

# PSQL in Action!



**[www.FireBase.com.br](http://www.FireBase.com.br)**

**Carlos H. Cantu (2005)**

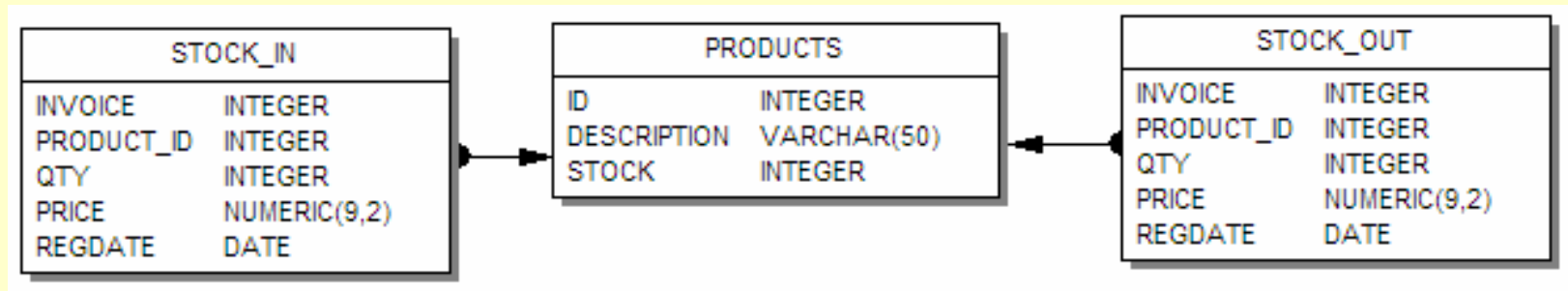


## Types of Stored Procedures

- **Executable Procedure**: Called by **Execute Procedure** – returns nothing or maximum of one row;
- **Select Stored Procedure**: Used as a data source for **SELECT** statements. Returns a dataset of one or more rows, or empty.

## Checking stock quantities and reporting bad products

- **DataBase structure:**



- **Procedure work:**

**Select Stored Proc** that at the same time will check for incorrect stock quantities, correct it (when possible), and returns a result set with each product status (OK or BAD);

## Why is it interesting?

- A single procedure will validate the data, correct it and return the status of the products;
- Demonstrates that you can use **select stored procedures** to modify (manipulate) records in tables, along with returning a result set;

- **What is it?**

**Procedure call itself many times.**

- **How many times?**

**Windows: up to 750 times**

**Linux: up to 1,000 times**

## Recursive Procedures

- **What is in the example?**

*A computer is build from many parts: HD, motherboard, CPU, monitor, printer, etc.*

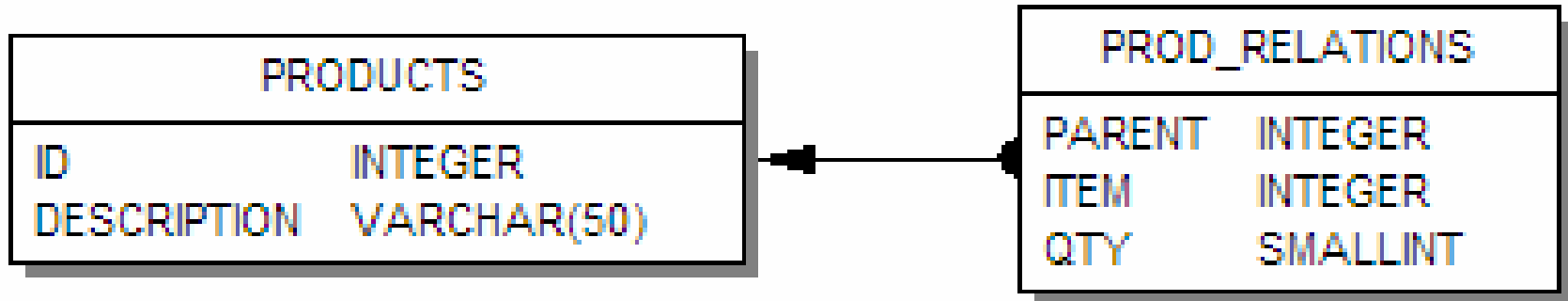


The example shows a recursive stored procedure that will receive two parameters:

**Product ID** and **quantity value**

**It will run recursively trough all the parts that compose the product and at the end will list all the parts and quantities needed to build it.**

# DataBase structure:



Very basic... Let's keep things simple 😊

- **DataBase structure:**

| INCOMES        |              |
|----------------|--------------|
| ID_NUM         | INTEGER      |
| EMISSION       | DATE         |
| EXPIRATION     | DATE         |
| RECEIVED       | DATE         |
| DESCRIPTION    | VARCHAR(80)  |
| TOTAL          | NUMERIC(9,2) |
| RECEIVED_VALUE | NUMERIC(9,2) |

| PAYMENTS    |              |
|-------------|--------------|
| ID_NUM      | INTEGER      |
| EMISSION    | DATE         |
| EXPIRATION  | DATE         |
| PAID        | DATE         |
| DESCRIPTION | VARCHAR(80)  |
| TOTAL       | NUMERIC(9,2) |
| PAID_VALUE  | NUMERIC(9,2) |

- ***Select Stored Procedure*** will merge data from both tables, calculating the ***starting balance*** and ***final balance***.



### Tips:

- If a record is **not** paid/received, the procedure should assume the expiration date, **otherwise** it must use the effective (paid/received) date.
- If a paid/received date is present, it **must use** the *paid\_value/**received\_value* to build the balance;
- The starting balance should **never** show a negative number 😊

### **Problem:**

**Sometimes users are browsing data on the screen and the displayed data is not valid anymore (too old) due to changes to the dataset done by other users in the network, or even by triggers and procedures that were run after the data was buffered in memory by the Delphi application.**

### **Solution:**

Use Firebird **EVENTS** to implement an auto-update feature where the user will be notified when some data changes on the underline table/dataset being displayed.

*PS: Firebird events are very **light** regarding resource usage and network traffic.*

### How to do this?

- Use triggers to **post events** to the applications notifying about changes in the data.
- Only necessary tables may use the auto-update mechanism.
- Delphi application will capture the event and decide if it needs to “refresh” the buffered data or not.

### Auxiliary structures:

- An auxiliary table that will store “data change notifications” with the *Connection ID* and *TimeStamp* that it was done.
- One Stored Procedure that will help to feed the auxiliary table with the required information.

### Auxiliary table structure:

```
CREATE TABLE EVENTS (  
    ID                INTEGER NOT NULL,  
    DATETIME          TIMESTAMP NOT NULL,  
    CONNECTION        BIGINT NOT NULL,  
    OPERATION         VARCHAR(10),  
    WHATTABLE         VARCHAR(32) NOT NULL  
);
```

### Problems? *Maybe!*

***As data refresh usually means a *refetch* of the data from the server to the client, the application must take care to not saturate the server with many fetches, specially if too many lines are being retrieved.***

### How to do that?

- Set a **minimum interval** between refreshes.
- Instead of refetch the data, only **tell** user that the displayed data have changed, and let him decide to refresh it (pressing the refresh button) or not.
- Take care of **not** considering the changes done by the user connection itself.
- **A MIX of the above.**



# Questions?

[cantu@firebase.com.br](mailto:cantu@firebase.com.br)

[www.firebase.com.br](http://www.firebase.com.br)