

Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Technologie databázových systémů 1

Semestrální projekt

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 21-05-2025

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 | 8 |
| 1.10. DD S06 L01 | 8 |
| 1.11. DD S06 L02-04 | 8 |
| 1.12. DD S07 L01 | 8 |
| 1.13. DD S07 L02 | 9 |
| 1.14. DD S07 L03 | 9 |
| 1.15. DD S09 L01 | 9 |
| 1.16. DD S09 L02 | 10 |
| 1.17. DD S10 L01 | 10 |
| 1.18. DD S10 L02 | 11 |
| 1.19. DD S11 L01 | 12 |
| 1.20. DD S11 L02-04 | 13 |
| 1.21. DD S15 L01 | 14 |
| 1.22. DD S16 L02 | 14 |
| 1.23. DD S16 L03 | 14 |
| 1.24. DD S17 L01 | 15 |
| 1.25. DD S17 L02 | 15 |
| 1.26. DD S17 L03 | 15 |
| 2. Programming with SQL | 16 |
| 2.1. SQL S01 L01 | 16 |

| | | |
|-------|-------------------|----|
| 2.2. | SQL S01 L02 | 17 |
| 2.3. | SQL S01 L03 | 17 |
| 2.4. | SQL S02 L01 | 18 |
| 2.5. | SQL S02 L02 | 18 |
| 2.6. | SQL S02 L03 | 18 |
| 2.7. | SQL S03 L01 | 19 |
| 2.8. | SQL S03 L02 | 19 |
| 2.9. | SQL S03 L03 | 19 |
| 2.10. | SQL S03 L04 | 20 |
| 2.11. | SQL S04 L02 | 22 |
| 2.12. | SQL S04 L03 | 23 |
| 2.13. | SQL S05 L01 | 23 |
| 2.14. | SQL S05 L02 | 24 |
| 2.15. | SQL S05 L03 | 24 |
| 2.16. | SQL S06 L01 | 25 |
| 2.17. | SQL S06 L02 | 25 |
| 2.18. | SQL S06 L03 | 26 |
| 2.19. | SQL S06 L04 | 26 |
| 2.20. | SQL S07 L01 | 27 |
| 2.21. | SQL S07 L02 | 27 |
| 2.22. | SQL S07 L03 | 28 |
| 2.23. | SQL S08 L01 | 29 |
| 2.24. | SQL S08 L02 | 30 |
| 2.25. | SQL S08 L03 | 31 |
| 2.26. | SQL S10 L01 | 32 |
| 2.27. | SQL S10 L02 | 32 |
| 2.28. | SQL S10 L03 | 33 |
| 2.29. | SQL S11 L01 | 33 |
| 2.30. | SQL S11 L03 | 34 |
| 2.31. | SQL S12 L01 | 34 |
| 2.32. | SQL S12 L02 | 35 |
| 2.33. | SQL S13 L01 | 35 |
| 2.34. | SQL S13 L03 | 35 |
| 2.35. | SQL S14 L01 | 36 |
| 2.36. | SQL S15 L01 | 37 |

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 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 their 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_ThumbnaillImage | 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.

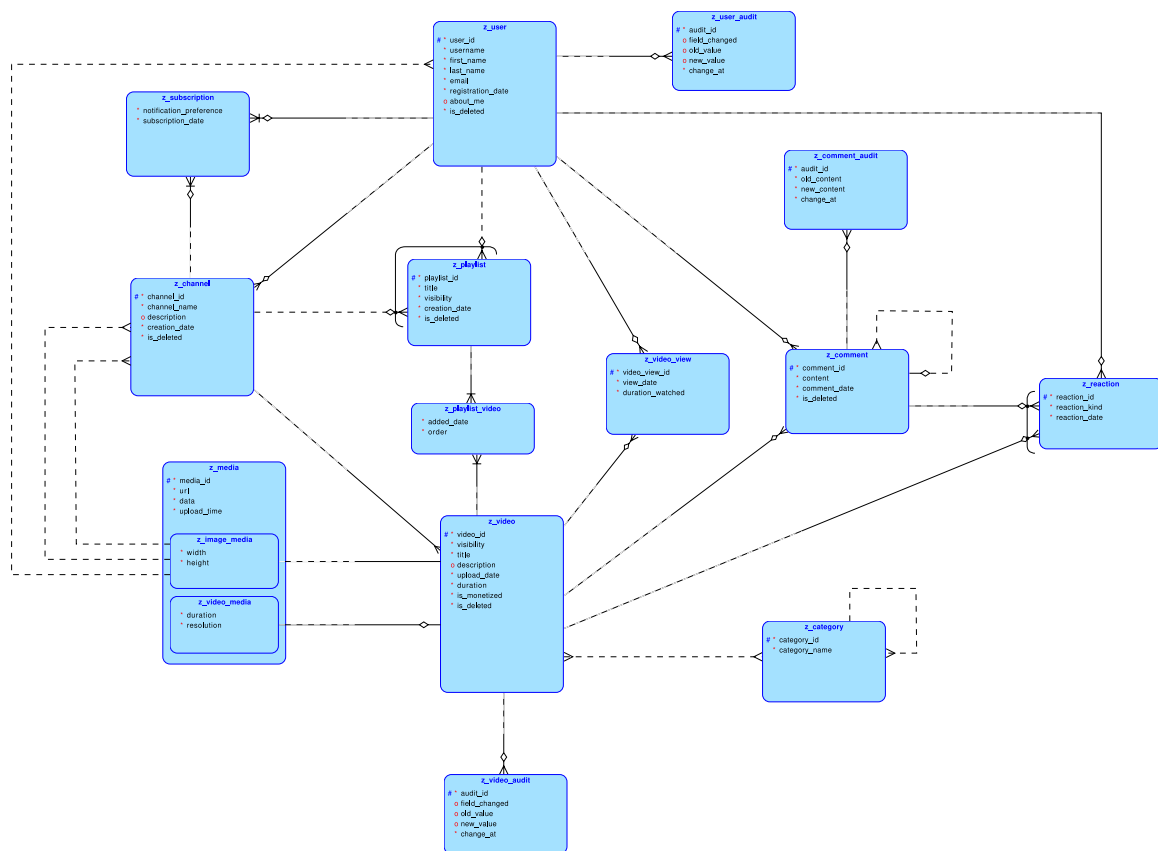


Figure 1: ER Diagram

1.5. DD S30 L04

Matrix diagram with relationships, draw for your solution.

Should be read starting from the left.

| | 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 | | | has subscribers | created by user | has videos | | |
| comment | | | has replies | | | | | receives reactions | | written by user | on video | | |
| image_media | | used by channel as pfp and banner | | — | is subtype of | | | | | used in user for pfp | | | |
| media | | | | is supertype of | — | | | | | | | is supertype of | |
| playlist | | created by | | | | — | contains videos | | | created by user | | | |
| playlist_video | | | | | | belongs in | — | | | | references video | | |
| reaction | | | on comment | | | | | — | | by user | on video | | |
| subscription | | to channel | | | | | | | — | by user | | | |
| 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 | | viewed by users | — | has video file | has views |
| video_media | | | | | is subtype of | | | | | | referenced by | — | |
| video_view | | | | | | | | | | made by user | 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`, `data` and `created_at` attributes from `z_media` but introduce additional attributes:

- `z_image_media`: width, height, format
- `z_video_media`: duration, resolution, codec

1.7. DD S04 L02

Description of business rules for your project.

- **Channel Ownership**
 - channel cannot be reassigned to another user post-creation
 - enforced by a BEFORE UPDATE trigger
- **Reaction Targeting**
 - 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 constraint
- **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, etc.

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
 - supports reuse of shared images
- **Non-Transferable Binding**
 - `z_comment.user_id` is locked after creation via a BEFORE UPDATE trigger
 - non-transferable, immutable relationship

1.9. DD S05 L03

Have at least one M:N relationship without information and one M:N relationship with information 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
- playlist_id and video_id of table z_playlist_video are identifying relationships, because the combination of them is the PK of z_playlist_video table

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.

- *Normal Form Recap*
 - 1NF - no repeating groups or arrays; each attribute holds atomic (indivisible) values
 - 2NF - must be in 1NF + no partial dependency on part of a composite key (i.e., every non-key attribute must depend on the whole primary key)
 - 3NF - must be in 2NF + no transitive dependencies (i.e., non-key attributes must not depend on other non-key attributes)

All the tables are in 3NF, because

- all non-key attributes functionally depend only on the primary key and
- there are no transitive dependencies.

1.12. DD S07 L01

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

The table 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).

Both are actually hierarchical as well as recursive.

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 (via triggers). 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 refers to the design of data structures or database schemas in a way that is flexible, reusable, and abstracted from specific business cases. Rather than hardcoding entities or attributes for every use case (e.g., product_category, video_category, blog_category), a generic model creates unified, abstract structures that can serve many domains.

Example of domain specific non-general tables:

```
CREATE TABLE video_category (...);  
CREATE TABLE blog_category (...);
```

In this design each use case has its own separate table.

Generic model would use something like this:

```
CREATE TABLE category (  
    category_id INTEGER PRIMARY KEY,  
    parent_category_id INTEGER REFERENCES category(category_id),  
    name VARCHAR2(100)  
);  
  
CREATE TABLE entity_category (  
    entity_type VARCHAR2(50), -- e.g., 'video', 'blog'  
    entity_id INTEGER,  
    category_id INTEGER,  
    PRIMARY KEY (entity_type, entity_id, category_id)  
);
```

where entity_type may hold the name of the target table and the entity_id holds the id of the record it refers to.

This approach is advantageous when traditional design patterns like inheritance, polymorphic arcs would necessitate creations of large amounts of 'helper' tables. Generic modeling mitigates this problem of bloating the database model with such 'helper' tables by allowing storing field names and table names as field values of another table.

Although this pattern is somewhat of an anti-pattern it allows for scalable and simple models.

In the project there is z_user_audit table that holds historical data of the z_user table. In this table there is field_changed attribute which values are field names of the z_user table.

1.19. DD S11 L01

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

- **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
 - 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
 - in z_playlist, only one of user_id or channel_id can be non-null
 - enforced via CHECK and TRIGGER

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.

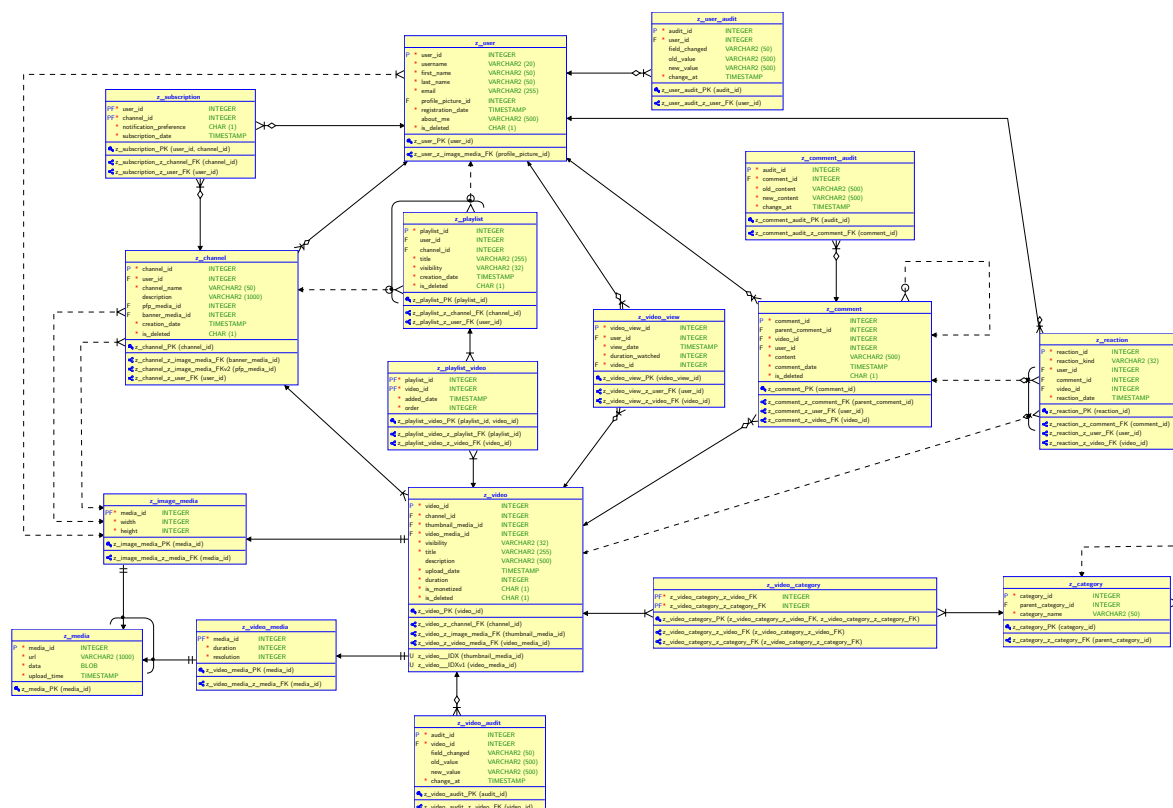


Figure 2: ER Diagram

Key changes from conceptual model

- Relations are modeled by adding foreign keys to tables.
- Many to many relations are modeled by adding an 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`.

```
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`.

```
SELECT * FROM z_user
WHERE LOWER(username) = 'phatt-23';
```

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

```
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`.

```
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 precedence.

```
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* attr [*ASC/DESC*] - Sorting by using one or more attributes.

```
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*.

```
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

```
-- 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

```
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;

-- 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

```
-- MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC
SELECT
    SYSDATE AS now,
    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

```
-- TO_CHAR, TO_NUMBER, TO_DATE
SELECT
    TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI') AS formatted_date_varchar2,
    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

```
-- NVL, NVL2, NULLIF, COALESCE
SELECT
    NVL(NULL, 'default') AS nvl_example,           -- 'default'
    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

```
-- DECODE, CASE, IF-THEN-ELSE equivalent (only CASE is SQL standard)
SELECT
    DECODE(2,           -- expr switch
        1, 'one',      -- expr1 ⇒ 'one'
        2, 'two',      -- expr2 ⇒ 'two'
        'other'        -- _ ⇒ 'other'
    ) AS decode_example,
    CASE 3
        WHEN 1 THEN 'one'
        WHEN 2 THEN 'two'
        WHEN 4 THEN 'four'
        ELSE 'other' END
    AS case_example
FROM dual;
```

2.7. SQL S03 L01

- NATURAL JOIN, CROSS JOIN

```
SELECT * FROM z_user NATURAL JOIN z_playlist;  -- joins by common column names
SELECT * FROM z_user CROSS JOIN z_channel;  -- cartesian product, can't specify join
clause
```

2.8. SQL S03 L02

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

```
SELECT *
FROM z_video_view
JOIN z_video USING(video_id);  -- USING(column1, column2, ...)

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 ()

```
SELECT * FROM z_video v LEFT OUTER JOIN z_video_view vv ON v.video_id = vv.video_id;
-- same as
SELECT * FROM z_video v LEFT 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 one table)
- Hierarchical querying – tree structure of START WITH, CONNECT BY PRIOR, LEVEL

```
SELECT * FROM z_comment;

DECLARE
    v_parent_comment_id INT := 12;
    v_user_id INT := 12;
    v_video_id INT := 1;
BEGIN
    FOR i IN 0..10 LOOP
        INSERT INTO z_comment (parent_comment_id, video_id, user_id, content)
        VALUES (
            (SELECT comment_id FROM z_comment WHERE comment_id = v_parent_comment_id),
            (SELECT video_id FROM z_video WHERE video_id = v_video_id),
            (SELECT user_id FROM z_user WHERE user_id = v_user_id),
            'hello, world! reply to comment ' || i
        );
    END LOOP;
END;
```

```

-- 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;

DECLARE
    v_level INT := 0;
    v_comment_id Z_COMMENT.comment_id%TYPE := 0;
    v_parent_comment_id Z_COMMENT.comment_id%TYPE := 0;
    v_content Z_COMMENT.content%TYPE := '';
BEGIN
    FOR v_rec IN (
        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
    ) LOOP
        FOR v_tab IN 1..v_rec.LEVEL LOOP
            DBMS_OUTPUT.PUT(' ');
        END LOOP;

        DBMS_OUTPUT.PUT_LINE(v_rec.comment_id || ' → ' || v_rec.content);
    END LOOP;
END;

```

2.11. SQL S04 L02

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

```
BEGIN
  FOR rec IN (SELECT * FROM z_user CROSS JOIN z_video)
  LOOP
    INSERT INTO z_video_view(user_id, duration_watched, video_id)
    VALUES (rec.user_id, rec.duration / 2, rec.video_id);
  END LOOP;
END;

WITH t_video_view AS (
  SELECT
    v.video_id AS video_id,
    v.duration AS duration,
    COUNT(vv.video_view_id) AS view_count
  FROM z_video v
  JOIN z_video_view vv ON v.video_id = vv.video_id
  GROUP BY v.video_id, v.duration
), t_reaction_count AS (
  SELECT (
    SELECT COUNT(*) FROM z_reaction
    WHERE video_id = v.video_id AND reaction_kind = 'like'
  ) AS like_count,
  (
    SELECT COUNT(*) FROM z_reaction
    WHERE video_id = v.video_id AND reaction_kind = 'dislike'
  ) AS dislike_count
  FROM z_video v
)
SELECT
  AVG(view_count),
  COUNT(*),
  MIN(duration),
  MAX(duration),
  SUM(like_count),
  AVG(dislike_count),
  VARIANCE(dislike_count), -- Var(X) = E((X - mean) ** 2)
  SQRT(VARIANCE(dislike_count)) AS standard_deviation,
  STDDEV(dislike_count)
FROM t_video_view, t_reaction_count;
```

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_media_id) FROM z_video;
SELECT COUNT(DISTINCT visibility) FROM z_video;

WITH t AS (
    SELECT v.video_id AS video_id, NULLIF(COUNT(VIDEO_VIEW_ID), 0) AS view_count
    FROM z_video v LEFT JOIN z_video_view vv ON v.video_id = vv.video_id
    GROUP BY v.video_id
)
SELECT
    AVG(NVL(view_count, 0)),
    -- not the same AVG, null is disregarded by default
    AVG(view_count)
FROM t;
```

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(*) > 1;
```


2.14. SQL S05 L02

- ROLLUP, CUBE, GROUPING 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 * FROM z_user;
SELECT * FROM z_user WHERE email LIKE '%yahoo.com';
```

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

```
SELECT * FROM z_user WHERE email LIKE '%gmail.com'; -- 3
SELECT * FROM z_user WHERE first_name = 'Shelly'; -- 1
```

```
SELECT username FROM z_user WHERE first_name = 'Shelly'
INTERSECT -- 1
SELECT username FROM z_user WHERE email LIKE '%gmail.com';
```

```
SELECT username FROM z_user WHERE email LIKE '%gmail.com'
MINUS -- 2
SELECT username FROM z_user WHERE first_name = 'Shelly';
```

```
SELECT username FROM z_user
UNION ALL
SELECT channel_name FROM z_channel
ORDER BY 1; -- first item in the select list
-- ORDER BY username; -- could also be this though
```

2.16. SQL S06 L01

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

```
SELECT * FROM z_channel;
```

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

```
SELECT v.channel_id, v.title FROM z_video v  
WHERE (v.channel_id, v.visibility) IN (SELECT p.channel_id, p.visibility FROM z_playlist  
p);
```

```
-- users with at least one video
```

```
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  
    )  
);
```

2.17. SQL S06 L02

- One-line subqueries

```
SELECT channel_name,  
    (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

```
-- users with private playlists
SELECT username FROM z_user
WHERE user_id IN (SELECT user_id FROM z_playlist WHERE visibility = 'private');

-- playlists that belong to deleted channels
SELECT playlist_id, channel_id FROM z_playlist
WHERE channel_id = ANY (
    SELECT channel_id
    FROM z_channel
    WHERE is_deleted = 1
);

-- playlist with the highest "order"
SELECT pv1.playlist_id, "order" FROM z_playlist_video pv1
WHERE "order" ≥ ALL (
    SELECT "order" FROM z_playlist_video
);
```

2.19. SQL S06 L04

- WITH ... AS() subquery construction

```
SELECT SYSDATE - 30 AS back_x_days, ADD_MONTHS(SYSDATE, 2) AS back_x_months FROM dual;

-- channels that uploaded videos last x years
WITH recent_videos AS (
    SELECT * FROM z_video
    WHERE upload_date > ADD_MONTHS(SYSDATE, -2 * 12)
)
SELECT channel_id, COUNT(*) AS video_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 VALUES (1, 11, 'Technology');

-- Column-specified INSERT
INSERT INTO z_category(parent_category_id, category_name)
VALUES (11, 'Technology');

INSERT INTO z_user (username, first_name, last_name, email)
VALUES ('newuser', 'New', 'User', 'new@user.com');

-- 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, is_deleted)
VALUES ('defaultuser', 'Def', 'User', 'def@user.com', DEFAULT);

-- MERGE example
MERGE INTO z_user u                                -- destination
  USING
  (
    SELECT 1 AS user_id, 'mergeuser' AS username FROM DUAL -- data source
  ) src
  ON (u.user_id = src.user_id)                       -- condition
  WHEN MATCHED THEN                                 -- update clause
    UPDATE SET u.username = src.username || '_updated'
  WHEN NOT MATCHED THEN                             -- insert clause
    INSERT (user_id, username, first_name, last_name, email)
    VALUES (
      src.user_id,
      src.username,
      'M',
      'U',
      'mu@example.com'
    );

DELETE FROM z_user WHERE username LIKE 'mergeuser%';
SELECT * FROM z_user;
SELECT * FROM z_user WHERE username LIKE 'mergeuser%';
```

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, RECORDS DELIMITED BY NEWLINE, FIELDS, LOCATION**

```
DROP TABLE Z_TEST_OBJECT;

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

SELECT * FROM z_test_object;

-- 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
INSERT INTO z_test_renamed(id, name)
SELECT user_id, username FROM z_user;

SELECT * FROM z_test_renamed;

TRUNCATE TABLE z_test_renamed;
```

2.24. SQL S08 L02

- TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIMEZONE
- INTERVAL YEAR 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, -- stores intervals using year and month
  iv2 INTERVAL DAY TO SECOND, -- stores intervals using days, hours, minutes, and
seconds including fractional seconds
  c CHAR(10),
  vc VARCHAR2(100),
  txt CLOB,
  n NUMBER,
  bin BLOB
);

SELECT * FROM z_types_demo;

DROP TABLE z_types_demo;

CREATE TABLE z_demo_student (
  years_of_undergrad INTERVAL YEAR TO MONTH
);

INSERT INTO z_demo_student VALUES (INTERVAL '10-2' YEAR TO MONTH);
INSERT INTO z_demo_student VALUES (INTERVAL '2-3' YEAR TO MONTH);
INSERT INTO z_demo_student VALUES (INTERVAL '9' MONTH);

SELECT * FROM z_demo_student;

DROP TABLE z_demo_student;
```

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_demo_student ADD email NVARCHAR2(100);
ALTER TABLE z_demo_student MODIFY email NVARCHAR2(300);
ALTER TABLE z_demo_student DROP COLUMN email;

-- FLASHBACK
FLASHBACK TABLE z_demo_student TO BEFORE DROP;

-- COMMENT
COMMENT ON TABLE z_demo_student IS 'Holds student info and stuff';

-- SET UNUSED
-- source: https://forums.oracle.com/ords/apexds/post/set-unused-and-drop-column-4001
-- The SET UNUSED option marks one or more columns as unused so
-- that they can be dropped when the demand on system resources
-- is lower. Specifying this clause does not actually remove the
-- target columns from each row in the table (that is, it does not
-- restore the disk space used by these columns).Therefore, the
-- response time is faster than if you executed the DROP clause.
-- Unused columns are treated as if they were dropped, even though
-- their column data remains in the tables rows. After a column
-- has been marked as unused, you have no access to that column.
ALTER TABLE z_demo_student SET UNUSED (email);
```


2.26. SQL S10 L01

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

```
-- 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;

SELECT * FROM z_user_copy;
DROP TABLE z_user_copy;
```

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.

```
-- 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

```
-- 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
 - WITH CHECK OPTION
 - WITH READ ONLY
- about Simple vs. Complex VIEW

```
-- 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
-- any INSERT or UPDATE done through the view must result
-- in rows that are still visible in the view
-- → that is, they must satisfy the view's WHERE condition.
CREATE OR REPLACE VIEW v_public_playlists AS
SELECT *
FROM z_playlist
WHERE visibility = 'public'
WITH CHECK OPTION;

SELECT *
FROM v_public_playlists;

-- 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

```
-- 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

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

ALTER SEQUENCE seq_demo_student_id INCREMENT BY 5;

CREATE TABLE z_demo_student (
    id NUMBER DEFAULT seq_demo_student_id.nextval,
    name VARCHAR2(40 CHAR)
);

INSERT INTO z_demo_student(name) VALUES ('john');
INSERT INTO z_demo_student(name) VALUES ('doe');

SELECT * FROM z_demo_student;

DROP SEQUENCE seq_demo_student_id;
DROP TABLE z_demo_student;
```

2.32. SQL S12 L02

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

```
-- Index creation
CREATE INDEX idx_z_demo_student_name ON z_demo_student(name);
```

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 SELECT, 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 email FROM z_user;
SELECT email 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]', 'non_number') FROM DUAL;
SELECT REGEXP_INSTR('aa1b2c3', '[0-9]') FROM DUAL; -- returns index 3 because number '1'
is at that position
SELECT REGEXP_SUBSTR(
    'abc123def456aaa111bbb222',
    '[0-9]+',
    10,                               -- starting pos in source string
    3                                 -- occurrence
)
FROM DUAL;
SELECT REGEXP_COUNT('x1y2z3', '[0-9]') FROM DUAL;
```

2.35. SQL S14 L01

- Transactions, COMMIT, ROLLBACK, SAVEPOINT

```
SELECT * FROM z_user WHERE user_id = 1;

-- Transaction control
DECLARE
    v_success INT := 0;
BEGIN
    SAVEPOINT before_update;

    UPDATE z_user SET email = 'new@email.com' WHERE user_id = 1;

    IF v_success = 0 THEN
        RAISE_APPLICATION_ERROR(50001, 'Transaction failed :(');
    END IF;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Committed changes');
EXCEPTION WHEN OTHERS THEN
    ROLLBACK TO before_update;
    DBMS_OUTPUT.PUT_LINE('Rolling back changes');
END;
```

2.36. SQL S15 L01

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

```
-- 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
LEFT JOIN z_video_view vv ON v.video_id = vv.video_id
WHERE vv.VIDEO_VIEW_ID IS NULL
;

SELECT *
FROM z_video v,
     z_video_view vv
WHERE v.video_id = vv.video_id(+) -- the side with (+) is 'added'
      AND vv.VIDEO_VIEW_ID IS NULL
;

SELECT *
FROM z_video_view vv
LEFT JOIN z_video v ON vv.video_id = v.video_id
;

SELECT *
FROM z_video_view vv,
     z_video v
WHERE v.video_id(+) = vv.video_id;
```