Foreign Keys & JOIN Queries

Tutorial 2

Foreign keys

Foreign key declaration

A constraint put on a column in one table to reference the primary key of another table. Establishes a referential integrity constraint between two tables.

Key points:

- Ensures data consistency
- Creates parent-child relationship
- Prevents orphaned records
- Enables joins

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY
AUTO_INCREMENT,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id)
REFERENCES customers(customer_id)
);
```

Reference actions

- `CASCADE`: Propagate changes
- `SET NULL`: Set foreign key to NULL
- `SET DEFAULT`: Set to default value
- `RESTRICT/NO ACTION`: Prevent operation
- `ON DELETE SET DEFAULT`: Set to default on delete

```sql
CREATE TABLE OrderItems (
    item_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    FOREIGN KEY (order_id)
    REFERENCES Orders(order_id)
        ON DELETE CASCADE,
    FOREIGN KEY (product_id)
    REFERENCES Products(product_id)
        ON DELETE SET NULL
);
```

JOIN Queries

INNER JOIN

Returns only matching rows from both tables

```
-- Basic syntax
SELECT columns
FROM table1
INNER JOIN table2 ON table1.column =
table2.column;

SELECT s.name, s.student_id, d.dept_name
FROM Students s
INNER JOIN Departments d ON
s.major_dept_id = d.dept_id;
```

LEFT JOIN

Returns all rows from left table, matching rows from right, NULL if no match

```
-- All students, even those without majors
SELECT s.name, d.dept_name
FROM Students s
LEFT JOIN Departments d ON s.major_dept_id
= d.dept_id;

-- Result includes:
-- "Alice, Computer Science"
-- "Bob, NULL" (undeclared major)
```

RIGHT JOIN

Returns all rows from right table, matching rows from left, NULL if no match

```
-- All departments, even those with no
students
SELECT d.dept_name, s.name
FROM Students s
RIGHT JOIN Departments d ON
s.major_dept_id = d.dept_id;

-- Result includes:
-- "Computer Science, Alice"
-- "Astronomy, NULL" (department exists
but no majors)
```

FULL OUTER JOIN

Since MySQL doesn't support FULL OUTER JOIN directly, we can simulate it using UNION of LEFT and RIGHT JOIN.

```
-- All students AND all departments
SELECT s.name, d.dept_name
FROM Students s
LEFT JOIN Departments d ON
s.major_dept_id = d.dept_id
UNION
SELECT s.name, d.dept_name
FROM Students s
RIGHT JOIN Departments d ON
s.major_dept_id = d.dept_id;
```

```
-- Result includes:
-- "Alice, Computer Science"
-- "NULL, Astronomy" (department
exists but no majors)
-- "Bob, NULL" (undeclared major)
```

ANTI-JOIN – Subtraction operation

Since MySQL doesn't have a native `MINUS` or `EXCEPT` operator, we use `LEFT JOIN` with `WHERE ... IS NULL`

```
-- Pattern: Find rows in A that don't exist in B
SELECT A.*
FROM TableA A
LEFT JOIN TableB B ON A.key = B.key
WHERE B.key IS NULL;
```

Example:

```
-- Courses that no one is taking
SELECT c.course_id, c.title
FROM Course c
LEFT JOIN Takes t ON c.course_id = t.course_id
WHERE t.course_id IS NULL;
```

Multi-Table JOIN example

Finding all courses taken by Computer Science majors

```
SELECT s.name, c.title, g.grade
FROM Students s
INNER JOIN Departments d
    ON s.major_dept_id = d.dept_id
INNER JOIN Courses c
    ON c.offered_by = d.dept_id
INNER JOIN Grades g
    ON g.student_id = s.student_id
    AND g.course_id = c.course_id
WHERE d.dept_name = 'Computer Science';
```

Summary

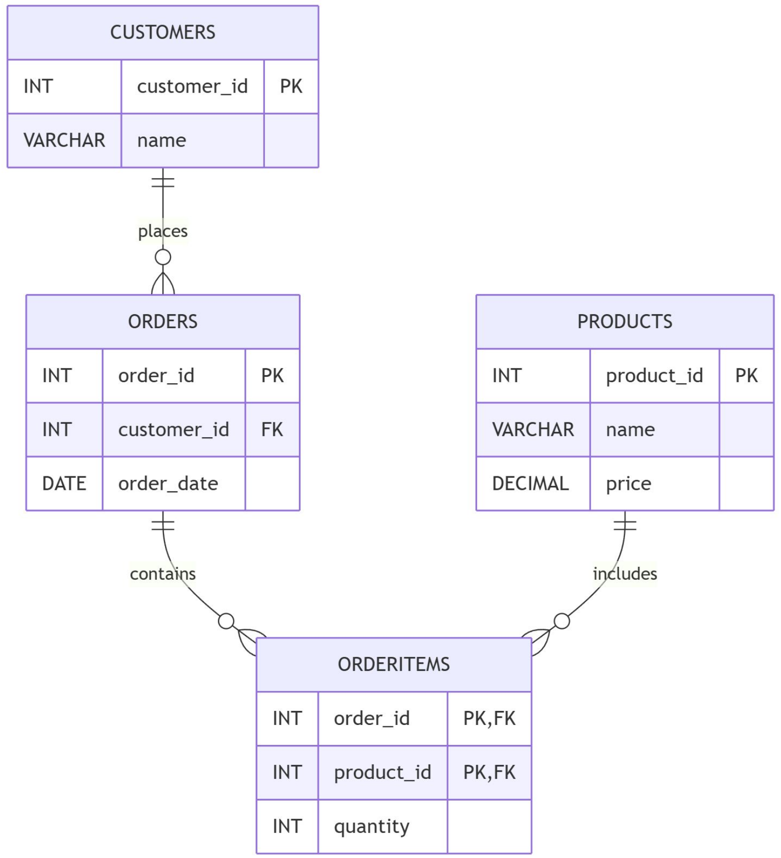
```
INNER JOIN:     [A ∩ B]          (Overlap only)
LEFT JOIN:      [A] + overlap    (All A, matching B)
RIGHT JOIN:     [B] + overlap    (All B, matching A)
FULL JOIN:      [A ∪ B]           (Everything)


MySQL JOIN Simulation:
- INNER JOIN: Standard
- LEFT JOIN: Standard
- RIGHT JOIN: Standard
- FULL JOIN: LEFT JOIN UNION RIGHT JOIN
- ANTI-JOIN: LEFT JOIN WHERE right.key IS NULL
```

Example

Foreign key referential actions:

- When a customer is deleted, all related orders are automatically deleted
- When an order is removed, all order items are automatically removed
- When a product is being removed, prevent removal if there are associated order items

Sample data to fill the tables



Example - Cont.

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES
customers(customer_id)
        ON DELETE CASCADE
);
```

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(10,2)
);
CREATE TABLE OrderItems (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (order_id) REFERENCES
orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES
products(product_id) ON DELETE RESTRICT
);
```

Example - Cont.

```sql
-- Find all customers who bought 'Laptop', with their purchase
details
SELECT c.name, o.order_date, p.name as product_name, oi.quantity
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
INNER JOIN OrderItems oi ON o.order_id = oi.order_id
INNER JOIN Products p ON oi.product_id = p.product_id
WHERE p.name = 'Laptop'
ORDER BY o.order_date DESC;
```

Query Optimization

Finding all orders placed by customer Alice Chen (customer_id = 1) on or after March 17, 2024.

```
SELECT *
FROM (
    SELECT * FROM orders
    WHERE order_date >= '2024-03-17'
) t
WHERE customer_id = 1;


SELECT *
FROM orders
WHERE order_date >= '2024-03-17'
  AND customer_id = 1;
```

Project operation

Projection is an operation that selects specific columns from a table, without changing the rows.

It controls which attributes (columns) appear in the result.

Example:

Selecting the names and prices of all products

```sql
SELECT name, price
FROM products;
```

Cartesian product

The Cartesian product combines every row of one table with every row of another table.

If table A has m rows and table B has n rows, the result has m × n rows.

Note: Cartesian products are rarely used alone in real systems because they produce very large result sets

Example:

Listing all possible combinations of customers and products

```
SELECT c.name AS customer_name,
    p.name AS product_name
FROM customers c, products p;
```

Lab Assignment

Task 1: SELECT Query optimization

Display the names and prices of all products that cost more than $100 and name starting with “L”

1. Write and execute a nested query
2. Write and execute optimized query
3. Compare the results and execution time

Task 2: Project operation

Show the customer names and order dates for all orders

1. Write an sql query for the operation
2. Take screenshot of execution results

Task 3: Cartesian product

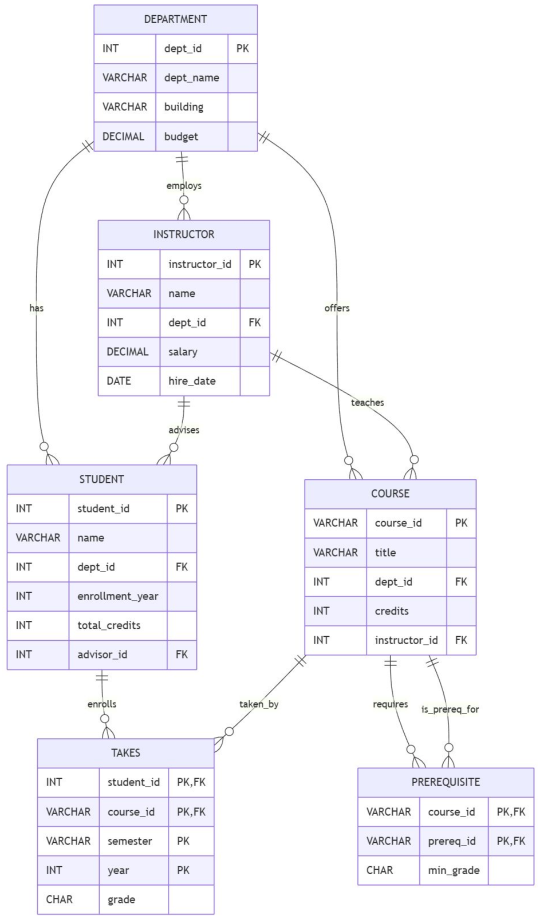
Show all combinations of orders and products, regardless of whether the product appears in the order.

1. Write an sql query for the operation
2. Take screenshot of execution results

Task 4: Table initialization

When a department is removed:

- All courses offered by that department are automatically removed (CASCADE from Course.dept_id)
- Instructors in that department have their department set to NULL (SET NULL from Instructor.dept_id)
- Students majoring in that department have their department set to NULL (SET NULL from Student.dept_id)



Task 4: Table initialization

When an instructor is removed:

- All courses taught by that instructor have their instructor set to NULL (SET NULL from Course.instructor_id)
- All students advised by that instructor have their advisor set to NULL (SET NULL from Student.advisor_id)

When a student is removed:

- All course registrations (Takes records) for that student are automatically removed (CASCADE from Takes.student_id)

Task 4: Table initialization

When a course is being removed:

- Prevent removal if there are students currently taking the course (RESTRICT from Takes.course_id)
- Prevent removal if it's a prerequisite for other courses (RESTRICT from Prerequisite.prereq_id)
- All prerequisite relationships where this course is the main course are automatically removed (CASCADE from Prerequisite.course_id)

Sample data to fill the tables

Task 5: JOIN Queries

1. Find all students and their assigned advisors using INNER JOIN
2. Find all students with or without advisors using LEFT JOIN
3. Find all instructors and any students they advise using RIGHT JOIN
4. Find all students and all instructors, matching where advisor relationships exist, using UNION (full outer join)
5. Find students, their courses, and departments
6. Find students, their courses, instructors, and grades

ASSIGNMENT DETAILS

Instructions for submission

- Deadline
  - Hard deadline is: **18:00, 10.02.2025 (1 week)**
  - No late submissions are accepted!

- Submission format
  - Submit via gradescope.com

Evaluation criteria

- Task 1: 10%
- Task 2: 10%
- Task 3: 10%
- Task 4: 35%
- Task 5: 35%

Notes

- You must use the following data in assignment
  - Data for tasks 1, 2, 3
  - Data for tasks 4, 5
- Your assignment submission must include
  - SQL query for each task
  - Screenshot of output

Q&A