Cryptocurrency Trading Platform

Machida Hiroaki
himachid@bu.edu

MET CS579 A1 Database Management (2019 Fall)

Table of Contents

Chapter 1 Introduction

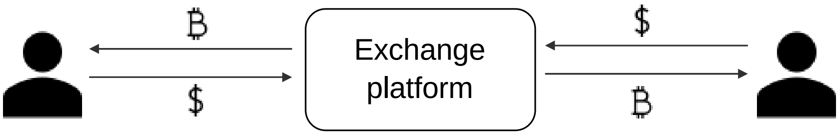
Project summary:

The project summary is as follows.

#	Item	Detail
1	Name of the project	Cryptocurrency Trading Platform
2	Number of database components	Tables(entities): 5 (8) Forms: 3 Queries: 5
3	Brief plan for completing the project	1. Project idea approval - 9/30 2. E-R diagram approval - 10/15 3. Relational model approval - 10/31 4. Database and application implementation - 11/30 5. Formal report and brief presentation - 12/9

Business:

A cryptocurrency exchange platform that users can exchange money for cryptocurrency, and vice versa.



Problem and Opportunity:

The problem is that people who want cryptocurrency need to find someone who want to sell cryptocurrency. The opportunity of this application is that users don't need to find someone who sells cryptocurrency, but just need to place orders.

Use case:

The general use case is as follows.

#	Use case	Information needs
1	A user opens an account.	Account
2	The user deposits money.	Transaction (Deposit), Balance, Currency
3	The user places an order.	Order, Currency
4	The order is settled.	Transaction (Settlement), Balance, Currency
5	The user withdraws cryptocurrency.	Transaction (Withdraw), Balance, Currency

Figure 1. Use case

Chapter 2 Simple E-R Diagram

User creates an account and balances by currency. The user places an order. When the user deposit, withdraw or the orders are settled, transactions are created.

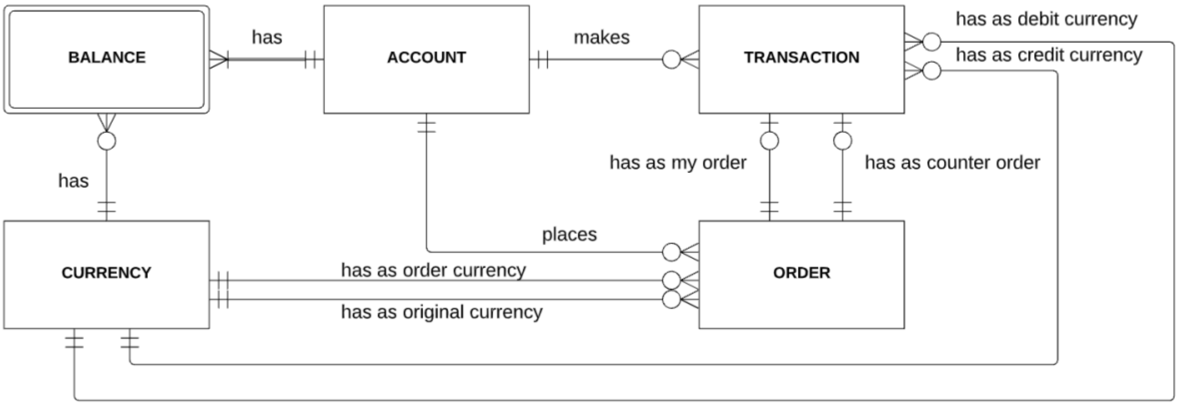


Figure 2. Simple E-R Diagram

Chapter 3 E-R Diagram with attributes and identifiers

Transaction is a super type of deposit, settlement, and withdrawal.

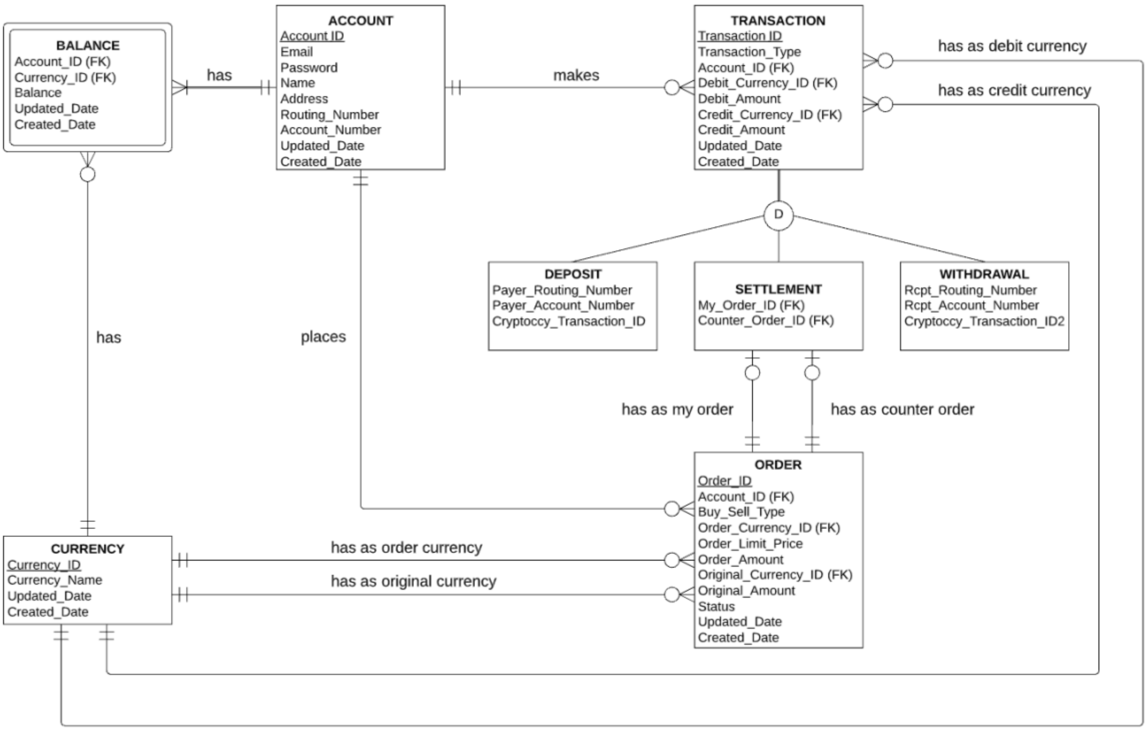


Figure 3. E-R Diagram with attributes and identifiers

Chapter 4 Normalization

The collection of normalized relations and functional dependencies:
There were no needs for normalization because functional dependencies or transitive dependencies are already separated into different tables.

Brief discussion as to the normal form(s) achieved:
The structure of database is the third normal form because no repeating columns (the first normal form), no functional dependencies (the second normal form), and no transitive dependencies (the third normal form).

Methods used to achieve these normal forms, and reasons why any de-normalization was done:
There was no need to change the structure since it was already the third normal form.

Chapter 5 Metadata for tables

The metadata for tables is shown below.
Each table has one primary key, except balance. The balance's keys are primary keys of account and currency.
Each table, except account and currency, has foreign keys accordingly.
No other constraints, such as index, are defined since not required.

Metadata for tables

```
CREATE TABLE "ACCOUNT"
(       "ACCOUNT_ID" VARCHAR2(20) NOT NULL,
        "EMAIL" VARCHAR2(200) NOT NULL,
        "PASSWORD" VARCHAR2(30) NOT NULL,
        "NAME" VARCHAR2(100) NOT NULL,
        "ADDRESS" VARCHAR2(200) NOT NULL,
        "ROUTING_NUMBER" VARCHAR2(9) NOT NULL,
        "ACCOUNT_NUMBER" VARCHAR2(12) NOT NULL,
        "UPDATED_DATE" DATE NOT NULL,
        "CREATED_DATE" DATE NOT NULL,
         CONSTRAINT "ACCOUNT_PK" PRIMARY KEY ("ACCOUNT_ID")
);

CREATE TABLE "CURRENCY"
(       "CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "CURRENCY_NAME" VARCHAR2(20) NOT NULL,
        "UPDATED_DATE" DATE NOT NULL,
        "CREATED_DATE" DATE NOT NULL,
         CONSTRAINT "CURRENCY_PK" PRIMARY KEY ("CURRENCY_ID")
);

CREATE TABLE "BALANCE"
(       "ACCOUNT_ID" VARCHAR2(20) NOT NULL,
        "CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "BALANCE" NUMBER NOT NULL,
        "UPDATED_DATE" DATE NOT NULL,
        "CREATED_DATE" DATE NOT NULL,
         CONSTRAINT "BALANCE_FK1" FOREIGN KEY ("ACCOUNT_ID")
          REFERENCES "ACCOUNT" ("ACCOUNT_ID"),
         CONSTRAINT "BALANCE_FK2" FOREIGN KEY ("CURRENCY_ID")
          REFERENCES "CURRENCY" ("CURRENCY_ID")
) ;

CREATE TABLE "ORDER_"
(       "ORDER_ID" VARCHAR2(20) NOT NULL,
        "ACCOUNT_ID" VARCHAR2(20) NOT NULL,
        "BUY_SELL_TYPE" VARCHAR2(4) NOT NULL,
        "ORDER_CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "ORDER_LIMIT_PRICE" NUMBER NOT NULL,
        "ORDER_AMOUNT" NUMBER NOT NULL,
        "ORIGINAL_CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "ORIGINAL_AMOUNT" NUMBER NOT NULL,
        "STATUS" VARCHAR2(10) NOT NULL,
        "UPDATED_DATE" DATE NOT NULL,
        "CREATED_DATE" DATE NOT NULL,
         CONSTRAINT "ORDER_PK" PRIMARY KEY ("ORDER_ID"),
         CONSTRAINT "ORDER__FK1" FOREIGN KEY ("ACCOUNT_ID")
          REFERENCES "ACCOUNT" ("ACCOUNT_ID"),
         CONSTRAINT "ORDER__FK2" FOREIGN KEY ("ORDER_CURRENCY_ID")
          REFERENCES "CURRENCY" ("CURRENCY_ID"),
         CONSTRAINT "ORDER__FK3" FOREIGN KEY ("ORIGINAL_CURRENCY_ID")
          REFERENCES "CURRENCY" ("CURRENCY_ID")
);

CREATE TABLE "TRANSACTION"
(        "TRANSACTION_ID" VARCHAR2(20) NOT NULL,
        "TRANSACTION_TYPE" VARCHAR2(10) NOT NULL,
        "ACCOUNT_ID" VARCHAR2(20) NOT NULL,
        "DEBIT_CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "DEBIT_AMOUNT" NUMBER NOT NULL,
        "CREDIT_CURRENCY_ID" VARCHAR2(20) NOT NULL,
        "CREDIT_AMOUNT" NUMBER NOT NULL,
        "UPDATED_DATE" DATE NOT NULL,
        "CREATED_DATE" DATE NOT NULL,
        "PAYER_ROUTING_NUMBER" VARCHAR2(9),
        "PAYER_ACCOUNT_NUMBER" VARCHAR2(12),
        "CRYPTOCCY_TRANSACTION_ID" VARCHAR2(34),
        "MY_ORDER_ID" VARCHAR2(20),
        "COUNTER_ORDER_ID" VARCHAR2(20),
        "RCPT_ROUTING_NUMBER" VARCHAR2(9),
        "RCPT_ACCOUNT_NUMBER" VARCHAR2(12),
        "CRYPTOCCY_TRANSACTION_ID2" VARCHAR2(64),
         CONSTRAINT "TRANSACTION_PK" PRIMARY KEY ("TRANSACTION_ID"),
         CONSTRAINT "TRANSACTION_FK1" FOREIGN KEY ("ACCOUNT_ID")
          REFERENCES "ACCOUNT" ("ACCOUNT_ID"),
         CONSTRAINT "TRANSACTION_FK2" FOREIGN KEY ("DEBIT_CURRENCY_ID")
          REFERENCES "CURRENCY" ("CURRENCY_ID"),
         CONSTRAINT "TRANSACTION_FK3" FOREIGN KEY ("CREDIT_CURRENCY_ID")
          REFERENCES "CURRENCY" ("CURRENCY_ID"),
         CONSTRAINT "TRANSACTION_FK4" FOREIGN KEY ("MY_ORDER_ID")
          REFERENCES "ORDER_" ("ORDER_ID"),
         CONSTRAINT "TRANSACTION_FK5" FOREIGN KEY ("COUNTER_ORDER_ID")
          REFERENCES "ORDER_" ("ORDER_ID")
);
```

Figure 4. create.sql

Chapter 6 Example printouts

The use cases below and additional use case for stored procedure are executed. Front-end is no implemented, so expected forms are shown, besides the queries and results.

#	Use case	Information needs
1	A user opens an account.	Account
2	The user deposits money.	Transaction (Deposit), Balance, Currency
3	The user places an order.	Order, Currency
4	The order is settled.	Transaction (Settlement), Balance, Currency
5	The user withdraws cryptocurrency.	Transaction (Withdraw), Balance, Currency

Figure 5. Use case

1. A user opens an account:

Create an account and balances by each currency.

Form

Item	Field
Email	Ex. spasfield0@scientificamerican.com
Password	Ex. IhDJXC6qqS
Name	Ex. Selia Pasfield
Address	Ex. 209 Lighthouse Bay Hill
Routing Number	Ex. 270090287
Account Number	Ex. 742340997414

Query

```
--A user opens an account.
INSERT INTO ACCOUNT VALUES('21', 'uaubri0@cyberchimps.com', 'NLHokv0',
'Ursuline Aubri', '9866 Ridgeview Junction', '236105411', '341000165174', sysdate,
sysdate);
INSERT INTO BALANCE VALUES('21', '1' ,0, sysdate, sysdate);
INSERT INTO BALANCE VALUES('21', '2' ,0, sysdate, sysdate);
INSERT INTO BALANCE VALUES('21', '3' ,0, sysdate, sysdate);
COMMIT;
```

Result

```
1 row inserted.


1 row inserted.
```

```
1 row inserted.


1 row inserted.


Commit complete.
```

2. The user deposits money:

Get a current balance, create a transaction and update the current balance.

Query

```
--The user deposits money.
SELECT * FROM BALANCE WHERE ACCOUNT_ID = '21' and (CURRENCY_ID = '1' OR
CURRENCY_ID = '2');
INSERT INTO TRANSACTION VALUES('11', 'DEPOSIT', '21', '2', 0, '2', 10000, sysdate,
sysdate, '236105411', '341000165174', null, null, null, null, null, null);
UPDATE BALANCE SET BALANCE=10000, UPDATED_DATE = sysdate WHERE
ACCOUNT_ID = '21' and CURRENCY_ID = '2';
COMMIT;
```

Result

```
ACCOUNT_ID          CURRENCY_ID            BALANCE UPDATED_ CREATED_
-------------------- -------------------- ---------- -------- --------
21                  1                           0 19-11-26 19-11-26
21                  2                           0 19-11-26 19-11-26


1 row inserted.


1 row updated.


Commit complete.
```

3. The user places an order:

Get the current balance and create an order.

Form

Item	Field
Email	Ex. spasfield0@scientificamerican.com
Password	Ex. IhDJXC6qqS
Name	Ex. Selia Pasfield
Address	Ex. 209 Lighthouse Bay Hill
Routing Number	Ex. 270090287

Account Number	Ex. 742340997414

Query

```
--The user places an order.
SELECT * FROM ORDER_ WHERE ORDER_CURRENCY_ID = '1' and
ORIGINAL_CURRENCY_ID = '2' and STATUS = 'ACTIVE';
INSERT INTO ORDER_ VALUES('12', '21', 'BUY', '1', 7100, 0.1, '2', 710, 'ACTIVE',
sysdate, sysdate);
COMMIT;
```

Result

```
ORDER_ID          ACCOUNT_ID        BUY_ ORDER_CURRENCY_ID
ORDER_LIMIT_PRICE ORDER_AMOUNT ORIGINAL_CURRENCY_ID
ORIGINAL_AMOUNT STATUS     UPDATED_ CREATED_
------------------- ------------------- ---- ------------------- ----------------- ------------ ---------------
----- --------------- ---------- -------- --------
1              2                SELL 1                    7200         2 2
14400 ACTIVE    19-11-26 19-11-26
2              4                SELL 1                    7150         3 2
21450 ACTIVE    19-11-26 19-11-26
3              6                SELL 1                    7130         1 2
7130 ACTIVE    19-11-26 19-11-26
4              8                BUY  1                    7050         2 2
14100 ACTIVE    19-11-26 19-11-26
5              10               BUY  1                    7000         10 2
70000 ACTIVE    19-11-26 19-11-26


1 row inserted.


Commit complete.
```

4. The order is settled:

Another order placed, mark orders as settled, create transactions and update balances.

Query

```
--The order is settled.
SELECT * FROM ORDER_ WHERE ORDER_CURRENCY_ID = '1' and
ORIGINAL_CURRENCY_ID = '2' and STATUS = 'ACTIVE';
INSERT INTO ORDER_ VALUES('13', '1', 'SELL', '1', 7100, 0.1, '2', 710, 'SETTLED',
sysdate, sysdate);
UPDATE ORDER_ SET STATUS = 'SETTLED', UPDATED_DATE = sysdate WHERE
ORDER_ID = '12';
INSERT INTO TRANSACTION VALUES('12', 'SETTLEMENT', '21', '2', 7100, '1', 0.1,
sysdate, sysdate, null, null, null, '12', '13', null, null, null);
```

```
INSERT INTO TRANSACTION VALUES('13', 'SETTLEMENT', '1', '1', 0.1, '2', 7100,
sysdate, sysdate, null, null, null, '13', '12', null, null, null);
SELECT * FROM BALANCE WHERE ACCOUNT_ID = '21' and (CURRENCY_ID = '1' OR
CURRENCY_ID = '2');
SELECT * FROM BALANCE WHERE ACCOUNT_ID = '1' and (CURRENCY_ID = '1' OR
CURRENCY_ID = '2');
UPDATE BALANCE SET BALANCE = 0.1, UPDATED_DATE = sysdate WHERE
ACCOUNT_ID = '21' and CURRENCY_ID = '1';
UPDATE BALANCE SET BALANCE = 2900, UPDATED_DATE = sysdate WHERE
ACCOUNT_ID = '21' and CURRENCY_ID = '2';
UPDATE BALANCE SET BALANCE = 666124.9, UPDATED_DATE = sysdate WHERE
ACCOUNT_ID = '1' and CURRENCY_ID = '1';
UPDATE BALANCE SET BALANCE = 913351, UPDATED_DATE = sysdate WHERE
ACCOUNT_ID = '1' and CURRENCY_ID = '2';
COMMIT;
```

Result

```
ORDER_ID            ACCOUNT_ID          BUY_ ORDER_CURRENCY_ID
ORDER_LIMIT_PRICE ORDER_AMOUNT ORIGINAL_CURRENCY_ID
ORIGINAL_AMOUNT STATUS     UPDATED_ CREATED_
------------------- ------------------- ---- ------------------- ---------------- ----------- ---------------
----- --------------- ---------- -------- --------
1                   2                   SELL 1                   7200        2 2
14400 ACTIVE     19-11-26 19-11-26
2                   4                   SELL 1                   7150        3 2
21450 ACTIVE     19-11-26 19-11-26
3                   6                   SELL 1                   7130        1 2
7130 ACTIVE     19-11-26 19-11-26
4                   8                   BUY  1                   7050        2 2
14100 ACTIVE     19-11-26 19-11-26
5                   10                  BUY  1                   7000        10 2
70000 ACTIVE     19-11-26 19-11-26
12                  21                  BUY  1                   7100        .1 2
710 ACTIVE     19-11-26 19-11-26

6 rows selected.


1 row inserted.


1 row updated.


1 row inserted.
```

```
1 row inserted.


ACCOUNT_ID          CURRENCY_ID            BALANCE UPDATED_ CREATED_
-------------------- -------------------- ---------- -------- --------
21                   1                       0 19-11-26 19-11-26
21                   2                   10000 19-11-26 19-11-26


ACCOUNT_ID          CURRENCY_ID            BALANCE UPDATED_ CREATED_
-------------------- -------------------- ---------- -------- --------
1                    1                  666125 19-11-26 19-11-26
1                    2                  920451 19-11-26 19-11-26


1 row updated.


1 row updated.


1 row updated.


1 row updated.


Commit complete.
```

5. The user withdraws cryptocurrency:
Get the current balance, update it and create a transaction.
Form

Item	Field
Currency Name	Ex. BTC
Address	Ex. 13KQ7EmvXyoxfNZ5YWTUtLotGiuk7DeEDU
Amount	Ex. 1

Query

```
--The user withdraws cryptocurrency.
SELECT * FROM BALANCE WHERE ACCOUNT_ID = '21' and CURRENCY_ID = '1';
UPDATE BALANCE SET BALANCE = 0, UPDATED_DATE = sysdate WHERE ACCOUNT_ID
= '21' and CURRENCY_ID = '1';
INSERT INTO TRANSACTION VALUES('14', 'WITHDRAW', '21', '1', 0.1, '1', 0, sysdate,
sysdate, null, null, null, null, null, null, null, '18MnkkPLjQZJiZvQbjcUbrx56LCkRyXFXP');
COMMIT;
```

Result

```
ACCOUNT_ID         CURRENCY_ID           BALANCE UPDATED_ CREATED_
------------------- ------------------- ---------- -------- --------
21                 1                         .1 19-11-26 19-11-26


1 row updated.


1 row inserted.


Commit complete.
```

6. (Additional use case) Add bitcoin to every user:
Get every user's balance as a cursor, then update the balance.
Stored Procedure

```
create or replace PROCEDURE bitcoin_campaign (btc IN NUMBER, r OUT VARCHAR2)
IS
  CURSOR c1 IS SELECT ACCOUNT_ID, BALANCE FROM BALANCE WHERE
CURRENCY_ID = '1';
BEGIN
  DBMS_OUTPUT.PUT_LINE('..start..');
  FOR rec IN c1 LOOP
    UPDATE BALANCE SET BALANCE = rec.BALANCE + btc, UPDATED_DATE = sysdate
WHERE ACCOUNT_ID = rec.ACCOUNT_ID and CURRENCY_ID = '1';
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('..end..');
  r := 'OK';
EXCEPTION
  WHEN others THEN
    DBMS_OUTPUT.PUT_LINE('..error..');
        r := 'NG';
END
;
```

Query

```
DECLARE
  BTC NUMBER;
  R VARCHAR2(200);
BEGIN
  BTC := 00.1;

  BITCOIN_CAMPAIGN(
    BTC => BTC,
    R => R
  );
  /* Legacy output:
DBMS_OUTPUT.PUT_LINE('R = ' || R);
*/
  :R := R;
--rollback;
END;
```

Result

```
Connecting to the database project.
..start..
..end..
Process exited.
Disconnecting from the database project.
```

Chapter 7 Conclusion

Making a procedure from scratch was new to me.

a) your experience with the project
Which steps were the most difficult?
Creating initial data is the most difficult because of its volume. The second is the procedure because I have not created it from scratch and am not familiar with the grammar.

Which were the easiest?
Come up with an idea.

What did you learn that you did not imagine you would have?
I did not imagine that I needed to do try and error on making procedure because I have some experience in PL/SQL. It turned out that I am familiar only with business logic, not PL/SQL itself.

If you had to do it all over again, what would you have done differently?
Add user session entity.

b) If the proposed benefits can be realized by the new system
The problem is that people who want cryptocurrency need to find someone who want to sell cryptocurrency. The benefit of this application is that users don't need to find someone who sells cryptocurrency, but just need to place orders. The benefit can be realized if this service is released.

c) any final comments and conclusions
Setting up the oracle SQL environment was also good experience.