Keynote
SWO’99 Wisla, Poland

Round-Trip Engineering with Databases

Leszek A. Maciaszek
Macquarie University, Sydney, Australia

Topics

- What is round-trip engineering (RTE)?
- What do we expect from RTE with database (RTE/DB)?

Main enquiries:
- do RTE/DB tools and methods deliver?
- what are the shortcomings?
- can they deliver? (what processes are needed?)

Examples of some RTE/DB scenarios
Process model for RTE/DB

Modern software development...

- is iterative \rightarrow \text{stream of executable releases}
- is incremental \rightarrow each new release extended with new features
- requires that system models are refined and transformed through analysis, design and implementation phases
- requires RTE support, in particular for the design-implementation cycle (lower eng.):
  - design changes to be forward engineered
  - implementation changes to be reverse engineered

Objective

the design documentation reflects the current state of the database implementation at all times (or - to be realistic - at "synchronization" times)

Pre-conditions:
- forward engineering (FE) generates the data and procedural parts of a DB intention (and test DB extension)
- reverse engineering (RE) recovers design models (incl. procedural parts) from the DB implementation
- FE generates incremental code and re-instates the DB extension in successive iterations

STD for RTE/DB

RTE/DB - can it be automatic in the simplest case?

no changes, just FE to a new DB followed by RE to a new PDM
Scenario 1 - addition of null-allowing column to table

```sql
ALTER TABLE T_Employee
ADD birth_date smalldatetime null
```

Addition of a null-allowing column to a table.

**Categories of changes:**
- Addition of a column
- Addition of a user data type
- Addition of a view definition

**The process:**
- Add new object (e.g. a table) to PDM
- Generate SQL database modification script
- Execute the script on DB (i.e. synchronize DB)
- Archive PDM

**Notes:**
- Column 'birth_date' cannot be added to table 'T_Employee' because it does not allow nulls.
- A default value on birth_date would not eliminate the server error.
- Adding a new column to a table in PDM and generating "Modify Database" script will not work (Example next)

**Scenario 2 - addition of not-null column to table**

In PDM, the NOT NULL birth_date column added to T_Employee table.

```
T_Employee
```

**Categories of changes:**
- Addition of a not-null column to a table.
- Addition of a referential integrity that requires a not-null foreign key in an existing table.

**The process:**
- Add PDM
- Generate SQL database modification script
- Execute the script on DB (i.e. synchronize DB)
- Archive PDM

**Notes:**
- The original PDM was dropped (PDM and DB) re-created (FE) and data re-inserted with new insert scripts (DB). Also, invalidated FK indexes have to be repaired manually in PDM.

**Scenario 2 - addition of not-null column to table (cont.)**

Modify Database script can be generated in PDM, but – as expected – does not execute on DB. (Note that the setting of a default value on birth_date would not eliminate the server error).

```
ALTER table T_Employee
ADD birth_date smalldatetime
    default '1-JAN-80' not null
```

Server Message: Number 40901, Severity 16

ALTER TABLE only allows columns to be added which can contain nulls. Column 'birth_date' cannot be added to table 'T_Employee' because it does not allow nulls.

This means that the table has to be dropped (PDM and DB), re-created (FE) and data re-inserted with new insert scripts (DB). Also, invalidated FK indexes have to be repaired manually in PDM.
**Scenario 2 - addition of not-null column to table (cont.)**

**Other limitations:**

- Dropping a table with the primary key pointed to from other tables breaks the referential integrity of DB. This integrity is not guaranteed to be restored when the table is re-created and data re-inserted. (The integrity is verified when foreign keys are inserted in “child” tables, not when the primary keys are inserted in “parent” table.)

  This risk can be alleviated by first deleting the records from the table and then dropping it. The Delete action causes appropriate triggers to fire and requires remedial actions on child tables - thus allowing to drop the table later in a safe manner.

**Scenario 2 - the process**

- Cut (remove) the table that requires new column(s) from PDM.
- In PDM, modify any invalidated indexes on foreign keys that pointed to the primary key of the table just removed. Archive PDM.
- Paste the table back to PDM and re-establish foreign keys and indexes.
- Add a not-null column.
- Generate SQL database modification script.
- Execute the script on the database.
- Re-insert the data into the re-created (ie. dropped and created) table.
- Archive PDM.

**Scenario 3 - RE of more problematic schema additions**

**Categories of changes, as in Scenario 2:**

- Addition of a not-null column to a table.
- Addition of a referential integrity that requires a not-null foreign key in an existing table.

**Scenario 3 - the process**

- In DB, copy records from the table to a temporary table, delete records from the original table and drop it.
- In DB, drop any invalidated indexes on foreign keys that pointed to the primary key of the dropped table.
- In DB, create a new table with a not-null column and re-establish foreign keys and indexes.
- In DB, re-insert the data into the re-created table.
- In PDM, reverse-engineer the table (normally by using an ODBC driver to connect to the database).
- Archive PDM.

**Scenario 4 - FE of schema deletions**

**Categories of changes:**

- Deletion of a column from a table.
- Deletion of a table.

**Scenario 4 - deletion of column from table**

- PDM - Modify Database script:
  - A straightforward activity with no technical risks.
  - The recommended process is that the PDM designers modify PDM and DB.
  - The impact on client programs can be severe.

- PDM - Modify Database script:
  - Creates a temporary table tmp_T_Employee.
  - Inserts into tmp_T_Employee from T_Employee.
  - Drops T_Employee.
  - Creates the new table tmp_T_Employee.
  - Selectively inserts into T_Employee from tmp_T_Employee.
  - Drops tmp_T_Employee.
  - Modifies and re-creates any triggers affected by the change.
Scenario 4 - the process

- Archive PDM.
- In PDM, delete a column or a table.
- Generate SQL database modification script in PDM.
- Execute the script on DB.
- Archive PDM.

Scenario 5 - FE of additional business rules on schema

Categories of changes

- Addition of declarative business rules (such as data entry validation).
- Addition of procedural business rules implemented in triggers.

- Newly generated database rules and triggers do not validate the prior database content.
- The recommended process is that the PDM designers modify PDM and DB.

Scenario 5 - non-modifiable column

- PDM - Modify Database script:

```sql
/* Update trigger "tu_t_employee" for table "T_Employee" */
create trigger tu_t_employee on T_Employee for update as
... if update(birth_date)
if exists (select 1
from   inserted I, deleted d
where  i.birth_date != d.birth_date)
begin
select @errno = 30001,
@errmsg = 'Column "birth_date" cannot be modified.'
Goto error
end
return
error:
raiserror @errno @errmsg
rollback  transaction
end
```

Scenario 6 - RE to new PDM after changes in DB relationships & indexes

Categories of changes

- PK and FK constraints specified declaratively or by system procedures (but all referential integrity constraints implemented procedurally through triggers).
- Primary key (PK) and foreign key (FK) indexes specified on DB tables.

- RE creates a challenge for a CASE/4GL tool and the reverse-engineered PDM may contain some flaws.

- PK and FK constraints specified declaratively or by system procedures.
- Indexes cannot be reverse-engineered on individual basis - the entire table would have to be reverse-engineered.

Scenario 6 - example after RE

Three relationships re-engineered as only one relationship.
Scenario 6 - the process

- In DB, make necessary changes.
- In DB, re-load the data.
- Pass the changes to PDM designers.
- Conduct selective RE at specified synchronization time.
- Archive PDM.

Scenario 7 - RE to new PDM after changes in DB triggers & procedures

Categories of changes
Triggers modified in DB.
- Stored procedures created or modified in DB.
- RE creates a challenge for a CASE/4GL tool and the reverse-engineered PDM may contain some flaws.
- A care should be taken so that the triggers & stored procedures once modified in DB are not automatically re-generated later in FE activities.

Scenario 7 - the process

- In DB, make necessary changes.
- In PDM, reverse-engineer the modified DB.
- From now on, the triggers and stored procedures can only be modified in DB, not in PDM (they need to be marked in PDM as User-Defined).
- Archive PDM.

Main considerations and conclusions

- The PDM model can be archived and versioned by the CASE tool, but a typical relational DB does not have a built-in capability to maintain DB versions (other than by creating a new DB).
- After initial generation of DB, the need for changes to PDM are frequently “discovered” during programming; therefore, the programmers should be able to modify DB as needed as long as the modifications are populated back to current PDM. The populate event should be done in bulk at specific synchronization times and PDM should be then archived.
- Any later changes to the archived PDM that need to be synchronized with DB, should be forward engineered to a new DB instance.
- A cycle can begin with design phase or with programming phase - the process model requires only that the phases do not conflict, i.e. the programmers do not modify DB in the design phase and the designers do not modify PDM in the programming phase.

References

References (cont.)


