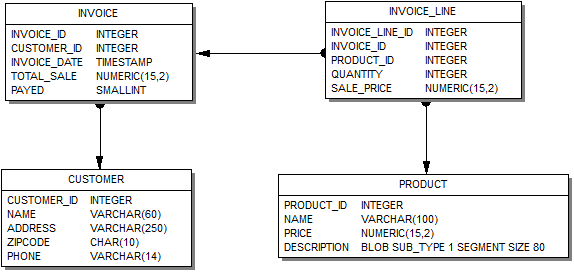
The examples.fdb DatabaseBefore we explore the process of creating applications in various programming languages, we will walk through the creation and preparation of the database that is used as the back-end to all of the sample projects.

The applications work with a database based on the model illustrated in this diagram:



**Disclaimer**

This chapter does not attempt to provide a tutorial about database design or SQL syntax. The model is made as simple as possible to avoid cluttering application development techniques with topics about database modeling and development. We hope some readers might be enlightened by our use of stored procedures to maintain interrelated data. The scripts are all here for you to refer to as you work your way through the projects.

## The requirements for your real-life projects are undoubtedly different from and much more complicated than those for our example projects. Database Creation Script

The tool used here to create the database from a script is isql, that is installed with all the other executables in every Firebird server installation. You could use any other administration tool for Firebird, such as FlameRobin, SQLLY Studio, IBExpert or others.

We will assume that you are working in Windows. Obviously, the formats of path names will differ on other file systems (Linux, Apple Mac, etc.) but the isql tool works the same on all platforms.

Run isql and enter the following script after the SQL> prompt appears :

**CREATE DATABASE** 'localhost:D:\fbdata\2.5\examples.fdb'

**USER** 'SYSDBA' **PASSWORD** 'masterkey'

**PAGE\_SIZE** 8192 **DEFAULT CHARACTER SET** UTF8;

IMPORTANT

The straight single quotes around the user and password arguments are not optional and are not interchangeable with curly quotes, double quotes

or any other kind of quotes.

The user whose name and password are cited in the CREATE DATABASE statement becomes the owner of the database and has full access to all metadata objects. It is not essential that SYSDBA be the owner of a database. Any user can be the owner, which has the same access as SYSDBA in this database.

The actively supported versions of Firebird support the following page sizes: 4096, 8192 and 16384. The page size of 8192 is good for most cases.

The optional DEFAULT CHARACTER SET clause specifies the default character set for string data types. Character sets are applied to the CHAR, VARCHAR and BLOB TEXT data types. You can study the list of available language encodings in an [Appendix to the Firebird Language Reference](https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25-appx06-charsets.html" \l "fblangref25-appx06-tbl-charsets) manual.

All up-to-date programming languages support UTF8 so we choose this encoding.

Now we can exit the isql session by typing the following command:

**EXIT**;

## Database Aliases

Databases are accessed locally and remotely by their physical file path on the server. Before you start to use a database, it is useful and wise to register an alias for its file path and to use the alias for all connections. It saves typing and, to some degree, it offers a little extra security from snoopers by obscuring the physical location of your database file in the connection string.

In Firebird 2.5, the alias of a database is registered in the aliases.conf file in the following way

examples = D:\fbdata\2.5\examples.fdb

In Firebird 3.0, the alias of a database is registered in the databases.conf file. Along with the alias for the database, some database-level parameters can be configured there: page cache size, the size of RAM for sorting and several others, e.g., examples = D:\fbdata\3.0\examples.fdb

{

DefaultDbCachePages = 16K

TempCacheLimit = 512M

}

TIP

You can use an alias even before the database exists. It is valid to substitute the full file path with the alias in the CREATE DATABASE statement.

## Creating the Database Objects

Now let us create a script for building the database objects.

### Domains

First, we define some domains that we will use in column definitions.

**CREATE DOMAIN** D\_BOOLEAN **AS**

**SMALLINT**

**CHECK** (**VALUE IN** (0, 1));

**COMMENT ON DOMAIN** D\_BOOLEAN **IS**

'Boolean type. 0 - FALSE, 1- TRUE';

**CREATE DOMAIN** D\_MONEY **AS**

**NUMERIC**(15,2);

**CREATE DOMAIN** D\_ZIPCODE **AS**

**CHAR**(10) **CHARACTER SET** UTF8

**CHECK** (**TRIM**(**TRAILING FROM VALUE**) **SIMILAR TO** '[0-9]+');

**COMMENT ON DOMAIN** D\_ZIPCODE **IS**

'Zip code';

#### BOOLEAN Type

In Firebird 3.0, there is a native BOOLEAN type. Some drivers do not support it, due to its relatively recent appearance in Firebird's SQL lexicon. With that in mind,, our applications will be built on a database that will work with either Firebird 2.5 or Firebird 3.0. IMPORTANT

Before Firebird 3, servers could connect clients to databases that were created under older Firebird versions. Firebird 3 can connect them only to databases that were created on or restored under Firebird 3.

### Primary Tables

Now let us proceed to the primary tables. The first will be the CUSTOMER table. We will create a sequence (a generator) for its primary key and a corresponding trigger for implementing it as an auto-incrementing column. We will do the same for each of the tables.

**CREATE GENERATOR** GEN\_CUSTOMER\_ID;

**CREATE TABLE** CUSTOMER (

CUSTOMER\_ID **INTEGER NOT NULL**,

**NAME VARCHAR**(60) **NOT NULL**,

ADDRESS **VARCHAR**(250),

ZIPCODE D\_ZIPCODE,

PHONE **VARCHAR**(14),

**CONSTRAINT** PK\_CUSTOMER **PRIMARY KEY** (CUSTOMER\_ID)

);

**SET** TERM ^ ;

**CREATE OR ALTER TRIGGER** CUSTOMER\_BI **FOR** CUSTOMER

**ACTIVE BEFORE INSERT POSITION** 0

**AS**

**BEGIN**

**IF** (NEW.CUSTOMER\_ID **IS NULL**) **THEN**

NEW.CUSTOMER\_ID = **NEXT VALUE FOR** GEN\_CUSTOMER\_ID;

**END**

^

**SET** TERM ; ^

**COMMENT ON TABLE** CUSTOMER **IS**

'Customers';

**COMMENT ON COLUMN** CUSTOMER.CUSTOMER\_ID **IS**

'Customer Id';

**COMMENT ON COLUMN** CUSTOMER.**NAME IS**

'Name';

**COMMENT ON COLUMN** CUSTOMER.ADDRESS **IS**

'Address';

**COMMENT ON COLUMN** CUSTOMER.ZIPCODE **IS**

'Zip Code';

**COMMENT ON COLUMN** CUSTOMER.PHONE **IS**

'Phone';

|  |
| --- |
| **Note**   * In Firebird 3.0, you can use IDENTITY columns as auto-incremental fields. The script for creating the table would then be as follows:   **CREATE TABLE** CUSTOMER (  CUSTOMER\_ID **INTEGER GENERATED BY DEFAULT AS IDENTITY**,  **NAME VARCHAR**(60) **NOT NULL**,  ADDRESS **VARCHAR**(250),  ZIPCODE D\_ZIPCODE,  PHONE **VARCHAR**(14),  **CONSTRAINT** PK\_CUSTOMER **PRIMARY KEY** (CUSTOMER\_ID)  );     * In Firebird 3.0, you need the USAGE privilege to use a sequence (generator) so you will have to add the following line to the script:   **GRANT USAGE ON SEQUENCE** GEN\_CUSTOMER\_ID **TO TRIGGER** CUSTOMER\_BI; |

Now we construct a script for creating the PRODUCT table:

**CREATE GENERATOR** GEN\_PRODUCT\_ID;

**CREATE TABLE** PRODUCT (

PRODUCT\_ID **INTEGER NOT NULL**,

**NAME VARCHAR**(100) **NOT NULL**,

PRICE D\_MONEY **NOT NULL**,

DESCRIPTION **BLOB SUB\_TYPE** 1 **SEGMENT SIZE** 80,

**CONSTRAINT** PK\_PRODUCT **PRIMARY KEY** (PRODUCT\_ID)

);

**SET** TERM ^;

**CREATE OR ALTER TRIGGER** PRODUCT\_BI **FOR** PRODUCT

**ACTIVE BEFORE INSERT POSITION** 0

**AS**

**BEGIN**

**IF** (NEW.PRODUCT\_ID **IS NULL**) **THEN**

NEW.PRODUCT\_ID = **NEXT VALUE FOR** GEN\_PRODUCT\_ID;

**END**

^

**SET** TERM ;^

**COMMENT ON TABLE** PRODUCT **IS**

'Goods';

**COMMENT ON COLUMN** PRODUCT.PRODUCT\_ID **IS**

'Product Id';

**COMMENT ON COLUMN** PRODUCT.**NAME IS**

'Name';

**COMMENT ON COLUMN** PRODUCT.PRICE **IS**

'Price';

**COMMENT ON COLUMN** PRODUCT.DESCRIPTION **IS**

'Description';

|  |
| --- |
| **Note**  In Firebird 3.0, you need to add the command for granting the USAGE privilege for a sequence (generator) to the script:  **GRANT USAGE ON SEQUENCE** GEN\_PRODUCT\_ID **TO TRIGGER** PRODUCT\_BI; |

### Secondary Tables

The script for creating the INVOICE table:

**CREATE GENERATOR** GEN\_INVOICE\_ID;

**CREATE TABLE** INVOICE (

INVOICE\_ID **INTEGER NOT NULL**,

CUSTOMER\_ID **INTEGER NOT NULL**,

INVOICE\_DATE **TIMESTAMP**,

TOTAL\_SALE D\_MONEY,

PAID D\_BOOLEAN **DEFAULT** 0 **NOT NULL**,

**CONSTRAINT** PK\_INVOICE **PRIMARY KEY** (INVOICE\_ID)

);

**ALTER TABLE** INVOICE **ADD CONSTRAINT** FK\_INVOCE\_CUSTOMER

**FOREIGN KEY** (CUSTOMER\_ID) **REFERENCES** CUSTOMER (CUSTOMER\_ID);

**CREATE INDEX** INVOICE\_IDX\_DATE **ON** INVOICE (INVOICE\_DATE);

**SET** TERM ^;

**CREATE OR ALTER TRIGGER** INVOICE\_BI **FOR** INVOICE

**ACTIVE BEFORE INSERT POSITION** 0

**AS**

**BEGIN**

**IF** (NEW.INVOICE\_ID **IS NULL**) **THEN**

NEW.INVOICE\_ID = **GEN\_ID**(GEN\_INVOICE\_ID,1);

**END**

^

**SET** TERM ;^

**COMMENT ON TABLE** INVOICE **IS**

'Invoices';

**COMMENT ON COLUMN** INVOICE.INVOICE\_ID **IS**

'Invoice number';

**COMMENT ON COLUMN** INVOICE.CUSTOMER\_ID **IS**

'Customer Id';

**COMMENT ON COLUMN** INVOICE.INVOICE\_DATE **IS**

'The date of issuance invoices';

**COMMENT ON COLUMN** INVOICE.TOTAL\_SALE **IS**

'Total sum';

**COMMENT ON COLUMN** INVOICE.PAID **IS**

'Paid';

The INVOICE\_DATE column is indexed because we will be filtering invoices by date to enable the records to be selected by a work period that will be application-defined by a start date and an end date.

|  |
| --- |
| **Note**  In Firebird 3.0, you will need to add the command for granting the USAGE privilege for a sequence (generator) to the script:  **GRANT USAGE ON SEQUENCE** GEN\_INVOICE\_ID **TO TRIGGER** INVOICE\_BI; |

The script for creating the INVOICE\_LINE table:

**CREATE GENERATOR** GEN\_INVOICE\_LINE\_ID;

**CREATE TABLE** INVOICE\_LINE (

INVOICE\_LINE\_ID **INTEGER NOT NULL**,

INVOICE\_ID **INTEGER NOT NULL**,

PRODUCT\_ID **INTEGER NOT NULL**,

QUANTITY **NUMERIC**(15,0) **NOT NULL**,

SALE\_PRICE D\_MONEY **NOT NULL**,

**CONSTRAINT** PK\_INVOICE\_LINE **PRIMARY KEY** (INVOICE\_LINE\_ID)

);

**ALTER TABLE** INVOICE\_LINE **ADD CONSTRAINT** FK\_INVOICE\_LINE\_INVOICE

**FOREIGN KEY** (INVOICE\_ID) **REFERENCES** INVOICE (INVOICE\_ID);

**ALTER TABLE** INVOICE\_LINE **ADD CONSTRAINT** FK\_INVOICE\_LINE\_PRODUCT

**FOREIGN KEY** (PRODUCT\_ID) **REFERENCES** PRODUCT (PRODUCT\_ID);

**SET** TERM ^;

**CREATE OR ALTER TRIGGER** INVOICE\_LINE\_BI **FOR** INVOICE\_LINE

**ACTIVE BEFORE INSERT POSITION** 0

**AS**

**BEGIN**

**IF** (NEW.INVOICE\_LINE\_ID **IS NULL**) **THEN**

NEW.INVOICE\_LINE\_ID = **NEXT VALUE FOR** GEN\_INVOICE\_LINE\_ID;

**END**

^

**SET** TERM ;^

**COMMENT ON TABLE** INVOICE\_LINE **IS**

'Invoice lines';

**COMMENT ON COLUMN** INVOICE\_LINE.INVOICE\_LINE\_ID **IS**

'Invoice line Id';

**COMMENT ON COLUMN** INVOICE\_LINE.INVOICE\_ID **IS**

'Invoice number';

**COMMENT ON COLUMN** INVOICE\_LINE.PRODUCT\_ID **IS**

'Product Id';

**COMMENT ON COLUMN** INVOICE\_LINE.QUANTITY **IS**

'Quantity';

**COMMENT ON COLUMN** INVOICE\_LINE.SALE\_PRICE **IS**

'Price';

|  |
| --- |
| **Note**  In Firebird 3.0, you will need to add the command for granting the USAGE privilege for a sequence (generator) to the script:  **GRANT USAGE ON SEQUENCE** GEN\_INVOICE\_LINE\_ID **TO TRIGGER** INVOICE\_LINE\_BI; |

### Stored Procedures

Some parts of the business logic will be implemented by means of stored procedures.

#### Adding an invoice

The procedure for adding a new invoice is quite simple:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_ADD\_INVOICE (

INVOICE\_ID **INTEGER**,

CUSTOMER\_ID **INTEGER**,

INVOICE\_DATE **TIMESTAMP** = **CURRENT\_TIMESTAMP**)

**AS**

**BEGIN**

**INSERT INTO** INVOICE (

INVOICE\_ID,

CUSTOMER\_ID,

INVOICE\_DATE,

TOTAL\_SALE,

PAID

)

**VALUES** (

:INVOICE\_ID,

:CUSTOMER\_ID,

:INVOICE\_DATE,

0,

0

);

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_ADD\_INVOICE **IS**

'Adding Invoice';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE.INVOICE\_ID **IS**

'Invoice number';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE.CUSTOMER\_ID **IS**

'Customer Id';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE.INVOICE\_DATE **IS**

'Date';

**GRANT INSERT ON** INVOICE **TO PROCEDURE** SP\_ADD\_INVOICE;

#### Editing an invoice

The procedure for editing an invoice is a bit more complicated. We will include a rule to block further editing of an invoice once it is paid. We will create an exception that will be raised if an attempt is made to modify a paid invoice.

**CREATE EXCEPTION** E\_INVOICE\_ALREADY\_PAYED 'Change is impossible, invoice paid.';

The stored procedure for editing an invoice:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_EDIT\_INVOICE (

INVOICE\_ID **INTEGER**,

CUSTOMER\_ID **INTEGER**,

INVOICE\_DATE **TIMESTAMP**)

**AS**

**BEGIN**

**IF** (**EXISTS**(**SELECT** \*

**FROM** INVOICE

**WHERE** INVOICE\_ID = :INVOICE\_ID

**AND** PAID = 1)) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

**UPDATE** INVOICE

**SET** CUSTOMER\_ID = :CUSTOMER\_ID,

INVOICE\_DATE = :INVOICE\_DATE

**WHERE** INVOICE\_ID = :INVOICE\_ID;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_EDIT\_INVOICE **IS**

'Editing invoice';

**COMMENT ON PARAMETER** SP\_EDIT\_INVOICE.INVOICE\_ID **IS**

'Invoice number';

**COMMENT ON PARAMETER** SP\_EDIT\_INVOICE.CUSTOMER\_ID **IS**

'Customer Id';

**COMMENT ON PARAMETER** SP\_EDIT\_INVOICE.INVOICE\_DATE **IS**

'Date';

**GRANT SELECT**,**UPDATE ON** INVOICE **TO PROCEDURE** SP\_EDIT\_INVOICE;

|  |
| --- |
| **Note**  In Firebird 3.0, the USAGE privilege is required for exceptions so we need to add the following line:  **GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_EDIT\_INVOICE; |

#### Deleting an invoice

The procedure SP\_DELETE\_INVOICE checks whether the invoice is paid and raises an exception if it is:

**SET** TERM ^ ;

**CREATE OR ALTER PROCEDURE** SP\_DELETE\_INVOICE (

INVOICE\_ID **INTEGER**)

**AS**

**BEGIN**

**IF** (**EXISTS**(**SELECT** \* **FROM** INVOICE

**WHERE** INVOICE\_ID = :INVOICE\_ID

**AND** PAID = 1)) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

**DELETE FROM** INVOICE **WHERE** INVOICE\_ID = :INVOICE\_ID;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_DELETE\_INVOICE **IS**

'Deleting invoices';

**GRANT SELECT**,**DELETE ON** INVOICE **TO PROCEDURE** SP\_DELETE\_INVOICE;

|  |
| --- |
| **Note**  In Firebird 3.0, the USAGE privilege is required for exceptions so it is necessary to add the following line:  **GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_DELETE\_INVOICE; |

#### Paying an invoice

We will add one more procedure for paying an invoice:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_PAY\_FOR\_INVOICE (

INVOICE\_ID **INTEGER**)

**AS**

**BEGIN**

**IF** (**EXISTS**(**SELECT** \*

**FROM** INVOICE

**WHERE** INVOICE\_ID = :INVOICE\_ID

**AND** PAID = 1)) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

**UPDATE** INVOICE

**SET** PAID = 1

**WHERE** INVOICE\_ID = :INVOICE\_ID;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_PAY\_FOR\_INVOICE **IS**

'Payment of invoices';

**COMMENT ON PARAMETER** SP\_PAY\_FOR\_INVOICE.INVOICE\_ID **IS**

'Invoice number';

**GRANT SELECT**,**UPDATE ON** INVOICE **TO PROCEDURE** SP\_PAY\_FOR\_INVOICE;

|  |
| --- |
| **Note**  In Firebird 3.0, the USAGE privilege is required for exceptions so we need to add the following line:  **GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_PAY\_FOR\_INVOICE; |

#### Invoice Line Items

Procedures for managing invoice items will check whether the invoice is paid and block any attempt to alter the line items of paid invoices. They will also correct the invoice total according to the amount of the product sold and its price.

##### Adding an invoice line item

The procedure for adding a line item to an invoice:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_ADD\_INVOICE\_LINE (

INVOICE\_ID **INTEGER**,

PRODUCT\_ID **INTEGER**,

QUANTITY **INTEGER**)

**AS**

**DECLARE** sale\_price D\_MONEY;

**DECLARE** paid D\_BOOLEAN;

**BEGIN**

**SELECT**

paid

**FROM**

invoice

**WHERE**

invoice\_id = :invoice\_id

**INTO** :paid;

*-- It does not allow you to edit already paid invoice.*

**IF** (paid = 1) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

**SELECT**

price

**FROM**

product

**WHERE**

product\_id = :product\_id

**INTO** :sale\_price;

**INSERT INTO** invoice\_line (

invoice\_line\_id,

invoice\_id,

product\_id,

quantity,

sale\_price)

**VALUES** (

**NEXT VALUE FOR** gen\_invoice\_line\_id,

:invoice\_id,

:product\_id,

:quantity,

:sale\_price);

*-- Increase the amount of the account.*

**UPDATE** invoice

**SET** total\_sale = **COALESCE**(total\_sale, 0) + :sale\_price \* :quantity

**WHERE** invoice\_id = :invoice\_id;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_ADD\_INVOICE\_LINE **IS**

'Adding line invoices';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE\_LINE.INVOICE\_ID **IS**

'Invoice number';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE\_LINE.PRODUCT\_ID **IS**

'Product Id';

**COMMENT ON PARAMETER** SP\_ADD\_INVOICE\_LINE.QUANTITY **IS**

'Quantity';

**GRANT SELECT**, **UPDATE ON** INVOICE **TO PROCEDURE** SP\_ADD\_INVOICE\_LINE;

**GRANT SELECT ON** PRODUCT **TO PROCEDURE** SP\_ADD\_INVOICE\_LINE;

**GRANT INSERT ON** INVOICE\_LINE **TO PROCEDURE** SP\_ADD\_INVOICE\_LINE;

*-- only Firebird 3.0 and above*

**GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_ADD\_INVOICE\_LINE;

**GRANT USAGE ON SEQUENCE** GEN\_INVOICE\_LINE\_ID **TO PROCEDURE** SP\_ADD\_INVOICE\_LINE;

##### Editing an invoice line item

The procedure for editing an invoice line item:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_EDIT\_INVOICE\_LINE (

INVOICE\_LINE\_ID **INTEGER**,

QUANTITY **INTEGER**)

**AS**

**DECLARE** invoice\_id **INT**;

**DECLARE** price D\_MONEY;

**DECLARE** paid D\_BOOLEAN;

**BEGIN**

**SELECT**

product.price,

invoice.invoice\_id,

invoice.paid

**FROM**

invoice\_line

**JOIN** invoice **ON** invoice.invoice\_id = invoice\_line.invoice\_id

**JOIN** product **ON** product.product\_id = invoice\_line.product\_id

**WHERE**

invoice\_line.invoice\_line\_id = :invoice\_line\_id

**INTO**

:price,

:invoice\_id,

:paid;

*-- It does not allow you to edit an already paid invoice.*

**IF** (paid = 1) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

*-- Update price and quantity.*

**UPDATE** invoice\_line

**SET** sale\_price = :price,

quantity = :quantity

**WHERE** invoice\_line\_id = :invoice\_line\_id;

*-- Now update the amount of the account.*

**MERGE INTO** invoice

**USING** (

**SELECT**

invoice\_id,

**SUM**(sale\_price \* quantity) **AS** total\_sale

**FROM** invoice\_line

**WHERE** invoice\_id = :invoice\_id

**GROUP BY** invoice\_id) L

**ON** invoice.invoice\_id = L.invoice\_id

**WHEN MATCHED THEN**

**UPDATE SET** total\_sale = L.total\_sale;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_EDIT\_INVOICE\_LINE **IS**

'Editing invoice line';

**COMMENT ON PARAMETER** SP\_EDIT\_INVOICE\_LINE.INVOICE\_LINE\_ID **IS**

'Invoice line id';

**COMMENT ON PARAMETER** SP\_EDIT\_INVOICE\_LINE.QUANTITY **IS**

'Quantity';

**GRANT SELECT**,**UPDATE ON** INVOICE\_LINE **TO PROCEDURE** SP\_EDIT\_INVOICE\_LINE;

**GRANT SELECT**,**UPDATE ON** INVOICE **TO PROCEDURE** SP\_EDIT\_INVOICE\_LINE;

**GRANT SELECT ON** PRODUCT **TO PROCEDURE** SP\_EDIT\_INVOICE\_LINE;

*-- only Firebird 3.0 and above*

**GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_EDIT\_INVOICE\_LINE;

##### Deleting an invoice line item

The procedure for deleting an invoice line item from an invoice:

**SET** TERM ^;

**CREATE OR ALTER PROCEDURE** SP\_DELETE\_INVOICE\_LINE (

INVOICE\_LINE\_ID **INTEGER**)

**AS**

**DECLARE** invoice\_id **INT**;

**DECLARE** price D\_MONEY;

**DECLARE** quantity **INT**;

**BEGIN**

**IF** (**EXISTS**(**SELECT** \*

**FROM** invoice\_line

**JOIN** invoice **ON** invoice.invoice\_id = invoice\_line.invoice\_id

**WHERE** invoice.paid = 1

**AND** invoice\_line.invoice\_line\_id = :invoice\_line\_id)) **THEN**

**EXCEPTION** E\_INVOICE\_ALREADY\_PAYED;

**DELETE FROM** invoice\_line

**WHERE** invoice\_line.invoice\_line\_id = :invoice\_line\_id

**RETURNING** invoice\_id, quantity, sale\_price

**INTO** invoice\_id, quantity, price;

*-- Reduce the amount of the account.*

**UPDATE** invoice

**SET** total\_sale = total\_sale - :quantity \* :price

**WHERE** invoice\_id = :invoice\_id;

**END**

^

**SET** TERM ;^

**COMMENT ON PROCEDURE** SP\_DELETE\_INVOICE\_LINE **IS**

'Deleting invoice item';

**COMMENT ON PARAMETER** SP\_DELETE\_INVOICE\_LINE.INVOICE\_LINE\_ID **IS**

'Code invoice item';

#### Privileges for Procedures

**GRANT SELECT**,**DELETE ON** INVOICE\_LINE **TO PROCEDURE** SP\_DELETE\_INVOICE\_LINE;

**GRANT SELECT**,**UPDATE ON** INVOICE **TO PROCEDURE** SP\_DELETE\_INVOICE\_LINE;

*-- only Firebird 3.0 and above*

**GRANT USAGE ON EXCEPTION** E\_INVOICE\_ALREADY\_PAYED **TO PROCEDURE** SP\_DELETE\_INVOICE\_LINE;

#### Roles and Privileges for Users

Now we need to create roles and grant the corresponding privileges. We will create two roles: MANAGER and SUPERUSER. MANAGER will have a limited set of privileges while SUPERUSER will have access to practically everything in the database that is used by the project application.

**CREATE ROLE** MANAGER;

**CREATE ROLE** SUPERUSER;

The MANAGER role can read any table and use the corresponding procedures to manage invoices:

**GRANT SELECT ON** CUSTOMER **TO** MANAGER;

**GRANT SELECT ON** INVOICE **TO** MANAGER;

**GRANT SELECT ON** INVOICE\_LINE **TO** MANAGER;

**GRANT SELECT ON** PRODUCT **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_ADD\_INVOICE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_ADD\_INVOICE\_LINE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_DELETE\_INVOICE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_DELETE\_INVOICE\_LINE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_EDIT\_INVOICE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_EDIT\_INVOICE\_LINE **TO** MANAGER;

**GRANT EXECUTE ON PROCEDURE** SP\_PAY\_FOR\_INVOICE **TO** MANAGER;

**GRANT USAGE ON SEQUENCE** GEN\_INVOICE\_ID **TO** MANAGER;

The SUPERUSER role can read any table, edit the primary tables directly and use the procedures to manage invoices:

**GRANT SELECT**, **INSERT**, **UPDATE**, **DELETE ON** CUSTOMER **TO** SUPERUSER;

**GRANT SELECT ON** INVOICE **TO** SUPERUSER;

**GRANT SELECT ON** INVOICE\_LINE **TO** SUPERUSER;

**GRANT SELECT**, **INSERT**, **UPDATE**, **DELETE ON** PRODUCT **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_ADD\_INVOICE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_ADD\_INVOICE\_LINE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_DELETE\_INVOICE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_DELETE\_INVOICE\_LINE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_EDIT\_INVOICE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_EDIT\_INVOICE\_LINE **TO** SUPERUSER;

**GRANT EXECUTE ON PROCEDURE** SP\_PAY\_FOR\_INVICE **TO** SUPERUSER;

**GRANT USAGE ON SEQUENCE** GEN\_CUSTOMER\_ID **TO** SUPERUSER;

**GRANT USAGE ON SEQUENCE** GEN\_INVOICE\_ID **TO** SUPERUSER;

**GRANT USAGE ON SEQUENCE** GEN\_PRODUCT\_ID **TO** SUPERUSER;

These statements create some users and assign roles to them:

**CREATE USER** IVAN **PASSWORD** 'z12a';

**CREATE USER** ANNA **PASSWORD** 'lh67';

**GRANT** MANAGER **TO** ANNA;

**GRANT** MANAGER **TO** IVAN **WITH ADMIN OPTION**;

**GRANT** SUPERUSER **TO** IVAN;

The user IVAN can assign the MANAGER role to other users.

## Saving and Running the Script

Save our script to a text file named *examples.sql*.

Now you have three choices: you can

* download the ready-made script files using the following links: <https://github.com/sim1984/example-db_2_5/archive/1.0.zip> and <https://github.com/sim1984/example-db_3_0/archive/1.0.zip>;   
  or
* run the script *examples.sql* that you just created yourself;  
  or
* download the ready-made database, complete with sample data. Links are provided at the end of this chapter.

Now, to run our script in the database created earlier:

isql -user sysdba -password masterkey "localhost:examples" -i "d:\examples-db\examples.sql"

The argument "localhost:examples" uses an alias in place of the file path. It assumes that an alias named 'examples' actually exists, of course! The -i switch is an abbreviation of -input and its argument should be the path to the script file you just saved.

### Loading Test Data

Now that the database is created and built, you can populate it with test data. Various tools are available to help with that. If you prefer not to do it yourself, you can download a copy of the built database already loaded with the test data we used in the sample projects, from one of the following links:

<https://github.com/sim1984/example-db_2_5/releases/download/1.0/examples.fdb>

or

<https://github.com/sim1984/example-db_3_0/releases/download/1.0/examples.fdb>

Reminder

A database built by Firebird 2.5 will not be accessible by a Firebird 3 server, nor vice versa. Make sure you download the correct database for your needs.