, wheel, git url, etc.

Python virtual environments

Python virtual environments are the second biggest source of trouble for newbies. Basically, it's an isolated space where you can install specific versions of Python packages without affecting the global Python installation on your system. You can have any number of such environments, and if you have multiple versions of Python installed on your system, each virtual environment can use a different version of Python.

The virtual environment must first be created, creating the necessary directory structure with the necessary files, including scripts for various shells used to activate and deactivate it in the command line environment. Activation mainly makes changes to the file path settings (and deactivation resets everything), so calling `python`, `pip`, etc. from the command line will use the programs from the relevant virtual environment, and use the additional modules installed there.

So far everything is relatively easy. The problem is that there is no single way to create and manage such a virtual environment. Since Python 3.5, the `venv` module from the standard library is the recommended tool, but there are a whole host of tools that create and manage virtual environments. To begin with, it will be enough for you to familiarize yourself with the two that cover the main two scenarios of using virtual environments.

The first one is the `venv`, which you can use to create a virtual environment for running our examples and for your own experiments. When you will no longer need such a virtual environment, it is enough to simply delete its directory structure.

The second one is the `pipx` tool. In principle, it is a variant of the `pip` installer that installs applications into their own virtual environments. We recommend using it to install applications written in Python that you want to use regularly.

The venv module

The `venv`[5] module is used to create virtual environments, and in addition to being used for this purpose programmatically from applications, it also offers a simple interface from the command line, which you activate with the command:

```
python -m venv
```

If you add the `-h` parameter, the usual help is displayed.

To create a virtual environment, use the command:

```
python -m venv myenv
```

The second argument is the location to create the virtual environment. The above command will create a virtual environment in the `myenv` folder in the current directory.

If you have multiple versions of Python installed, a virtual environment is created for the version that is set as the default. If you want to bind the environment to a different installed version, you need to run the appropriate version of Python interpreter (as we will show later).

To activate it:

- On Windows run:

```
cmd.exe    myenv\Scripts\activate.bat
PowerShell myenv\Scripts\Activate.ps1
```

- On Linux use:

```
bash/zsh   source myenv/bin/activate
fish       source myvenv/bin/activate.fish
csh/tcsh   source myvenv/bin/activate.csh
PowerShell myvenv/bin/Activate.ps1
```

If you want to later reactivate an existing virtual environment, follow the same instructions about activating a virtual environment. There's no need to create a new virtual environment.

> *Once activated, the shell prompt should change to also display the name of the virtual environment.*

If you want to switch projects or leave your virtual environment, deactivate the environment with:

```
deactivate
```

The pipx tool

If you install the applications you want to use using pip directly into your Python installation, you risk problems with possible conflicts in the dependencies of individual applications. This is because you cannot install multiple versions of the same library in a single Python environment, and if different applications work with the same extension library but require different versions of it, you have a serious problem. Likewise, you may run into problems when updating your application's dependencies, where other applications using the same modules may not be ready to work with the newer version yet and may start to malfunction.

However, you need to install `pipx`[6] first, as it is not a standard part of the Python installation. On Linux, you can typically find it in one of your distribution's repositories. If you can't find it there or you're working on a different platform, you need to install it using:

```
pip install pipx
```

or

```
python -m pip install pipx
```

One of the nice features of using `pipx` is that the applications it installs can be directly called without having to activate their virtual environment first. The executable files of the application are stored in one directory, which must be defined in the PATH. Therefore, the first thing to do after installing `pipx` is to call:

```
pipx ensurepath
```

Now you can globally install an application by running:

```
pipx install PACKAGE
```

This automatically creates a virtual environment, installs the package, and adds the package's associated applications (entry points) to a location on your `PATH`.

By default, `pipx` uses the same package index as pip, PyPI. `pipx` can also install from all other sources pip can, such as a local directory, wheel, git url, etc.

If you have multiple versions of Python installed, the virtual environment is created for the version that is set as the default. If you want to use the a different installed version for application, you need to use `--python` switch, for example:

```
pipx install --python 3.9 PACKAGE
```

will install application with Python 3.9.

To list all `pipx` commands, use:

```
pipx --help
```

For command-specific help then:

```
pipx command --help
```

Working with multiple Python versions

If you have several versions of the Python language installed, one of them acts as the default. You can find the version of the default environment with the command:

```
python --version
```

If you are working on Linux, the individual versions are typically available under their version name, e.g. the command:

```
python3.9 --version
```

will run Python version 3.9 interpreter if installed.

On Windows, a launcher called `py` is available to run a specific version of Python. Although this is an optional part of the Python installation, it is installed unless you choose otherwise during installation.

When you type `py` at the command line, the launcher invokes the current default Python interpreter.

To see which versions of Python are available to `py`, type:

```
py -0p.
```

You'll be presented with a list of all the known interpreters in the system, their version numbers, and their full paths. The interpreter marked with an asterisk is the default.

To invoke a specific edition of Python, type `py` followed by the switch in the left-hand column for the appropriate version. For instance, to launch the 64-bit edition of Python 3.9, you would type:

```
py -3.9-64.
```

Note that if you provide only a version number, and not a bitness indicator, you'll default to whichever version of Python is native to your machine's processor type. On a 64-bit machine, that would be the 64-bit edition. So if you just typed `py -3.9`, you'd get the 64-bit version of Python 3.9.

If you don't specify a bitness, and only one bitness of a particular version is installed, that bitness will be loaded by default.

To run a Python script with the `py` launcher, simply substitute `py` and its command-line switches for `python`. For example to install package into Python 3.9, use:

```
py -3.9 -m pip install PACKAGE
```

Conclusion

To experiment with the demo scripts featured in this magazine, ensure you have the following:

Python Installation: Python version 3.8 or higher is required, unless specified otherwise. It's crucial to match the architecture of Python (64-bit or 32-bit) with the Firebird version you are using.

Virtual Environment Setup: Create and activate a virtual environment to keep dependencies isolated.

Simple script examples: If the example is provided as a script with a `.py` extension and requires additional libraries, check for specifications in the `requirements.txt` file. Install these libraries into the environment using:

```
pip install -r requirements.txt
```

You can then run the example using:

```
python SCRIPT-FILE
```

Examples as distribution packages: If the example is distributed as a package (typically with a `.whl` extension), install it using:

```
pip install PACKAGE-FILENAME
```

Such examples usually export one or more entry points, accessible from the command line in the activated virtual environment. Details on their availability and usage are provided in the article.

If you find an example particularly useful and wish to use it frequently, consider installing it using `pipx`.

Resources:

1. Microsoft Windows Terminal 2. Python downloads

3. Python Package Index 4. pip documentation

5. venv documentation 6. pipx documentation

...And now for something completely different

Folks, as you know, we should have *Jeff Dunham* **and his *Achmed the Dead Terrorist*** here tonight, but he didn't make it in time due to premature detonation, so, instead, we are proud to introduce the ***Averege IT Joe*** to you. Give him a warm welcome!

..Joe is taking the mic during applause...

Hi folks! How are you tonight? Good, yeah? Well, me not so much. I have to share with you a truly horror story that happened to me recently...

You see, I was invited to an interview at an IT company where I was applying for a job. At first everything went well! So I was sitting in the HR office with this lovely young HR person and she was really impressed with my resume. I can handle C# for the frontend flawlessly, I have a knack for web technologies (you name it) and Python to glue things together and extend things here and there. And then she asked, "I see you've worked with Oracle, MSSQL Server and Postgresql. Can you handle Firebird? We use it for almost everything..."

That was quite a shock! "Firebird what?" I said.

“That's a database from the guy who invented BLOBs!” she explained.

"Blobs you say?" That got me back in the saddle! "Yeah, I think I remember him. I saw him on the news thought. He's one of the most wanted terrorists, right?"

Now SHE looked a little shocked, so I explained further, "One of his blobs ate Cincinnatti, didn't he? In Cleveland? I think I read that or saw it on TV."

She kept looking at me strangely, so I added, "And Elvis is missing too!"

Then suddenly she started laughing, and when she stopped she said, "That was really good! You'll definitely fit in!".

So I got the job. But that was the first clue that something really strange was going on with this company.

Then, I first met with my new colleges. And the first thing my future boss asked me was "Tell us a joke!". So I told them about my little chat with the HR lady, and they liked it so much, that one of them literally wet himself... with spilled coffee.

Then my boss said, "A good one for good one! Here is one from our jokes..."

So MySQL, PostgreSQL and Oracle walk into the bar. And the moment they finish their first drink, they start bragging about their strengths.

MySQL says "I'm so fast; I process transactions like a ninja!"

PostgreSQL adds, "Well, I have all the advanced features and extensibility. I'm a wise choice for complex tasks!"

Oracle joins in: "I've been around forever doing things at the enterprise level. I practically invented databases!"

Suddenly they notice the Firebird sitting quietly in the corner sipping a drink. "Hey Firebird, what's your claim to fame?"

And Firebird smiles and says, "Well, I'm not here to brag, I'm here to quietly save the day."

The others exchange confused looks, and then MySQL whispers to PostgreSQL and Oracle, "I think we should switch bars. The drinks here have an odd aftertaste."

And they started laughing perhaps even more than before!

That was probably the second clue...

The boss noticed that I wasn't laughing as much as they were and asked, "Didn't you like it?"

And I said "Well... to be honest... I don't think I quite got it."

And then he and his Scooby-gang started laughing again! I really don't know what would have happened if there was more coffee around!

But otherwise they are nice people...

And suddenly, one from my colleagues said, "Don't worry, you will eventually get it. And when that happens, it will be legen ... wait for it .. dary!"

That moment I figured they binge watch HIMYM too much. But I'm ok with that. At least until they try to make me watch all nine bloody seasons with them...

Don't laugh, I used to work at a company where the top boss loved star trek... So every meeting ended with some damn star trek quote, and you certainly couldn't ask for a raise without a proper star trek reference thrown in. So everyone in the company had to know star trek intimately...

Look, don't laugh! That's terrible! Do you know how many bloody star trek movies and tv shows are out there? I'm not Sheldon, I had to leave the company...

But otherwise they were also nice people...

But back to this new company because the real horror story is yet to unfold...

So after some time and when I had settled in properly, I was sent alone to one of the company's customers to upgrade the CMS system we had sold them. And everything was fine in the beginning, I upgraded our software, changed the configuration, and then I needed to make sure that the database server was configured correctly. So I went to their office manager and asked where they had the Firebird.

He gave me a blank look and said, "Firebird what?"

“Well, that software gadget that manages all your bloody data!” I explained.

"Oh that! Ask our database administrator..."

"And where will I find him?" I asked.

"The last time I saw him, he had an office in the first basement, next to the servers..." he replied.

So I ventured into the first basement, and found their Firebird admin. He was sitting right next to a softly humming server in an otherwise quiet office.

Too quiet...

Turns out the poor guy died years ago and no one noticed because they had no damn reason to call or visit him!

Yes! Just as you said, sir... Blood and bloody ashes!

When I saw this, I rolled my eyes and said, "Why, why my Lord! All I asked for was quiet, easy database work, and you gave me this? What have I done to you?"

And then suddenly a very old, thick, dusty Oracle manual fell on my head from the top shelf...

...and that was the third clue.

So I dusted off the poor skeleton, quietly closed the door from the outside, quietly checked out and went home. Because I finally got it, and in a truly legendary way...

Because if they haven't noticed yet, what are the chances they will before I retire, right?

Good evening people!

...a standing ovation as Joe leaves the stage...

Elevate Your Database Performance with HQbird

Are you managing large-scale Firebird databases or overseeing a multitude of servers? HQbird is your premier solution for optimizing performance, ensuring seamless maintenance, and reducing support costs.

Maximize Efficiency for Big Data

With HQbird, big Firebird databases are no longer a big problem. Our advanced optimization techniques ensure that your large datasets perform at their peak, delivering faster query responses and streamlined data processing.

Maintenance Made Easy

High-load systems require meticulous care, and HQbird delivers just that. Automatic backup, backup-restore, sweeping, maintenance of indices statistics, and more. HQbird maintenance techniques are designed to keep your systems running smoothly, preventing downtime and ensuring continuous operation even under the heaviest loads.

Cut Costs, Not Corners

Managing extensive Firebird databases can be costly, but it doesn't have to be. HQbird helps decrease your support expenses without compromising on quality, giving you the best value for your investment.

Choose HQbird for a robust, efficient, and cost-effective database management experience. Transform your data handling capabilities today!

Let's schedule 30 minutes call (in English or Portuguese) to discuss how HQbird can solve performance problems in your Firebird based system: support@ib-aid.com

https://ib-aid.com/hqbird

ibsurgeon
Advanced FirebirdSQL Tools & Services