

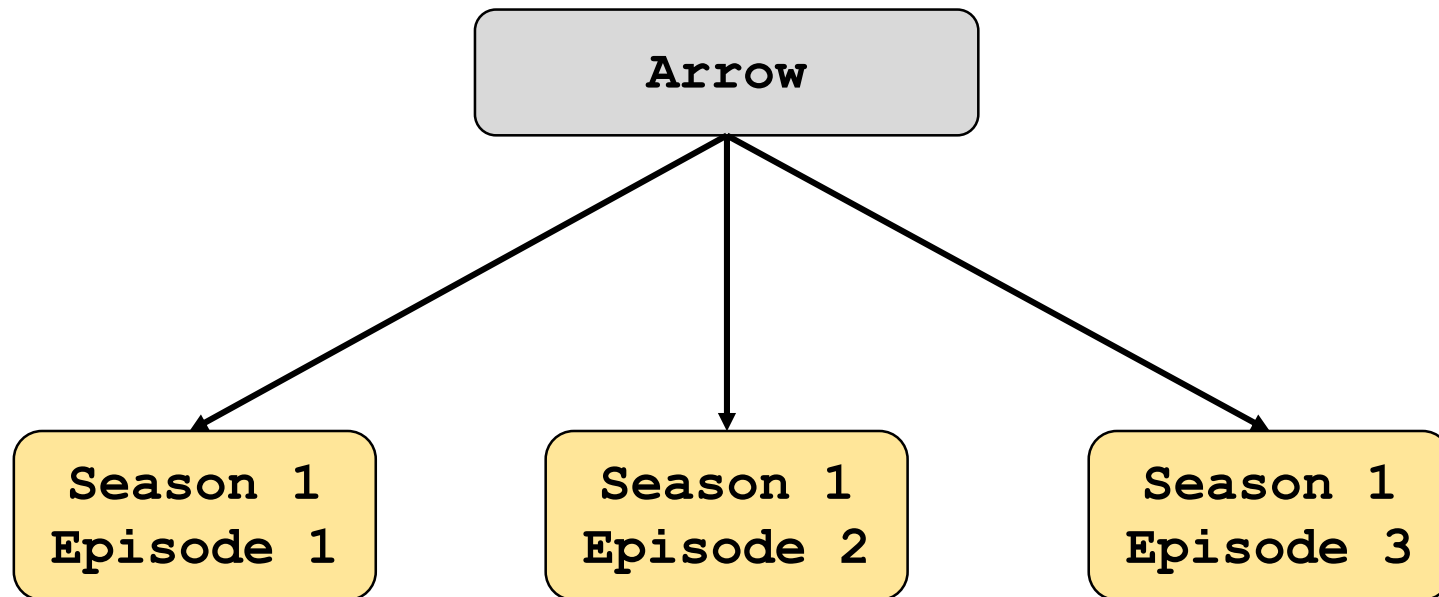


# Day 23

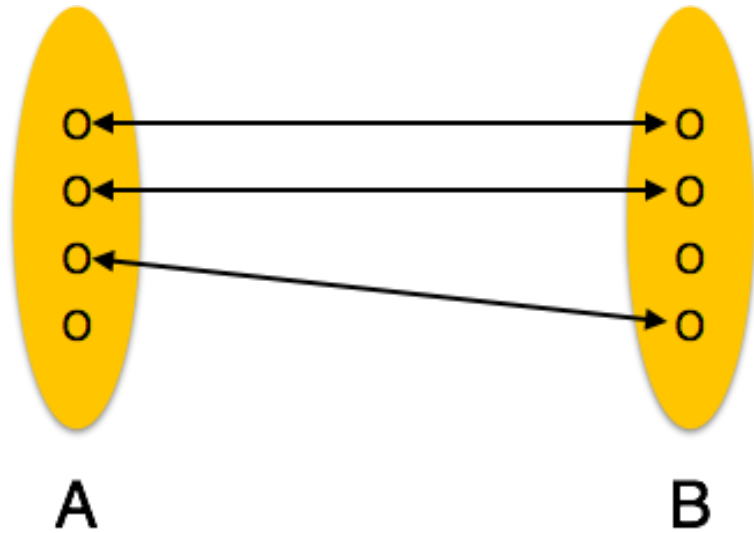


# Relationships

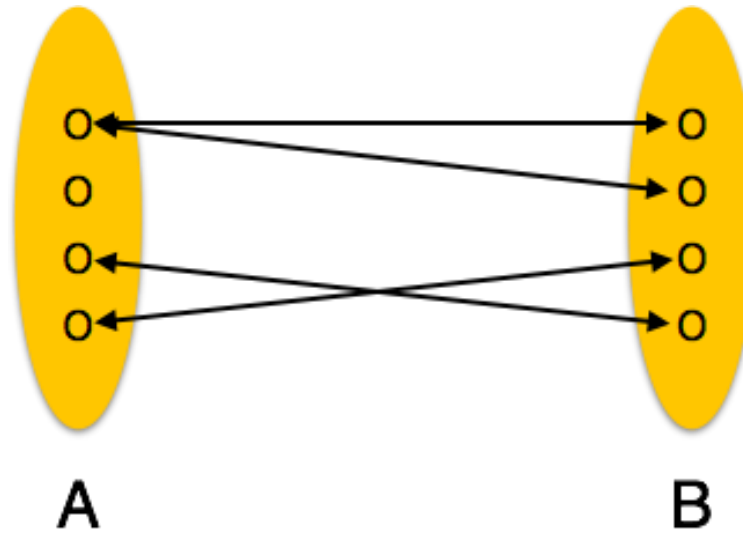
- Defines association between entities/tables
  - Eg. A TV program and the episodes



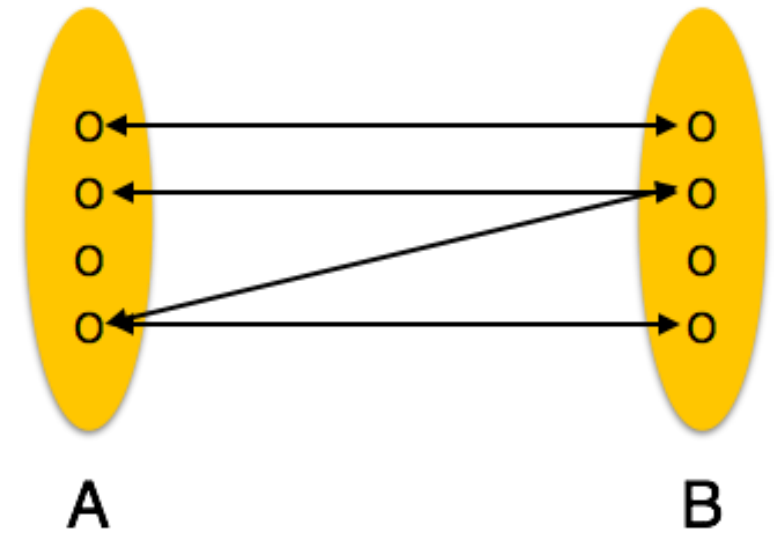
# Relationship Cardinality



One to one



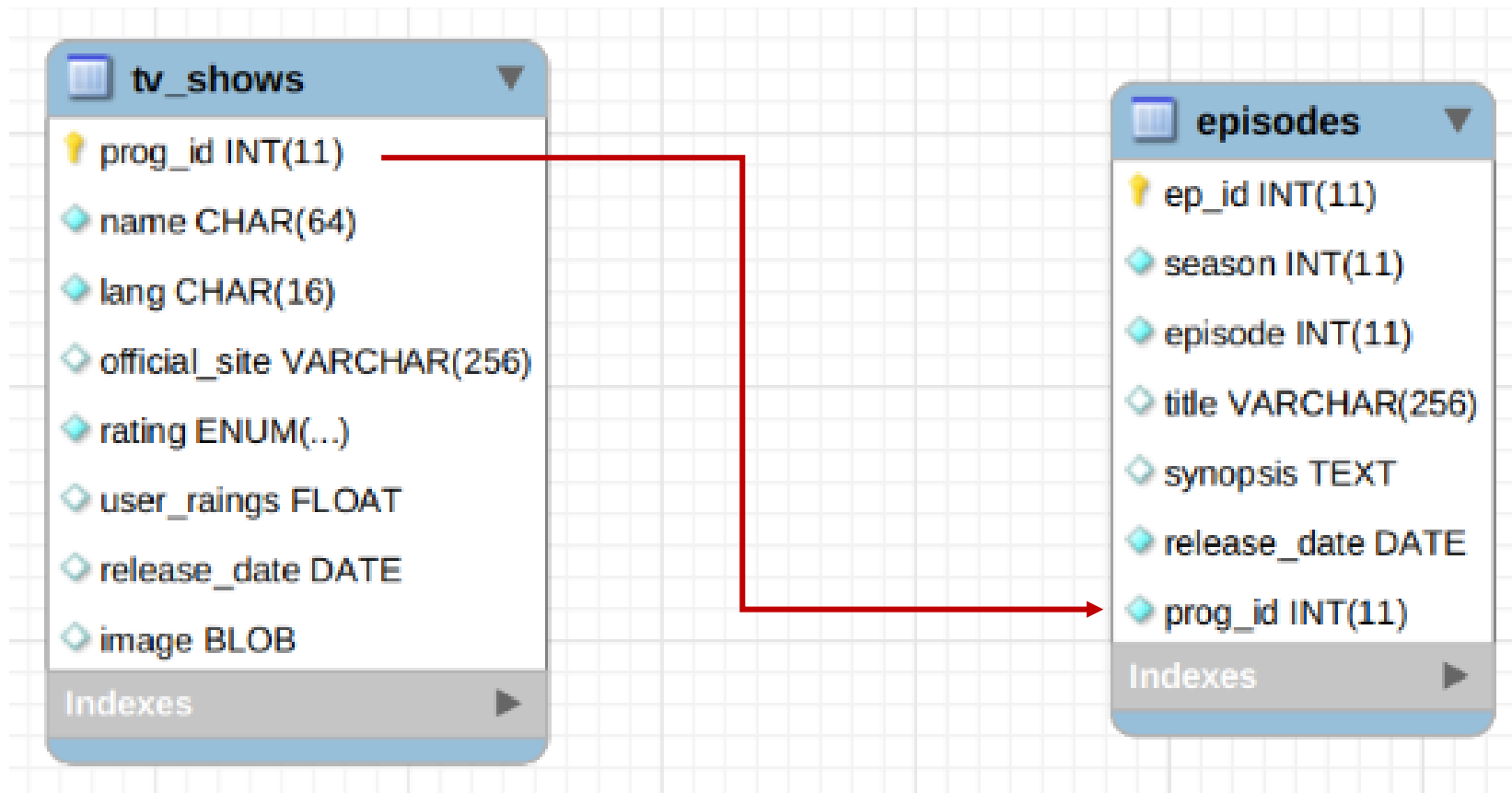
One to Many/  
Many to one



Many to many



# Relating Tables



Relate tables by embedding the primary key of one table into another



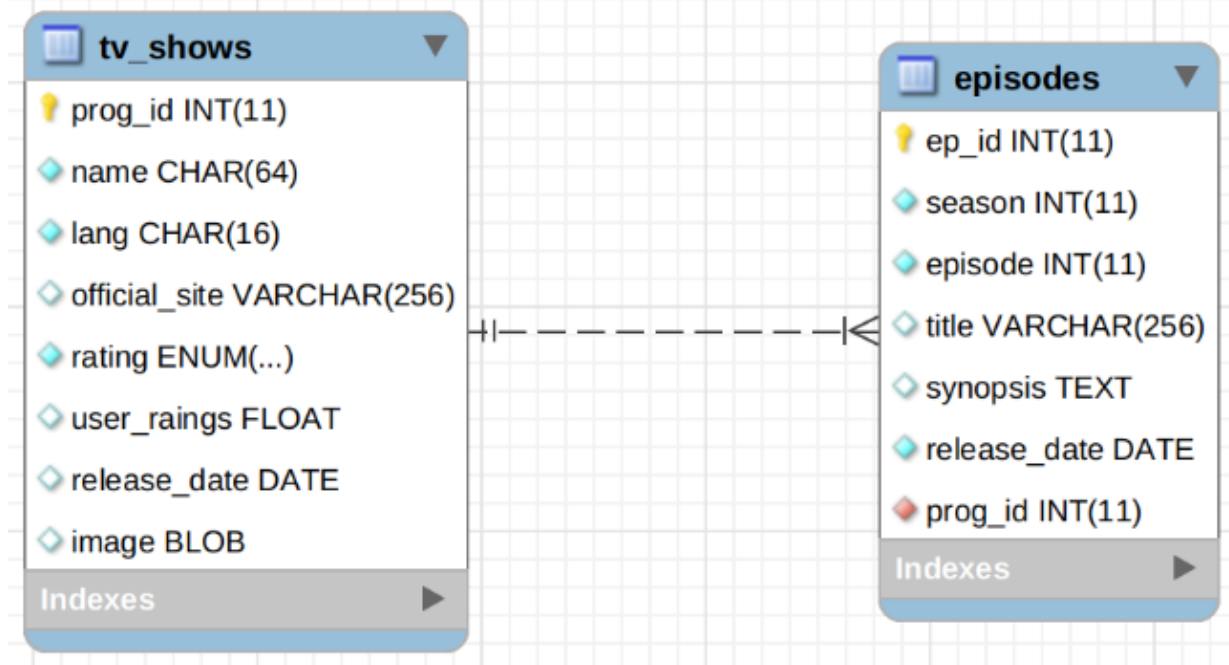
# Foreign Key

- A field in a table that references the primary key of another table
  - The foreign key field stores only values found in the primary key field
  - Called the foreign key constraint
- Referential integrity - foreign key constraint
  - Cannot delete a record if there are foreign keys referencing it
  - Cannot insert values in a foreign key field that are not present in the referencing primary key field
- Foreign key field
  - Do not have to be unique
  - Can be null even if the primary key field that it references is not



# Table with Foreign Key

```
create table episodes (  
    ep_id          int(11) auto_increment,  
    season         int not null,  
    episode        int not null,  
    title          varchar(256),  
    synopsis       text,  
    release_date   date not null,  
    prog_id        int not null,  
  
    primary key(ep_id),  
    constraint fk_prog_id  
        foreign key(prog_id)  
        references tv_shows(progs_id)  
);
```

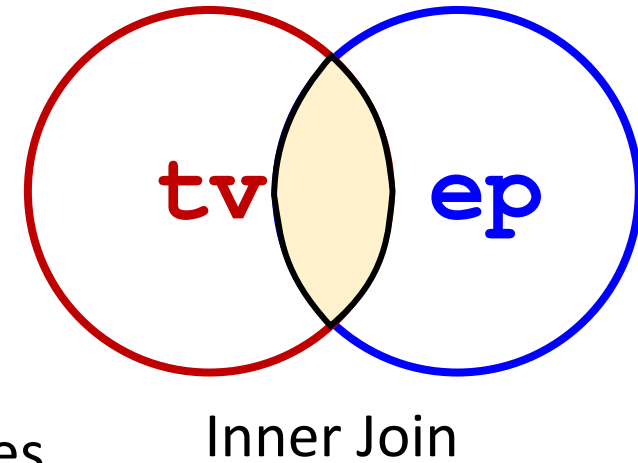


'Link' prog\_id field in episodes table to the prog\_id column in tv\_shows table



# Joins

- Combining 2 or more tables to produce a new table
  - Typically based on some common fields between the 2 tables
  - Eg. tables are related with a foreign key
- Default joins are inner join



```
select tv.name, tv.lang, ep.season, ep.episode, ep.title,  
       from tv_shows as tv  
       inner join episodes as ep  
       on tv.tv_id = ep.tv_id;
```



# Inner Join Examples

```
select tv.name, tv.lang, ep.season, ep.episode, ep.title,  
       from tv_shows as tv  
       join episodes as ep  
       on tv.tv_id = ep.tv_id  
       where tv.name like "%New%"
```

List all episodes from TV programs  
with the word **New** in its title

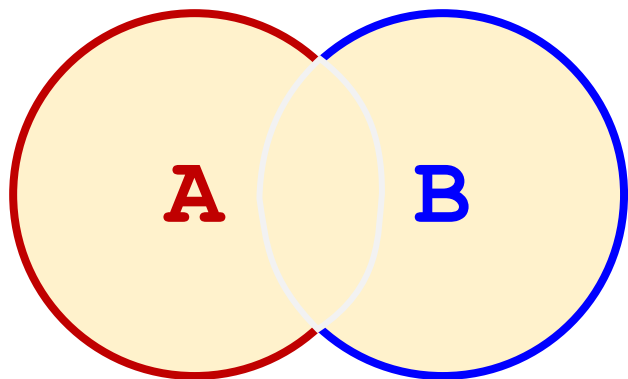
```
select tv.name, count(distinct ep.season)  
       from tv_shows as tv  
       join episodes as ep  
       on tv.tv_id = ep.tv_id  
       group by tv.name
```

Count the number of seasons aired  
for each TV program

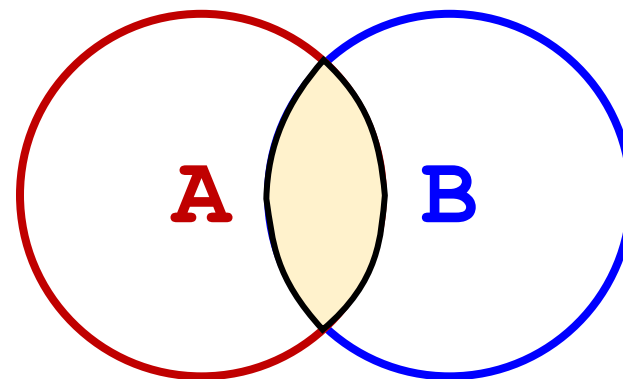




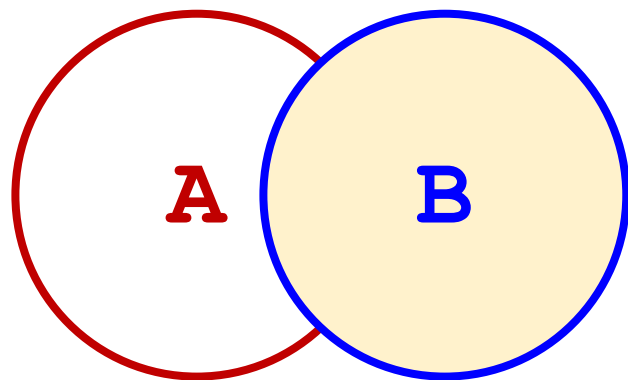
# Types of Joins



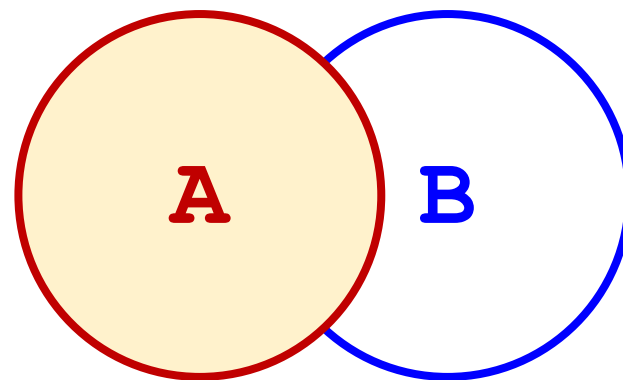
Full Join



Inner Join



Right Join

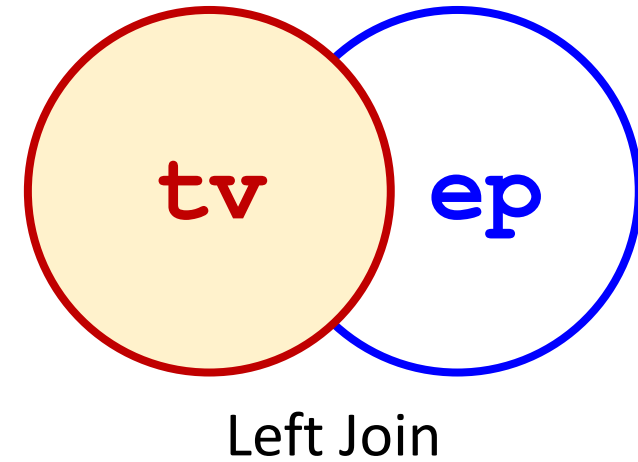


Left Join



# Left Join

Return rows from the left table even if there are no matches in the right table

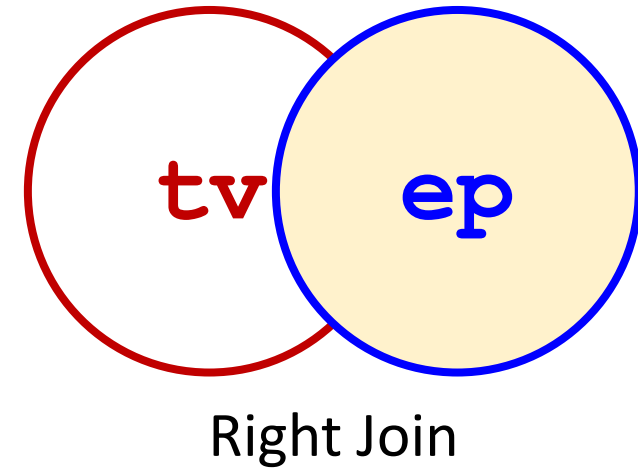


```
select tv.name, tv.lang, ep.season, ep.episode, ep.title,  
       from tv_shows as tv  
       left join episodes as ep  
       on tv.tv_id = ep.tv_id  
       where tv.name like "%New%"
```



# Right Join

Return rows from the right table even if there are no matches in the left table

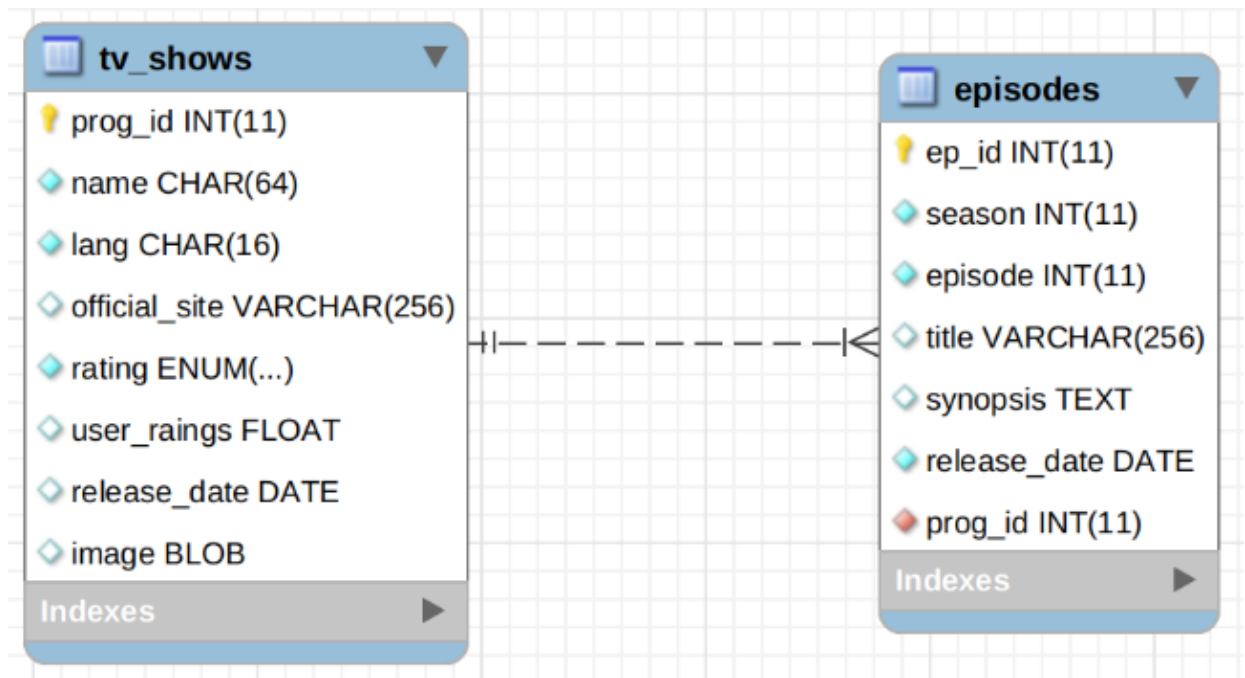


```
select tv.name, tv.lang, ep.season, ep.episode, ep.title,  
       from tv_shows as tv  
       right join episodes as ep  
       on tv.tv_id = ep.tv_id  
       where tv.name like "%New%"
```



# Relationship and Joins

- Foreign keys are constraints to enforce referential integrity
- If you based a join on between 2 tables on their foreign key, then the foreign key provide some guarantees that there will always be result
  - Eg. `tv_shows` will always return `episodes`





# Referential Integrity

- Foreign keys constraint prevents a parent key from being delete or update
  - Eg. Delete a record in `tv_program` table where there are references from `episodes`
  - Eg. Update/change the `prog_id` in `tv_program` table where there are references from `episodes` table
  - Default behaviour

Default behaviour  
can be omitted

```
constraint fk_prog_id
foreign key(prog_id)
references tv_shows(prog_id)
on delete restrict
on update restrict
```



# Foreign Key Behaviour

- Cascade - apply the action to all records that references the primary key as foreign key

- Eg. delete all records that references the parent key when the parent key is delete

```
constraint fk_prog_id
foreign key(prog_id)
references tv_shows(prog_id)
on delete cascade
on update restrict
```

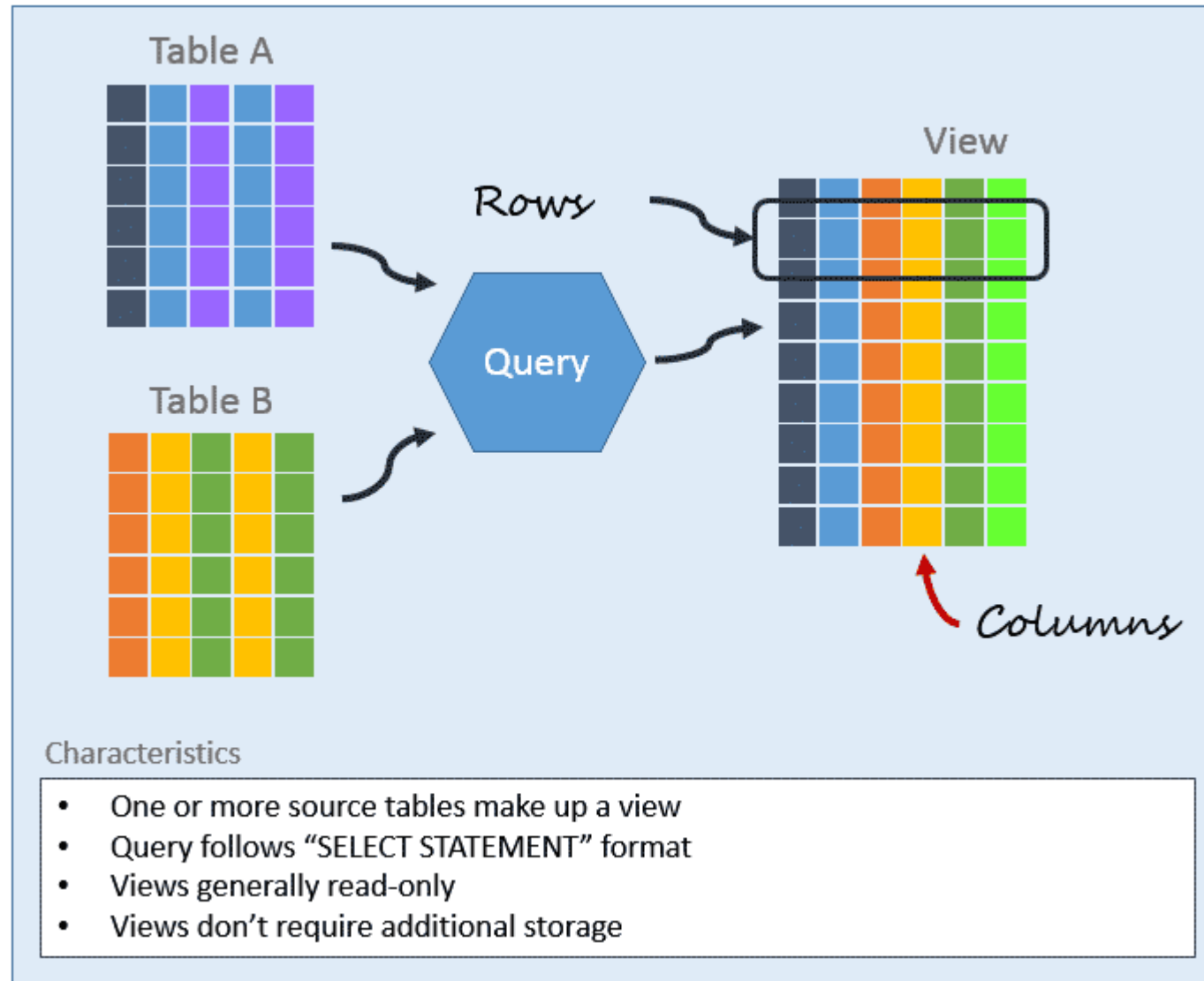
- Set null - set the foreign key column to null when the parent key is deleted

```
constraint fk_prog_id
foreign key(prog_id)
references tv_shows(prog_id)
on delete set null
on update restrict
```



# Database Views

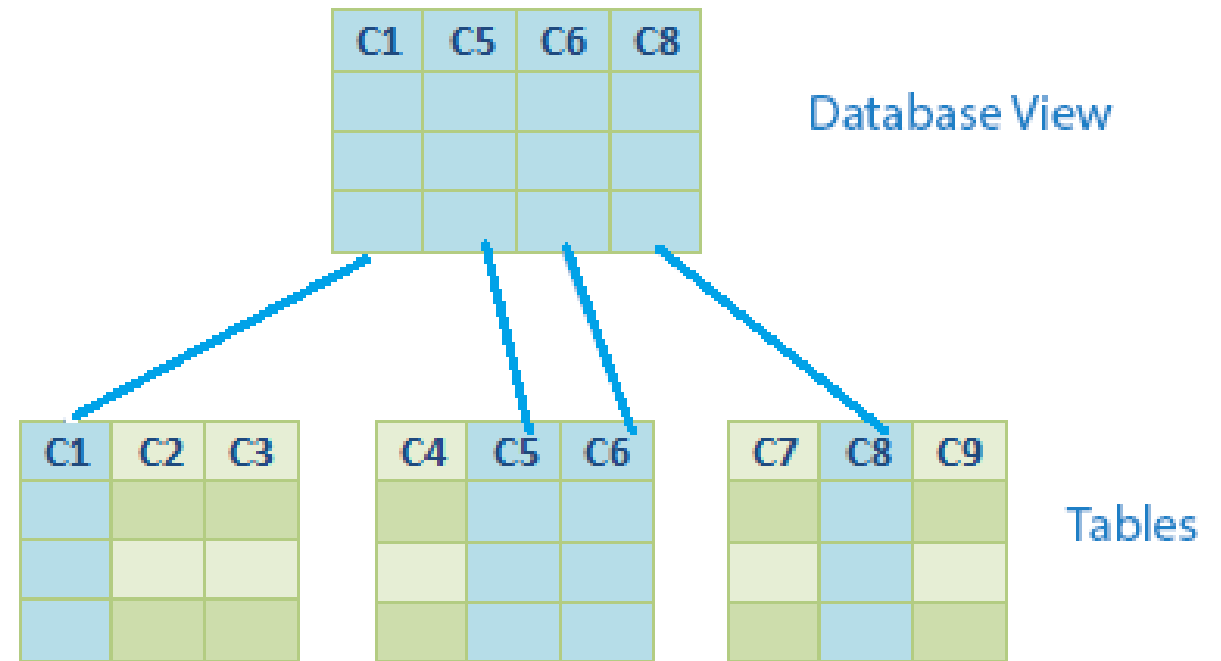
## Anatomy of a View





# Database Views

- View is a table that is defined from one or more queries
- The table is computed when it is accessed
  - No data are stored
- Use cases
  - Providing a conditional view of a table
    - Eg. this months orders
  - Aggregated results from a join
    - Eg. list of all inventories with warehouse location and quantities







# Views Example

View's name

Query that defines the view

Query

```
create view aug_shows as
select * from tv_shows tv
where tv.release_date > '2019-08-01'
```

Query

```
create view shows_episodes as
select ep.ep_id, as id, tv.name as name,
       ep.season as season, ep.episode as episode,
       ep.title as title
from tv_shows tv
join episodes ep
on tv_shows.prog_id = ep.prog_id
```

Good practice to provide names to columns from a view



# Modifying Views

- Views derived from a single query can be modified

```
create view shows_general as
  select * from tv_shows tv
    where tv.rating like 'G' }
```

Single query

```
insert into shows_general(prog_id, name, ...)
  values (100, 'SpongeBob SquarePants' ...)
```

- View derived from joins or aggregation cannot be modified

```
create view episodes_per_season as
  select tv.name as name,
    ep.season as season, count(ep.episode) as episodes
  from tv_shows tv join episodes as ep
    on tv.prog_id = ep.prog_id
  group by tv.name, ep.season
```



# Normalization

- The process of eliminating redundancy from the database
- It is a series of steps performed on one or more tables to ensure that every field in a row of record can be found by just the primary key
  - Non-key column is directly dependent on the primary key
- The result is eliminating redundancies
  - Fewer or no data anomalies since you only have to update the data once
  - More efficient updates



# First Normal Form - 1NF

- Every row must be unique
  - Table must have primary key
  - Can create table without primary key
- No repeating same group of attributes viz. should not have multiple columns with the same attribute
  - Eg. multiple columns for different email addresses

```
create table supplier (  
    supplier_id char(10) primary key,  
    name varchar(255) non null,  
    products varchar(255)  
);
```

↓  
"mouse, keyboard, USB cable"

```
create table supplier (  
    supplier_id char(10) primary key,  
    name varchar(255) non null,  
    product0 varchar(32), ← "mouse"  
    product1 varchar(32) ← "keyboard"  
    product2 varchar(32) ← "USB cable"  
);
```



# First Normal Form - 1NF

supplier_id	name	prod_name
acme	ACME Corp	Jet Propelled Unicycle
orsbone	Orsbone Corp	Globlin Sparks, Bat Hoverboard
acme	ACME Corp	Triple-Strength Fortified Leg Muscle Vitamins



Column with repeating values



# First Normal Form - 1NF

```
create table supplier (  
    supplier_id char(10) primary key,  
    name varchar(255) non null  
);
```

supplier_id	name
acme	ACME Corp
orsbone	Orsbone Corp

```
create table product (  
    product_id char(10) primary key,  
    name varchar(255) non null,  
    supplier_id char(10) non null,  
    constraint fk_supplier_id  
        foreign key(supplier_id)  
        references supplier(supplier_id)  
);
```

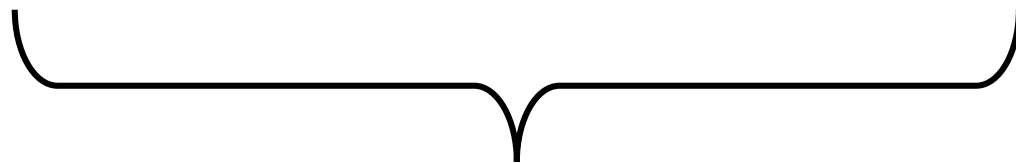
product_id	name	supplier_id
0	Goblin Sparks	orsbone
1	Bat Hoverboard	orsbone
2	Jet Propelled Unicycle	acme
3	Triple-Strength Fortified Leg Muscle Vitamins	acme



# Second Normal Form - 2NF

- Must be in 1NF
- If the non primary key fields are dependent on the primary key(s)
  - Primary key(s) - may be composite

warehouse_id	product_id	location	quantity
tuas	1	Tuas	100
woodlands	1	Woodlands	100
tuas	2	Tuas	100



Composite primary key


location field is dependent only on  
warehouse\_id not on product\_id



# Second Normal Form - 2NF

warehouse_id	location
tuas	Tuas
woodlands	Woodlands

Primary key



warehouse_id	product_id	quantity
tuas	1	100
woodlands	1	100
tuas	2	100

Composite primary key

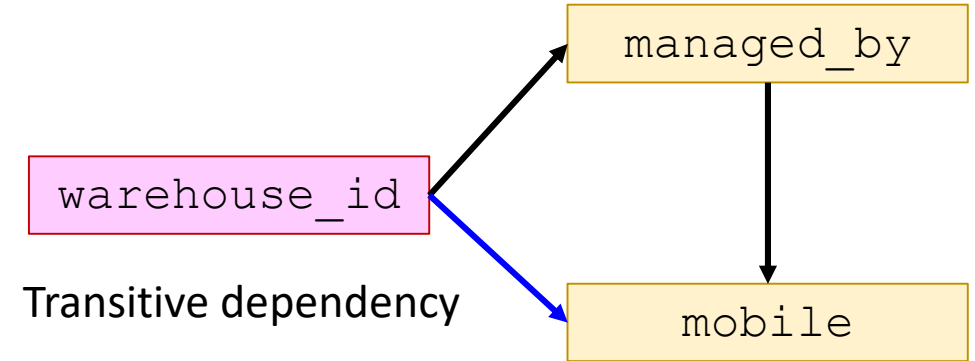






# Third Normal Form - 3NF

- Must be in 2NF
- Has no transitive dependencies
  - The non primary key fields must be dependent on only the primary key(s)
  - Changing one field may cause changes to another field



warehouse_id	location	managed_by	mobile
tuas	Tuas	Alfred	555-1234
woodlands	Woodlands	Jeeves	555-6789

Primary key

We know that Tuas is managed by Alfred whose phone number is 555-1234.

So we can infer the `mobile` number from `managed_by` which is not a primary key



# Third Normal Form - 3NF

warehouse_id	location	manager_id
tuas	Tuas	1
woodlands	Woodlands	2

Primary key

manager_id	name	mobile
1	Alfred	555-1234
2	Jeeves	555-6789



# Appendix



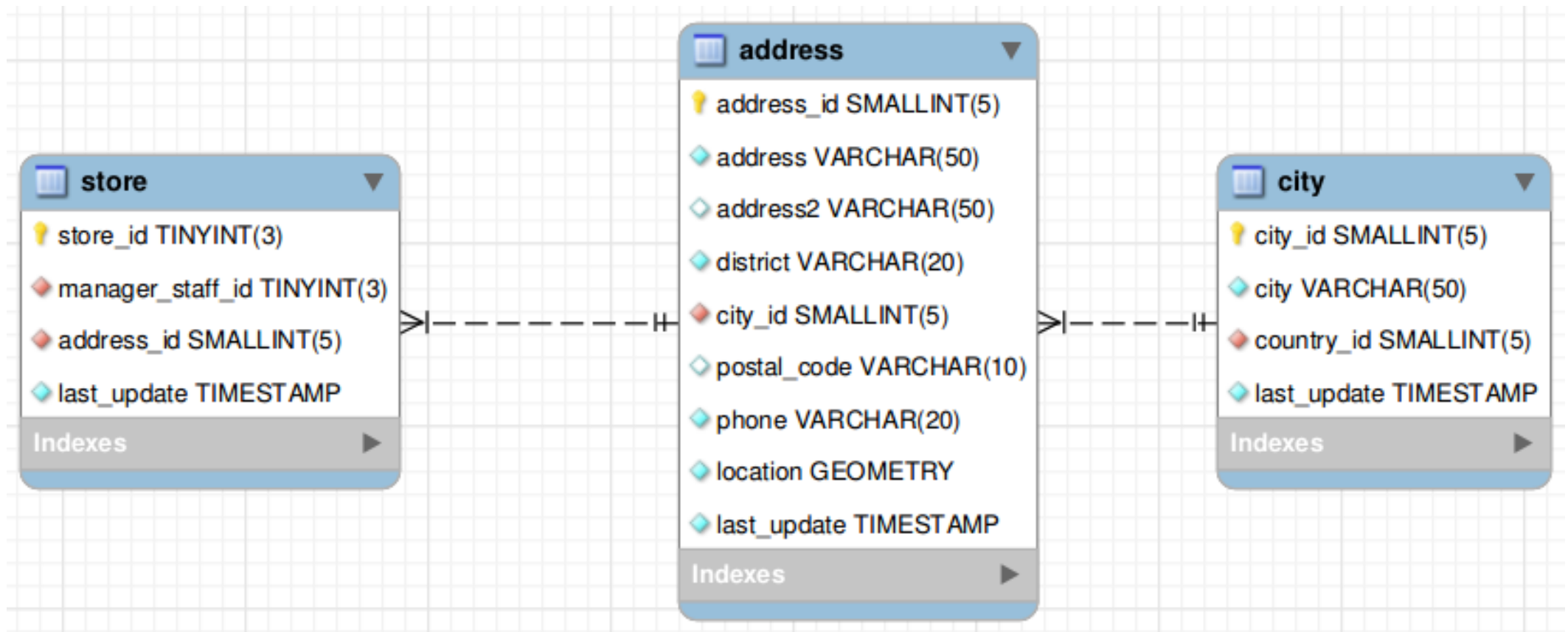
# Sub Queries

- Queries within a query
  - To be used by the outer query
- The inner query must be a select statement enclosed in parenthesis
- The inner query can return
  - A single value viz. single row, single column
  - A single row, viz. multiple row, single columns
  - A table viz. multiple rows, multiple columns
    - Temporary table



# Example - Single Value

Find all the store from a particular city eg. Lethbridge





# Example - Single Value

```
select city.city_id  
  from city  
 where city.city = 'Lethbridge'
```

← Returns a single value

```
select *  
  from store  
 left join address  
 where address.city_id = (  
     select city.city_id  
     from city  
     where city.city = 'Lethbridge'  
 )
```

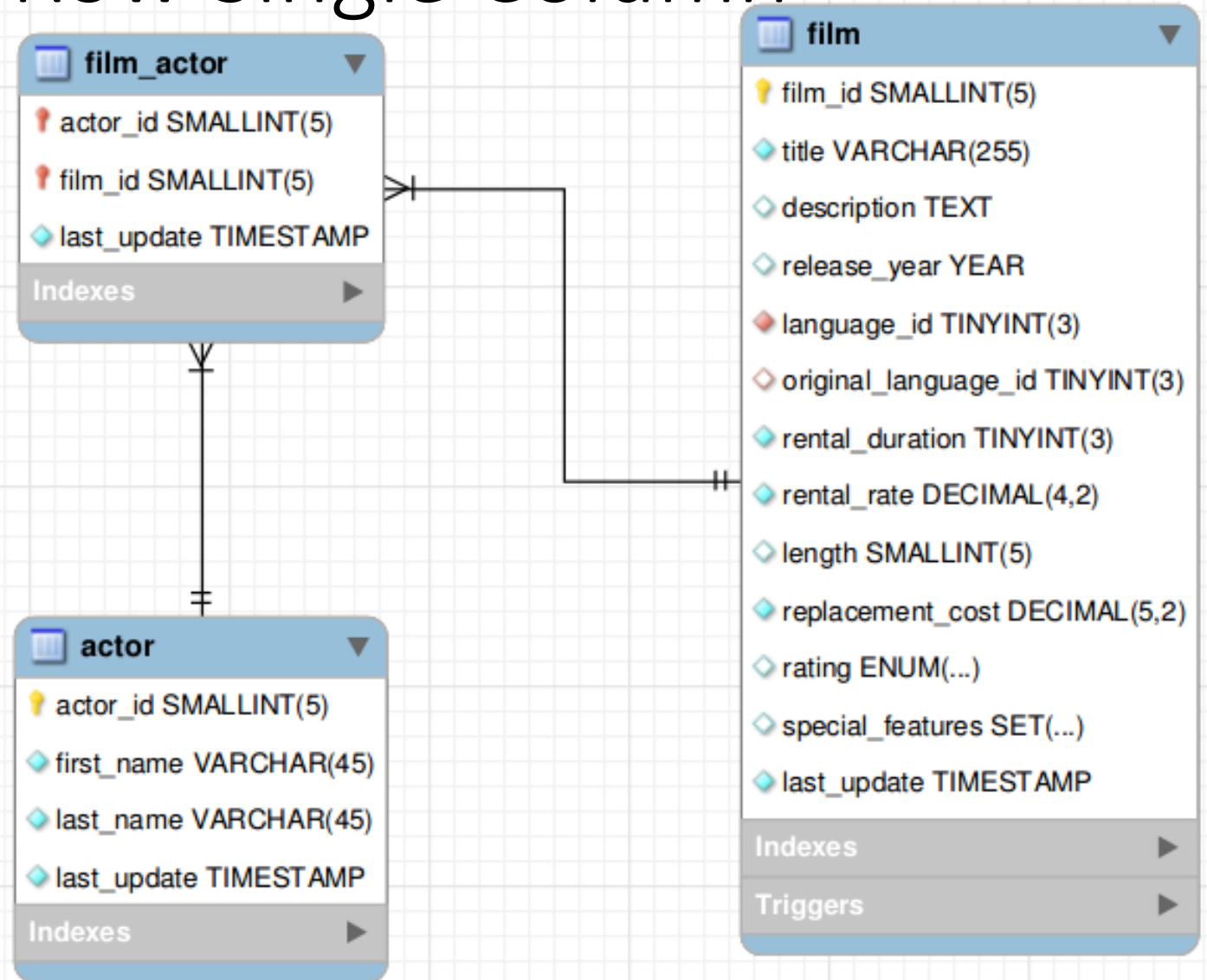
Inner query returns a single value  
which is used by the outer query  
in the where clause

} Inner query inside  
a parenthesis



# Example - Multiple Row Single Column

Get all the movies with  
actors whose last name is  
Davis





# Example - Multiple Row Single Column

```
select distinct film_actor.film_id
  from actor
 right join film_actor
on actor.actor_id = film_actor.actor_id
where actor.last_name like 'Davis'
```

Inner query returns a  
single column which is  
used to check membership

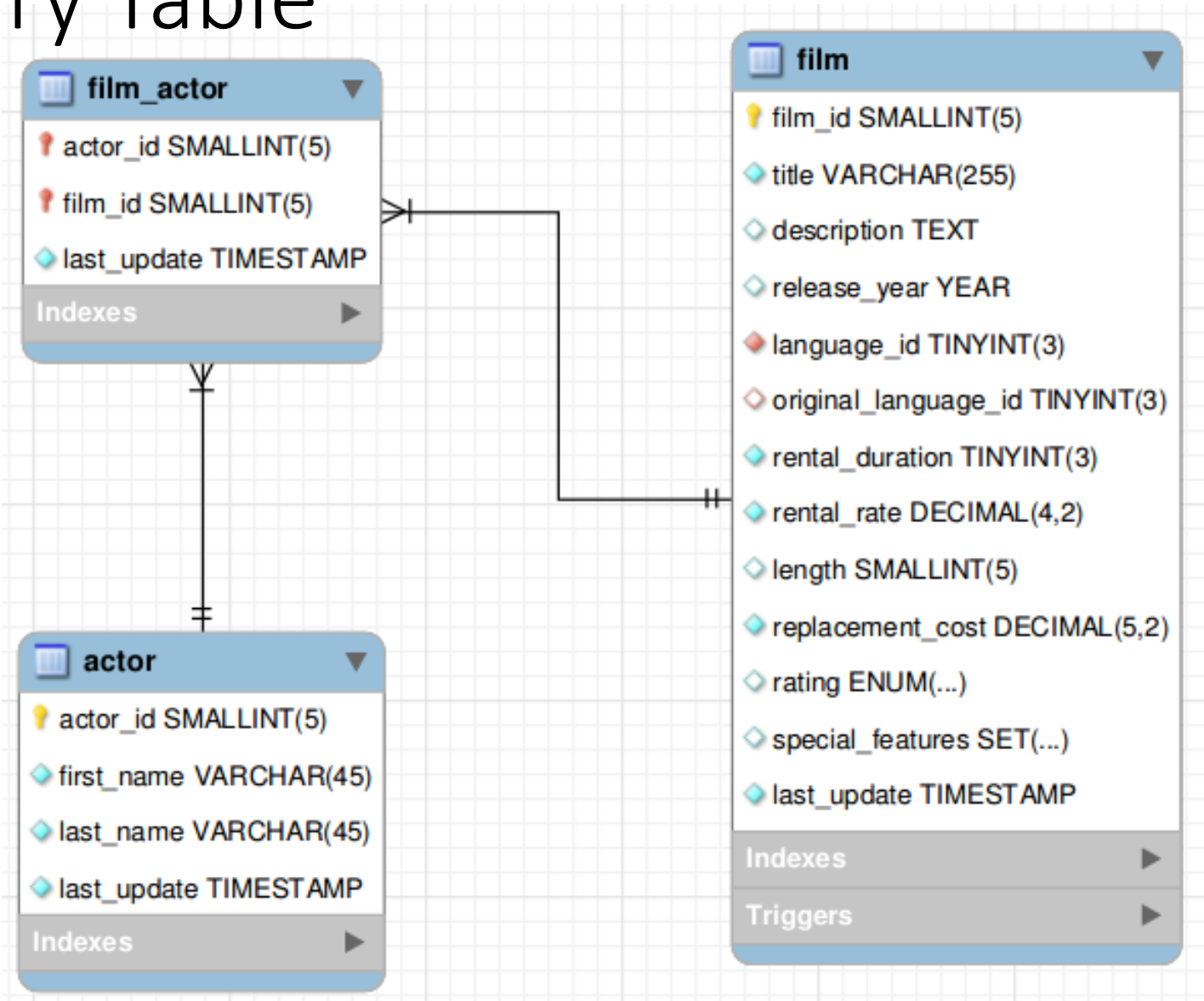
```
select film.title
  from film
 where film.film_id in (
    select distinct film_actor.film_id
      from actor
 right join film_actor
on actor.actor_id = film_actor.actor_id
where actor.last_name like 'Davis'
  )
```





# Example - Temporary Table

Get all the actors who has appeared in more than 25 films





# Example - Temporary Table

Inner query returns a temporary table. All components of the table must be aliased

```
select film_count_tbl.firstname, film_count_tbl.lastname
  from (
    select actor.last_name as lastname,
           actor.first_name as first_name,
           count(film_actor.film_id) as film_count
      from actor
     left join film_actor
        on actor.actor_id = film_actor.actor_id
     group by lastname, firstname
  ) as film_count_tbl
 where film_count_tbl.film_count > 25
```