1. List of Students and Their Enrollments

```
SELECT
   Students.student_id,
   Students.first_name,
   Enrollments.enrollment_id,
   Enrollments.course_name,
         Enrollments.enrollment_date
FROM
   Students
JOIN
   Enrollments ON Students.student_id = Enrollments.student_id;
```

2. Retrieve a list of all students and the courses they are currently enrolled in, including course details.

```
SELECT
   Students.student_id,
   Students.first_name,
   Enrollments.course_name,
   Courses.course_description
FROM
   Students
LEFT JOIN
   Enrollments ON Students.student_id = Enrollments.student_id
LEFT JOIN
   Courses ON Enrollments.course_name = Courses.course_name;
```

3. Find the students who do not have assigned advisors.

```
SELECT student_id, first_name, last_name
FROM Students
WHERE student_id NOT IN (SELECT student_id FROM Advisors);
```

4. Identify the student(s) with the highest GPA and their academic records.

```
SELECT
   Students.student_id,
   Students.first_name,
   AVG(Transcripts.gpa_for_course) AS total_gpa
FROM
```

```
    Students
JOIN
    Transcripts ON Students.student_id = Transcripts.student_id
GROUP BY
    Students.student_id, Students.first_name
HAVING
    AVG(Transcripts.gpa_for_course) = (SELECT MAX(avg_gpa) FROM (SELECT
AVG(gpa_for_course) AS avg_gpa FROM Transcripts GROUP BY student_id) AS
avg_table);
```

5. Calculate the average GPA for students in each major.

```
SELECT
    Students.student_id,
    Students.first_name,
    Students.last_name,
    Students.major,
    AVG(Transcripts.gpa_for_course) AS total_gpa
FROM
    Students
JOIN
    Transcripts ON Students.student_id = Transcripts.student_id
GROUP BY
    Students.student_id, Students.first_name, Students.last_name, Students.major
ORDER BY
    Students.major;
```

6. Determine which departments offer the most courses by counting the number of courses offered in each department.

```
SELECT
    Departments.department_id,
    Departments.department_name,
    COUNT(Courses.course_id) AS course_count
FROM
    Departments
LEFT JOIN
    Courses ON Departments.department_id = Courses.department_id
GROUP BY
    Departments.department_id, Departments.department_name
ORDER BY
    course_count DESC;
```

7. List faculty advisors along with the students they advise.

```
SELECT
    Faculty.faculty_name,
    Advisors.advisor_name,
    Students.first_name,
    Students.last_name
FROM
    Faculty
JOIN
    Advisors ON Faculty.advisor_id = Advisors.advisor_id
LEFT JOIN
    Students ON Advisors.student_id = Students.student_id
ORDER BY
    Faculty.faculty_name, Advisors.advisor_name, Students.last_name, Students.first_name;
```

8. Find the student groups with the most members and list the group names and member counts.

```
SELECT
    StudentGroups.group_name,
    COUNT(Students.student_id) AS member_count
FROM
    StudentGroups
LEFT JOIN
    Students ON StudentGroups.group_id = Students.member_of
GROUP BY
    StudentGroups.group_name
ORDER BY
    member_count DESC;
```

9. Calculate the occupancy rate of the university's student housing facilities.

```
SELECT
    housing_id,
    building_name,
   (SUM(current_occupancy) * 100.0 / SUM(capacity)) AS occupancy_rate
FROM
    Housing
GROUP BY
    housing_id, building_name
ORDER BY housing_id;
```

10. Compute the average cost of meal plans for different student groups (e.g., freshmen, sophomores, etc.).

```
SELECT
   Students.class_year,
   AVG(MealPlans.price) AS average_meal_cost
FROM
   Students
JOIN
   MealPlans ON Students.class_year = MealPlans.meal_for
GROUP BY
   Students.class_year;
```

11. Calculate the total tuition revenue generated by each academic department.

```
SELECT
   Departments.department_name,
   SUM(StudentFees.tuition_fee) AS total_tuition_revenue
FROM
   Departments
JOIN
   Courses ON Departments.department_id = Courses.department_id
JOIN
   Enrollments ON Courses.course_name = Enrollments.course_name
JOIN
   StudentFees ON  Enrollments.student_id = StudentFees.student_id
GROUP BY
   Departments.department_id, Departments.department_name
ORDER BY
   total_tuition_revenue DESC;
```

12. Find the number of available library resources and the number checked out by students.

```
SELECT COUNT(resource_id) AS total_resources
FROM Library;

SELECT COUNT(resource_id) AS checked_resources
FROM Library
WHERE checkout_status = 'Checked Out';
```

13. Calculate the number of student visits to health services and their average visit duration.

```
SELECT
   SUM(total_visits) AS total_visits_sum,
   AVG(visit_duration_minutes) AS average_duration
FROM HealthServices;
```

14. List student achievements (awards, honors) and group them by the student's department.

```
SELECT
   Departments.department_name,
   StudentAchievements.achievement_type,
   StudentAchievements.achievement_description,
   StudentAchievements.award_name,
   StudentAchievements.award_date
FROM
   StudentAchievements
JOIN
   Departments ON StudentAchievements.department_id = Departments.department_id
ORDER BY
   Departments.department_name, StudentAchievements.award_date DESC;
```

15. Determine the percentage of students who have participated in internships.

```
SELECT
   COUNT(DISTINCT Students.student_id) AS total_students,
   COUNT(DISTINCT Internships.student_id) AS students_with_internships,
   CONCAT(ROUND((COUNT(DISTINCT Internships.student_id) * 100.0 /
COUNT(DISTINCT Students.student_id)), 2), '%') AS internship_percentage
FROM
   Students
LEFT JOIN
   Internships ON Students.student_id = Internships.student_id;
```

16. Find the countries where students have studied abroad and the number of students in each country.

```
SELECT
   StudyAbroad.country,
   COUNT(DISTINCT StudyAbroad.student_id) AS students_count
```

```
FROM StudyAbroad
GROUP BY StudyAbroad.country;
```

17. List the upcoming campus events and their details, sorted by date.

```
SELECT
   event_name,
   event_date,
   event_location,
   organizer,
   description
FROM
   StudentEvents
WHERE
   isUpcoming = true
ORDER BY
   event_date;
```

18. Determine which departments produce the most employed alumni.

```
SELECT
   Departments.department_name,
   COUNT(alumni_id) AS employed_alumni_count
FROM
   Alumni
JOIN
   Departments ON Alumni.department_id = Departments.department_id
WHERE
   Alumni.employment_status = 'Employed'
GROUP BY
   Departments.department_name
ORDER BY
   employed_alumni_count DESC;
```

19. Identify faculty members who have expertise in specific research areas, based on their academic records.

```
SELECT Instructors.name AS faculty_member, Instructors.research_area
FROM Instructors
WHERE research_area IS NOT NULL;
```

20. Analyze the historical enrollment data to identify trends in student enrollment over the past few years.

```
SELECT
   EXTRACT(YEAR FROM enrollment_date) AS enrollment_year,
   SUM(registered_students) AS total_registered_students
FROM
   Enrollments
GROUP BY
   enrollment_year
ORDER BY
   enrollment_year;
```

21. Verify if students enrolling in advanced courses meet the prerequisites by checking their transcript records.

```
SELECT
   Transcripts.student_id,
   Courses.course_name,
   CASE
      WHEN Transcripts.LetterGrade != 'F' AND Transcripts.gpa_for_course > 0 THEN
TRUE
      ELSE FALSE
   END AS prerequisites_perfoms
FROM Transcripts
JOIN Courses ON Transcripts.course_name = Courses.course_name
WHERE Courses.isAdvanced = TRUE;
```

22. List students with outstanding fees, including the total amount owed.

```
SELECT
   student_id,
   (tuition_fee + student_fee + dorm_fee + food_fee) AS total_amount_owed,
   payment_status
FROM StudentFees
WHERE payment_status = 'Unpaid';
```

23. Identify instructors who are teaching multiple courses in the same term and list the courses they are teaching.

```
SELECT
```

```
    Instructors.instructor_id,
    Instructors.name AS instructor_name,
    Instructors.course_term,
    STRING_AGG(Courses.course_name, ', ') AS list_of_courses
FROM Instructors
JOIN Courses ON Instructors.course_term = Courses.term AND Instructors.course_name =
Courses.course_name
GROUP BY Instructors.instructor_id, instructor_name, Instructors.course_term
HAVING COUNT(Instructors.course_name) > 1;
```

24. Calculate statistics on student diversity, such as the distribution of gender, ethnicity, or nationality.

```
SELECT
    gender,
    COUNT(*) AS count
FROM Students
GROUP BY gender;


SELECT
    ethnicity,
    COUNT(*) AS count
FROM Students
GROUP BY ethnicity;


SELECT
    nationality,
    COUNT(*) AS count
FROM Students
GROUP BY nationality;
```

25. Find the most popular combinations of courses (sets of courses taken together) among students.

```
SELECT
    student_id,
    STRING_AGG(course_name, ', ' ORDER BY course_name) AS course_combination,
    COUNT(DISTINCT course_name) AS course_count
FROM Enrollments
GROUP BY student_id
```

```
HAVING COUNT(DISTINCT course_name) > 1
ORDER BY course_count DESC;
```

26. Compare the academic performance (GPA) of students based on their faculty advisors.

```
SELECT
    Faculty.advisor_id AS faculty_advisor,
    Advisors.advisor_name,
    ROUND(AVG(Transcripts.gpa_for_course), 2) AS average_gpa
FROM Faculty
JOIN Advisors ON Faculty.advisor_id = Advisors.advisor_id
JOIN Students ON Advisors.student_id = Students.student_id
JOIN Transcripts ON Students.student_id = Transcripts.student_id
GROUP BY Faculty.advisor_id, Advisors.advisor_name
ORDER BY average_gpa DESC;
```

27. Identify student groups that have members from a wide range of majors, promoting interdisciplinary collaboration.

```
SELECT
    StudentGroups.group_name,
    STRING_AGG(Students.first_name, ', ') AS studen_names,
    STRING_AGG(Students.major, ', ') AS major_groups
FROM
    StudentGroups
JOIN
    Students ON StudentGroups.group_id = Students.member_of
GROUP BY
    StudentGroups.group_name
HAVING
    COUNT(DISTINCT Students.major) > 1
ORDER BY
    StudentGroups.group_name;
```

28. List courses with consistently high enrollment, helping with scheduling and resource allocation.

```
SELECT
    Courses.course_id,
    Courses.course_name,
    ROUND(AVG(Enrollments.registered_students), 2) AS average_enrollment
```

```
FROM Courses
JOIN Enrollments ON Courses.course_name = Enrollments.course_name
GROUP BY Courses.course_id, Courses.course_name
HAVING AVG(Enrollments.registered_students) > 100;
```

29. Calculate the average time it takes students to graduate, considering their major and any changes in degree programs.

```
SELECT
    Students.major,
    AVG(EXTRACT(YEAR FROM AGE(COALESCE(Graduation.graduation_date,
DegreeProgress.expected_graduation_date), Admissions.admission_date))) AS
average_graduation_time
FROM Students
JOIN Admissions ON Students.student_id = Admissions.student_id
LEFT JOIN DegreeProgress ON Students.student_id = DegreeProgress.student_id
LEFT JOIN Graduation ON Students.student_id = Graduation.student_id
GROUP BY Students.major;
```

30. Determine if students who complete internships have a higher graduation rate compared to those who do not.

```
SELECT
    CASE WHEN Internships.endedIntern = TRUE THEN 'Ended Internship' ELSE 'Didnt
end Internship' END AS internship_status,
    CONCAT(ROUND(AVG(CAST(Graduation.isGraduated AS INT)) * 100, 2), '%') AS
graduation_rate
FROM
    Internships
JOIN
    Graduation ON Internships.student_id = Graduation.student_id
GROUP BY
    internship_status;
```