DATABASE MANAGEMENT SYSTEM

Chapter 4
SQL

Bikash Khadka Shah

MCSE, MSDA, OCP, RHCE, RHCVA, CCNA, CEH

9841766620 | 9801076620

Bikash Khadka Shah - MCSE, MSDA, OCP, RHCE, RHCVA, CCNA, CEH

The SQL Query Language

- SQL: It stands for Structured Query Language. It allows us to access and manipulate the data.
- SQL is primarily used to interact with RDBMS, allowing users to create, modify, and query relational databases.
- SQL is used for the following:
  - Modifying database table and index structures;
  - Adding, updating and deleting rows of data; and
  - Retrieving subsets of information from within relational database management systems .
  - Retrieve data can be used for transaction processing, analytics applications and other applications that require communicating with a relational database.

SQL Query Language

- Structured Query Language or SQL is the standard language for Relation Database System.
- All relational database management systems like MySQL, MS Access, Oracle, Postgres and SQL Server use SQL as standard database language.
- It:
  - Allows users to access data in relational database management systems.
  - Allows users to describe the data.
  - Allows users to define the data in database and manipulate that data. Allows users to create and drop databases and tables

SQL Query Language

- SQL commands can be categorized into:
- Data definition language (DDL), used to define the database structure or table: create, alter, drop
- Data manipulation language (DML), used to manage data within table: insert, delete, update, select
- Transaction control language (TCL), used to apply the changes permanently save into database: commit, rollback, save point
- Data Control Language (DCL), used to give privileges to access limited data: grant, revoke

DDL: CREATE TABLE

- Syntax– CREATE TABLE <tablename>

(

<col1 definition> [col1 constraints>],

<col2 definition> [col2 constraints>],

.

.

<coln definition> [coln constraints>],

);

- Attribute data types and domains
- Attribute constraints and attribute defaults
- Key and referential integrity constraints
- Create statements are also used to create other database objects like views, procedures or even database

DDL: ALTER TABLE

- ALTER TABLE table_name RENAME TO new_table_name;
- ALTER TABLE table_name ADD column_name datatype[(size)];
- ALTER TABLE table_name MODIFY column_name column_datatype[(size)];
- ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;
- ALTER TABLE table_name DROP COLUMN column_name;

DDL: Drop Statements

- These statements delete the database objects.
- DROP TABLE tablename;
- DROP VIEW viewname;
- DROP PROCEDURE procedurename;
- DROP DATABASE databasename;

DML: Insert

- INSERT INTO tablename(column_list)

VALUES (value_list);

Order of value_list should match the order of column_list.

- INSERT INTO tablename VALUES(value_list);

Order of value_list must match order of column definition in create table.

- INSERT INTO tablename VALUES

(value_list1),

(value_list2),

.

.;

- Inserts multiple records with single statements

DML: Update

- UPDATE tablename SET

column1 = value1,

column2 = value2,

.

.

[WHERE condition];

- Where clause is optional; if present it filters on the records for which update will be applied.
- If where clause is not present, all records will be updated

DML: Delete

- DELETE FROM tablename [WHERE condition];
- Where clause is optional; if present it filters the records to delete
- If where clause is not present, all records will be
- If where clause is not present, all records will be deleted

DML: Select

- SELECT column_list FROM table_list [WHERE condition] [ORDER BY column ASC|DESC];
- WHERE clause filters the records to retrieve.
- ORDER BY clause controls the sorting or result.
- Table list can have one or multiple tables; if there
- Table list can have one or multiple tables; if there are multiple tables in FROM clause, we need to provide join condition in WHERE clause.

Data Control Language

- Data Control Language - DCL - includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.
- List of DCL commands:
  - GRANT: This command gives users access privileges to the database.
  - REVOKE: This command withdraws the user's access privileges given by using the GRANT command

Transaction control language

- Transaction control language (TCL) - commands deal with the transaction within the database.
- List of TCL commands:
  - COMMIT: Commits a Transaction.
  - ROLLBACK: Rollbacks a transaction in case of any error occurs.
  - SAVEPOINT: Sets a savepoint within a transaction

Aggregate queries

- SQL provide aggregate functions that perform a calculation on a set of values, and returns a single value.
- Most commonly used aggregate functions are:
  - COUNT counts how many rows are in a particular column.
  - SUM adds together all the values in a particular column.
  - MIN and MAX return the lowest and highest values in a particular column, respectively.
  - AVG calculates the average of a group of selected values.
- Except for COUNT(*), aggregate functions ignore null values.
- Aggregate functions are often used with the GROUP BY clause of the SELECT statement, when selected columns contain both aggregate and non aggregate columns.
- We can also use HAVING clause along with aggregate function to filter on result of aggregate function

Aggregate queries…

- SELECT COUNT(*) FROM tablename;
- SELECT COUNT(1) FROM tablename;
- SELECT COUNT(column_name) FROM tablename;
- SELECT MAX(age) FROM staff;
- SELECT MIN(age) FROM staff;
- SELECT MIN(age) FROM staff;
- SELECT SUM(salary) FROM staff;
- SELECT AVG(salary) FROM staff;
- SELECT department, SUM(salary) FROM staff GROUP BY department HAVING SUM(SALARY) > 50000;

Set Operations

```
SELECT column_name FROMtable1
 UNION
SELECT column_name FROMtable2;
```

```
SELECT column_name FROM table1
 INTERSECT
SELECT column_name FROM table2;
```

```
SELECT column_name FROMtable1
 UNION
 SELECT column_name FROMtable2;
```

```
SELECT column_name FROM table1
 MINUS
SELECT column_name FROM table2;
```

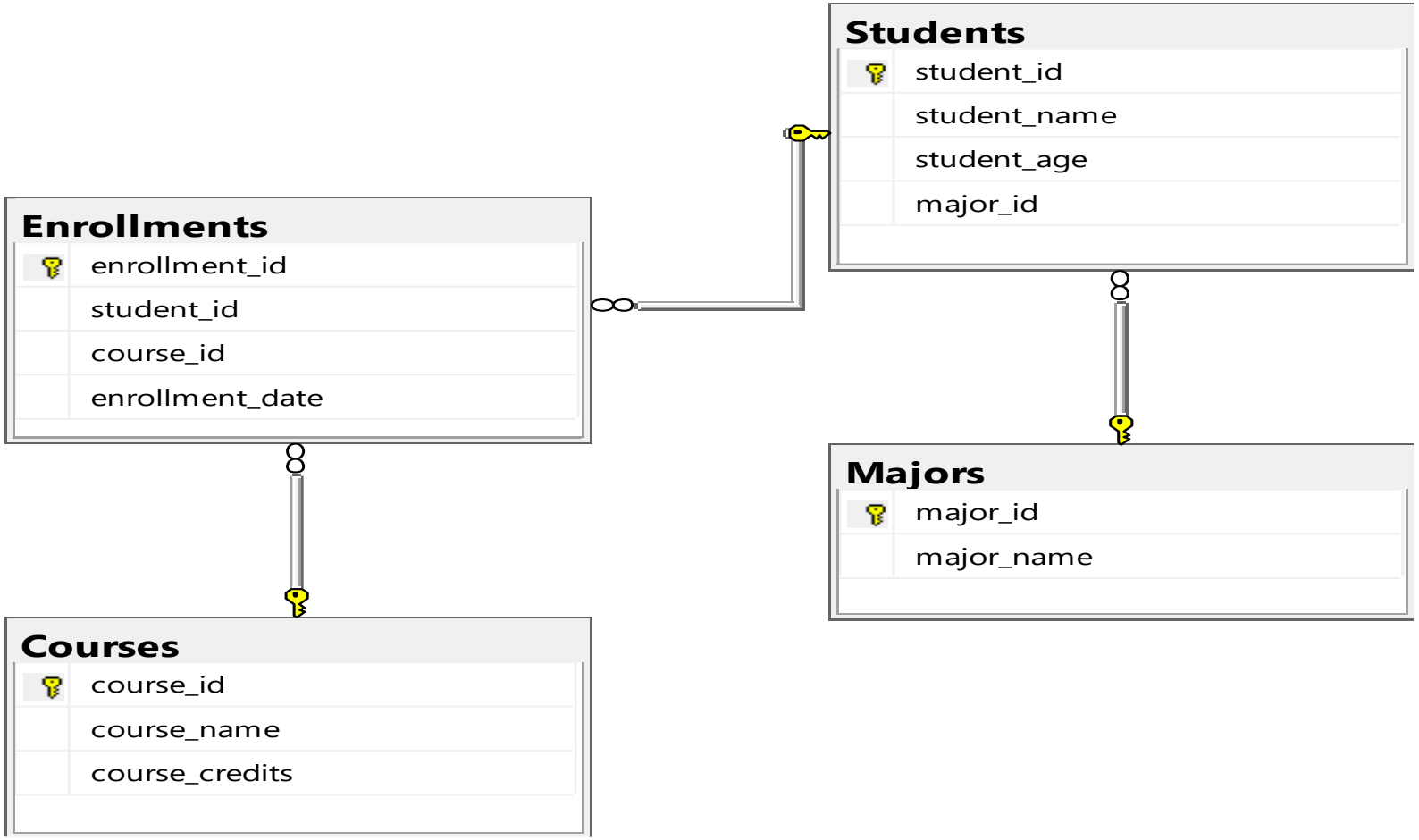
Join Operations

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

```
SELECT columns
FROM table1
RIGHT JOIN table2 ON
table1.column = table2.column;
```

```
SELECT columns
FROM table1
LEFT JOIN table2 ON
table1.column = table2.column;
```

Creating a University Database in SQL



Creating a Database and Using It:

- CREATE DATABASE UniversityDB;

Using the Database:

- USE UniversityDB;

Create Table

Majors Table (Referenced Table):

Assuming a Majors table is referenced in Students with a major_id:

```
CREATE TABLE IF NOT EXISTS Majors
    major_id INT PRIMARY KEY,
    major_name VARCHAR(50)
);
```

#	Name	Type
1	major_id	int(11)
2	major_name	varchar(50)

Students Table (Referencing Majors Table):

```
CREATE TABLE IF NOT EXISTS Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    student_age INT,
    major_id INT,
    FOREIGN KEY (major_id) REFERENCES
Majors(major_id)
);
```

#	Name	Type
1	student_id	int(11)
2	student_name	varchar(50)
3	student_age	int(11)
4	major_id	int(11)

Courses Table:

```
CREATE TABLE IF NOT EXISTS Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    course_credits INT
);
```

#	Name
1	course_id
2	course_name
3	course_credits

Enrollments Table (Referencing Students and Courses):

```
CREATE TABLE IF NOT EXISTS Enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

#	Name	Type
1	enrollment_id	int(11)
2	student_id	int(11)
3	course_id	int(11)
4	enrollment_date	date

Inserting Records:
Inserting into Majors table

```
INSERT INTO Majors (major_id, major_name) VALUES
(1, 'Computer Science’),
(2, 'Electrical Engineering’),
(3, 'Biology’),
(4, 'Business Administration’),
(5, 'Psychology');
```

major_id	major_name
1	Computer Science
2	Electrical Engineering
3	Biology
4	Business Administration
5	Psychology

Inserting into Students table

```
INSERT INTO Students (student_id,
student_name, student_age,
major_id) VALUES
(1, 'John Doe', 20, 1),
(2, 'Jane Smith', 22, 3),
(3, 'Alice Johnson', 21, 2),
(4, 'Michael Brown', 23, 4),
(5, 'Emily Davis', 20, 5);
```

student_id	student_name	student_age	major_id
1	John Doe	20	1
2	Jane Smith	22	3
3	Alice Johnson	21	2
4	Michael Brown	23	4
5	Emily Davis	20	5

Inserting into Courses table

```
INSERT INTO Courses (course_id, course_name, course_credits)
VALUES (101, 'Introduction to Programming', 3),
(102, 'Biology 101', 4),
(103, 'Business Management', 3),
(104, 'Electrical Circuits', 4),
(105, 'Psychology Basics', 3);
```

course_id	course_name	course_credits
101	Introduction to Programming	3
102	Biology 101	4
103	Business Management	3
104	Electrical Circuits	4
105	Psychology Basics	3

Inserting into Enrollments table

```
INSERT INTO Enrollments
(enrollment_id, student_id,
course_id, enrollment_date)
VALUES
(1, 1, 101, '2023-09-05’),
(2, 2, 102, '2023-09-10’),
(3, 3, 104, '2023-09-12’),
(4, 4, 103, '2023-09-15’),
(5, 5, 105, '2023-09-20');
```

enrollment_id	student_id	course_id	enrollment_date
1	1	101	2023-09-05
2	2	102	2023-09-10
3	3	104	2023-09-12
4	4	103	2023-09-15
5	5	105	2023-09-20

Querying Multiple Relations:

Joins: Use JOIN statements to retrieve data from multiple tables based on related columns.

Example Query

```
SELECT Students.student_name,
Majors.major_name
FROM Students
INNER JOIN Majors ON
Students.major_id = Majors.major_id;
```

student_name	major_name
John Doe	Computer Science
Jane Smith	Biology
Alice Johnson	Electrical Engineering
Michael Brown	Business Administration
Emily Davis	Psychology

Creating Relations in SQL:

Foreign Keys: Establish relationships between tables using foreign keys.

Example:

```
ALTER TABLE Enrollments
ADD FOREIGN KEY (student_id) REFERENCES Students(student_id);
```

Before Alter Relation

#	Name	Type
1	enrollment_id	int(11)
2	student_id	int(11)
3	course_id	int(11)
4	enrollment_date	date

After Alter Relation

#	Name	Type
1	enrollment_id	int(11)
2	student_id	int(11)
3	course_id	int(11)
4	enrollment_date	date

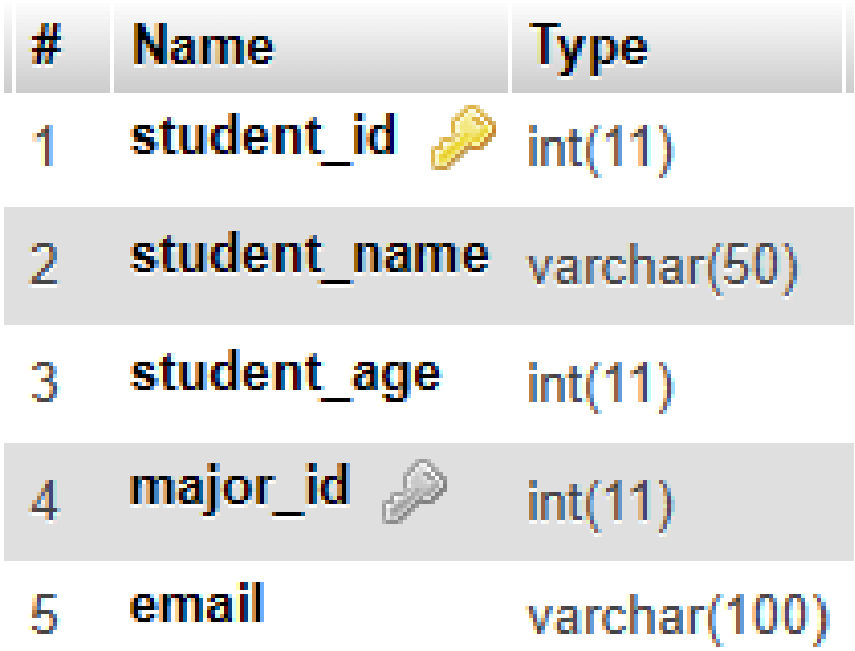
Destroying and Altering Relations:

- Dropping Relations: Use DROP TABLE to remove a table and its data.
- Altering Relations: Use ALTER TABLE to modify existing tables (add columns, modify constraints, etc.).

Example:

```
ALTER TABLE Students
ADD COLUMN email VARCHAR(100);

DROP TABLE Courses;
```



#	Name	Type
1	student_id	int(11)
2	student_name	varchar(50)
3	student_age	int(11)
4	major_id	int(11)
5	email	varchar(100)

Adding and Deleting Tuples:

Inserting Data: Use INSERT INTO to add new records (tuples) into tables.
Deleting Data: Utilize DELETE FROM to remove specific rows from tables.

Example

student_id	student_name	student_age	major_id	email
1	John Doe	20	1	*NULL*
2	Jane Smith	22	3	*NULL*
3	Alice Johnson	21	2	*NULL*
4	Michael Brown	23	4	*NULL*
5	Emily Davis	20	5	*NULL*

INSERT INTO Students (student_id, student_name, student_age, major_id)
VALUES (6, 'Sarah Johnson', 19, 2);

5	Emily Davis	20	5	*NULL*
6	Sarah Johnson	19	2	*NULL*

- DELETE FROM Students WHERE student_id = 6;

student_id	student_name	student_age	major_id	email
1	John Doe	20	1	*NULL*
2	Jane Smith	22	3	*NULL*
3	Alice Johnson	21	2	*NULL*
4	Michael Brown	23	4	*NULL*
5	Emily Davis	20	5	*NULL*
6	Sarah Johnson	19	2	*NULL*

Before Delete

student_id	student_name	student_age	major_id	email
1	John Doe	20	1	*NULL*
2	Jane Smith	22	3	*NULL*
3	Alice Johnson	21	2	*NULL*
4	Michael Brown	23	4	*NULL*
5	Emily Davis	20	5	*NULL*

After Delete

Integrity Constraints (ICs)

- IC: condition that must be true for any instance of the database; e.g., domain constraints.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Primary and Candidate Keys in SQL:

Primary Keys: Unique identifiers for each record in a table.

Candidate Keys: Attributes that can uniquely identify a tuple.

Example:

```
CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) UNIQUE
);
```

Foreign Keys, Referential Integrity in SQL:

- Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer
- Ensures relationships between tables remain consistent.
- Default is NO ACTION (delete/update is rejected)
- CASCADE (also delete all tuples that refer to deleted tuple)
- SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id));
```

```
CREATE TABLE Enrolled (sid CHAR(20), cid CHAR(20), grade CHAR(2), PRIMARY KEY (sid,cid),

FOREIGN KEY (sid) REFERENCES Students ON DELETE CASCADE ON UPDATE SET DEFAULT )
```

Enforcing Referential Integrity:

By utilizing foreign key constraints, you ensure that references from one table to another are valid.

- **Constudent_ider Students and Enrolled; student_id in Enrolled is a foreign key that references Students.**
- **What should be done if an Enrolled tuple with a non-existent student id is inserted? (Reject it!)**
- **What should be done if a Students tuple is deleted?**

➢ **Also delete all Enrolled tuples that refer to it.**

➢ **Disallow deletion of a Students tuple that is referred to.**

➢ **Set student_id in Enrolled tuples that refer to it to a default student_id.**

(In SQL, also: Set student_id in Enrolled tuples that refer to it to a

special value null, denoting `unknown' or `inapplicable'.)

Similar if primary key of Students tuple is updated.

```
ALTER TABLE Enrollments
ADD FOREIGN KEY (course_id) REFERENCES Courses(course_id);
```

[If error occur while altering table first delete record from reference table of update record according to reference table UPDATE Enrollments SET course_id = NULL]

Categories of SQL Commands

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

Data Definition Language (DDL) commands are used to define, modify, and delete the structure of database objects.

CREATE TABLE: Create a new table.

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    student_age INT,
    major_id INT,
    FOREIGN KEY (major_id)
REFERENCES Majors(major_id)
);
```

- ALTER TABLE: Modify an existing table's structure.

```
ALTER TABLE Students
ADD COLUMN email VARCHAR(100);
```

- DROP TABLE: Delete a table from the database.

```
DROP TABLE Students;
```

Data Manipulation Language (DML): commands are used to manipulate data within the database.

- SELECT: Retrieve data from tables.

```
SELECT student_name, student_age
FROM Students;
```

- INSERT INTO: Add new records into a table

```
INSERT INTO Students (student_id,
student_name, student_age,
major_id)
VALUES (1, 'John Doe', 20, 1);
```

- UPDATE: Modify existing records in a table.

```
UPDATE Students SET
student_age = 21 WHERE
student_id = 1;
```

- DELETE FROM: Remove records from a table.

```
DELETE FROM Students WHERE
student_id = 1;
```

Data Control Language (DCL)commands are used to control access and permissions within the database.

- GRANT: Give specific privileges to a user.

GRANT SELECT ON Students TO user1;

- REVOKE: Remove specific privileges from a user.

REVOKE SELECT ON Students FROM user1;

Data Transaction Control Language (TCL) commands are used to manage transactions within the database.

- COMMIT: Save the transaction and make changes permanent.

```
COMMIT;
```

- ROLLBACK: Undo changes made during the transaction.

```
ROLLBACK;
```

Data Manipulation Statements:

- SELECT - The Basic Form:

SELECT is used to retrieve data from a database.

SELECT column1, column2 FROM table_name WHERE condition;

SELECT * FROM Students;

Subqueries:

- Subqueries or nested queries are queries within another SQL query.
- They can be used within SELECT, INSERT, UPDATE, or DELETE statements.

Example:

SELECT column1 FROM table1 WHERE column2 IN (SELECT column3 FROM table2);

SELECT student_name FROM Students WHERE student_id IN (SELECT student_id FROM Enrollments WHERE course_id = 101);

Functions:

- SQL functions perform calculations on data and return results.

Examples: COUNT(), SUM(), AVG(), MAX(), MIN().

```
SELECT COUNT(student_id) AS student_count
FROM Students;
```

GROUP BY Feature

- GROUP BY is used to group rows that have the same values.
- Typically used with aggregate functions (SUM(), COUNT(), etc.).

Example:

```
SELECT department_id, COUNT(*) AS student_count
FROM Students
GROUP BY department_id;
```

Updating the Database:

- UPDATE statement modifies existing records in a table.

UPDATE table_name SET column1 = value1 WHERE condition;

Example:

```
UPDATE Students
SET student_age = student_age + 1
WHERE department_id = 1;
```

Views:

- Views are virtual tables generated by SQL query results. They can simplify complex queries and enhance security.

```
CREATE VIEW view_name AS SELECT column1, column2 FROM table_name
WHERE condition;
```

Example:

```
CREATE VIEW StudentDetails AS
SELECT student_id, student_name, student_age
FROM Students;
```

Embedded SQL:

- Embedded SQL allows SQL statements to be embedded within a programming language.(example in a Python script using an embedded SQL query).
- It facilitates interaction between the database and the application.

Declaring Variables and Exceptions:

- SQL allows declaring variables for temporary storage of data and handling exceptions for error conditions.
- In SQL, you can declare variables and handle exceptions in certain database management systems like PL/pgSQL in PostgreSQL or PL/SQL in Oracle. However, the syntax may vary between different database systems. Here's a general example using PL/pgSQL:

```
-- Declaring a variable
DO $$
DECLARE
    student_count INT;
BEGIN
    -- Initializing the variable
    student_count := 0;

    -- Exception handling
    BEGIN
        -- Perform operations that might raise an exception
        SELECT COUNT(*) INTO student_count FROM Students;

        -- Display the result
        RAISE NOTICE 'Total students: %', student_count;
    EXCEPTION
        WHEN division_by_zero THEN
            RAISE NOTICE 'Error: Division by zero';
        WHEN others THEN
            RAISE NOTICE 'An error occurred';
    END;
END $$;
```

Embedding SQL Statements:

Embedding SQL statements within other languages or frameworks to perform database operations.

Transaction Processing:

SQL supports transactional control with commands like COMMIT, ROLLBACK to ensure data consistency and integrity.

Consistency and Isolation:

Refers to the level of data integrity and isolation between multiple transactions in a database.

Atomicity and Durability:

- Atomicity ensures that transactions are executed completely or not at all.
- Durability ensures that committed transactions persist even in the event of system failure.
- Ensuring that transactions are atomic (either fully completed or fully rolled back) and durable (committed changes are permanent)

Dynamic SQL:

- Dynamic SQL allows the creation and execution of SQL statements at runtime, enabling flexibility in query construction.
- Generating and executing SQL statements dynamically based on conditions or variables within a program or application.