Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Technologie databázových systémů 1

Semestrální projekt

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 11-05-2025

Obsah

1. Database Design 3
   1.1. DD S01 L02 3
   1.2. DD S02 L02 3
   1.3. DD S03 L01 3
   1.4. DD S03 L02 5
   1.5. DD S30 L04 6
   1.6. DD S04 L01 7
   1.7. DD S04 L02 7
   1.8. DD S05 L01 7
   1.9. DD S05 L03 7
   1.10. DD S06 L01 7
   1.11. DD S06 L02-04 8
   1.12. DD S07 L01 8
   1.13. DD S07 L02 8
   1.14. DD S07 L03 8
   1.15. DD S09 L01 8
   1.16. DD S09 L02 9
   1.17. DD S10 L01 9
   1.18. DD S10 L02 9
   1.19. DD S11 L01 10
   1.20. DD S11 L02-04 10
   1.21. DD S15 L01 11
   1.22. DD S16 L02 11
   1.23. DD S16 L03 12
   1.24. DD S17 L01 12
   1.25. DD S17 L02 12
   1.26. DD S17 L03 12

2. Programming with SQL 13
   2.1. SQL S01 L01 13
   2.2. SQL S01 L02 13

2.3. SQL S01 L03 14
2.4. SQL S02 L01 14
2.5. SQL S02 L02 14
2.6. SQL S02 L03 14
2.7. SQL S03 L01 15
2.8. SQL S03 L02 15
2.9. SQL S03 L03 15
2.10. SQL S03 L04 15
2.11. SQL S04 L02 16
2.12. SQL S04 L03 16
2.13. SQL S05 L01 16
2.14. SQL S05 L02 16
2.15. SQL S05 L03 17
2.16. SQL S06 L01 17
2.17. SQL S06 L02 18
2.18. SQL S06 L03 18
2.19. SQL S06 L04 18
2.20. SQL S07 L01 18
2.21. SQL S07 L02 19
2.22. SQL S07 L03 19
2.23. SQL S08 L01 19
2.24. SQL S08 L02 20
2.25. SQL S08 L03 20
2.26. SQL S10 L01 21
2.27. SQL S10 L02 21
2.28. SQL S10 L03 21
2.29. SQL S11 L01 22
2.30. SQL S11 L03 22
2.31. SQL S12 L01 22
2.32. SQL S12 L02 23
2.33. SQL S13 L01 23
2.34. SQL S13 L03 23
2.35. SQL S14 L01 23
2.36. SQL S15 L01 24
2.37. SQL S16 L03 24

Database Design

1.1. DD S01 L02

Explain the difference between the concept of data and information.

Data are raw facts or figures without context. For example, a number like 512, a string like ‘hello world’, or a timestamp like ‘2025-05-10 19:32:51’ are all data. Information is data that has been processed or structured in a way that adds meaning. For instance, interpreting 512 as the duration (in seconds) of a video uploaded on ‘2025-05-10 19:32:51’ transforms raw data into actionable information.

1.2. DD S02 L02

Entities, instances, attributes and identifiers - describe in examples on your project.

- Entity: z_user (represents a user)
- Instance: A specific record in the table. Example: A user with user_id = 1234, username = ‘jdoe’, email = ‘john.doe@example.com’, and status = ‘active’.
- Attributes: Properties of an entity. For z_user, attributes include username, email, first_name, last_name, profile_picture_id, etc.
- Identifiers: Unique attributes or combinations that distinguish one instance from another. Example: user_id is the primary key of z_user, uniquely identifying each user.

1.3. DD S03 L01

- *Describe all relations in your database in English, including cardinality and membership obligation - each relation in two sentences.*

Name	Source	Cardinality Optionality	Target
Category_Has_SubCategories	z_category	0..* : 0..1	z_category
Channel_Has_BannerImage	z_media_image	0..* : 0..1	z_channel
Channel_Has_Playlists	z_channel	0..* : 0..1	z_playlist
Channel_Has_ProfileImage	z_image_media	0..* : 0..1	z_channel
Channel_Has_Subscriptions	z_channel	0..* : 1..1	z_subscription
Channel_Has_Videos	z_channel	0..* : 1..1	z_video
Comment_Has_Reaction	z_comment	0..* : 1..1	z_reaction
Comment_Has_SubComment	z_comment	0..* : 0..1	z_comment
Playlist_Has_Videos	z_playlist	0..* : 1..1	z_playlist_video
User_Has_Channels	z_user	0..* : 1..1	z_channel
User_Has_Comments	z_user	0..* : 1..1	z_comment
User_Has_Playlists	z_user	0..* : 0..1	z_playlist
User_Has_ProfileImage	z_image_media	0..* : 0..1	z_user
User_Has_Reactions	z_user	0..* : 1..1	z_reaction
User_Has_Subscriptions	z_user	0..* : 1..1	z_subscription
User_Has_VideoViews	z_user	0..* : 1..1	z_video_view
Video_Has_Comments	z_video	0..* : 1..1	z_comment
Video_Has_Reactions	z_video	0..* : 1..1	z_reaction
Video_Has_ThumbnailImage	z_image_media	0..1 : 1..1	z_video
Video_Has_VideoFile	z_video_media	0..1 : 1..1	z_video
Video_Has_VideoViews	z_video	0..* : 1..1	z_video_view
Video_IsIn_Playlists	z_video	0..* : 1..1	z_playlist_video
z_video_category	z_video	0..* : 0..*	z_category

Table 1: Relations with cardinality and optionality

1.4. DD S03 L02

Draw an ER diagram according to conventions.

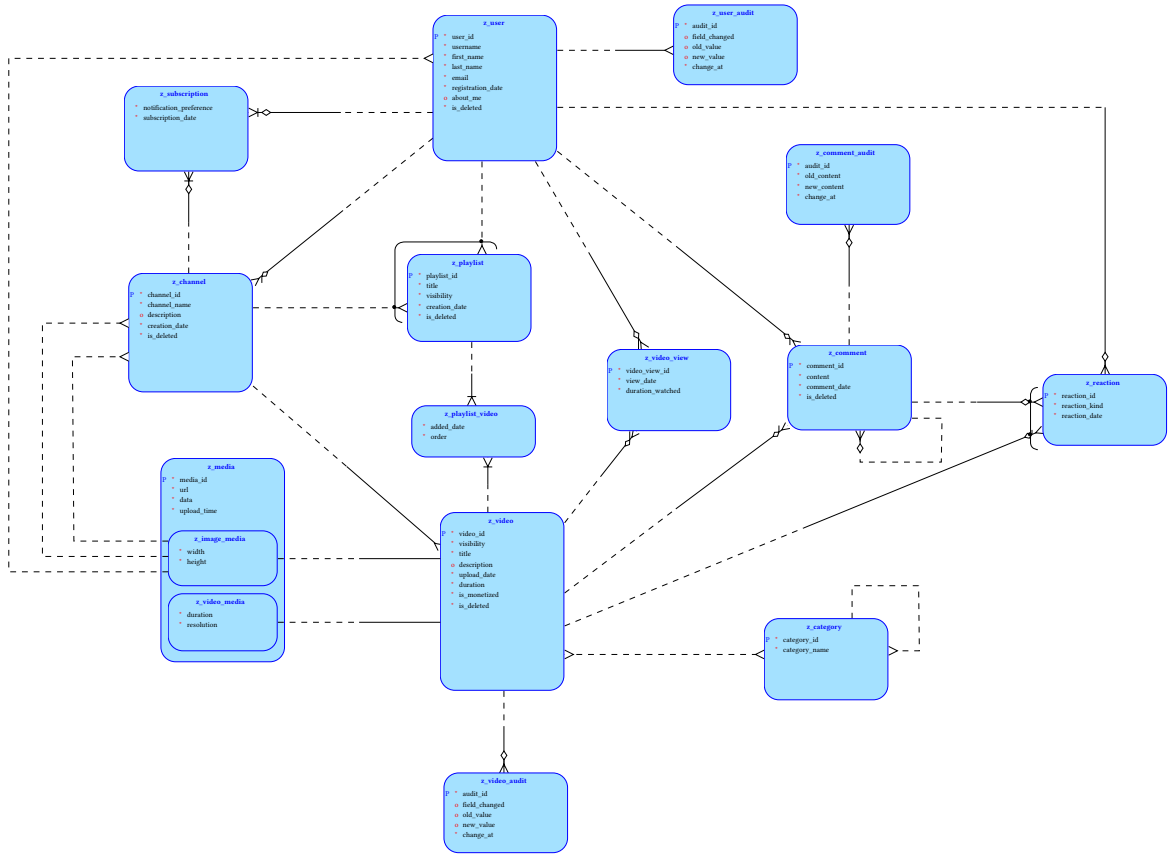

Figure 1: ER Diagram

1.5. DD S30 L04

Matrix diagram with relationships, draw for your solution.

	category	channel	comment	image_media	media	playlist	playlist_video	reaction	subscription	user	video	video_media	video_view
category	has subcategories										categorizes videos		
channel		–		has banner	references media	creates playlists		receives reactions	has subscribers	created by user	has videos		
comment			–					receives reactions		written by user	on video		
image_media				–	references media					used in user/channel/video			
media					–								
playlist						–	contains videos			created by user/channel			
playlist_video							–				references video		
reaction			on comment					–		by user	on video		
subscription									–	by user	to channel		
user		creates channels	writes comments	has profile picture		creates playlists		creates reactions	makes subscriptions	–			makes views
video	is categorized	belongs to channel	has comments	has thumbnail image			is in playlists	receives reactions		uploaded by user	–	has video file	has views
video_media					references media							–	
video_view											on video		–

Table 2: Relationship Matrix

1.6. DD S04 L01

Supertypes and subtypes – define at least one instance of a supertype and a subtype in your project.

The entity z_media is a generalization for all media. z_image_media and z_video_media are specializations that inherit media_id from z_media but introduce additional attributes:

- z_image_media: width, height, format
- z_video_media: duration, resolution, codec This models shared vs. specific media characteristics.

1.7. DD S04 L02

Description of business rules for your project.

- **Channel Ownership:** A channel cannot be reassigned to another user post-creation (enforced by a BEFORE UPDATE trigger).
- **Reaction Targeting:** A reaction must target either a video or a comment, but never both (CHECK constraint).
- **Playlist Creator Arc:** Only one of user_id or channel_id can be set for a playlist (enforced by CHECK).
- **Soft Deletion:** Deletions are logical via is_deleted, not physical.
- **Non-Transferability:** Certain foreign keys are immutable to preserve referential integrity (enforced with triggers).
- **Data Checks** - Start time must start before end time. Video cannot have negative views.

1.8. DD S05 L01

Include at least one transferable and one non-transferable binding in your project.

- **Transferable Binding:** z_video.thumbnail_media_id → z_image_media.media_id is portable and supports reuse of shared images.
- **Non-Transferable Binding:** z_comment.user_id is locked after creation via a BEFORE UPDATE trigger — representing a non-transferable, immutable relationship.

1.9. DD S05 L03

Have at least one M:N relationship without information and one M:N relationship with inf-rmation in your project.

- **Without Information:** z_video_category links videos to categories. No extra attributes.
- **With Information:** z_playlist_video includes added_date and order, which add context beyond the M:N relation.

1.10. DD S06 L01

Incorporate at least one 1:N identifying relationship into your project, with the fact that the transferred foreign key will also be the key in the new table (identifying relationship).

z_video_media.media_id is also the PK and a FK to z_media.media_id. This is a strong identifying relationship: each video media must be a media.

1.11. DD S06 L02-04

Have your schema in first normal form - no non-atomic attributes. In second normal form – no subkey bindings. In third normal form - no links between secondary attributes.

Yes. Schema is in 1NF, 2NF and 3NF.

1.12. DD S07 L01

Try to define ARC in your project (can be defined in ORACLE SQL Developer Data M-deler).

z_playlist uses an exclusive arc between user_id and channel_id. Only one of them may be non-null to indicate ownership. This is enforced via a CHECK constraint in DDL and supported in the modeler using arcs.

1.13. DD S07 L02

Try to define hierarchical and recursive relations in your project.

z_category is hierarchical via parent_category_id (categories can nest). z_comment is recursive: replies point to another comment_id (enabling threaded discussions).

1.14. DD S07 L03

Describe how you record historical data in your system.

The system uses shadow/audit tables to preserve historical data. For example, every update to the z_comment.content or z_user is tracked in separate audit tables that store the previous and new value along with a timestamp and a reference to the original entity.

```
CREATE TABLE z_comment_audit (
  audit_id INTEGER PRIMARY KEY,
  comment_id INTEGER NOT NULL,
  old_content VARCHAR2(500),
  new_content VARCHAR2(500),
  change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_comment_audit FOREIGN KEY (comment_id) REFERENCES z_comment(comment_id)
);

CREATE TABLE z_user_audit (
  audit_id INTEGER PRIMARY KEY,
  user_id INTEGER NOT NULL,
  field_changed VARCHAR2(50),
  old_value VARCHAR2(255),
  new_value VARCHAR2(255),
  change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_user_audit FOREIGN KEY (user_id) REFERENCES z_user(user_id)
);
```

1.15. DD S09 L01

Demonstrate saving changes over time on your project.

When a tracked column such as z_user.email or z_comment.content is modified, the system logs the previous and new values. This allows administrators or analysts to review the evolution of content or user states over time.

1.16. DD S09 L02

Try journaling in your project, i.e. saving past historical data (for example salary changes, workplace changes, etc.).

For user changes:

```
CREATE OR REPLACE TRIGGER trg_user_change
  BEFORE UPDATE OF status ON z_user
  FOR EACH ROW
BEGIN
  IF :OLD.email =/= :NEW.email THEN
    INSERT INTO z_user_audit(user_id, field_changed, old_value, new_value, change_date)
    VALUES (:OLD.user_id, 'email', :OLD.email, :NEW.email, CURRENT_TIMESTAMP);
  END IF;

  IF :OLD.first_name =/= :NEW.first_name THEN
    INSERT INTO z_user_audit(user_id, field_changed, old_value, new_value, change_date)
    VALUES (:OLD.user_id, 'first_name', :OLD.first_name, :NEW.first_name,
CURRENT_TIMESTAMP);
  END IF;

  ... -- other attributes
END;
```

For comment content edits:

```
CREATE OR REPLACE TRIGGER trg_comment_content_change
  BEFORE UPDATE OF content ON z_comment
  FOR EACH ROW
BEGIN
  IF :OLD.content =/= :NEW.content THEN
    INSERT INTO z_comment_audit(comment_id, old_content, new_content, change_date)
    VALUES (:OLD.comment_id, :OLD.content, :NEW.content, CURRENT_TIMESTAMP);
  END IF;
END;
```

1.17. DD S10 L01

Revise your design according to conventions for the readability of your schema.

ok.

1.18. DD S10 L02

Generic modeling – consider, possibly describe or use a generic model of data structures in your solution, how this approach is more advantageous compared to traditional data structure design methods.

Generic modeling introduces abstraction by designing reusable, flexible structures instead of tightly coupled entity-specific tables. This approach is particularly advantageous when dealing with heterogeneous media types and polymorphic associations.

In the project:

- z_media is a supertype representing a general media file.
- Subtypes like z_image_media and z_video_media specialize it by adding specific attributes (e.g., resolution, codec, width, height).

- This allows any table to reference z_media without needing to know whether it's a video or image and thus avoiding duplication and simplifying integration.

Advantages:
- **Scalability:** New media types (e.g., audio or documents) can be added with minimal schema changes.
- **Consistency:** All media share a common identifier and storage structure.
- **Simplified Constraints:** Referential integrity and file management logic are centralized (in one table).

1.19. DD S11 L01

Describe examples of integrity constraints on your project for entities, bindings, attributes, and user-defined integrity.

Integrity constraints in the project:

Entity constraints:
- z_user.user_id, z_video.video_id, etc. are primary keys → unique and not null.
- z_video.visibility has a CHECK constraint: must be 'draft', 'public', 'unlisted', or 'private'.

Binding (relationship) constraints:
- z_comment.video_id must exist in z_video (FOREIGN KEY).
- z_subscription has a composite PK on (user_id, channel_id) to prevent duplicates.

Attribute constraints:
- z_user.email must be UNIQUE (enforced via alternate key in practice).
- z_video.duration must be a positive integer.
- z_reaction.reaction_kind must be 'like' or 'dislike'.

User-defined constraints:
- ARC logic in z_reaction: either video_id or comment_id must be present, but not both (enforced via CHECK and TRIGGER).
- In z_playlist, only one of user_id or channel_id can be non-null (CHECK constraint PlaylistCreator).

1.20. DD S11 L02-04

Generate a relational schema from your conceptual model and note the changes that have occurred in the schema and why.

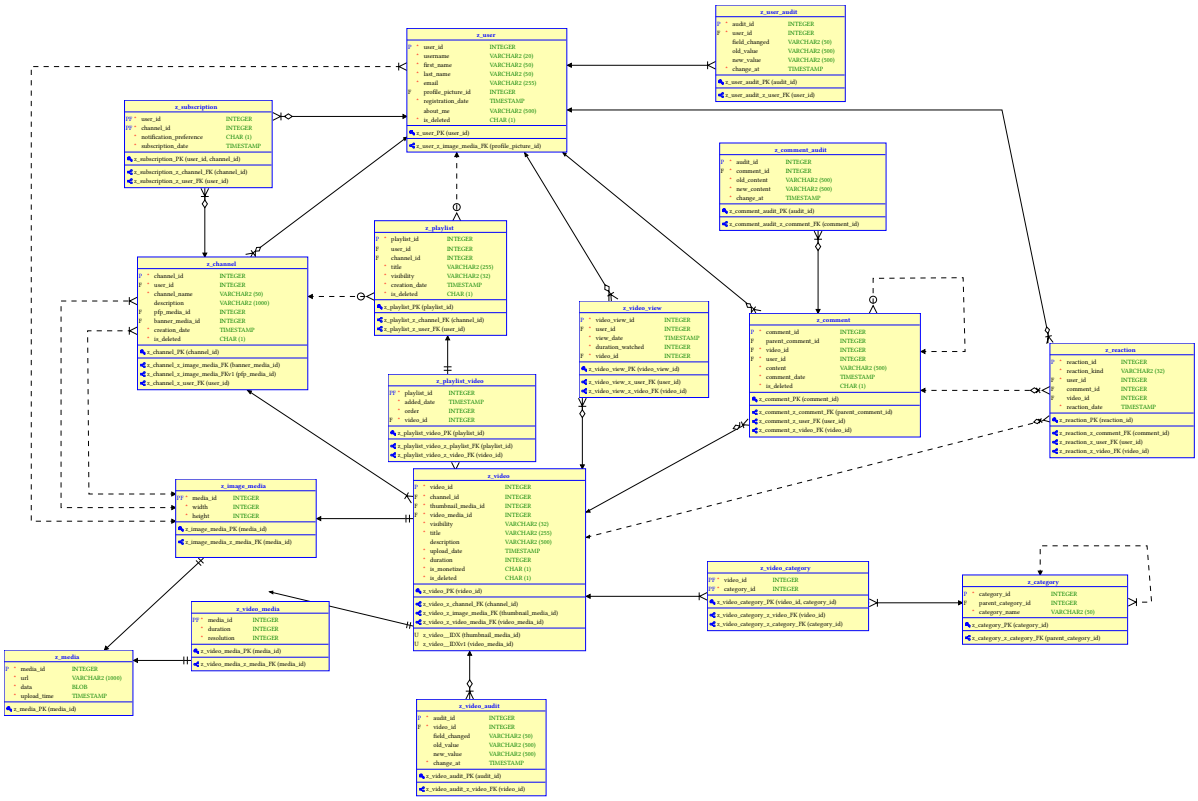

Figure 2: ER Diagram

Key changes from conceptual model:

- Relations are modeled by adding foreign keys to tables.
- Many to many relations are modeled by adding a associative/junction table.
- Subtype implementation in SQL using shared key rather than classical inheritance
- ARC constraints require TRIGGERS due to SQL limitations on exclusive foreign key references

1.21. DD S15 L01

Write query for concatenate strings by pipes || , and CONCAT(). Write query with SELECT DISTINCT.

```sql
SELECT DISTINCT first_name || ' ' || last_name AS full_name FROM z_user;

SELECT DISTINCT CONCAT(first_name, CONCAT(' ', last_name)) AS full_name FROM z_user;
```

1.22. DD S16 L02

Write query with WHERE condition for selecting rows and use functions LOWER, UPPER, INITCAP.

```sql
WHERE clause with string functions:

SELECT * FROM z_user
WHERE LOWER(username) = 'admin';

SELECT * FROM z_user
WHERE UPPER(status) = 'ACTIVE';
```

```sql
SELECT * FROM z_user
WHERE INITCAP(first_name) = 'John';
```

1.23. DD S16 L03

Write query with BETWEEN ... AND, LIKE (%, _), IN(), IS NULL and IS NOT NULL.

```sql
SELECT * FROM z_video
WHERE duration BETWEEN 60 AND 300;

SELECT * FROM z_user
WHERE email LIKE '%.cz';

SELECT * FROM z_user
WHERE username LIKE '_ohn';

SELECT * FROM z_user
WHERE status IN ('active', 'suspended');

SELECT * FROM z_user
WHERE profile_picture_id IS NULL;

SELECT * FROM z_user
WHERE profile_picture_id IS NOT NULL;
```

1.24. DD S17 L01

Write query with AND, OR, NOT and precendence.

```sql
SELECT * FROM z_user
WHERE (status = 'active' AND is_deleted = '0')
   OR (status = 'suspended' AND NOT is_deleted = '1');
```

1.25. DD S17 L02

Write query with ORDER BY atr [ASC/DESC] - Sorting by using one or more attributes.

```sql
SELECT * FROM z_video
ORDER BY upload_date DESC;

SELECT * FROM z_video
ORDER BY visibility ASC, like_count DESC;
```

1.26. DD S17 L03

Write query with single row functions and column functions MIN, MAX, AVG, SUM, COUNT.

```sql
SELECT COUNT(*) AS total_users FROM z_user;

SELECT MIN(duration) AS shortest_video, MAX(duration) AS longest_video FROM z_video;

SELECT AVG(duration) AS average_duration FROM z_video;

SELECT SUM(view_count) AS total_views FROM z_video;
```

Programming with SQL

2.1. SQL S01 L01

- LOWER, UPPER, INITCAP
- CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, TRIM, REPLACE
- Use virtual table DUAL

```sql
-- LOWER, UPPER, INITCAP
SELECT
  LOWER('HeLLo WorlD') AS lower,
  UPPER('HeLLo WorlD') AS upper,
  INITCAP('heLLo worLD') AS initcap
FROM dual;

-- CONCAT, SUBSTR, LENGTH, INSTR
SELECT
  CONCAT('Hello', 'World') AS concatenated,
  SUBSTR('abcdef', 2, 3) AS substr_example, -- 'bcd'
  LENGTH('Oracle') AS len,
  INSTR('banana', 'a') AS first_a_position
FROM dual;

-- LPAD, RPAD
SELECT
  LPAD('42', 5, '0') AS lpad_example,  -- '00042'
  RPAD('ok', 5, '.') AS rpad_example   -- 'ok...'
FROM dual;

-- TRIM, REPLACE
SELECT
  TRIM(' x ') AS trimmed,                -- 'x'
  REPLACE('abracadabra', 'a', 'X') AS replaced
FROM dual;
```

2.2. SQL S01 L02

- ROUND, TRUNC round for two decimal places, whole thousands MOD

```sql
-- ROUND and TRUNC to 2 decimal places
SELECT
  ROUND(123.4567, 2) AS round_2dec,     -- 123.46
  TRUNC(123.4567, 2) AS trunc_2dec      -- 123.45
FROM dual;

-- ROUND and TRUNC to whole thousands
SELECT
  ROUND(98765, -3) AS round_thousands,  -- 99000
  TRUNC(98765, -3) AS trunc_thousands   -- 98000
FROM dual;
```

```sql
-- MOD to get remainder
SELECT
  MOD(17, 5) AS remainder               -- 2
FROM dual;
```

2.3. SQL S01 L03

- MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC
- System constant SYSDATE

```sql
-- MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC
SELECT
  MONTHS_BETWEEN(DATE '2025-05-01', DATE '2025-03-01') AS months_between,
  ADD_MONTHS(SYSDATE, 2) AS two_months_later,
  NEXT_DAY(SYSDATE, 'MONDAY') AS next_monday,
  LAST_DAY(SYSDATE) AS last_day_of_month,
  ROUND(SYSDATE, 'MONTH') AS rounded_to_month,
  TRUNC(SYSDATE, 'MONTH') AS trunc_to_month
FROM dual;
```

2.4. SQL S02 L01

- TO_CHAR, TO_NUMBER, TO_DATE

```sql
-- TO_CHAR, TO_NUMBER, TO_DATE
SELECT
  TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI') AS formatted_date,
  TO_NUMBER('12345.67', '99999.99') AS to_number_example,
  TO_DATE('2025-12-01', 'YYYY-MM-DD') AS to_date_example
FROM dual;
```

2.5. SQL S02 L02

- NVL, NVL2, NULLIF, COALESCE

```sql
-- NVL, NVL2, NULLIF, COALESCE
SELECT
  NVL(NULL, 'default') AS nvl_example,
  NVL2(NULL, 'a', 'b') AS nvl2_null,          -- 'b'
  NVL2('X', 'a', 'b') AS nvl2_notnull,        -- 'a'
  NULLIF(100, 100) AS nullif_equal,           -- NULL
  COALESCE(NULL, NULL, 'first non-null') AS coalesce_example
FROM dual;
```

2.6. SQL S02 L03

- DECODE, CASE, IF-THEN-ELSE

```sql
-- DECODE, CASE, IF-THEN-ELSE equivalent (only CASE is SQL standard)
SELECT
  DECODE(2, 1, 'one', 2, 'two', 'other') AS decode_example,
  CASE 3 WHEN 1 THEN 'one'
         WHEN 2 THEN 'two'
         ELSE 'other' END AS case_example
FROM dual;
```

2.7. SQL S03 L01

- NATURAL JOIN, CROSS JOIN

```sql
SELECT * FROM z_user NATURAL JOIN z_playlist;
SELECT * FROM z_user CROSS JOIN z_channel;
```

2.8. SQL S03 L02

- JOIN ... USING(atr)
- JOIN .. ON (joining condition)

```sql
SELECT * FROM z_video_view JOIN z_video USING(video_id);
SELECT * FROM z_video_view JOIN z_video ON (z_video_view.video_id = z_video.video_id);
```

2.9. SQL S03 L03

- LEFT OUTER JOIN ... ON ()
- RIGHT OUTER JOIN ... ON ()
- FULL OUTER JOIN ... ON ()

```sql
SELECT * FROM z_video v LEFT OUTER JOIN z_video_view vv ON v.video_id = vv.video_id;
SELECT * FROM z_video v RIGHT OUTER JOIN z_video_view vv ON v.video_id = vv.video_id;
SELECT * FROM z_video v FULL OUTER JOIN z_video_view vv ON v.video_id = vv.video_id;
```

2.10. SQL S03 L04

- Joining 2x of the same table with renaming (link between superiors and subordinates

in -ne table)
- Hierarchical querying – tree structure of START WITH, CONNECT BY PRIOR, LEVEL

dive

```sql
-- Self-join to link parent/child comments
SELECT a.comment_id AS parent_id, b.comment_id AS child_id
FROM z_comment a
JOIN z_comment b ON a.comment_id = b.parent_comment_id;
```

```
-- Hierarchical query for comment tree
SELECT LEVEL, comment_id, parent_comment_id, content
FROM z_comment
START WITH parent_comment_id IS NULL
CONNECT BY PRIOR comment_id = parent_comment_id;
```

2.11. SQL S04 L02

- AVG, COUNT, MIN, MAX, SUM, VARIANCE, STDDEV

```
SELECT
  AVG(view_count),
  COUNT(*),
  MIN(duration),
  MAX(duration),
  SUM(like_count),
  VARIANCE(dislike_count),
  STDDEV(dislike_count)
FROM z_video;
```

2.12. SQL S04 L03

- COUNT, COUNT(DISTINCT), NVL
- Difference between COUNT (*) a COUNT (attribute)
- Why using NVL for aggregation functions

```
SELECT COUNT(*) FROM z_video;
SELECT COUNT(thumbnail_id) FROM z_video;
SELECT COUNT(DISTINCT visibility) FROM z_video;
SELECT SUM(NVL(view_count, 0)) FROM z_video;
```

2.13. SQL S05 L01

- GROUP BY
- HAVING

```
SELECT channel_id, COUNT(*) AS total_videos FROM z_video GROUP BY channel_id;
SELECT channel_id, COUNT(*) AS total_videos FROM z_video GROUP BY channel_id HAVING
COUNT(*) > 5;
```

2.14. SQL S05 L02

- ROLLUP, CUBE, ROUPING SETS

```
```

```
SELECT channel_id, visibility, COUNT(*) FROM z_video GROUP BY ROLLUP(channel_id,
visibility);
SELECT channel_id, visibility, COUNT(*) FROM z_video GROUP BY CUBE(channel_id,
visibility);
SELECT channel_id, visibility, COUNT(*) FROM z_video GROUP BY GROUPING SETS ((channel_id),
(visibility));
```

2.15. SQL S05 L03

- Multiple operations in SQL – UNION, UNION ALL, INTERSECT, MINUS
- ORDER BY for set operations

```
SELECT username FROM z_user WHERE is_deleted = 0
UNION
SELECT channel_name FROM z_channel WHERE is_deleted = 0;

SELECT username FROM z_user
INTERSECT
SELECT email FROM z_user;

SELECT username FROM z_user
MINUS
SELECT email FROM z_user;

SELECT username FROM z_user
UNION ALL
SELECT channel_name FROM z_channel
ORDER BY 1;
```

2.16. SQL S06 L01

- Nested queries
- Result as a single value
- Multi-column subquery
- EXISTS, NOT EXISTS

```
SELECT username FROM z_user
WHERE user_id = (SELECT user_id FROM z_channel WHERE channel_name = 'TechWorld');

SELECT channel_id, title FROM z_video
WHERE (channel_id, visibility) IN (SELECT channel_id, visibility FROM z_playlist);

SELECT * FROM z_user u
WHERE EXISTS (
  SELECT 1 FROM z_video v
  WHERE v.channel_id IN (
    SELECT c.channel_id FROM z_channel c WHERE c.user_id = u.user_id
  )
);
```

```sql
SELECT * FROM z_user u
WHERE NOT EXISTS (
  SELECT 1 FROM z_video v
  WHERE v.channel_id IN (
    SELECT c.channel_id FROM z_channel c WHERE c.user_id = u.user_id
  )
);
```

2.17. SQL S06 L02

- One-line subqueries

```sql
SELECT username,
       (SELECT COUNT(*) FROM z_video v WHERE v.channel_id = c.channel_id) AS video_count
FROM z_channel c;
```

2.18. SQL S06 L03

- Multi-line subqueries IN, ANY, ALL
- NULL values in subqueries

```sql
SELECT username FROM z_user
WHERE user_id IN (
  SELECT user_id FROM z_playlist WHERE visibility = 'private'
);

SELECT user_id FROM z_playlist
WHERE channel_id = ANY (
  SELECT channel_id FROM z_channel WHERE is_deleted = 0
);

SELECT user_id FROM z_playlist
WHERE channel_id = ALL (
  SELECT channel_id FROM z_channel WHERE user_id IS NOT NULL
);
```

2.19. SQL S06 L04

- WITH .. AS() subquery construction

```sql
WITH recent_videos AS (
  SELECT * FROM z_video WHERE upload_date > SYSDATE - 30
)
SELECT channel_id, COUNT(*) FROM recent_videos GROUP BY channel_id;
```

2.20. SQL S07 L01

- INSERT INTO Tab VALUES()

- INSERT INTO Tab (atr, atr) VALUES()
- INSERT INTO Tab AS SELECT ...

```
-- Simple INSERT
INSERT INTO z_category (category_id, category_name) VALUES (1, 'Technology');

-- Column-specified INSERT
INSERT INTO z_user (username, first_name, last_name, email, status, last_login)
VALUES ('newuser', 'New', 'User', 'new@user.com', 'active', SYSDATE);

-- INSERT INTO ... SELECT
INSERT INTO z_playlist (user_id, title, visibility, creation_date)
SELECT user_id, 'Auto Playlist', 'public', SYSDATE FROM z_user WHERE username = 'newuser';
```

2.21. SQL S07 L02

- UPDATE Tab SET atr= .... WHERE condition
- DELETE FROM Tab WHERE atr=...

```
-- UPDATE
UPDATE z_video SET title = 'Updated Title' WHERE video_id = 1;

-- DELETE
DELETE FROM z_comment WHERE comment_id = 1;
```

2.22. SQL S07 L03

- DEFAULT, MERGE, Multi-Table Inserts

```
-- DEFAULT usage
INSERT INTO z_user (username, first_name, last_name, email, status, last_login,
is_deleted)
VALUES ('defaultuser', 'Def', 'User', 'def@user.com', 'active', SYSDATE, DEFAULT);

-- MERGE example
MERGE INTO z_user u
USING (SELECT 1 AS user_id, 'mergeuser' AS username FROM DUAL) src
ON (u.user_id = src.user_id)
WHEN MATCHED THEN UPDATE SET u.username = src.username
WHEN NOT MATCHED THEN INSERT (user_id, username, first_name, last_name, email, status,
last_login)
VALUES (src.user_id, src.username, 'M', 'U', 'mu@example.com', 'active', SYSDATE);
```

2.23. SQL S08 L01

- Objects in databases – Tables, Indexes, Constraint, View, Sequence, Synonym
- CREATE, ALTER, DROP, RENAME, TRUNCATE
- CREATE TABLE (atr DAT TYPE, DEFAULT NOT NULL)
- ORGANIZATION EXTERNAL, TYPE ORACLE_LOADER, DEFAULT DICTIONARY, ACCESS

PARAMETERS, REC-RDS DELIMITED BY NEWLINE, FIELDS, LOCATION

```
-- CREATE with constraints
CREATE TABLE z_test_object (
  id NUMBER PRIMARY KEY,
  name NVARCHAR2(100) DEFAULT 'N/A' NOT NULL
);

-- ALTER, RENAME, DROP
ALTER TABLE z_test_object ADD description NVARCHAR2(200);
RENAME z_test_object TO z_test_renamed;
DROP TABLE z_test_renamed;

-- TRUNCATE
TRUNCATE TABLE z_playlist;
```

2.24. SQL S08 L02

- TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIMEZONE
- INTERVAL YEAT TO MONTH, INTERVAL DAY TO SECOND
- CHAR, VARCHAR2, CLOB
- about NUMBER
- about BLOB

```
-- Advanced data types
CREATE TABLE z_types_demo (
  ts TIMESTAMP,
  ts_tz TIMESTAMP WITH TIME ZONE,
  ts_ltz TIMESTAMP WITH LOCAL TIME ZONE,
  iv1 INTERVAL YEAR TO MONTH,
  iv2 INTERVAL DAY TO SECOND,
  c CHAR(10),
  vc VARCHAR2(100),
  txt CLOB,
  n NUMBER,
  bin BLOB
);
```

2.25. SQL S08 L03

- ALTER TABLE (ADD, MODIFY, DROP), DROP, RENAME
- FLASHBACK TABLE Tab TO BEFORE DROP (view USER_RECYCLEBIN)
- DELETE, TRUNCATE
- COMMENT ON TABLE
- SET UNUSED

```
-- ALTER operations
ALTER TABLE z_user ADD bio NVARCHAR2(100);
ALTER TABLE z_user MODIFY email NVARCHAR2(300);
```

```sql
ALTER TABLE z_user DROP COLUMN bio;

-- FLASHBACK (assumes privilege enabled)
FLASHBACK TABLE z_user TO BEFORE DROP;

-- COMMENT
COMMENT ON TABLE z_user IS 'Holds user accounts';

-- SET UNUSED
ALTER TABLE z_user SET UNUSED (about_me);
```

2.26. SQL S10 L01

- CREATE TABLE (NOT NULL AND UNIQUE constraint)
- CREATE TABLE Tab AS SELECT ...
- Own vs. system naming CONSTRAINT conditions

```sql
-- Table with named constraints
CREATE TABLE z_example_constraints (
  id NUMBER CONSTRAINT pk_example PRIMARY KEY,
  name NVARCHAR2(100) CONSTRAINT nn_example_name NOT NULL,
  code NVARCHAR2(10) CONSTRAINT uq_example_code UNIQUE
);

-- CREATE TABLE AS SELECT
CREATE TABLE z_user_copy AS SELECT * FROM z_user;
```

2.27. SQL S10 L02

- CONSTRAINT – NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY (atr REFERENCES

Tab(atr)), CHECK
- Foreign keys, ON DELETE, ON UPDATE, RESTRICT, CASCADE, etc.

```sql
-- Foreign key with ON DELETE CASCADE
CREATE TABLE z_fk_demo (
  id NUMBER PRIMARY KEY,
  user_id NUMBER,
  CONSTRAINT fk_demo_user FOREIGN KEY (user_id) REFERENCES z_user(user_id) ON DELETE
CASCADE
);

-- CHECK constraint
ALTER TABLE z_fk_demo ADD CONSTRAINT ck_demo_id CHECK (id > 0);
```

2.28. SQL S10 L03

- about USER_CONSTRAINTS

```sql
-- Inspect user constraints
SELECT constraint_name, constraint_type, table_name
FROM user_constraints
WHERE table_name = 'Z_USER';
```

2.29. SQL S11 L01

- CREATE VIEW
- about FORCE, NOFORCE
- WITCH CHECK OPTION
- WITH READ ONLY
- about Simple vs. Compex VIEW

```sql
-- Simple view
CREATE OR REPLACE VIEW v_active_users AS
SELECT user_id, username, email FROM z_user WHERE is_deleted = 0;

-- View with CHECK OPTION
CREATE OR REPLACE VIEW v_public_playlists AS
SELECT * FROM z_playlist WHERE visibility = 'public'
WITH CHECK OPTION;

-- Read-only view
CREATE OR REPLACE VIEW v_readonly AS
SELECT video_id, title FROM z_video
WITH READ ONLY;
```

2.30. SQL S11 L03

- INLINE VIEW Subquery in the form of a table SELECT atr FROM (SELECT * FROM Tab)

alt_tab

```sql
-- Inline view
SELECT alt.title FROM (
  SELECT title FROM z_video WHERE is_deleted = 0
) alt;
```

2.31. SQL S12 L01

- CREATE SEQUENCE name INCREMENT BY n START WITH m, (NO)MAXVALUE,

(NO)MINVALUE, (NO)CYCLE, (NO)CACHE
- about ALTER SEQUENCE

```sql
-- Sequence creation and alteration
CREATE SEQUENCE seq_channel_id INCREMENT BY 1 START WITH 100
NOMAXVALUE NOCYCLE NOCACHE;
```

```
ALTER SEQUENCE seq_channel_id INCREMENT BY 5;
```

2.32. SQL S12 L02

- CREATE INDEX, PRIMARY KEY, UNIQUE KEY, FOREIGN KEY

```
-- Index creation
CREATE INDEX idx_user_email ON z_user(email);
```

2.33. SQL S13 L01

- GRANT ... ON ... TO ... PUBLIC
- about REVOKE
- What rights can be assigned to which objects? (ALTER, DELETE, EXECUTE, INDEX,

INSERT, REFERENCES, SELECT, UPDATE) – (TABLE, VIEW, SEQUENCE, PROCEDURE)

```
-- Granting privileges
GRANT SELECT, INSERT, UPDATE ON z_user TO PUBLIC;
GRANT DELETE ON z_video TO some_user;

-- Revoking privileges
REVOKE INSERT, UPDATE ON z_user FROM PUBLIC;
```

2.34. SQL S13 L03

- Regular expressions
- REGEXP_LIKE, REGEXP_REPLACE, REGEXP_INSTR, REGEXP_SUBSTR, REGEXP_COUNT

```
-- Regular expressions
SELECT * FROM z_user WHERE REGEXP_LIKE(email, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-
z]{2,}$');
SELECT REGEXP_REPLACE('123-456-7890', '[^0-9]', '') FROM DUAL;
SELECT REGEXP_INSTR('a1b2c3', '[0-9]') FROM DUAL;
SELECT REGEXP_SUBSTR('abc123def456', '[0-9]+', 1, 2) FROM DUAL;
SELECT REGEXP_COUNT('x1y2z3', '[0-9]') FROM DUAL;
```

2.35. SQL S14 L01

- Transactions, COMMIT, ROLLBACK, SAVEPOINT

```
-- Transaction control
SAVEPOINT before_update;
UPDATE z_user SET email = 'new@email.com' WHERE user_id = 1;
```

```
ROLLBACK TO before_update;
COMMIT;
```

2.36. SQL S15 L01

- Alternative join notation without JOIN with join condition in WHERE
- Left and right connection using atrA = atrB (+)

```
-- Join using WHERE (legacy style)
SELECT * FROM z_user u, z_channel c WHERE u.user_id = c.user_id;

-- LEFT and RIGHT JOIN using (+) operator
SELECT * FROM z_video v, z_video_view vv WHERE v.video_id = vv.video_id(+);
SELECT * FROM z_video_view vv, z_video v WHERE v.video_id(+) = vv.video_id;
```

2.37. SQL S16 L03

- Recapitulation of commands and parameters - complete everything that was not mentioned in the previous points here