Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

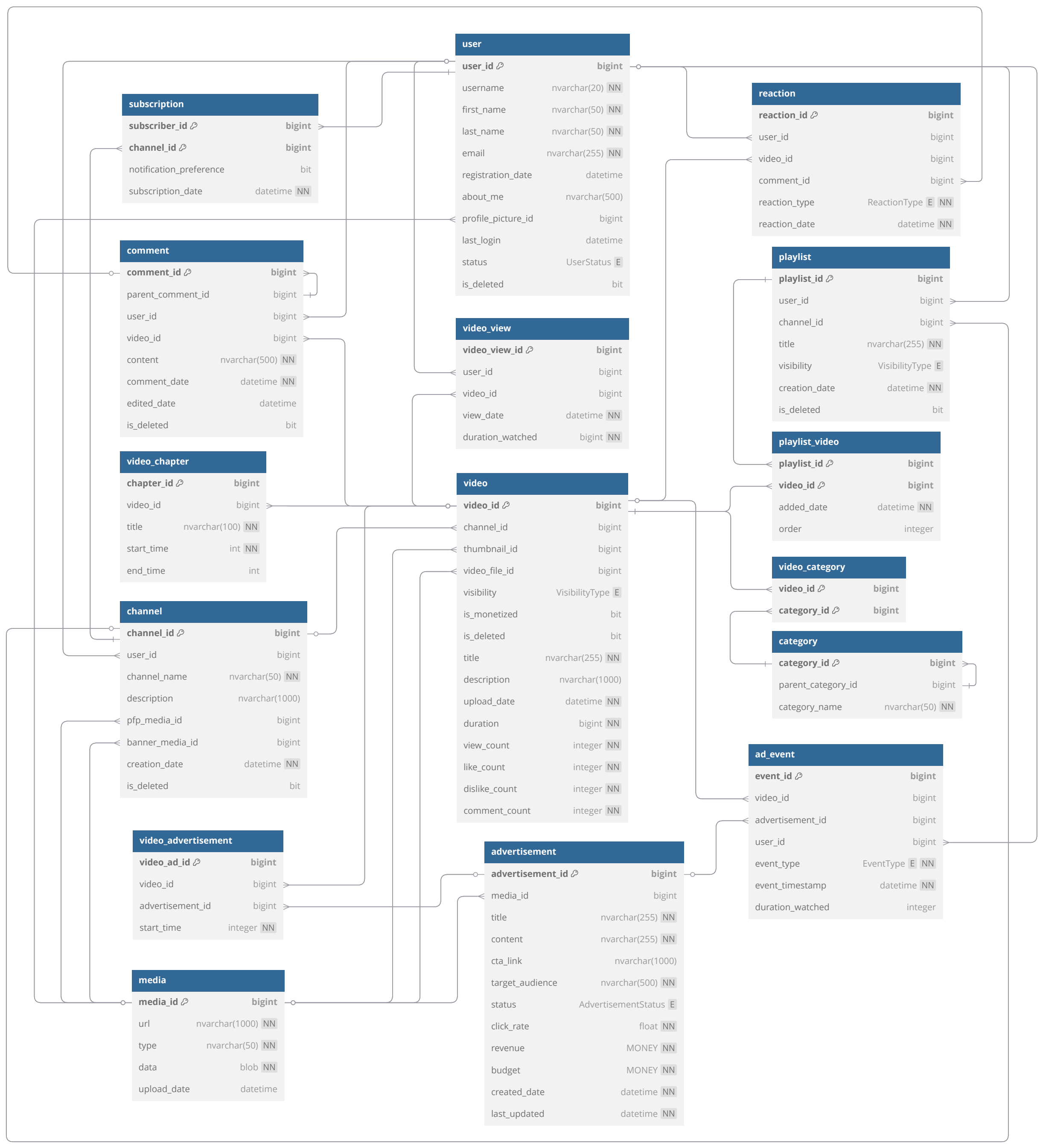
Databázové systémy 2

Projekt – YouTube-like platforma

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 24. 04. 2025

Relational Model



user		
user_id	**bigint**	
username	nvarchar(20)	NN
first_name	nvarchar(50)	NN
last_name	nvarchar(50)	NN
email	nvarchar(255)	NN
registration_date	datetime	
about_me	nvarchar(500)	
profile_picture_id	bigint	
last_login	datetime	
status	UserStatus	E
is_deleted	bit	

subscription		
subscriber_id	**bigint**	
channel_id	**bigint**	
notification_preference	bit	
subscription_date	datetime	NN

reaction		
reaction_id	**bigint**	
user_id	bigint	
video_id	bigint	
comment_id	bigint	
reaction_type	ReactionType	E NN
reaction_date	datetime	NN

comment		
comment_id	**bigint**	
parent_comment_id	bigint	
user_id	bigint	
video_id	bigint	
content	nvarchar(500)	NN
comment_date	datetime	NN
edited_date	datetime	
is_deleted	bit	

playlist		
playlist_id	**bigint**	
user_id	bigint	
channel_id	bigint	
title	nvarchar(255)	NN
visibility	VisibilityType	E
creation_date	datetime	NN
is_deleted	bit	

video_view		
video_view_id	**bigint**	
user_id	bigint	
video_id	bigint	
view_date	datetime	NN
duration_watched	bigint	NN

playlist_video		
playlist_id	**bigint**	
video_id	**bigint**	
added_date	datetime	NN
order	integer	

video_chapter		
chapter_id	**bigint**	
video_id	bigint	
title	nvarchar(100)	NN
start_time	int	NN
end_time	int	

video		
video_id	**bigint**	
channel_id	bigint	
thumbnail_id	bigint	
video_file_id	bigint	
visibility	VisibilityType	E
is_monetized	bit	
is_deleted	bit	
title	nvarchar(255)	NN
description	nvarchar(1000)	
upload_date	datetime	NN
duration	bigint	NN
view_count	integer	NN
like_count	integer	NN
dislike_count	integer	NN
comment_count	integer	NN

video_category		
video_id	**bigint**	
category_id	**bigint**	

category		
category_id	**bigint**	
parent_category_id	bigint	
category_name	nvarchar(50)	NN

channel		
channel_id	**bigint**	
user_id	bigint	
channel_name	nvarchar(50)	NN
description	nvarchar(1000)	
pfp_media_id	bigint	
banner_media_id	bigint	
creation_date	datetime	NN
is_deleted	bit	

ad_event		
event_id	**bigint**	
video_id	bigint	
advertisement_id	bigint	
user_id	bigint	
event_type	EventType	E NN
event_timestamp	datetime	NN
duration_watched	integer	

video_advertisement		
video_ad_id	**bigint**	
video_id	bigint	
advertisement_id	bigint	
start_time	integer	NN

advertisement		
advertisement_id	**bigint**	
media_id	bigint	
title	nvarchar(255)	NN
content	nvarchar(255)	NN
cta_link	nvarchar(1000)	
target_audience	nvarchar(500)	NN
status	AdvertisementStatus	E
click_rate	float	NN
revenue	MONEY	NN
budget	MONEY	NN
created_date	datetime	NN
last_updated	datetime	NN

media		
media_id	**bigint**	
url	nvarchar(1000)	NN
type	nvarchar(50)	NN
data	blob	NN
upload_date	datetime	

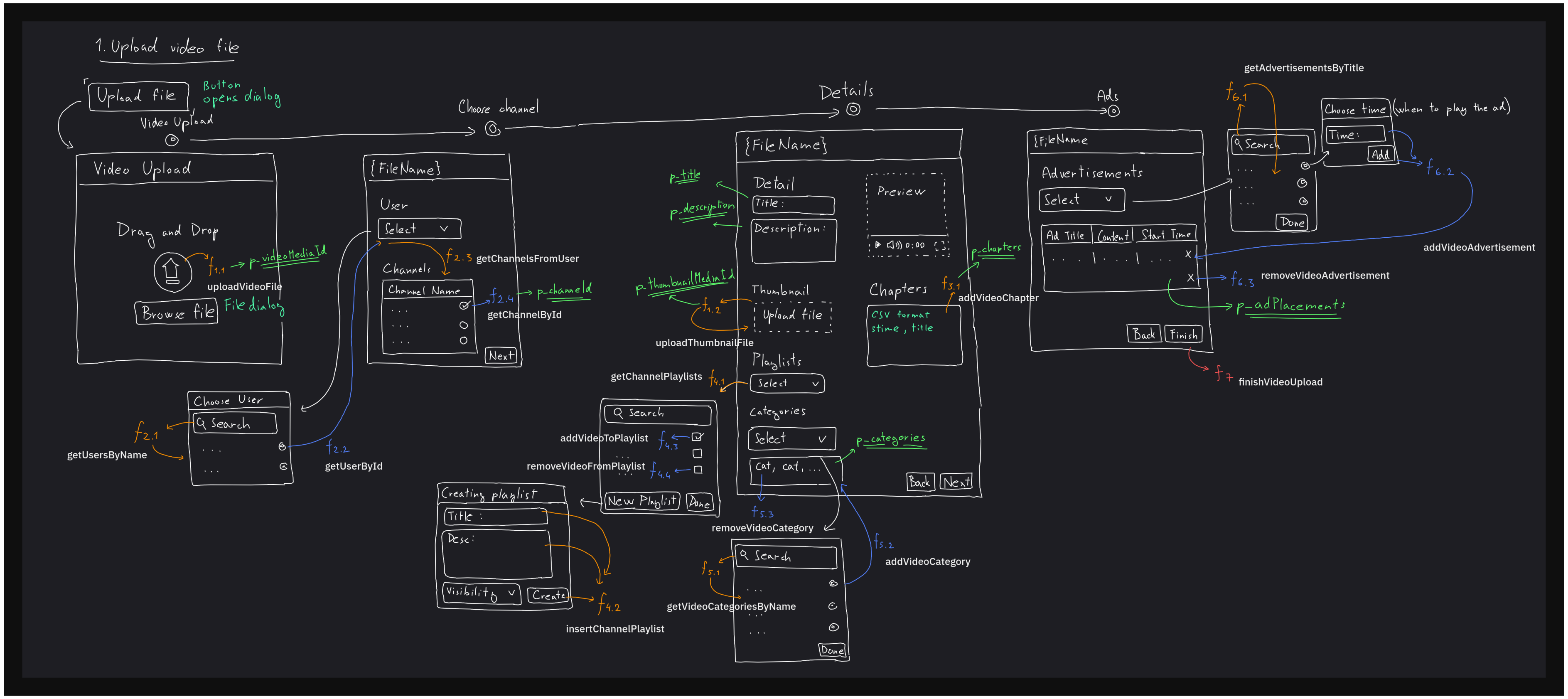
Form Design

Database functions (functions touching the database)

Application functions

Parameters for final transaction function

Transaction function



Functions

```
///////////////////////////////////
/// Functions ////////////////////
///////////////////////////////////

// 9 DB functions (touching the database) + 9 APP functions (purely in application)

// DB
func uploadVideoFile            (filepath string)               Media
// DB
func uploadThumbnailFile        (filepath string)               Media
// DB
func getUsersByName             (searchUsername string)         []User
// APP
func getUserById                (userId int)                    User
// DB
func getChannelsFromUser        (userId int)                    []Channel
// APP
func getChannelById             (channelId int)                 Channel
// APP
func addVideoChapter            (videoId int, title string,
                                 startTime int, endTime int?)   void
// DB
func getChannelPlaylists        (channelId int)                 []Playlist
// DB
func insertChannelPlaylist      (userId int, channelId int,
                                 title string,
                                 visibility VisibilityEnum)     Playlist
// APP
func addVideoToPlaylist         (playlistId int)                void
// APP
func removeVideoFromPlaylist    (playlistId int)                void
// DB
func getVideoCategoriesByName   (categoryName string)           []Category
// APP
func addVideoCategory           (category string)               void
// APP
func removeVideoCategory        (category string)               void
// DB
func getAdvertisementsByTitle   (searchAdTitle string)          []Advertisement
// APP
func addVideoAdvertisement      (advertisementId int,
                                 startTime int)                 void
// APP
func removeVideoAdvertisement   (videoAdvertisementId int)      void
// DB
func finishVideoUpload(
    channelId           int,                    // Channel ID where the video is uploaded
    title               string,                 // Title of the video
    description         string,                 // Video description (optional)
    visibility          VisibilityEnum,         // (public, private, unlisted, draft)
    isMonetized         bool,                   // Monetization flag (1 or 0)
    thumbnailMediaId    int,                    // Media ID of uploaded thumbnail
    videoMediaId        int,                    // Media ID of uploaded video file
    duration            int,                    // Duration of video (seconds)
    playlistIds         []int,                  // IDs of playlists to add video
    categoryIds         []int,                  // IDs of categories
    adPlacements        []AdPlacement,          // Ads placed in the video (if monetized)
    chapters            []Chapter,              // Video chapters
) Video                                         // Newly inserted video
```

```
////////////////////////////////////////////////////////////////////
// 1. upload thumbnail and video files /////////////////////////////
////////////////////////////////////////////////////////////////////


// 1.1 DB -> p_videoMediaId
func uploadVideoFile(filepath string) Media {
    var url string := createUrl(filepath)
    var blob Blob := getBlobData(filepath)

    var media Media := INSERT INTO media(url, type, data)
                       VALUES (url, 'video', blob)
    return media
}


// 1.2 DB -> p_thumbnailMediaId
func uploadThumbnailFile(filepath string) Media {
    var url string := createUrl(filepath)
    var blob Blob := getBlobData(filepath)

    var media Media := INSERT INTO media(url, type, data)
                       VALUES (url, 'thumbnail', blob)
    return media
}




////////////////////////////////////////////////////////////////////
// 2. identify channel for which to upload the video ///////////////
////////////////////////////////////////////////////////////////////


// 2.1 DB (active and not-deleted)
func getUsersByName(searchUsername string) []User {
    var users []User := SELECT *
                          FROM user
                         WHERE username LIKE '%searchUsername%'
                           AND status == 'active'
                           AND is_deleted == 0
    return users
}


// 2.2 APP
func getUserById(userId int) User


// 2.3 DB
func getChannelsFromUser(userId int) []Channel {
    var channels []Channel := SELECT *
                                FROM channel
                               WHERE user_id == userId
                                 AND is_deleted == 0
    return channels
}


// 2.4 APP -> p_channelId
func getChannelById(channelId int) Channel
```

```
//////////////////////////////////////////////////////////////////////
// 3. add chapters to the video /////////////////////////////////////
//////////////////////////////////////////////////////////////////////


// 3.1 APP -> +p_chapters
func addVideoChapter(videoId int, title string, startTime int, endTime int?)



//////////////////////////////////////////////////////////////////////
// 4. add this video to channel playlists //////////////////////////
//////////////////////////////////////////////////////////////////////


// 4.1 DB (not-deleted)
func getChannelPlaylists(channelId int) []Playlist {
    var playlists []Playlist := SELECT *
                                  FROM playlist
                                 WHERE channel_id = channelId
                                   AND is_deleted = 0
    return playlists
}


// 4.2 DB
func insertChannelPlaylist(
    userId int, channelId int, title string, visibility VisibilityEnum
) Playlist {
    var playlist Playlist := INSERT INTO playlist(
                                user_id, channel_id, title, visibility
                             ) VALUES (userId, channelId, title, visibility)
    return playlist
}


// 4.3 APP -> +p_playlistsIds
func addVideoToPlaylist(playlistId int)


// 4.4 APP -> -p_playlistsIds
func removeVideoFromPlaylist(playlistId int)


//////////////////////////////////////////////////////////////////////
// 5. assing the video to categories ////////////////////////////////
//////////////////////////////////////////////////////////////////////


// 5.1 DB
func getVideoCategoriesByName(categoryName string) []Category {
    var categories []Category := SELECT *
                                   FROM category
                                  WHERE category_name LIKE '%categoryName%'
    return categories
}


// 5.2 APP -> +p_categories
func addVideoCategory(category string)

// 5.3 APP -> -p_categories
func removeVideoCategory(category string)
```

```
////////////////////////////////////////////////////////////////////
// 6. add advertisements to the video //////////////////////////////
////////////////////////////////////////////////////////////////////


// 6.1 DB (where status == 'active')
func getAdvertisementsByTitle(searchAdTitle string) []Advertisement {
    var ads []Advertisement := SELECT *
                                 FROM advertisement
                                WHERE title LIKE '%searchAdTitle%'
                                  AND status = 'active'
    return ads:
}


// 6.2 APP -> +p_adPlacements
func addVideoAdvertisement(advertisementId int, startTime int)


// 6.3 APP -> -p_adPlacements
func removeVideoAdvertisement(videoAdvertisementId int)




////////////////////////////////////////////////////////////////////
// 7. finalize by calling a transaction ////////////////////////////
////////////////////////////////////////////////////////////////////


// DB Transaction
func finishVideoUpload(
    channelId           int,                // Channel ID where the video is uploaded
    title               string,             // Title of the video
    description         string,             // Video description (optional)
    visibility          VisibilityEnum,     // (public, private, unlisted, draft)
    isMonetized         boolean,            // Monetization flag (1 or 0)
    thumbnailMediaId    int,                // Media ID of uploaded thumbnail
    videoMediaId        int,                // Media ID of uploaded video file
    duration            int,                // Duration of video (seconds)
    playlistIds         []int,              // IDs of playlists to add video
    categoryIds         []int,              // IDs of categories
    adPlacements        []AdPlacement,      // Ads placed in the video (if monetized)
    chapters            []Chapter,          // Video chapters
) Video                                     // Newly inserted video



.
```

Transaction

```
//////////////////////////////////////////////////
// TRANSACTION MINISPECIFICATION ////////////////
//////////////////////////////////////////////////

type AdPlacement struct {
    adId       int
    startTime  int
}

type Chapter struct {
    title       string
    startTime   int
    endTime?    int
}

type VisibilityEnum enum {
    PUBLIC
    PRIVATE
    UNLISTED
    DRAFT
}

func finalizeVideoUpload(
    p_channelId             int,                // Channel ID where video is uploaded
    p_title                 string,             // Title of the video
    p_description           string,             // Video description (optional)
    p_visibility            VisibilityEnum,     // (public, private, unlisted, draft)
    p_isMonetized           bool,               // Monetization flag (1 or 0)
    p_thumbnailMediaId      int,                // Media ID of uploaded thumbnail
    p_videoMediaId          int,                // Media ID of uploaded video file
    p_duration              int,                // Duration of video (seconds)
    p_playlistIds           []int,              // IDs of playlists to add video
    p_categoryIds           []int,              // IDs of categories
    p_adPlacements          []AdPlacement,      // Ads placed in the video
    p_chapters              []Chapter,          // Video chapters
) Video {                                       // Newly inserted video
    beginTransaction()

    // CHANNEL VALIDATION: Verify that the provided channel exists.
    if empty(SELECT * FROM channel WHERE channel_id = p_channelId) {
        rollback()
        raise Error("Channel with id {p_channelId} doesnt exist")
    }


    // ADS VALIDATION: All ads should exist and be active.
    for advertisement in p_advertisements {
        var ads []Advertisement = SELECT * FROM advertisement
                                  WHERE advertisement_id = advertisement.adId
        if empty(ads) {
            rollback()
            raise Error("Ad with id of {advertisement.adId} doesn't exist.")
        }

        var ad = single(ads)
        if ad.status != AdvertisementStatus.ACTIVE {
            rollback()
            raise Error("Ad with if of {advertisement.adId} isn't active.")
        }
    }
    // MEDIA VALIDATION:
    // Verify that the thumbnail and video file exist and are of correct media type.
    // (for video media.type = 'video' and for thumbnail media.type = 'thumbnail')

    // If the query result is empty then no thumbnail with provided id exists.
    if empty(SELECT * FROM media WHERE media_id = p_thumbnailMediaId) {
        rollback()
        raise Error("Thumbnail media doesn't exist.")
    }

    // Get the thumbnail media by id
    // and check that thumbnail media has its type set to 'thumbnail'.
    var thumbnailMedia media := single(SELECT * FROM media
                                       WHERE media_id = p_thumbnailMediaId)
    if thumbnailMedia.type != 'thumbnail' {
        rollback()
        raise Error("For thumbnail media expected type 'thumbnail'.")
    }


    // If the query result is empty then no video media with provided id exists.
    if empty(SELECT * FROM media WHERE media_id = p_videoMediaId) {
        rollback()
        raise Error("Video media doesn't exist.")
    }

    // Get the video media by id and check that its type is set to 'video'.
    var videoMedia media := single(SELECT * FROM media WHERE media_id = p_videoMediaId)
    if videoMedia.type != 'video' {
        rollback()
        raise Error("For video media expected type 'video'.")
    }


    // PLAYLIST VALIDATION:
    // Validate that each playlist is owned by the selected channel.
    var channelPlaylists []int = SELECT playlist_id FROM playlist
                                 WHERE channel_id = p_channelId
    for playlistId in p_playlistIds {
        if !channelPlaylists.contains(playlistId) {
            rollback()
            raise Error("Playlist {playlistId} doesn't belong \
                         to channel with id {p_channelId}")
        }
    }


    // CHAPTER VALIDATION:
    // Ensure chapters are in sequential order and non-overlapping.
    for chapter in chapters.sort(start_time, Order.ASCENDING) {

        // If the end_time is defined, then it must come after start_time.
        if chapter.end_time != null && chapter.end_time <= chapter.start_time {
            rollback()
            raise Error("Chapter end time must be greater than start time")
        }

        // The current chater's end_time must be
        // less then the start_time of the next chapter.
        if currentChapter.end_time > nextChapter.start_time {
            rollback()
            raise Error("Chapters cannot overlap")
        }
    }
    // Insert the video record
    var newVideo video := INSERT INTO video (
                            channel_id,  thumbnail_id,  video_file_id,
                            visibility,  is_monetized,  is_deleted,
                            title,       description,   upload_date,    duration,
                            view_count,  like_count,    dislike_count,  comment_count
                        ) VALUES (
                            p_channelId,  p_thumbnailMediaId, p_videoMediaId,
                            p_visibility, p_isMonetized,      false,
                            p_title,      p_description,      GETDATE(), p_duration,
                            0,            0,                  0,         0
                        )


    // Insert each selected category into video_category table.
    for categoryId in p_category_ids {
        INSERT INTO video_category (video_id, category_id)
        VALUES (newVideo.video_id, categoryId)
    }


    // Insert each selected playlist into playlist_video table.
    for playlistId in p_playlist_ids {
        // Determine the maximum order currently in the playlist.
        var nextOrder int := SELECT COALESCE(MAX(order), 0) + 1
                             FROM playlist_video
                            WHERE playlist_id = playlistId

        INSERT INTO playlist_video(playlist_id, video_id, added_date, order)
        VALUES (p_playlistId, newVideo.video_id, GETDATE(), nextOrder)
    }


    // Ads only allowed if video is monetized, skip if monetization is disabled.
    // If video is monetized, insert advertisements into video_advertisement join table.
    if p_isMonetized == true {
        for advertisement in p_adPlacements {
            INSERT INTO video_advertisement (video_id, advertisement_id, start_time)
            VALUES (newVideo.video_id, advertisement.adId, advertisement.startTime)
        }
    }


    // Insert chapters into video_chapter table.
    for chapter in p_chapters {
        INSERT INTO video_chapter (video_id, title, start_time, end_time)
        VALUES (
            newVideo.video_id, p_chapterTitle,
            chapter.startTime, chapter.endTime
        )
    }


    // End Transaction.
    endTransaction()


    // Return id of the newly inserted video.
    return newVideo.video_id
}


.
```