Safire Database System V1

End User and Software Developer Guide

Product-facing edition

This guide describes database concepts, files, tables, records, field types, dictionary rules, query and form binding, migrations, maintenance, backups, restore preview, diagnostics, and developer-facing database objects.

Contents

1. Purpose and audience

This guide describes the Safire Database System from two practical viewpoints.

End users and support staff use it to understand where application data lives, what safe maintenance actions are available, how backups and restore previews work, and what information is useful when support is needed.

Software developers use it to understand database concepts, dictionary definitions, database files, table and record structure, field types, keys, relations, queries, transactions, browse and update forms, migrations, maintenance, diagnostics, and distribution bundles.

The guide is product-facing. It describes the Safire database model and the public application contract. It does not depend on private development folders, temporary proof files, milestone names, or hidden implementation notes.

2. Product overview

The Safire Database System is the data layer used by Safire business applications. It stores structured business data, keeps record identity stable, enforces dictionary rules, maintains keys and indexes, supports safe migrations, and provides maintenance functions such as verify, compact, backup, restore preview, and diagnostic bundle creation.

Safire applications normally interact with the database through high-level objects and dictionary-driven screens. A user sees browses, update forms, lookup windows, reports, dashboards, and business workflows. A developer sees database definitions, table definitions, fields, keys, relations, validation rules, and query objects.

The database system is designed around the following product principles:

• Source and dictionary definitions describe the intended data model.

• Tables store records with stable field definitions.

• Keys and indexes make browse, lookup, and query operations repeatable.

• Migrations change schema in a controlled and reversible way where possible.

• Maintenance tools check and repair product-owned data structures.

• Backups are created before risky operations.

• Restore preview shows what would be restored before live data is touched.

• Diagnostic bundles explain problems without requiring the user to inspect private storage structures.

3. End user summary

For an end user, the database is simply the application's data store. You do not normally open database files directly. You open the Safire application and work with customer records, stock records, invoices, assets, orders, jobs, reports, or whichever business records the application provides.

The safest user-level rules are:

• Use the application's backup command before major imports, upgrades, or bulk edits.

• Use restore preview before restoring any backup set.

• Do not copy live database files while the application is writing data.

• Do not manually edit database storage files.

• Keep the complete database folder or container together.

• When reporting a problem, send the diagnostic bundle requested by support, not random partial files.

• Let the application finish migrations before closing it.

• If a verify command reports errors, stop using the affected application until support has reviewed the report.

The database system is designed so that normal maintenance can be done through application commands, not by manually changing storage files.

4. Key concepts

A Safire database is made up of predictable concepts.

Database - the whole data store for an application or application module.

Dictionary - the formal description of tables, fields, keys, relations, validation rules, display names, default values, browse columns, update forms, and migration rules.

Table - a named collection of records of the same type, such as Customers, Products, Suppliers, Orders, or StockMovements.

Record - one stored business item in a table. A customer record stores one customer. A product record stores one product.

Field - one named value inside a record, such as CustomerNumber, Name, Balance, Active, CreatedAt, or ProductCode.

Key - one or more fields used to uniquely identify records or to define a stable order.

Index - maintained access structure used to find and order records efficiently.

Relation - a rule connecting records in different tables, such as Orders belonging to Customers.

Query - a filtered, sorted, paged, or grouped read operation over one or more tables through the Safire database object model.

Transaction - a protected group of changes that must all succeed together or be rolled back together.

Migration - a controlled schema change, such as adding a field, renaming a field, changing a default, or recreateing an index.

Maintenance - safe operations such as verify, compact, backup, restore preview, schema dump, statistics dump, and diagnostic bundle creation.

5. Database file and folder model

The physical layout may be stored as a folder, a container, or a combination of both depending on the selected deployment style. Developers and users should rely on Safire tools rather than undocumented byte layouts.

A product-facing database layout contains these logical parts:

Item	Purpose	User edits directly	Developer edits directly
Database catalog	Identifies the database, version, tables, and storage units	No	No
Dictionary definition	Describes tables, fields, keys, relations, forms, and validation	No	Through Safire source or dictionary tools
Table storage	Stores records for each table	No	No
Index storage	Maintains key order and lookup paths	No	No
Change journal	Protects active updates and recovery windows	No	No
Migration history	Records completed schema changes	No	Through migration tools
Backup sets	Restorable copies of database state	No	Created through tools
Diagnostic traces	Maintenance and support evidence	No	Read through tools
Release notes	Human-readable database change notes	Yes	Yes
Upgrade plan	Declares intended schema changes	No	Through release tools

A typical deployment may show only a single database container and backup files. A developer or support installation may show a folder with visible subfolders for schema, data, indexes, journals, backups, and diagnostics. Both forms represent the same product model.

6. File descriptions

The following product-facing file descriptions are used in this guide. The actual runtime may store several logical files inside one physical container.

File or unit	Typical extension	Description	Safe to copy while open
Database container	.safdb	Main database container or database folder marker	No
Dictionary file	.safdct	Public schema and form metadata	Only when application is closed
Table store	.saftbl	Stored records for one or more tables	No
Index store	.safidx	Recreateable lookup and ordering structures	No
Journal file	.safjnl	Active transaction and recovery journal	No
Migration file	.safmig	Migration plan or migration history	Only through tools
Backup set	.safbak	Database backup created by Safire	Yes, after creation completes
Diagnostic bundle	.safdiag	Support bundle containing safe diagnostic output	Yes
Release manifest	.safrel	Describes distributed database schema and required upgrade actions	Yes

Never treat the extension list as permission to edit a file manually. The supported path is always through Safire database commands or application maintenance screens.

7. Tables and records

A table stores records with the same structure. Every table has a name, a record layout, one or more keys, optional validation rules, optional relations, and optional browse/update form metadata.

A record is stored according to the table definition. Each record contains values for the fields defined in that table. Fields may have defaults, required rules, display names, formatting rules, lookup rules, and validation rules.

Recommended table design rules:

• Give every table a stable primary identity field.

• Use clear business names for fields.

• Keep display labels separate from storage names.

• Use required fields only where the business rule truly requires them.

• Prefer explicit defaults for status, active flags, date created, and numeric counters.

• Do not use a display value as the only identity for a record.

• Add indexes for common browse and lookup paths.

• Store audit fields where business accountability matters.

Example record definition:

```
Table Customer
    Field CustomerId      AutoNumber  Required Unique
    Field AccountNumber   Text(20)     Required Unique
    Field Name            Text(120)    Required
    Field Phone           Text(40)
    Field Email           Text(160)
    Field Balance         Money        Default 0
    Field Active          Boolean      Default true
    Field CreatedAt       DateTime     Default Now
```

```
End
```

8. Record identity

Every business table should have a stable identity. A stable identity is not the same thing as a display code.

A good identity field:

• never changes after the record is created;

• is unique in the table;

• is small enough to index efficiently;

• is not based on a value that users may rename;

• is not reused after deletion;

• can be referenced safely by related records.

A display code may still be useful. For example, AccountNumber or ProductCode may be visible to users and can also be unique. But the internal identity should remain stable even if a visible code later changes due to business policy.

Recommended pattern:

```
Field CustomerId     AutoNumber Required Unique
Field AccountNumber  Text(20)    Required Unique
Field Name           Text(120)   Required
```

Relations should normally point to the stable identity field. Browses and forms can show the friendly display code and name.

9. Field type overview

Safire field types are intended to be understandable to business developers while still being precise enough for stable storage and validation.

Field type	Use for	Example
Text(n)	Short text with maximum length	Name, Code, Phone
LongText	Notes and descriptions	Customer notes
Integer	Whole numbers in normal business range	Quantity, Count
LongInteger	Larger whole numbers	Large counters
Decimal(p,s)	Exact decimal values	Weight, rate, measurement
Money	Currency values	Balance, Price, Cost
Boolean	True/false values	Active, Taxable
Date	Calendar date	InvoiceDate
Time	Time of day	OpeningTime
DateTime	Date and time together	CreatedAt
AutoNumber	System assigned numeric identity	CustomerId
Guid	Globally unique identity	ExternalId
Enum	Controlled list of values	Status
Binary	Small binary value	Hash, token
Blob	Large binary content	Image, attachment
JsonText	Structured text payload	Settings, metadata

Use the narrowest type that still represents the business requirement clearly.

10. Text and long text fields

Text fields store ordinary strings. Use a maximum length that matches the business need, not an arbitrary large value.

Recommended examples:

Purpose	Recommended type	Notes
Customer name	Text(120)	Long enough for trading names
Product code	Text(40)	Often indexed and unique
Email address	Text(160)	Validate format separately
Phone number	Text(40)	Store as text, not number
Postal code	Text(20)	Store leading zeroes safely
Short status	Enum or Text(20)	Prefer Enum for controlled values
Notes	LongText	Not normally indexed

Text rules:

• Store phone numbers, postal codes, tax numbers, and registration numbers as text.

• Do not store formatted currency or dates as text unless the value is only a label.

• Use trimming rules carefully; some codes may intentionally include spaces.

• Use case-insensitive search only where the business rule expects it.

• Do not index long notes unless a search feature specifically requires it.

11. Numeric fields

Numeric fields represent values that the application calculates, compares, sorts, or totals.

Type	Use	Notes
Integer	Common whole numbers	Quantities, counts, levels
LongInteger	Larger whole numbers	Large counters, imported IDs
Decimal(p,s)	Exact numeric values	Measurements, exchange rates, percentages
Money	Currency	Totals, balances, unit prices

Numeric design rules:

• Use Money for currency values.

• Use Decimal for exact business measurements.

• Do not use floating point style values for financial totals.

• Store percentages as Decimal with clear scale rules.

• Decide whether negative values are allowed.

• Define default zero values only where zero is a meaningful business value.

• Use null only when missing and zero have different meanings.

Example:

```
Field Quantity       Integer      Required Default 0
Field UnitPrice      Money        Required Default 0
Field DiscountRate   Decimal(7,4) Default 0
Field Total          Money        ReadOnly
```

12. Date and time fields

Date and time values should be stored as date-aware types, not as display text.

Type	Use
Date	Calendar date without time
Time	Time of day without date
DateTime	Combined date and time

Rules:

• Use Date for invoice dates, birth dates, due dates, and period dates.

• Use Time for shift start, shift end, opening time, closing time, and schedule time.

• Use DateTime for audit fields such as CreatedAt, UpdatedAt, PostedAt, and ImportedAt.

• Store display formatting separately from stored values.

• Do not use a blank text field to represent an unknown date.

• Use null for unknown, not-applicable, or not-yet-recorded dates.

• Define time zone policy at application level when data crosses regions.

Example:

```
Field InvoiceDate   Date      Required
Field DueDate       Date
Field CreatedAt     DateTime  Default Now
Field UpdatedAt     DateTime
```

13. Boolean, enum, and status fields

Boolean fields store true or false. Use them when there are only two states.

Enum fields store one value from a controlled list. Use them when the business value has several named states.

Examples:

```
Field Active Boolean Default true

Enum InvoiceStatus
    Draft
    Posted
    Paid
    Cancelled
End

Field Status InvoiceStatus Required Default Draft
```

Guidelines:

• Use Boolean for yes/no values such as Active, Locked, Taxable, Printed.

• Use Enum for statuses such as Draft, Posted, Paid, Cancelled.

• Do not use free text for important statuses.

• Do not delete enum values that already exist in stored records; retire them or migrate records first.

• Define display labels separately if users need friendly wording.

14. Binary and attachment fields

Binary and blob fields store non-text data. Use them only when the application truly needs to keep binary content inside the database.

Type	Use	Notes
Binary	Small fixed or variable binary value	Hashes, small tokens
Blob	Large binary value	Images, documents, attachments

Guidelines:

• Keep large attachments outside primary browse paths.

• Do not display blob content in normal record lists.

• Store file name, content type, size, and checksum as separate metadata where useful.

• Include binary data policy in backup planning.

• Use diagnostic bundles carefully so private attachments are not included unless support explicitly needs them.

Example:

```
Field Photo           Blob
Field PhotoName       Text(160)
Field PhotoType       Text(80)
Field PhotoSize       LongInteger
Field PhotoChecksum   Text(128)
```

15. Null, empty, and default values

Safire distinguishes three common states:

State	Meaning	Example
Null	No value recorded	Unknown birth date
Empty	A recorded blank value	Optional note is blank
Default	Value supplied automatically	Active defaults to true

Rules:

• Use null when unknown has a different meaning from blank or zero.

• Use empty text when the user intentionally leaves optional text blank.

• Use zero only when zero is a real numeric value.

• Use defaults for normal starting values such as Active, CreatedAt, and Balance.

• Do not use magic values such as 1900-01-01 to mean unknown.

Example:

```
Field Email       Text(160) NullAllowed
Field Balance     Money     Required Default 0
Field Active      Boolean   Required Default true
Field DateClosed  Date      NullAllowed
```

16. Keys and indexes

Keys and indexes make records findable and browsable.

A **primary key** uniquely identifies a record. A **unique key** prevents duplicate values. A **browse key** defines a stable user-facing order. A **lookup key** supports fast find-as-you-type and selection controls.

Recommended key patterns:

Table	Key	Purpose
Customer	CustomerId	Stable identity
Customer	AccountNumber	Unique business code
Customer	Name, CustomerId	Stable name browse
Product	ProductId	Stable identity
Product	ProductCode	Unique stock code
Product	Description, ProductId	Stable description browse
Order	OrderId	Stable identity

Table	Key	Purpose
Order	CustomerId, OrderDate, OrderId	Customer order history

Rules:

• Every browse order should be stable, even when display values repeat.

• Add the identity field as the final component of non-unique browse keys.

• Recreate indexes after schema changes that affect indexed fields.

• Do not rely on physical record order.

• Do not index large text or blob fields unless a specialized feature requires it.

17. Relations and lookups

Relations describe how records connect.

Example relation:

```
Relation CustomerOrders
    Parent Customer.CustomerId
    Child  SalesOrder.CustomerId
    OnParentDelete Restrict
End
```

Common relation rules:

Rule	Meaning
Restrict	Parent cannot be deleted while child records exist
Cascade	Child records follow parent delete or update behavior
Clear	Child reference is cleared when parent is removed
Warn	User is warned before action continues

Lookup controls use relations and lookup keys to help users choose valid records.

Example lookup:

```
Lookup CustomerLookup
    Source Customer
    Key NameLookup
    Display AccountNumber, Name
    Return CustomerId
End
```

Lookup rules:

• Return the stable identity field, not only the display label.

• Show enough fields for the user to pick the correct record.

• Apply active/inactive filters where appropriate.

• Validate that the selected record still exists when saving.

18. Dictionary-driven browse forms

A browse form displays a list of records from a table or query. The dictionary can define columns, sort order, filters, lookup actions, edit actions, and validation context.

Example browse definition:

```
Browse CustomerBrowse
    Source Customer
    Sort Name, CustomerId
    Columns AccountNumber, Name, Phone, Balance, Active
    Filter Active = true
    Action New CustomerUpdate
```

```
    Action Edit CustomerUpdate
End
```

Good browse design:

• Show the most useful identifying fields first.

• Use a stable sort order.

• Keep columns readable and not overloaded.

• Use filters for active records, periods, branches, or user responsibility.

• Provide clear actions for add, edit, delete, print, export, and view history.

• Avoid loading unnecessary large fields into normal browses.

Browse forms should be query-backed when the list needs filtering, paging, lookup constraints, or derived display values.

19. Dictionary-driven update forms

An update form lets the user add or edit one record. The dictionary can define which fields are shown, how they are grouped, which fields are required, what defaults are applied, and what validation runs before save.

Example update definition:

```
UpdateForm CustomerUpdate
    Source Customer
    Field AccountNumber Required
    Field Name Required
    Field Phone
    Field Email Validate EmailFormat
    Field Balance ReadOnly
    Field Active
    BeforeSave ValidateCustomer
End
```

Good update form rules:

• Group fields in the order users think about the record.

• Use clear labels and help text.

• Use lookup controls instead of forcing users to type foreign identities.

• Keep calculated fields read-only.

• Validate required fields before saving.

• Validate relations before saving.

• Confirm destructive changes.

• Write audit fields automatically where required.

20. Validation binding

Validation binding connects dictionary rules to forms, database writes, imports, and workflows.

Validation can happen at several levels:

Level	Example
Field validation	Required, maximum length, numeric range, valid date
Record validation	End date must be after start date
Relation validation	Customer must exist and be active
Workflow validation	Cannot post invoice before it balances
Migration validation	New required field must have a default or fill rule

Example:

```
Validation CustomerEmailRule
    Field Customer.Email
    When Email IsNotEmpty
    Require EmailFormat(Email)
    Message "Enter a valid email address."
End
```

Validation messages should tell the user what to fix, not expose storage details.

21. Query model

A Safire query is a product-owned read operation. It can filter, sort, page, and project records without making the user or developer depend on private storage details.

Common query operations:

Operation	Meaning
Where	Filter records by one or more conditions
SortBy	Define stable order
Select	Choose returned fields
Page	Return a page of rows
Limit	Limit maximum rows
Range	Read between two values
StartsWith	Text prefix lookup
Contains	Text containment lookup where supported
Count	Return a count
Exists	Test whether a record exists

Example:

```
Query ActiveCustomerBrowse
    From Customer
    Where Active = true
    SortBy Name, CustomerId
    Select AccountNumber, Name, Phone, Balance
    PageSize 50
End
```

Queries used by forms should be stable and predictable. Always include a final identity component in sort order where duplicate display values are possible.

22. Transactions

A transaction protects a group of changes. Either all changes commit, or all changes roll back.

Use transactions for:

• posting invoices;

• moving stock;

• importing multiple related records;

• creating parent and child records together;

• applying migrations;

• bulk updates;

• recreateing critical indexes.

Example:

```
transaction = Database.BeginTransaction()
```

```
try
    Order.Save()
    StockMovement.Save()
    CustomerBalance.Update()
    transaction.Commit()
catch error
    transaction.Rollback()
    ShowError(error.Message)
end
```

Transaction rules:

• Keep transactions short.

• Do not wait for user input inside an active transaction.

• Roll back on any unexpected error.

• Backup before large migration transactions.

• Record enough diagnostic information to understand failures.

23. Record locking and concurrency

The database system should prevent unsafe overlapping writes. The exact locking policy depends on the application mode and deployment configuration, but the product rules are consistent.

Locking goals:

• prevent two writes from corrupting the same record;

• detect when a record changed since it was loaded;

• keep transactions consistent;

• explain conflicts clearly;

• avoid unnecessary long locks.

Recommended developer pattern:

• Read the record with its version value.

• Show the update form.

• Save only if the stored version still matches.

• If the version changed, show a conflict message and let the user reload.

Example conflict message:

```
This record was changed by another action after you opened it. Reload the record before saving.
```

Do not hide conflicts by silently overwriting newer data.

24. Migrations

A migration is a controlled change to the database schema or stored data shape.

Common migrations:

Migration	Example
Add field	Add Customer.Email
Drop field	Remove unused legacy field after backup
Rename field	Rename TelNo to Phone
Change default	Active default changes to true
Recreate index	Recreate NameLookup after field length change
Add relation	Add Customer to Order relation

Migration	Example
Data fill	Fill new field from existing fields

Migration rules:

• Every migration has a version.

• Every migration declares what it changes.

• Risky migrations require backup first.

• Required new fields need a default or fill rule.

• Field rename should preserve existing values.

• Index recreate should happen after indexed schema changes.

• Restore preview should be available before live restore.

• Migration history should show what was applied and when.

Example migration:

```
Migration AddCustomerEmail
    FromVersion 1
    ToVersion 2
    AddField Customer.Email Text(160) NullAllowed
    RecreateIndex Customer.NameLookup
End
```

25. Database maintenance

Maintenance commands help keep the database healthy.

Command	Purpose	Typical user
Verify	Check schema, tables, indexes, and relations	User or support
Compact	Reclaim wasted space where supported	Support or administrator
Backup	Create a backup set	User or administrator
Restore preview	Show what restore would do	User or support
Schema report	Display dictionary and stored schema	Developer or support
Statistics report	Show table counts and index information	Developer or support
Recreate indexes	Recreate index structures	Support or migration tool
Diagnostic bundle	Gather safe support evidence	User or support

Maintenance rules:

• Backup before compact.

• Backup before migration.

• Run verify after migration.

• Run restore preview before restore.

• Do not interrupt compact or index recreate.

• Save diagnostic bundles before attempting repeated repairs.

26. Backup and restore preview

A backup set is a restorable copy of the database state created by Safire tools.

A good backup set includes:

• database identity;

• schema version;

• dictionary snapshot;

• table data;

• index recreate information;

• migration history;

• checksum manifest;

• creation date and time;

• application identity;

• optional diagnostic summary.

A restore preview should report:

• source backup identity;

• target database identity;

• schema version difference;

• tables that would be replaced;

• estimated record counts;

• warnings;

• whether existing data would be overwritten;

• whether the restore is compatible.

Restore preview does not change live data. It is a safety check. A restore should require explicit user confirmation after preview.

27. Distribution bundles

A database-capable application can be distributed with database definitions, release notes, upgrade plans, sample data when permitted, and maintenance commands.

A database distribution bundle should contain:

Item	Purpose
Application files	The runnable application and required support files
Database dictionary	Table, field, key, relation, validation, and form definitions
Upgrade plan	Schema changes required for this version
Backup policy	When backup is required before upgrade
Release notes	Human-readable explanation of database changes
Verification command	Confirms the bundle is complete
Diagnostic command	Creates a support bundle if installation or upgrade fails

Rules:

• Never upgrade live data without a compatible dictionary.

• Always run pre-upgrade checks.

• Create a backup set before destructive or structural changes.

• Verify the database after upgrade.

• Keep release notes understandable to non-developers.

28. Safire database objects for developers

Developers use database objects rather than manipulating storage files directly.

Object	Purpose
Database	Open, create, close, verify, backup, compact, restore preview
Table	Insert, update, delete, find, select, count
Record	Holds field values for one table row
Query	Defines filtered, sorted, paged reads
Transaction	Begin, commit, rollback grouped work
Relation	Describes parent-child connections
Index	Defines lookup and ordering paths
Migration	Applies controlled schema changes
DbMaintenance	Runs verify, compact, backup, report, diagnostics
DbError	Structured database error information

Example:

```
db = Database.Open("MainData")
customers = db.Table("Customer")
query = customers.Query()
query.Where("Active", Equals, true)
query.SortBy("Name", Ascending)
query.PageSize(50)
rows = query.Execute()
```

29. Database object reference

Database

Method	Purpose
Open(name)	Open an existing database
Create(name, dictionary)	Create a new database from a dictionary
Close()	Close the database safely
Table(name)	Get a table object
BeginTransaction()	Start a transaction
Verify()	Run consistency checks
Compact()	Compact storage where supported
Backup(target)	Create a backup set
RestorePreview(source)	Preview restore actions
SchemaReport()	Return schema description
StatisticsReport()	Return table and index statistics

Table

Method	Purpose
NewRecord()	Create a new unsaved record
Insert(record)	Add a record
Update(record)	Save changes to an existing record
Delete(key)	Delete a record by key
Find(key, value)	Find one record
Query()	Create a query over the table
Count()	Count records

Method	Purpose
Validate(record)	Validate without saving

Record

Method	Purpose
Get(field)	Read a field value
Set(field, value)	Set a field value
IsNull(field)	Check missing value state
SetNull(field)	Mark field as missing
Validate()	Validate this record
Clone()	Copy values for safe editing

30. Query object reference

The Query object is used for browse forms, lookups, reports, and developer reads.

Method	Purpose
Where(field, operator, value)	Add a filter condition
AndWhere(field, operator, value)	Add another required condition
OrWhere(field, operator, value)	Add an alternative condition
SortBy(field, direction)	Add sort order
Select(fields)	Choose returned fields
Page(number, size)	Return one page
Limit(count)	Limit maximum row count
Execute()	Return records or rows
Count()	Return count for same filter
Exists()	Return true if any row matches

Operators:

Operator	Meaning
Equals	Equal to
NotEquals	Not equal to
GreaterThan	Greater than
GreaterOrEqual	Greater than or equal to
LessThan	Less than
LessOrEqual	Less than or equal to
Between	Between two values
StartsWith	Text begins with value
Contains	Text contains value where supported
IsNull	Missing value
IsNotNull	Present value
InList	Value is in a supplied list

Example:

```
query = db.Table("Product").Query()
query.Where("Active", Equals, true)
query.AndWhere("Description", StartsWith, SearchText)
query.SortBy("Description", Ascending)
query.SortBy("ProductId", Ascending)
```

```
query.Page(1, 100)
products = query.Execute()
```

31. Error model

Database errors should be structured and useful. A database operation should return or raise a DbError with enough information for the application to handle it.

Error area	Example
Open error	Database not found, not accessible, incompatible version
Schema error	Missing table, missing field, invalid migration
Validation error	Required field missing, duplicate key, invalid relation
Transaction error	Commit failed, rollback completed, conflict detected
Storage error	Write failed, journal recovery needed, checksum mismatch
Maintenance error	Verify failed, compact interrupted, backup target unavailable
Permission error	Application is not allowed to read or write target location

Recommended DbError fields:

Field	Purpose
Code	Stable machine-readable error code
Message	Safe user-facing message
Detail	Developer/support detail
Table	Table name if applicable
Field	Field name if applicable
Operation	Operation being performed
Recoverable	Whether retry or repair may be possible
SuggestedAction	Recommended next step

Example user message:

```
The customer cannot be saved because Account Number is required.
```

Example support detail:

```
Validation failed on Customer.AccountNumber during Insert.
```

32. Diagnostics and support bundles

A diagnostic bundle collects information support can use to understand a database problem.

A safe diagnostic bundle may contain:

- application identity;
- database identity;
- schema version;
- table names and record counts;
- index names and verify results;
- migration history;
- recent maintenance results;
- error summaries;
- configuration summary;
- checksums;

• release notes;

• restore preview reports.

A diagnostic bundle should not include private business records unless the user explicitly approves that level of data collection.

Support workflow:

1. Stop repeated repair attempts. 2. Run verify. 3. Create diagnostic bundle. 4. Create a backup set if possible. 5. Send the diagnostic bundle to support. 6. Wait for repair instructions.

The diagnostic bundle should make support faster without exposing more data than necessary.

33. Performance guidelines

Good database performance starts with good dictionary design.

Guidelines:

• Add indexes for common lookup and browse paths.

• Keep browse queries narrow; return only fields the screen needs.

• Avoid loading large text or binary fields into normal lists.

• Use paging for large browses.

• Use stable sort orders.

• Use transactions for bulk inserts and updates.

• Recreate affected indexes after schema changes.

• Archive old records when the business process permits it.

• Run compact after large delete or archive operations where supported.

• Use statistics reports to find unexpectedly large tables.

Common slow patterns:

Pattern	Better approach
Browse without index	Add browse key or lookup index
Loading all records	Use filters and paging
Loading blob fields in list	Load only when detail screen opens
Updating many records one by one without transaction	Use a transaction
Sorting by non-indexed field in large table	Add an index or redesign browse

34. Security and data protection

Database protection is a shared responsibility between the application, operating system, deployment layout, and user procedures.

Recommended practices:

• Store data in an application-owned data location.

• Restrict write permission to trusted application users and services.

• Protect backups with the same care as live data.

• Do not send diagnostic bundles without reviewing the data policy.

• Avoid storing passwords or secrets as plain fields.

• Mask sensitive values in screens and reports where appropriate.

• Use audit fields for important business changes.

• Keep database files out of casual document folders.

• Do not email live database containers unless encrypted and approved.

Audit field pattern:

```
Field CreatedAt   DateTime Default Now ReadOnly
Field CreatedBy   Text(80) ReadOnly
Field UpdatedAt   DateTime
Field UpdatedBy   Text(80)
Field VersionNo   LongInteger Default 1
```

35. Application data locations

The exact data location is chosen by the application and installer policy. The database system supports the following location styles:

Location style	Use
Per-user data	Single user desktop application data
Per-machine data	Shared application data on one workstation
Portable folder	Demonstration, training, or controlled portable deployments
Managed application folder	Application-controlled deployment with explicit permissions
Server-managed folder	Centralized deployment controlled by an administrator

Rules:

• Keep data separate from application program files when users need write access.

• Keep backups outside the live data folder when possible.

• Keep temporary files separate from durable database files.

• Never point two independent application copies at the same writable data folder unless the edition explicitly supports that mode.

• Use the application's maintenance screen to confirm the active data location.

36. Import and export

Import and export features should use the dictionary for validation and conversion.

Import rules:

• Validate field names before importing.

• Validate field types before saving.

• Preview row counts and errors before committing large imports.

• Use transactions for import batches.

• Write an import summary.

• Backup before imports that update existing records.

Export rules:

• Export only the fields selected by the user or developer.

• Respect user permissions.

• Mask sensitive values where required.

• Include column headings.

• Include filter and date range details in export summary.

Example import flow:

```
Select file
Map columns
Preview first rows
Validate all rows
Show error report
Create backup set
Import inside transaction
Show summary
```

37. Reports and dashboards

Reports and dashboards should use query-backed data paths. They should not read storage files directly.

Report data rules:

• Use dictionary field names and display labels.

• Use query filters for date range, customer, branch, product, or status.

• Use stable sort order.

• Avoid loading unused fields.

• Use transactions only when report generation needs a consistent snapshot.

• Include report parameters in report output where useful.

Dashboard rules:

• Use summary queries.

• Avoid refreshing large tables too often.

• Cache safe summary values where appropriate.

• Show last refresh time.

• Make drill-down actions explicit.

38. Developer source examples

The following examples show the intended readability of Safire database definitions. Exact syntax may evolve, but the product concepts remain stable.

Customer table:

```
Database BusinessData

Table Customer
    Field CustomerId      AutoNumber Required Unique
    Field AccountNumber   Text(20)   Required Unique
    Field Name            Text(120)  Required
    Field Phone           Text(40)
    Field Email           Text(160)
    Field Balance         Money      Default 0
    Field Active          Boolean    Default true
    Field CreatedAt       DateTime   Default Now

    Key PrimaryCustomer   CustomerId Primary
    Key AccountLookup     AccountNumber Unique
    Key NameBrowse        Name, CustomerId
End
```

Order table:

```
Table SalesOrder
    Field OrderId       AutoNumber Required Unique
    Field CustomerId    LongInteger Required
    Field OrderNumber   Text(30) Required Unique
    Field OrderDate     Date Required
    Field Status        Text(20) Required Default "Draft"
    Field Total         Money Default 0
```

```
    Key PrimaryOrder    OrderId Primary
    Key CustomerOrders  CustomerId, OrderDate, OrderId
End

Relation CustomerSalesOrders
    Parent Customer.CustomerId
    Child  SalesOrder.CustomerId
    OnParentDelete Restrict
End
```

39. Maintenance command examples

The command names below are product-facing examples. Actual applications may expose the same operations through menus or maintenance screens.

Verify database:

```
safire db verify BusinessData
```

Create backup set:

```
safire db backup BusinessData --to Backups
```

Preview restore:

```
safire db restore-preview Backups\BusinessData_2026_07_01.safbak --target BusinessData
```

Compact database:

```
safire db compact BusinessData
```

Dump schema report:

```
safire db schema BusinessData --out schema_report.txt
```

Create diagnostic bundle:

```
safire db diagnostics BusinessData --out support.safdiag
```

User-facing maintenance screens should call the same product operations.

40. Supported operation matrix

This matrix summarizes the V1 product-facing operation groups.

Operation group	Supported path
Create database from dictionary	Yes
Open and close database	Yes
Insert record	Yes
Update record	Yes
Delete record	Yes
Find by key	Yes
Browse by key	Yes
Query filter and sort	Yes
Query paging	Yes
Lookup binding	Yes
Validation binding	Yes
Transaction commit and rollback	Yes
Add field migration	Yes
Drop field migration	Yes with backup policy
Rename field migration	Yes
Default fill migration	Yes
Index recreate	Yes

Operation group	Supported path
Verify database	Yes
Compact database	Yes where storage permits
Backup set creation	Yes
Restore preview	Yes
Diagnostic bundle	Yes
Dictionary-driven browse form	Yes
Dictionary-driven update form	Yes
Distribution bundle verification	Yes

41. Troubleshooting guide

Symptom	Likely area	Recommended action
Application cannot open data	Location, permission, missing file	Confirm data location and permissions
Verify reports index error	Index storage	Run recreate indexes after backup
New field is blank after upgrade	Migration default/fill rule	Review migration report
Duplicate key error	Unique key violation	Correct duplicate business value
Lookup shows old records	Filter or active flag	Check lookup filter definition
Browse duplicates rows	Sort order not stable	Add identity field to browse sort
Save button reports validation error	Field or record rule	Correct highlighted fields
Backup fails	Target folder or permission	Choose writable backup location
Restore preview incompatible	Wrong backup or schema version	Select matching backup set
Compact interrupted	Maintenance interruption	Run verify and contact support
Diagnostic bundle too large	Includes large traces or attachments	Create summary-only bundle if available

When troubleshooting, avoid manual file edits. First create a backup set if the database can still be opened, then run verify, then create a diagnostic bundle.

42. Administrator checklist

Before first live use:

• Confirm the data location.

• Confirm backup location.

• Create a test backup set.

• Run restore preview on the test backup set.

• Confirm user permissions.

• Confirm diagnostic bundle creation.

• Confirm the application can reopen after restart.

Before upgrade:

• Read release notes.

• Create backup set.

• Confirm backup set exists.

• Run upgrade preview if available.

• Apply upgrade.

• Run verify.

• Open critical screens.

• Confirm record counts where appropriate.

Before support handoff:

• Record application version.

• Run verify.

• Create diagnostic bundle.

• Note the last successful backup date.

• Note the exact user-facing error message.

43. Developer checklist

Before adding a table:

• Define stable identity field.

• Define required business fields.

• Define primary key.

• Define browse key.

• Define lookup key if the table is selected from other forms.

• Define relations.

• Define validation rules.

• Define browse and update forms.

• Add migration.

• Add backup requirement if data is live.

Before changing a field:

• Check whether the field is indexed.

• Check whether the field is used by forms.

• Check whether the field is used by reports.

• Check whether the field is used by relations.

• Define migration behavior.

• Define default or conversion rule.

• Recreate affected indexes.

• Run verify.

Before distribution:

• Include dictionary.

• Include upgrade plan.

• Include release notes.

• Include verify command.

• Include backup guidance.

• Test extracted bundle verification.

44. Field attributes reference

Fields can carry attributes that control validation, display, storage behavior, and form behavior.

Attribute	Applies to	Meaning
Required	All non-binary fields	User or workflow must supply a value
Unique	Fields or key components	Value cannot duplicate another record
Default	Most field types	Value supplied when a new record is created
NullAllowed	Most field types	Missing value is permitted
ReadOnly	Calculated or protected fields	User cannot edit the value directly
Hidden	Form fields	Field is stored but not shown by default
DisplayLabel	Form and report fields	Friendly caption for users
HelpText	Form fields	Explanation shown to user
Format	Date, time, number, money	Display formatting rule
MinValue	Numeric/date fields	Minimum accepted value
MaxValue	Numeric/date fields	Maximum accepted value
MaxLength	Text fields	Maximum character count
Lookup	Identity/reference fields	Field value must be chosen from a lookup source
Calculated	Derived fields	Value is computed, not typed by user
Audit	Audit fields	Managed by application workflow

Example:

```
Field CreditLimit Money
    Default 0
    MinValue 0
    DisplayLabel "Credit Limit"
    HelpText "Maximum approved customer account balance."
End
```

45. Table attributes reference

A table definition can include attributes that describe identity, display behavior, security policy, audit behavior, and default maintenance behavior.

Attribute	Purpose
DisplayName	Friendly name shown in screens and reports
PluralName	Friendly plural name
PrimaryKey	Main identity key
DefaultBrowse	Browse used when opening the table from navigation
DefaultUpdate	Update form used for add/edit actions
AuditEnabled	Application records create/update information
SoftDelete	Records are marked inactive instead of physically deleted
MigrationPolicy	Rules for schema changes
BackupBeforeUpgrade	Requires backup before structural change
VerifyAfterUpgrade	Requires verify after upgrade

Example:

```
Table Product
    DisplayName "Product"
    PluralName "Products"
    PrimaryKey ProductId
    DefaultBrowse ProductBrowse
```

```
    DefaultUpdate ProductUpdate
    AuditEnabled true
    BackupBeforeUpgrade true
End
```

46. Key and index attributes reference

Keys and indexes should be declared clearly so forms, lookups, reports, and maintenance tools can use them consistently.

Attribute	Meaning
Primary	Main identity key for the table
Unique	Duplicate key values are not allowed
DuplicateAllowed	Duplicate values are allowed
Ascending	Sort component ascending
Descending	Sort component descending
CaseSensitive	Text comparison observes case
IgnoreCase	Text comparison ignores case where supported
ActiveOnly	Key is normally used with active records only
RecreateAfterMigration	Key should be recreated after affected schema changes

Examples:

```
Key PrimaryProduct ProductId Primary Unique
Key ProductCodeLookup ProductCode Unique
Key ProductDescriptionBrowse Description Ascending, ProductId Ascending
Key ActiveProductBrowse Active Ascending, Description Ascending, ProductId Ascending
```

A browse key should end with a stable identity field when earlier fields can repeat.

47. Relation attributes reference

Relations protect integrity between parent and child records.

Attribute	Meaning
Parent	Parent table and field
Child	Child table and field
Required	Child must point to an existing parent
Optional	Child may have no parent
RestrictDelete	Parent delete is blocked when children exist
CascadeDelete	Child records are removed with parent where allowed
ClearChild	Child field is cleared when parent is removed
ValidateOnSave	Relation is checked before saving child
DisplayLookup	Lookup used to choose parent

Example:

```
Relation ProductStockMovements
    Parent Product.ProductId
    Child StockMovement.ProductId
    Required true
    RestrictDelete true
    ValidateOnSave true
    DisplayLookup ProductLookup
End
```

48. Browse metadata reference

Browse metadata controls how a list screen presents records.

Metadata item	Purpose
Source	Table or query used by the browse
Columns	Fields shown in list order
Sort	Stable order used by the browse
Filter	Normal filter applied before display
PageSize	Number of rows requested at a time
Locator	Field or fields used for quick find
RowActions	Actions such as view, edit, delete, print
ToolbarActions	Actions shown above the list
EmptyMessage	Text shown when no rows match

Example:

```
Browse ProductBrowse
    Source Product
    Columns ProductCode, Description, QtyOnHand, SellingPrice, Active
    Sort Description, ProductId
    Filter Active = true
    PageSize 100
    Locator Description
    RowAction Edit ProductUpdate
    ToolbarAction New ProductUpdate
End
```

49. Update form metadata reference

Update form metadata controls data entry and editing.

Metadata item	Purpose
Source	Table being edited
Field order	Order of fields on the form
Group	Visual grouping of fields
Required marker	Shows required fields
Lookup editor	Uses a lookup instead of free typing
Read-only editor	Displays protected value without allowing edit
Validation message	Friendly correction message
BeforeSave	Record-level validation or transformation
AfterSave	Follow-up action after successful save

Example:

```
UpdateForm ProductUpdate
    Source Product
    Group "Identity"
        Field ProductCode Required
        Field Description Required
    End
    Group "Stock"
        Field QtyOnHand ReadOnly
        Field ReorderLevel
    End
    Group "Pricing"
        Field CostPrice
        Field SellingPrice
    End
    BeforeSave ValidateProduct
End
```

50. Migration manifest structure

A migration manifest describes a versioned database change.

Recommended fields:

Field	Meaning
MigrationId	Stable migration identity
FromVersion	Required starting schema version
ToVersion	Resulting schema version
Description	Human-readable reason for change
RequiresBackup	Whether a backup set must exist first
Steps	Ordered migration steps
Verification	Checks to run after migration
RollbackNote	What rollback or restore path is available

Example:

```
MigrationManifest AddProductReorderLevel
    MigrationId "AddProductReorderLevel"
    FromVersion 4
    ToVersion 5
    RequiresBackup true
    Step AddField Product.ReorderLevel Integer Default 0
    Step RecreateIndex Product.ActiveProductBrowse
    Verify Table Product
End
```

51. Backup set manifest structure

A backup set manifest identifies what was saved and how it can be checked.

Recommended manifest fields:

Field	Meaning
BackupId	Unique backup identity
CreatedAt	Date and time backup was created
Application	Application that created it
DatabaseName	Source database name
SchemaVersion	Schema version at backup time
TableSummary	Table names and record counts
ChecksumManifest	File or block checksums
CreatedBy	User or maintenance action that created it
Notes	Optional user notes

Restore tools should read the manifest before offering restore. If the manifest is missing or damaged, restore should stop unless a support recovery path is explicitly chosen.

52. Diagnostic bundle manifest structure

A diagnostic bundle manifest explains what support evidence is included.

Recommended manifest fields:

Field	Meaning
DiagnosticId	Unique diagnostic identity
CreatedAt	Date and time created

Field	Meaning
Application	Application identity
DatabaseName	Database identity
SchemaVersion	Current schema version
VerifyResult	Latest verify result
MaintenanceHistory	Recent maintenance actions
ErrorSummary	Recent safe error summaries
TableSummary	Table names and counts
PrivacyLevel	Whether business data is excluded, summarized, or included

The default diagnostic bundle should exclude business record content. If support needs actual records, the user should approve that separately.

53. Sample inventory dictionary

This sample shows how a stock table can be described using the same product concepts.

```
Table Product
    Field ProductId      AutoNumber Required Unique
    Field ProductCode    Text(40) Required Unique
    Field Description    Text(160) Required
    Field Department     Text(80)
    Field QtyOnHand      Decimal(14,4) Default 0
    Field ReorderLevel   Decimal(14,4) Default 0
    Field CostPrice      Money Default 0
    Field SellingPrice   Money Default 0
    Field Active         Boolean Default true

    Key PrimaryProduct ProductId Primary Unique
    Key ProductCodeLookup ProductCode Unique
    Key ProductDescriptionBrowse Description, ProductId
End

Table StockMovement
    Field MovementId     AutoNumber Required Unique
    Field ProductId      LongInteger Required
    Field MovementDate   DateTime Required Default Now
    Field MovementType   Text(30) Required
    Field Quantity       Decimal(14,4) Required
    Field Reference      Text(80)

    Key PrimaryMovement MovementId Primary Unique
    Key ProductHistory ProductId, MovementDate, MovementId
End
```

The Product table is browseable by description and searchable by code. The StockMovement table stores movement history and relates each movement to a product identity.

54. Glossary

Backup set - a restorable database copy created by Safire tools.

Browse - a list screen showing records.

Catalog - database-level metadata identifying tables and storage units.

Dictionary - public schema and UI-related data model definition.

Diagnostic bundle - support file containing safe evidence about database health and failures.

Field - a named value inside a record.

Index - maintained lookup and ordering structure.

Journal - recovery information for active changes.

Key - one or more fields used for identity or order.

Lookup - selection control that returns a valid related record identity.

Migration - controlled schema or stored data shape change.

Query - product-owned filtered, sorted, or paged read operation.

Record - one stored item in a table.

Relation - rule connecting parent and child records.

Restore preview - safety report showing what a restore would do before live data is changed.

Table - collection of records with the same structure.

Transaction - protected group of changes that commits or rolls back as one unit.

Update form - data entry screen for adding or editing a record.

Verify - maintenance operation that checks database consistency.