Contents

6 **19**

1 SQL

1.1

SQL SQL SQL SQL

1.2

```
erDiagram
    USERS ||--o{ ENROLLMENTS : "enrolls"
    COURSES ||--o{ ENROLLMENTS : "enrolled_in"
    COURSES ||--o{ LESSONS : "contains"
    USERS ||--o{ USER_PROGRESS : "tracks"
    LESSONS ||--o{ USER_PROGRESS : "progressed_in"
    CATEGORIES ||--o{ COURSES : "categorizes"

    USERS {
        int user_id PK
        varchar name
        varchar email
        date birth_date
        enum gender
        datetime created_at
        datetime updated_at
        boolean is_active
    }

    CATEGORIES {
        int category_id PK
        varchar name
        text description
        datetime created_at
    }

    COURSES {
        int course_id PK
        varchar title
        text description
        int category_id FK
        decimal price
        int duration_hours
        enum difficulty_level
        datetime created_at
        boolean is_published
    }

    LESSONS {
        int lesson_id PK
        int course_id FK
        varchar title
        text content
        int duration_minutes
        int sequence_number
        datetime created_at
    }
```

```
    ENROLLMENTS {
        int enrollment_id PK
        int user_id FK
        int course_id FK
        datetime enrollment_date
        decimal amount_paid
        enum status
    }

    USER_PROGRESS {
        int progress_id PK
        int user_id FK
        int lesson_id FK
        datetime completed_at
        int time_spent_minutes
    }
```

---

2 (DDL) -

DDL ” ” ” ” CREATE ALTER DROP

2.1 1.1 CREATE -

2.1.1

```
--
CREATE DATABASE online_education_platform
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

2.1.2

```
-- 1.    -
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,  --
    name VARCHAR(100) NOT NULL,                 --
    email VARCHAR(255) NOT NULL UNIQUE,         --
    birth_date DATE,                            --
    gender ENUM('male', 'female', 'other') DEFAULT 'other',  --
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  --
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true,

    --
    CONSTRAINT chk_age CHECK (birth_date <= date('now', '-13 years')),  --     13
    INDEX idx_email (email),                    --
    INDEX idx_created_at (created_at)
);

-- 2.
CREATE TABLE categories (
    category_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(50) NOT NULL UNIQUE,
    description TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- 3.    -
CREATE TABLE courses (
    course_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    category_id INTEGER NOT NULL,
    price DECIMAL(10,2) DEFAULT 0.00,
    duration_hours INTEGER DEFAULT 0,
    difficulty_level ENUM('beginner', 'intermediate', 'advanced') DEFAULT 'beginner',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    is_published BOOLEAN DEFAULT false,

    --
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,

    --
    CONSTRAINT chk_price CHECK (price >= 0),
    CONSTRAINT chk_duration CHECK (duration_hours >= 0),

    --
    INDEX idx_category (category_id),
    INDEX idx_price (price),
    INDEX idx_difficulty (difficulty_level)
);
```

2.1.3 -

```sql
--
CREATE INDEX idx_users_name ON users(name);

--
CREATE INDEX idx_course_category_price ON courses(category_id, price);

--
CREATE UNIQUE INDEX idx_users_email_unique ON users(email);

--    SQLite
CREATE INDEX idx_active_users ON users(name) WHERE is_active = true;
```

2.1.4 -

```sql
--
CREATE VIEW active_user_course_stats AS
SELECT
    u.user_id,
    u.name,
    u.email,
    COUNT(e.course_id) as enrolled_courses,
    SUM(e.amount_paid) as total_spent,
    AVG(c.price) as avg_course_price
FROM users u
LEFT JOIN enrollments e ON u.user_id = e.user_id
LEFT JOIN courses c ON e.course_id = c.course_id
WHERE u.is_active = true
GROUP BY u.user_id, u.name, u.email;
```

2.2 1.2 ALTER -

```
--
ALTER TABLE users ADD COLUMN phone VARCHAR(20);
ALTER TABLE users ADD COLUMN profile_picture_url TEXT;

--
ALTER TABLE courses ALTER COLUMN title SET NOT NULL;  --
ALTER TABLE courses ALTER COLUMN description TYPE TEXT;  --

--
ALTER TABLE users ADD CONSTRAINT chk_phone_format
CHECK (phone IS NULL OR length(phone) >= 10);

--
ALTER TABLE users DROP CONSTRAINT chk_phone_format;

--
ALTER TABLE users RENAME COLUMN phone TO phone_number;

--
ALTER TABLE users RENAME TO platform_users;
ALTER TABLE platform_users RENAME TO users;  --
```

2.3 1.3 DROP -

```
--
DROP INDEX idx_users_name;

--
DROP VIEW active_user_course_stats;

--
-- DROP TABLE courses;  --

--
-- DROP DATABASE online_education_platform;
```

2.4 1.4 TRUNCATE -

```
--
TRUNCATE TABLE user_progress;  -- DELETE FROM table

--   TRUNCATE  DELETE
```

DDL 1. snake_case 2. 3. 4. DDL

---

3 (DML) -

DML INSERT UPDATE DELETE

3.1 2.1 INSERT -

3.1.1

```
-- 1.
INSERT INTO categories (name, description)
VALUES ('  ', '         ');
```

```sql
INSERT INTO categories (name, description) VALUES
('  ', '         '),
('  ', '   UI/UX     '),
('  ', '         ');

-- 2.
INSERT INTO users (name, email, birth_date, gender) VALUES
(' ', 'zhang.san@email.com', '1995-05-15', 'male'),
(' ', 'li.si@email.com', '1992-08-22', 'female'),
(' ', 'wang.wu@email.com', '1988-12-03', 'male'),
(' ', 'zhao.liu@email.com', '1990-03-18', 'female');

-- 3.
INSERT INTO courses (title, description, category_id, price, duration_hours, difficulty_level, is_published) V
('Python    ', '  Python           ', 1, 199.00, 40, 'beginner', true),
('    ', ' Python     pandas numpy matplotlib', 2, 299.00, 30, 'intermediate', true),
('UI    ', 'Figma Sketch      ', 3, 159.00, 25, 'beginner', true);
```

3.1.2

```sql
-- 1.     INSERT ... SELECT
INSERT INTO user_progress (user_id, lesson_id, completed_at, time_spent_minutes)
SELECT
    u.user_id,
    l.lesson_id,
    datetime('now', '-' || abs(random()) % 30 || ' days'),  --    30
    30 + abs(random()) % 60  -- 30-90
FROM users u
CROSS JOIN lessons l
WHERE u.user_id <= 2 AND l.lesson_id <= 3;  --

-- 2.     UPSERT
INSERT INTO users (email, name, updated_at)
VALUES ('zhang.san@email.com', '    ', datetime('now'))
ON CONFLICT(email) DO UPDATE SET
    name = excluded.name,
    updated_at = excluded.updated_at;

-- 3.
INSERT INTO enrollments (user_id, course_id, enrollment_date, amount_paid, status) VALUES
(1, 1, datetime('now'), 199.00, 'active'),
(1, 2, datetime('now'), 299.00, 'active'),
(2, 1, datetime('now'), 199.00, 'active'),
(3, 3, datetime('now'), 159.00, 'completed'),
(4, 1, datetime('now'), 199.00, 'active');
```

3.2 2.2 UPDATE -

3.2.1

```sql
-- 1.
UPDATE users
SET name = '  ', updated_at = datetime('now')
WHERE email = 'zhang.san@email.com';

-- 2.
UPDATE courses
SET price = price * 0.8  --  8
WHERE category_id = 1 AND is_published = true;
```

```sql
-- 3.  CASE
UPDATE courses
SET difficulty_level =
    CASE
        WHEN duration_hours < 20 THEN 'beginner'
        WHEN duration_hours BETWEEN 20 AND 40 THEN 'intermediate'
        ELSE 'advanced'
    END
WHERE difficulty_level IS NULL;
```

3.2.2

```sql
-- 1.
UPDATE users
SET is_active = false
WHERE user_id IN (
    SELECT DISTINCT u.user_id
    FROM users u
    LEFT JOIN enrollments e ON u.user_id = e.user_id
    WHERE e.enrollment_date < date('now', '-365 days')
    OR e.enrollment_date IS NULL
);

-- 2.  JOIN
UPDATE courses
SET price = c.price * 1.1
FROM courses c
JOIN categories cat ON c.category_id = cat.category_id
WHERE cat.name = '  ' AND c.created_at < date('now', '-180 days');
```

3.3 2.3 DELETE -

3.3.1

```sql
-- 1.
DELETE FROM user_progress
WHERE completed_at < date('now', '-365 days');

-- 2.
DELETE FROM enrollments
WHERE course_id IN (
    SELECT course_id
    FROM courses
    WHERE is_published = false AND created_at < date('now', '-90 days')
);
```

3.3.2

```sql
-- 1.
--  deleted_at
ALTER TABLE users ADD COLUMN deleted_at DATETIME NULL;

--
UPDATE users
SET deleted_at = datetime('now'), is_active = false
WHERE user_id = 1;

--
```

```
SELECT * FROM users WHERE deleted_at IS NULL;
```

DML

```
flowchart TD
    A[DML ] --> B{   ?}
    B -->| | C[BEGIN TRANSACTION]
    B -->| | D[  ]
    C --> E[ DML ]
    E --> F{  }
    F -->| | G[COMMIT]
    F -->| | H[ROLLBACK]
    D --> I[  ]
    G --> I
    H --> I
```

1.
2. UPDATE DELETE
3.
4.

---

4 (TCL) -

TCL

4.1 3.1 ACID

```
mindmap
  root((ACID))
    Atomicity



    Consistency



    Isolation



    Durability
```

4.2 3.2

4.2.1

```
-- 1.
BEGIN TRANSACTION;  --

--
INSERT INTO users (name, email, birth_date, gender)
VALUES ('  ', 'newuser@email.com', '1995-01-01', 'male');

--      ID SQLite
```

```
--         last_insert_rowid()

--
INSERT INTO user_progress (user_id, lesson_id, completed_at)
SELECT last_insert_rowid(), lesson_id, NULL
FROM lessons WHERE course_id = 1;

--
--
COMMIT;  --

--
-- ROLLBACK;  --
```

4.2.2 Savepoint -

```
BEGIN TRANSACTION;

--
SAVEPOINT user_creation;

INSERT INTO users (name, email, birth_date, gender)
VALUES ('  1', 'test1@email.com', '1990-01-01', 'male');

INSERT INTO users (name, email, birth_date, gender)
VALUES ('  2', 'test2@email.com', '1991-01-01', 'female');

--
SAVEPOINT course_enrollment;

--
INSERT INTO enrollments (user_id, course_id, enrollment_date, amount_paid)
VALUES (999, 1, datetime('now'), 199.00);  --  ID

--
ROLLBACK TO SAVEPOINT course_enrollment;

--
RELEASE SAVEPOINT course_enrollment;

COMMIT;
```

4.3 3.3

```
--
-- SQLite

-- 1.    (Read Uncommitted)
PRAGMA read_uncommitted = true;

-- 2.    (Read Committed) -
--

-- 3.    (Repeatable Read)
--

-- 4.    (Serializable) - SQLite
--
```

4.3.1

```
--

--  1
BEGIN TRANSACTION;
SELECT price FROM courses WHERE course_id = 1;  --    199.00
--      2

--  2
BEGIN TRANSACTION;
UPDATE courses SET price = 299.00 WHERE course_id = 1;
COMMIT;

--   1
SELECT price FROM courses WHERE course_id = 1;  --    299.00
COMMIT;
```

4.4 3.4

4.4.1 1

```
BEGIN TRANSACTION;

-- 1.
INSERT INTO users (name, email, birth_date, gender)
VALUES (' ', 'liming@email.com', '1992-06-15', 'male');

-- 2.
INSERT INTO user_activity_log (user_id, activity_type, created_at)
VALUES (last_insert_rowid(), 'registration', datetime('now'));

-- 3.
INSERT INTO enrollments (user_id, course_id, enrollment_date, amount_paid, status)
VALUES (last_insert_rowid(), 1, datetime('now'), 0.00, 'active');

COMMIT;
```

4.4.2 2

```
--
BEGIN TRANSACTION;

SAVEPOINT before_purchase;

-- 1.
-- 2.
-- 3.
INSERT INTO enrollments (user_id, course_id, enrollment_date, amount_paid, status)
VALUES (1, 2, datetime('now'), 299.00, 'active');

-- 4.
UPDATE users
SET updated_at = datetime('now')
WHERE user_id = 1;

-- 5.
-- INSERT INTO payment_log (user_id, amount, payment_method, transaction_id)
-- VALUES (1, 299.00, 'credit_card', 'TXN123456789');
```

```
--
COMMIT;

--
-- ROLLBACK TO SAVEPOINT before_purchase;
```

TCL

```
flowchart LR
    A[   ] --> B[BEGIN TRANSACTION]
    B --> C[   ]
    C --> D{  ?}
    D -->| | E[COMMIT]
    D -->| | F[ROLLBACK]
    E --> G[   ]
    F --> H[   ]
    G --> I[ ]
    H --> I
```

1.
2.
3.
4.

---

5 (DQL) - SELECT

SELECT

5.1 4.1 SQL -

```
flowchart TD
    A[FROM  ] --> B[JOIN  ]
    B --> C[WHERE  ]
    C --> D[GROUP BY  ]
    D --> E[HAVING  ]
    E --> F[SELECT  ]
    F --> G[DISTINCT  ]
    G --> H[ORDER BY  ]
    H --> I[LIMIT/OFFSET  ]

    style A fill:#f9f,stroke:#333,stroke-width:2px
    style F fill:#bbf,stroke:#333,stroke-width:2px
```

```
--
INSERT INTO lessons (course_id, title, content, duration_minutes, sequence_number) VALUES
(1, 'Python  ', 'Python    ', 45, 1),
(1, '    ', 'Python    ', 38, 2),
(1, '  ', '    ', 52, 3),
(1, '   ', '     ', 41, 4),
(2, '    ', 'pandas    ', 35, 1),
(2, '  ', '     ', 48, 2),
(2, '   ', 'matplotlib seaborn ', 55, 3),
(3, 'Figma ', 'Figma    ', 30, 1),
(3, '  ', '     ', 42, 2);
```

5.2 4.2 FROM -

5.2.1

```
-- 1.
SELECT * FROM users;

-- 2.
SELECT u.name, u.email, u.created_at
FROM users u
WHERE u.is_active = true;

-- 3.
SELECT course_stats.*
FROM (
    SELECT
        course_id,
        COUNT(*) as enrollment_count,
        AVG(amount_paid) as avg_paid
    FROM enrollments
    WHERE status = 'active'
    GROUP BY course_id
) course_stats
WHERE course_stats.enrollment_count > 1;
```

5.3 4.3 JOIN -

5.3.1 JOIN

```
graph LR
    A[LEFT TABLE]
    B[RIGHT TABLE]

    subgraph "INNER JOIN"
        C[    ]
    end

    subgraph "LEFT JOIN"
        D[    <br/>   ]
    end

    subgraph "RIGHT JOIN"
        E[    <br/>   ]
    end

    subgraph "FULL OUTER JOIN"
        F[    ]
    end
```

```
-- 1. INNER JOIN -
SELECT
    u.name as student_name,
    c.title as course_title,
    e.enrollment_date,
    e.amount_paid
FROM users u
INNER JOIN enrollments e ON u.user_id = e.user_id
INNER JOIN courses c ON e.course_id = c.course_id
WHERE u.is_active = true;

-- 2. LEFT JOIN -
```

```sql
SELECT
    u.name,
    u.email,
    COUNT(e.course_id) as course_count,
    COALESCE(SUM(e.amount_paid), 0) as total_spent
FROM users u
LEFT JOIN enrollments e ON u.user_id = e.user_id
GROUP BY u.user_id, u.name, u.email;

-- 3.
SELECT
    u.name as student_name,
    c.title as course_title,
    cat.name as category_name,
    COUNT(up.progress_id) as lessons_completed,
    COUNT(l.lesson_id) as total_lessons,
    ROUND(
        COUNT(up.progress_id) * 100.0 / COUNT(l.lesson_id),
        2
    ) as completion_percentage
FROM users u
JOIN enrollments e ON u.user_id = e.user_id
JOIN courses c ON e.course_id = c.course_id
JOIN categories cat ON c.category_id = cat.category_id
LEFT JOIN lessons l ON c.course_id = l.course_id
LEFT JOIN user_progress up ON u.user_id = up.user_id
    AND l.lesson_id = up.lesson_id
WHERE e.status = 'active'
GROUP BY u.user_id, c.course_id, u.name, c.title, cat.name
ORDER BY completion_percentage DESC;
```

5.3.2 -

```sql
--
SELECT DISTINCT
    u1.name as current_user,
    u2.name as classmate,
    c.title as course_title
FROM users u1
JOIN enrollments e1 ON u1.user_id = e1.user_id
JOIN enrollments e2 ON e1.course_id = e2.course_id
JOIN users u2 ON e2.user_id = u2.user_id
JOIN courses c ON e1.course_id = c.course_id
WHERE u1.user_id != u2.user_id  --
    AND u1.name = ' '
ORDER BY c.title, u2.name;
```

5.4 4.4 WHERE -

5.4.1

```sql
-- 1.
SELECT * FROM courses
WHERE price > 200
    AND difficulty_level = 'intermediate'
    AND is_published = true;

-- 2.
SELECT * FROM users
```

```sql
WHERE birth_date BETWEEN '1990-01-01' AND '1999-12-31'
    AND created_at >= datetime('now', '-30 days');

-- 3.
SELECT * FROM courses
WHERE title LIKE '%Python%'  --  Python
    OR title LIKE ' %';      --  " "

-- 4.
SELECT * FROM courses
WHERE category_id IN (1, 2)  --
    AND difficulty_level NOT IN ('advanced');
```

5.4.2

```sql
-- 1. NULL
SELECT * FROM users
WHERE birth_date IS NOT NULL
    AND (phone IS NULL OR length(phone) < 11);

-- 2.
SELECT * FROM users
WHERE email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$';

-- 3.
SELECT * FROM enrollments
WHERE date(enrollment_date) = date('now')  --
    OR strftime('%w', enrollment_date) = '0';  --

-- 4.
SELECT * FROM users
WHERE user_id IN (
    SELECT DISTINCT user_id
    FROM enrollments
    WHERE amount_paid > 200
);
```

5.5 4.5 GROUP BY -

5.5.1

```sql
-- 1.
SELECT
    c.title,
    COUNT(*) as enrollment_count,
    AVG(e.amount_paid) as avg_payment,
    MIN(e.amount_paid) as min_payment,
    MAX(e.amount_paid) as max_payment,
    SUM(e.amount_paid) as total_revenue
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.title
ORDER BY enrollment_count DESC;

-- 2.
SELECT
    strftime('%Y-%m', enrollment_date) as month,
    COUNT(*) as enrollments,
    SUM(amount_paid) as monthly_revenue,
```

```
    COUNT(DISTINCT user_id) as unique_students
FROM enrollments
GROUP BY strftime('%Y-%m', enrollment_date)
ORDER BY month;
```

5.5.2 HAVING -

```
--         > 2
SELECT
    c.title,
    COUNT(*) as enrollment_count,
    AVG(e.amount_paid) as avg_payment
FROM courses c
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.title
HAVING COUNT(*) > 1  --       enrollment_count
    AND AVG(e.amount_paid) > 150
ORDER BY enrollment_count DESC;

--   HAVING
SELECT
    cat.name as category,
    COUNT(DISTINCT c.course_id) as course_count,
    COUNT(DISTINCT e.user_id) as student_count,
    SUM(e.amount_paid) as category_revenue
FROM categories cat
JOIN courses c ON cat.category_id = c.category_id
JOIN enrollments e ON c.course_id = e.course_id
GROUP BY cat.category_id, cat.name
HAVING course_count >= 1
    AND category_revenue > 200
    AND student_count > 1;
```

5.6 4.6

5.6.1 -

```
-- 1.
SELECT
    u.name,
    c.title,
    e.amount_paid,
    ROW_NUMBER() OVER (ORDER BY e.amount_paid DESC) as payment_rank,
    RANK() OVER (ORDER BY e.amount_paid DESC) as payment_rank_with_ties,
    DENSE_RANK() OVER (ORDER BY e.amount_paid DESC) as dense_payment_rank
FROM users u
JOIN enrollments e ON u.user_id = e.user_id
JOIN courses c ON e.course_id = c.course_id;

-- 2.
SELECT
    u.name,
    c.title,
    cat.name as category,
    e.amount_paid,
    AVG(e.amount_paid) OVER (PARTITION BY cat.category_id) as category_avg_payment,
    e.amount_paid - AVG(e.amount_paid) OVER (PARTITION BY cat.category_id) as payment_diff_from_avg
FROM users u
JOIN enrollments e ON u.user_id = e.user_id
```

```
JOIN courses c ON e.course_id = c.course_id
JOIN categories cat ON c.category_id = cat.category_id;

-- 3.
SELECT
    enrollment_date,
    amount_paid,
    SUM(amount_paid) OVER (ORDER BY enrollment_date) as running_total,
    LAG(amount_paid, 1) OVER (ORDER BY enrollment_date) as previous_payment,
    LEAD(amount_paid, 1) OVER (ORDER BY enrollment_date) as next_payment
FROM enrollments
ORDER BY enrollment_date;
```

5.6.2 (CTE)

```
-- 1.  CTE
WITH course_stats AS (
    SELECT
        c.course_id,
        c.title,
        COUNT(e.enrollment_id) as enrollment_count,
        AVG(e.amount_paid) as avg_payment
    FROM courses c
    LEFT JOIN enrollments e ON c.course_id = e.course_id
    GROUP BY c.course_id, c.title
),
popular_courses AS (
    SELECT * FROM course_stats
    WHERE enrollment_count >= 2
)
SELECT
    title,
    enrollment_count,
    ROUND(avg_payment, 2) as avg_payment
FROM popular_courses
ORDER BY enrollment_count DESC;

-- 2.  CTE
WITH RECURSIVE course_recommendations AS (
    --
    SELECT
        u.user_id,
        u.name,
        c.course_id,
        c.title,
        1 as level
    FROM users u
    JOIN enrollments e ON u.user_id = e.user_id
    JOIN courses c ON e.course_id = c.course_id
    WHERE u.name = ' '

    UNION ALL

    --
    SELECT
        cr.user_id,
        cr.name,
        c2.course_id,
        c2.title,
```

```
        cr.level + 1
    FROM course_recommendations cr
    JOIN courses c1 ON cr.course_id = c1.course_id
    JOIN courses c2 ON c1.category_id = c2.category_id
    WHERE cr.level < 2
        AND c2.course_id NOT IN (
            SELECT course_id FROM course_recommendations
        )
)
SELECT DISTINCT name, title, level
FROM course_recommendations
ORDER BY level, title;
```

5.6.3

```
-- 1.
SELECT
    u.name,
    u.email,
    (SELECT COUNT(*) FROM enrollments e WHERE e.user_id = u.user_id) as course_count,
    (SELECT MAX(amount_paid) FROM enrollments e WHERE e.user_id = u.user_id) as max_payment
FROM users u;

-- 2.
SELECT u.name, u.email
FROM users u
WHERE EXISTS (
    SELECT 1 FROM enrollments e
    WHERE e.user_id = u.user_id
        AND e.amount_paid > 200
);

-- 3.
SELECT
    c.title,
    c.price,
    (
        SELECT COUNT(*)
        FROM enrollments e
        WHERE e.course_id = c.course_id
    ) as enrollment_count,
    CASE
        WHEN (SELECT COUNT(*) FROM enrollments e WHERE e.course_id = c.course_id) > 2
        THEN '  '
        WHEN (SELECT COUNT(*) FROM enrollments e WHERE e.course_id = c.course_id) > 0
        THEN '  '
        ELSE '  '
    END as popularity
FROM courses c
WHERE c.is_published = true;
```

5.7 4.7

```
-- 1. UNION -
SELECT name as person_name, 'Student' as role FROM users
WHERE user_id IN (SELECT DISTINCT user_id FROM enrollments)
UNION
SELECT title as person_name, 'Course' as role FROM courses
WHERE is_published = true;
```

```sql
-- 2. INTERSECT -
SELECT user_id FROM enrollments WHERE course_id = 1
INTERSECT
SELECT user_id FROM enrollments WHERE course_id = 2;

-- 3. EXCEPT -
SELECT user_id FROM users WHERE is_active = true
EXCEPT
SELECT DISTINCT user_id FROM enrollments;
```

5.8 4.8 ORDER BY LIMIT -

```sql
-- 1.
SELECT
    u.name,
    c.title,
    e.amount_paid,
    e.enrollment_date
FROM users u
JOIN enrollments e ON u.user_id = e.user_id
JOIN courses c ON e.course_id = c.course_id
ORDER BY
    e.amount_paid DESC,  --
    e.enrollment_date ASC,  --
    u.name;  --

-- 2.
SELECT
    u.name,
    u.birth_date,
    CASE
        WHEN date('now') - birth_date > 365.25 * 30 THEN '30+'
        WHEN date('now') - birth_date > 365.25 * 25 THEN '25-30'
        ELSE '25 '
    END as age_group
FROM users u
ORDER BY
    CASE
        WHEN date('now') - birth_date > 365.25 * 30 THEN 1
        WHEN date('now') - birth_date > 365.25 * 25 THEN 2
        ELSE 3
    END,
    u.name;

-- 3.
SELECT
    u.name,
    c.title,
    e.amount_paid
FROM users u
JOIN enrollments e ON u.user_id = e.user_id
JOIN courses c ON e.course_id = c.course_id
ORDER BY e.enrollment_date DESC
LIMIT 10 OFFSET 0;  --    10

--
WITH numbered_results AS (
    SELECT
```

```
        u.name,
        c.title,
        e.amount_paid,
        ROW_NUMBER() OVER (ORDER BY e.enrollment_date DESC) as row_num
    FROM users u
    JOIN enrollments e ON u.user_id = e.user_id
    JOIN courses c ON e.course_id = c.course_id
)
SELECT name, title, amount_paid
FROM numbered_results
WHERE row_num BETWEEN 11 AND 20;  --
```

5.9

5.9.1

```
--
EXPLAIN QUERY PLAN
SELECT
    u.name,
    COUNT(e.course_id) as course_count
FROM users u
LEFT JOIN enrollments e ON u.user_id = e.user_id
WHERE u.is_active = true
GROUP BY u.user_id, u.name;
```

5.9.2

```
flowchart TD
    A[  ] --> B[  ]
    A --> C[  ]
    A --> D[   ]

    B --> E[  ]
    B --> F[  ]
    B --> G[  ]

    C --> H[ SELECT *]
    C --> I[    ]
    C --> J[ JOIN ]

    D --> K[ LIMIT]
    D --> L[  ]
    D --> M[  ]
```

SELECT

1. FROM→WHERE→GROUP BY→HAVING→SELECT→ORDER BY
2. WHERE JOIN
3. **N+1** JOIN
4. LIMIT
5. INNER JOIN LEFT JOIN
6.

6

SQL SQL

SQL

SQL SQL