Using SQL to access a REST service requires a little extra syntax. Pyrrho allows a modified CREATE VIEW syntax called RESTView:

CREATE VIEW id OF rowtype AS GET url

where rowtype is like a table definition, giving column names and domains. The url part can be a single-quoted string, but syntactically it is just Pyrrho-specific Metadata and various flags and additional items are possible for specifying mime types etc. Full details are in the Pyrrho manual, and we will see some examples in this demo.

To support transactions, we base our implementation on the use of entity-tags as in RFC 7232 (Fielding & Reschke, 2014). Very few actual REST services comply with this standard, so Pyrrho provides a suitable service. Then, as we will see, a REST round-trip is a branch transaction, that occurs at the main transaction commit point. The motivation for this example is the classic example of a transfer between two bank accounts, but here it is between two different banks, each with their own databases and need for serialized transactions. Communication between the banks will be over the Internet.

This time the PyrrhoSvr needs to be started up with the command line shown to start the HTTP service (by default at localhost:8180) and provide some diagnostic feedback (type this line into the console window: do not paste from this document in case the minus signs are replaced by dashes):

PyrrhoSvr -d:\DATA +s -H

Names of banks and accounts are kept short to fit on the presentation slides. In the blue window, give the command

PyrhoCmd DB
to create the database for bank B. At the SQL> prompt set up an accounts table for bank B with the SQL statements

```sql
create table AB (id int, bal int)
insert into AB values (100,1000),(110,1400)
```

This gives us two bank accounts id=100 with balance 1000, and 110 with balance 1400. To enable access to this database over the network, at least one role and one user are required:

```sql
create role DB
grant DB to "machine\user"
```

where the current user identity is inserted surrounded by straight double-quotes. (You can find what this is with the command select user.) And we define a RESTView for bank C to use:

```sql
create view RC of (id int,bal int) as get etag milli 'http://localhost:8180/DC/DC/AC'
```

The url is enclosed in single-quotes as specified above, and the two metadata flags tell Pyrrho to enforce RFC 7232 with millisecond precision.

Do the same, mutatis mutandis, for bank C in the green window:

PyrrhoCmd DC

```sql
create table AC (id int, bal int)
insert into AC values (200,2000),(220,1800)
create role DC
grant DC to "machine\user"
```

```sql
create view RC of (id int,bal int) as get etag milli 'http://localhost:8180/DC/DC/AC'
```

The url is enclosed in single-quotes as specified above, and the two metadata flags tell Pyrrho to enforce RFC 7232 with millisecond precision.
create view RB of (id int, bal int) as get etag milli 'http://localhost:8180/DB/DB/AB'

This gives us two bank accounts id=200 with balance 2000, and 220 with balance 1800.

We refer to these windows as blue and green for clarity. We now have the following situation:

The current state of the account in the two banks can be verified by table AB in the blue window and table AC in the green window.

Two overlapping transactions will conflict on account 200. In this slide, bank B starts a transaction and takes £50 from account 100 to be transferred. Nothing is written to disk yet.

begin transaction

update AB set bal=bal-50 where id=100
The report on the update makes the point that the new record is in the transaction, rather than the database.

[163 @ 1:02:12]

In the green window, someone in bank C starts a transaction to transfer money from account 200 to remote account 110.

```
begin transaction
update AC set bal=bal-30 where id=200
```

[164 @ 1:02:38]

Back in the blue window, bank C is taking the next step in its transaction, adding the £50 to remote account 200.

```
update RC set bal=bal+50 where id=200
```

This is still in the transaction, and nothing has been committed. The response on bank B indicates that the changes are not for the local bank.
But we can see in the server window that there has been some Internet activity. There are two round trips. First, bank B checks that 200 is a valid account number for bank C. Colour-coding to distinguish output from the blue and green server threads,

RoundTrip HEAD http://localhost:8180/DC/DC/AC
HTTP HEAD /DC/DC/AC
Received If-Unmodified-Since: Wed, 26 Oct 2022 10:10:10.066 GMT
Returning ETag: 
-->

Next, it gets an ETag for the current balance in account 200:

http://localhost:8180/DC/DC select ID,BAL from AC where (ID=200) and ID=200
HTTP POST /DC/DC
select ID,BAL from AC where (ID=200) and ID=200
Received If-Unmodified-Since: Wed, 26 Oct 2022 10:10:10.066 GMT
Returning ETag: "23,96,96"
-->
1 rows
Response ETag: 23,96,96

Bank B has noted this ETag.

[165 @ 1:02:52]

The green teller also starts a transfer, of their £30 from account 200, to account 110,

update RB set bal=bal+30 where id=110

and receives an ETag for remote account 110.
At this point both tellers are about to commit their transaction. The first to do so will succeed, but no changes have been made yet.

[166 @ 1:03:38]

The green teller gets there first, and commits their transaction.

commit

Looking at the feedback in the server window:

HTTP HEAD /DB/DB
Received If-Match: 23,114,114
Returning ETag:

RoundTrip POST http://localhost:8180/DB/DB update AB set BAL=(BAL+30) where (ID=110) and ID=110

Looking at the feedback in the server window:

HTTP HEAD /DB/DB
Received If-Match: 23,114,114
Returning ETag:

RoundTrip POST http://localhost:8180/DB/DB update AB set BAL=(BAL+30) where (ID=110) and ID=110

HTTP POST /DB/DB
update AB set BAL=(BAL+30) where (ID=110) and ID=110
Received ETag:
Returning ETag:

RoundTrip POST http://localhost:8180/DB/DB update AB set BAL=(BAL+30) where (ID=110) and ID=110

HTTP HEAD /DB/DB
Received If-Match: 23,114,114
Returning ETag:

RoundTrip POST http://localhost:8180/DB/DB update AB set BAL=(BAL+30) where (ID=110) and ID=110

I records in transaction DC
0 records in transaction AB
1 records affected in DC
HTTP POST /DB/DB
update AB set BAL=(BAL+30) where (ID=110) and ID=110

Received If-Match: 339
Received If-Unmodified-Since: Wed, 26 Oct 2022 09:10:24.835 GMT
Returning ETag "23,114,355"
---> OK

The first step is a HEAD request to verify the ETag sent to the green database is still valid. (It is.) Then the update is POSTed to bank B.

On receiving OK from the remote bank B, the change is committed to local account 200 in bank C. This will invalidate the ETag previously sent to the blue client.

The response “1 records affected” to the commit indicates that the change to local account 200 is now recorded durably in the database.

[167 @ 1:04:20]

Sure enough, when the blue client attempts to commit,

commit

and sends its change to bank C, bank C reports that the ETag is invalid, and the transaction is aborted. The failure of this HEAD roundtrip rolls back the blue transaction entirely.

[168 @ 1:04:53]

The teller in bank B can verify that the transfer from account 100 has not occurred, while bank C’s transaction transferring £30 has been committed in both databases.

[169 @ 11:05:11]
[For further examples, see section 6.10 in the SourceIntro document in the distribution.]