Executive Summary:

This project presents the implementation of a student academic information system for the University of Toronto, developed as Phase II of a multi-stage database design initiative. Building on the conceptual and enhanced ER models created in Phase I, this phase transitions the design into a fully implemented MySQL relational database that supports critical business processes: student admissions, teaching staff assignment, and course enrollment. The resulting system integrates data across applicants, students, academic programs, course offerings, and teaching responsibilities, enabling robust query capabilities and reliable reporting for operational decision-making.

At the core of the system is a normalized relational schema comprising more than twenty interrelated entities. These include support for complex modeling features such as weak entities (e.g., supporting_document, tutorial), composite keys (e.g., section, enrollment), and hierarchically structured tables for bachelor, master, and PhD program subtypes. Primary keys and foreign key relationships have been carefully defined to preserve data integrity, and ENUMs and CHECK constraints enforce controlled value sets. Advanced business logic—such as prerequisites for course enrollment, academic level-specific enrollment rules, and teaching role categorization—was addressed through schema design and supplementary test logic.

The schema was implemented in MySQL, with comprehensive data population scripts developed to simulate a realistic academic environment. A broad set of SQL test cases was created to validate relational integrity constraints, including primary key uniqueness, foreign key enforcement, ENUM validations, NULL constraints, and cascading deletions. Additional queries were written to assess complex, non-schema-enforceable constraints, such as prerequisite fulfillment across multiple course instances, repeat course enrollment logic based on performance assumptions, and student-course eligibility across programs and years of study.

To support institutional decision-making, three key analytical reports were developed:

1. **Admissions Trends**: Assesses conversion rates from application to offer, analyzes applicant decisions, and identifies anomalies in the admissions pipeline.

2. **Teaching Staff Workload**: Presents a unified view of faculty responsibilities across lectures and tutorials, enabling academic administrators to evaluate teaching distribution by semester and position.

3. **Enrollment Trends**: Analyzes enrolled credit load by faculty and academic year for an individual student, highlighting academic progression and cross-departmental course distribution.

All reporting logic was implemented using SQL SELECT queries and visualized via Excel tables and charts. Report outputs include percentage breakdowns, pivot tables, and CTE-based aggregations to support nuanced trend analysis. The results highlight the system's ability to generate timely and actionable insights for university staff, while also serving as a demonstration of effective relational modeling in support of academic operations.

This report demonstrates not only the technical implementation of a relational database schema but also the ability to support real-world academic workflows through rigorous testing and reporting. The system was collaboratively developed with clearly delineated responsibilities among team members, and the result is a scalable, maintainable database solution tailored to the needs of a large academic institution.

Table of contents

1.The Relational Model

The relational model implemented in this project was derived directly from the Enhanced ER (EER) diagram created in Phase I. This section outlines the rationale for each key mapping decision, demonstrating how the conceptual design was transformed into a fully normalized, relational schema using MySQL. The ER diagram submitted in Phase I served as the foundation, with adjustments made during implementation to improve scalability, enforce integrity, and accommodate system constraints. The refined ER and relational schema diagrams are included in the appendices for reference.

1.1. Mapping of Regular Entities

All strong entities from the conceptual model—such as applicant, application, student, program, course, and teaching_staff—were mapped directly to relational tables. Each entity was assigned a single-column primary key, using natural or surrogate keys depending on the domain logic:

- applicant_id is auto-incremented for uniqueness.
- student_number and staff_id are stored as fixed-length character keys for institutional consistency.
- program_name and faculty_name use descriptive, unique values as primary keys to reflect real-world identifiers.

1.2. Handling of Subtypes

The program entity was further divided into subtypes (bachelor, master, phd) to represent specialization details unique to each degree type. This was implemented using vertical partitioning:

- A shared primary key program_name links the subtype tables back to the base program table.
- Each subtype table includes degree-specific attributes, such as specialization (for bachelor), master_type, and research_area.
- The use of ENUM('bachelor','master','phd') in the base table enforces valid degree classification at the schema level.

1.3. Mapping of Weak Entities

Several weak entities were mapped using composite primary keys and foreign keys to enforce dependency:

- supporting_document is dependent on application and uses (application_id, document_name) as its primary key.
- section and tutorial depend on course_id, with their own local identifiers (section_code, tutorial_code), forming composite keys.
- These dependencies are enforced through foreign keys and cascading delete behaviors to preserve referential integrity.

1.4. Mapping of Relationships

Relationships between entities were handled based on cardinality and participation:

- **1:N relationships** (e.g., faculty to program, program to course, application to admission_offer) were mapped using foreign keys on the N-side.
- **M:N relationships**, such as student enrollments in multiple courses across terms, were modeled via junction tables:
  - enrollment uses (student_number, course_id) as a composite primary key, with optional fields like tutorial_code.
  - This structure allows tracking of multiple enrollments in the same course across different years/terms (since course_id is unique per instance).

1.5. Normalization and Data Integrity

All relations were normalized to **Third Normal Form (3NF)**. No transitive or partial dependencies exist within the schema. Controlled vocabularies and enumerations (e.g., application_status, position, term, course_level) were enforced using MySQL ENUM types to prevent invalid data entry.

Attributes were chosen based on actual institutional semantics:

- application_status, applicant_decision, degree_type, and employment_type reflect distinct, controlled states.
- Dates, phone numbers, and capacity constraints were defined with appropriate formats and domain types.

1.6. Schema Improvements Over Phase I

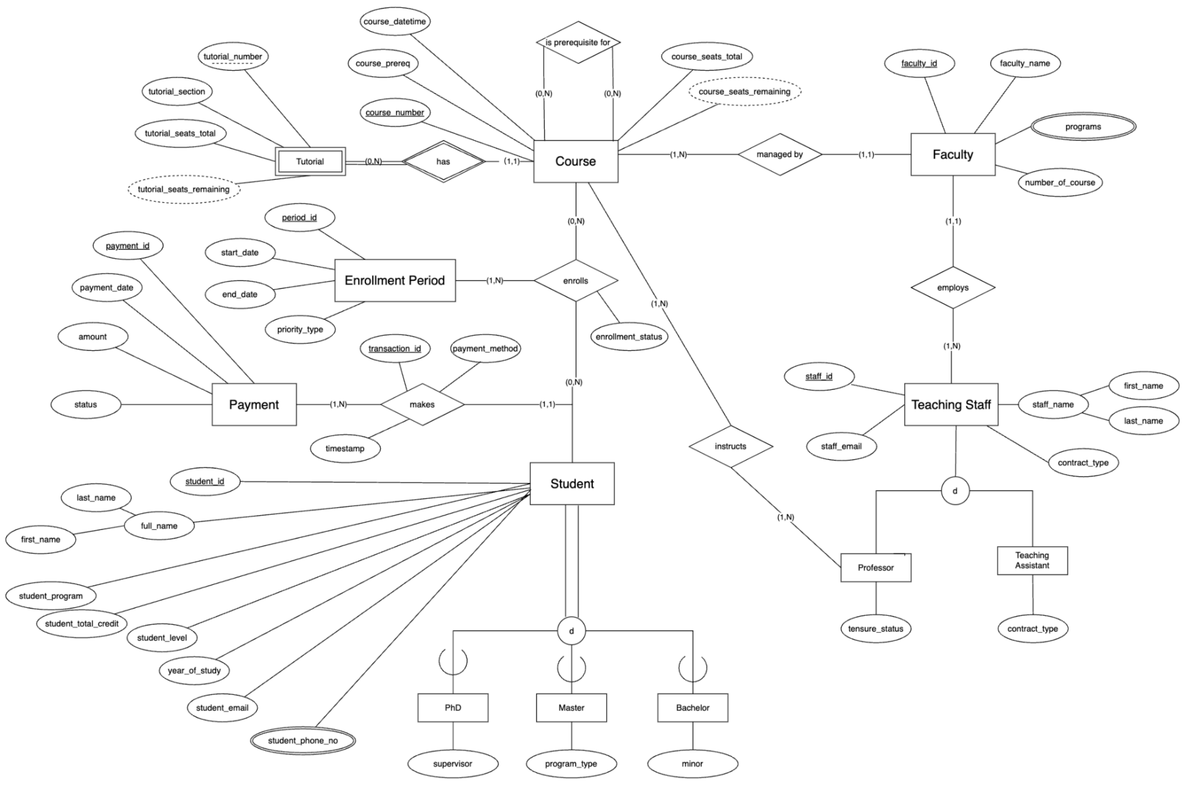
Several enhancements were made to the ER model during Phase II:

- **Enrollment Period and Priority**: Initially planned, this was deferred to reduce scope complexity and improve testability.
- **Prerequisite Handling**: Modeled separately via the prerequisite table using course codes to avoid foreign key limitations between instances of the same course.
- **Teaching Assignments**: Sections and tutorials were unified in reporting logic via SQL, rather than creating a redundant junction table.
- **Support for Repeat Enrollments**: By distinguishing course_id per term, the system allows tracking of multiple course attempts, preserving historical data.

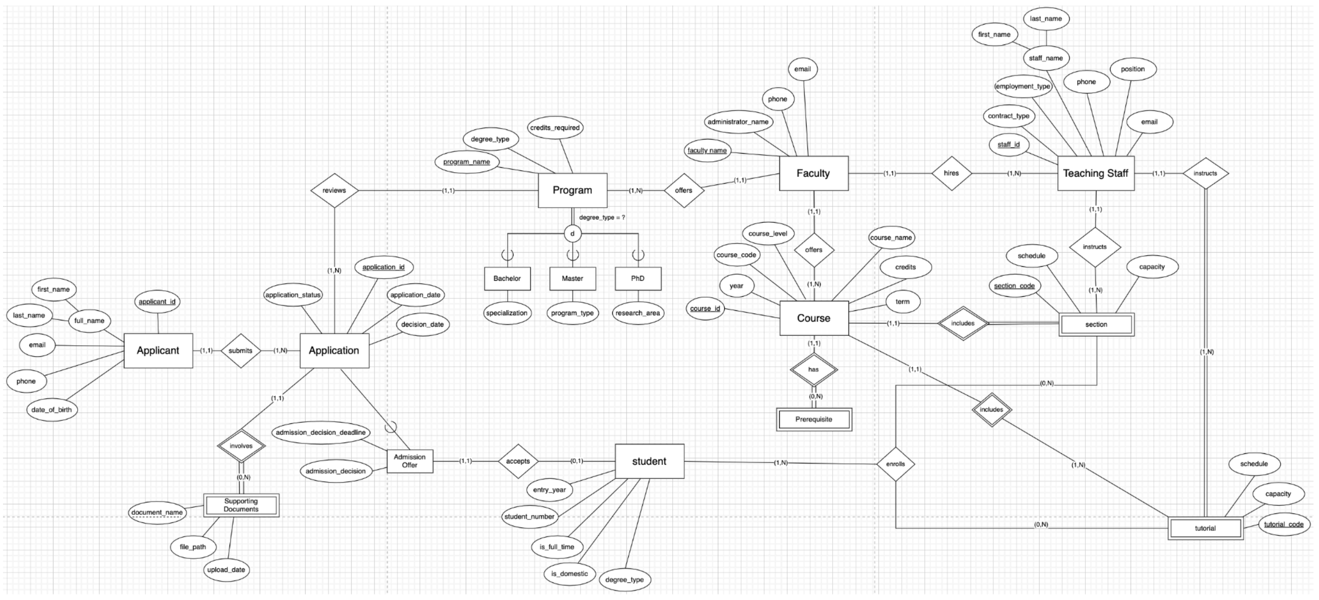
1.7. Diagram Integration

The original EER diagram submitted in Phase I has been updated to reflect new weak entities (tutorial, supporting_document) and clearer mappings for subtypes and relationships. The corresponding relational schema diagram was generated post-implementation to verify mapping accuracy and is included in the Appendix.

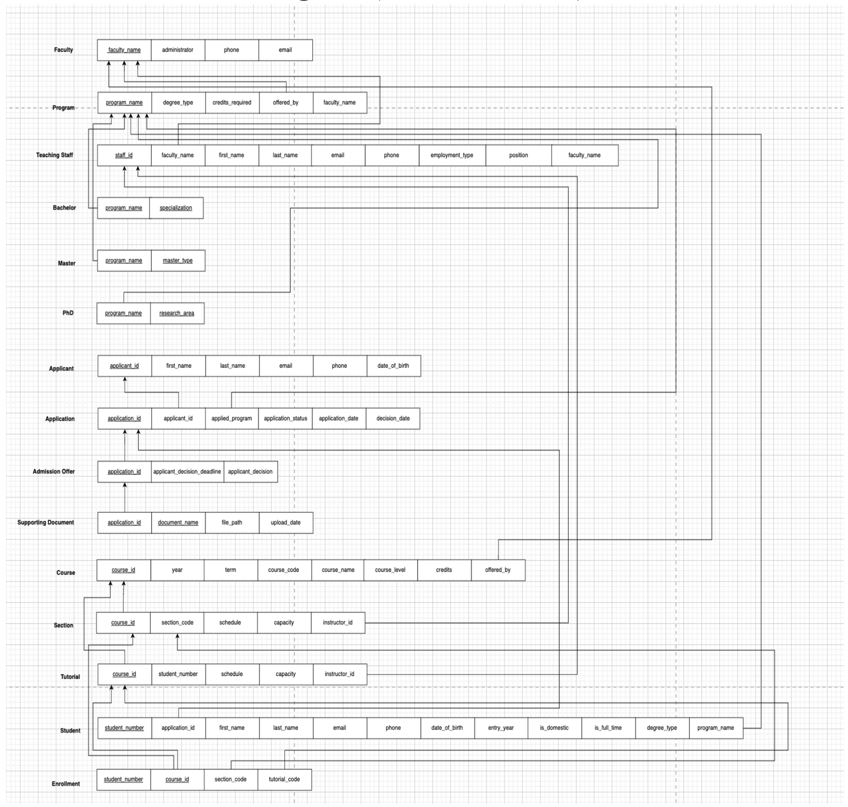
Phase 1 - EER Model (live link to view):



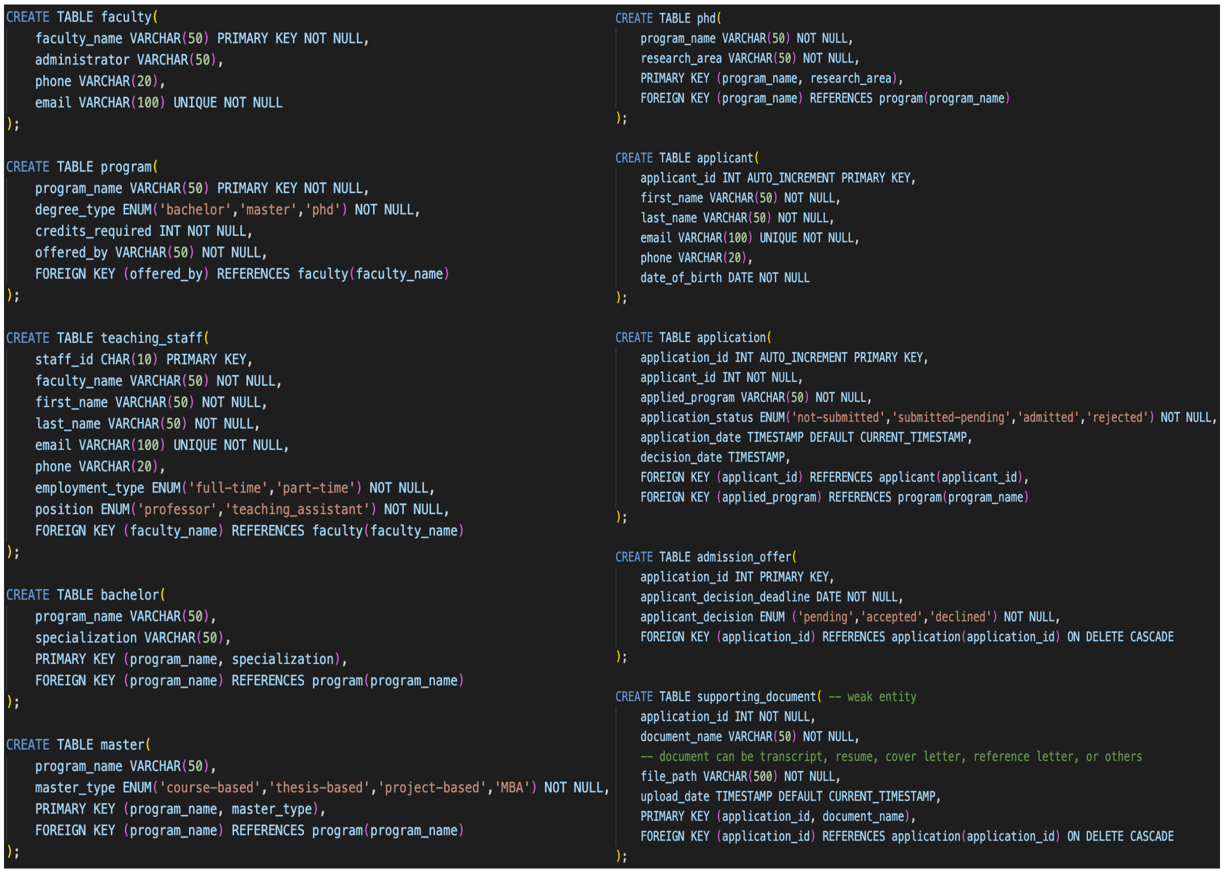
Phase 2 - EER Model (live link to view):



Phase 2 - Relational diagram (live link to view):



All the SQL CREATE TABLE statements (Please refer to script1.sql)



```sql
CREATE TABLE faculty(
    faculty_name VARCHAR(50) PRIMARY KEY NOT NULL,
    administrator VARCHAR(50),
    phone VARCHAR(20),
    email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE program(
    program_name VARCHAR(50) PRIMARY KEY NOT NULL,
    degree_type ENUM('bachelor','master','phd') NOT NULL,
    credits_required INT NOT NULL,
    offered_by VARCHAR(50) NOT NULL,
    FOREIGN KEY (offered_by) REFERENCES faculty(faculty_name)
);

CREATE TABLE teaching_staff(
    staff_id CHAR(10) PRIMARY KEY,
    faculty_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    employment_type ENUM('full-time','part-time') NOT NULL,
    position ENUM('professor','teaching_assistant') NOT NULL,
    FOREIGN KEY (faculty_name) REFERENCES faculty(faculty_name)
);

CREATE TABLE bachelor(
    program_name VARCHAR(50),
    specialization VARCHAR(50),
    PRIMARY KEY (program_name, specialization),
    FOREIGN KEY (program_name) REFERENCES program(program_name)
);

CREATE TABLE master(
    program_name VARCHAR(50),
    master_type ENUM('course-based','thesis-based','project-based','MBA') NOT NULL,
    PRIMARY KEY (program_name, master_type),
    FOREIGN KEY (program_name) REFERENCES program(program_name)
);
```

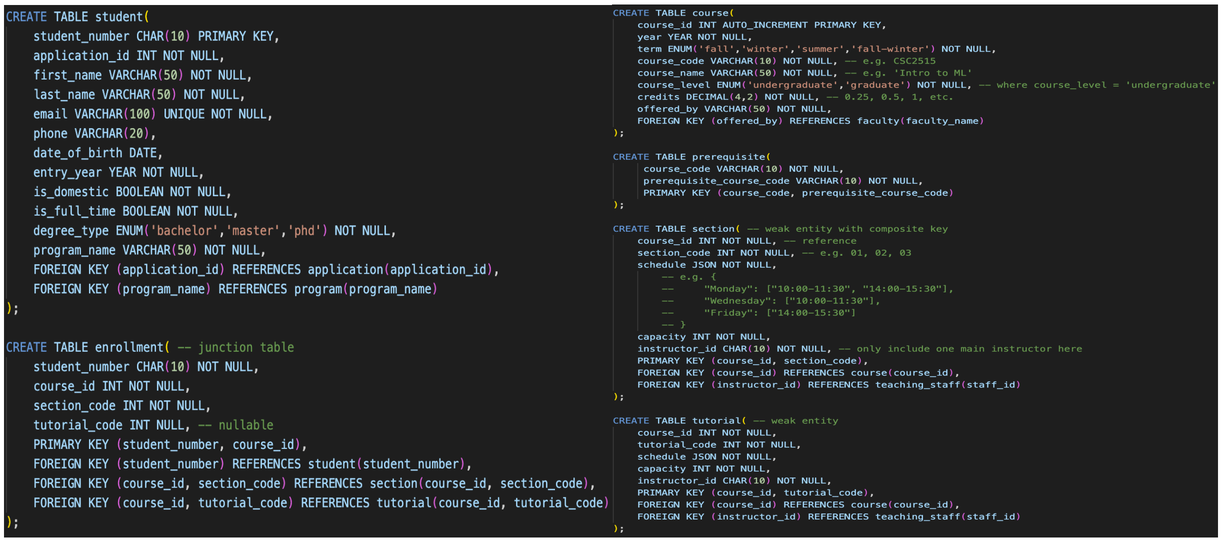
```sql
CREATE TABLE phd(
    program_name VARCHAR(50) NOT NULL,
    research_area VARCHAR(50) NOT NULL,
    PRIMARY KEY (program_name, research_area),
    FOREIGN KEY (program_name) REFERENCES program(program_name)
);

CREATE TABLE applicant(
    applicant_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    date_of_birth DATE NOT NULL
);

CREATE TABLE application(
    application_id INT AUTO_INCREMENT PRIMARY KEY,
    applicant_id INT NOT NULL,
    applied_program VARCHAR(50) NOT NULL,
    application_status ENUM('not-submitted','submitted-pending','admitted','rejected') NOT NULL,
    application_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    decision_date TIMESTAMP,
    FOREIGN KEY (applicant_id) REFERENCES applicant(applicant_id),
    FOREIGN KEY (applied_program) REFERENCES program(program_name)
);

CREATE TABLE admission_offer(
    application_id INT PRIMARY KEY,
    applicant_decision_deadline DATE NOT NULL,
    applicant_decision ENUM ('pending','accepted','declined') NOT NULL,
    FOREIGN KEY (application_id) REFERENCES application(application_id) ON DELETE CASCADE
);

CREATE TABLE supporting_document( -- weak entity
    application_id INT NOT NULL,
    document_name VARCHAR(50) NOT NULL,
    -- document can be transcript, resume, cover letter, reference letter, or others
    file_path VARCHAR(500) NOT NULL,
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (application_id, document_name),
    FOREIGN KEY (application_id) REFERENCES application(application_id) ON DELETE CASCADE
);
```



```sql
CREATE TABLE student(
    student_number CHAR(10) PRIMARY KEY,
    application_id INT NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    date_of_birth DATE,
    entry_year YEAR NOT NULL,
    is_domestic BOOLEAN NOT NULL,
    is_full_time BOOLEAN NOT NULL,
    degree_type ENUM('bachelor','master','phd') NOT NULL,
    program_name VARCHAR(50) NOT NULL,
    FOREIGN KEY (application_id) REFERENCES application(application_id),
    FOREIGN KEY (program_name) REFERENCES program(program_name)
);

CREATE TABLE enrollment( -- junction table
    student_number CHAR(10) NOT NULL,
    course_id INT NOT NULL,
    section_code INT NOT NULL,
    tutorial_code INT NULL, -- nullable
    PRIMARY KEY (student_number, course_id),
    FOREIGN KEY (student_number) REFERENCES student(student_number),
    FOREIGN KEY (course_id, section_code) REFERENCES section(course_id, section_code),
    FOREIGN KEY (course_id, tutorial_code) REFERENCES tutorial(course_id, tutorial_code)
);
```

```sql
CREATE TABLE course(
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    year YEAR NOT NULL,
    term ENUM('fall','winter','summer','fall-winter') NOT NULL,
    course_code VARCHAR(10) NOT NULL, -- e.g. CSC2515
    course_name VARCHAR(50) NOT NULL, -- e.g. 'Intro to ML'
    course_level ENUM('undergraduate','graduate') NOT NULL, -- where course_level = 'undergraduate'
    credits DECIMAL(4,2) NOT NULL, -- 0.25, 0.5, 1, etc.
    offered_by VARCHAR(50) NOT NULL,
    FOREIGN KEY (offered_by) REFERENCES faculty(faculty_name)
);

CREATE TABLE prerequisite(
     course_code VARCHAR(10) NOT NULL,
     prerequisite_course_code VARCHAR(10) NOT NULL,
     PRIMARY KEY (course_code, prerequisite_course_code)
);

CREATE TABLE section( -- weak entity with composite key
    course_id INT NOT NULL, -- reference
    section_code INT NOT NULL, -- e.g. 01, 02, 03
    schedule JSON NOT NULL,
        -- e.g. {
        --     "Monday": ["10:00-11:30", "14:00-15:30"],
        --     "Wednesday": ["10:00-11:30"],
        --     "Friday": ["14:00-15:30"]
        -- }
    capacity INT NOT NULL,
    instructor_id CHAR(10) NOT NULL, -- only include one main instructor here
    PRIMARY KEY (course_id, section_code),
    FOREIGN KEY (course_id) REFERENCES course(course_id),
    FOREIGN KEY (instructor_id) REFERENCES teaching_staff(staff_id)
);

CREATE TABLE tutorial( -- weak entity
    course_id INT NOT NULL,
    tutorial_code INT NOT NULL,
    schedule JSON NOT NULL,
    capacity INT NOT NULL,
    instructor_id CHAR(10) NOT NULL,
    PRIMARY KEY (course_id, tutorial_code),
    FOREIGN KEY (course_id) REFERENCES course(course_id),
    FOREIGN KEY (instructor_id) REFERENCES teaching_staff(staff_id)
);
```

2.Detailed Data Dictionary

This section presents the final data dictionary, which outlines all attributes from the relational model. Each attribute is defined using SQL data types and includes applicable constraints, a description of its purpose, and representative example values. The data dictionary is organized by key domain areas, including Applicant, Application, Admission Offer, Teaching Staff, Course, and Enrollment.

Applicant

Attribute Name	Domain	Meaning	Example Values
Applicant ID	A unique 10-character string	Unique identifier for each applicant	A20240001
Full Name	Text/string	The complete legal name of the applicant	William Chen
Email	Valid email format	Contact email for communication	wchen@mail.utoronto.ca
Date of Birth	Date (YYYY-MM-DD)	Applicant's birthdate	2005-02-12
first_name	String	The first name of the student	Diana
last_name	String	The last name of the student	Malynovska
full_name	Composite of first_name, last_name	The full name of the student	Diana Malynovska
student_phone_number	String (Phone number format)	The contact number of the student	+1-416-555-1234
student_level	Enum	The current education level of the student	Undergraduate, Master, PhD
student_program	String	The academic program of the student	Computer Science, History
year_of_study	Integer	Academic year the student is currently in	1, 2, 3, 4
major	String	The student's major	Computer Science, Biology
minor	String	The student's minor (if applicable)	Statistics, Physics
supervisor	String	Name of the student's supervisor (if applicable)	Dr. John Doe

Application

Attribute Name	Domain	Meaning	Example Values
Application ID	A unique integer	Unique ID for each application record	100101
Application Date	Date	Date the application was submitted	2024-01-15
Application Status	Enum: submitted-pending, admitted, rejected, not-submitted	The current status of the application	admitted
Decision Date	Date	Date the application decision was made	2024-03-01
priority_type	String	The priority level for enrollment	Regular, Early Access

Admission Offer

Attribute Name	Domain	Meaning	Example Values
Applicant Decision	Enum: accepted, declined, pending	Decision made by applicant after receiving offer	accepted

Teaching Staff

Attribute Name	Domain	Meaning	Example Values
Staff ID	A unique alphanumeric string	Unique identifier for each teaching staff member	TS10000001
Position	Enum: professor, teaching_assistant	The academic role of the teaching staff	professor
contract_type	Enum	Type of employment contract	Teaching, Research
professor_id	Integer	Unique identifier for each professor	5001, 5002
professor_email	String	Email address of the professor	john.doe@utoronto.ca
teaching_staff_email	String (Email)	Staff email address	jane.doe@utoronto.ca
tenure_status	Enum	Status of tenure for the staff member	Tenured, Non-Tenured
staff_name	String	Full name of the staff member	Jane Doe, Mark Smith

Course

Attribute Name	Domain	Meaning	Example Values
course_code	String (Alpha-numeric)	Unique identifier for a course	INF1343, CSC2515
course_datetime	List of DateTime	The schedule of the course (day and time)	["Monday 10:00-12:00", "Wednesday 14:00-16:00"]
course_faculty	Text/string	The faculty offering the course	Arts, Science, Information
course_prereq	List of Course Codes	List of prerequisite courses required for enrollment	["CSC100", "CSC121"]
course_seats_remaining	Integer	Derived attribute. Number of seats still available	10, 0
course_seats_total	Integer	Total number of seats available for the course	60, 100
course_section	Integer	Identifies different sections of the same course	1, 2, 2003

Credits	Decimal between 0.0 and 1.0	Credit weight of the course	0.5
Term	Text (e.g., Winter 2024)	Academic term when the course is offered	Winter 2024
Offered By	Text/string	Name of faculty or department offering the course	Faculty of Information

Enrollment

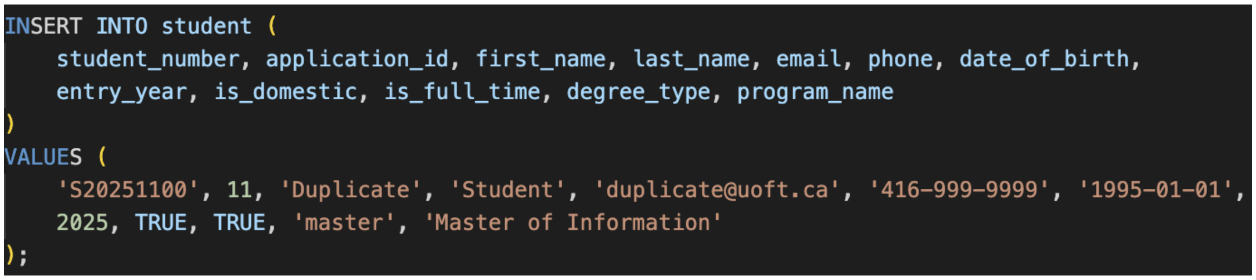
Attribute Name	Domain	Meaning	Example Values
enrollment_id	Auto-increment Integer	Unique identifier for an enrollment record	1001, 1002
enrollment_status	Text/string (enum)	The current status of the enrollment	Registered, Waitlisted, Dropped
Student Number	A unique student number	Identifies a student enrolled in the university	S20240001
enrollment_date	Date	The date when the student enrolled in the course	2025-01-10
Total Credit	Derived (numeric)	Sum of all credits earned by a student in a year	2.0 (see Report 3.1)
Faculty Credit %	Derived (percentage)	Percentage of total credits earned from a specific faculty	66.67 (see Report 3.2)
registration_id	Auto-increment Integer	Unique identifier for course registration	3001, 3002
transaction_id	Auto-increment Integer	Unique identifier for a payment record	7001, 7002
payment_amount	Decimal (2 places)	The amount paid for the course (CAD)	1200.50, 650.00
payment_date	Date	The date the payment was made	2025-01-15
payment_method	String	The method used to make the payment	Bank transfer, Credit card
waitlist_id	Auto-increment Integer	Unique identifier for a waitlist record	2001, 2002
position	Integer	The waitlist position of the student	1, 5
end_date	Date	The end date of the enrollment period	2025-01-20
period_id	Auto-increment Integer	Unique identifier for an enrollment period	4001, 4002
student_total_credits	Integer	Total credits accumulated by the student	0.25, 0.5, 1

3.Test Cases

3.1.1 Testing the SQL Schema Constraints

Test 1: Primary Key Violation

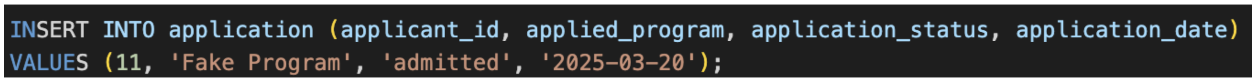
Purpose: To test whether the system prevents inserting duplicate primary keys into the student table.



```
INSERT INTO student (
    student_number, application_id, first_name, last_name, email, phone, date_of_birth,
    entry_year, is_domestic, is_full_time, degree_type, program_name
)
VALUES (
    'S20251100', 11, 'Duplicate', 'Student', 'duplicate@uoft.ca', '416-999-9999', '1995-01-01',
    2025, TRUE, TRUE, 'master', 'Master of Information'
);
```

Test 2: Foreign Key Violation

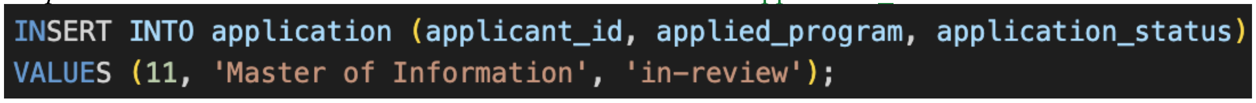
Purpose: To validate that the database rejects entries referencing non-existent foreign keys (applied_program that doesn't exist in the referenced table).



```
INSERT INTO application (applicant_id, applied_program, application_status, application_date)
VALUES (11, 'Fake Program', 'admitted', '2025-03-20');
```

Test 3: ENUM Constraint Violation

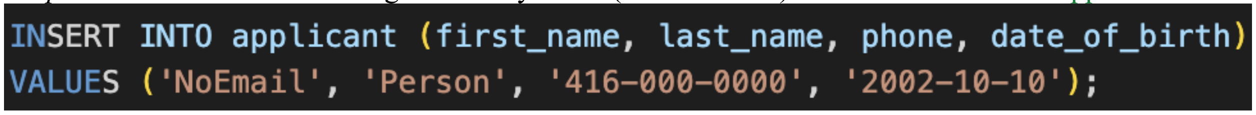
Purpose: To test the enforcement of valid ENUM values in the application_status column.



```
INSERT INTO application (applicant_id, applied_program, application_status)
VALUES (11, 'Master of Information', 'in-review');
```

Test 4: NOT NULL Constraint Violation

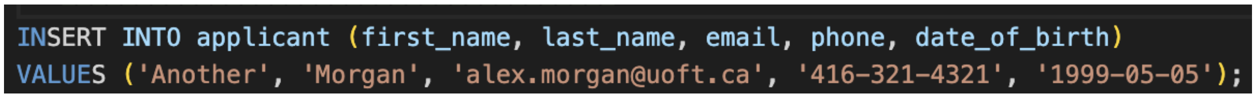
Purpose: To confirm that missing mandatory fields (such as email) are not allowed in the applicant table.



```
INSERT INTO applicant (first_name, last_name, phone, date_of_birth)
VALUES ('NoEmail', 'Person', '416-000-0000', '2002-10-10');
```

Test 5: Unique Constraint Violation

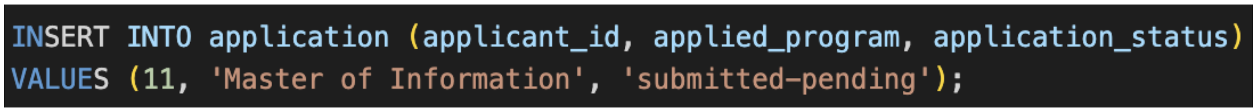
Purpose: To test the enforcement of unique constraints on fields like email in the applicant table.



```
INSERT INTO applicant (first_name, last_name, email, phone, date_of_birth)
VALUES ('Another', 'Morgan', 'alex.morgan@uoft.ca', '416-321-4321', '1999-05-05');
```

Test 6: Valid ENUM Case (Positive Test)

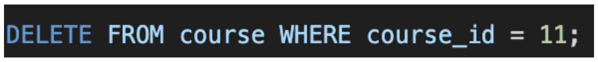
Purpose: To test a successful insertion using a valid ENUM value for application_status.



```
INSERT INTO application (applicant_id, applied_program, application_status)
VALUES (11, 'Master of Information', 'submitted-pending');
```

Test 7: Delete Restricted by Foreign Key

Purpose: To ensure that deletion of a course that is referenced by another entity (such as tutorial) is restricted.



```
DELETE FROM course WHERE course_id = 11;
```

Test 8: CHECK Constraint Violation

Purpose: To test the enforcement of value boundaries, such as preventing negative tutorial capacities.

```
INSERT INTO tutorial (course_id, tutorial_code, schedule, capacity, instructor_id)
VALUES (1, 101, '{"day": "Monday", "time": "10:00 AM"}', -10, 'T12345');
```

Test 9: Nullable Foreign Key
Purpose: To validate that nullable foreign keys (like tutorial_code) are handled correctly during insertions.

```
INSERT INTO enrollment (student_number, course_id, section_code, tutorial_code)
VALUES ('S20251100', 1, 101, NULL);
```

Test 10: Multiple Foreign Key Violations
Purpose: To test the behavior when multiple invalid foreign keys (student_number, course_id) are used in the same insertion.

```
INSERT INTO enrollment (student_number, course_id, section_code, tutorial_code)
VALUES ('INVALID123', 999, 101, 201);
```

3.1.2 Identifying Complex constraints

As our system integrates multiple core business processes ranging from student admission to course enrollment, it must account for several complex, and sometimes subtle, constraints within the database. Below, we outline four of the most notable constraints that should be enforced:

1. **Course Prerequisites in Enrollment:**
   In the current design, the prerequisite table indicates which courses are required before enrolling in another course. However, this table serves only as a reference, where no foreign keys enforce the prerequisite relationships. This is due to the fact that course_id in the course table is specific to a course instance in a particular year and term, while the prerequisite table uses course_code to define relationships. Because of this mismatch, we cannot efficiently validate whether a student has completed the prerequisite(s) before enrolling in a higher-level course. Currently, a viable but inefficient way to validate this is to, for each higher-level course a student is enrolled in, SELECT its prerequisites (including any nested prerequisites) from the prerequisite table, and compare them to a list of course_codes SELECTed from the enrollment table filtered by the student's student_number. This approach relies on matching course_code rather than course_id.

2. **Alternative Prerequisite Combinations:**
   Some courses may allow multiple prerequisite paths. For example, CSC121 might accept either CSC110 as a standalone prerequisite or a combination of CSC103 and CSC107. Our current schema design for prerequisites does not support such logical "OR" conditions, making it impossible to enforce these flexible prerequisite combinations.

3. **Course Retakes Based on Performance:**
   The enrollment table effectively restricts students to a single section and (at most) one tutorial per course instance, as defined by the composite primary key (student_number, course_id). However, students may retake the same course in a different year or term (resulting in a different course_id). In practice, retaking a course should typically be allowed only if the student previously failed it. While we initially considered implementing an academic_record table to

track course outcomes, we ultimately chose to exclude it to keep the system simpler. Nevertheless, a more realistic design would include constraints based on student performance to regulate course retakes.
To check for violations, a SELECT query could identify students enrolled in the same course code across multiple years where there is no record of a failed attempt.
4. **Program-Specific Course and Load Restrictions:**
In real-world educational systems, different programs often have rules regarding which courses their students may take, as well as limits on the course load per term. These kinds of constraints are important to ensure students follow structured academic pathways. Our current database does not implement such program-specific rules, but these are commonly enforced in modern university systems.

4.Key Reports

An analysis to forecast yield rates, optimize offer policies, and spot anomalies in operational execution, our system empowers admissions officers. Three key report groups rooted in the relational schema implemented in MySQL were designed to support data-driven decision-making at the University of Toronto. Each report addresses a distinct business operational domain:

1. Admissions trends
2. Teaching Staff Workload and
3. Enrollment Trends

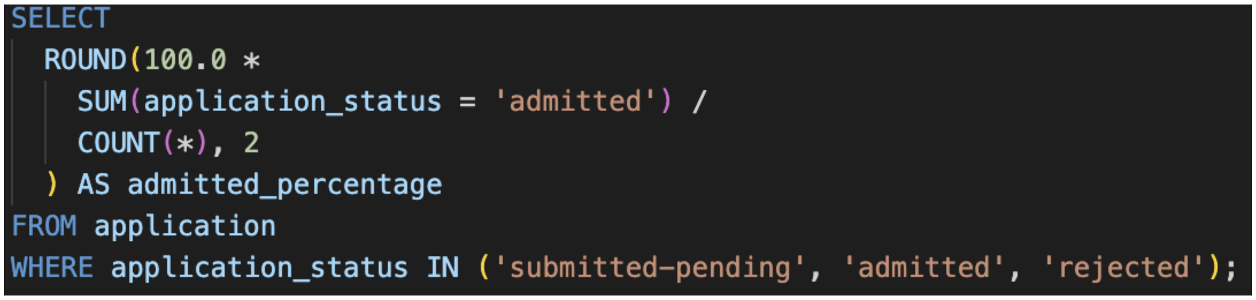
Report 1: Admission Analysis Objective

This analysis supports business decisions by admissions officers to forecast yield rates, optimize offer policies, and spot anomalies in operational execution. The report provides a macro view of the admissions pipeline:

1. Evaluate the conversion rate from application to admission: Percentage of Applicants Admitted
2. Track outcomes for admitted students (accepted, declined, pending)
3. Verify data integrity by ensuring all admitted applicants have corresponding offers.

This report draws data from two core tables: application and admission_offer. From the application table, the application_status attribute is used to calculate the number of applicants in each stage of the admissions pipeline. The admission_offer table is joined on application_id to analyze applicant decisions (accepted, declined, pending) and verify that every admitted application is followed by an official offer record.

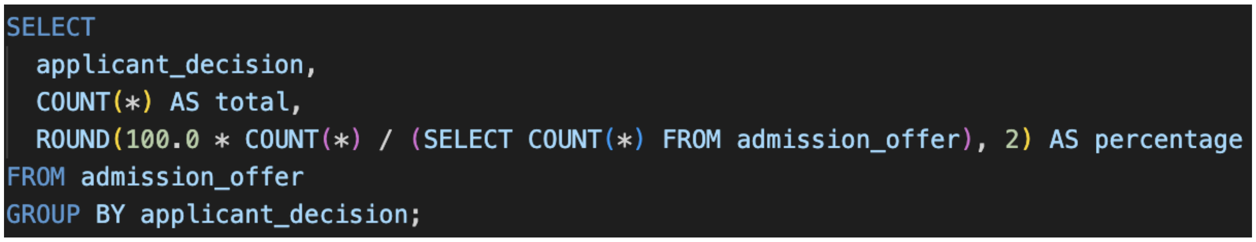
This query calculates the percentage of admitted applications out of all submitted applications, excluding not-submitted statuses. It uses boolean logic inside SUM() for conditional aggregation.



```sql
SELECT
  ROUND(100.0 *
    SUM(application_status = 'admitted') /
    COUNT(*), 2
  ) AS admitted_percentage
FROM application
WHERE application_status IN ('submitted-pending', 'admitted', 'rejected');
```

Figure 4: Percentage of admitted applications SQL

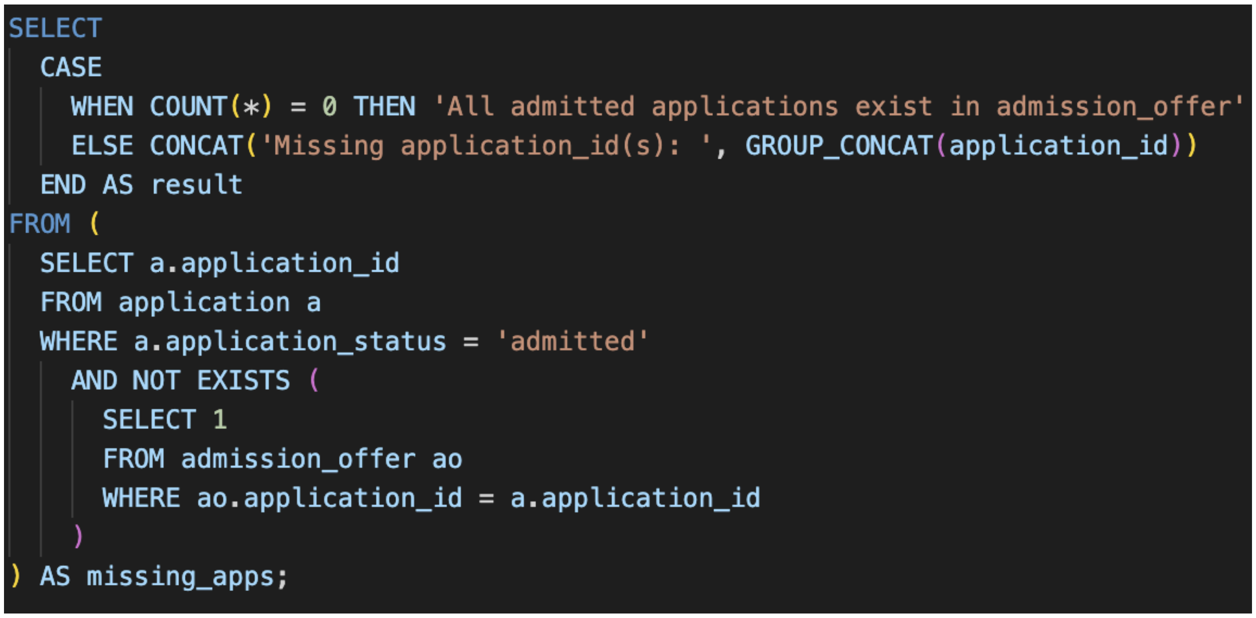
This query determines the distribution of applicant decisions (accepted, declined, pending). It facilitates analysis of offer conversion and applicant behavior.



```sql
SELECT
  applicant_decision,
  COUNT(*) AS total,
  ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM admission_offer), 2) AS percentage
FROM admission_offer
GROUP BY applicant_decision;
```

Figure 5: Outcomes for Admitted Applicants

This query ensures that every admitted application has a corresponding admission_offer. It has a role of data integrity audit, checking for breaks in business process automation.

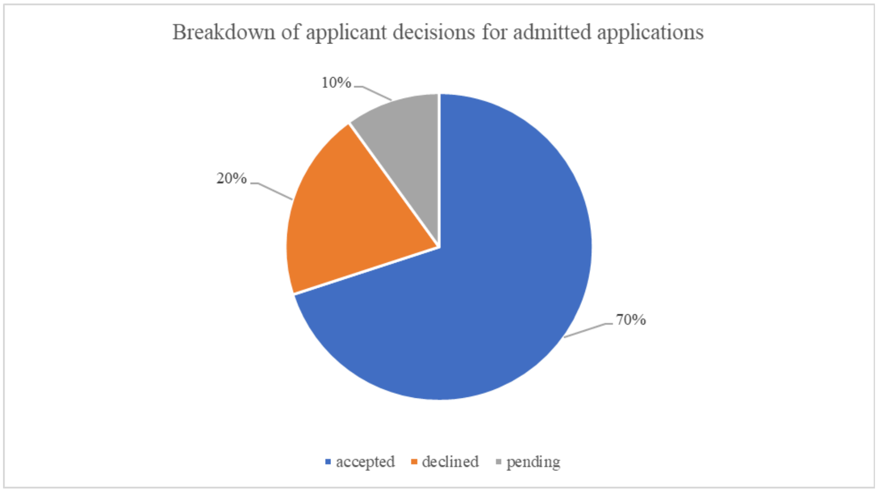


```sql
SELECT
  CASE
    WHEN COUNT(*) = 0 THEN 'All admitted applications exist in admission_offer'
    ELSE CONCAT('Missing application_id(s): ', GROUP_CONCAT(application_id))
  END AS result
FROM (
  SELECT a.application_id
  FROM application a
  WHERE a.application_status = 'admitted'
    AND NOT EXISTS (
      SELECT 1
      FROM admission_offer ao
      WHERE ao.application_id = a.application_id
    )
) AS missing_apps;
```

Figure 6: Integrity audit for missing offers

Test results:

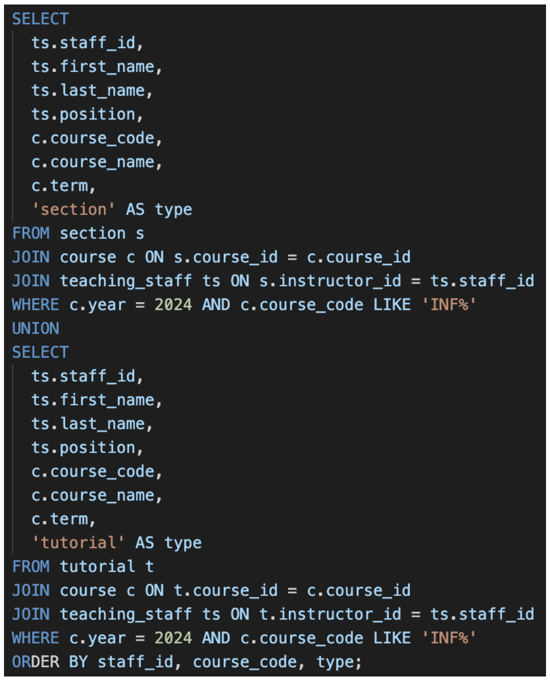
applicant_decision	**total**	**percentage**
accepted	7	70%
declined	2	20%
pending	1	10%



Report 2: Teaching Staff Workload

The report integrates data from the `teaching_staff`, `section`, `tutorial`, and `course` tables. Instructor information such as `staff_id`, `first_name`, `last_name`, and `position` is sourced from `teaching_staff`. Course assignment details are retrieved from both `section` and `tutorial`, which are joined with the `course` table to obtain metadata such as `course_code`, `course_name`, and `term`. The UNION of section and tutorial responsibilities allows for a consolidated view of all teaching loads.

This query gives a unified view of staff assignments. Academic administrators will use this report to evaluate faculty load distribution and manage hiring or scheduling needs. It will help to visualize the teaching responsibilities of professors and teaching assistants across all courses (lectures and tutorials) offered. UNION is used to combine sections and tutorials into a unified output and filters ensure only INF courses in 2024 are included.



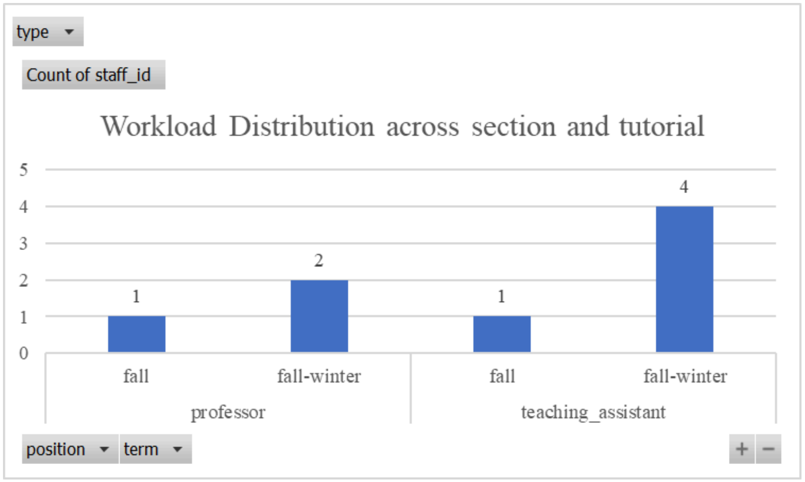
```
SELECT
  ts.staff_id,
  ts.first_name,
  ts.last_name,
  ts.position,
  c.course_code,
  c.course_name,
  c.term,
  'section' AS type
FROM section s
JOIN course c ON s.course_id = c.course_id
JOIN teaching_staff ts ON s.instructor_id = ts.staff_id
WHERE c.year = 2024 AND c.course_code LIKE 'INF%'
UNION
SELECT
  ts.staff_id,
  ts.first_name,
  ts.last_name,
  ts.position,
  c.course_code,
  c.course_name,
  c.term,
  'tutorial' AS type
FROM tutorial t
JOIN course c ON t.course_id = c.course_id
JOIN teaching_staff ts ON t.instructor_id = ts.staff_id
WHERE c.year = 2024 AND c.course_code LIKE 'INF%'
ORDER BY staff_id, course_code, type;
```

Test Results:
We will find all the INF courses (sections and tutorials) in 2024 and list out all the teaching staff responsible for them. We will report on the number of courses distributed across the various sections and semesters.

staff_id	**first_name**	**last_name**	**position**	**course_code**	**course_name**	**term**	**type**
TS10000001	Alice	Nguyen	professor	INF1204	Statistics for Info	fall	section
TS10000001	Alice	Nguyen	professor	INF1668	Eight near	fall-winter	section
TS10000001	Alice	Nguyen	professor	INF7203	Anything ability million	fall-winter	section
TS10000003	Carol	Smith	teaching_assistant	INF1204	Statistics for Info	fall	tutorial
TS10000003	Carol	Smith	teaching_assistant	INF7203	Anything ability million	fall-winter	tutorial
TS10000005	Eva	Patel	teaching_assistant	INF1668	Eight near	fall-winter	tutorial
TS10000005	Eva	Patel	teaching_assistant	INF4869	Tree tell fast	fall-winter	section

TS10000005	Eva	Patel	teaching_assistant	INF4869	Tree tell fast	fall-winter	tutorial

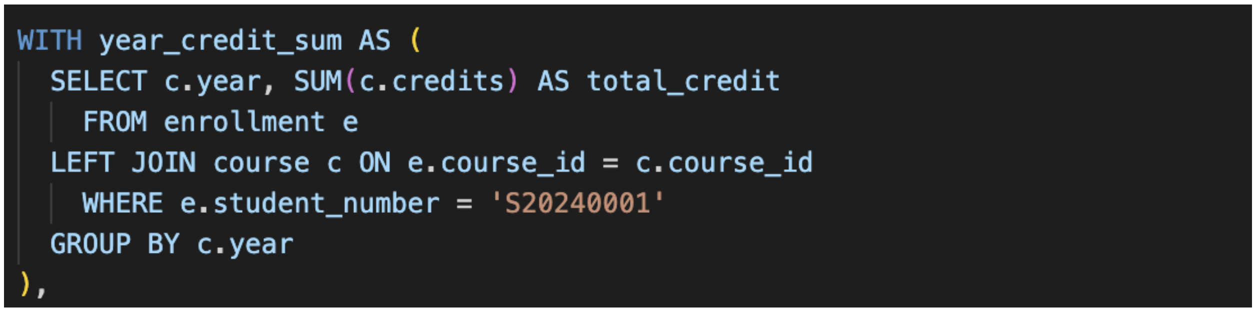
Pivot table to count the number of teaching staff assigned according to their role and the academic semester.



Report 3: Enrollment Trends

This report relies on data from the enrollment, course, and student tables. The enrollment table tracks which courses each student has registered for via student_number and course_id. It is joined with the course table to retrieve the credit value (credits), year (year), and offering unit (offered_by). Student information such as student_number and program enrollment is pulled from the student table to isolate trends by individual and program.

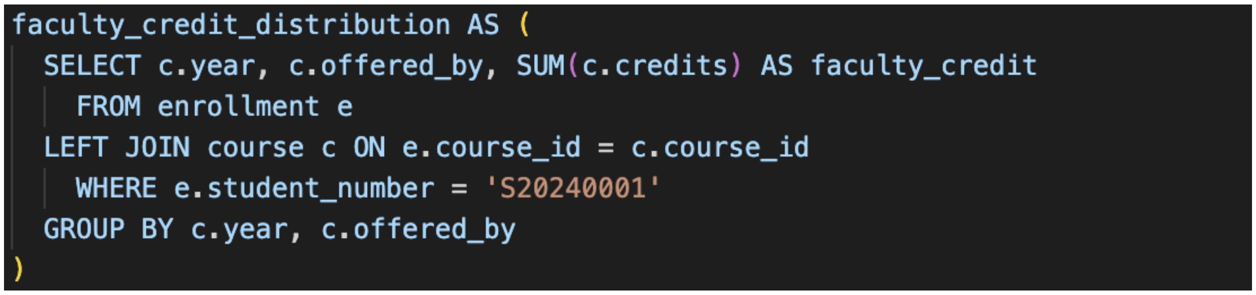
This query gives the total credits per year. The Common Table Expression (CTE) calculates total credits earned annually.



```
WITH year_credit_sum AS (
  SELECT c.year, SUM(c.credits) AS total_credit
    FROM enrollment e
  LEFT JOIN course c ON e.course_id = c.course_id
    WHERE e.student_number = 'S20240001'
  GROUP BY c.year
),
```

Figure 7: Total Credits Per Year

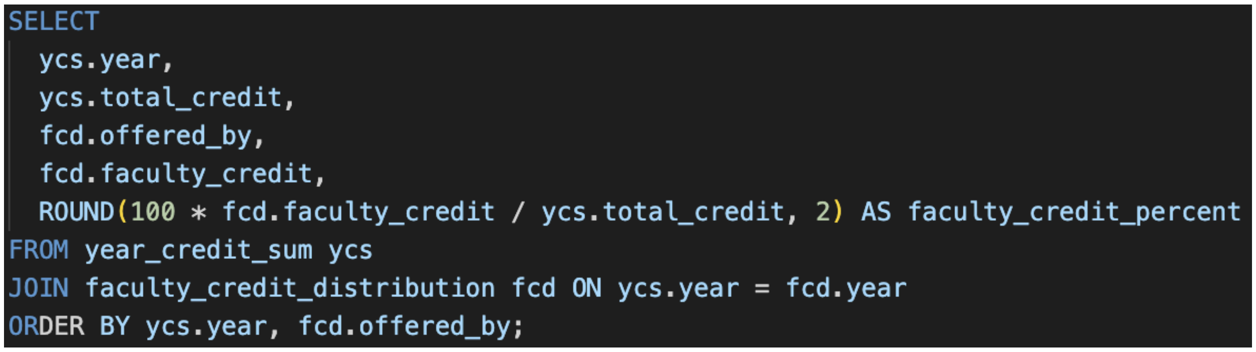
This query gives credit distribution by faculty per year. This CTE calculates how the student's credits are split across departments/faculties.



```
faculty_credit_distribution AS (
  SELECT c.year, c.offered_by, SUM(c.credits) AS faculty_credit
    FROM enrollment e
  LEFT JOIN course c ON e.course_id = c.course_id
    WHERE e.student_number = 'S20240001'
  GROUP BY c.year, c.offered_by
)
```

Figure 8: Credit Distribution by Faculty Per Year

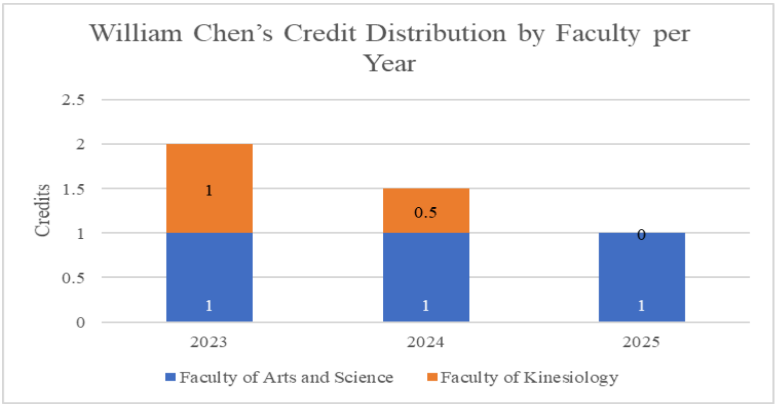
This query combines yearly and faculty credit totals.



```
SELECT
  ycs.year,
  ycs.total_credit,
  fcd.offered_by,
  fcd.faculty_credit,
  ROUND(100 * fcd.faculty_credit / ycs.total_credit, 2) AS faculty_credit_percent
FROM year_credit_sum ycs
JOIN faculty_credit_distribution fcd ON ycs.year = fcd.year
ORDER BY ycs.year, fcd.offered_by;
```

For a specific student (William Chen, student number: S20240001), we will calculate the total number of credits earned each year. We will analyze the distribution of his enrolled credits across the various faculties offering the courses, per year.

year	**total_credit**	**offered_by**	**faculty_credit**	**faculty_credit_percent**
2023	2	Faculty of Arts and Science	1	50
2023	2	Faculty of Kinesiology	1	50
2024	1.5	Faculty of Arts and Science	1	66.67
2024	1.5	Faculty of Kinesiology	0.5	33.33
2025	1	Faculty of Arts and Science	1	100



Statement of Individual Contributions:

- **Kyle Thomas** - Worked on the data dictionary and key reports.
- **Diana Malynovska** - Worked on the executive summary, EER & Relational diagram, formatting of the assignment and contributed to reviewing.
- **Steven Tsai** - Worked on the development of the database schema and the creation of example data (script1.sql), ensuring alignment with the ER diagram; identified the four complex constraints as detailed in 1.4.2; implemented the three key report queries in script3.sql.
- **Hon Wa Ng** - Worked on Conceptual Data Model, contributed to reviewing Data Model with the team.