Edition-Based Redefinition Made Easy

How to Upgrade Your Application with no Downtime (and no Additional Costs)

Oren Nakdimon

www.db-oriented.com

✉ oren@db-oriented.com

☎ +972-54-4393763

@DBoriented

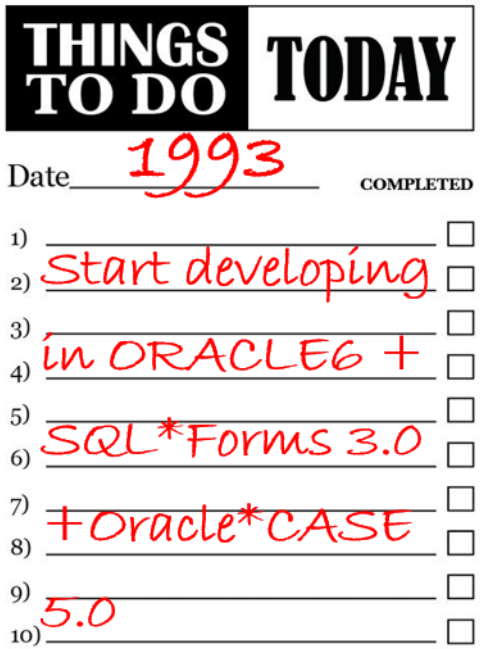
dbORIENTED

moovit

dbORIENTED

Follow @DBoriented

Who Am I ?



http://db-oriented.com

500+ Technical Experts Helping Peers Globally

ORACLE ACE Program

ORACLE ACE Director

ORACLE ACE

ORACLE ACE Associate

---

3 Membership Tiers

- Oracle ACE Director
- Oracle ACE
- Oracle ACE Associate

bit.ly/OracleACEProgram

Connect:

oracle-ace_ww@oracle.com

Facebook.com/oracleaces

@oracleace

Oracle Groundbreakers

Nominate yourself or someone you know: acenomination.oracle.com

Edition-Based Redefinition

Edition-Based Redefinition

Agenda

The problem EBR solves

What EBR is

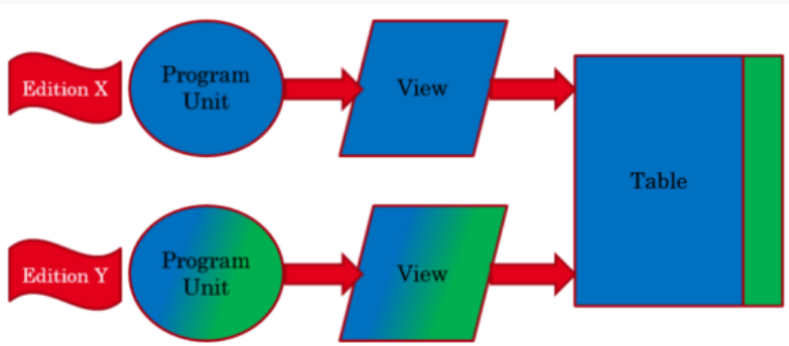
Demo

Additional benefits of EBR

What's in Today's Session (and what's not)

✓	✗
The main and major principles	All the details and nuances
Step-by-Step Examples	Deep philosophy
Guidelines	
System with EBR from day one	Converting non-EBR system

My EBR Blog Post Series



This is an index to a series of posts I have been writing about Edition-Based Redefinition.
New entries will be added as soon as they are published.

- Part 1: Overview and Setup [1-Dec-2017]
- Part 2: Locking, Blocking and ORA-04068 [5-Dec-2017]
- Part 3: Editions and Editioned Objects [15-Dec-2017]
- Part 4: Invalidation and Actualization of Dependent Objects [9-Jan-2018]
- Part 5: Explicit Actualization of Dependent Objects [12-Jan-2018]
- Part 6: Editionable and Non-Editionable, Editioned and Non-Editioned [29-Apr-2018]
- Part 7: Editioning Views [21-May-2018]
- Part 8: The Last Planned Downtime [23-May-2018]
- Part 9: Adding a New Column [25-May-2018]
- Part 10: Data Dictionary Views for Editioning Views [29-Jun-2018]
- Part 11: Database-Level Default Edition [18-Mar-2019]
- Part 12: Editions and Services [10-May-2019]

One Comment

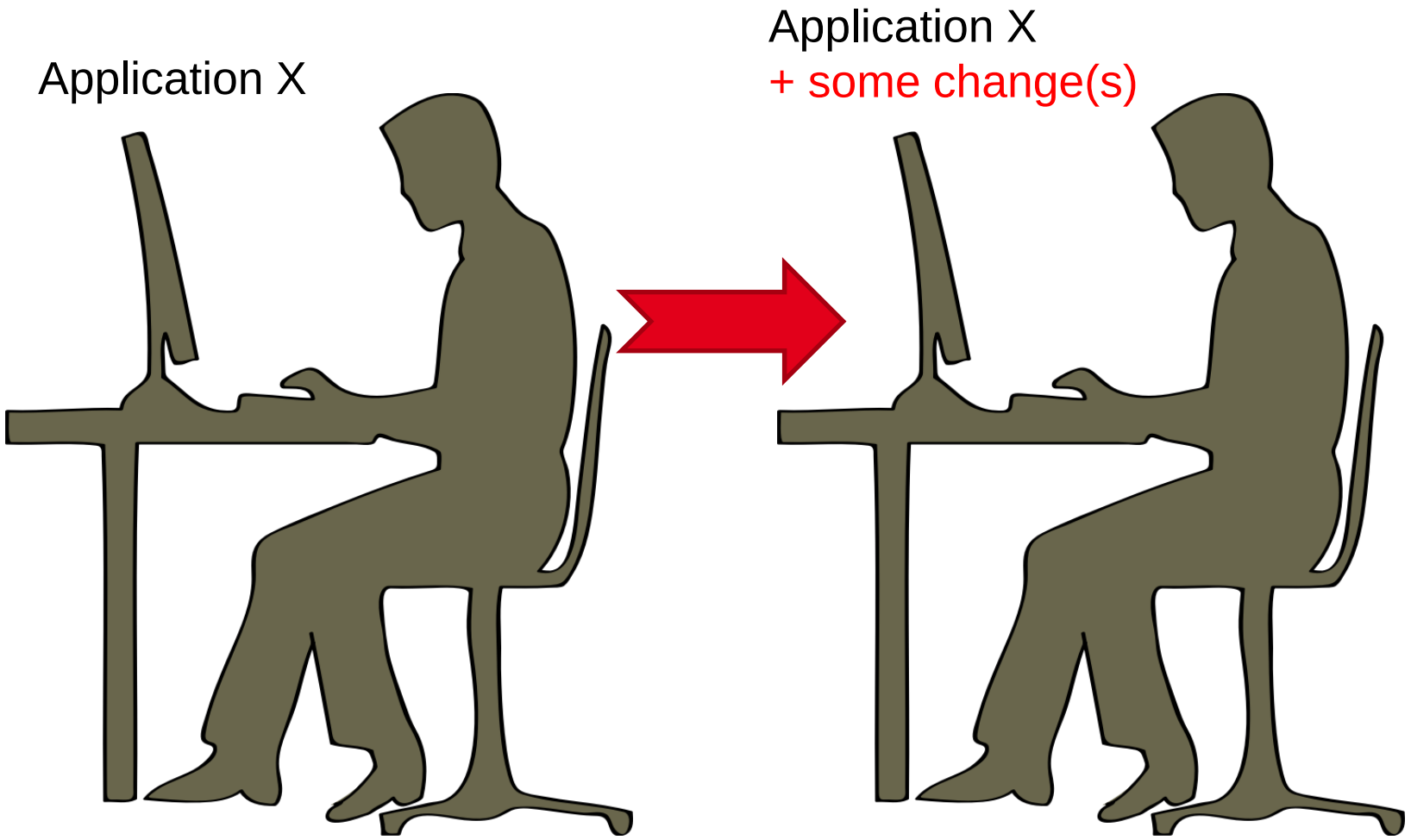
Make sure to test it thoroughly before you decide to apply it in production

Application Upgrades

Application Upgrades

- Upgrades are inherent to every application lifecycle
- They may be:
  - Small or big
  - Frequent or rare
  - Simple or complex
  - With or without schema changes
  - Introducing new functionality, changing existing functionality, or removing functionality

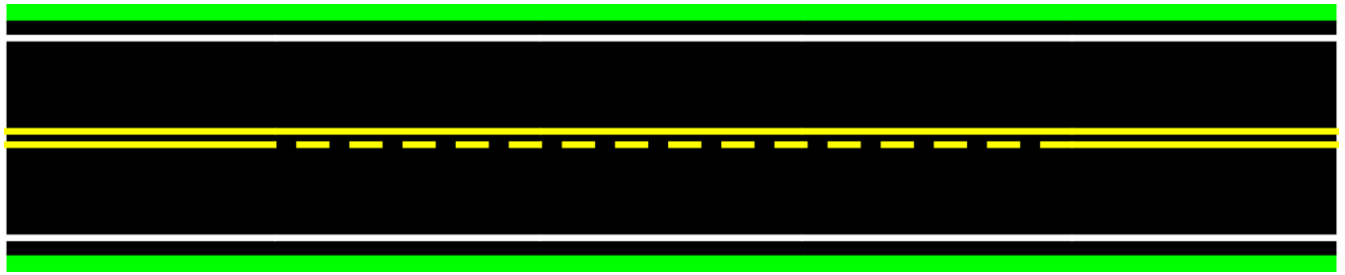
Application Upgrades



Application Upgrades

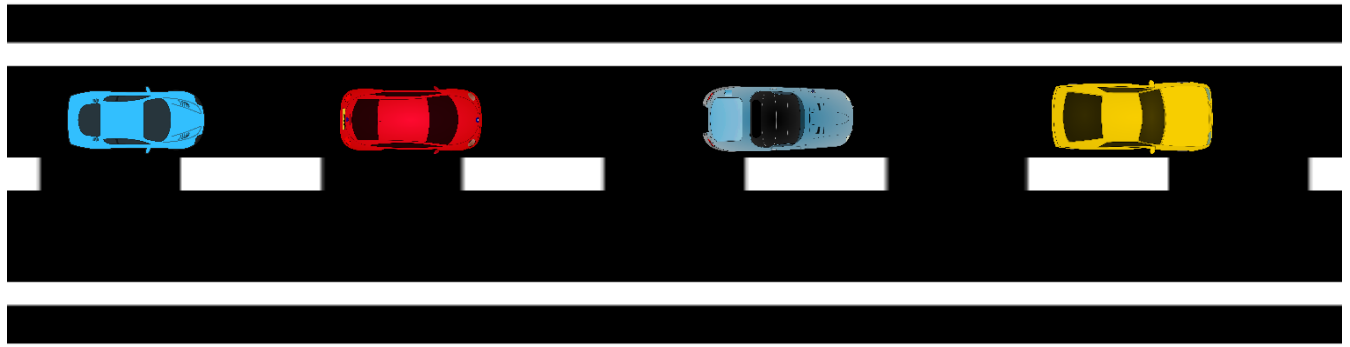
Pre-Upgrade Version

Post-Upgrade Version

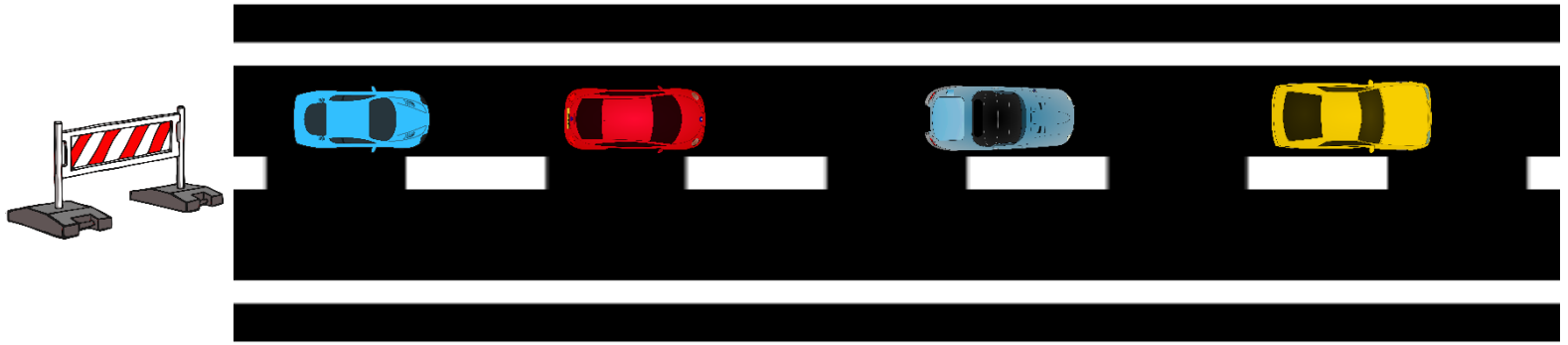


Offline Application Upgrade

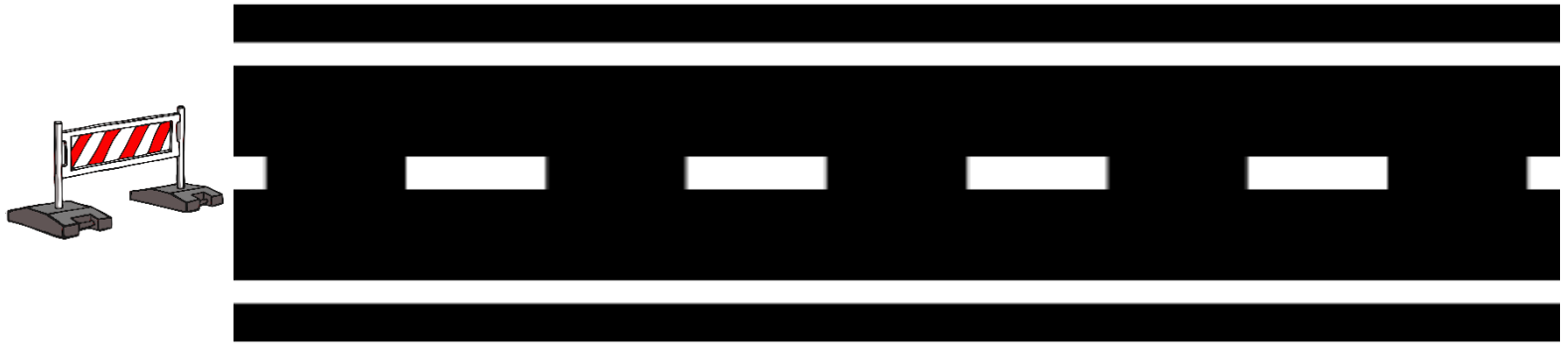
Offline Upgrade (Cold Cutover)



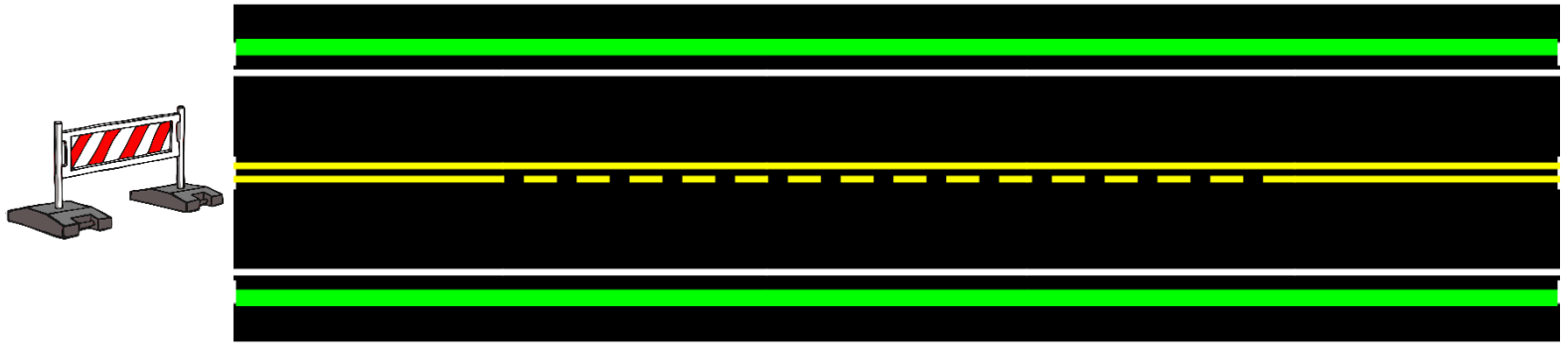
Offline Upgrade (Cold Cutover)



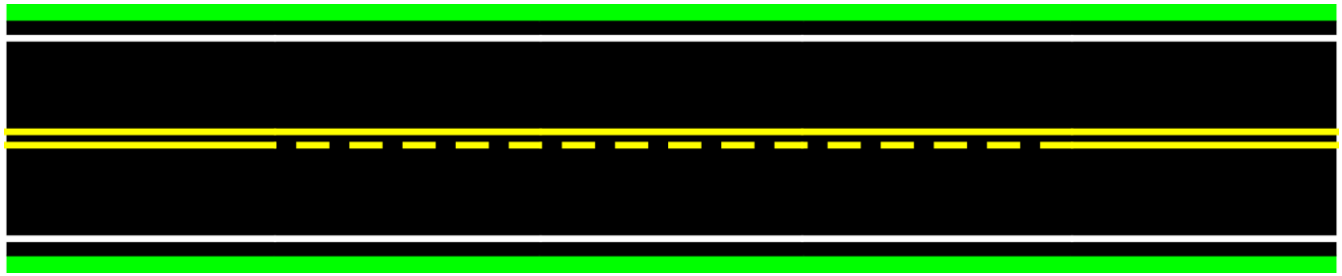
Offline Upgrade (Cold Cutover)



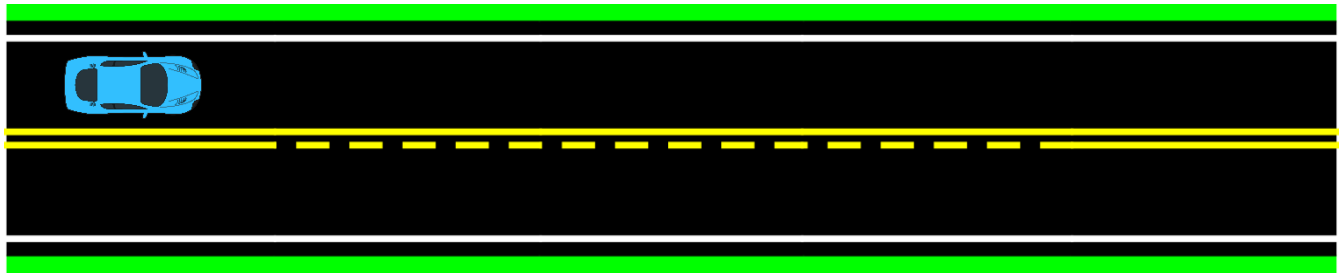
Offline Upgrade (Cold Cutover)



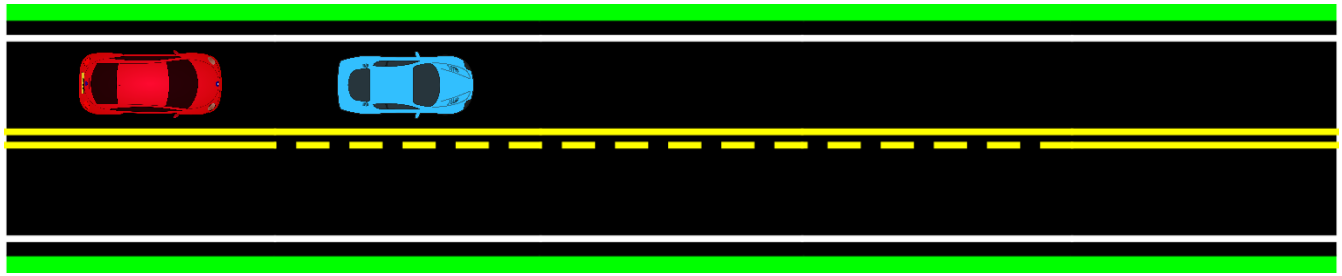
Offline Upgrade (Cold Cutover)



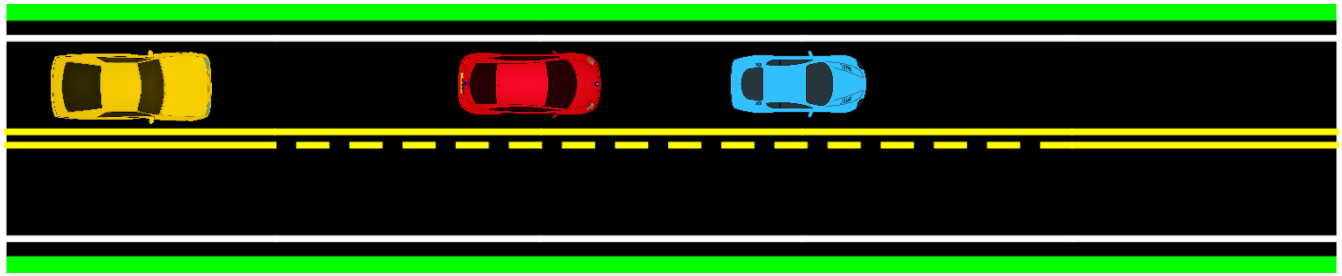
Offline Upgrade (Cold Cutover)



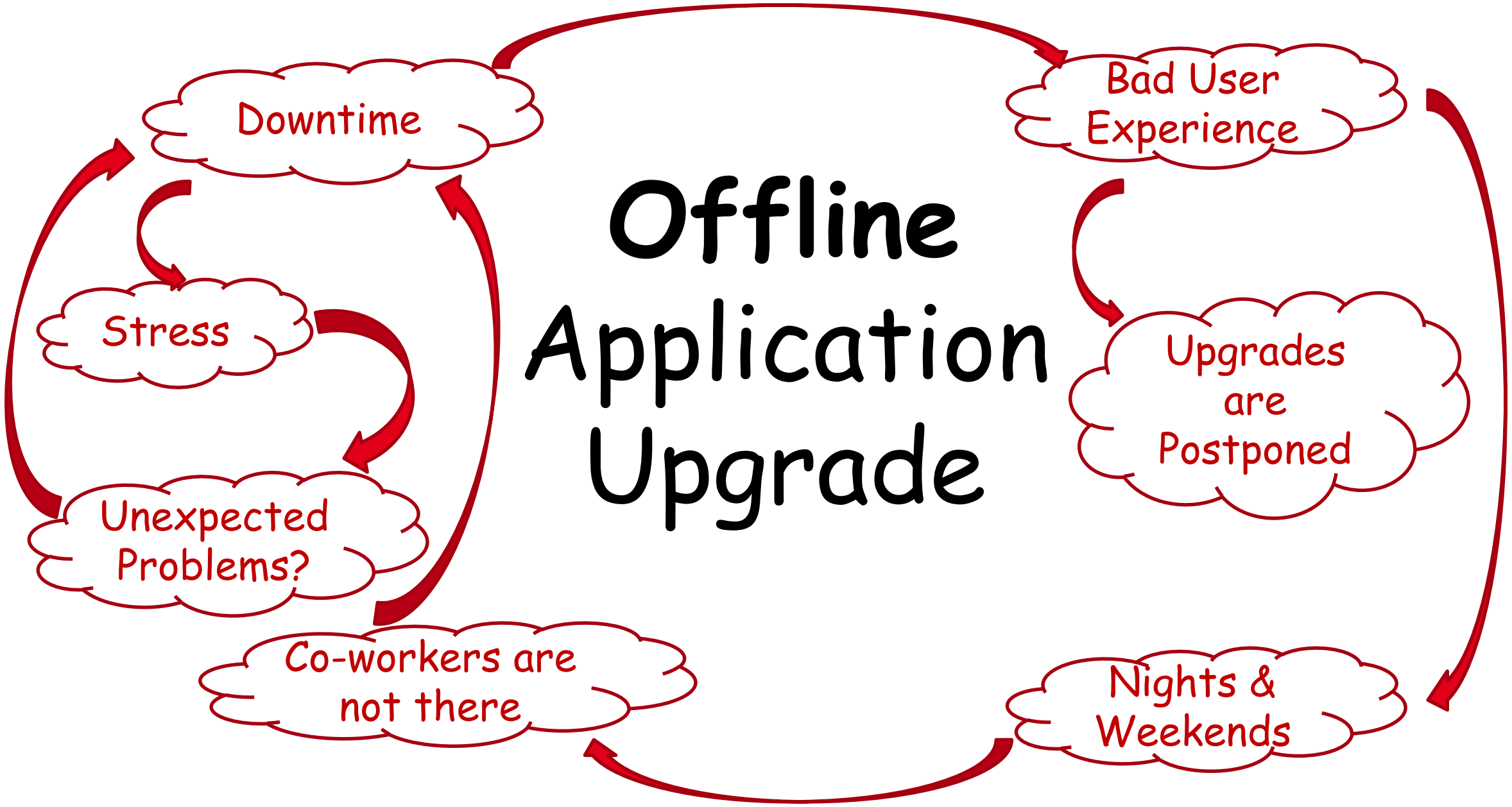
Offline Upgrade (Cold Cutover)



Offline Upgrade (Cold Cutover)



Offline Application Upgrade



Invalidations

What Could Possibly Go Wrong?

Locking and Blocking

ORA-04068

This presentation is available in http://db-oriented.com/presentations

db ORIENTED

About | Presentations | Edition-Based Redefinition (EBR) | Constraint Creation Optimization | Wri

EBR – Part 2: Locking, Blocking and ORA-04068

2 Replies

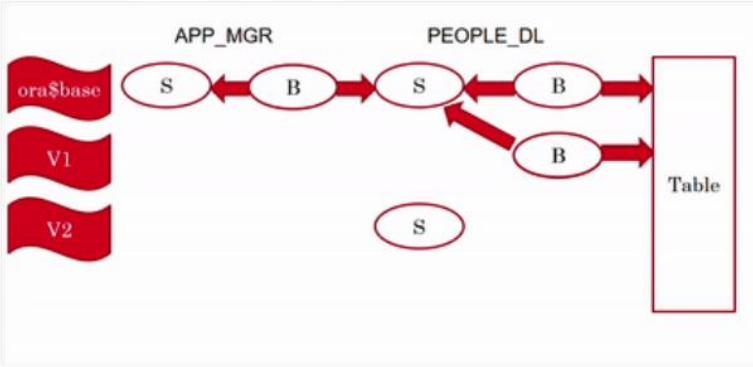
```
JOB_NAME             START_TIME END_TIME   PROGRESS
-------------------- ---------- ---------- -------------------------
session#1 (end user)  09:03:12   09:03:22   **********
session#2 (end user)  09:03:13   09:03:23    **********
session#3 (developer) 09:03:14   09:03:23     *********
session#4 (end user)  09:03:15   09:03:33      ******************
session#5 (end user)  09:03:16   09:03:33       ******************
session#6 (end user)  09:03:17   09:03:33        ******************
session#7 (end user)  09:03:18   09:03:33         ****************
session#8 (end user)  09:03:19   09:03:33          **************
```

db ORIENTED

About | Presentations | Edition-Based Redefinition (EBR) | Constraint Creation Optimization

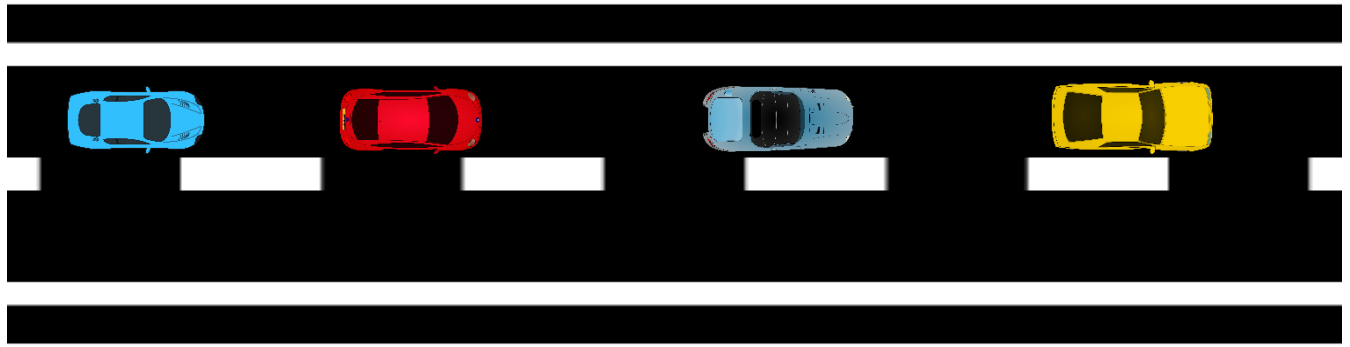
EBR – Part 4: Invalidation and Actualization of Dependent Objects

4 Replies

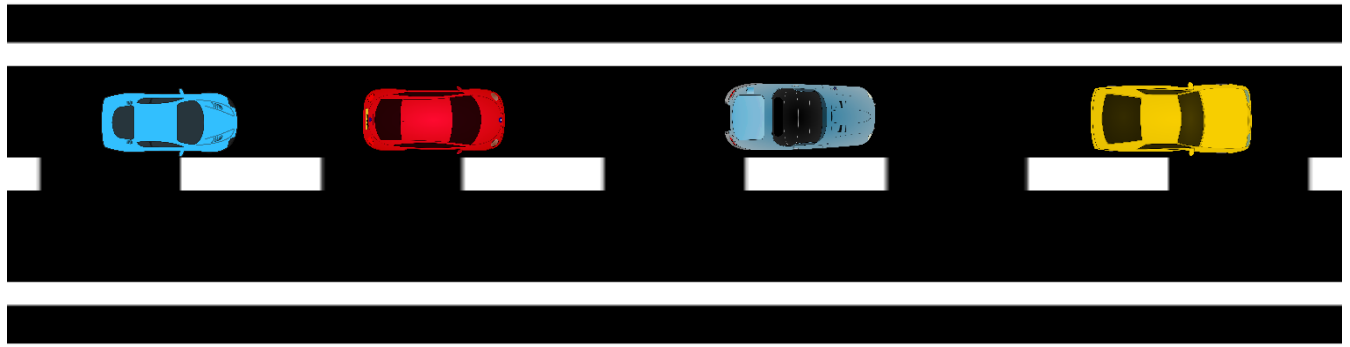


Online Application Upgrade

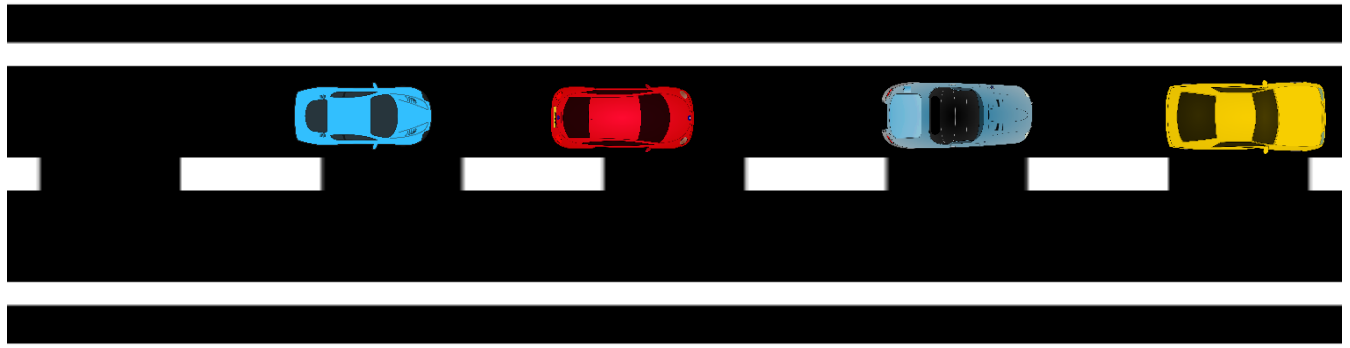
Online Upgrade (Hot Rollover)

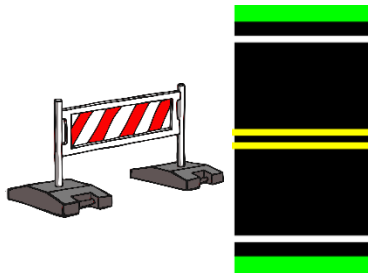


Online Upgrade (Hot Rollover)

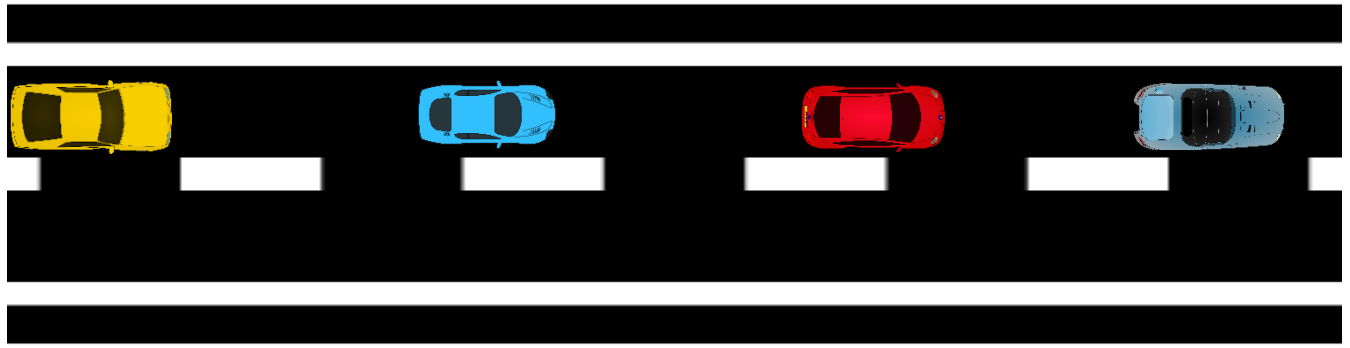


Online Upgrade (Hot Rollover)

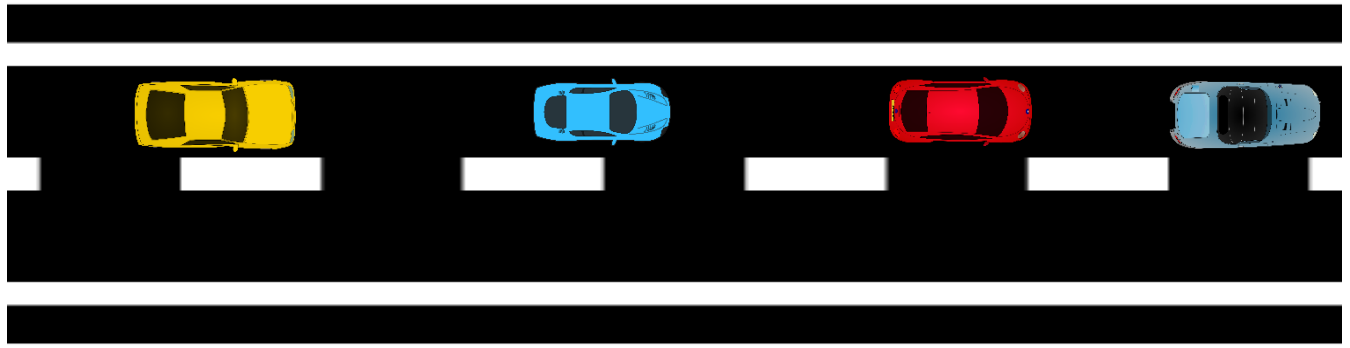


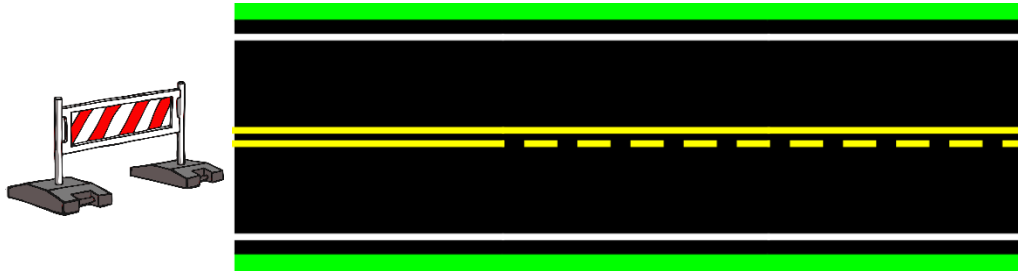


Online Upgrade (Hot Rollover)

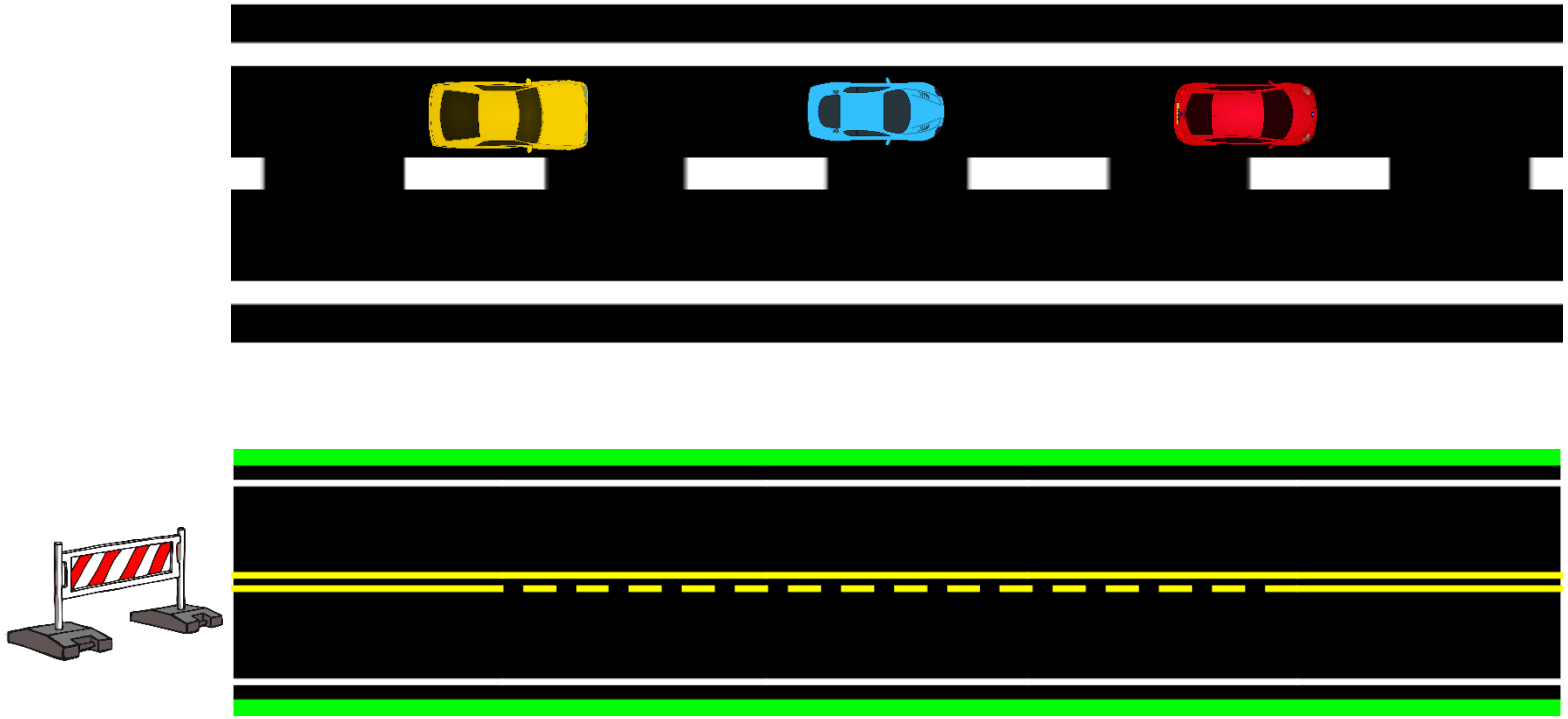


Online Upgrade (Hot Rollover)

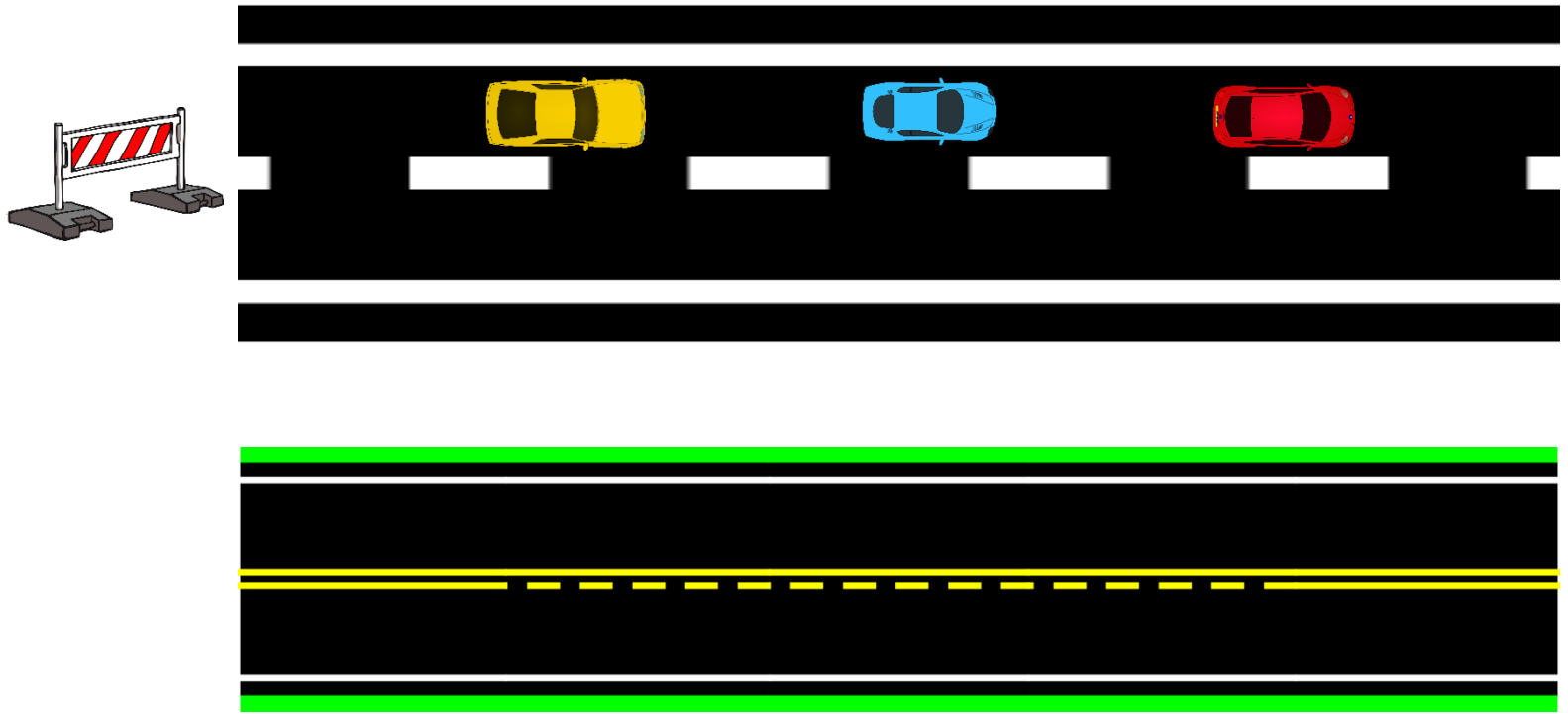




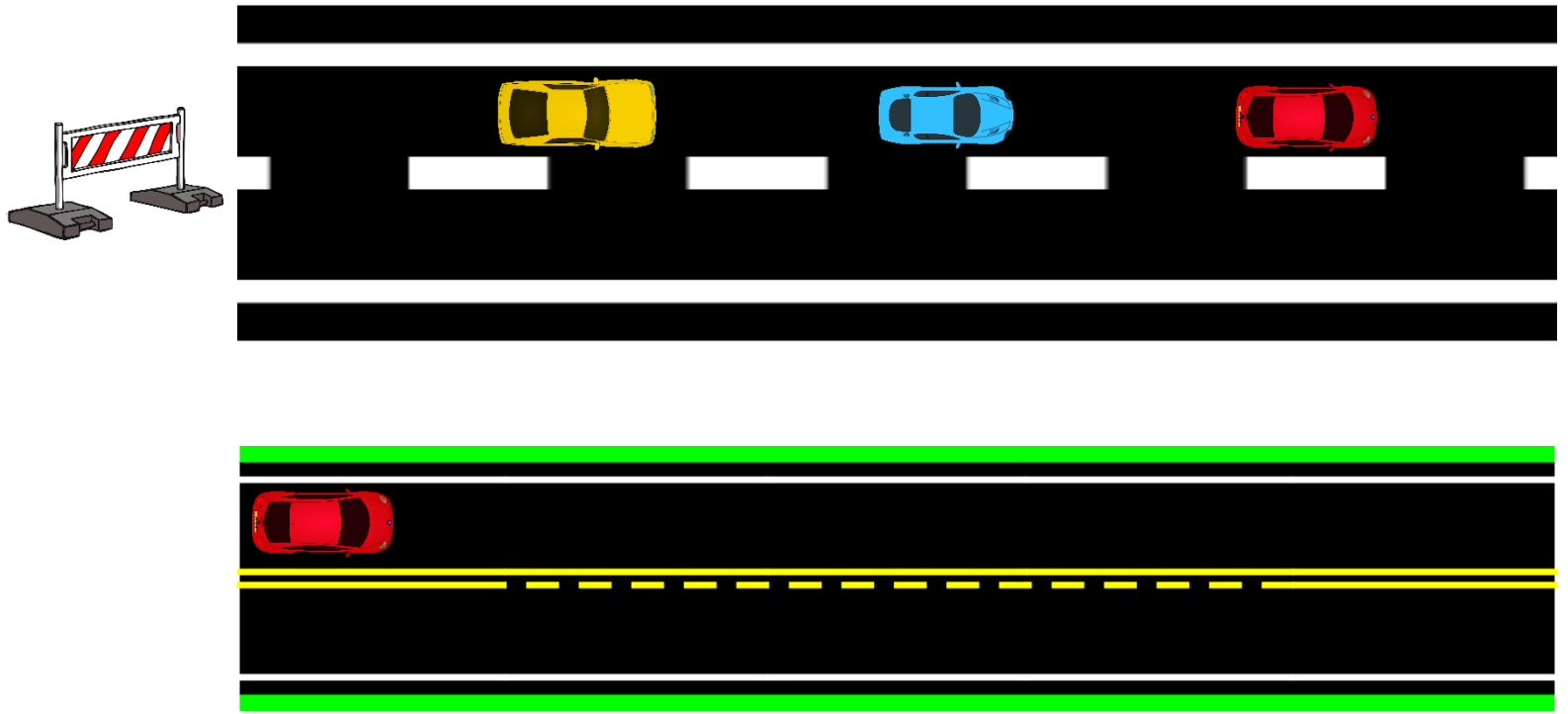
Online Upgrade (Hot Rollover)



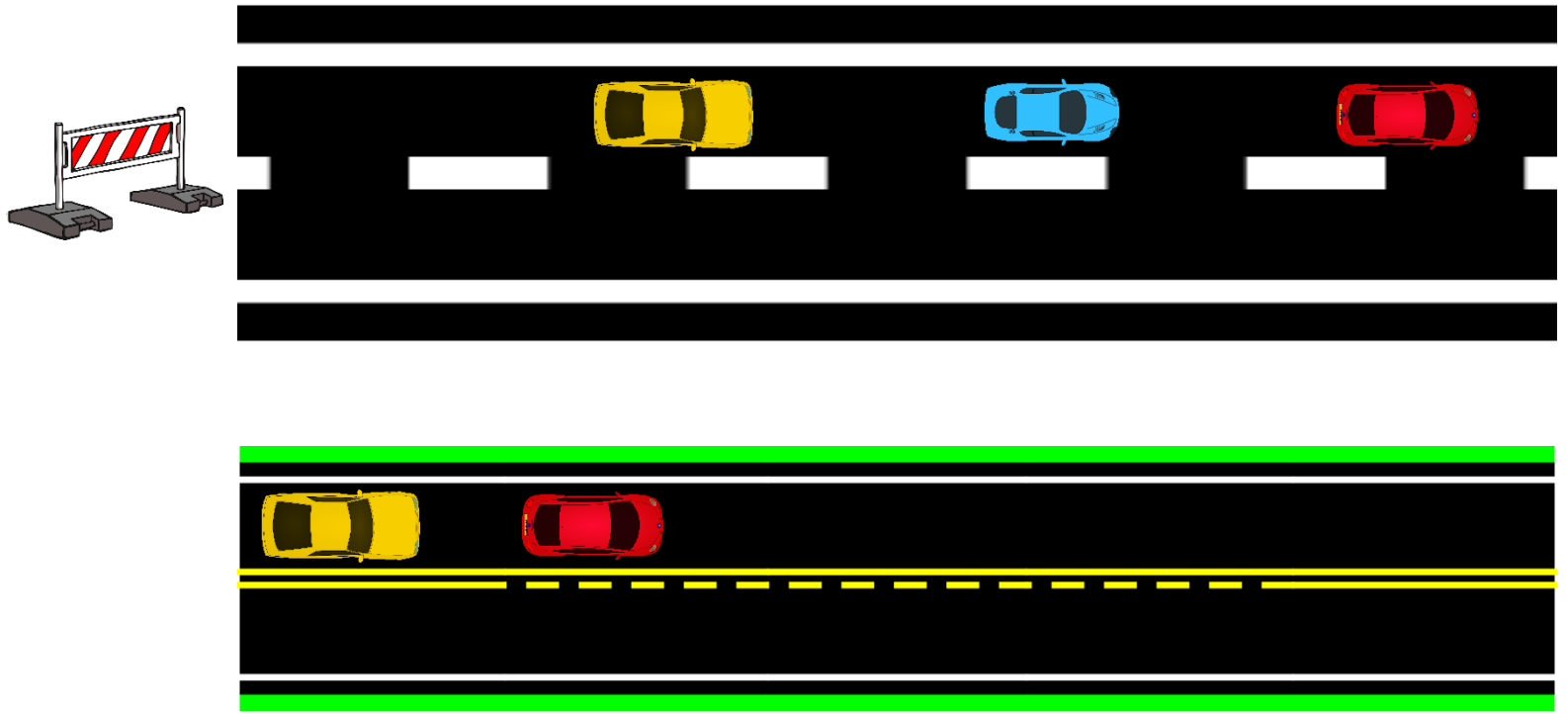
Online Upgrade (Hot Rollover)



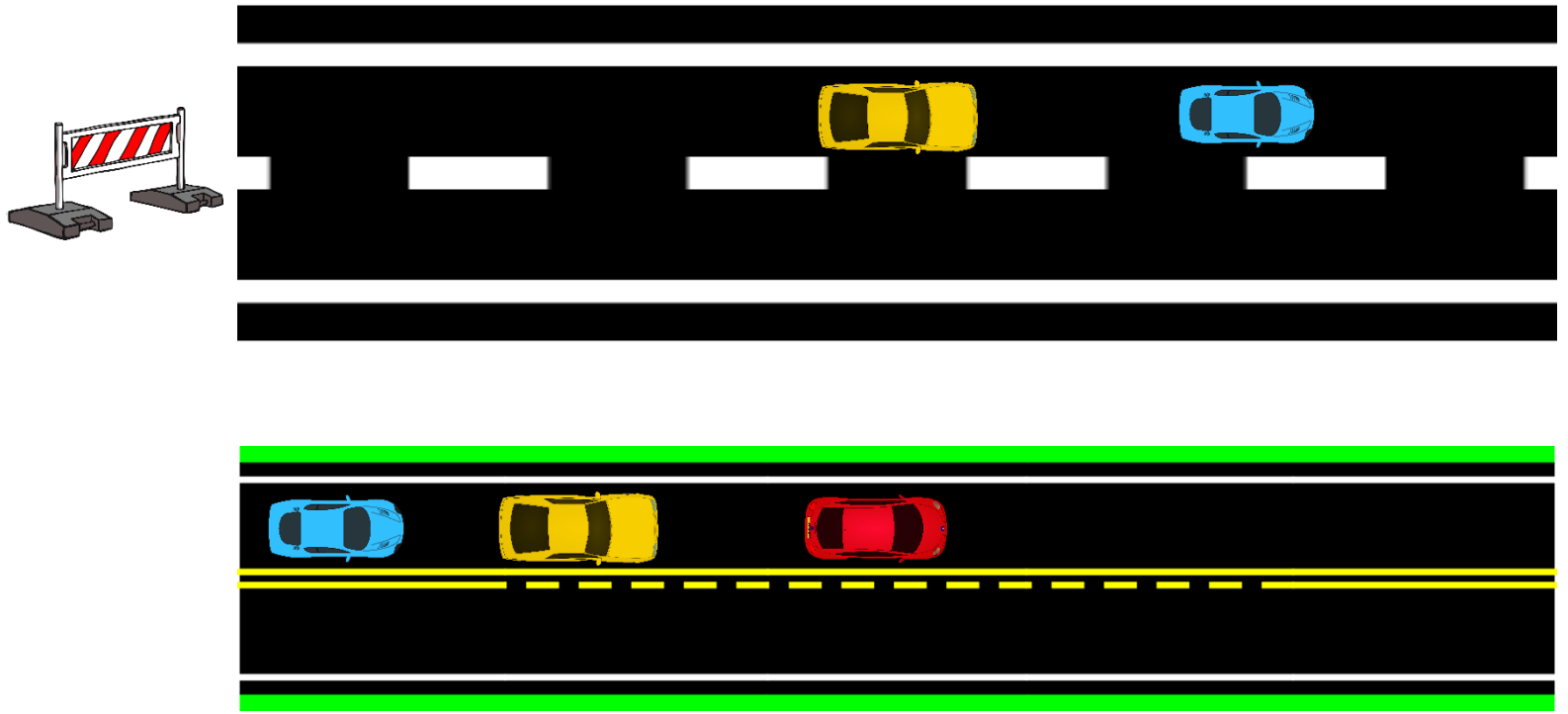
Online Upgrade (Hot Rollover)



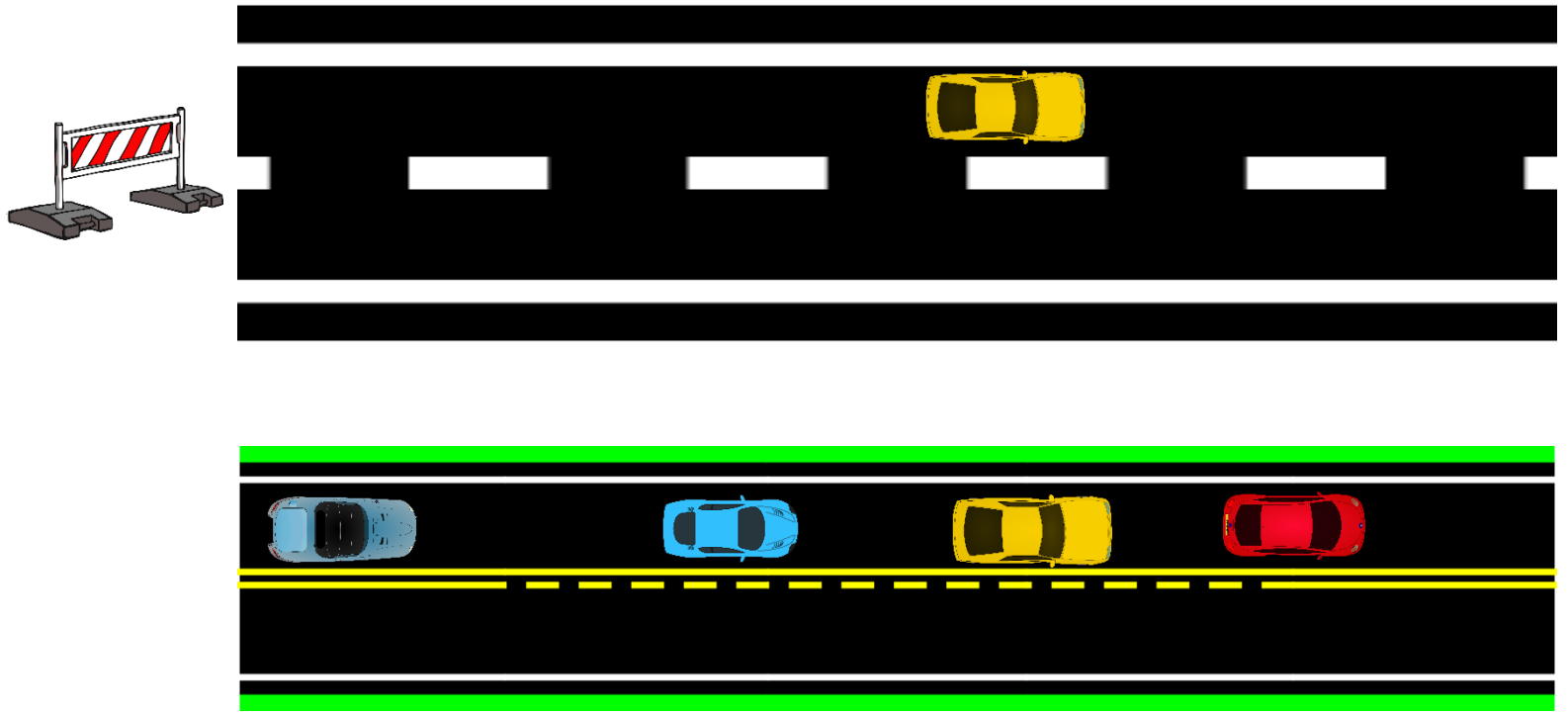
Online Upgrade (Hot Rollover)



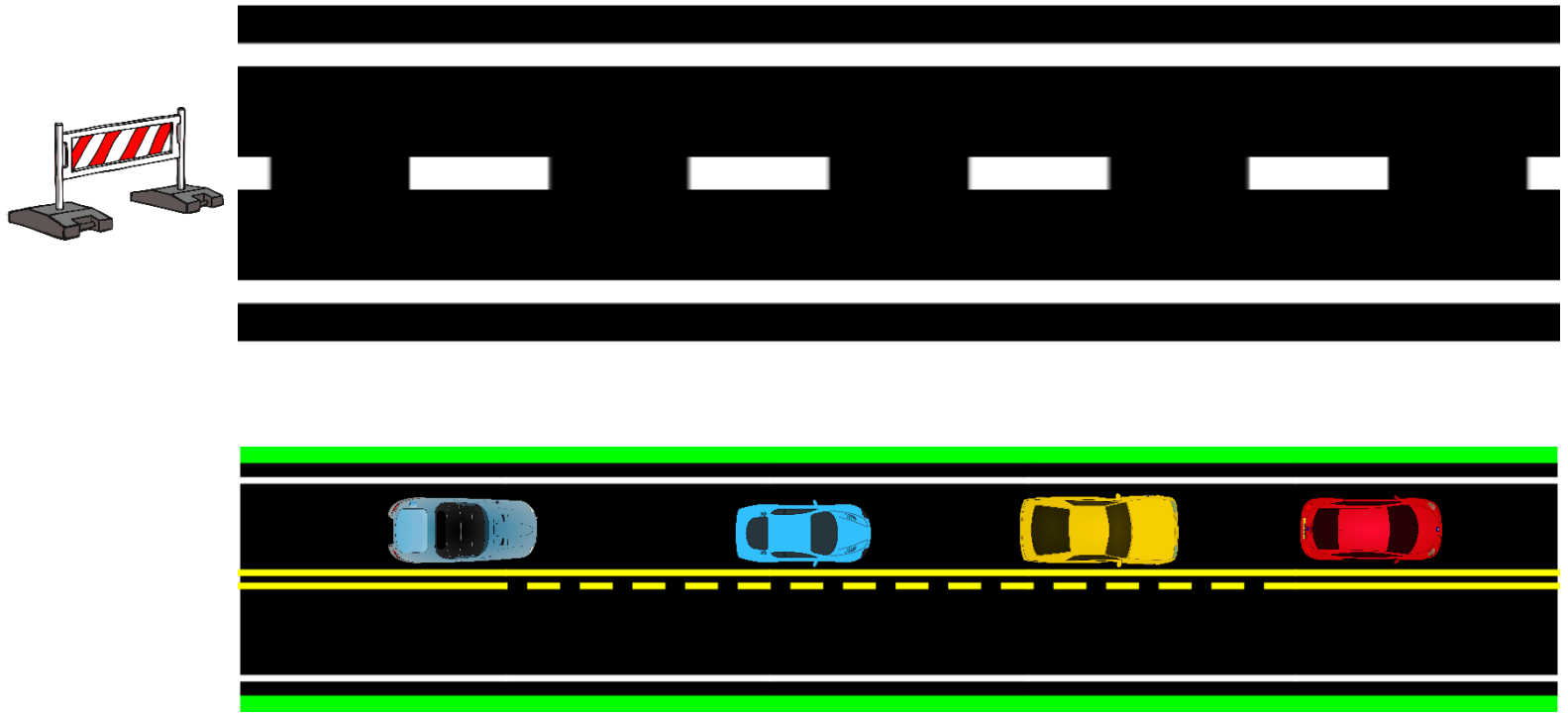
Online Upgrade (Hot Rollover)



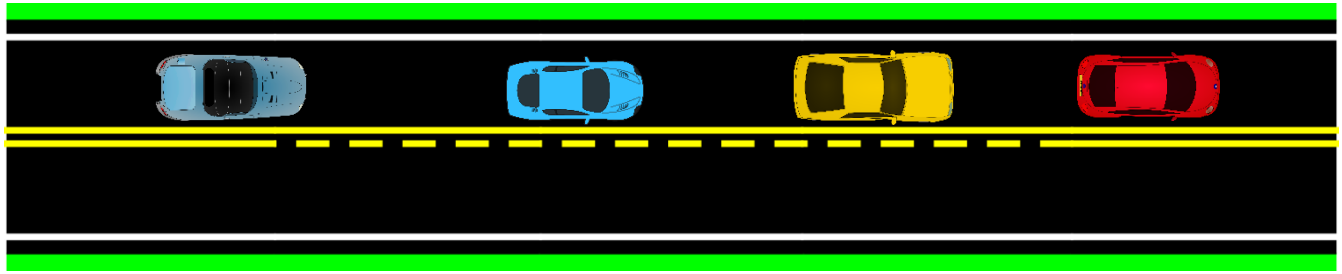
Online Upgrade (Hot Rollover)



Online Upgrade (Hot Rollover)



Online Upgrade (Hot Rollover)



EBR is a **feature set** that lets you upgrade the database component of an application while it is in use, thereby minimizing or eliminating downtime

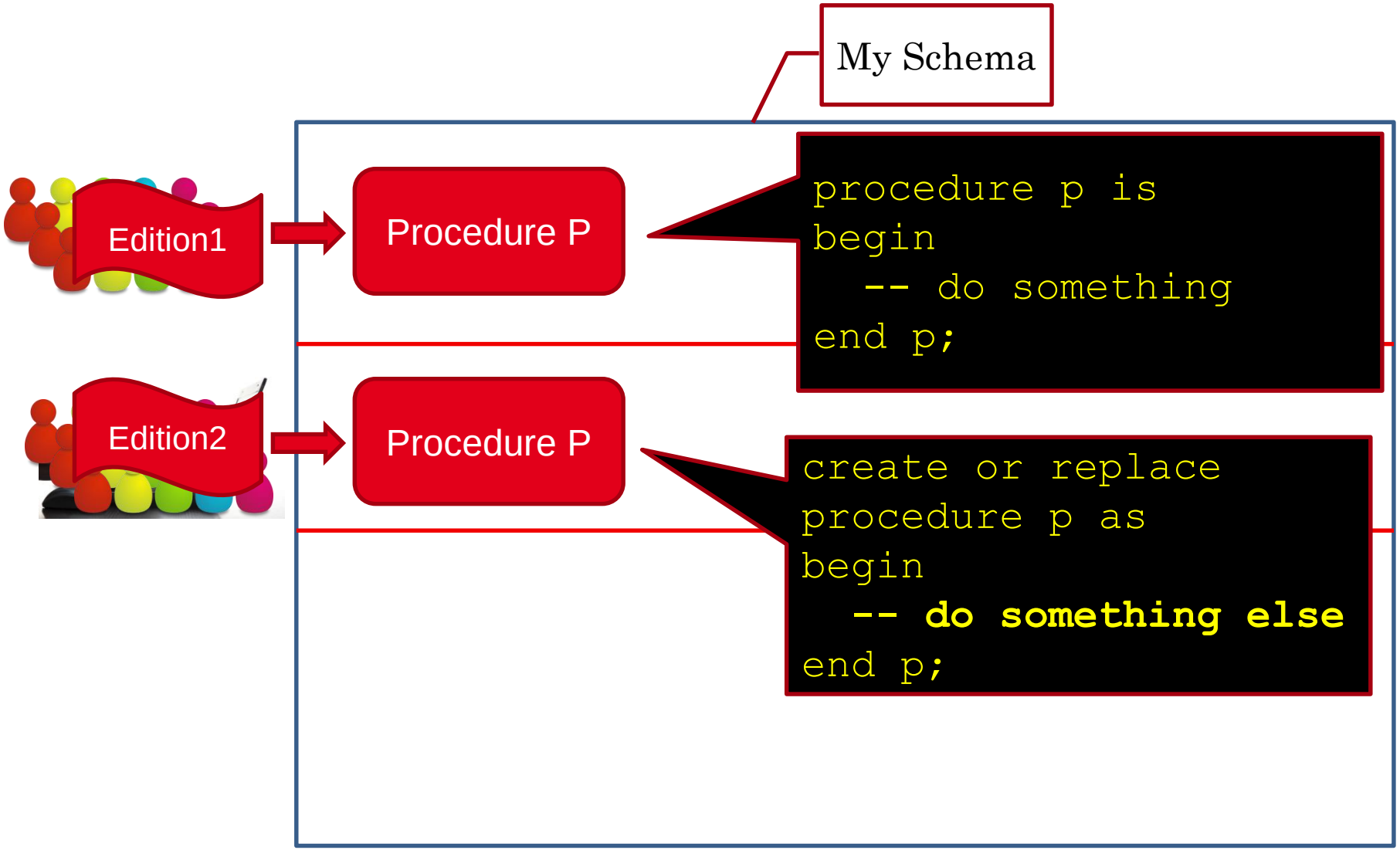
Edition-Based Redefinition

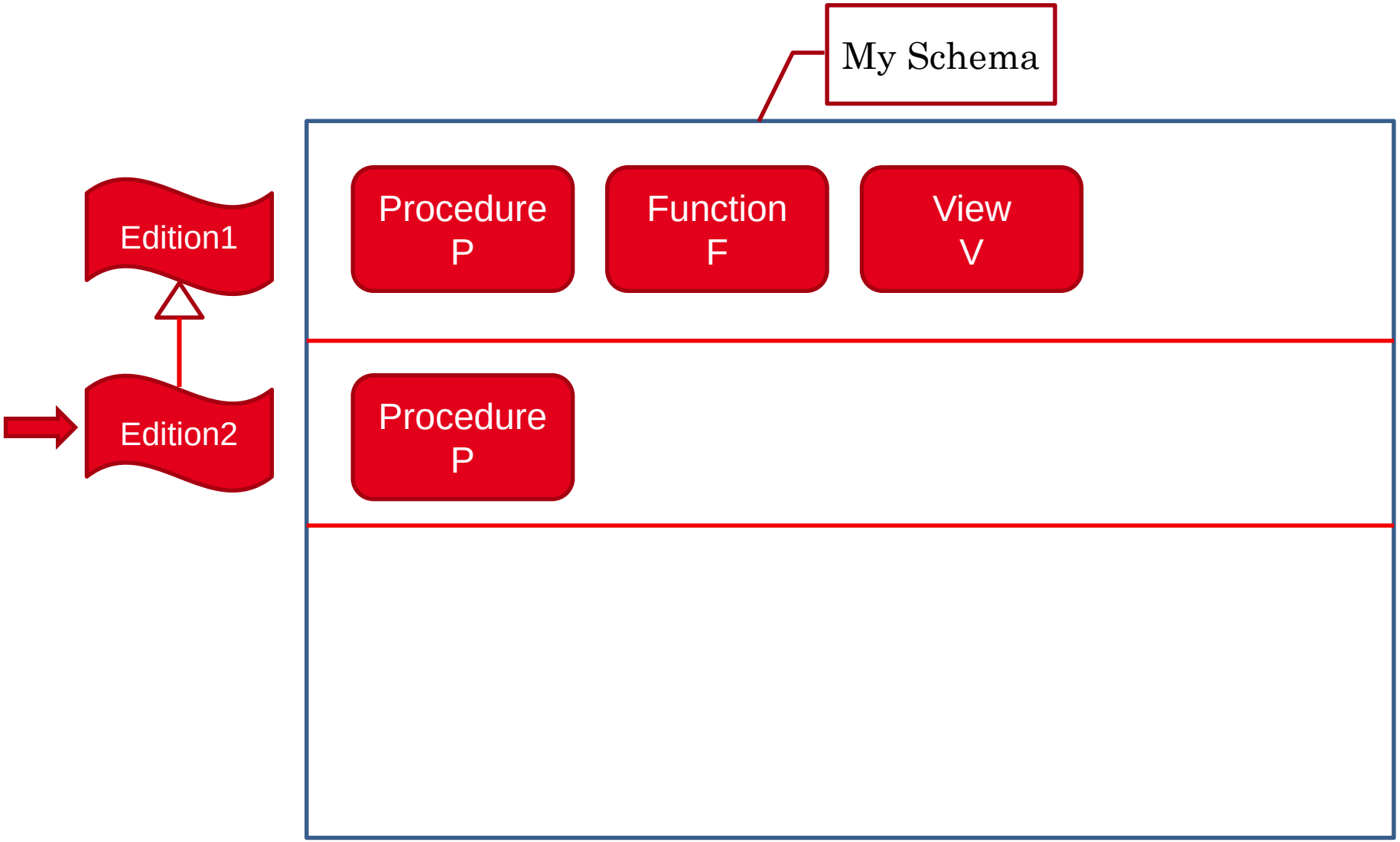
Introduced in 11.2

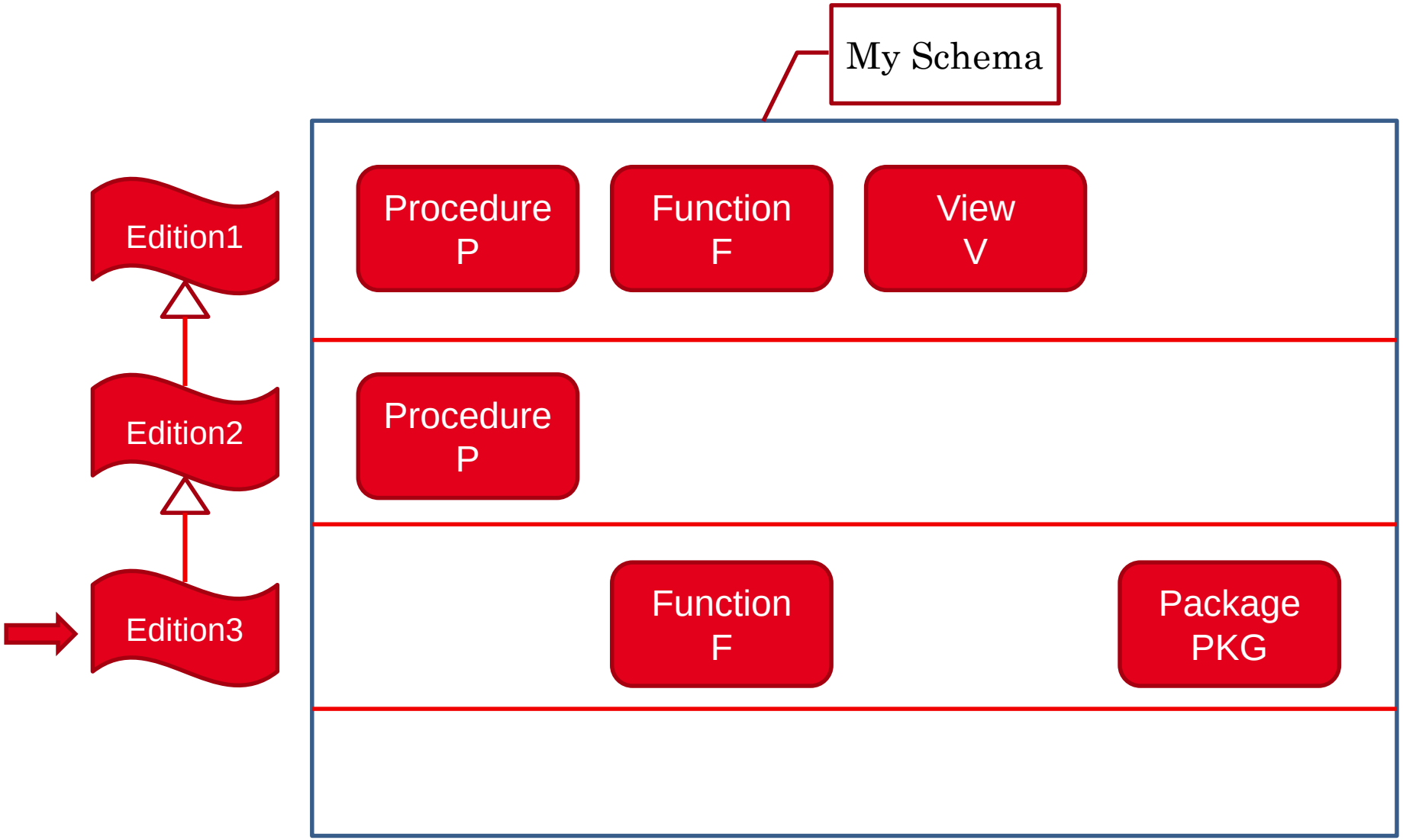
Supported in all editions

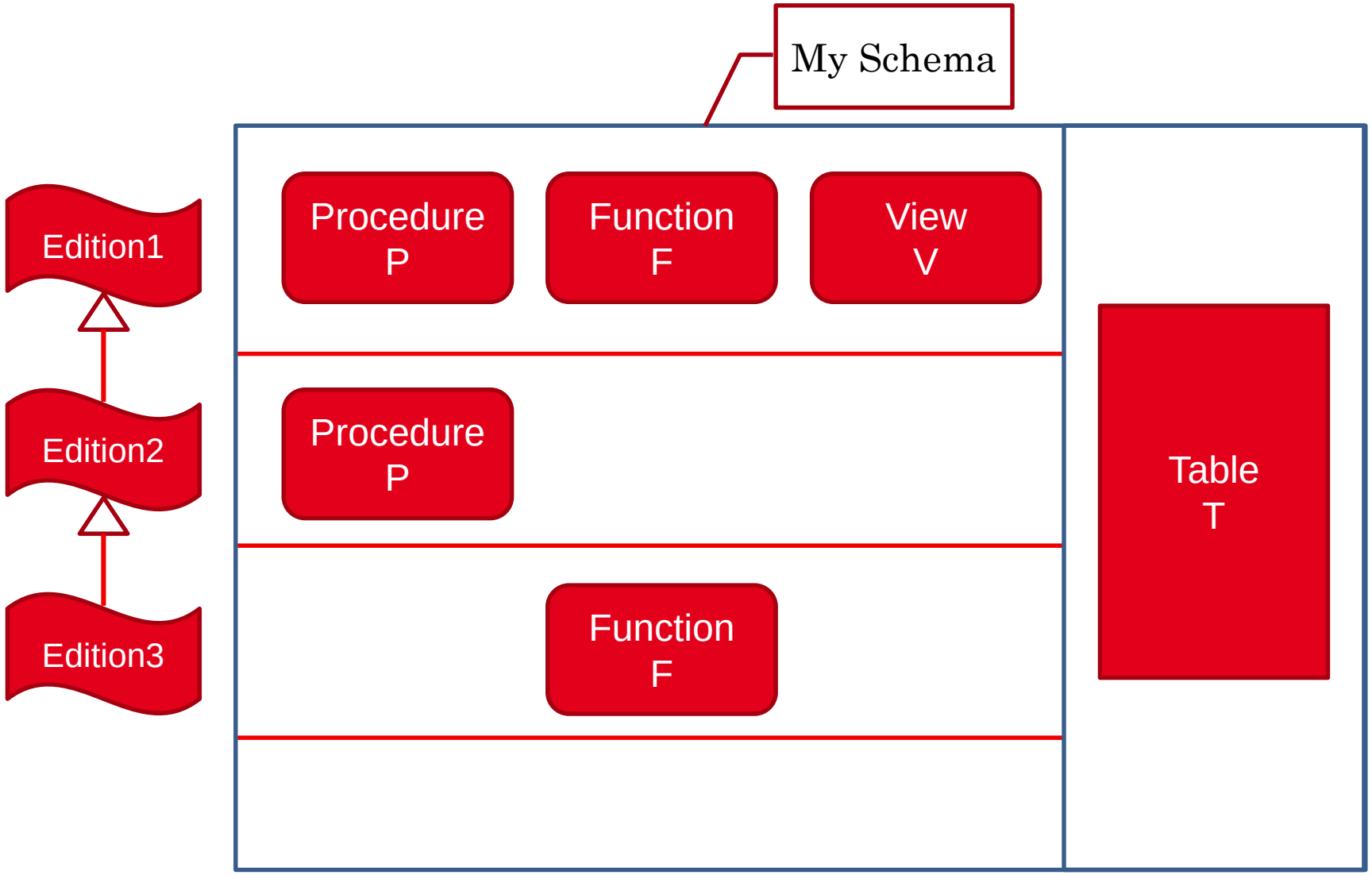
"The Killer Feature of Oracle 11g Release 2"
(Tom Kyte, Oracle Magazine, January 2010)

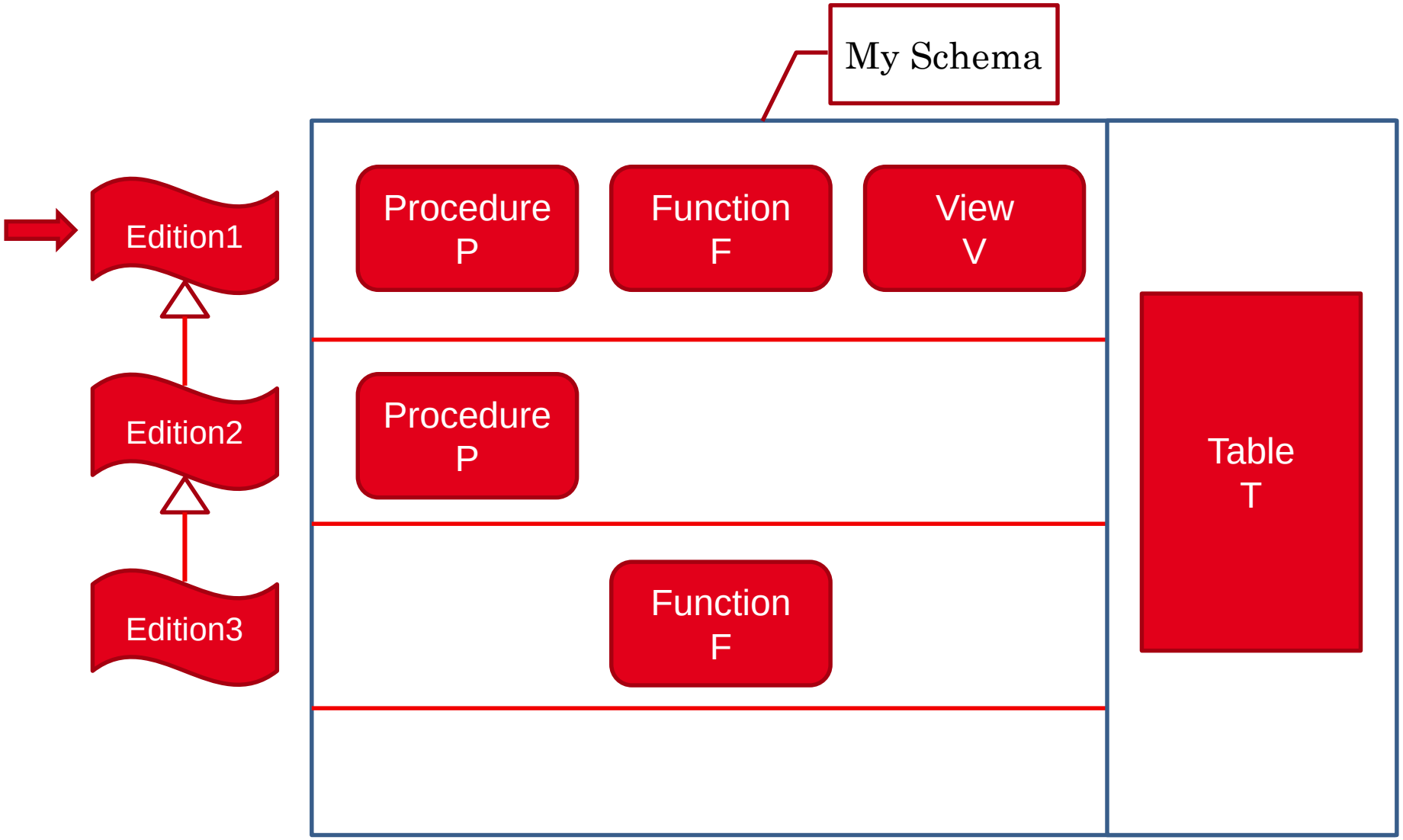
No extra license/option

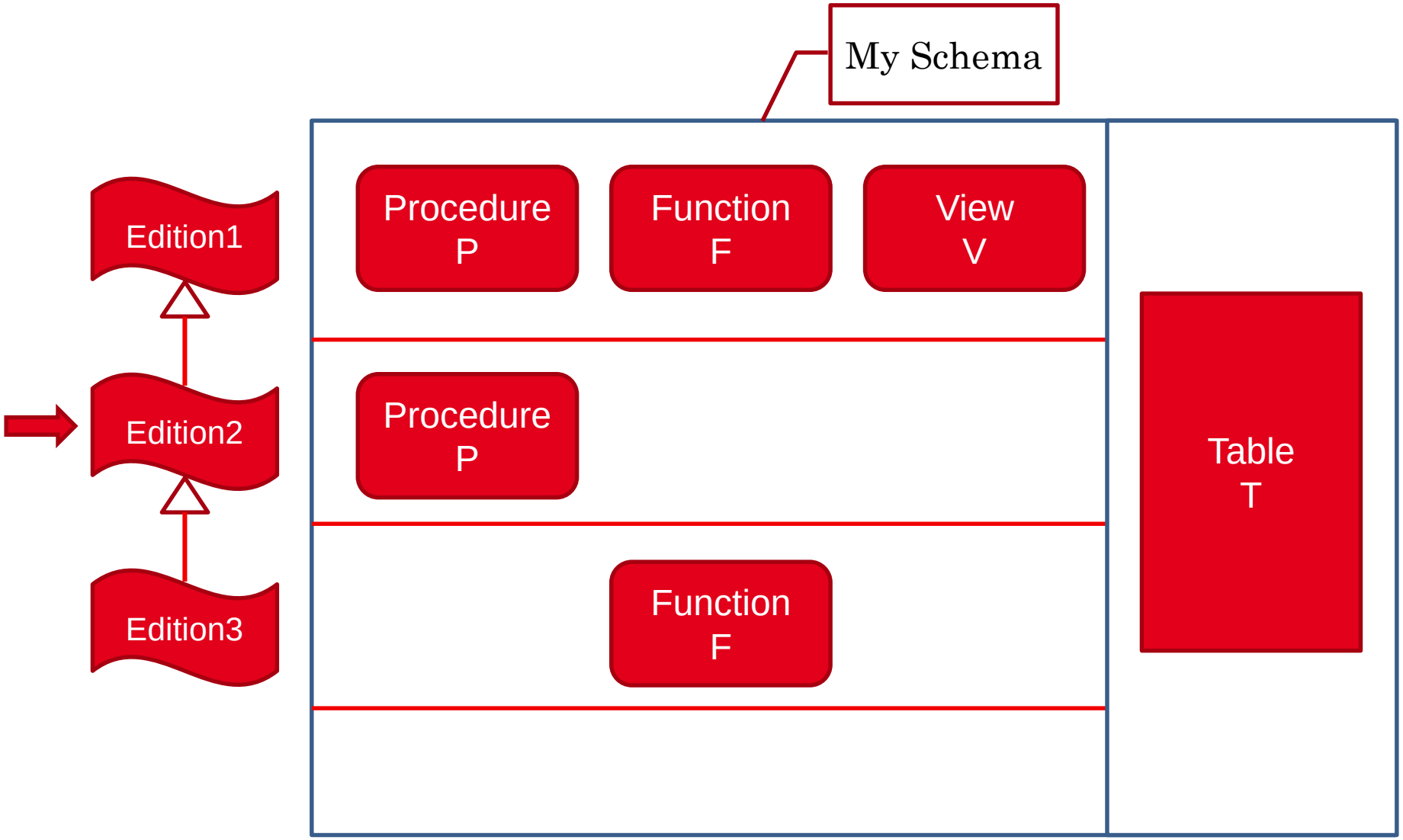


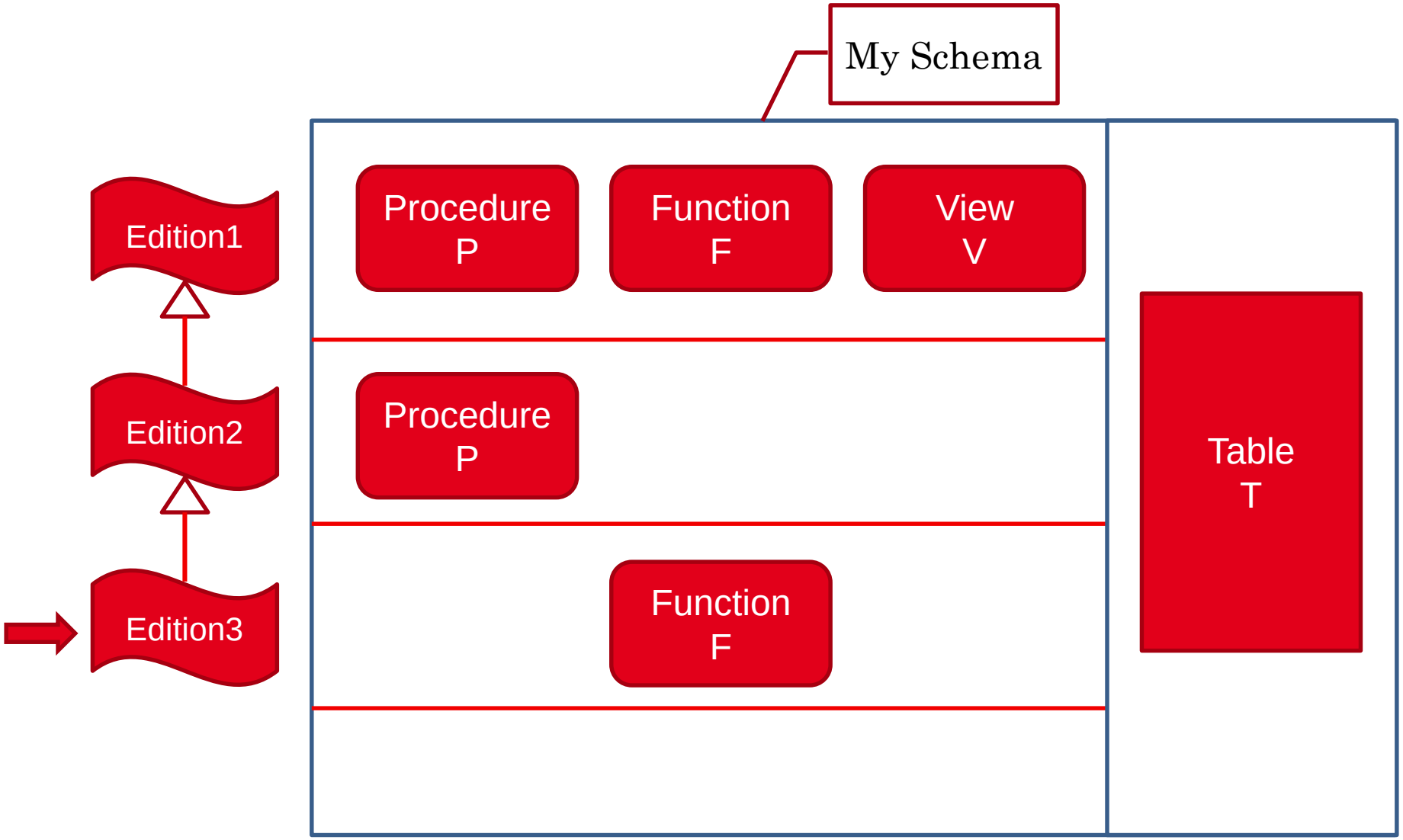








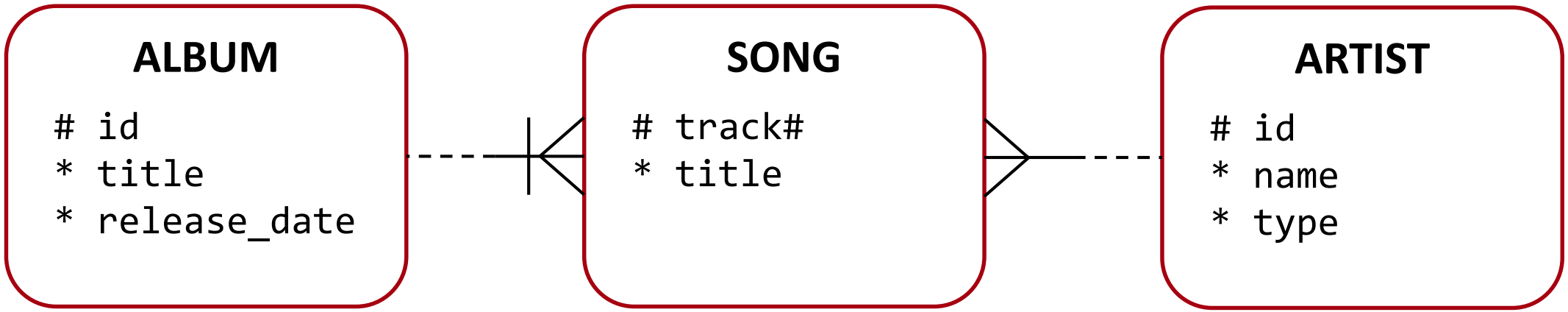




Demo

Entity Relationship Diagram

No need to think of EBR



APIs


```
add_artist
add_album
get_albums
get_album_songs
```

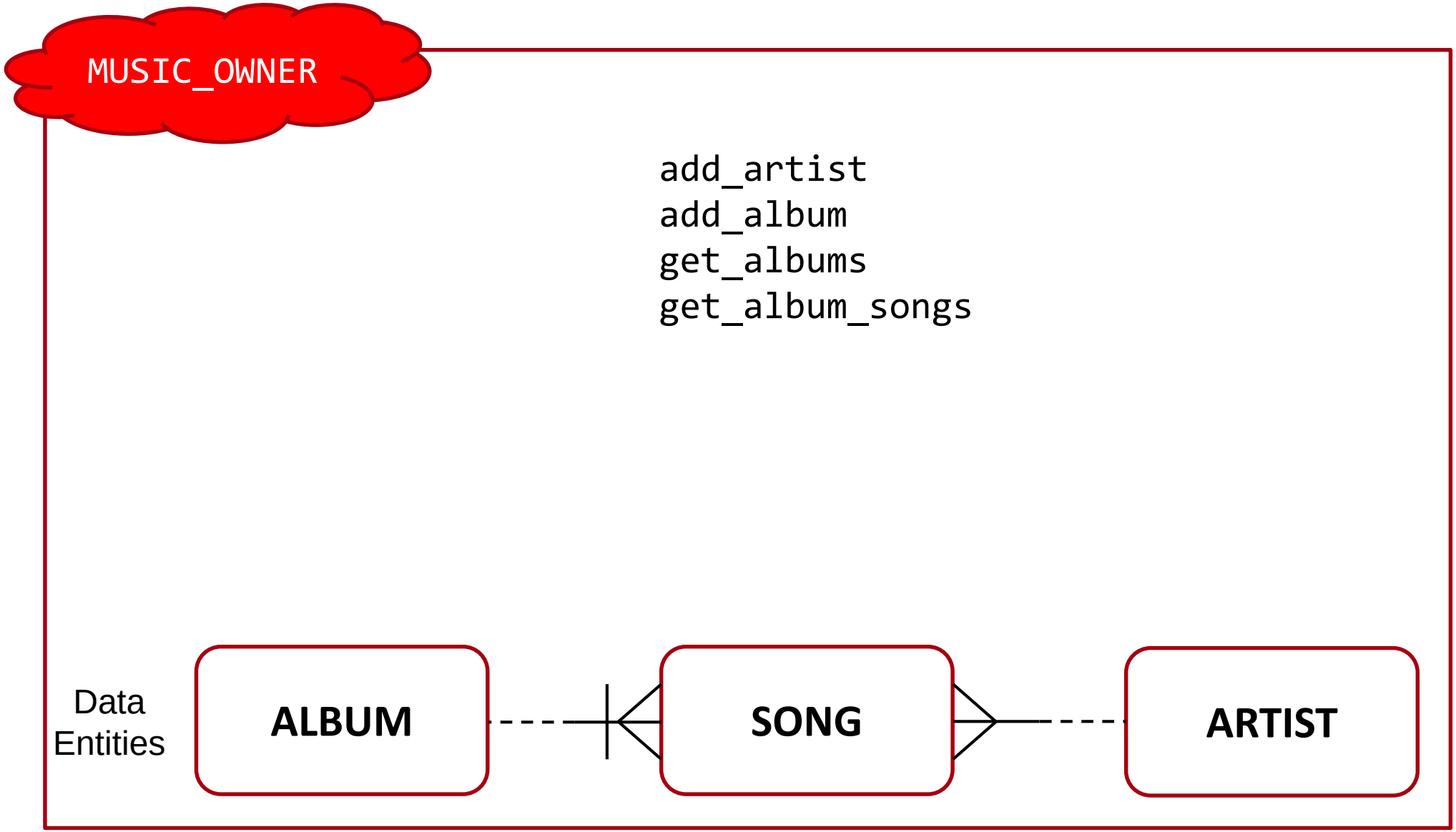
Data Entities

High Level Design

```
add_artist
add_album
get_albums
get_album_songs
```

Data Entities

High Level Design



High Level Design

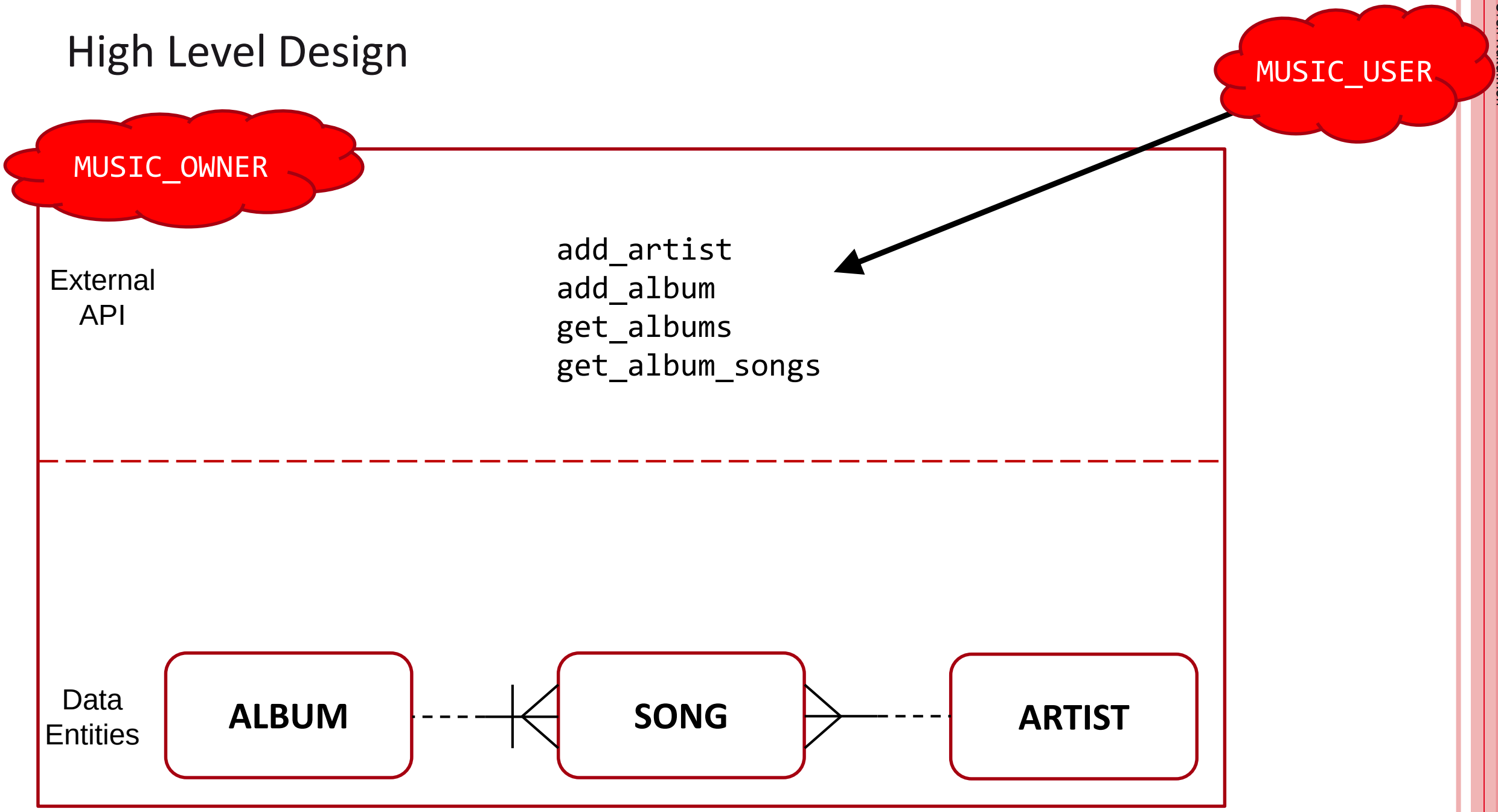
MUSIC_USER

MUSIC_OWNER

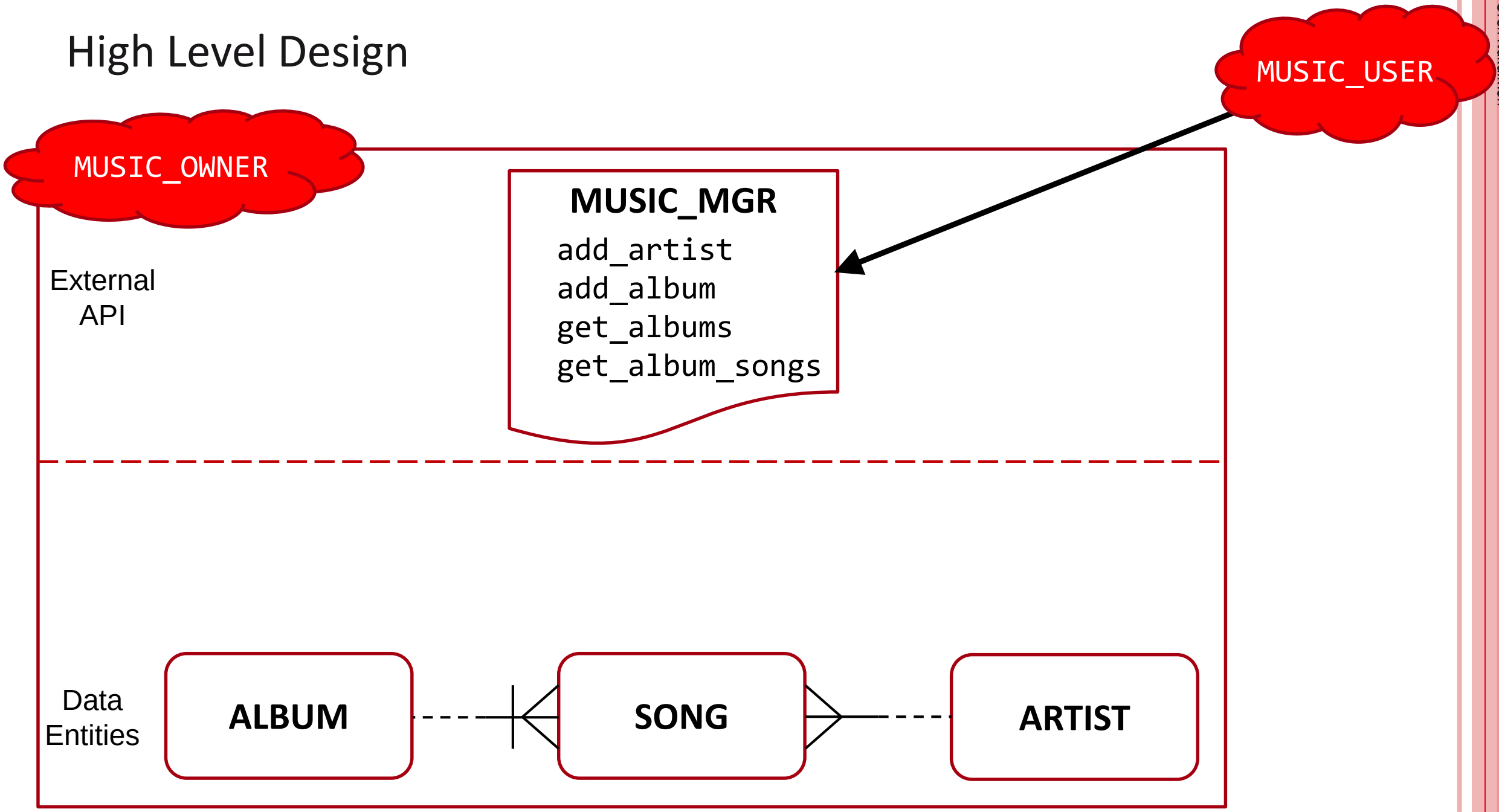
```
add_artist
add_album
get_albums
get_album_songs
```

Data Entities

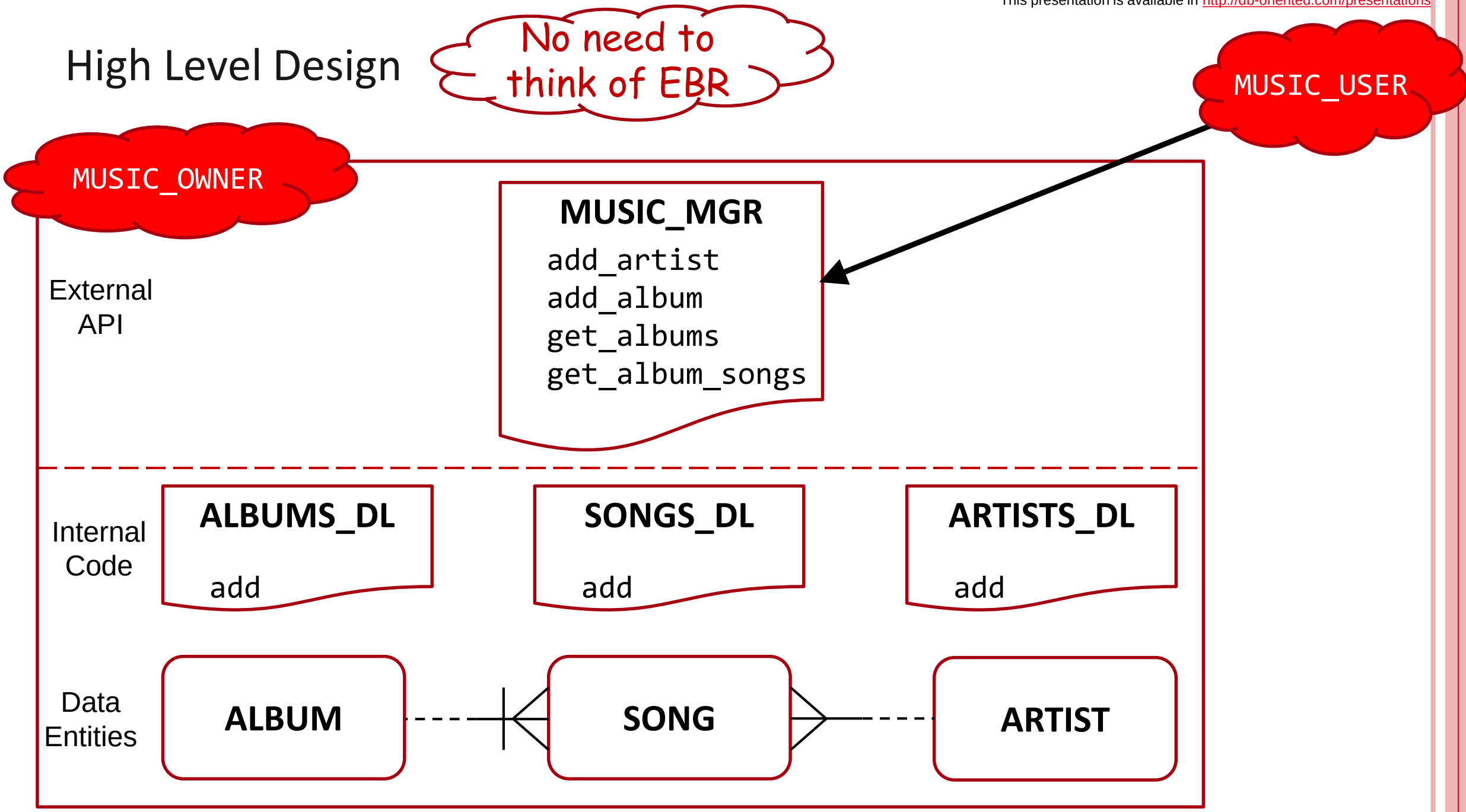
High Level Design



High Level Design



High Level Design



One-Time Setup


```
create user music_owner identified by pwd;
```

```
grant
  create session,
  create table,
  create view,
  create procedure,
  create sequence,
  create type,
  create trigger,
  create job,
  create synonym,
  unlimited tablespace
to
  music_owner;
```


```
create user music_user identified by pwd;
```

```
grant
  create session
to
  music_user;
```


```
alter user music_owner enable editions;
```

Version #1

The Upgrade
(or initial installation)

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition


```
create edition v1;
```


```
create edition v1;
```

```
grant use on edition v1 to music_owner;
```

The Upgrade

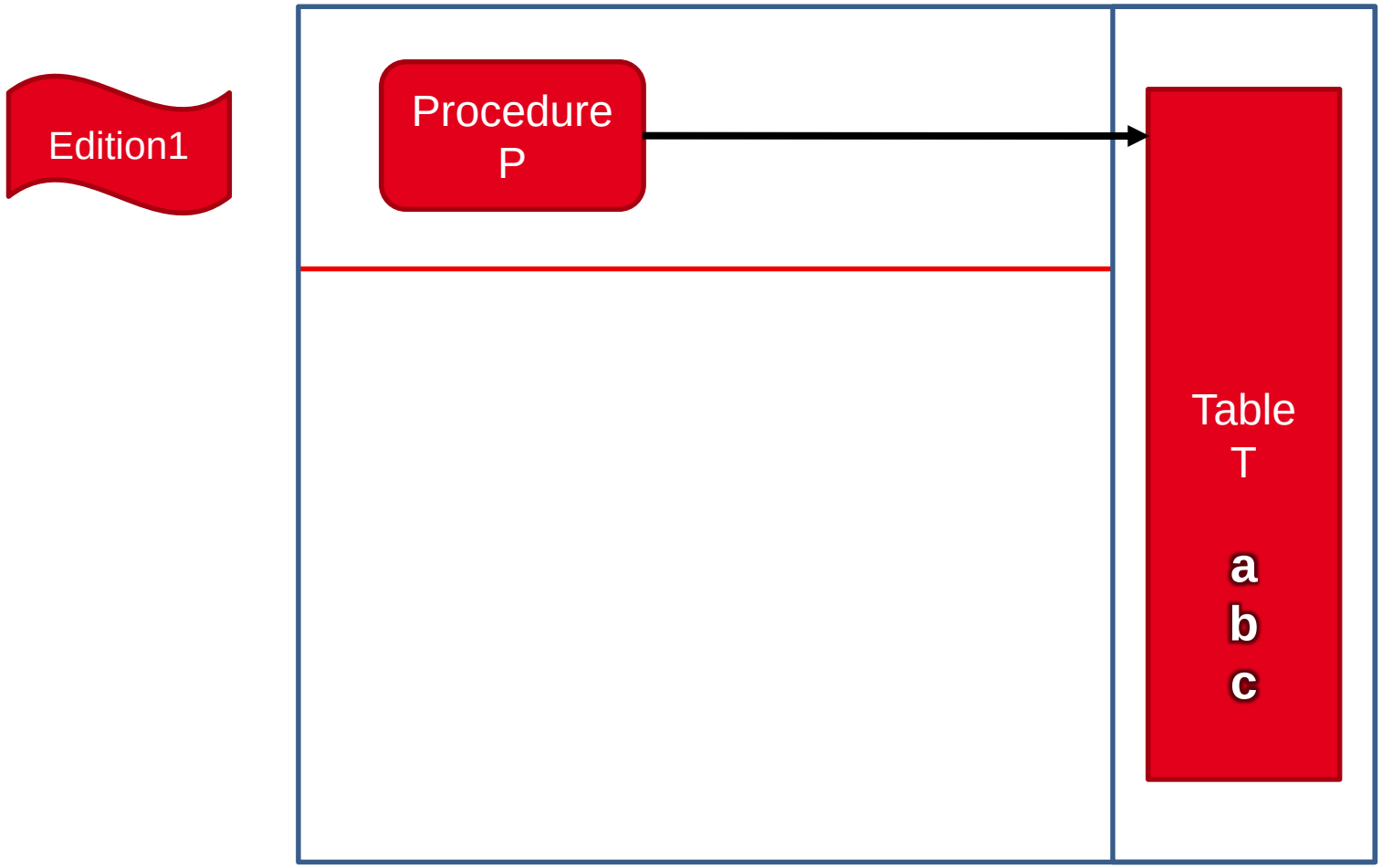
Create a New Edition

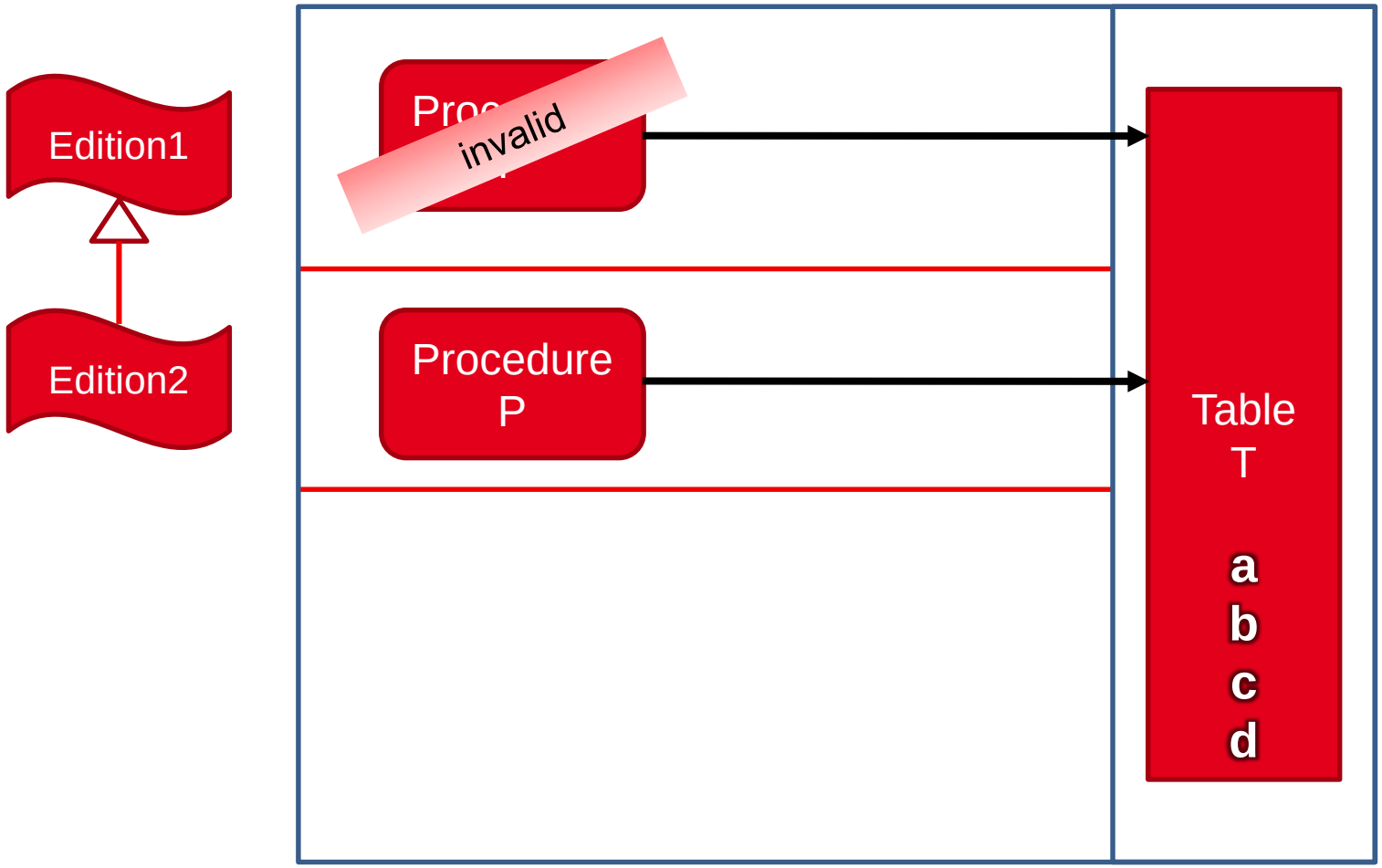
Create/Alter Non-Editioned Objects

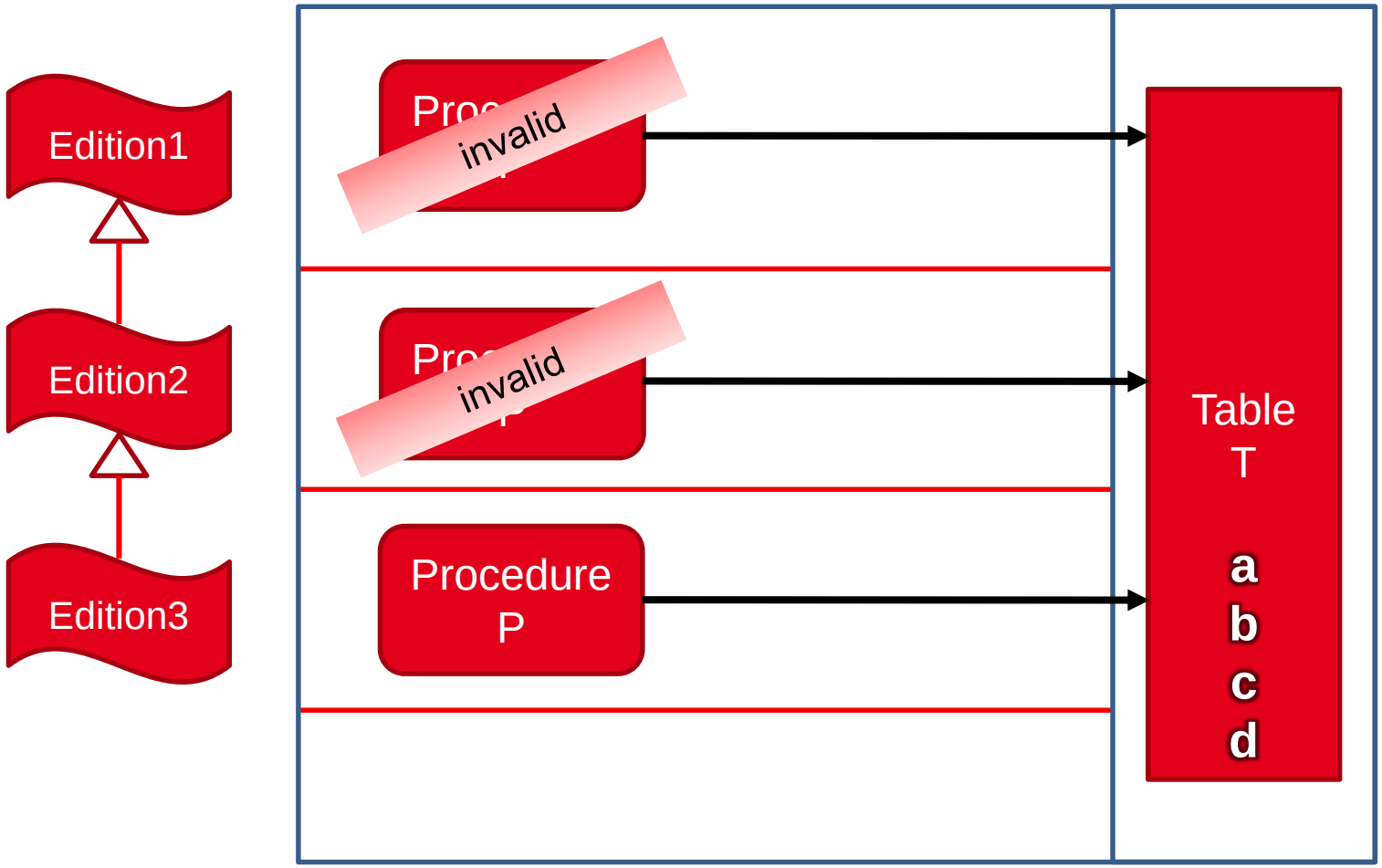
Create/Replace Editioned Objects in the New Edition

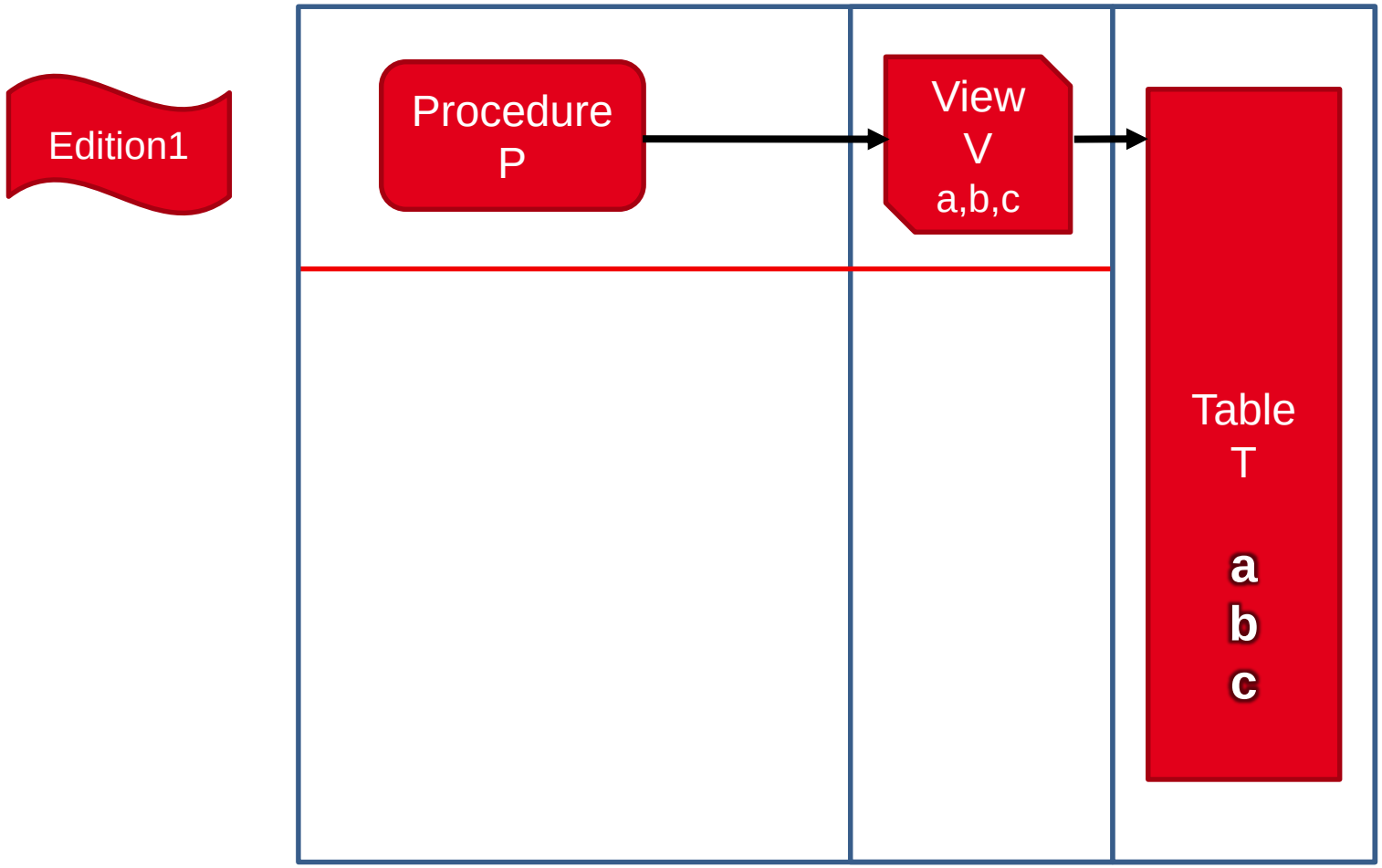
Expose the New Edition

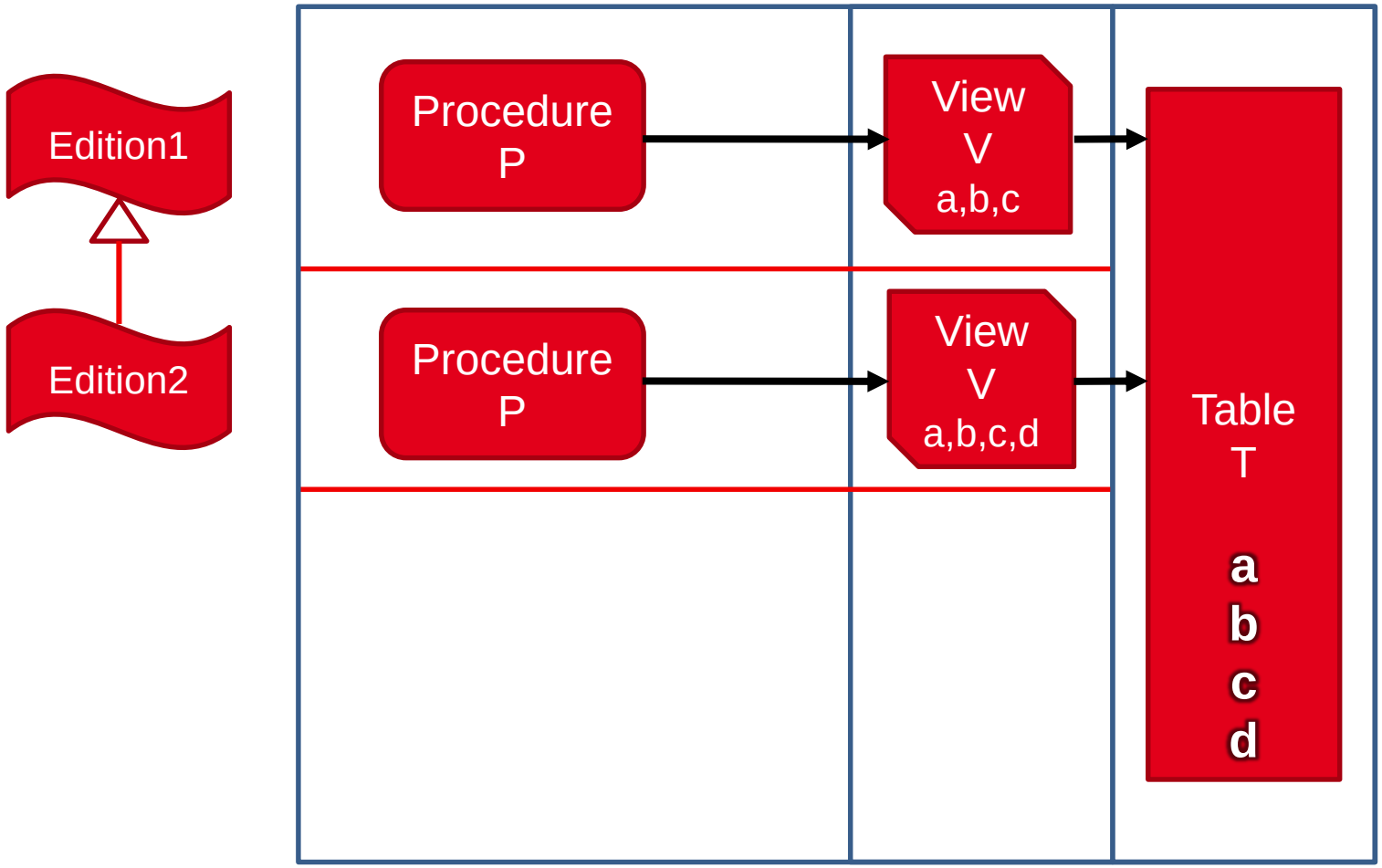
```
alter session set edition=v1;
```

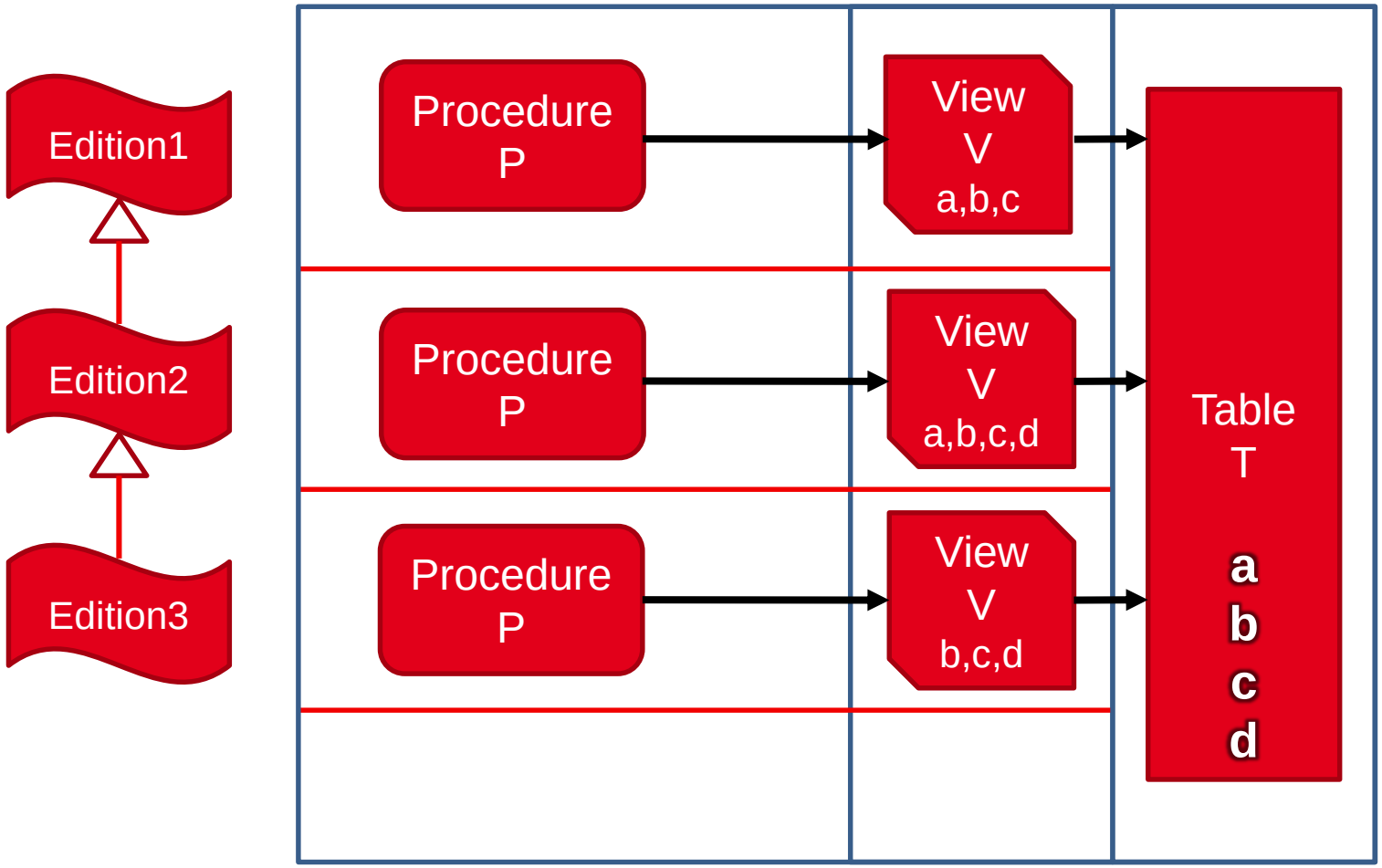




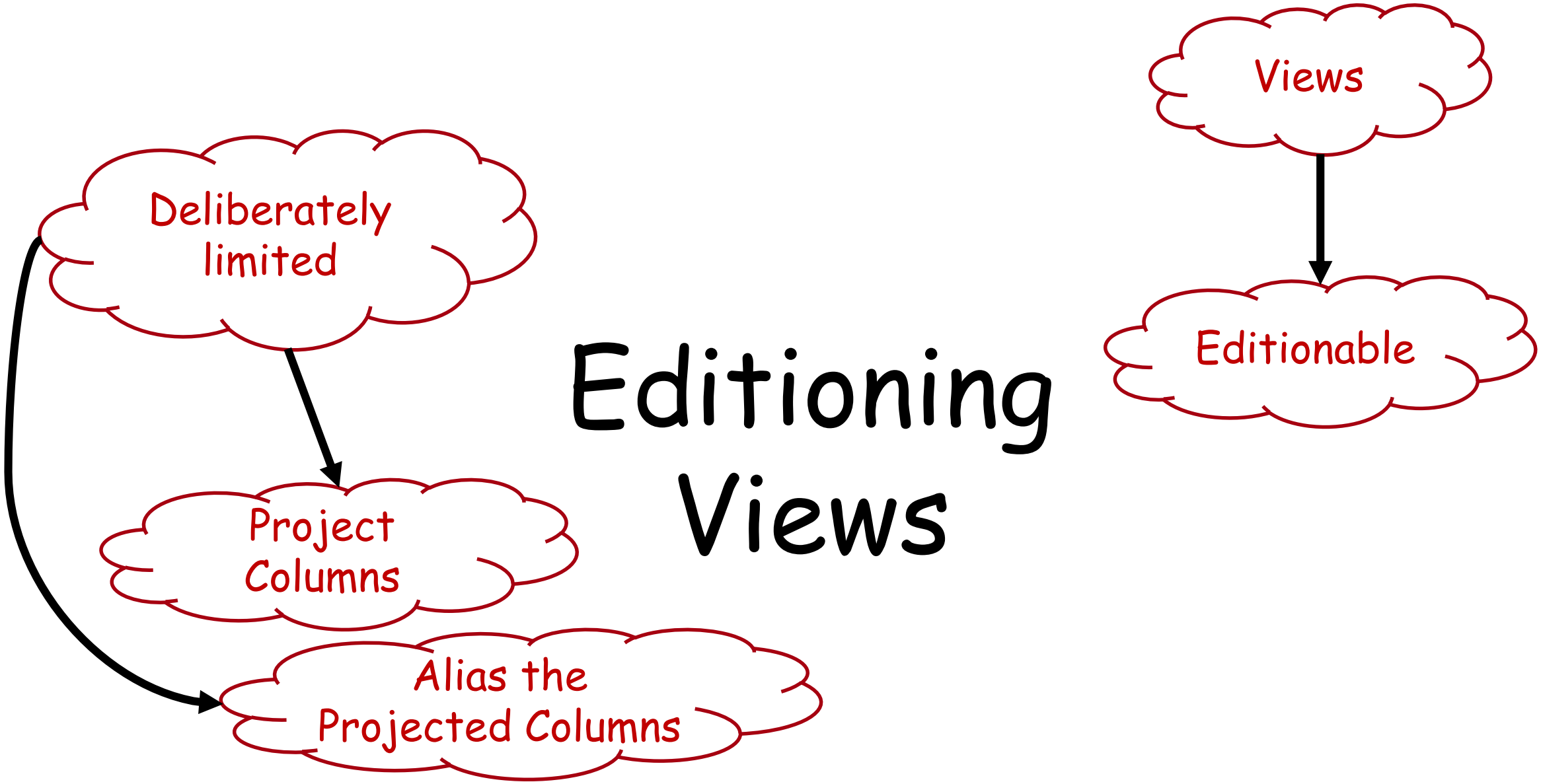








Editioning Views



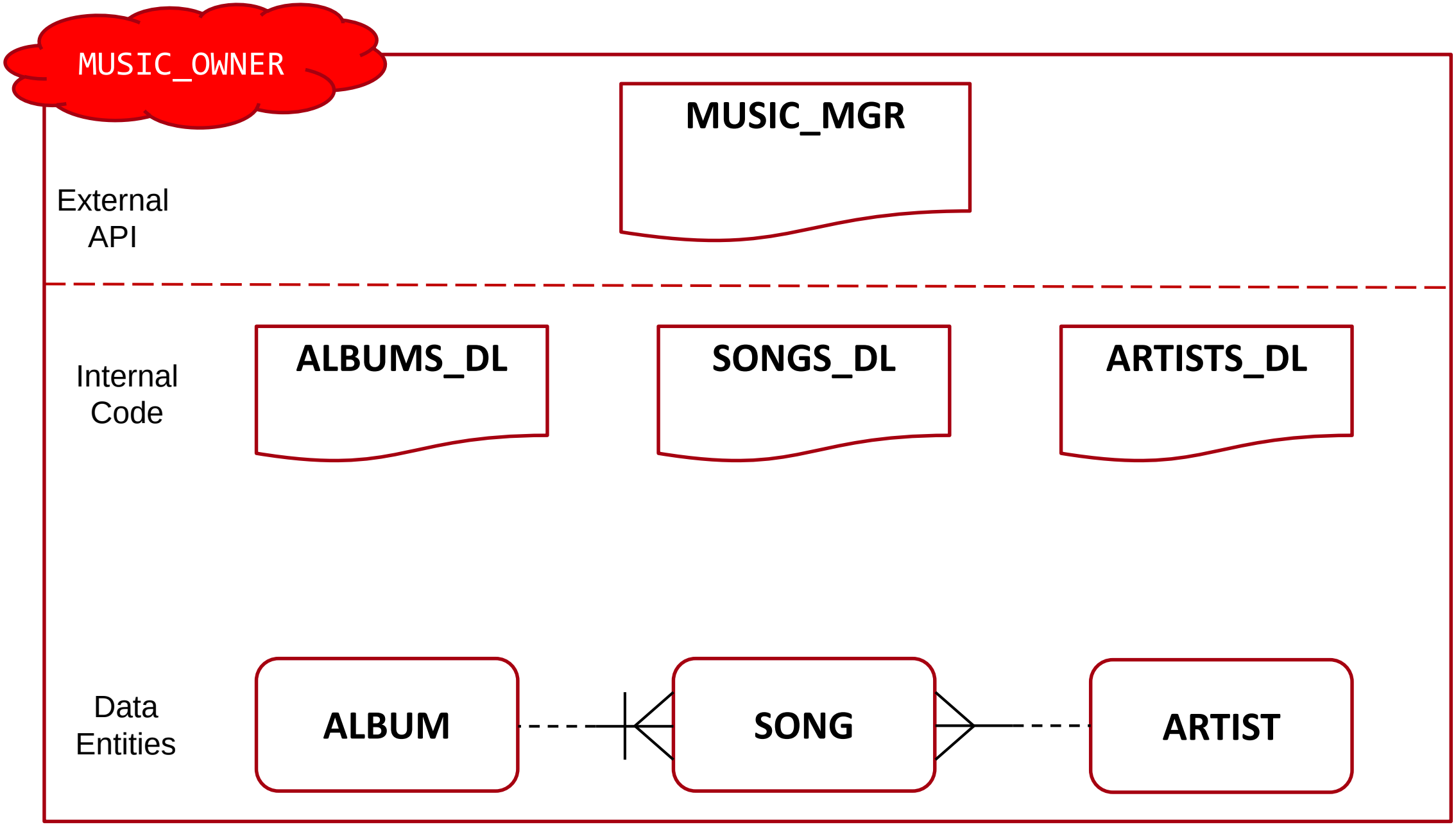
Simulate different table structures in different editions

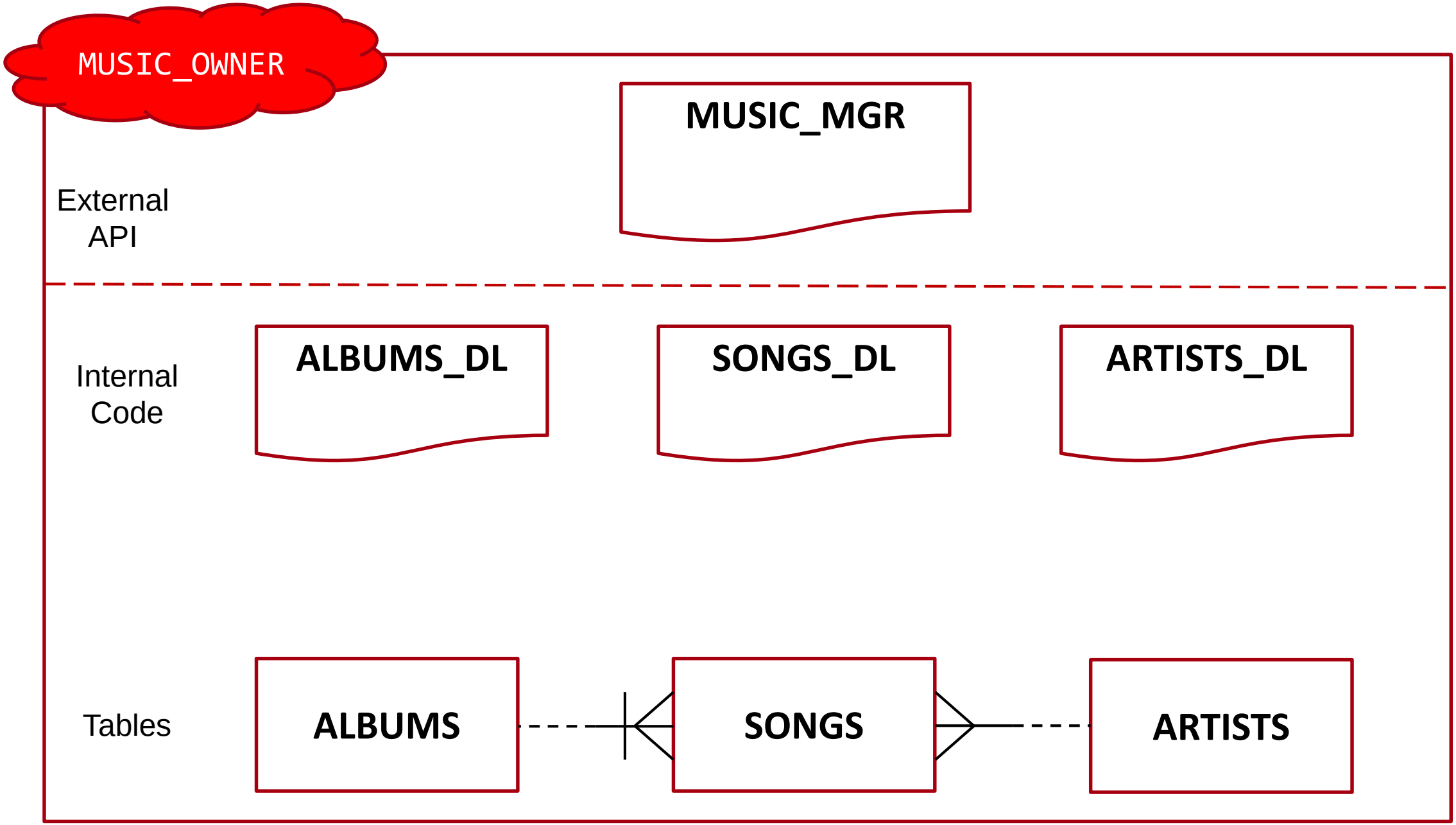
Editioning Views

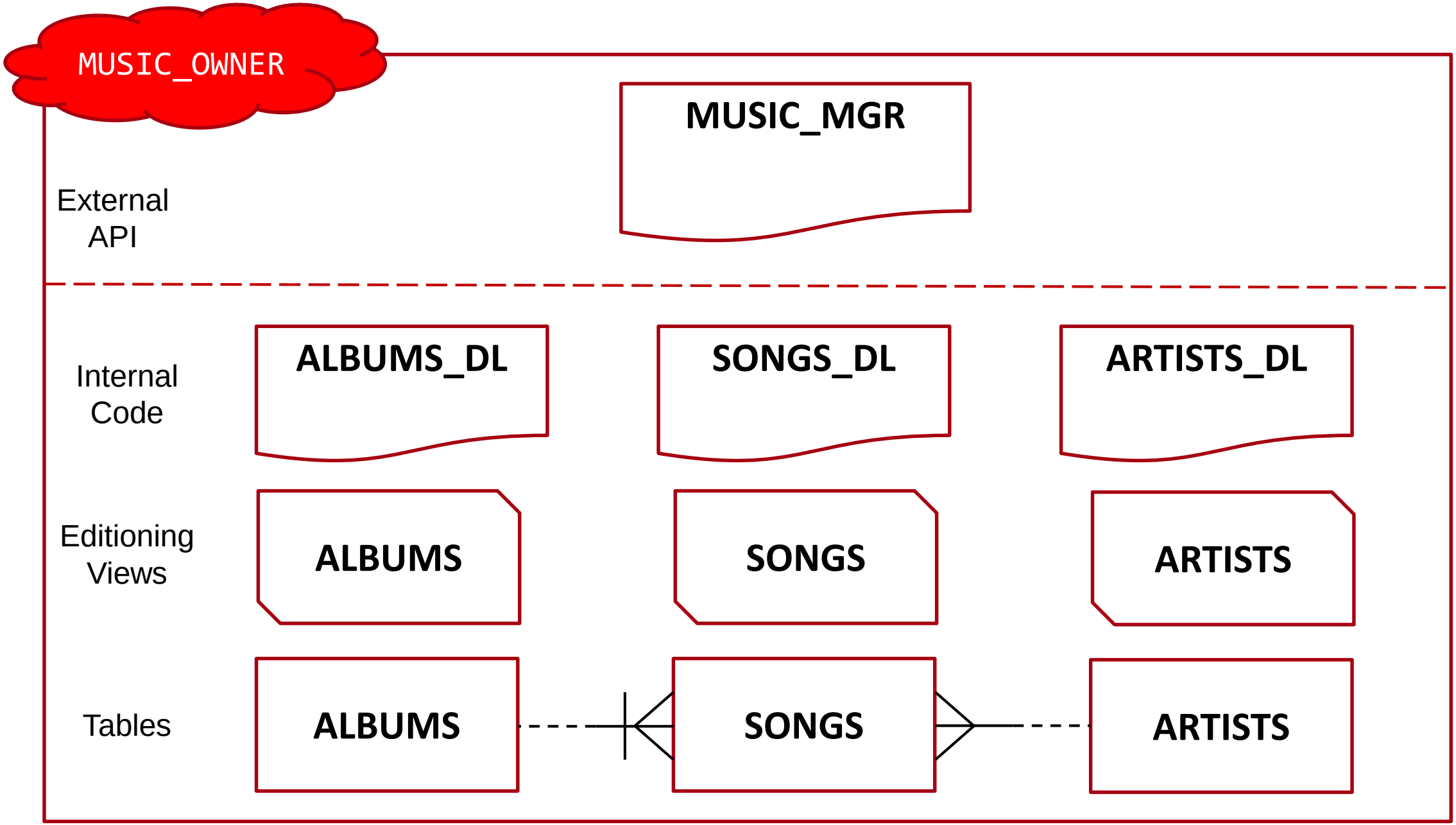
Prevent Invalidations

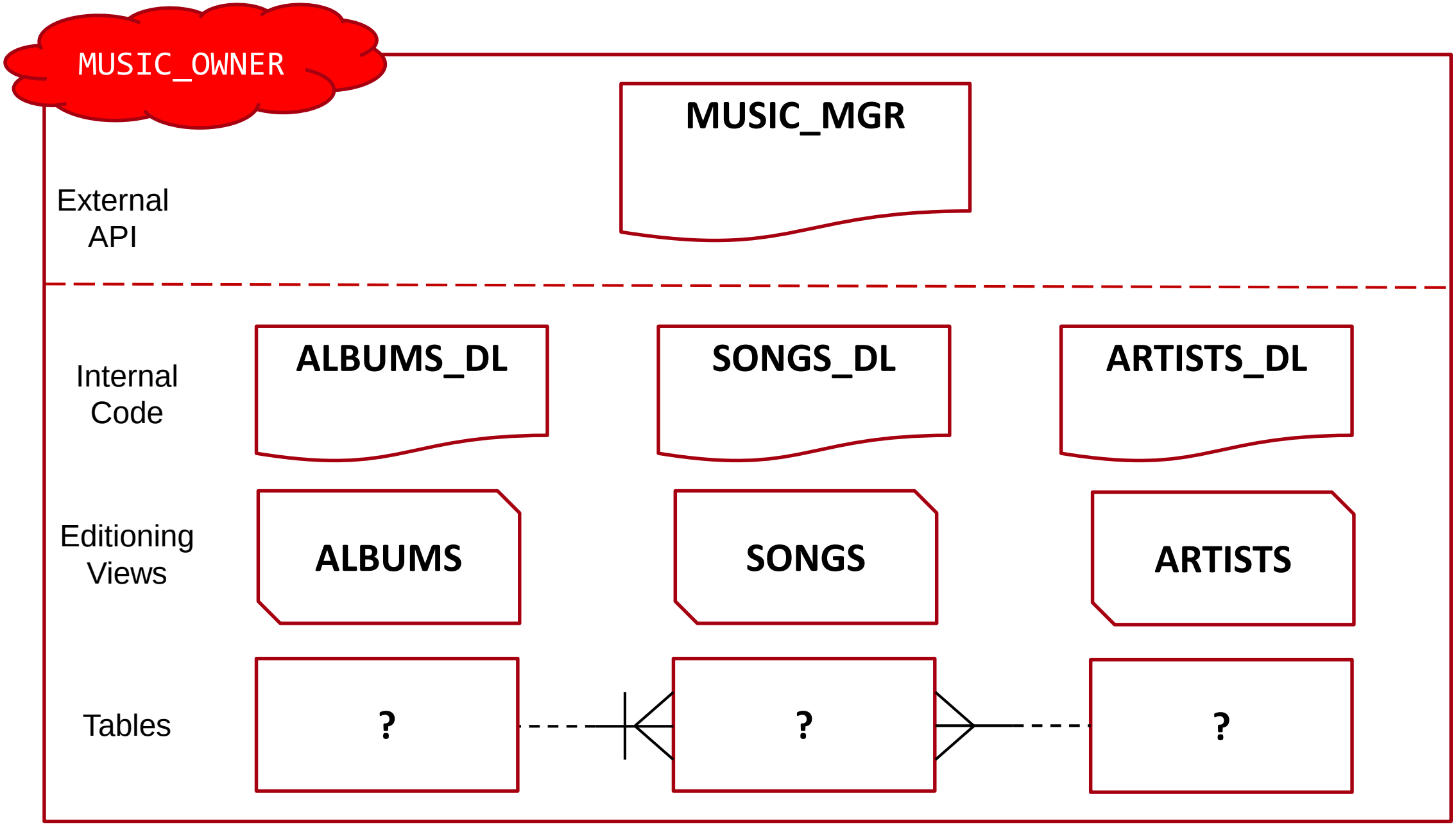
Editioning Views are the Interface between the Application and the Tables

Never refer to tables in your code, only to editioning views



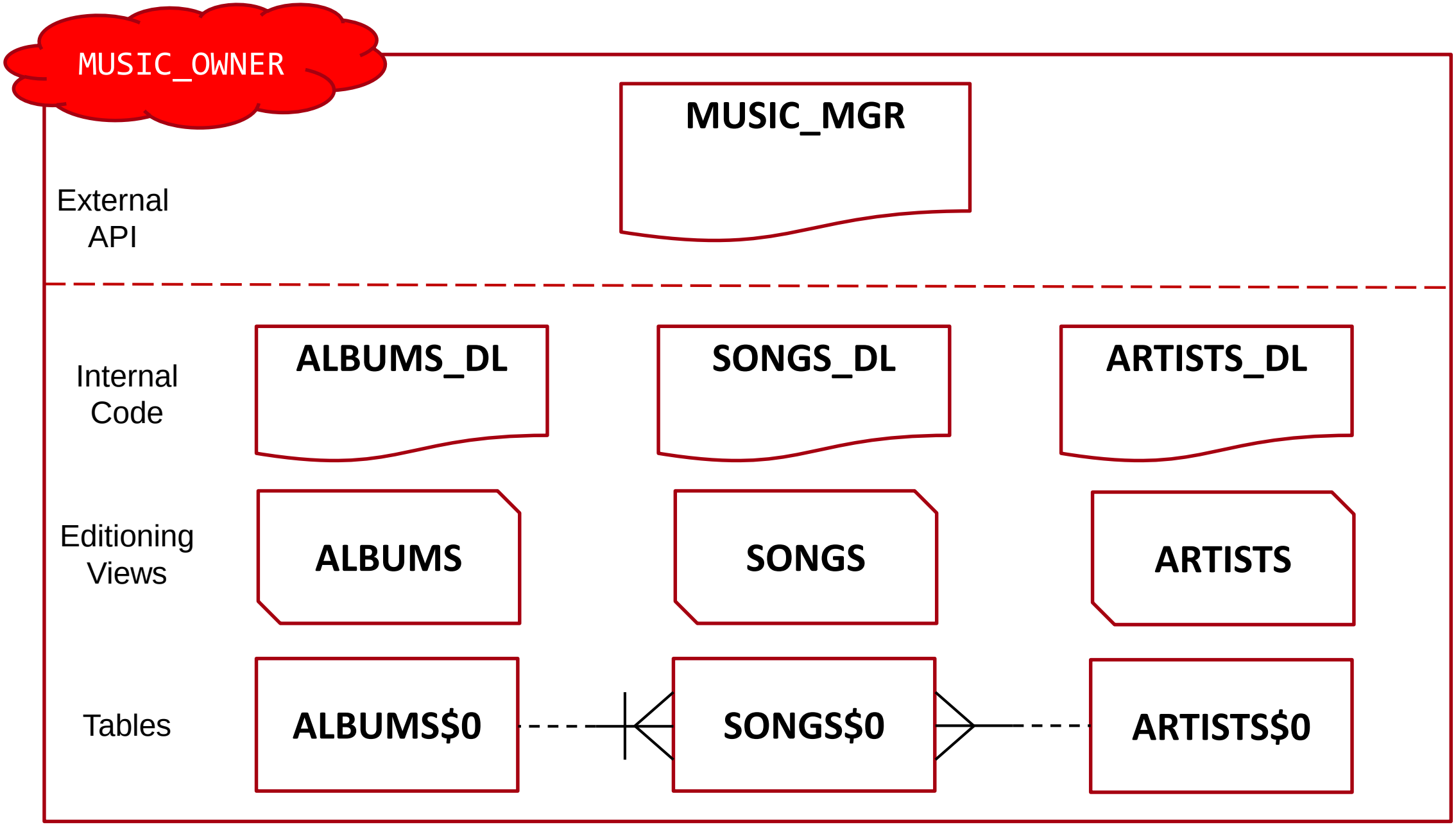






Define **Your** Naming Convention

Adhere to Your Naming Convention



MUSIC_OWNER

```
create table artists$0 (
  id
    integer
    generated as identity
    not null
    constraint artists$0_pk primary key,
  name
    varchar2(100)
    not null,
  type
    varchar2(6)
    not null
    constraint artists$0_chk_type check (type in ('Person','Band'))
);
```

MUSIC_OWNER

```
create editioning view artists as
  select
    id,
    name,
    type
  from
    artists$0;
```

MUSIC_OWNER

```
create table albums$0 (
  id
    integer
    generated as identity
    not null
    constraint albums$0_pk primary key,
  title
    varchar2(100)
    not null,
  release_date
    date
    not null
);
```

MUSIC_OWNER

```
create editioning view albums as
  select
    id,
    title,
    release_date
  from
    albums$0;
```

MUSIC_OWNER

```
create table songs$0 (
  album_id
    not null
    constraint songs$0_fk_album_id references albums$0,
  track#
    number(2)
    not null,
    constraint songs$0_pk primary key (album_id,track#),
  title
    varchar2(100)
    not null,
  artist_id
    not null
    constraint songs$0_fk_atrist_id references artists$0
);
```

```
create index songs$0_artist_id_ix on songs$0 (artist_id);
```

MUSIC_OWNER

```
create editioning view songs as
  select
    album_id,
    track#,
    title,
    artist_id
  from
    songs$0;
```

MUSIC_OWNER

```
create or replace package artists_dl
as
  procedure add
  (
    i_name in artists.name%type,
    i_type in artists.type%type,
    o_id   out artists.id%type
  );
end artists_dl;
/
```

```
create or replace package body artists_dl
as
  procedure add
  (
    i_name in artists.name%type,
    i_type in artists.type%type,
    o_id   out artists.id%type
  ) is
  begin
    insert into artists
      (name,
       type)
    values
      (i_name,
       i_type)
    returning id into o_id;
  end add;
end artists_dl;
```


```
create or replace package albums_dl as
  procedure add
  (
    i_title        in albums.title
    i_release_date in albums.relea
    o_id           out albums.id%t
  );
end albums_dl;
```

```
create or replace package body albums_dl as
  procedure add
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    o_id           out albums.id%type
  ) is
  begin
    insert into albums
      (title,
       release_date)
    values
      (i_title,
       i_release_date)
    returning id into o_id;
  end add;
end albums_dl;
```

```
create type song_t as object (
  track# number(2),
  title varchar2(100),
  artist_id integer
)
/
```

```
create type song_tt as
  table of song_t
/
```

```
create or replace package songs_dl as
  procedure add
  (
    i_album_id in songs.album_id%type,
    i_songs    in song_tt
  );
end songs_dl;
```


```
create or replace package body songs_dl as
  procedure add
  (
    i_album_id in songs.album_id%type,
    i_songs    in song_tt
  ) is
  begin
    forall i in indices of i_songs
      insert into songs
        (album_id,
         track#,
         title,
         artist_id)
       values
         (i_album_id,
          i_songs(i).track#,
          i_songs(i).title,
          i_songs(i).artist_id);
  end add;
end songs_dl;
```

MUSIC_OWNER

```
create or replace package music_mgr as

    procedure add_artist
    (
        i_name in artists.name%type,
        i_type in artists.type%type,
        o_id   out artists.id%type
    );

    procedure add_album
    (
        i_title        in albums.title%type,
        i_release_date in albums.release_date%type,
        i_songs        in song_tt,
        o_id           out albums.id%type
    );

    procedure get_albums
    (
        o_albums out sys_refcursor
    );

    procedure get_album_songs
    (
        i_album_id in albums.id%type,
        o_songs    out sys_refcursor
    );

end music_mgr;
```

```
create or replace package music_mgr as

    procedure add_artist
    (
        i_name in artists.name%type,
        i_type in artists.type%type,
        o_id   out artists.id%type
    );

    procedure add_album…

    procedure get_albums…

    procedure get_album_songs…

end music_mgr;
```

MUSIC_OWNER

```
create or replace package body music_mgr as

    procedure add_artist
    (
        i_name in artists.name%type,
        i_type in artists.type%type,
        o_id   out artists.id%type
    ) is
    begin
        artists_dl.add(i_name => i_name, i_type => i_type, o_id => o_id);
        commit;
    end add_artist;


.
.
.
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
 procedure add_album
 (
   i_title        in albums.title%type,
   i_release_date in albums.release_date%type,
   i_songs        in song_tt,
   o_id           out albums.id%type
 ) is
 begin
   albums_dl.add(i_title => i_title,
                 i_release_date => i_release_date,
                 o_id => o_id);
   songs_dl.add(i_album_id => o_id, i_songs => i_songs);
   commit;
 end add_album;
.
.
```

```
create or replace package body music_mgr as
.
.
    procedure get_albums
    (
        o_albums out sys_refcursor
    ) is
    begin
        open o_albums for
            select a.id,
                   a.title,
                   a.release_date
            from   albums a
            order by a.title;
    end get_albums;
.
.
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
    procedure get_album_songs
    (
        i_album_id in albums.id%type,
        o_songs    out sys_refcursor
    ) is
    begin
        open o_songs for
            select s.track#,
                   s.title,
                   s.artist_id
            from   songs s
            where  s.album_id = i_album_id
            order by s.track#;
    end get_album_songs;

end music_mgr;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition

Exposing the Post-Upgrade Version

- Validate (and actualize) necessary objects
- Grant permissions on new objects to the application user
- Create necessary synonyms

- Grant use on the new edition to the application user
- Associate the new edition with a service

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

Grant use on the new edition to the application user

Associate the new edition with a service


```
exec dbms_utility.compile_schema(user,compile_all=>false)
```

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

Grant use on the new edition to the application user

Associate the new edition with a service


```
grant execute on music_mgr to music_user;
grant execute on song_t to music_user;
grant execute on song_tt to music_user;
```

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

Grant use on the new edition to the application user

Associate the new edition with a service


```
create synonym music_user.music_mgr for music_owner.music_mgr;
create synonym music_user.song_t for music_owner.song_t;
create synonym music_user.song_tt for music_owner.song_tt;
```

Exposing the Post-Upgrade Version

- Validate (and actualize) necessary objects
- Grant permissions on new objects to the application user
- Create necessary synonyms

- Grant use on the new edition to the application user
- Associate the new edition with a service


```
grant use on edition v1 to music_user;
```

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

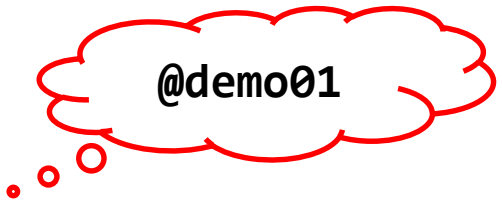
Grant permissions on new objects to the application user

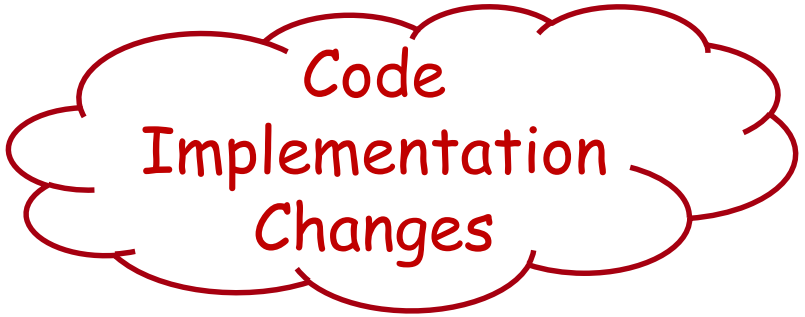
Create necessary synonyms

Grant use on the new edition to the application user

Associate the new edition with a service


```
begin

  dbms_service.create_service(
    service_name => 'MUSIC_SERVICE_A',
    network_name => 'MUSIC_SERVICE_A',
    edition => 'V1'
  );

  dbms_service.start_service(
    service_name => 'MUSIC_SERVICE_A‘
  );

end;
```





Version #2

Change **get_album_songs** to return **artist_name** instead of **artist_id**

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition


```
create edition v2;
```


```
create edition v2;
```

```
grant use on edition v2 to music_owner;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition


```
alter session set edition=v2;
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
    procedure get_album_songs
    (
        i_album_id in albums.id%type,
        o_songs    out sys_refcursor
    ) is
    begin
        open o_songs for
            select s.track#,
                   s.title,
                   a.name artist_name
            from   songs s,
                   artists a
            where  s.album_id = i_album_id
            and    a.id = s.artist_id
            order  by s.track#;
    end get_album_songs;

end music_mgr;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

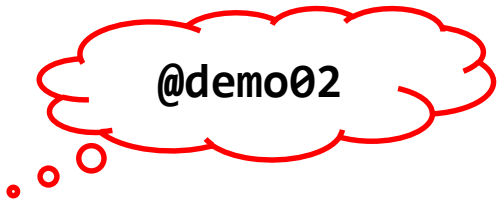
Grant use on the new edition to the application user

Associate the new edition with a service

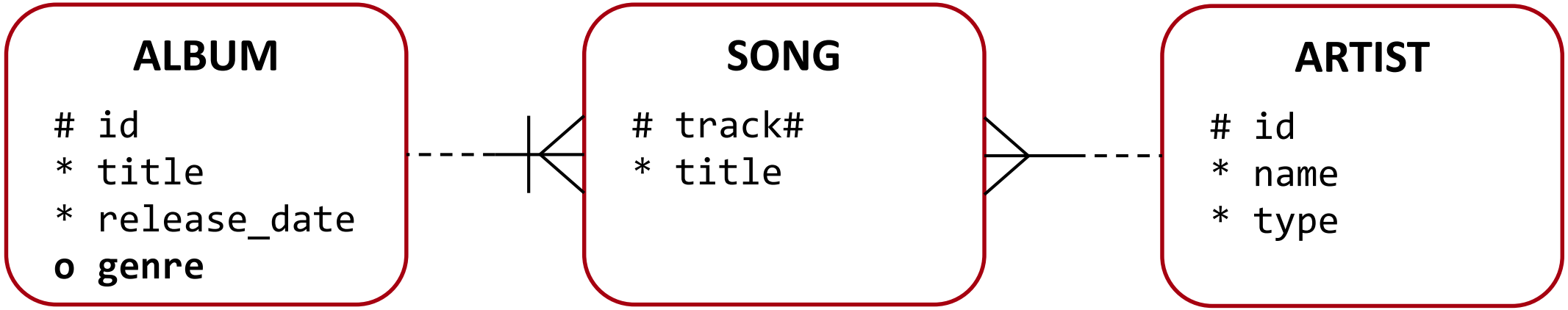

```
exec dbms_utility.compile_schema(user,compile_all=>false)
```

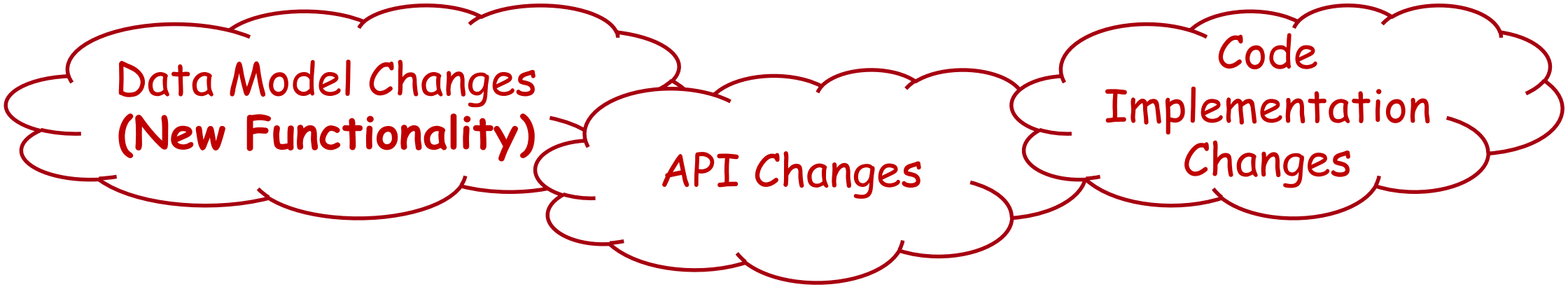

```
grant use on edition v2 to music_user;
```


```
begin

  dbms_service.modify_service(
    service_name => 'MUSIC_SERVICE_A',
    edition => 'V2',
    modify_edition => TRUE)
  );

end;
```



Entity Relationship Diagram





Version #3

Add **genre** to **albums**

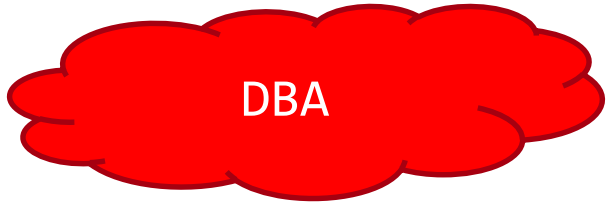
The Upgrade

Create a New Edition

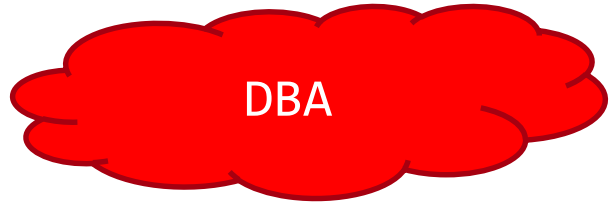
Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition




```
create edition v3;
```




```
create edition v3;
```

```
grant use on edition v3 to music_owner;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition


```
alter session set edition=v3;
```

MUSIC_OWNER

```
alter table albums$0 add (
  genre
    varchar2(100)
);
```

```
create or replace editioning view albums as
  select
    id,
    title,
    release_date,
    genre
  from
    albums$0;
```

MUSIC_OWNER

```
create or replace package albums_dl as
    procedure add
    (
        i_title        in albums.title%type,
        i_release_date in albums.release_date%type,
        o_id           out albums.id%type
    );

    procedure set_genre
    (
        i_id    in albums.id%type,
        i_genre in albums.genre%type
    );

end albums_dl;
/
```

```
create or replace package body albums_dl as
.
.
    procedure set_genre
    (
        i_id    in albums.id%type,
        i_genre in albums.genre%type
    ) is
    begin
        update albums a
        set    a.genre = i_genre
        where  a.id = i_id;
    end set_genre;

end albums_dl;
/
```

```
create or replace package music_mgr as
.
.

    procedure set_album_genre
    (
        i_album_id in albums.id%type,
        i_genre    in albums.genre%type
    );
.
.
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
    procedure set_album_genre
    (
        i_album_id in albums.id%type,
        i_genre    in albums.genre%type
    ) is
    begin
        albums_dl.set_genre(i_id => i_album_id, i_genre => i_genre);
        commit;
    end set_album_genre;
.
.
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
    procedure get_albums
    (
        o_albums out sys_refcursor
    ) is
    begin
        open o_albums for
            select a.id,
                   a.title,
                   a.release_date,
                   a.genre
            from   albums a
            order  by a.title;
    end get_albums;
.
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

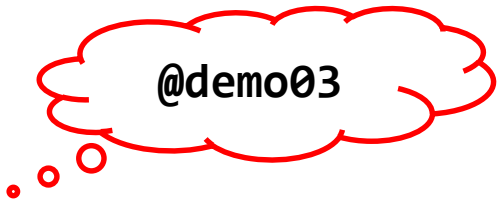
Grant use on the new edition to the application user

Associate the new edition with a service

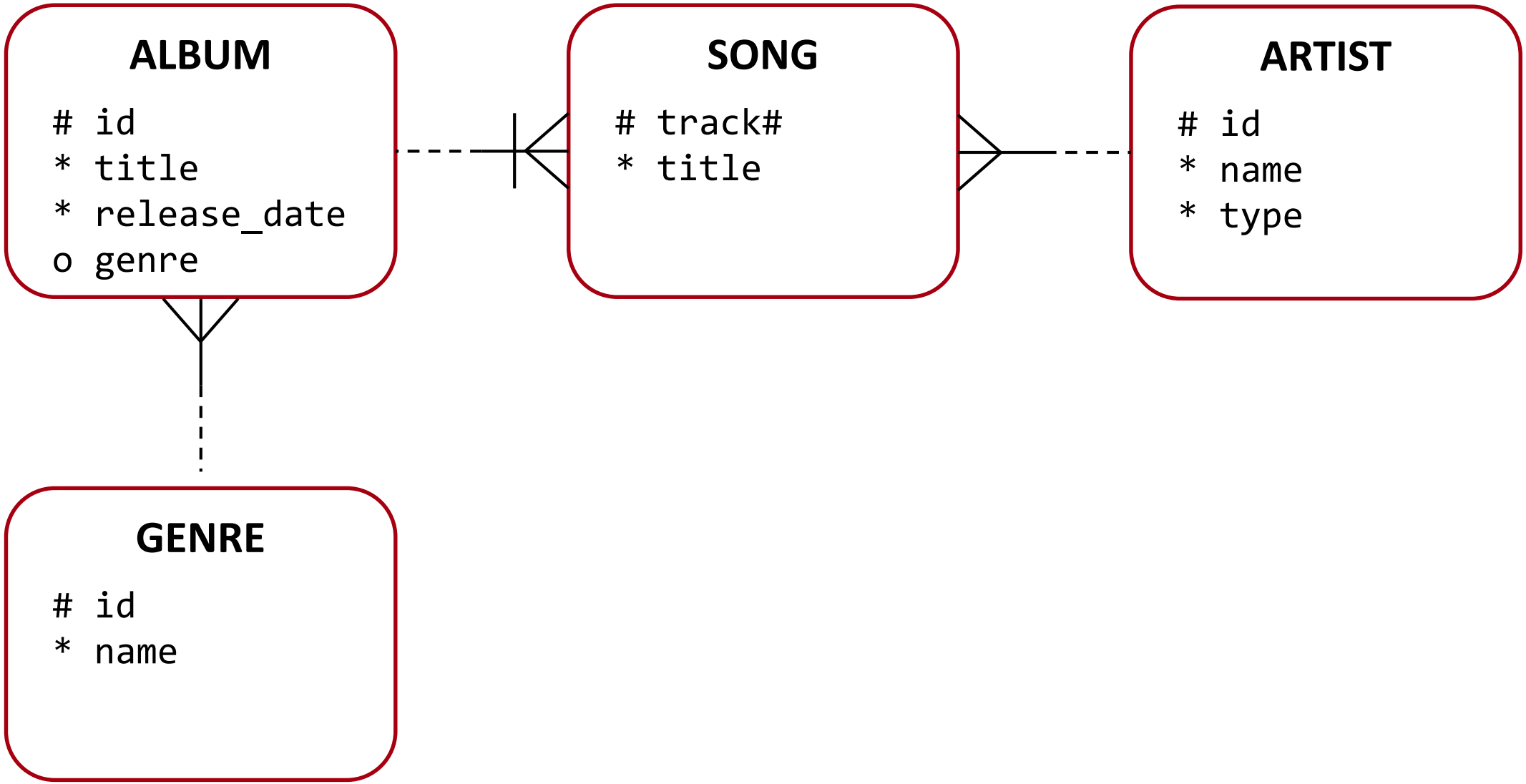

```
exec dbms_utility.compile_schema(user,compile_all=>false)
```

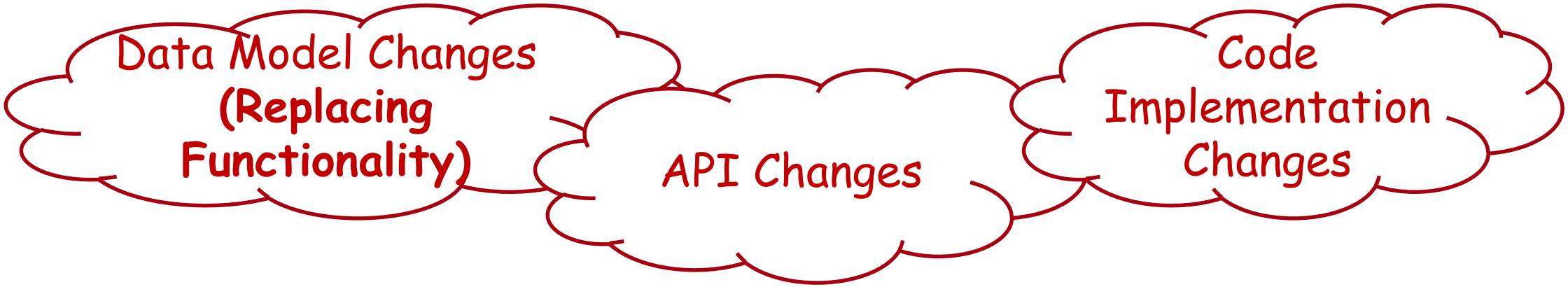

```
grant use on edition v3 to music_user;
```


```
begin

  dbms_service.modify_service(
    service_name => 'MUSIC_SERVICE_A',
    edition => 'V3',
    modify_edition => TRUE)
  );

end;
```



Entity Relationship Diagram





Version #4

Replace the free text **genre** column with a foreign key to a new table

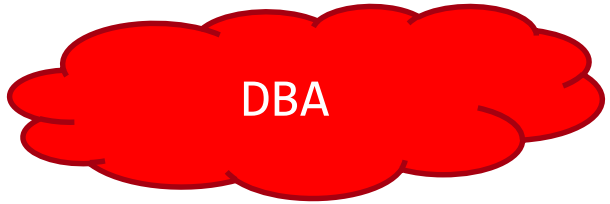
The Upgrade

Create a New Edition

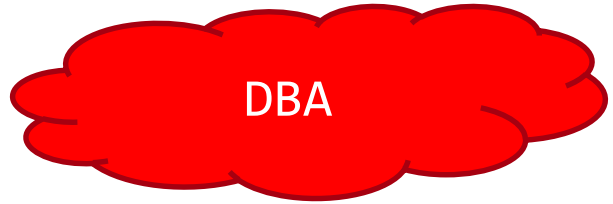
Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition




```
create edition v4;
```




```
create edition v4;
```

```
grant use on edition v4 to music_owner;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition


```
alter session set edition=v4;
```

MUSIC_OWNER

```
create table genres$0 (
  id
    number(3)
    not null
    constraint genres$0_pk primary key,
  name
    varchar2(20)
    not null
);
```

```
create editioning view genres as
  select
    id,
    name
  from
    genres$0;
```

MUSIC_OWNER

```
insert into genres (id,name) values (1,'Classical');
insert into genres (id,name) values (2,'Country');
insert into genres (id,name) values (3,'Electronic');
insert into genres (id,name) values (4,'Folk');
insert into genres (id,name) values (5,'Hip-Hop');
insert into genres (id,name) values (6,'Jazz');
insert into genres (id,name) values (7,'Latin');
insert into genres (id,name) values (8,'Pop');
insert into genres (id,name) values (9,'Rock');
insert into genres (id,name) values (10,'R&B');
insert into genres (id,name) values (11,'Soul');
```

MUSIC_OWNER

```
alter table albums$0 add (
  genre_id
    constraint albums$0_fk_genre_id references genres$0 (id)
);
```

```
create or replace editioning view albums as
  select
    id,
    title,
    release_date,
    genre_id
  from
    albums$0;
```

MUSIC_OWNER

```
create or replace package albums_dl as
    procedure add
    (
        i_title        in albums.title%type,
        i_release_date in albums.release_date%type,
        o_id           out albums.id%type
    );

    procedure set_genre_id
    (
        i_id       in albums.id%type,
        i_genre_id in albums.genre_id%type
    );

end albums_dl;
/
```

```
create or replace package body albums_dl as
.
.
    procedure set_genre_id
    (
        i_id       in albums.id%type,
        i_genre_id in albums.genre_id%type
    ) is
    begin
        update albums a
        set    a.genre_id = i_genre_id
        where  a.id = i_id;
    end set_genre_id;

end albums_dl;
/
```

MUSIC_OWNER

```
create or replace package body music_mgr as
.
.
    procedure set_album_genre_id
    (
        i_album_id in albums.id%type,
        i_genre_id in albums.genre_id%type
    ) is
    begin
        albums_dl.set_genre_id(i_id => i_album_id,
                               i_genre_id => i_genre_id);
        commit;
    end set_album_genre_id;
.
.
```

```
create or replace package body music_mgr as
.
    procedure get_albums
    (
        o_albums out sys_refcursor
    ) is
    begin
        open o_albums for
            select a.id,
                   a.title,
                   a.release_date,
                   g.name genre
            from   albums a,
                   genres g
            where  g.id(+) = a.genre_id
            order  by a.title;
    end get_albums;
```

MUSIC_OWNER

```
create or replace trigger albums_fce_trig
    before insert or update of genre on albums$0
    for each row
    forward crossedition
    disable
begin
    if :new.genre is null then
        :new.genre_id := null;
    else
        select g.id
        into   :new.genre_id
        from   genres g
        where  upper(g.name) = upper(:new.genre);
    end if;
end albums_fce_trig;
/
```

```
alter trigger albums_fce_trig enable;
```


```
merge into albums$0 a
using genres g
on (upper(a.genre) = upper(g.name))
when matched then
update set a.genre_id = g.id;
```


```
create or replace trigger albums_rce_trig
    before insert or update of genre_id on albums$0
    for each row
    reverse crossedition
    disable
begin
    if :new.genre_id is null then
        :new.genre := null;
    else
        select g.name
        into   :new.genre
        from   genres g
        where  g.id = :new.genre_id;
    end if;
end albums_rce_trig;
/
```

```
alter trigger albums_rce_trig enable;
```

The Upgrade

Create a New Edition

Create/Alter Non-Editioned Objects

Create/Replace Editioned Objects in the New Edition

Expose the New Edition

Exposing the Post-Upgrade Version

Validate (and actualize) necessary objects

Grant permissions on new objects to the application user

Create necessary synonyms

Grant use on the new edition to the application user

Associate the new edition with a service


```
exec dbms_utility.compile_schema(user,compile_all=>false)
```


```
grant use on edition v4 to music_user;
```

DBA

```
begin

  dbms_service.create_service(
    service_name => 'MUSIC_SERVICE_B',
    network_name => 'MUSIC_SERVICE_B',
    edition => 'V4'
  );

  dbms_service.start_service(
    service_name => 'MUSIC_SERVICE_B'
  );

end;
```

@demo04

Beyond Online Upgrades

- Upgrades can be done at any time
- Upgrades can take as long as needed
- DB-side upgrade can be done independently of App Server upgrade readiness
- Flexible exposure of new versions
  - Testing of the new version before it is exposed to the end users
  - Different types of App Servers may use different editions

Thank You☺

Oren Nakdimon

www.db-oriented.com

✉ oren@db-oriented.com

☎ +972-54-4393763

@DBoriented

dbORIENTED