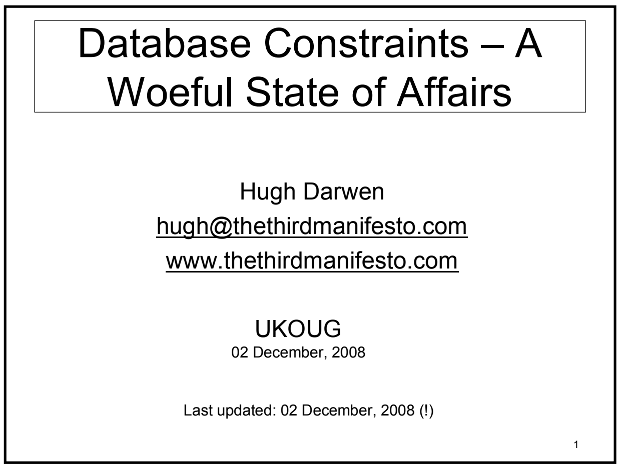


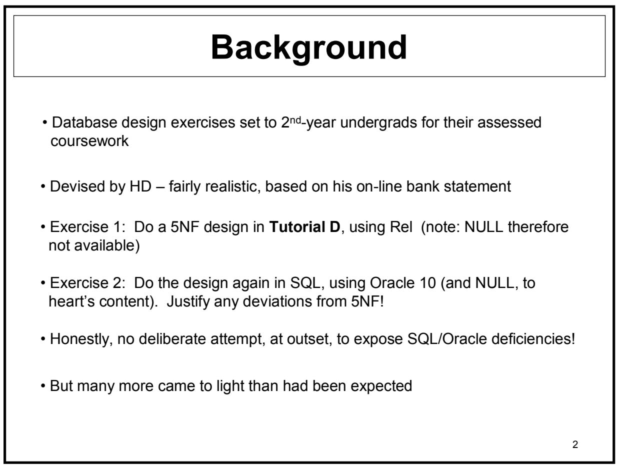
Database Constraints – A Woeful State of Affairs

Hugh Darwen
hugh@thethirdmanifesto.com
www.thethirdmanifesto.com

UKOUG
02 December, 2008

Last updated: 02 December, 2008 (!)

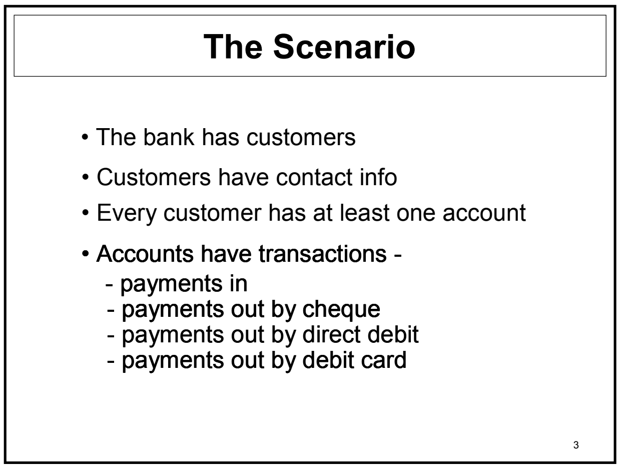
1



Background

- Database design exercises set to 2nd-year undergrads for their assessed coursework
- Devised by HD – fairly realistic, based on his on-line bank statement
- Exercise 1: Do a 5NF design in **Tutorial D**, using Rel (note: NULL therefore not available)
- Exercise 2: Do the design again in SQL, using Oracle 10 (and NULL, to heart's content). Justify any deviations from 5NF!
- Honestly, no deliberate attempt, at outset, to expose SQL/Oracle deficiencies!
- But many more came to light than had been expected

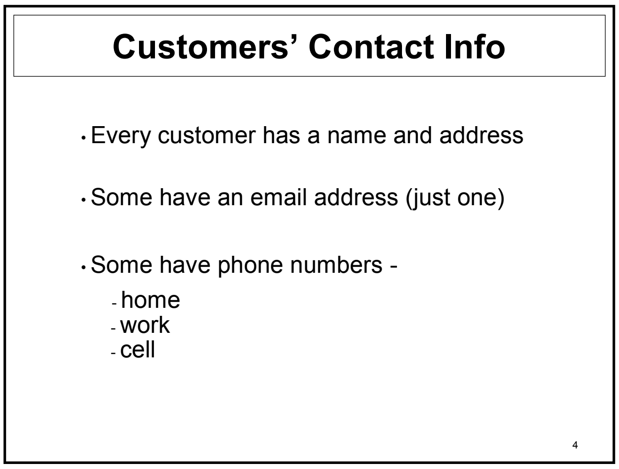
2



The Scenario

- The bank has customers
- Customers have contact info
- Every customer has at least one account
- Accounts have transactions -
  - payments in
  - payments out by cheque
  - payments out by direct debit
  - payments out by debit card

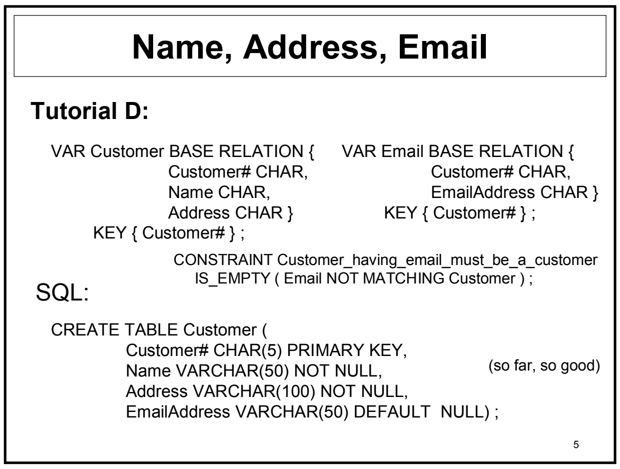
3



Customers' Contact Info

- Every customer has a name and address
- Some have an email address (just one)
- Some have phone numbers -
  - home
  - work
  - cell

4



Name, Address, Email

Tutorial D:

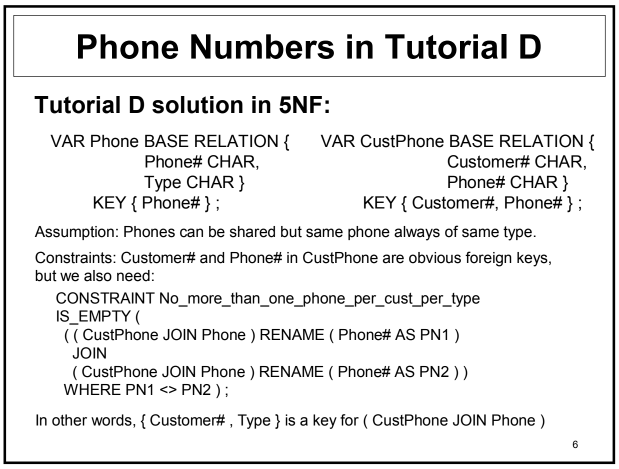
```
VAR Customer BASE RELATION {     VAR Email BASE RELATION {
            Customer# CHAR,                 Customer# CHAR,
            Name CHAR,                      EmailAddress CHAR }
            Address CHAR }              KEY { Customer# } ;
    KEY { Customer# } ;

            CONSTRAINT Customer_having_email_must_be_a_customer
               IS_EMPTY ( Email NOT MATCHING Customer ) ;
```

SQL:

```
CREATE TABLE Customer (
        Customer# CHAR(5) PRIMARY KEY,
        Name VARCHAR(50) NOT NULL,              (so far, so good)
        Address VARCHAR(100) NOT NULL,
        EmailAddress VARCHAR(50) DEFAULT  NULL) ;
```

5



Phone Numbers in Tutorial D

Tutorial D solution in 5NF:

```
VAR Phone BASE RELATION {     VAR CustPhone BASE RELATION {
        Phone# CHAR,                      Customer# CHAR,
        Type CHAR }                       Phone# CHAR }
    KEY { Phone# } ;              KEY { Customer#, Phone# } ;
```

Assumption: Phones can be shared but same phone always of same type.

Constraints: Customer# and Phone# in CustPhone are obvious foreign keys, but we also need:

```
CONSTRAINT No_more_than_one_phone_per_cust_per_type
IS_EMPTY (
  ( ( CustPhone JOIN Phone ) RENAME ( Phone# AS PN1 )
    JOIN
    ( CustPhone JOIN Phone ) RENAME ( Phone# AS PN2 ) )
  WHERE PN1 <> PN2 ) ;
```

In other words, { Customer# , Type } is a key for (CustPhone JOIN Phone)

6

Phone Numbers in SQL

Try (doesn't work):

```
CREATE TABLE Phone (
            Phone# VARCHAR(30) PRIMARY KEY,
            Type CHAR(4) NOT NULL ) ;

CREATE TABLE CustPhone (
            Customer# CHAR(5),
            Phone# VARCHAR(30),
            PRIMARY KEY ( Customer#, Phone# ),
            FOREIGN KEY ( Customer# ) REFERENCES Customer,
            FOREIGN KEY ( Phone# )  REFERENCES Phone ) ;

CREATE ASSERTION Key_for_CustPhone_JOIN_Phone
    CHECK ( ( SELECT COUNT(*) FROM CustPhone NATURAL JOIN Phone ) =
              ( SELECT COUNT DISTINCT ( Customer#, Type )
                FROM CustPhone NATURAL JOIN Phone ) ) ;
```

But Oracle doesn't have CREATE ASSERTION   (and what if it did?)

7

Phone Numbers in SQL

So, we resort to a hack, violating 5NF:

```
CREATE TABLE Phone (
            Phone# VARCHAR(30) PRIMARY KEY,
            Type CHAR(4) NOT NULL ,
             UNIQUE ( Phone#, Type )  );  -- redundant constraint!!

CREATE TABLE CustPhone (
            Customer# CHAR(5),
            Phone# VARCHAR(30),
            Type CHAR(4) NOT NULL ,
            PRIMARY KEY ( Customer#, Phone# ),
            FOREIGN KEY ( Customer# ) REFERENCES Customer,
            FOREIGN KEY ( Phone#, Type ) -- thanks to that redundant constraint,
                REFERENCES Phone ( Phone#, Type ) , -- to address 5NF violation
            UNIQUE ( Customer#, Type ) ) ; -- instead of key for a join!
```

8 constraints instead of 6, and an utterly unobvious (though well publicised) solution

8

Requirements Arising

1. A foreign key should be allowed to reference a proper superkey, such as (Phone#, Type)
2. It should be possible to declare a key for a derived table, especially a simple join such as CustPhone with Phone …
3. … and for a foreign key to reference that key
4. In any case, support CREATE ASSERTION (it's been in the ISO SQL standard since 1992)

9

Customers' Accounts

SQL try (doesn't work):

```
CREATE TABLE Account (
                    Account# CHAR(8),
                    Customer# CHAR(5) NOT NULL,
                    Type VARCHAR(8) NOT NULL,
                    CHECK ( Type IN ( 'current', 'savings', 'mortgage' ) ),
                    DateOpened DATE NOT NULL,
        PRIMARY KEY ( Account# ),
        FOREIGN KEY ( Customer# ) REFERENCES Customer ) ;
```

Recall: Every customer has at least one account

But foreign key in Customer can't reference Customer# in Account (not a key), and Oracle won't let us do CREATE ASSERTION.

In any case, foreign keys in both directions make updating difficult (and deferred constraint checking, if available, is deprecated)

10

Requirements Arising

1. A foreign key should be allowed to reference a nonkey, such as Customer# in Account (though the term foreign key then ceases to be appropriate—it becomes an inclusion dependency)
2. It should be possible to do more than one update in a single statement, as in **Tutorial D:**

```
INSERT Customer RELATION { … } ,  /* note the comma */
INSERT Account   RELATION { … }  ;
```

11

Transactions

Preferred solution has a separate table for each type, avoiding use of NULL:

- They all have transaction number, date and time received, amount.
- Payments in have an account number and a source.
- Payments by cheque have an account number, cheque number, payee, date written.
- Payments by direct debit have an account number and a payee.
- Payments by debit card have a card number (instead of account number) and a payee.

12

Transaction Tables in SQL

1. Payments in:

```
CREATE TABLE PaymentIn (
                    Transaction# INTEGER,
                    Account# CHAR(8),
                    DateRcvd DATE NOT NULL,
                    TimeRcvd TIME NOT NULL,
                    Amount DECIMAL(9,2) NOT NULL,
                    Source VARCHAR(100) NOT NULL,
          PRIMARY KEY ( Transaction#, Account# ),
          FOREIGN KEY Account# REFERENCES Account,
          CHECK ( DateRcvd >=
                    ( SELECT DateOpened
                      FROM Account
                      WHERE Account.Account# = PaymentIn.Account# ) ) );
```

Problem 1: Oracle doesn't allow subqueries in constraints.

13

Transaction Tables in SQL

2. Payments by cheque:

```
CREATE TABLE PaymentByCheque (
                    Transaction# INTEGER,
                    Account# CHAR(8),
                    DateRcvd DATE NOT NULL,
                    TimeRcvd TIME NOT NULL,
                    Amount DECIMAL(9,2) NOT NULL,
                    Cheque# CHAR(6) NOT NULL,
                    DateWritten DATE NOT NULL,
                    Payee VARCHAR(100) NOT NULL,
          PRIMARY KEY ( Transaction#, Account# ),
          FOREIGN KEY Account# REFERENCES Account,
          CHECK ( DateRcvd >= etc. ),
          CHECK ( ( Transaction#, Account# ) NOT IN
                    ( SELECT Transaction#, Account#
                      FROM PaymentIn ) ) );
```

Problem 2: Oracle still doesn't allow subqueries in constraints.

14

Transaction Tables in SQL

3. Payments by direct debit:

```
CREATE TABLE PaymentByDD (
                    Transaction# INTEGER,
                    Account# CHAR(8),
                    DateRcvd DATE NOT NULL,
                    TimeRcvd TIME NOT NULL,
                    Amount DECIMAL(9,2) NOT NULL,
                    Payee VARCHAR(100) NOT NULL,
          PRIMARY KEY ( Transaction#, Account# ),
          FOREIGN KEY Account# REFERENCES Account,
          CHECK ( DateRcvd >= etc. ),
          CHECK ( ( Transaction#, Account# ) NOT IN
                    ( SELECT Transaction#, Account# FROM PaymentIn
                      UNION
                      SELECT Transaction#, Account# FROM PaymentByCheque )
                    ) ) ;
```

Problem 3: That NOT IN constraint is getting worse …

15

Transaction Tables in SQL

4. Payments by debit card:

```
CREATE TABLE PaymentByCard (
                    Transaction# INTEGER,
                    Card# CHAR(8), -- account# implied by card#, see FK below
                    DateRcvd DATE NOT NULL,
                    TimeRcvd TIME NOT NULL,
                    Amount DECIMAL(9,2) NOT NULL,
                    Payee VARCHAR(100) NOT NULL,
          PRIMARY KEY ( Transaction#, Card# ),
          FOREIGN KEY Card# REFERENCES DebitCard,
          CHECK ( DateRcvd >= etc. ) ;
```

Problem 4: How, now, to make sure the same transaction number hasn't been used for some other transaction for the same account? Needs a complicated CREATE ASSERTION!

16

Subtable Approach for Transactions?

Can Problem 4 be addressed by having a "parent" transaction table for all the stuff common to all, and "subtables" for the four transaction types?

Well, no, not according to the usual approach for subtables, because each subtable is supposed to "inherit" the key of the supertable, and PaymentByCard doesn't have Account#. Need to be able to have a joined table as a subtable?

But even support for subtables is inadequate. In our example, we need a constraint to require each transaction to be of exactly one of the special types (a "foreign distributed key"). And then the cyclic constraint problem reappears!

17

Requirements Arising

1. Need to allow a key to span several tables, such that no key value can appear in more than one of them (so all transaction numbers for same account are unique in our example).
2. A joined table to be allowed as one of the tables in such a key, as already noted for regular keys (so payments by debit card can be handled).
3. Special constructs to be supported for *temporal* constraints (e.g., so a transaction can't be received before the account is open).

18

An Unanticipated Problem

Students were asked to use just one table for transactions, using NULL for inapplicable column values (and some very complicated constraints).

This table has PRIMARY KEY (Transaction#, Account#)
(accepting redundant Account# for payments by card, raising a problem similar to the one we met with phone numbers)

But for payments by cheque we need UNIQUE (Cheque#, Account#) too. Note that Cheque# is NULL for other kinds of payment.

Unfortunately, Oracle doesn't conform to the ISO standard definition of UNIQUE. Instead, two rows with same Account# and NULL for Cheque# are deemed to violate the constraint!

19

Requirement Arising

Remove all support for NULL
(joke) (sort of)

20

Research Needed

- New shorthands to be devised for commonly arising constraints

  E.g. The "distributed key", whose uniqueness scope is over several tables, and the corresponding "foreign distributed key".

- Optimiser to be able to detect constraints, expressed in the general way, for which efficient execution plans can be devised.

```
E.g. NOT EXISTS ( SELECT * FROM T1
                  WHERE C1 IN ( SELECT C1 FROM T2 ) )
```

  or, once table comparison is permitted (as it should be):

```
SELECT DISTINCT C1 FROM T1 = SELECT DISTINCT C1 FROM T2
```

- **Multiple assignment**, new to the DBMS world, needs investigation too.

21

The End

22