



# DelphiDay

italian conference

## Firebird JSON & Schema

Fabio Codebue



delphiForce



# Fabio Codebue

**P-Soft – Firebird Foundation**



[www.p-soft.biz](http://www.p-soft.biz)



[f.codebue@p-soft.biz](mailto:f.codebue@p-soft.biz)



[@fcodebue](https://twitter.com/fcodebue)



[@delphiforce](https://github.com/delphiforce)



[linkedin.com/in/fcodebue](https://linkedin.com/in/fcodebue)



## DelphiDay

italian conference

19-20 Giugno 2025  
Piacenza



wintech  
italia



## OPEN-SOURCE PROJECTS

[github.com/delphiforce](https://github.com/delphiforce)

### eInvoice4D

[github.com/delphiforce/eInvoice4D](https://github.com/delphiforce/eInvoice4D)

### Firebird Resource

[github.com/fcodebue/firebirdsql-resource](https://github.com/fcodebue/firebirdsql-resource)

Firebird Foundation z.s.  
Presidium Member



19-20 Giugno 2025  
Piacenza



delphiForce



wintech  
italia



# Intro a Firebird 6

# 1



Fabio Codebue

**DelphiDay**  
italian conference



# Firebird 6 an overview

---

- Alpha Release - Q3 2025 - **IN PROGRESS**
- Beta Release - Q4 2025
- Final Release - Q1 2026



Fabio Codebue





# News on Firebird 6

---

## Major features

#5431	SQL-compliant JSON functions	PENDING PR
#1055	Support for tablespaces	IN REVIEW
#7954	Shared metadata cache	IN PROGRESS
#1115, #3031	SQL-standard compliant ROW data type	PENDING PR
#1113	Support for SQL schemas	IN REVIEW



Fabio Codebue







# Minor features (Already completed)

---

- Feature SET AUTOTERM in ISQL
- Allow collation to be a part of data type
- ALTER ... {SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY}
- **Allow to create database with different owner**
- **EXPLAIN statement** and RDB\$SQL package
- SQL:2023 ANY\_VALUE aggregate function
- Improve DECLARE VARIABLE to accept complete value expressions
- **CALL statement, Named arguments for function call, EXECUTE PROCEDURE** and procedure record source
- Allow DEFAULT keyword in argument list
- ALTER FUNCTION ... {DETERMINISTIC | NOT DETERMINISTIC}
- **DROP [IF EXISTS] statement**
- SQL standard FORMAT clause for CAST between string types and datetime



Fabio Codebue





# Firebird Tablespace



# 2



Fabio Codebue

**DelphiDay**  
italian conference





# Tablespace meaning

In relational database systems, a tablespace is

- a **logical storage unit**
- that groups together one or more physical files where database objects (like tables, indexes, etc.) are stored.





# Tablespace meaning

## ✓ Formal Definition (SQL Standard)

- A tablespace is a **named storage location** defined by the database administrator that holds the actual data files on disk.
- The SQL standard (ISO/IEC 9075) includes tablespaces in its specification, although not all databases implement them in the same way, and some may not support them at all.





# Why using tablespace

Using tablespaces

- allows fine-grained control over disk storage,
- performance optimization,
- and easier database maintenance.



Fabio Codebue







# Tablespace meaning

---

Some typical uses:

- **Performance tuning**  
Place large tables or indexes on separate physical disks to reduce I/O contention.
- **Data organization**  
Separate data by module or application logic (e.g., "finance", "hr", etc.).



Fabio Codebue



# Tablespace meaning

---

Some typical uses:

- **Backup strategies**

Group critical vs. non-critical data into different tablespaces for selective backup/restore.

- **Multi-tenancy**

Isolate tenants in a SaaS application by storing their data in separate tablespaces.



Fabio Codebue



# Tablespace syntax - TABLESPACE

```
CREATE TABLESPACE [IF NOT EXISTS] <TS NAME> FILE '/path/to/file'
```

```
ALTER TABLESPACE <TS NAME> SET FILE [TO] '/path/to/file'
```

Either absolute or relative path is allowed.

```
DROP TABLESPACE [IF EXISTS] <TS NAME>
```

For an existing tablespace, it is possible to add a comment using the COMMENT ON statement.

```
COMMENT ON TABLESPACE <TS NAME> IS {'text' | NULL}
```



Fabio Codebue







# Tablespace syntax - TABLE

```
CREATE TABLE ... [[IN] TABLESPACE {<TS NAME> | PRIMARY}]
```

It is also possible to specify a tablespace when creating a column or table constraint

```
<column/table constraint> ::= ... UNIQUE ... [[IN] TABLESPACE {<TS  
NAME> | PRIMARY}] | PRIMARY ... [[IN] TABLESPACE {<TS NAME> |  
PRIMARY}] | REFERENCES ... [[IN] TABLESPACE {<TS NAME> |  
PRIMARY}] ...
```

```
ALTER TABLE <TABLE NAME> SET TABLESPACE [TO] {<TS NAME> | PRIMARY}
```

The table data will be moved to the specified tablespace or to the main database. It is also possible to specify a tablespace when adding column or table constraints.



# Tablespace syntax - INDEX

```
CREATE INDEX ... [[IN] TABLESPACE {<TS NAME> | PRIMARY}]
```

By default, table indexes are created in the same tablespace as the table itself.

```
ALTER INDEX ... [SET TABLESPACE [TO] {<TS NAME> | PRIMARY}]
```

The index data will be moved to the specified tablespace or to the main database.







# Tablespace syntax - ODS

A new table **RDB\$TABLESPACES**:

- RDB\$TABLESPACE\_ID - INTEGER
- RDB\$TABLESPACE\_NAME - CHAR (63) # name of a tablespace
- RDB\$SECURITY\_CLASS - CHAR (63) # security class for tablespace
- RDB\$SYSTEM\_FLAG - SMALLINT # reserved
- RDB\$DESCRIPTION - BLOB TEXT # description of a tablespace
- RDB\$OWNER\_NAME - CHAR (63) # owner of a tablespace
- RDB\$FILE\_NAME - VARCHAR (255) # file where a tablespace data are located
- RDB\$OFFLINE - BOOLEAN # reserved for future
- RDB\$READ\_ONLY - BOOLEAN # reserved for future



Fabio Codebue







# Tablespace syntax - ODS

New field in **RDB\$INDICES**:

- RDB\$TABLESPACE\_NAME - CHAR (63)

New field in RDB\$RELATION\_FIELDS:

- RDB\$TABLESPACE\_NAME - CHAR (63)

New fields in RDB\$RELATIONS:

- RDB\$TABLESPACE\_NAME - CHAR (63)
- RDB\$POINTER\_PAGE - INTEGER # a number of the first pointer page of a relation
- RDB\$ROOT\_PAGE - INTEGER # a number of the root page of a relation





# Tablespace syntax - backup

## Logical backup

`gbak -b`

- works as usual for now.
- It gets data from a database
- transparently working with tablespaces.

This is an explicit option, not a default action.



Fabio Codebue





# Tablespace syntax - restore

## Logical restore

```
gbak -c -ts_map[ping] <path to file> -ts <tablespace> <path>
```

- option is required for correct database recovery if its backup contains tables or indexes saved in tablespaces.
- To do this, specify the path to file, which consists of lines with two values: the first column is the name of the tablespace, the second column is the new location of the tablespace.
- You can specify either an absolute path or a relative path.

```
TS1 /path/to/tablespace1.dat
```

```
TS2 /path/to/tablespace2.dat
```







# Tablespace syntax - nbackup

```
ALTER DATABASE {BEGIN | END} BACKUP
```

will put not only the main database file, but also all tablespaces into safe copy mode.

A delta file will be created for each tablespace.



Fabio Codebue





# Tablespace syntax - show

```
SHOW {TABLESPACES | TABLESPACE <TS NAME>}
```

- Displays a list of all ts names *in alphabetical order*
- or information about the specified ts specific tablespace



Fabio Codebue





# Tablespace syntax - replication

There is an **apply\_tablespaces\_ddl** parameter for replication.

If this parameter is disabled, tablespaces-related DDL statements and CREATE/ALTER TABLE/INDEX clauses will not be applied to the replica.

This is used if the replica has its own set of tablespaces or none at all.







# Tablespace syntax - replication

`apply_tablespaces_ddl_enabled`

- 1) Specify absolute path when creating TS. Then the replica machine should follow the directory hierarchy.
- 2) Specify relative path when creating TS
- 3) The first two options can be problematic if the master and replica are on the same computer.



Fabio Codebue



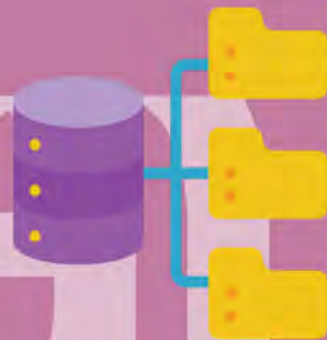
# Tablespace syntax - Limitations

- It's possible to create up to 253 tablespaces.
- Operators to move an index or table to a tablespace require an exclusive database lock.
- Miss support to move blobs into separate tablespace





# Firebird schema



# 3



Fabio Codebue





# Schema meaning

---

In relational database systems, a **database schema** is

- **a logical container or namespace that groups together related database objects** - such as tables, views, indexes, procedures, and other elements.
- Schemas help organize and isolate data within a single database, and are part of the SQL standard (e.g., SQL:2003+)



Fabio Codebue





# Schema meaning

---

## ✓ Informal Definition

- A schema is like a folder inside a database,
- where each folder can contain its own set of tables and other objects,
- allowing better organization and control.



Fabio Codebue





# Schema: Why is important now

- **Logical separation of data**: e.g., separate sales, hr, and inventory in the same database.
- **Multi-tenancy**: each tenant/customer can have their own schema.
- **Access control**: grant different privileges to different schemas.
- **Avoid naming conflicts**: same table name (e.g., users) in different schemas.







# Schema: Practical examples

---

- Use Case 1 – Modular Organization:
  - 
  - A company could have:
    - `finance.employees`
    - `sales.employees`
  -
- two tables named `employees`, but in different schemas.





# Schema: Practical examples

Use Case 2 – SaaS Multi-Tenant System:

- Each customer has their own schema:
  - tenant1.orders
  - tenant2.orders



Fabio Codebue





# Schema: Isolation

---

- Isolation refers to the **separation of database objects** (like tables, views, procedures) in different schemas,
- objects in one schema do not interfere with objects in another.
- Allows multiple modules or applications to coexist in the same database without name collisions or unwanted interaction.







# Schema: Isolation

---

## Benefits:

- Avoid conflicting object names (e.g., each schema can have its own users table).
- Encapsulate business logic within each schema.



Fabio Codebue





# Schema: Isolation

---

```
-- Create two isolated schemas
```

```
CREATE SCHEMA tenant1;
```

```
CREATE SCHEMA tenant2;
```

```
-- Create identical table structure in  
both schemas
```

```
CREATE TABLE tenant1.orders (  
    id INT PRIMARY KEY,  
    customer_name VARCHAR(100)  
);
```

```
CREATE TABLE tenant2.orders (  
    id INT PRIMARY KEY,  
    customer_name VARCHAR(100)  
);
```

```
-- Now we can query them separately
```

```
SELECT * FROM tenant1.orders;
```

```
SELECT * FROM tenant2.orders;
```



Fabio Codebue



# Schema: Security

---

Schema security **refers to controlling access to the schema** and the objects inside it. You can grant or revoke privileges to users or roles for:

- Using the schema (USAGE)
- Accessing specific objects within the schema (e.g., SELECT, INSERT)

This is key in multi-user environments or for compliance and data protection.



Fabio Codebue







# Schema: Security

---

- Grant access to use the schema

```
GRANT USAGE ON SCHEMA tenant1 TO USER alice;
```

- Grant SELECT permission on a specific table

```
GRANT SELECT ON tenant1.orders TO USER alice;
```

- Revoke write access (if it was previously granted)

```
REVOKE INSERT, UPDATE, DELETE ON tenant1.orders FROM  
USER alice;
```





# Schema: multi-tenancy

---

- Multi-tenancy means **serving multiple customers** or tenants using the same application and same database, while **keeping their data isolated**.
- Using one schema per tenant is a common SQL pattern for achieving this.



Fabio Codebue





# Schema: multi-tenancy

---

## Advantages

- Cleaner logical separation than row-level tenant\_id filters.
- Better access control and auditability.
- Easier backup/migration per tenant.



Fabio Codebue







# Schema: multi-tenancy

- Create schemas per tenant

```
CREATE SCHEMA tenant_acme;  
CREATE SCHEMA tenant_zenit;
```

- Create customer tables

```
CREATE TABLE tenant_acme.customers (  
    id INT PRIMARY KEY,  
    name VARCHAR(100)  
);  
  
CREATE TABLE tenant_zenit.customers (  
    id INT PRIMARY KEY,  
    name VARCHAR(100)  
);
```





# Schema: multi-tenancy

Grant access to specific users

```
GRANT USAGE ON SCHEMA tenant_acme TO USER acme_admin;  
GRANT SELECT, INSERT, UPDATE ON tenant_acme.customers TO USER  
acme_admin;
```

```
GRANT USAGE ON SCHEMA tenant_zenit TO USER zenit_admin;  
GRANT SELECT, INSERT, UPDATE ON tenant_zenit.customers TO USER  
zenit_admin;
```



Fabio Codebue





# Firebird JSON

4



Fabio Codebue





# Why JSON in SQL world?

- Relational databases (RDBMS) are traditionally based on structured data stored in tables with defined schemas.
- Modern applications - especially web and API-based systems - often deal with semi-structured or unstructured data, typically formatted as JSON.



Fabio Codebue



# Why JSON in SQL world?

Needs	JSON benefits
Flexible, evolving data models	JSON allows dynamic fields without altering schema
Integrate with APIs	JSON is the de facto standard for API communication
Store complex, nested data	JSON can model documents, arrays, and hierarchies
Combine structured + unstructured	Hybrid use of SQL and JSON lets you get the best of both worlds





# Why JSON in SQL world?

Example Scenarios:

- **Storing user preferences** or **configurations** without creating many nullable columns.
- **Persisting API request/response** bodies for auditing.
- Managing **event logs** or telemetry data in a flexible format.



Fabio Codebue







# JSON: SQL Standard support

ISO/IEC SQL Standard (SQL:2016+) introduces:

- **JSON Data Type** (JSON)
- JSON\_VALUE – extract a scalar value
- JSON\_QUERY – extract an object or array
- JSON\_OBJECT / JSON\_ARRAY  
construct JSON from SQL values
- IS JSON – check if a value is valid JSON



Fabio Codebue



# JSON: FB implementation datatype

Firebird 6 introduces **native JSON support** using a special JSON data type built on top of BLOB SUB\_TYPE TEXT

**Firebird's JSON column** is:

- Stored as text (internally a BLOB SUB\_TYPE TEXT)
- Parsed and validated by JSON-aware functions
- Indexable via computed fields (for scalar properties)



Fabio Codebue





# JSON: FB syntax – JSON\_ARRAY

`<JSON constructor null clause> ::= ABSENT ON NULL | NULL ON NULL`

`<JSON output clause> ::= RETURNING <data type> [ FORMAT <JSON representation> ]`

`<JSON representation> ::= JSON | SQL | AUTO`

`<JSON value expression> ::= <value expression> [ <JSON input clause> ]`

`<JSON input clause> ::= FORMAT <JSON representation>`

`<JSON representation> ::= JSON | SQL`

The JSON representation is declared in the standard as:

`<JSON representation> ::= JSON [ ENCODING { UTF8 | UTF16 | UTF32 } ] |`

`<implementation-defined JSON representation option>`

ENCODING is what I believe is the CHARSET of our string



Fabio Codebue







# JSON: FB syntax – JSON\_ARRAY

## Example

```
- select JSON_ARRAY(false, 'false', 'false' format json) from  
rdb$database;
```

=====

```
[false,"false",false]
```

```
- SELECT JSON_ARRAY(SELECT RDB$RELATION_ID FROM RDB$RELATIONS WHERE  
RDB$RELATION_ID < 5) FROM RDB$DATABASE;
```

=====

```
[0,1,2,3,4]
```





# JSON: FB syntax – JSON\_OBJECT

```
RECREATE TABLE OBJECT_SOURCE (ID INT, NUM_INT INTEGER, NUM_DF  
DECFLOAT(16), NUM_DEC DECIMAL(8,4), LOGIC BOOLEAN, STR_1 VARCHAR(100),  
STR_2 VARCHAR(100), STR_3 VARCHAR(100), STR_4 CHAR(20), JS_TIME TIME);
```

```
- INSERT INTO OBJECT_SOURCE (ID, STR_1, STR_4) VALUES(1, 'Simple  
{text} [with] different $.symbols: , false, true, null, 123.45e-3',  
'testing test');
```

```
- SELECT JSON_OBJECT('test_key': 'Double "quote" symbol' RETURNING  
VARCHAR(40)) as TEST_4 from RDB$DATABASE;
```





# JSON: FB syntax- JSON\_OBJECTAGG

- RECREATE TABLE OBJECTAGG\_SOURCE (ID INT UNIQUE NOT NULL, CATEGORY VARCHAR(20), BRAND VARCHAR(30), PRICE DEC(7,2), QNT INT, ACCESS BOOLEAN, DESC VARCHAR(50));
- INSERT INTO OBJECTAGG\_SOURCE VALUES(1, 'Sneakers', 'NIKE', 5000.23, 10, True, 'Sport style');
- INSERT INTO OBJECTAGG\_SOURCE VALUES(2, 'Sneakers', '\$ADIDAS', 6384.10, 2, False, '"Error" description');
- SELECT JSON\_OBJECTAGG(g.brand: (g.PRICE/1000) RETURNING VARCHAR(300)) as TEST\_19 from OBJECTAGG\_SOURCE as g where ID < 5;
- SELECT JSON\_OBJECTAGG(brand: desc RETURNING VARCHAR(300)) as TEST\_11 from OBJECTAGG\_SOURCE where ID < 7;







# JSON: FB JSON\_VALUE

---

- **JSON\_VALUE** take 2 main arguments: the JSON text itself and the JSON Path. Example:

```
SELECT JSON_VALUE(TEXT, '$.name') FROM RDB$DATABASE;
```



# JSON: FB JSON\_QUERY

```
RECREATE TABLE QUERY_SOURCE(ID INT UNIQUE NOT NULL,  
JSON_TEXT BLOB SUB_TYPE 1 NOT NULL);
```

```
- INSERT INTO QUERY_SOURCE VALUES(0, '{"element":  
[1,"text",{"data":"sample"},[2,3,4]],"tags":["Apple",  
{"id":"45-DB-87"},[1,2,345.65],"$3304.25"],"mass":  
[123,null,[],{"data":["new",87.6]},[{"product":  
["machine"]}],765,true},{},false,"inch"]}');
```

```
SELECT json_query(TEXT, '$.tags') FROM RDB$DATABASE
```





# JSON: FB JSON\_EXISTS

- JSON\_EXISTS Example:

```
SELECT JSON_EXISTS(JSON_TEXT,  
    '$[1][2][2]' ERROR ON ERROR) as TEST_43  
from EXISTS_SOURCE where ID = 0;
```

```
SELECT JSON_EXISTS(JSON_TEXT,  
    '$[0].element[3]' ERROR ON ERROR) as TEST_47  
from EXISTS_SOURCE where ID = 0;
```







# JSON: FB JSON\_TABLE

---

JSON\_TABLE is a feature used to

- convert JSON into a table.
- It functions similarly to a selectable procedure (or the new UNLIST function).



Fabio Codebue





# JSON: FB JSON\_TABLE

---

- The first <json\_path> is a JSON Path pointing to the root element, which will be split into columns.
- There are 4 types of columns:
  - Regular. Extracts a simple scalar value, like JSON\_VALUE
  - Formatted. Extracts an object or array, like JSON\_QUERY.
  - For ordinality: It is simply a counter.
  - Nested column. Allows to split input objects into more columns.





# JSON: FB JSON\_TABLE

```
SELECT * FROM JSON_TABLE(  
    '[{"codes":[42,43], "pass_word":"banana", "extra":{}},  
     {"codes":[-1], "pass_word":"rdb", "extra":{"error message":"Invalid user  
id"}}]', '$[*]'  
    COLUMNS (  
        ID FOR ORDINALITY,  
        PASS_WORD varchar(100) PATH '$.pass_word',  
        NESTED PATH 'lax $.codes[*]' columns (  
            CODE int PATH 'lax $'  
        ),  
        EXTRA varchar(300) CHARACTER SET UTF8 FORMAT JSON PATH '$.extra'  
    )  
) AS JT;
```

ID	PASS_WORD	CODE	EXTRA
1	banana	42	{}
2	banana	43	{}
3	rdb	-1	{"error message":"Invalid user id"}



Fabio Codebue







# JSON: Funzioni native JSON

The path syntax is quite intuitive. It specifies fields and array elements. \$ is the root element. For example, for JSON:

```
{"my_object":{"my_array":  
["first","second","third","fourth","fifth"]}}
```

using the path \$.my\_object.my\_array[1, 3 to 4, -1] will return

```
"second", "fourth", "fifth", "fifth"
```





# JSON: FB advance features

- Methods and mathematics:

```
$.first.abs() + $.second.floor() + 10
```

- Filters

```
$.array[*] ? (@ > 21)
```

```
$.myObject ? (@.string starts with "Hello")
```

- Child filters

```
$.primary ? (@.id == 42).secondary ? (@.name ==  
"error").value
```





# JSON: FB advance features

- Path after the method keyvalue()

`$.primary.keyvalue().value.secondary`

The path has two modes: lax and strict. In the first mode, parsing is more lenient, allowing auto unwrapping of arrays or auto wrapping of objects into arrays. For example

`lax $.array ? (@ > 21) with $ = {"array": [1, 2, 3]}`

transforms into `lax $.array[*] ? (@ > 21)`

`lax $.value[*] ? (@ > 21) with $ = {"value": 42}`

transforms into `lax $.value ? (@ > 21)`

All functionality and additional clauses (ON ERROR, ON EMPTY) are implemented according to the standard.







# Fabio Codebue

**P-Soft – Firebird Foundation**

 [www.p-soft.biz](http://www.p-soft.biz)

 [f.codebue@p-soft.biz](mailto:f.codebue@p-soft.biz)

 [@fcodebue](https://twitter.com/fcodebue)

 [@delphiforce](https://github.com/delphiforce)

 [linkedin.com/in/fcodebue](https://linkedin.com/in/fcodebue)



19-20 Giugno 2025  
Piacenza



wintech  
italia



THANK YOU



Fabio Codebue

