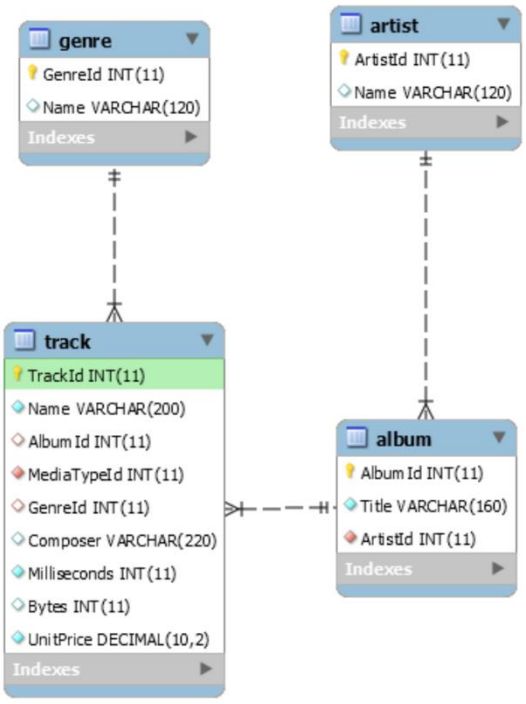
Views – Updateable views

Many views are updateable, but some are not. The MySQL manual lists these conditions which limit the updateability of a view. The first rule of thumb should be obvious. If the view does not include a column from the base tables or excludes rows, then these data elements are inaccessible.

```
create or replace view song_view as
select t.Name as song_name, t.Composer as composer,
       t.Milliseconds as milliseconds, t.UnitPrice as unit_price,
       g.Name as genre, al.Title as album_titie, ar.Name as artist
from   chinook.track t
       join chinook.genre g using (genreid)
       join chinook.album al using (albumid)
       join chinook.artist ar using (artistid)
where  g.Name not in ('Opera', 'Easy Listening', 'Drama', 'Comedy', 'TV Shows');
```



Now we can write queries against the view:

```
select distinct genre from song_view where genre like 'P%';
```

genre
Pop

INSERT statements

But the insert statement below fails with this error: Error Code: 1393. Cannot modify more than one base table through a join view 'chinook.song_view'

This should not be surprising because there is a specific order of inserts due to foreign key constraints. First a genre and an artist must be created; then an album, then a track.

```
insert into song_view (song_name, composer, milliseconds, unit_price, genre, album_title, artist)
values ('Teenage Dirtbag', 'Brendan Brown', 247000, 0.99, 'Power Pop', 'Wheatus', 'Wheatus');
```

```
Error Code: 1393. Cannot modify more than one base table through a join view 'chinook.song_view'
```

Fine, so we will do this “table” at a time.

```
insert into song_view (genre) values ('Power Pop');
```

```
Error Code: 1423. Field of view 'chinook.song_view' underlying table doesn't have a default value
```

So the problem was the view did not contain the field GenreID (why would it?), and that field was not set to autoincrement. So the insert statement above would give us a null primary key. No worries, right? We will just make GenreID autoincrement, except MySQL balks again.

```
Error 1833: Cannot change column 'GenreId': used in a foreign key constraint 'FK_TrackGenreId' of table
'chinook.track' SQL Statement: ALTER TABLE `chinook`.`genre`  CHANGE COLUMN `GenreId` `GenreId` INT(11)
NOT NULL AUTO_INCREMENT
```

Since GenreID was in a foreign key relationship, MySQL wouldn’t allow it to be altered. So we need to drop the foreign key constraint, *then* apply the auto_increment value to GenreID, then recreate the foreign key constraint.

Finally, our insert statement works to add the ‘Power Pop’ genre to the view works! But the genre table has no foreign keys, so this is deceptively simple. Adding the album ‘Wheatus’ is going to require us to associate it to the artist ‘Wheatus’ and we are back with trying to modify more than one table with a join view.

We are still stuck with four insert statements to make the song ‘Teenage Dirtbag.’ **But because we did not include the primary keys and the foreign keys in the view, we will never be able to create records that need foreign keys.**

So we can replace the view definition as this:

```
create or replace view song_view as
select t.TrackId as track_id, t.Name as song_name, t.Composer as composer,
       t.Milliseconds as milliseconds, t.UnitPrice as unit_price,
       t.AlbumId as track_album_id, t.GenreId as track_genre_id,
       g.GenreId as genre_id, g.Name as genre,
       al.AlbumId as album_id, al.Title as album_title, al.ArtistId as album_artist_id,
       ar.ArtistId as artist_id, ar.Name as artist
from   chinook.track t
       join chinook.genre g using (genreid)
       join chinook.album al using (albumid)
       join chinook.artist ar using (artistid)
where  g.Name not in ('Opera', 'Easy Listening', 'Drama', 'Comedy', 'TV Shows');
```

Note that we now have all the primary keys and foreign keys as view fields. We can now insert this song using four insert statements

```
insert into song_view (genre) values ('Power Pop');

insert into song_view (artist) values ('Wheatus');

insert into song_view (album_title, album_artist_id) values ('Wheatus', ????);

insert into song_view (song_name, composer, milliseconds, unit_price, track_genre_id, track_album_id)
values ('Teenage Dirtbag', 'Brendan Brown', 247000, 0.99, ????, ????);
```

The first two are easy. But the insert into the album table requires the autogenerated primary key of the previous insert. And the final insert into the track table requires the previously generated genre_id and album_id. A database programmer could obtain those keys to fill in the question marks. Alternatively, the database programmer could just make all inserts run against the base tables (or better yet, base table views that exactly mimic the structure of the base tables.)

UPDATE statements

```
select track_id, song_name, artist, composer
from   song_view
where  artist = 'Queen' and composer like '%Mercury%';
```

Houston, we have a problem. There is no consistency in the composer field for Queen songs.

track_id	song_name	composer
425	It's A Hard Life	Freddie Mercury
2256	Killer Queen	Mercury, Freddie
2281	My Melancholy Blues	Mercury

Here an update statement works quite nicely.

```
update song_view
set    composer = 'Freddie Mercury'
where  composer in ('Mercury, Freddie', 'Mercury');
```

track_id	song_name	artist	composer
425	It's A Hard Life	Queen	Freddie Mercury
2256	Killer Queen	Queen	Freddie Mercury
2281	My Melancholy Blues	Queen	Freddie Mercury

But this update will not work for familiar reasons – we are trying to make one update statement update the genre table and the track table.

```
update song_view
set    composer = 'Black Sabbath',
       genre = 'Rock'
where  song_name = 'War Pigs' and artist = 'Cake';
```

Not all views can allow DML (update, insert, delete) transactions.

The view must have the same basic structure as the base tables. This mean that a view cannot use:

- functions or window functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list

There are other limitations which can be read in the MySQL manual.

DELETE statements

Join views do not allow deletion (this differs from INSERT and UPDATE).