Design Patterns in PL/SQL and SQL

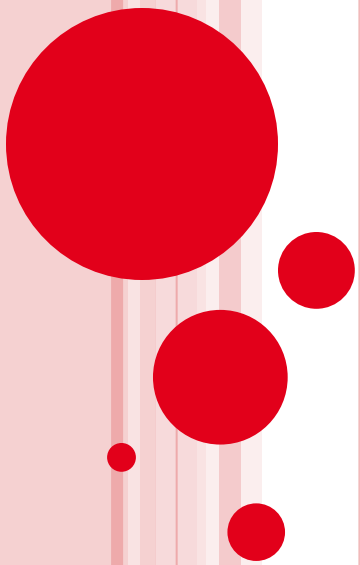
Oren Nakdimon

https://linktr.ee/oren

oren@db-oriented.com

https://db-oriented.com

dbORIENTED

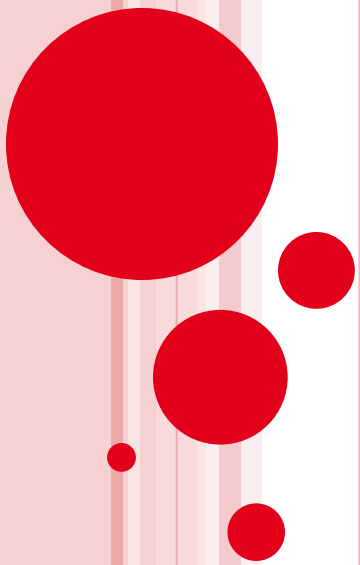


Design Patterns

General Definition

In software engineering, a software **design pattern** is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

https://en.wikipedia.org/wiki/Software_design_pattern



Design Patterns

For Oracle Developers

dbORIENTED

Data Model

SQL

PL/SQL

Concrete Task

General Task

Suggested Solution

Challenges

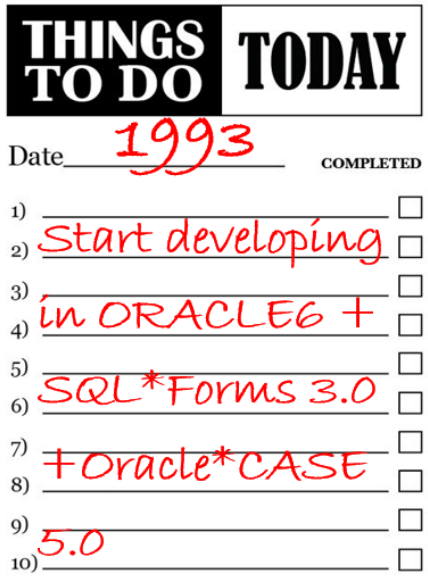
Variations

"Mini-Patterns"

https://db-oriented.com

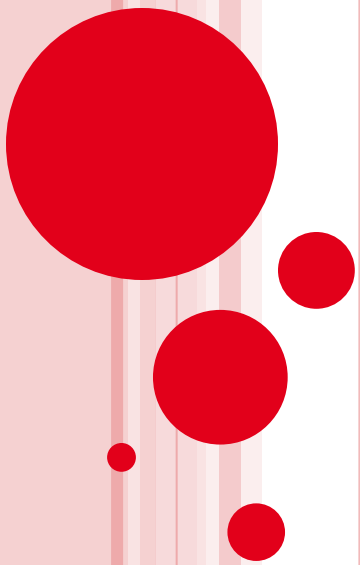
Israel Oracle User Group
ארגון משתמשי אורקל בישראל

https://linktr.ee/oren

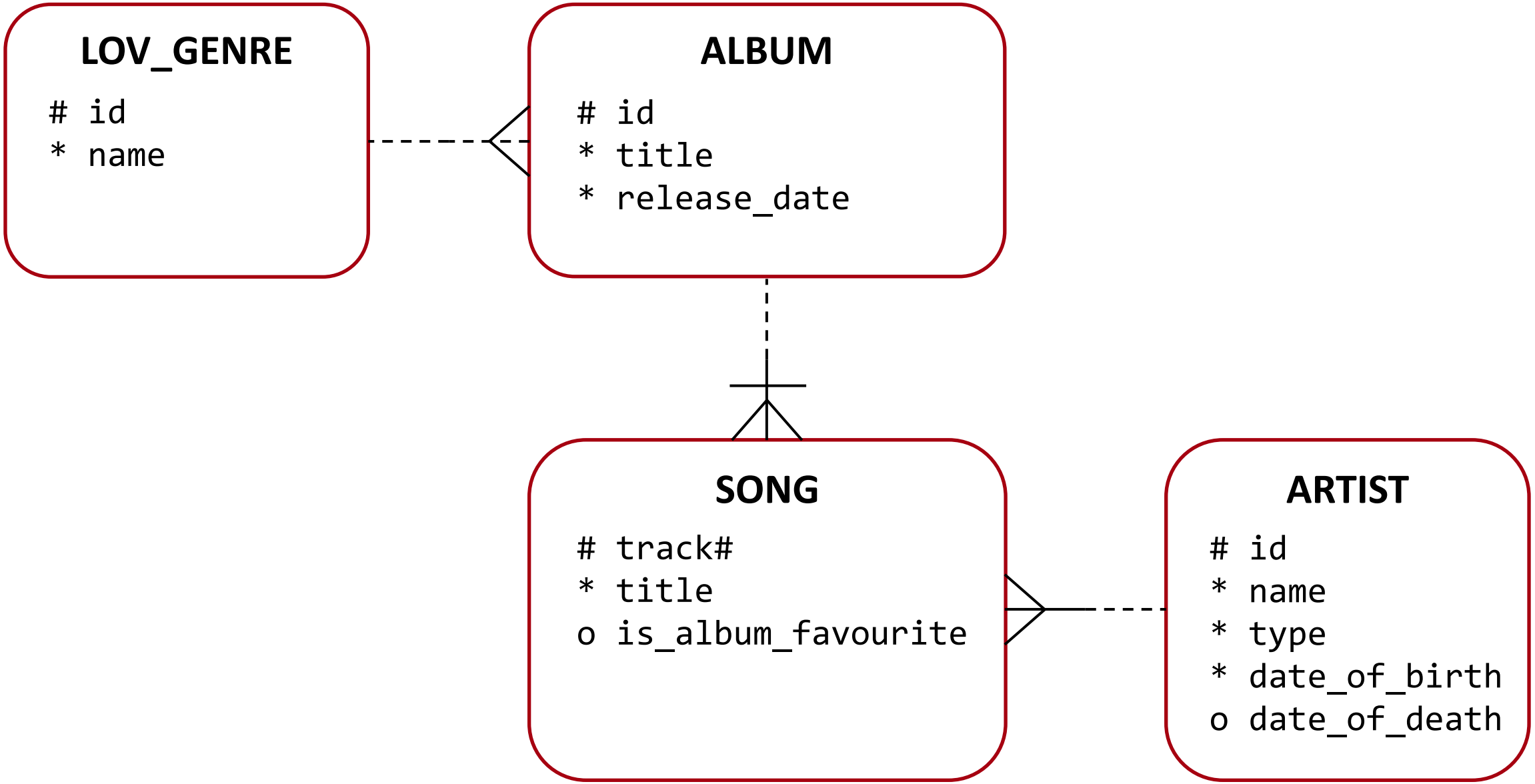


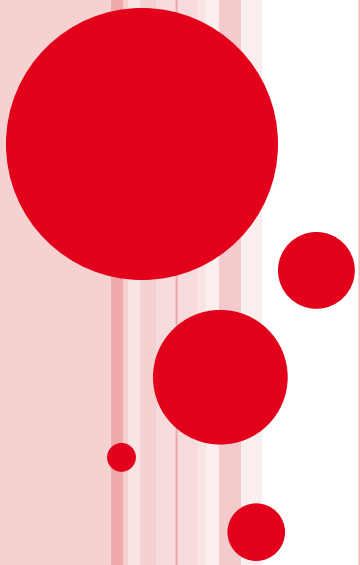
dbORIENTED

moovit



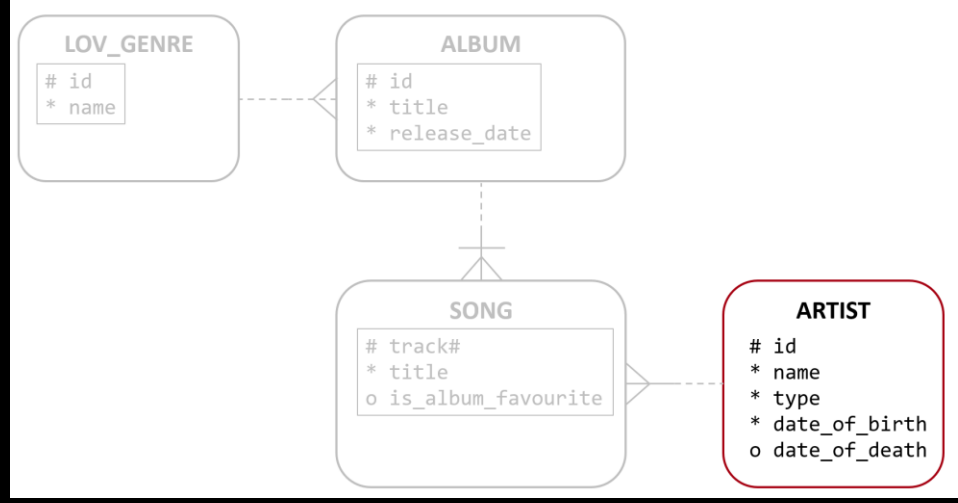
The Demo Model





Overlap Check

```
create table artists (
  id
    integer
    generated as identity
    not null
    constraint artists_pk primary key,
  name
    varchar2(100)
    not null,
  type
    varchar2(6)
    not null
    constraint artists_type_chk check (type in ('Person','Group')),
  date_of_birth
    date
    not null
    constraint artists_dob_chk check (date_of_birth = trunc(date_of_birth)),
  date_of_death
    date
    constraint artists_dod_chk check (date_of_death = trunc(date_of_death))
);
```



```
create table artists (
  id
    integer
    generated as identity
    not null
    constraint artists_pk primary key,
  name
    varchar2(100)
    not null,
  type
    varchar2(6)
    not null
    constraint artists_type_chk check (type in ('Person','Group')),
  date_of_birth
    date
    not null
    constraint artists_dob_chk check (date_of_birth = trunc(date_of_birth)),
  date_of_death
    date
    constraint artists_dod_chk check (date_of_death = trunc(date_of_death))
);
```

Mini-Pattern

Implementing a "pure" date attribute (with no time part)

```
create table artists (
  id
    integer
    generated as identity
    not null
    constraint artists_pk primary key
  name
    var
    not
  type
    var
    not
    constraint artists_type_chk check (type in ('Person','Group')),
  date_of_birth
    date
    not null
    constraint artists_dob_chk check (date_of_birth = trunc(date_of_birth)),
  date_of_death
    date
    constraint artists_dod_chk check (date_of_death = trunc(date_of_death))
);
```

```
create domain date_with_no_time as
  date
  constraint date_with_no_time_chk
    check (date_with_no_time = trunc(date_with_no_time));
```

23ai

Mini-Pattern

Implementing a “pure” date attribute (with no time part)

```
create table artists (
  id
    integer
    generated as identity
    not null
    constraint artists_pk primary key
  name
    var
    not
  type
    var
    not
    constraint artists_type_chk check (type in ('Person','Group')),
  date_of_birth
    date_with_no_time
    not null,

  date_of_death
    date_with_no_time

);
```

```
create domain date_with_no_time as
  date
  constraint date_with_no_time_chk
    check (date_with_no_time = trunc(date_with_no_time));
```

23ai

Mini-Pattern

Implementing a "pure" date attribute (with no time part)

```
create or replace package music is
  ...
  procedure get_people
  (
    i_from_date  in date,
    i_to_date    in date,
    o_artists_rc out sys_refcursor
  );
 ...
end music;
```

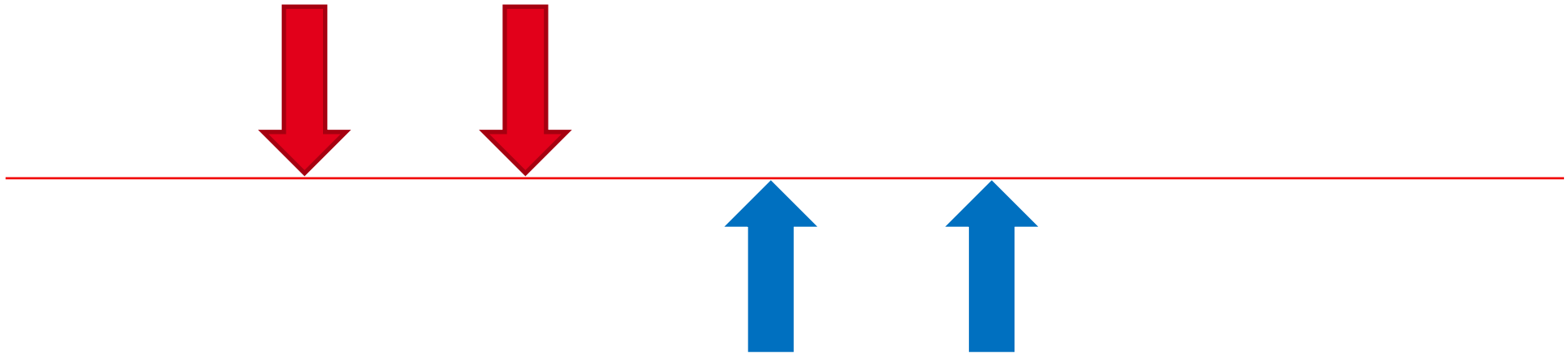
Find all the artists that were alive between 2 dates

Find rows that (partially or fully) overlap a given range

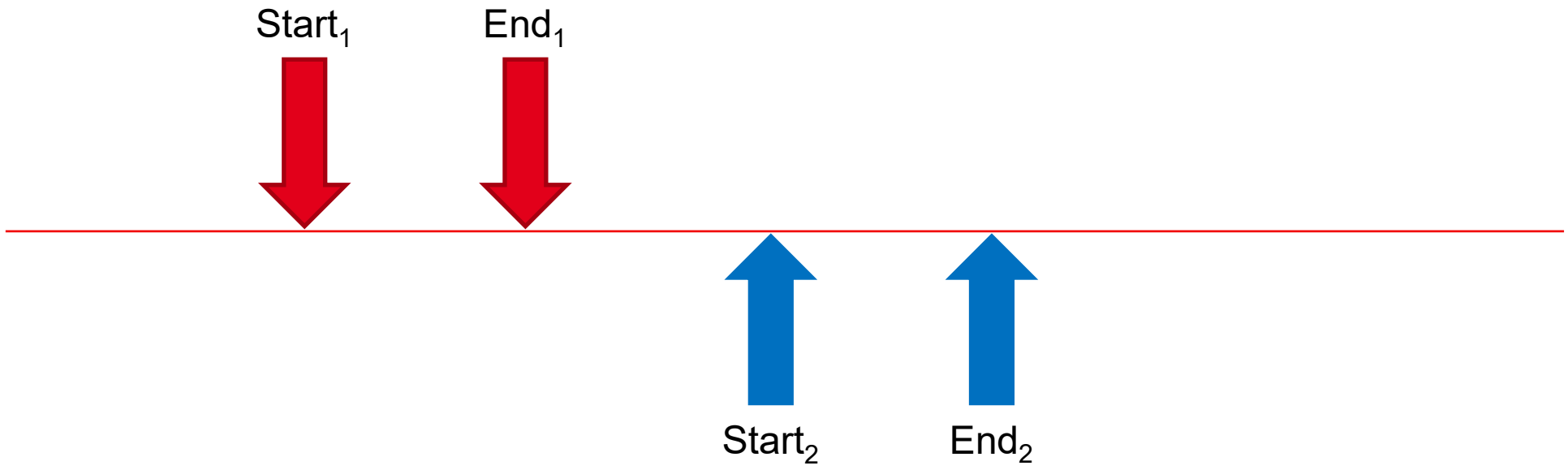
Get all the projects that were active in the last 7 days

Get the employees that worked in the company during 2019

Disjoint Ranges

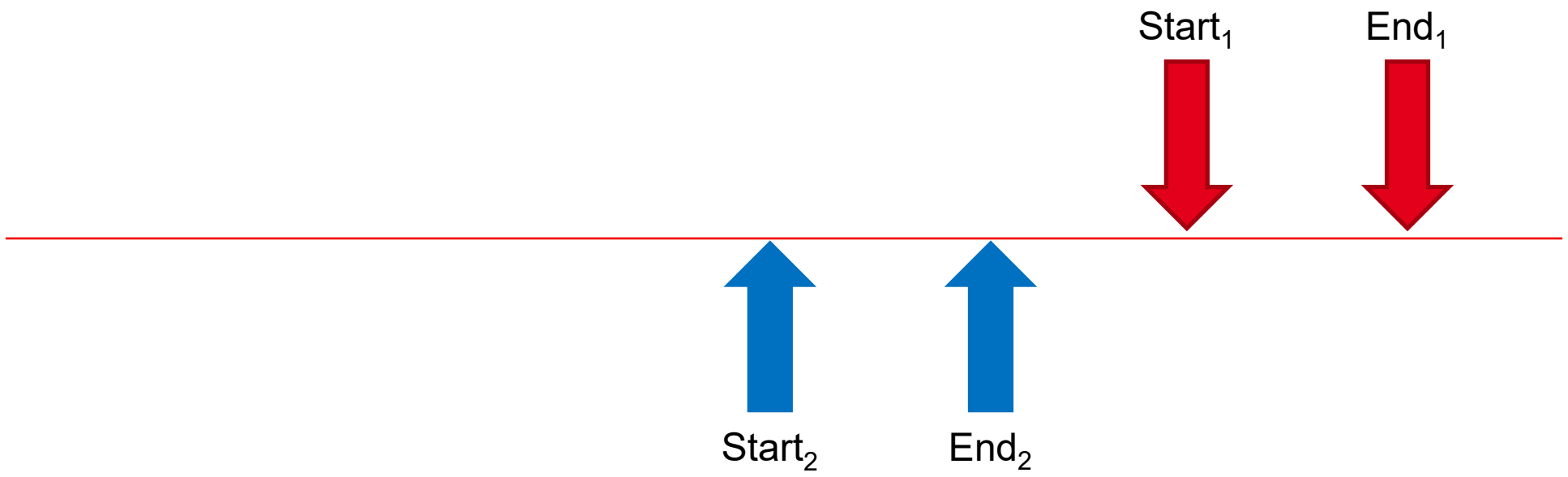


Disjoint Ranges



$Start_2 > End_1$

Disjoint Ranges

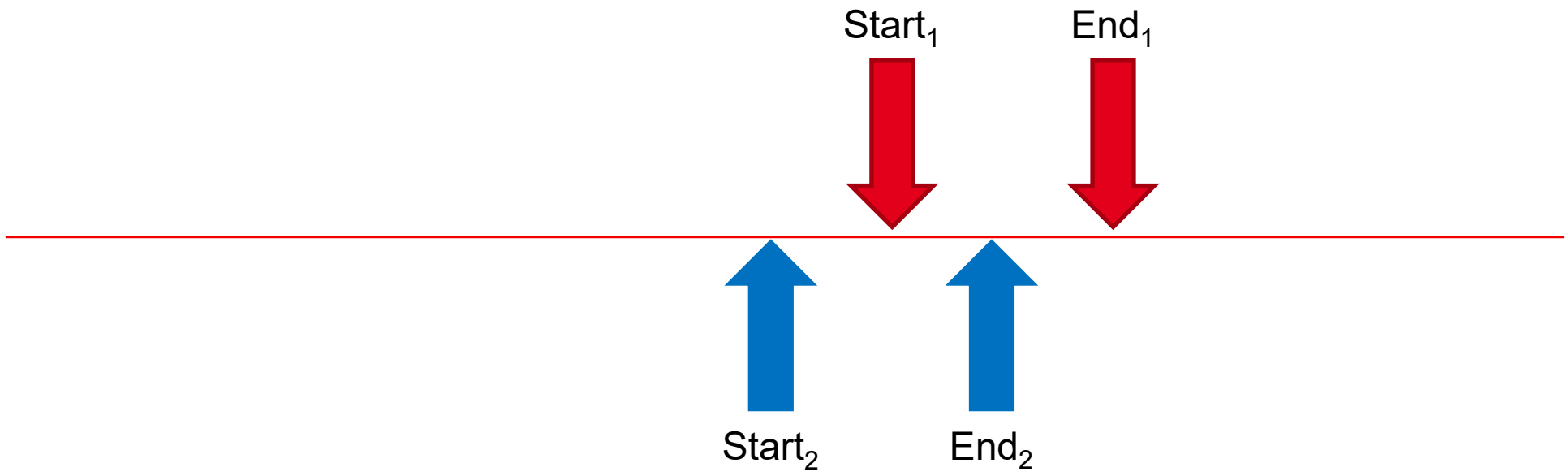


$Start_2 > End_1$

or

$Start_1 > End_2$

Overlapping Ranges

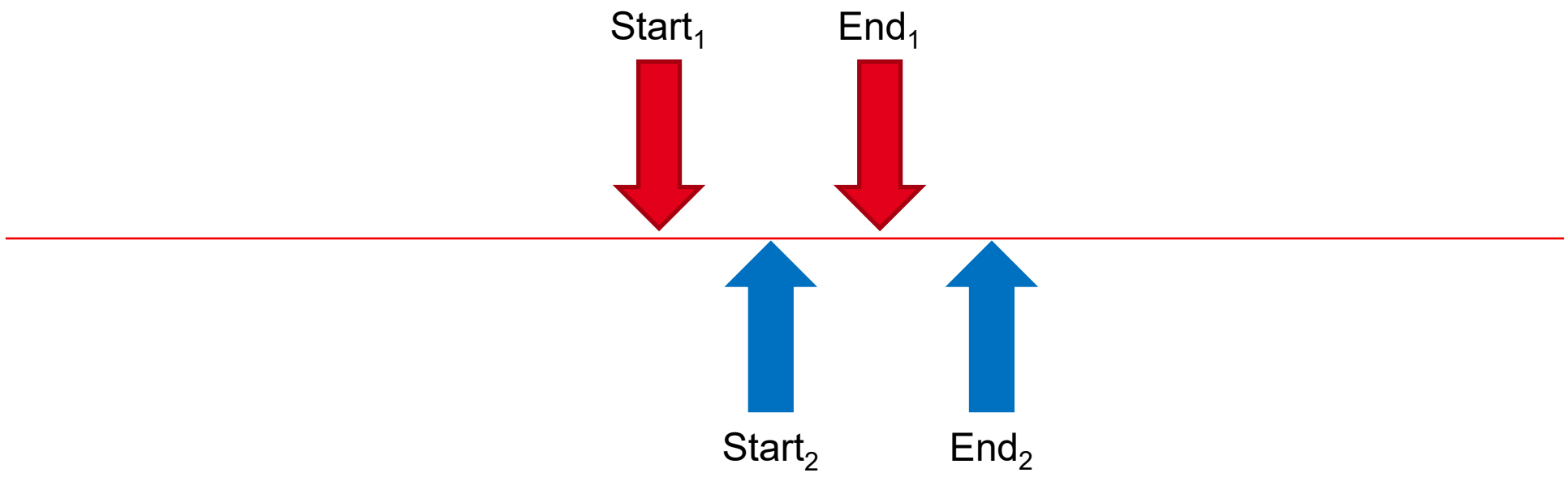


$Start_2 < End_1$

and

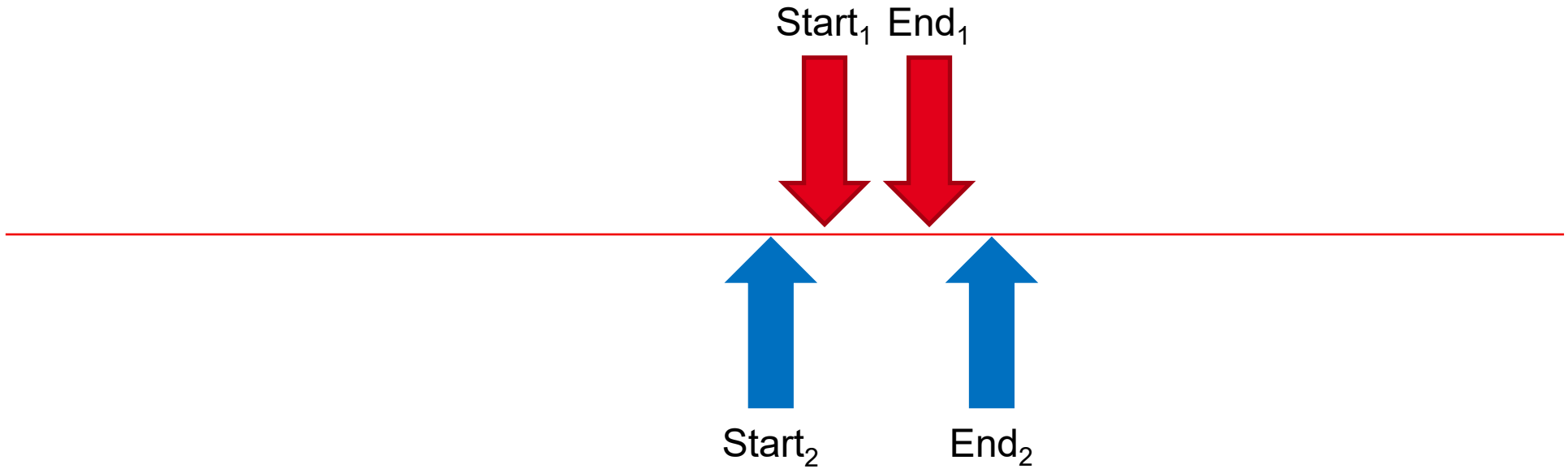
$Start_1 < End_2$

Overlapping Ranges



$$Start_2 < End_1$$
and
$$Start_1 < End_2$$

Overlapping Ranges



$Start_2 < End_1$

and

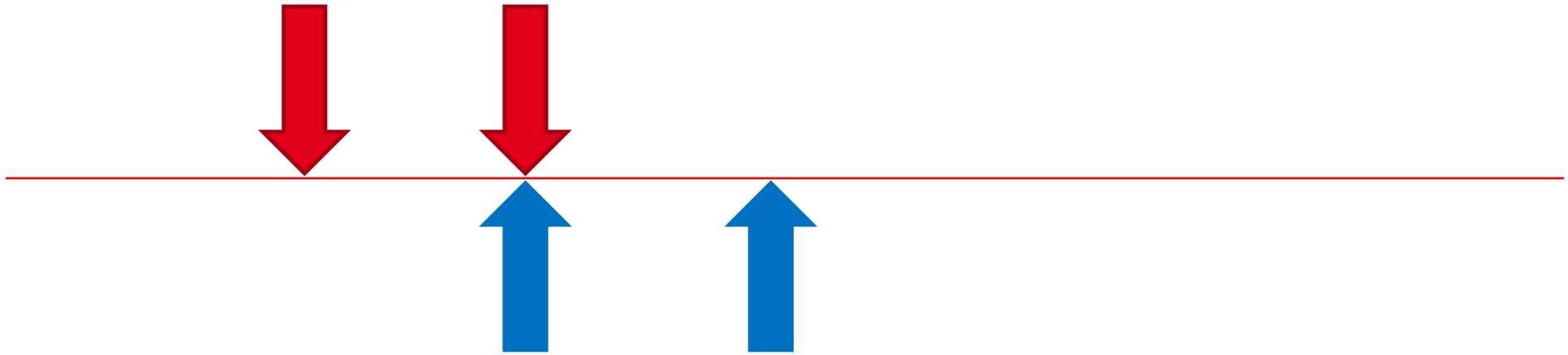
$Start_1 < End_2$

The Suggested Solution

```
StartColumn < EndParameter
        and
StartParameter < EndColumn
```

```
procedure get_people
 (
   i_from_date  in date,
   i_to_date    in date,
   o_artists_rc out sys_refcursor
 ) is
 begin
   open o_artists_rc for
     select ar.name,
            ar.date_of_birth,
            ar.date_of_death
     from   artists ar
     where  ar.type = 'Person'
     and    ar.date_of_birth < i_to_date
     and    i_from_date < ar.date_of_death
     order  by ar.date_of_birth,
               ar.name;
 end get_people;
```

Variation: Inclusive vs. Exclusive Ranges



The Suggested Solution

Exclusive Ranges

```
StartColumn < EndParameter
and
StartParameter < EndColumn
```

Inclusive Ranges

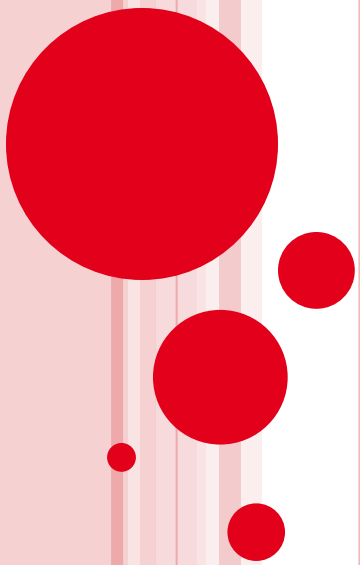
```
StartColumn <= EndParameter
and
StartParameter <= EndColumn
```

```
procedure get_people
 (
   i_from_date  in date,
   i_to_date    in date,
   o_artists_rc out sys_refcursor
 ) is
 begin
   open o_artists_rc for
     select ar.name,
            ar.date_of_birth,
            ar.date_of_death
     from   artists ar
     where  ar.type = 'Person'
     and    ar.date_of_birth <= i_to_date
     and    i_from_date <= ar.date_of_death
     order  by ar.date_of_birth,
               ar.name;
 end get_people;
```

Variation: Nullable vs. Mandatory Columns

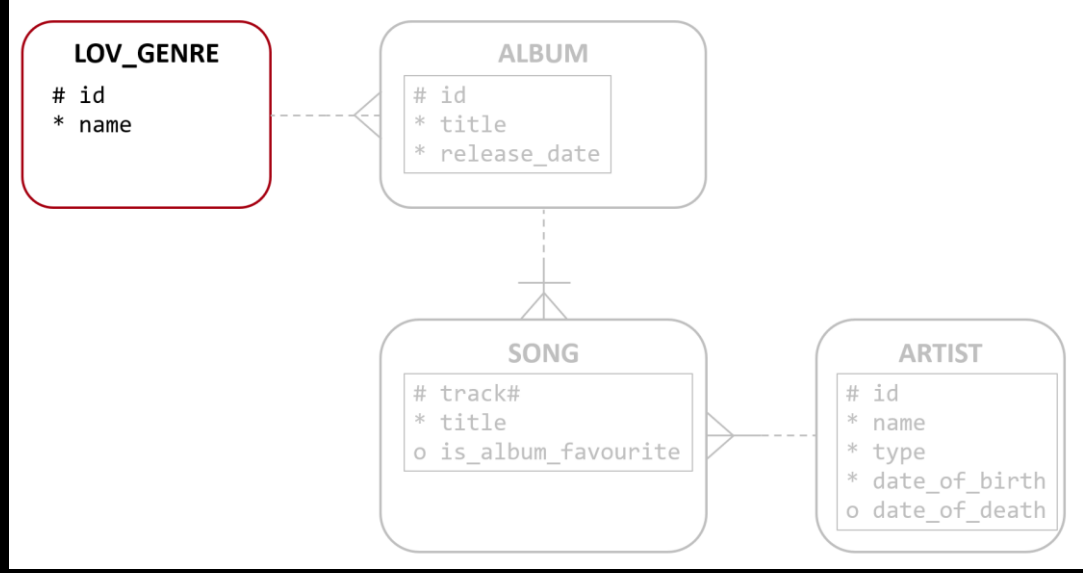
```
create table artists (
  id
    integer
    generated as identity
    not null
    constraint artists_pk primary key,
  name
    varchar2(100)
    not null,
  type
    varchar2(6)
    not null
    constraint artists_type_chk check (type in ('Person','Band')),
  date_of_birth
    date
    not null
    constraint artists_dob_chk check (date_of_birth = trunc(date_of_birth)),
  date_of_death
    date
    constraint artists_dod_chk check (date_of_death = trunc(date_of_death))
);
```

```
procedure get_people
 (
   i_from_date  in date,
   i_to_date    in date,
   o_artists_rc out sys_refcursor
 ) is
 begin
   open o_artists_rc for
     select ar.name,
            ar.date_of_birth,
            ar.date_of_death
     from   artists ar
     where  ar.type = 'Person'
     and    ar.date_of_birth <= i_to_date
     and    i_from_date <= nvl(ar.date_of_death, date '9999-12-31')
     order  by ar.date_of_birth,
               ar.name;
 end get_people;
```

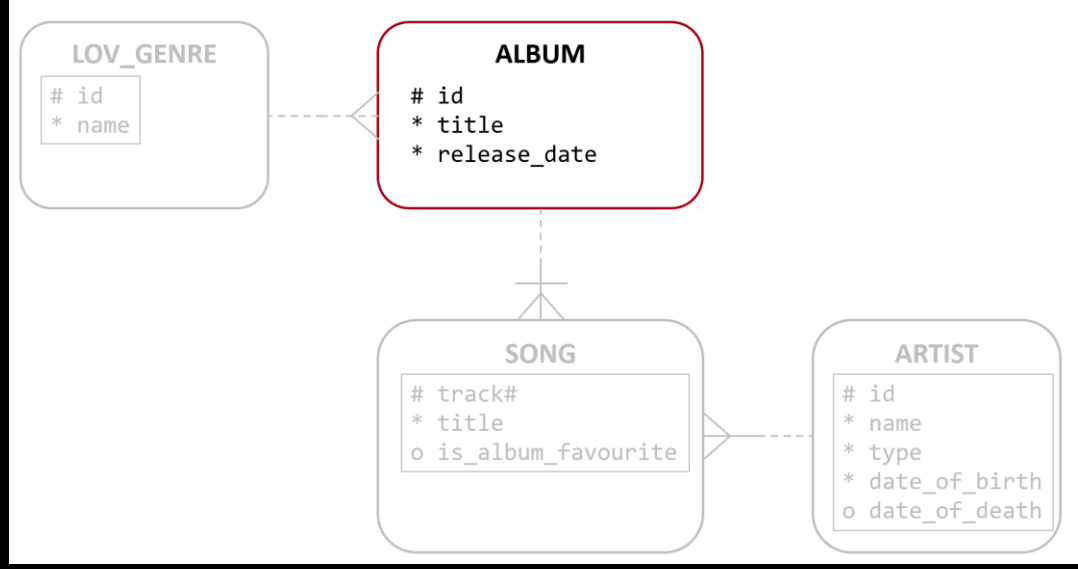


Extendable LOV

```
create table lov_genres (
  id
    integer
    generated as identity
    not null
    constraint lov_genres_pk primary key,
  name
    varchar2(20)
    not null
);
```



```
create table albums (
  id
    integer
    generated as identity
    not null
    constraint albums_pk primary key,
  title
    varchar2(100)
    not null,
  release_date
    date
    not null,
  genre_id
    constraint albums_fk_genre_id references lov_genres (id)
);
```



```
create table albums (
  id
    integer
    generated as identity
    not null
    constraint albums_pk primary key,
  title
    varchar2(100)
    not null,
  release_date
    date
    not null,
  genre_id
    constraint albums_fk_genre_id references lov_genres (id)
);
```

Mini-Pattern

The referencing column implicitly gets the data type of the referenced column

```
create or replace package music is
  ...
  procedure add_album
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    i_genre_name   in lov_genres.name%type default null,
    o_album_id     out albums.id%type
  );  ...
end music;
```

```
create or replace package body music is
  ...
  procedure add_album
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    i_genre_name   in lov_genres.name%type default null,
    o_album_id     out albums.id%type
  ) is
    l_genre_id lov_genres.id%type;
  begin

    l_genre_id := get_genre_id(i_genre_name);

    insert into albums
      (title,
       release_date,
       genre_id)
    values
      (i_title,
       i_release_date,
       l_genre_id)
    returning id into o_album_id;
  end add_album;
```

Get genre id by its name.
If it doesn’t exist, add it.

Look-up a reference table and add new values if not exist

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type

) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;


  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type

) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;


  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type

) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;


  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type

) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;


  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type

) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;
```

```
  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;


  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;
  exception
    when no_data_found then









  end;
  return l_id;
end get_genre_id;
```

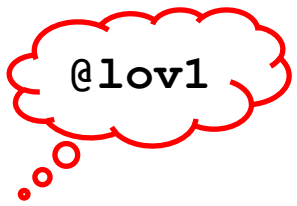
```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;
  exception
    when no_data_found then
      if i_add_if_not_exists then

        insert into lov_genres (name)
        values (i_genre_name)
        returning id into l_id;




      else
        raise_application_error(-20000,
                i_genre_name || ' not found');
      end if;
  end;
  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```

```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;
  exception
    when no_data_found then
      if i_add_if_not_exists then

          insert into lov_genres (name)
          values (i_genre_name)
          returning id into l_id;




      else
        raise_application_error(-20000,
                i_genre_name || ' not found');
      end if;
  end;
  return l_id;
end get_genre_id;
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is


  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```




```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = i_genre_name;
  exception
    when no_data_found then
      if i_add_if_not_exists then

          insert into lov_genres (name)
          values (i_genre_name)
          returning id into l_id;




      else
        raise_application_error(-20000,
                  i_genre_name || ' not found');
      end if;
  end;
  return l_id;
end get_genre_id;
```

Get genre id by its name.
If it doesn't exist, add it.

Look-up a reference table and add new values if not exist

- Name Standardization

```
create table lov_genres (
  id
    integer
    generated as identity
    not null
    constraint lov_genres_pk primary key,
  name
    varchar2(20)
    not null
);

alter table lov_genres add
  constraint lov_genres_name_chk check (name = initcap(trim(name)));
```

```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is
  l_clean_name lov_genres.name%type :=
                  initcap(trim(i_genre_name));
  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```


```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = l_clean_name;
  exception
    when no_data_found then
      if i_add_if_not_exists then

        insert into lov_genres (name)
        values (l_clean_name)
        returning id into l_id;

      else
        raise_application_error(-20000,
                  i_genre_name || ' not found');
      end if;
  end;
  return l_id;
end get_genre_id;
```

Get genre id by its name.
If it doesn't exist, add it.

Look-up a reference table and add new values if not exist

- Name Standardization
- Uniqueness

```
create table lov_genres (
  id
    integer
    generated as identity
    not null
    constraint lov_genres_pk primary key,
  name
    varchar2(20)
    not null
);

alter table lov_genres add
  constraint lov_genres_name_chk check (name = initcap(trim(name)));

alter table lov_genres add
  constraint lov_genres_name_uk unique (name);
```

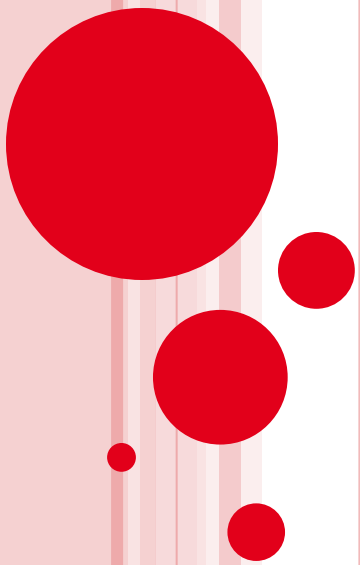
Get genre id by its name.
If it doesn't exist, add it.

Look-up a reference table and add new values if not exist

- Name Standardization
- Uniqueness
- Concurrency Control

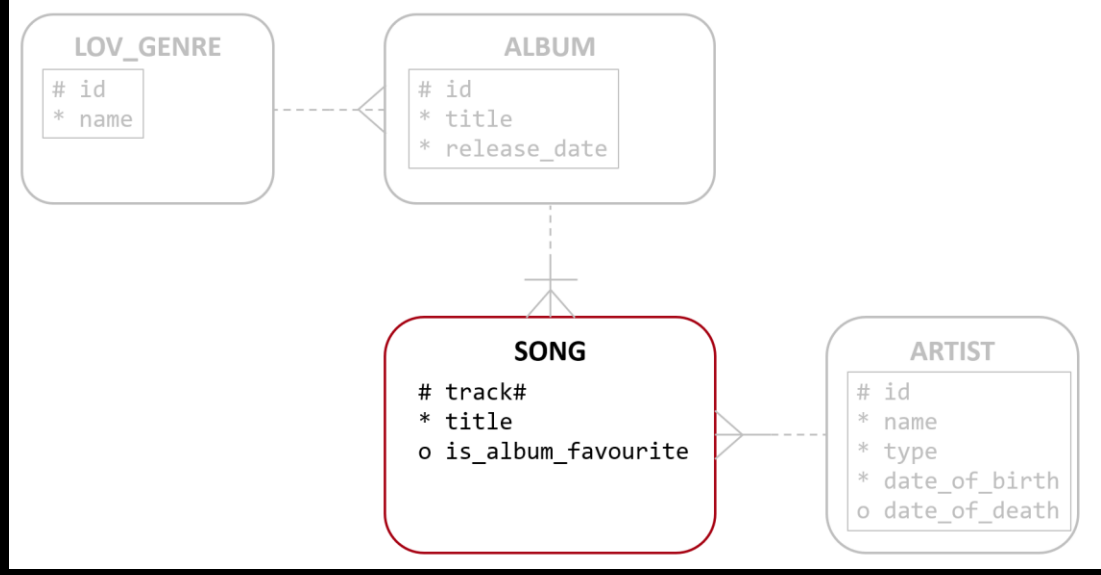
```
function get_genre_id
(
  i_genre_name        in lov_genres.name%type,
  i_add_if_not_exists in boolean default true
) return lov_genres.id%type
is
  l_clean_name lov_genres.name%type :=
                   initcap(trim(i_genre_name));
  l_id lov_genres.id%type;
begin
  if i_genre_name is null then
    return null;
  end if;
```


```
  begin
    select g.id into l_id
    from   lov_genres g
    where  g.name = l_clean_name;
  exception
    when no_data_found then
      if i_add_if_not_exists then
        begin
          insert into lov_genres (name)
          values (l_clean_name)
          returning id into l_id;
        exception
          when dup_val_on_index then
            l_id := get_genre_id(
                i_genre_name => l_clean_name,
                i_add_if_not_exists => false);
        end;
      else
        raise_application_error(-20000,
                i_genre_name || ' not found');
      end if;
  end;
  return l_id;
end get_genre_id;
```



Multi-Value Parameters

```
create table songs (
  album_id
    not null
    constraint songs_fk_album_id references albums
      on delete cascade,
  track#
    number(2)
    not null,
  constraint songs_pk primary key (album_id,track#),
  title
    varchar2(100)
    not null,
  artist_id
    not null
    constraint songs_fk_atrist_id references artists,
  is_album_favourite
    number(1)
    default 0
    not null
    constraint songs_favourite_boolean_chk check (is_album_favourite in (0,1))
);

create index songs_artist_id_ix on songs (artist_id) compress;
```



```
create table songs (
  album_id
    not null
    constraint songs_fk_album_id references albums
      on delete cascade,
  track#
    number(2)
    not null,
  constraint songs_pk primary key (album_id,track#),
  title
    varchar2(100)
    not null,
  artist_id
    not null
    constraint songs_fk_atrist_id references artists,
  is_album_favourite
    number(1)
    default 0
    not null
    constraint songs_favourite_boolean_chk check (is_album_favourite in (0,1))
);

create index songs_artist_id_ix on songs (artist_id) compress;
```

Mini-Pattern

Implementing a Boolean attribute

```
create table songs (
  album_id
    not null
    constraint songs_fk_album_id references albums
      on delete cascade,
  track#
    number(2)
    not null,
  constraint songs_pk primary key (album_id,track#),
  title
    varchar2(100)
    not null,
  artist_id
    not null
    constraint songs_fk_atrist_id references artists,
  is_album_favourite
    boolean
    default false
    not null

);

create index songs_artist_id_ix on songs (artist_id) compress;
```

23ai

Mini-Pattern

Implementing a Boolean attribute

```
create table songs (
  album_id
    not null
    constraint songs_fk_album_id references albums
      on delete cascade,
  track#
    number(2)
    not null,
  constraint songs_pk primary key (album_id,track#),
  title
    varchar2(100)
    not null,
  artist_id
    not null
    constraint songs_fk_atrist_id references artists,
  is_album_favourite
    number(1)
    default 0
    not null
    constraint songs_favourite_boolean_chk check (is_album_favourite in (0,1))
);

create index songs_artist_id_ix on songs (artist_id) compress;
```

Mini-Pattern

Implementing a weak entity

Add an album with its songs

Pass multiple entities to procedures

- Related Entities

```
create or replace package music is
  ...
  procedure add_album
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    i_genre_name   in lov_genres.name%type default null,
    i_songs        in ???,
    o_album_id     out albums.id%type
  );
 ...
end music;
```

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

```
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 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 music is
  ...
  procedure add_album
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    i_genre_name   in lov_genres.name%type default null,
    i_songs        in song_tt,
    o_album_id     out albums.id%type
  );
 ...
end music;
```

Add an album with its songs

Pass multiple entities to procedures

- Related Entities
- Performance – less roundtrips

```
create or replace package body music is
  ...
 procedure add_album
  (
    i_title        in albums.title%type,
    i_release_date in albums.release_date%type,
    i_genre_name   in lov_genres.name%type default null,
    o_album_id     out albums.id%type
  ) is
    l_genre_id lov_genres.id%type;
  begin

    l_genre_id := get_genre_id(i_genre_name);

    insert into albums
      (title,
       release_date,
       genre_id)
    values
      (i_title,
       i_release_date,
       l_genre_id)
    returning id into o_album_id;
  end add_album;
```

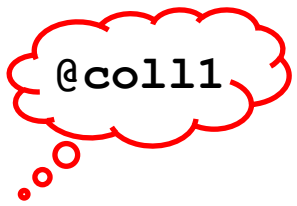
```
procedure add_album
 (
   i_title        in albums.title%type,
   i_release_date in albums.release_date%type,
   i_genre_name   in lov_genres.name%type default null,
   i_songs        in song_tt,
   o_album_id     out albums.id%type
 ) is
   l_genre_id lov_genres.id%type;
 begin
   l_genre_id := get_genre_id(i_genre_name);

   insert into albums
     (title,
      release_date,
      genre_id)
   values
     (i_title,
      i_release_date,
      l_genre_id)
   returning id into o_album_id;
```

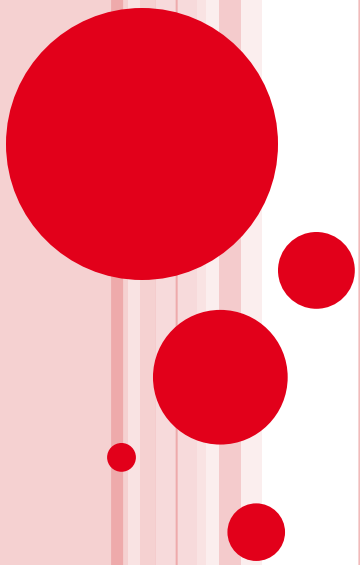
```
    forall i in indices of i_songs
      insert into songs
        (album_id,
         track#,
         title,
         artist_id)
      values
        (o_album_id,
         i_songs(i).track#,
         i_songs(i).title,
         i_songs(i).artist_id);
end add_album;
```

Add an album with its songs

Pass multiple entities to procedures

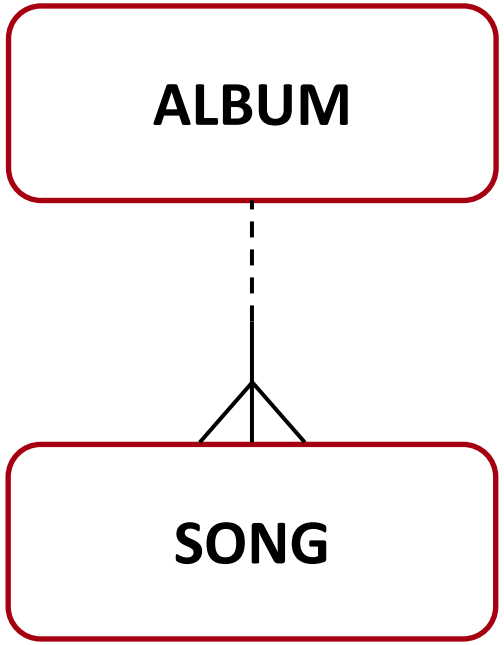
- Related Entities
- Performance – less roundtrips
- Performance – less context switches
- Atomicity



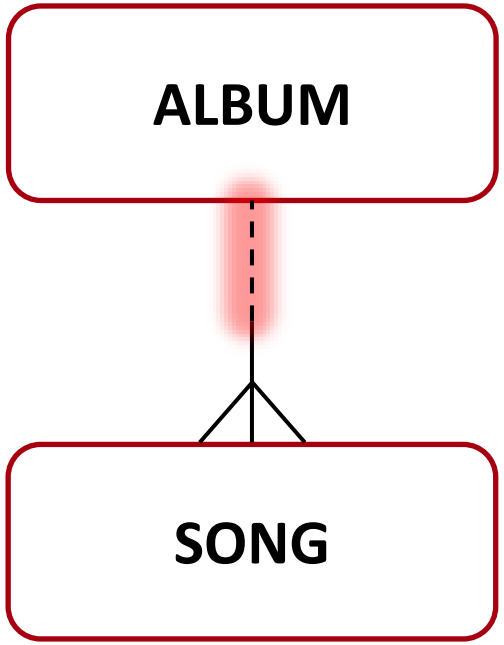


The Guardian Trigger

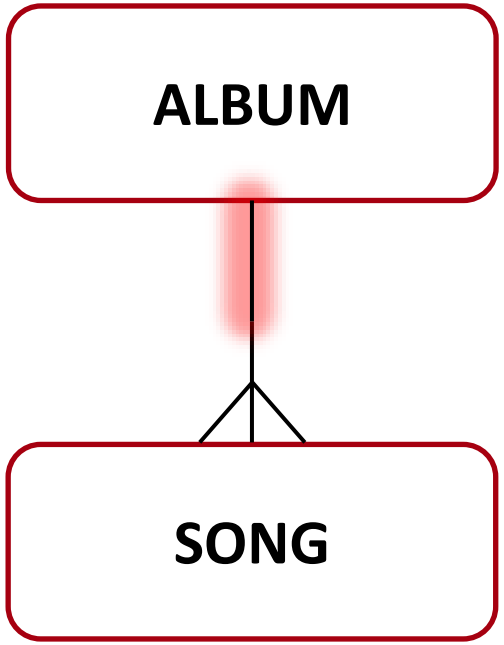
Enforce the rule: an album must have at least one song



Enforce the rule: an album must have at least one song

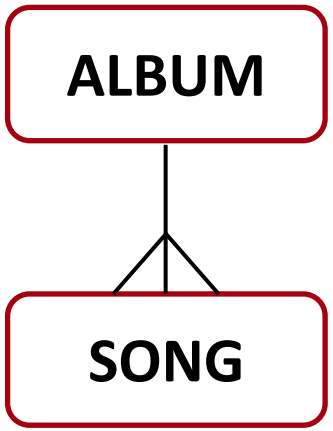


Enforce the rule: an album must have at least one song



Enforce the rule: an album must have at least one song

Whenever some DML is done on some table T,
another piece of code should be executed



Complex Validity Check

Logging

Maintaining Denormalized Entities

```
procedure add_album (
  i_title        in albums.title%type,
  i_release_date in albums.release_date%type,
  i_genre_name   in lov_genres.name%type default null,
  i_songs        in song_tt,
  o_album_id     out albums.id%type)
is
  l_genre_id lov_genres.id%type;
begin




  l_genre_id := get_genre_id(i_genre_name);



  insert into albums (title, release_date, genre_id)
    values (i_title, i_release_date, l_genre_id)
    returning id into o_album_id;
```

```
  forall i in indices of i_songs
    insert into songs
      (album_id,
       track#,
       title,
       artist_id)
    values
      (o_album_id,
       i_songs(i).track#,
       i_songs(i).title,
       i_songs(i).artist_id);









end add_album;
```

```
procedure add_album (
  i_title        in albums.title%type,
  i_release_date in albums.release_date%type,
  i_genre_name   in lov_genres.name%type default null,
  i_songs        in song_tt,
  o_album_id     out albums.id%type)
is
  l_genre_id lov_genres.id%type;
begin
  if i_songs is null or i_songs is empty then
    raise_application_error(
                  -20000, 'an album must have songs');
  end if;

  l_genre_id := get_genre_id(i_genre_name);



  insert into albums (title, release_date, genre_id)
    values (i_title, i_release_date, l_genre_id)
    returning id into o_album_id;
```

```
  forall i in indices of i_songs
    insert into songs
      (album_id,
       track#,
       title,
       artist_id)
    values
      (o_album_id,
       i_songs(i).track#,
       i_songs(i).title,
       i_songs(i).artist_id);









end add_album;
```

```
create or replace package body music is

  g_album_songs_integrity_enforced boolean not null := false;

  function is_album_songs_integrity_enforced return boolean is
  begin
    return g_album_songs_integrity_enforced;
  end is_album_songs_integrity_enforced;

  procedure set_album_songs_integrity_enforced(i_is_enforced in boolean) is
  begin
    g_album_songs_integrity_enforced := i_is_enforced;
  end set_album_songs_integrity_enforced;

  ...
end music;
```

```
create or replace package body music is

  g_album_songs_integrity_enforced boolean not null := false;

  function is_album_songs_integrity_enforced return boolean is
  begin
    return g_album_songs_integrity_enforced;
  end is_album_songs_integrity_enforced;

  procedure set_album_songs_integrity_enforced(i_is_enforced in boolean) is
  begin
    g_album_songs_integrity_enforced := i_is_enforced;
  end set_album_songs_integrity_enforced;

  ...
end music;
```

```
create or replace package body music is

  g_album_songs_integrity_enforced boolean not null := false;

  function is_album_songs_integrity_enforced return boolean is
  begin
    return g_album_songs_integrity_enforced;
  end is_album_songs_integrity_enforced;

  procedure set_album_songs_integrity_enforced(i_is_enforced in boolean) is
  begin
    g_album_songs_integrity_enforced := i_is_enforced;
  end set_album_songs_integrity_enforced;

  ...
end music;
```

```
create or replace package body music is

  g_album_songs_integrity_enforced boolean not null := false;

  function is_album_songs_integrity_enforced return boolean is
  begin
    return g_album_songs_integrity_enforced;
  end is_album_songs_integrity_enforced;

  procedure set_album_songs_integrity_enforced(i_is_enforced in boolean) is
  begin
    g_album_songs_integrity_enforced := i_is_enforced;
  end set_album_songs_integrity_enforced;

  ...
end music;
```

```
create or replace package music is

  function is_album_songs_integrity_enforced return Boolean;
  ...
end music;
```

```
create or replace package body music is

  g_album_songs_integrity_enforced boolean not null := false;

  function is_album_songs_integrity_enforced return boolean is
  begin
    return g_album_songs_integrity_enforced;
  end is_album_songs_integrity_enforced;

  procedure set_album_songs_integrity_enforced(i_is_enforced in boolean) is
  begin
    g_album_songs_integrity_enforced := i_is_enforced;
  end set_album_songs_integrity_enforced;

  ...
end music;
```

```
create or replace trigger albums_guardian_trigger
  before insert on albums
begin
  if not music.is_album_songs_integrity_enforced then
    raise_application_error(
      -20000,
      'to maintain albums and songs, please use the procedures in the music package');
  end if;
end albums_guardian_trigger;
/
```

```
create or replace trigger songs_guardian_trigger
  before delete or update of album_id on songs
begin
  if not music.is_album_songs_integrity_enforced then
    raise_application_error(
      -20000,
      'to maintain albums and songs, please use the procedures in the music package');
  end if;
end songs_guardian_trigger;
/
```

```
procedure add_album (
  i_title        in albums.title%type,
  i_release_date in albums.release_date%type,
  i_genre_name   in lov_genres.name%type default null,
  i_songs        in song_tt,
  o_album_id     out albums.id%type)
is
  l_genre_id lov_genres.id%type;
begin
  if i_songs is null or i_songs is empty then
    raise_application_error(
                  -20000, 'an album must have songs');
  end if;

  l_genre_id := get_genre_id(i_genre_name);



  insert into albums (title, release_date, genre_id)
    values (i_title, i_release_date, l_genre_id)
    returning id into o_album_id;
```

```
  forall i in indices of i_songs
    insert into songs
      (album_id,
       track#,
       title,
       artist_id)
    values
      (o_album_id,
       i_songs(i).track#,
       i_songs(i).title,
       i_songs(i).artist_id);









end add_album;
```

```
procedure add_album (
  i_title        in albums.title%type,
  i_release_date in albums.release_date%type,
  i_genre_name   in lov_genres.name%type default null,
  i_songs        in song_tt,
  o_album_id     out albums.id%type)
is
  l_genre_id lov_genres.id%type;
begin
  if i_songs is null or i_songs is empty then
    raise_application_error(
                  -20000, 'an album must have songs');
  end if;

  l_genre_id := get_genre_id(i_genre_name);

  set_album_songs_integrity_enforced(true);

  insert into albums (title, release_date, genre_id)
    values (i_title, i_release_date, l_genre_id)
    returning id into o_album_id;
```

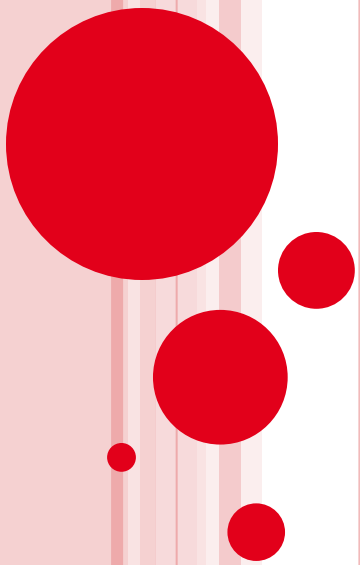
```
  forall i in indices of i_songs
    insert into songs
      (album_id,
       track#,
       title,
       artist_id)
    values
      (o_album_id,
       i_songs(i).track#,
       i_songs(i).title,
       i_songs(i).artist_id);

  set_album_songs_integrity_enforced
                                (false);



end add_album;
```

```
procedure add_album (
  i_title        in albums.title%type,
  i_release_date in albums.release_date%type,
  i_genre_name   in lov_genres.name%type default null,
  i_songs        in song_tt,
  o_album_id     out albums.id%type)
is
  l_genre_id lov_genres.id%type;
begin
  if i_songs is null or i_songs is empty then
    raise_application_error(
                  -20000, 'an album must have songs');
  end if;

  l_genre_id := get_genre_id(i_genre_name);

  set_album_songs_integrity_enforced(true);

  insert into albums (title, release_date, genre_id)
         (i_title, i_release_date, l_genre_id)
         g id into o_album_id;
```


```
  forall i in indices of i_songs
    insert into songs
      (album_id,
       track#,
       title,
       artist_id)
    values
      (o_album_id,
       i_songs(i).track#,
       i_songs(i).title,
       i_songs(i).artist_id);

  set_album_songs_integrity_enforced
                                  (false);
exception
  when others then
   set_album_songs_integrity_enforced
                                  (false);
   raise;
end add_album;
```

The Suggested Solution

- Put the logic in a procedure (or procedures)
- Use a global variable that indicates if the DML is allowed
  - Initialize it to FALSE
  - Set it to TRUE only for the duration of the procedure
- A **statement-level** trigger on the DML verifies the flag is on



Correlated Results

Return albums – and all their songs – based on various inputs

Return albums – and all their songs – based on various inputs

```
procedure get_albums
 (
   i_artist_name     in artists.name%type default null,
   i_genre_id_list   in int_tt default null,
   i_released_after  in albums.release_date%type default null,
   i_released_before in albums.release_date%type default null,
   o_albums_rc       out sys_refcursor,
   o_songs_rc        out sys_refcursor
 );
```

Return albums – and all their songs – based on various inputs

```
procedure get_albums
 (
   i_artist_name     in artists.name%type default null,
   i_genre_id_list   in int_tt default null,
   i_released_after  in albums.release_date%type default null,
   i_released_before in albums.release_date%type default null,
   o_albums_rc       out sys_refcursor,
   o_songs_rc        out sys_refcursor
 );
```

```
create type int_tt as table of integer
```

Return albums – and all their songs – based on various inputs

```
procedure get_albums
 (
   i_artist_name     in artists.name%type default null,
   i_genre_id_list   in int_tt default null,
   i_released_after  in albums.release_date%type default null,
   i_released_before in albums.release_date%type default null,
   o_albums_rc       out sys_refcursor,
   o_songs_rc        out sys_refcursor
 );
```

Return albums – and all their songs – based on various inputs

```
procedure get_albums
 (
   i_artist_name     in artists.name%type default null,
   i_genre_id_list   in int_tt default null,
   i_released_after  in albums.release_date%type default null,
   i_released_before in albums.release_date%type default null,
   o_albums_rc       out sys_refcursor,
   o_songs_rc        out sys_refcursor
 );
```

```
procedure get_albums (
  i_artist_name     in artists.name%type default null,
  i_genre_id_list   in int_tt default null,
  i_released_after  in albums.release_date%type default null,
  i_released_before in albums.release_date%type default null,
  o_albums_rc       out sys_refcursor,
  o_songs_rc        out sys_refcursor)
is
begin

    open o_albums_rc for
      select al.id,
             al.title,
             al.release_date,
             al.genre_id
      from   albums al
      where  ....................
      order  by al.id;
```

```
    open o_songs_rc for
      select s.album_id,
             s.track#,
             s.title,
             s.artist_id,
             s.is_album_favourite
      from   albums al,
             songs s
      where  ....................
      and    s.album_id = al.id
      order  by s.album_id,
                s.track#;
end get_albums;
```

```
procedure get_albums (
  i_artist_name     in artists.name%type default null,
  i_genre_id_list   in int_tt default null,
  i_released_after  in albums.release_date%type default null,
  i_released_before in albums.release_date%type default null,
  o_albums_rc       out sys_refcursor,
  o_songs_rc        out sys_refcursor)
is

  l_album_id_list int_tt;
begin



  select al.id
  bulk collect into l_album_id_list
  from   albums al
  where  (i_artist_name is null or
          exists (select null from songs s, artists ar
                  where  s.album_id = al.id
                  and    instr(ar.name, i_artist_name) > 0
                  and    s.artist_id = ar.id))
  and    (i_genre_id_list is null or al.genre_id member of i_genre_id_list)
  and    al.release_date >= nvl(i_released_after, date '0001-01-01')
  and    al.release_date <= nvl(i_released_before, date '9999-12-31');
```

```
procedure get_albums (
  i_artist_name     in artists.name%type default null,
  i_genre_id_list   in int_tt default null,
  i_released_after  in albums.release_date%type default null,
  i_released_before in albums.release_date%type default null,
  o_albums_rc       out sys_refcursor,
  o_songs_rc        out sys_refcursor)
is

  l_album_id_list int_tt;
begin


  select al.id
  bulk collect into l_album_id_list
  from   albums al
  where  (i_artist_name is null or
          exists (select null from songs
                  where  s.album_id = al
                  and    instr(ar.name, i
                  and    s.artist_id = ar
  and    (i_genre_id_list is null or al.g
  and    al.release_date >= nvl(i_releas
  and    al.release_date <= nvl(i_releas
```

```
  open o_albums_rc for
    select a.id, a.title, a.release_date, a.genre_id
    from   table(l_album_id_list) c,
           albums a
    where  a.id = c.column_value
    order  by a.id;

  open o_songs_rc for
    select s.album_id, s.track#, s.title, s.artist_id,
           s.is_album_favourite
    from   table(l_album_id_list) c,
           songs s
    where  s.album_id = c.column_value
    order  by s.album_id,
              s.track#;

end get_albums;
```

```
procedure get_albums (
  i_artist_name     in artists.name%type default null,
  i_genre_id_list   in int_tt default null,
  i_released_after  in albums.release_date%type default null,
  i_released_before in albums.release_date%type default null,
  o_albums_rc       out sys_refcursor,
  o_songs_rc        out sys_refcursor)
is

  l_album_id_list int_tt;
begin
  set transaction read only;

  select al.id
  bulk collect into l_album_id_list
  from   albums al
  where  (i_artist_name is null or
          exists (select null from songs
                  where  s.album_id = al
                  and    instr(ar.name,
                  and    s.artist_id = a
  and    (i_genre_id_list is null or al.
  and    al.release_date >= nvl(i_releas
  and    al.release_date <= nvl(i_releas
```

```
  open o_albums_rc for
    select a.id, a.title, a.release_date, a.genre_id
    from   table(l_album_id_list) c,
           albums a
    where  a.id = c.column_value
    order  by a.id;

  open o_songs_rc for
    select s.album_id, s.track#, s.title, s.artist_id,
           s.is_album_favourite
    from   table(l_album_id_list) c,
           songs s
    where  s.album_id = c.column_value
    order  by s.album_id,
              s.track#;

  commit;
end get_albums;
```

```
procedure get_albums (
  i_artist_name     in artists.name%type default null,
  i_genre_id_list   in int_tt default null,
  i_released_after  in albums.release_date%type default null,
  i_released_before in albums.release_date%type default null,
  o_albums_rc       out sys_refcursor,
  o_songs_rc        out sys_refcursor)
is
  pragma autonomous_transaction;
  l_album_id_list int_tt;
begin
  set transaction read only;

  select al.id
  bulk collect into l_album_id_list
  from   albums al
  where  (i_artist_name is null or
          exists (select null from songs
                  where  s.album_id = al
                  and    instr(ar.name, i
                  and    s.artist_id = ar
  and    (i_genre_id_list is null or al.g
  and    al.release_date >= nvl(i_releas
  and    al.release_date <= nvl(i_releas
```

```
  open o_albums_rc for
    select a.id, a.title, a.release_date, a.genre_id
    from   table(l_album_id_list) c,
           albums a
    where  a.id = c.column_value
    order  by a.id;

  open o_songs_rc for
    select s.album_id, s.track#, s.title, s.artist_id,
           s.is_album_favourite
    from   table(l_album_id_list) c,
           songs s
    where  s.album_id = c.column_value
    order  by s.album_id,
              s.track#;
  commit;
end get_albums;
```

Thank You☺

Oren Nakdimon

https://linktr.ee/oren

oren@db-oriented.com

https://db-oriented.com

dbORIENTED