CS186 Discussion #6

(ER Diagrams, Advanced SQL)

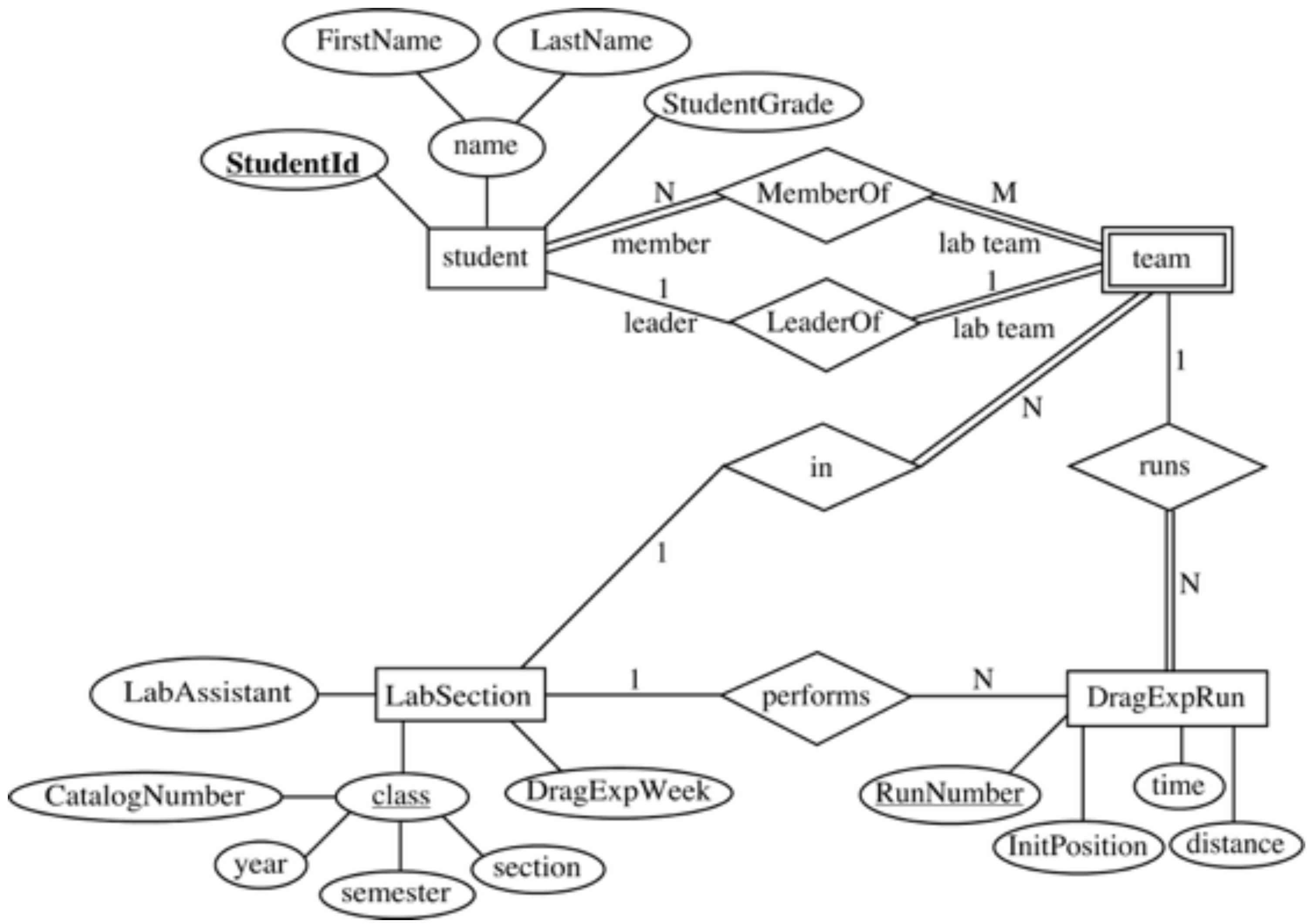
Announcements

- Midterm in class: Thursday 3/5, 2-3:30pm
- Review session: Sunday 3/1, 2-4pm @ 2050 VLSB

ER Diagrams

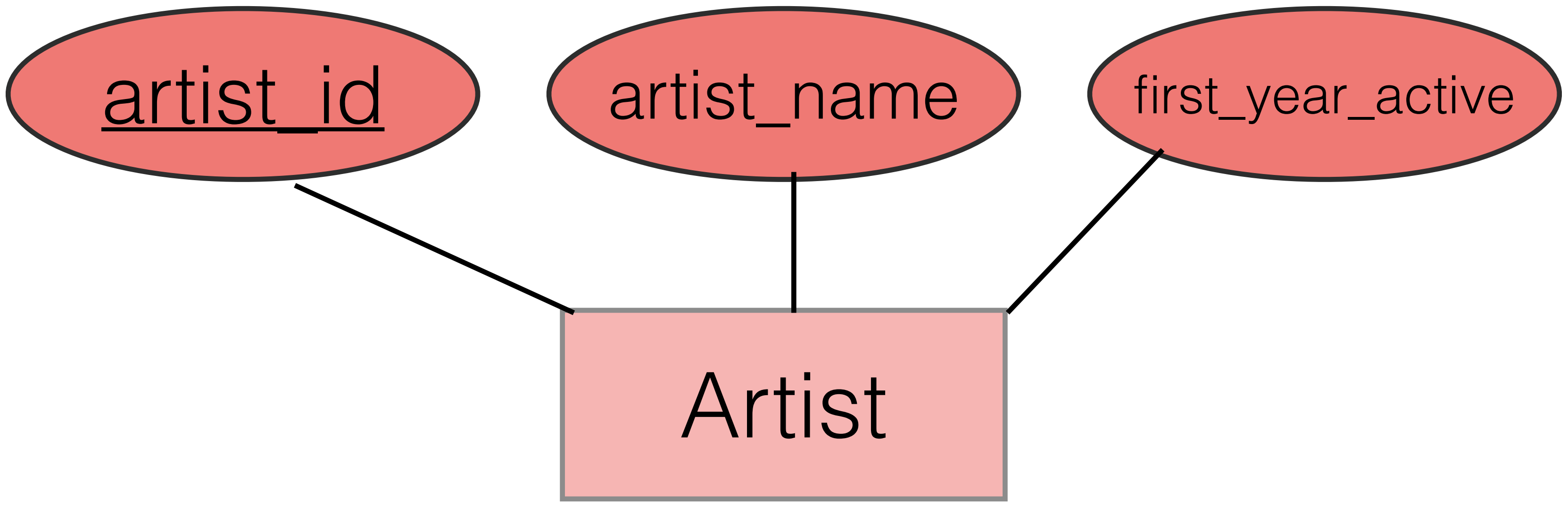
Motivation

- Visualize data schema



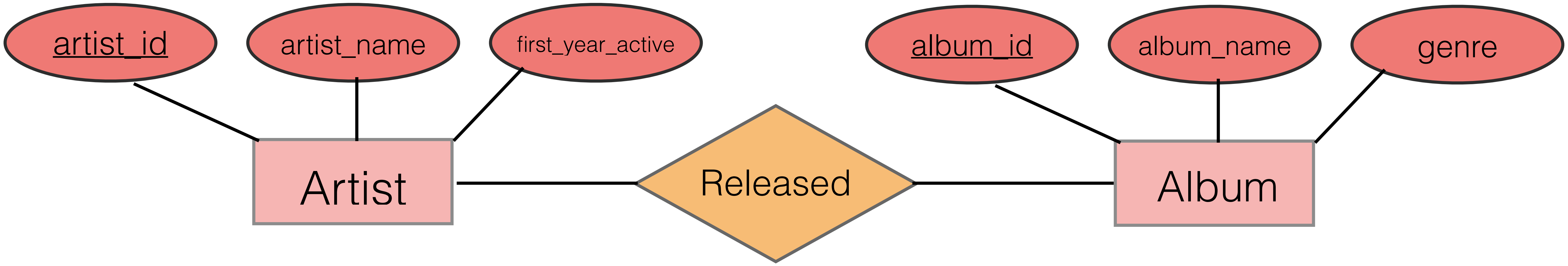
Entities

- Entity: “thing”
- Attribute: Property of the entity
  - Primary key underlined



Relationships

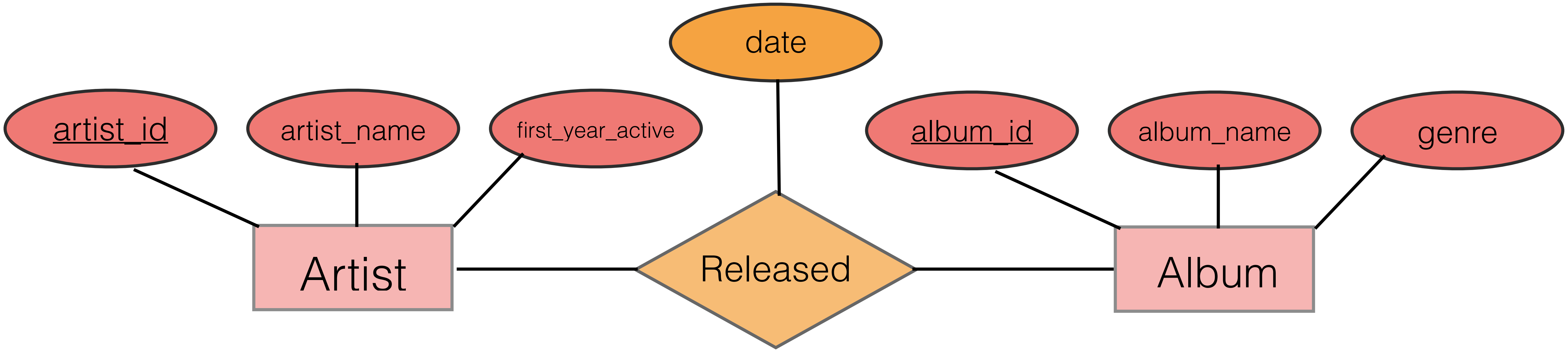
- How two entities interact



Artist 4 released album 2.

Relationships

- How two entities interact
  - Interactions can have attributes



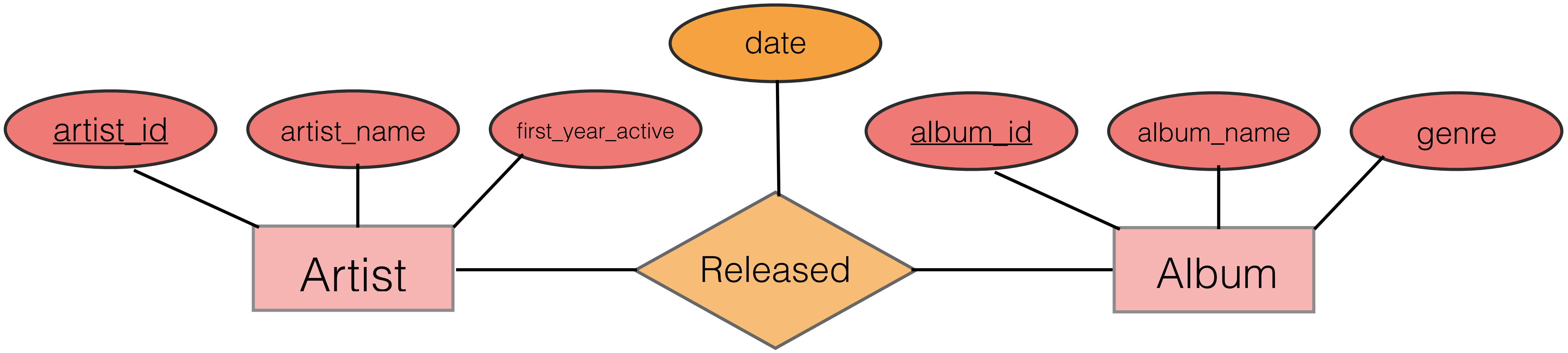
Artist 4 released album 2 on February 27, 2015.

Constraints

- Make relationship lines meaningful
  - Participation constraint (Partial/Total)
    - Total participation: participates at least once
  - Key/Non-key constraint
    - Key: Participates at most once

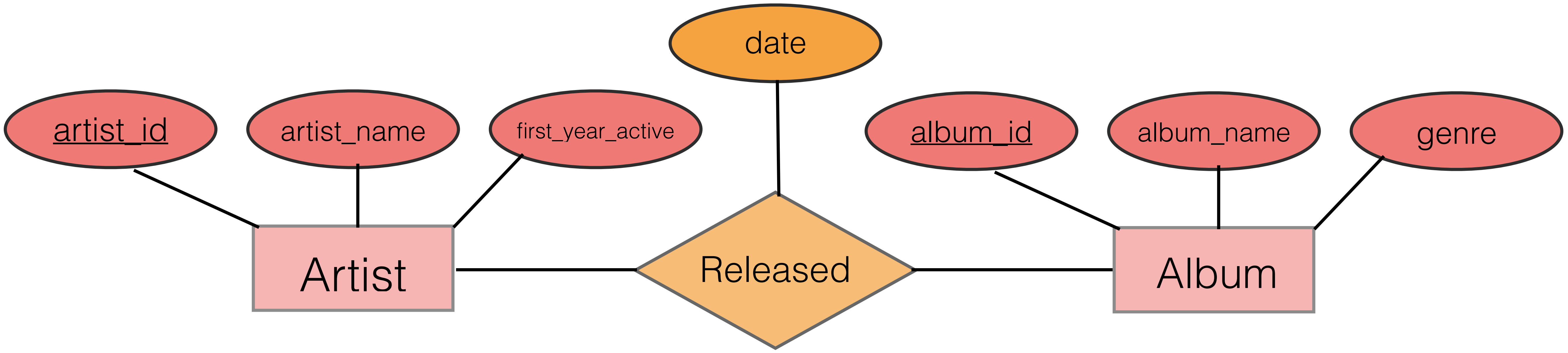
	Partial Participation	Total Participation
Non-Key	0 or More —	1 or More ━
Key	0 or 1 →	Exactly 1 ➡

Constraints



Non-Key constraint with partial participation

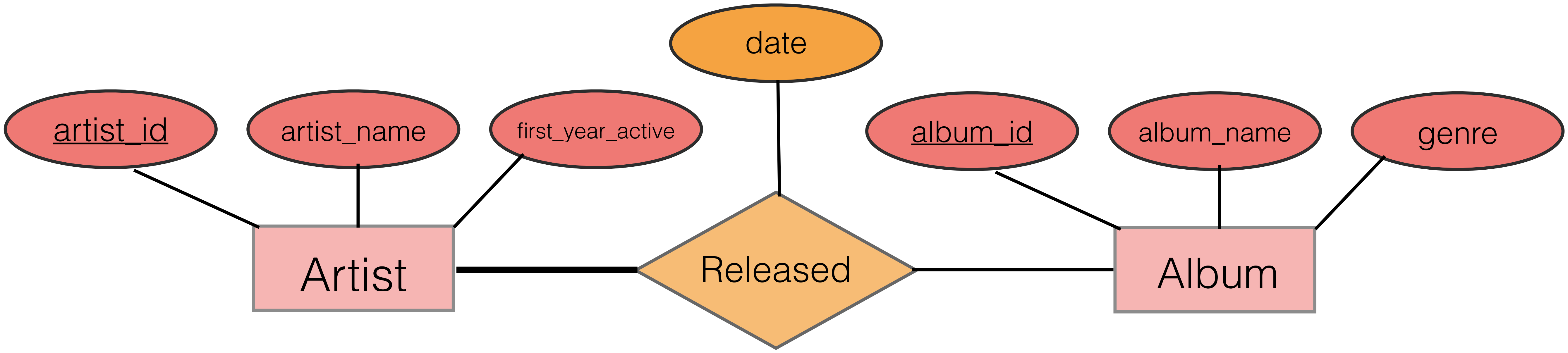
Constraints



Non-Key constraint with partial participation

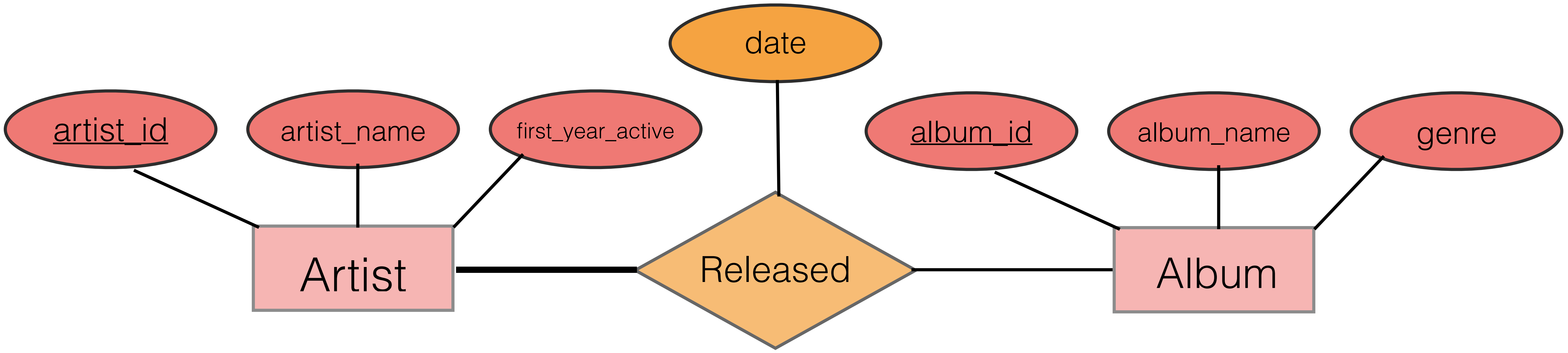
An artist can release an album zero or more times.

Constraints



Non-Key constraint with total participation

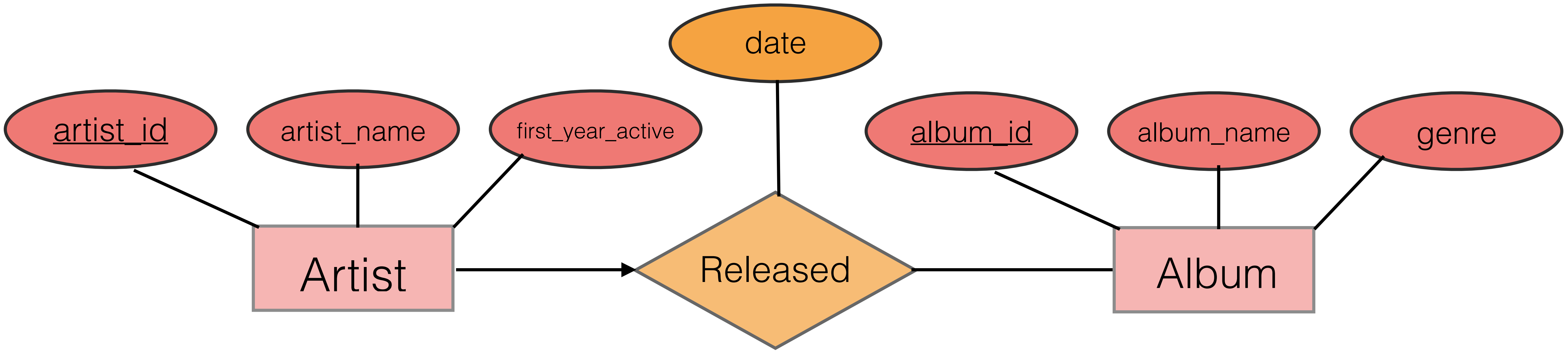
Constraints



Non-Key constraint with total participation

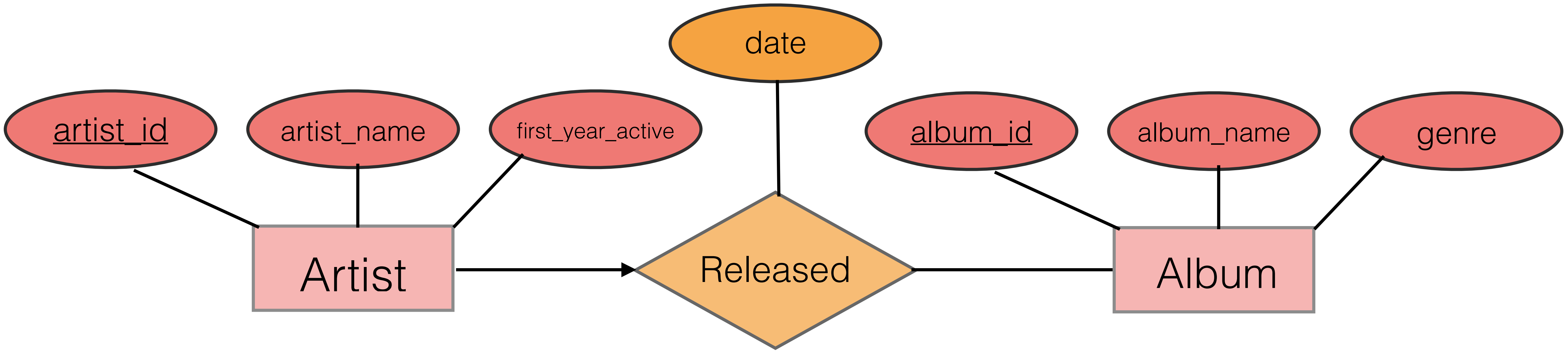
An artist can release an album one or more times.

Constraints



Key constraint with partial participation

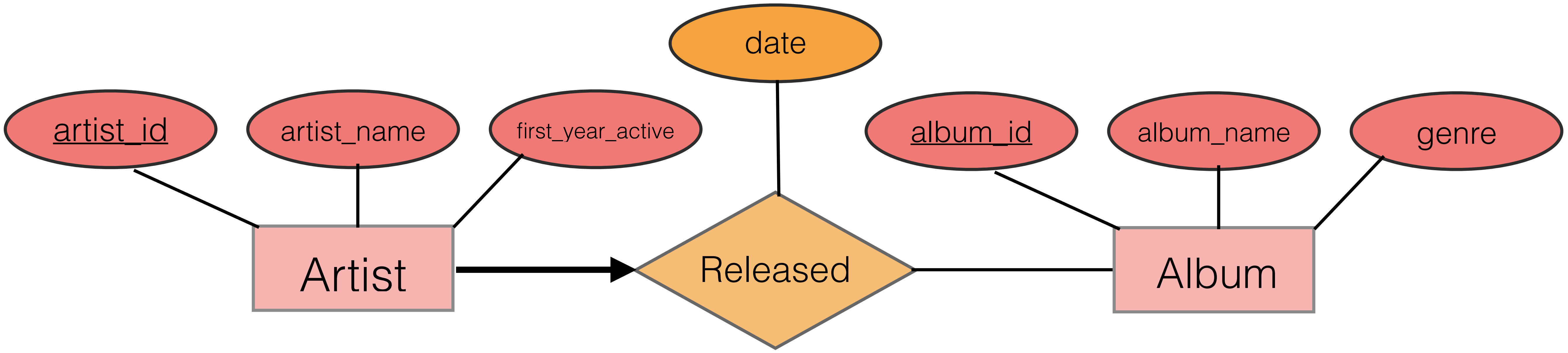
Constraints



Key constraint with partial participation

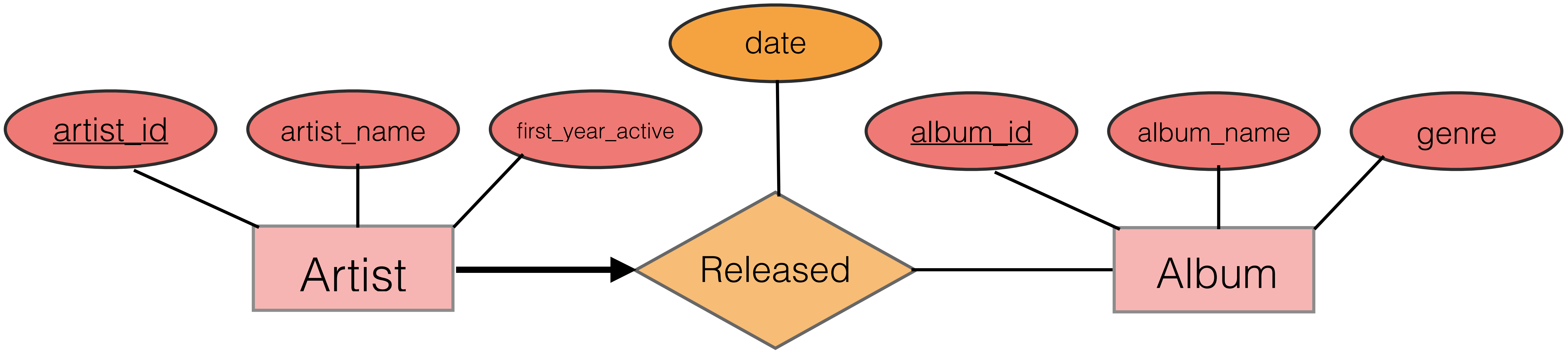
An artist can release an album zero or one times.

Constraints



Key constraint with total participation

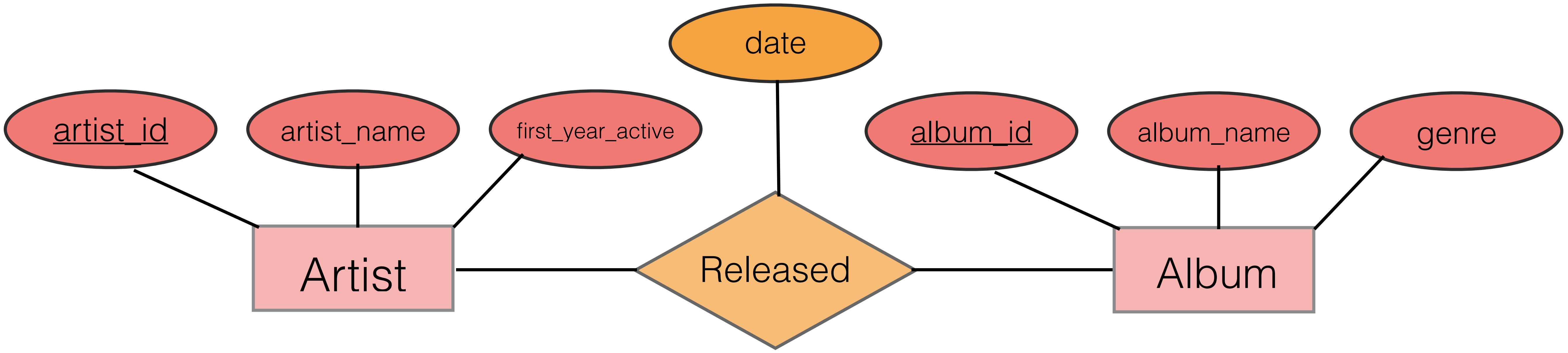
Constraints



Key constraint with total participation

An artist can release exactly one album.

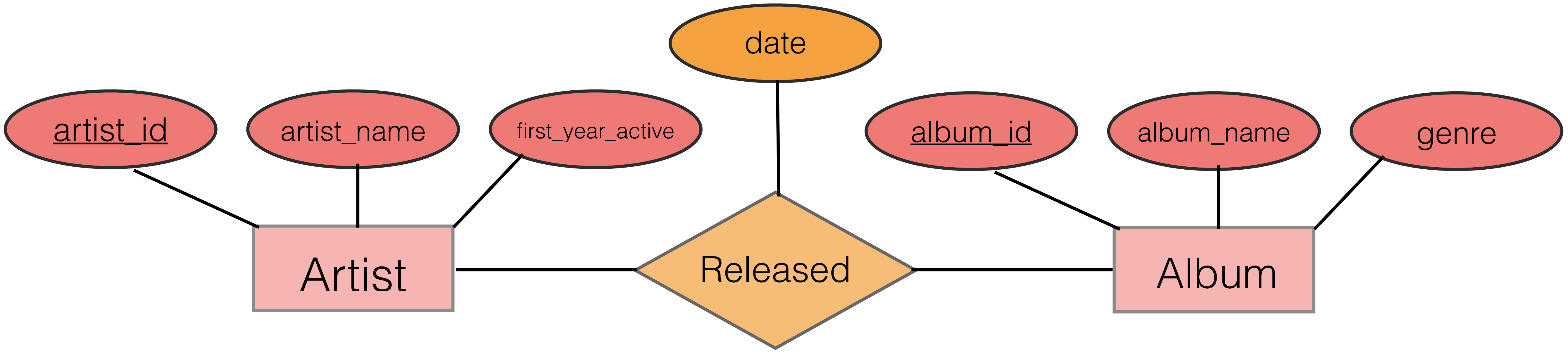
We want…



Non-Key constraint with partial participation

An artist can release an album zero or more times.

What constraint do we want from Album?



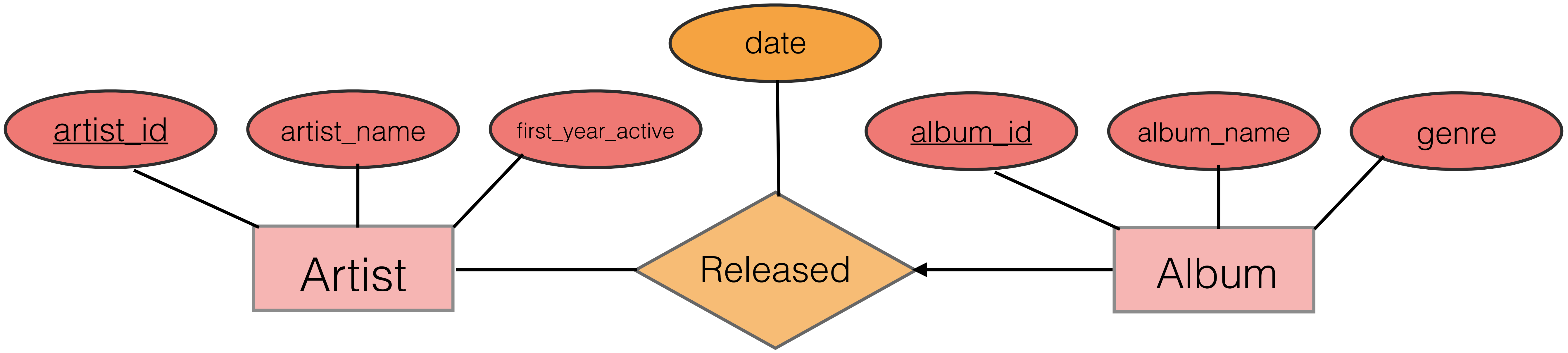
A. An album can be released 0 or more times. ———

B. An album can be released 1 or more times. ━━━

C. An album can be released 0 or 1 times. ⟶

D. An album is released exactly once. ➡

What constraint do we want from Album?

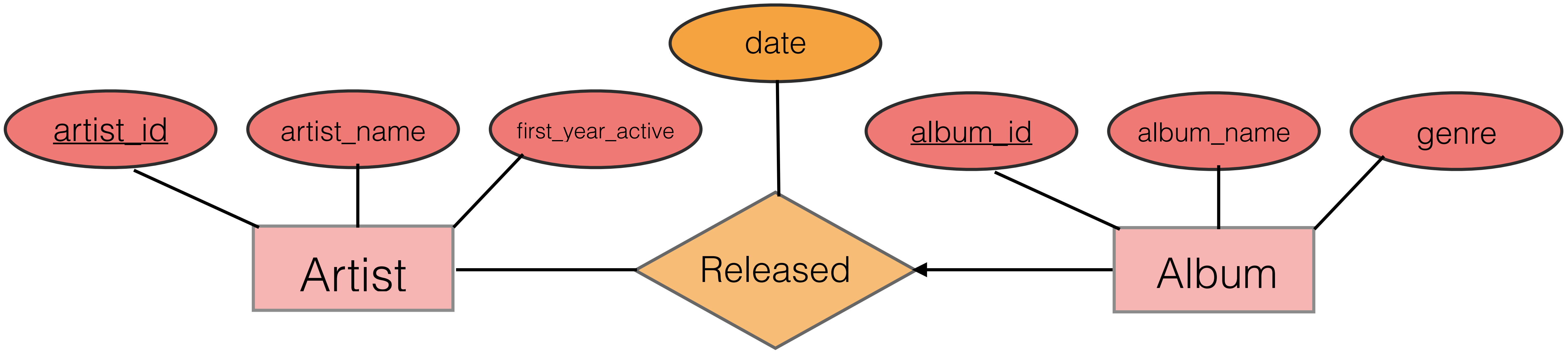


A. An album can be released 0 or more times. ——

B. An album can be released 1 or more times. ━━

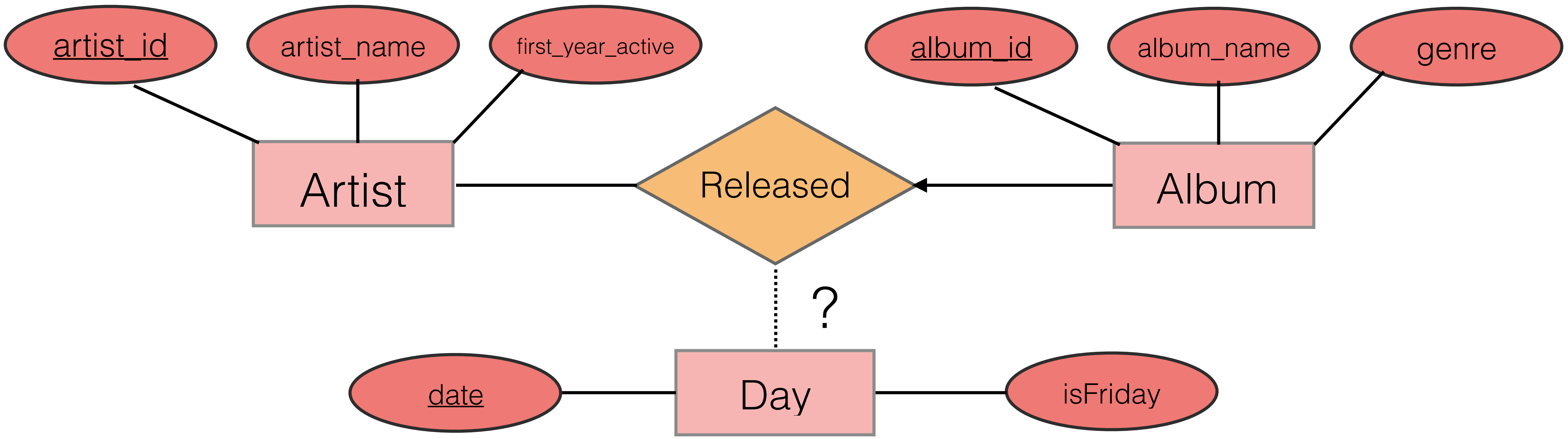
C. An album can be released 0 or 1 times. ⟶

D. An album is released exactly once. ➔



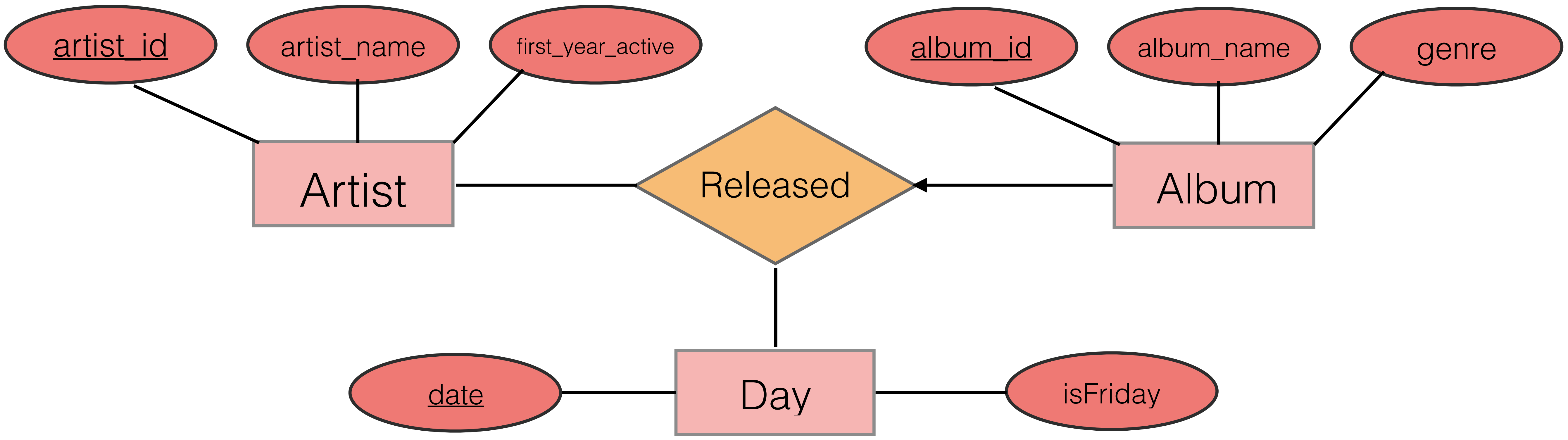
An artist may release zero or more albums.
An album may be released or unreleased.

Ternary Relations



An artist may release zero or more albums.
An album may be released or unreleased.
Releasing an album can occur ??? times a day.

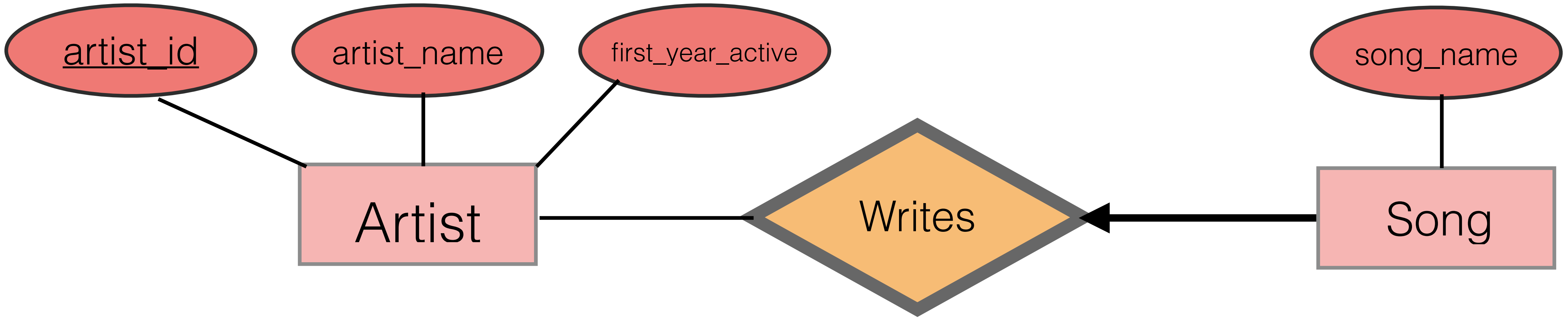
Ternary Relations



An artist may release zero or more albums.
An album may be released or unreleased.
Releasing an album can occur 0 or more times a day.

Weak Entities

- Weak entity can only be identified only when considering primary key of another (owner) entity.



- Song's key is actually (Artist.artist_id, Song.song_name)
- Can there be two songs with the same name?
    - How about by the same artist?
- Can a song exist without an artist?

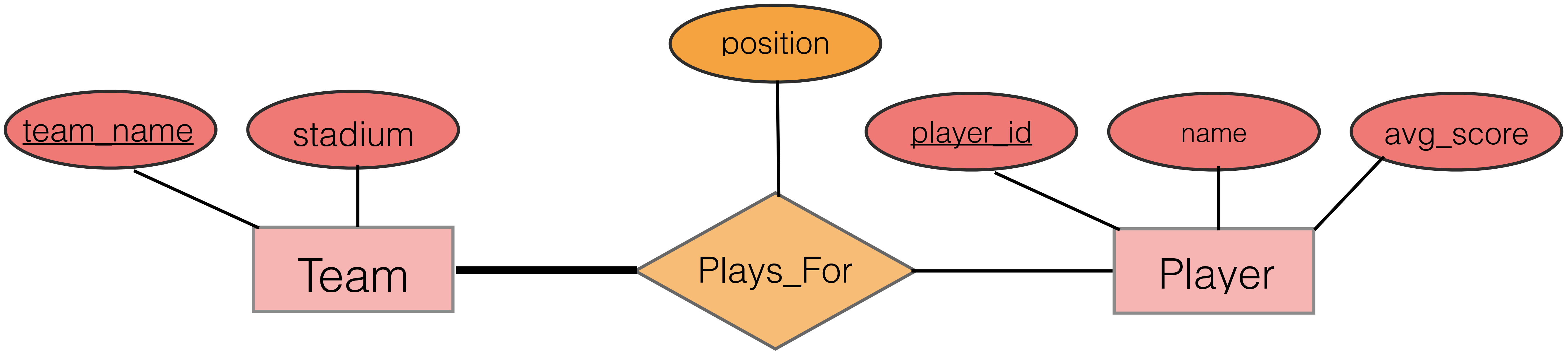
Worksheet #7, 8, 9

Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the "position" that the player plays in.

- Assume that a player can play in more than one team (Yes, our league has different rules!) and that a team needs at least one player.

Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the “position” that the player plays in.

- Assume that a player can play in more than one team (Yes, our league has different rules!) and that a team needs at least one player.

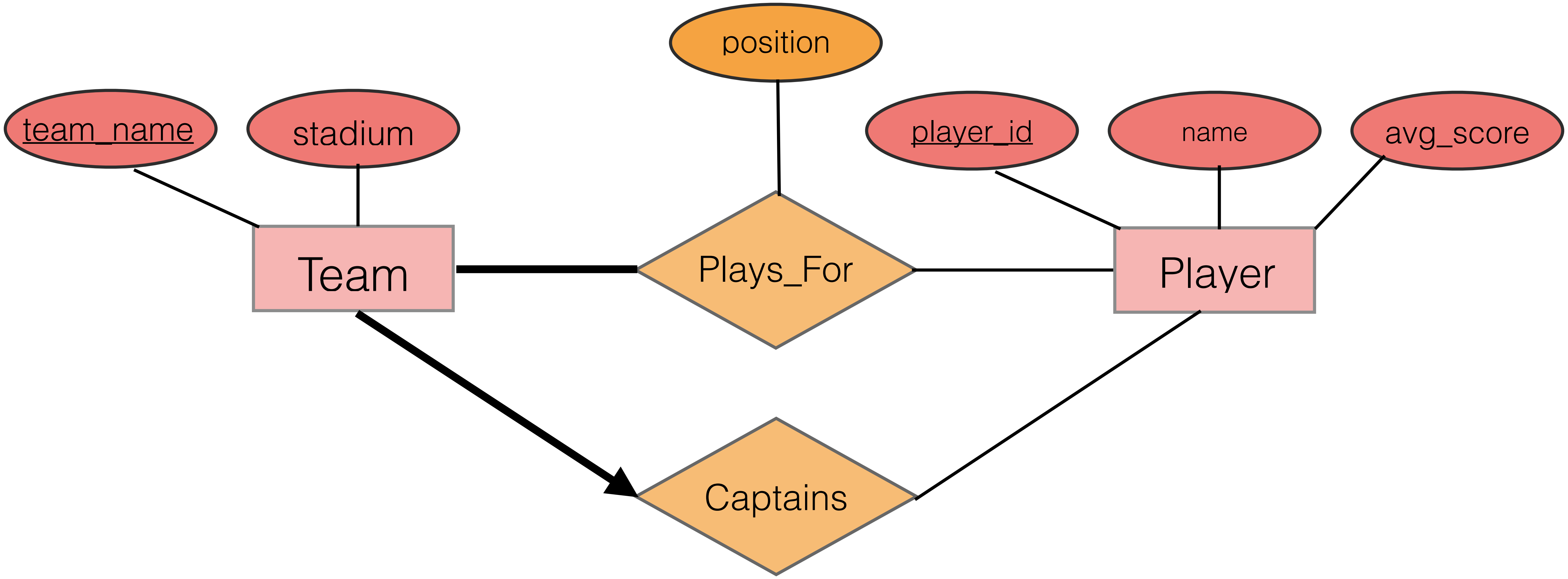


Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the “position” that the player plays in.

- Now let’s say we want to also track who is the captain of every team. How will the ER diagram change from the previous case? Note: Every team needs exactly one captain!

Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the “position” that the player plays in.

- Now let’s say we want to also track who is the captain of every team. How will the ER diagram change from the previous case? Note: Every team needs exactly one captain!

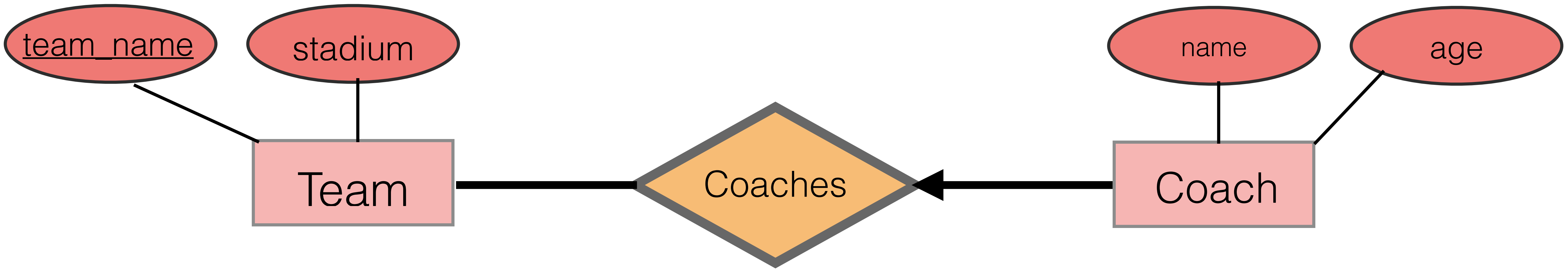


Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the “position” that the player plays in.

- Are there are any weak-entity relationships in either of our ER diagrams?

Every Team in our database will have a team_id, a team_name and a stadium where they play their games. Each Player will have a player_id, name and their average score (This can be used for any sport!). Finally our database will contain who Plays_For which team and also the “position” that the player plays in.

- Are there are any weak-entity relationships in either of our ER diagrams?
- No. A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
- Example of possible weak entity: Coaches



A team can have many coaches, but each coach exactly coaches one team.

Advanced SQL

SELECT [DISTINCT] <column list>
FROM <table name>
[**WHERE** <predicate>]
[**GROUP BY** <column list> [**HAVING**
<predicate>]]
[**ORDER BY** <column list>]
[**LIMIT** <#>];

SELECT [DISTINCT] <column list>
FROM <table name> [INNER | {LEFT | RIGHT| FULL} {OUTER}] JOIN <table name> ON <qualification list>
[**WHERE** <predicate>]…

Inner Join

```
SELECT s.sid, s.name, g.gpa
FROM Students s INNER JOIN Grades g
ON s.sid = g.sid;
```

Students

name	sid
Bob	1
Sue	3
Ron	2

Grades

sid	gpa
1	3.7
2	2.9

Inner Join

```
SELECT s.sid, s.name, g.gpa
FROM Students s INNER JOIN Grades g
ON s.sid = g.sid;
```

s.sid	s.name	g.gpa
1	Bob	3.7
2	Ron	2.9

Left Outer Join

```
SELECT s.sid, s.name, g.gpa
FROM Students s LEFT OUTER JOIN Grades g
ON s.sid = g.sid;
```

Students

name	sid
Bob	1
Sue	3
Ron	2

Grades

sid	gpa
1	3.7
2	2.9

Left Outer Join

```
SELECT s.sid, s.name, g.gpa
FROM Students s LEFT OUTER JOIN Grades g
ON s.sid = g.sid;
```

s.sid	s.name	g.gpa
1	Bob	3.7
2	Ron	2.9
3	Sue	

Right Outer Join

```
SELECT s.name, g.sid, g.gpa
FROM Students s RIGHT OUTER JOIN Grades g
ON s.sid = g.sid;
```

Students

name	sid
Bob	1
Sue	3
Ron	2

Grades

sid	gpa
1	3.7
2	2.9
5	4.0

Right Outer Join

```
SELECT s.name, g.sid, g.gpa
FROM Students s RIGHT OUTER JOIN Grades g
ON s.sid = g.sid;
```

s.name	g.sid	g.gpa
Bob	1	3.7
Ron	2	2.9
	5	4.0

Full Outer Join

```
SELECT s.name, s.sid, g.sid, g.gpa
FROM Students s FULL OUTER JOIN Grades g
ON s.sid = g.sid;
```

Students

name	sid
Bob	1
Sue	3
Ron	2

Grades

sid	gpa
1	3.7
2	2.9
5	4.0

Full Outer Join

```
SELECT s.name, s.sid, g.sid, g.gpa
FROM Students s FULL OUTER JOIN Grades g
ON s.sid = g.sid;
```

s.name	s.sid	g.sid	g.gpa
Bob	1	1	3.7
Sue	3		
Ron	2	2	2.9
		5	4.0

Worksheet #1-6

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all album id’s and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output.

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all album id’s and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output.

-
  ```
  SELECT Ar.artist_id, Ar.artist_name,
  Al.album_id, Al.album_name
  FROM Artists Ar
  LEFT OUTER JOIN Albums Al
  ON Ar.artist_id=Al.artist_id
  WHERE Ar.first_year_active >= 2000;
  ```

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find the id and name for each song released in the first year that the artist for the album was active.

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find the id and name for each song released in the first year that the artist for the album was active.

-
  ```
  SELECT S.song_id, S.song_name, Ar.artist_id,
  Ar.artist_name
  FROM Songs S
  INNER JOIN Albums Al
  ON S.album_id=Al.album_id
  INNER JOIN Artists Ar
  ON Al.artist_id=Ar.artist_id
  AND Al.year_released=Ar.first_year_active;
  ```

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find the id and name for each artist who has albums of genre “pop” and “rock”.

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find the id and name for each artist who has albums of genre “pop” and “rock”.

-
  ```
  SELECT Ar.artist_id, Ar.artist_name
  FROM Albums Al1
  INNER JOIN Albums Al2 ON
  Al1.artist_id=Al2.artist_id
  INNER JOIN Artists Ar ON
  Al2.artist_id=Ar.artist_id;
  WHERE Al1.genre="pop"
  AND Al2.genre="rock"
  ```

What is the difference between INNER JOIN and an implicit join (writing the join predicate as part of the WHERE statement)?

What is the difference between INNER JOIN and an implicit join (writing the join predicate as part of the WHERE statement)?

- Semantically, no difference.
- INNER JOIN makes the join more readable, especially if you have a lot of predicates in the WHERE statement.
- Since INNER JOIN requires an ON predicate, it is more difficult to forget a join condition

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all artists who released more songs in 2014 than Taylor Swift. Include the number of songs that each artist released.

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all artists who released more songs in 2014 than Taylor Swift. Include the number of songs that each artist released.

```
• WITH Taylor2014(cnt) AS
    (SELECT COUNT(*)
    FROM Songs S
    INNER JOIN Albums Al ON S.album_id=Al.album_id
    INNER JOIN Artists Ar ON Al.artist_id=Ar.artist_id
    WHERE Al.year_released=2014
    AND Ar.artist_name="Taylor Swift")
SELECT Ar.artist_id, Ar.artist_name, COUNT(*)
FROM Songs S, Taylor2014 T
INNER JOIN Albums Al ON S.album_id=Al.album_id
INNER JOIN Artists Ar ON Al.artist_id=Ar.artist_id
WHERE Al.year_released=2014
GROUP BY Ar.artist_id
HAVING COUNT(*) > T.cnt;
```

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all artists who released songs in 2014 or released songs that spent more than 10 weeks in the top 40.

```
Songs(song_id, song_name,album_id,
weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id,  album_name,  artist_id,
year_released, genre)
```

- Find all artists who released songs in 2014 or released songs that spent more than 10 weeks in the top 40.

```
• WITH Temp(artist_id) AS
    (SELECT Al.artist_id
    FROM Albums Al
    WHERE Al.year_released=2014
    UNION
    SELECT Al.artist_id
    FROM Albums Al
    INNER JOIN Songs S ON Al.album_id=S.album_id
    WHERE S.weeks_in_top_40>10)
SELECT Ar.artist_id, Ar.artist_name
FROM Artists Ar
INNER JOIN Temp T ON Ar.artist_id=T.artist_id;
```