**Stored Procedure Documentation: [dbo].[Exam\_Questions]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Exam\_Questions] table.

**1. Exam\_Questions\_Insert**

**Purpose:** Adds a new question to a specific exam.

**Parameters:**

* @Exam\_ID INT: The ID of the exam (Foreign Key to [Exam]).
* @Question\_ID INT: The ID of the question (Foreign Key to [Questions\_Banks]).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that neither @Exam\_ID nor @Question\_ID are NULL.
  + Verifies that the @Exam\_ID exists in the [Exam] table.
  + Verifies that the @Question\_ID exists in the [Questions\_Banks] table.
  + Verifies that the combination of @Exam\_ID and @Question\_ID does not already exist, preventing a Primary Key violation.
* **Execution:**
  + Inserts the new (Exam\_ID, Question\_ID) pair.
  + Uses the OUTPUT clause to return the [Inserted\_Exam\_ID] and [Inserted\_Question\_ID] to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK) and returns a clear, user-friendly error message.

**2. Exam\_Questions\_Select**

**Purpose:** Retrieves one or more exam-question links based on flexible filter criteria.

**Parameters:**

* @Exam\_ID INT = NULL: The ID of the exam to filter by.
* @Question\_ID INT = NULL: The ID of the question to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If both parameters are NULL, returns all records from the table.
  + If both are provided, returns the single record matching the full composite key.
  + If only @Exam\_ID is provided, returns all questions for that exam.
  + If only @Question\_ID is provided, returns all exams that include that question.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Exam\_Questions\_Update**

**Purpose:** Atomically updates a question within an exam. This is achieved by deleting the old question link and inserting the new one within a single transaction.

**Parameters:**

* @Exam\_ID INT: The ID of the exam (this ID does not change).
* @Old\_Question\_ID INT: The existing question ID to be replaced.
* @New\_Question\_ID INT: The new question ID to replace the old one.

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that all three parameters are non-NULL.
  + Verifies that the OLD record (@Exam\_ID, @Old\_Question\_ID) exists.
  + Verifies that the NEW record (@Exam\_ID, @New\_Question\_ID) does not already exist, preventing a PK violation.
  + Verifies that the @New\_Question\_ID exists in the [Question\_Bank] table.
* **Transaction:**
  + Begins a TRANSACTION.
  + DELETEs the OLD record.
  + INSERTs the NEW record.
  + COMMITs the transaction if both steps succeed.
* **Output:** Uses OUTPUT in both the DELETE and INSERT statements to return the [Old\_... ] and [New\_... ] data.
* **Smart Catch:** If any error occurs, the entire transaction is ROLLBACK, and a specific, user-friendly error message is returned.

**4. Exam\_Questions\_Delete**

**Purpose:** Deletes one or more exam-question records based on flexible filter criteria.

**Parameters:**

* @Exam\_ID INT = NULL: The ID of the exam.
* @Question\_ID INT = NULL: The ID of the question.

**Logic & Error Handling:**

* **Conditional Delete:**
  + **By Full PK:** If both @Exam\_ID and @Question\_ID are provided, it deletes the single matching record.
  + **By Partial PK (Exam):** If only @Exam\_ID is provided, it deletes *all* questions associated with that exam.
  + **By Partial PK (Question):** If only @Question\_ID is provided, it deletes *all* exam links for that question.
  + **Delete All:** If both parameters are NULL, it deletes *all* records from the table. It attempts to TRUNCATE for performance but will DELETE if foreign keys exist.
* **Pre-Checks:**
  + For single-record deletes, it checks if the record EXISTS first and returns an error if not found.
  + For partial-key deletes, it checks for records and returns an informational message if none are found.
* **Output:** Uses the OUTPUT clause to return all [Deleted\_... ] records. (Note: TRUNCATE does not use OUTPUT).
* **Smart Catch:** CatDetails and reports FK violations (Error 547) or other unexpected errors.

**Stored Procedure Documentation: [dbo].[Failed\_Students]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Failed\_Students] table.

**1. Failed\_Students\_Insert**

**Purpose:** Adds a new failure reason record for a specific student.

**Parameters:**

* @Student\_ID INT: The ID of the student (Foreign Key to [Student]).
* @Failure\_Reason NVARCHAR(255): The specific reason for failure.

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that neither @Student\_ID nor @Failure\_Reason are NULL.
  + Verifies that the @Student\_ID exists in the [Student] table.
  + Verifies that the combination of @Student\_ID and @Failure\_Reason does not already exist, preventing a Primary Key violation.
* **Execution:**
  + Inserts the new (Student\_ID, Failure\_Reason) pair.
  + Uses the OUTPUT clause to return the [Inserted\_Student\_ID] and [Inserted\_Failure\_Reason] to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK) and returns a clear, user-friendly error message.

**2. Failed\_Students\_Select**

**Purpose:** Retrieves one or more student failure records based on flexible filter criteria.

**Parameters:**

* @Student\_ID INT = NULL: The ID of the student to filter by.
* @Failure\_Reason NVARCHAR(255) = NULL: The failure reason to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If both parameters are NULL, returns all records from the table.
  + If both are provided, returns the single record matching the full composite key.
  + If only @Student\_ID is provided, returns all failure reasons for that student.
  + If only @Failure\_Reason is provided, returns all students associated with that failure reason.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Failed\_Students\_Update**

**Purpose:** Atomically updates a failure reason for a student. This is achieved by deleting the old reason and inserting the new one within a single transaction.

**Parameters:**

* @Student\_ID INT: The ID of the student (this ID does not change).
* @Old\_Failure\_Reason NVARCHAR(255): The existing failure reason to be replaced.
* @New\_Failure\_Reason NVARCHAR(255): The new failure reason to replace the old one.

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that all three parameters are non-NULL.
  + Verifies that the OLD record (@Student\_ID, @Old\_Failure\_Reason) exists.
  + Verifies that the NEW record (@Student\_ID, @New\_Failure\_Reason) does not already exist, preventing a PK violation.
* **Transaction:**
  + Begins a TRANSACTION.
  + DELETEs the OLD record.
  + INSERTs the NEW record.
  + COMMITs the transaction if both steps succeed.
* **Output:** Uses OUTPUT in both the DELETE and INSERT statements to return the [Old\_... ] and [New\_... ] data.
* **Smart Catch:** If any error occurs, the entire transaction is ROLLBACK, and a specific, user-friendly error message is returned.

**4. Failed\_Students\_Delete**

**Purpose:** Safely deletes a single, specific failure reason from a student. This procedure is designed to prevent accidental mass-deletes.

**Parameters:**

* @Student\_ID INT = NULL: The ID of the student.
* @Failure\_Reason NVARCHAR(255) = NULL: The failure reason to be deleted.

**Logic & Error Handling:**

* **Safety Pre-Checks:**
  + **CRITICAL:** Verifies that both @Student\_ID and @Failure\_Reason are non-NULL. If either is NULL, it returns an error and prevents *any* deletion.
  + Verifies that the specific record to be deleted actually exists, returning an error if not found.
* **Execution:**
  + Deletes the single record matching the full composite key.
  + Uses the OUTPUT clause to return the [Deleted\_Student\_ID] and [Deleted\_Failure\_Reason] to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., Error 547) if the delete is blocked by an unexpected foreign key constraint.

**Stored Procedure Documentation: [dbo].[Freelance\_Job]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Freelance\_Job] table.

**1. Freelance\_Job\_Insert**

**Purpose:** Adds a new freelance job record for a specific student.

**Parameters:**

* @Job\_ID INT: The unique Primary Key for the job.
* @Student\_ID INT: The ID of the student (Foreign Key to [Student]).
* @Job\_Earn DECIMAL(12, 2): The amount earned (required).
* @Job\_Date DATE: The date of the job (required).
* @Job\_Site NVARCHAR(255) = NULL: The website or source of the job (optional).
* @Description NVARCHAR(1000) = NULL: A description of the job (optional).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that @Job\_ID, @Student\_ID, @Job\_Earn, and @Job\_Date are non-NULL.
  + Verifies that the @Job\_ID does not already exist (Primary Key violation).
  + Verifies that the @Student\_ID exists in the [Student] table (Foreign Key violation).
* **Execution:**
  + Inserts the new job record.
  + Uses the OUTPUT clause to return all [Inserted\_...] columns to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK, 515 for NOT NULL) and returns a clear, user-friendly error message.

**2. Freelance\_Job\_Select**

**Purpose:** Retrieves one or more freelance job records based on filter criteria.

**Parameters:**

* @Job\_ID INT = NULL: The Primary Key of the job to retrieve.
* @Student\_ID INT = NULL: The ID of the student to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If @Job\_ID is provided, returns the single record for that job (highest priority).
  + If @Job\_ID is NULL and @Student\_ID is provided, returns all jobs for that student.
  + If both parameters are NULL, returns all records from the table.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Freelance\_Job\_Update**

**Purpose:** Updates all fields for an existing freelance job record.

**Parameters:**

* @Job\_ID INT: The Primary Key of the job to update.
* @Student\_ID INT: The (potentially new) ID of the student.
* @Job\_Earn DECIMAL(12, 2): The (potentially new) amount earned.
* @Job\_Date DATE: The (potentially new) date of the job.
* @Job\_Site NVARCHAR(255) = NULL: The (potentially new) job site.
* @Description NVARCHAR(1000) = NULL: The (potentially new) description.

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that the record with @Job\_ID exists.
  + Verifies that @Student\_ID, @Job\_Earn, and @Job\_Date are non-NULL.
  + Verifies that the @Student\_ID exists in the [Student] table.
* **Execution:**
  + Updates the record matching the @Job\_ID.
  + Uses the OUTPUT clause to return both the [Old\_... ] (from deleted) and [New\_... ] (from inserted) values for all columns.
* **Smart Catch:** Catches and reports specific SQL errors, just like the Insert procedure.

**4. Freelance\_Job\_Delete**

**Purpose:** Deletes a single freelance job or all jobs from the table.

**Parameters:**

* @Job\_ID INT = NULL: The Primary Key of the job to be deleted.

**Logic & Error Handling:**

* **Conditional Delete:**
  + **By Full PK:** If @Job\_ID is provided, it deletes the single matching record.
    - It checks if the record EXISTS first and returns an error if not found.
    - It uses OUTPUT to return all [Deleted\_... ] columns.
  + **Delete All:** If @Job\_ID is NULL, it deletes *all* records from the table.
    - It attempts to TRUNCATE for performance but will DELETE if the table is referenced by other foreign keys.
    - It prints a warning and success message to the console.
* **Smart Catch:** Catches and reports FK violations (Error 547) or other unexpected errors.

**Stored Procedure Documentation: [dbo].[Group]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Group] table.

**1. Group\_Insert**

**Purpose:** Creates a new group record, linking an intake, branch, and track.

**Parameters:**

* @Group\_ID INT: The unique Primary Key for the group.
* @Intake\_ID INT: The ID of the intake (Foreign Key to [Intake]).
* @Branch\_ID INT: The ID of the branch (Foreign Key to [Branch]).
* @Track\_ID INT: The ID of the track (Foreign Key to [Track]).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that @Group\_ID, @Intake\_ID, @Branch\_ID, and @Track\_ID are all non-NULL.
  + Verifies that the @Group\_ID does not already exist (Primary Key violation).
  + Verifies that all three Foreign Keys (@Intake\_ID, @Branch\_ID, @Track\_ID) exist in their respective parent tables.
* **Execution:**
  + Inserts the new group record.
  + Uses the OUTPUT clause to return all [Inserted\_...] columns to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK, 515 for NOT NULL) and returns a clear, user-friendly error message.

**2. Group\_Select**

**Purpose:** Retrieves one or more group records based on flexible filter criteria.

**Parameters:**

* @Group\_ID INT = NULL: The Primary Key of the group to retrieve.
* @Intake\_ID INT = NULL: The ID of the intake to filter by.
* @Branch\_ID INT = NULL: The ID of the branch to filter by.
* @Track\_ID INT = NULL: The ID of the track to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If @Group\_ID is provided, returns the single record for that group (highest priority).
  + If @Group\_ID is NULL, returns all records matching the combination of any provided FKs (e.g., all groups in a specific intake and branch).
  + If all parameters are NULL, returns all records from the table.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Group\_Update**

**Purpose:** Updates the associated intake, branch, and track for an existing group.

**Parameters:**

* @Group\_ID INT: The Primary Key of the group to update.
* @Intake\_ID INT: The (potentially new) ID of the intake.
* @Branch\_ID INT: The (potentially new) ID of the branch.
* @Track\_ID INT: The (potentially new) ID of the track.

**Logic & Error Handling:**

* **Note:** This procedure performs a standard UPDATE. It does *not* use the DELETE+INSERT pattern, as the Group\_ID (a surrogate key) is not being changed.
* **Pre-Checks:**
  + Verifies that the record with @Group\_ID exists.
  + Verifies that @Intake\_ID, @Branch\_ID, and @Track\_ID are non-NULL.
  + Verifies that all three (potentially new) Foreign Keys exist in their parent tables.
* **Execution:**
  + Updates the record matching the @Group\_ID.
  + Uses the OUTPUT clause to return both the [Old\_... ] (from deleted) and [New\_... ] (from inserted) values for all columns.
* **Smart Catch:** Catches and reports specific SQL errors, just like the Insert procedure.

**4. Group\_Delete**

**Purpose:** Intended to delete group records.

**Parameters:**

* @Group\_ID INT = NULL: The Primary Key of the group.
* @Intake\_ID INT = NULL: (Parameter is present but not used in logic).
* @Branch\_ID INT = NULL: (Parameter is present but not used in logic).
* @Track\_ID INT = NULL: (Parameter is present but not used in logic).

**Logic & Error Handling:**

* **Note on Current Logic:** As written, this procedure contains conflicting logic that prevents any delete operations.
  + The procedure first checks if @Group\_ID IS NULL and returns an error.
  + It then checks if @Group\_ID IS NOT NULL and *also* returns an error.
  + Due to these initial checks, the DELETE and DELETE ALL logic blocks within the procedure are unreachable, and no records can be deleted.
* **Smart Catch:** Includes a CATCH block to handle potential FK violations (Error 547) if the delete logic were reachable.

**Stored Procedure Documentation: [dbo].[Instructor]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Instructor] table.

**1. Instructor\_Insert**

**Purpose:** Adds a new instructor record to the table.

**Parameters:**

* @Instructor\_ID INT: The unique Primary Key for the instructor.
* @Instructor\_Fname NVARCHAR(50): The instructor's first name (required).
* @Instructor\_Lname NVARCHAR(50): The instructor's last name (required).
* @Instructor\_Gender NVARCHAR(10) = NULL: (e.g., 'Male', 'Female').
* @Instructor\_Birthdate DATE = NULL: The instructor's birthdate.
* @Instructor\_Marital\_Status NVARCHAR(50) = NULL: (e.g., 'Single', 'Married').
* @Instructor\_Salary INT = NULL: The instructor's salary.
* @Instructor\_Contract\_Type NVARCHAR(50) = NULL: (e.g., 'Full-Time', 'Part-Time').
* @Instructor\_Email NVARCHAR(150) = NULL: The instructor's email address.
* @Department\_ID INT = NULL: The ID of the department (Foreign Key to [Department]).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that @Instructor\_ID, @Instructor\_Fname, and @Instructor\_Lname are non-NULL.
  + Verifies that the @Instructor\_ID does not already exist (Primary Key violation).
  + Verifies that the @Department\_ID exists in the [Department] table, *if* one is provided.
* **Execution:**
  + Inserts the new instructor record.
  + Uses the OUTPUT clause to return all [Inserted\_...] columns to confirm success.
* **Smart Catch:**
  + Catches specific SQL errors (e.g., 547 for FK, 2627 for PK).
  + **Critically,** this block also catches all CHECK constraint violations (e.g., invalid gender, salary below 8000, or age violations) and returns a clear, user-friendly error message.

**2. Instructor\_Select**

**Purpose:** Retrieves one or more instructor records based on flexible filter criteria.

**Parameters:**

* @Instructor\_ID INT = NULL: The Primary Key of the instructor.
* @Department\_ID INT = NULL: The ID of the department to filter by.
* @Instructor\_Email NVARCHAR(150) = NULL: The email address to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If @Instructor\_ID is provided, returns the single record for that instructor (highest priority).
  + If @Instructor\_Email is provided, returns the single record for that email.
  + If only @Department\_ID is provided, returns all instructors in that department.
  + If all parameters are NULL, returns all records from the table.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Instructor\_Update**

**Purpose:** Updates all fields for an existing instructor record.

**Parameters:**

* (All parameters are identical to the Instructor\_Insert procedure).

**Logic & Error Handling:**

* **Note:** This procedure performs a standard UPDATE. It does *not* use the DELETE+INSERT pattern, as the Instructor\_ID (a surrogate key) is not being changed.
* **Pre-Checks:**
  + Verifies that the record with @Instructor\_ID exists.
  + Verifies that @Instructor\_Fname and @Instructor\_Lname are non-NULL.
  + Verifies that the @Department\_ID exists in the [Department] table, *if* one is provided.
* **Execution:**
  + Updates the record matching the @Instructor\_ID.
  + Uses the OUTPUT clause to return both the [Old\_... ] (from deleted) and [New\_... ] (from inserted) values for all columns.
* **Smart Catch:** Catches and reports specific SQL errors, including CHECK constraint violations, just like the Insert procedure.

**4. Instructor\_Delete**

**Purpose:** Intended to delete instructor records.

**Parameters:**

* @Instructor\_ID INT = NULL: The Primary Key of the instructor.

**Logic & Error Handling:**

* **Note on Current Logic:** As written, this procedure contains conflicting logic that prevents any delete operations.
  + The procedure first checks if @Instructor\_ID IS NULL and returns an error.
  + The ELSE block (which would handle @Instructor\_ID IS NOT NULL) is commented out, but the DELETE ALL logic that follows is unreachable because the first IF condition (@Instructor\_ID IS NULL) will always be false for it to be reached.
  + **Correction:** The ELSE block for "Delete All Records" is unreachable. The IF @Instructor\_ID IS NOT NULL block correctly handles deletion by ID, but the IF @Instructor\_ID IS NULL block *prevents* the "Delete All" logic from ever running.
* **Smart Catch:** Includes a CATCH block to handle potential FK violations (Error 547, e.g., if the instructor is still linked to a course) if the delete logic were reachable.

**Stored Procedure Documentation: [dbo].[Instructor\_Course]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Instructor\_Course] table, which links instructors to the courses they teach.

**1. Instructor\_Course\_Insert**

**Purpose:** Assigns an instructor to a specific course by creating a link record.

**Parameters:**

* @Instructor\_ID INT: The Primary Key of the instructor (Foreign Key to [Instructor]).
* @Course\_ID INT: The Primary Key of the course (Foreign Key to [Course]).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that both @Instructor\_ID and @Course\_ID are non-NULL.
  + Verifies that the composite key (Instructor\_ID, Course\_ID) does not already exist (Primary Key violation).
  + Verifies that the @Instructor\_ID exists in the [Instructor] table.
  + Verifies that the @Course\_ID exists in the [Course] table.
* **Execution:**
  + Inserts the new link record.
  + Uses the OUTPUT clause to return the [Inserted\_Instructor\_ID] and [Inserted\_Course\_ID] to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK, 515 for NOT NULL) and returns a clear, user-friendly error message.

**2. Instructor\_Course\_Select**

**Purpose:** Retrieves instructor-course links based on flexible filter criteria.

**Parameters:**

* @Instructor\_ID INT = NULL: The ID of the instructor to filter by.
* @Course\_ID INT = NULL: The ID of the course to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If both @Instructor\_ID and @Course\_ID are provided, returns the single record for that specific link.
  + If only @Instructor\_ID is provided, returns all courses for that instructor.
  + If only @Course\_ID is provided, returns all instructors for that course.
  + If all parameters are NULL, returns all records from the table.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Instructor\_Course\_Update**

**Purpose:** Updates an existing instructor-course link. This is achieved by atomically deleting the old link and inserting a new one within a single transaction.

**Parameters:**

* @Old\_Instructor\_ID INT: The existing instructor ID of the link.
* @Old\_Course\_ID INT: The existing course ID of the link.
* @New\_Instructor\_ID INT: The (potentially new) instructor ID.
* @New\_Course\_ID INT: The (potentially new) course ID.

**Logic & Error Handling:**

* **Note:** This procedure uses the DELETE+INSERT pattern, which is necessary when updating any part of a composite primary key.
* **Pre-Checks:**
  + Verifies that all four parameters are non-NULL.
  + Verifies that the old record (@Old\_Instructor\_ID, @Old\_Course\_ID) exists.
  + Verifies that the new record (@New\_Instructor\_ID, @New\_Course\_ID) does *not* already exist (PK violation).
  + Verifies that the @New\_Instructor\_ID exists in the [Instructor] table.
  + Verifies that the @New\_Course\_ID exists in the [Course] table.
* **Execution:**
  + Runs inside a TRANSACTION to ensure all-or-nothing execution.
  + DELETEs the old record, returning [Old\_...] columns via OUTPUT.
  + INSERTs the new record, returning [New\_...] columns via OUTPUT.
  + COMMITs the transaction.
* **Smart Catch:** Catches and reports specific SQL errors. If an error occurs, it ROLLBACKs the transaction.

**4. Instructor\_Course\_Delete**

**Purpose:** Deletes an instructor-course link.

**Parameters:**

* @Instructor\_ID INT = NULL: The instructor ID of the link.
* @Course\_ID INT = NULL: The course ID of the link.

**Logic & Error Handling:**

* **Note on Current Logic:** As written, this procedure enforces a "Safe Delete" pattern.
  + The first IF check (IF @Instructor\_ID IS NULL OR @Course\_ID IS NULL) returns an error if the full composite key is not provided.
  + This check effectively makes the DELETE ALL (ELSE IF ... IS NULL AND ... IS NULL) and "partial key error" (ELSE) blocks unreachable.
  + **Effective Behavior:** The procedure will **only** delete a record if the full, non-NULL composite key is provided, and will return an error otherwise.
* **By Full PK:**
  + Checks if the record EXISTS first and returns an error if not found.
  + DELETEs the specific record.
  + Uses OUTPUT to return the [Deleted\_Instructor\_ID] and [Deleted\_Course\_ID].
* **Smart Catch:** Catches and reports FK violations (Error 547) or other unexpected errors.

**Stored Procedure Documentation: [dbo].[Instructor\_Phone]**

This document details the CRUD (Create, Read, Update, Delete) stored procedures for managing the [dbo].[Instructor\_Phone] table, which stores contact numbers for instructors.

**1. Instructor\_Phone\_Insert**

**Purpose:** Adds a new phone number record for a specific instructor.

**Parameters:**

* @Instructor\_ID INT: The Primary Key of the instructor (Foreign Key to [Instructor]).
* @Phone NVARCHAR(20): The phone number (part of the composite Primary Key).

**Logic & Error Handling:**

* **Pre-Checks:**
  + Verifies that both @Instructor\_ID and @Phone are non-NULL.
  + Verifies that the composite key (Instructor\_ID, Phone) does not already exist (Primary Key violation).
  + Verifies that the @Instructor\_ID exists in the [Instructor] table (Foreign Key violation).
* **Execution:**
  + Inserts the new phone record.
  + Uses the OUTPUT clause to return the [Inserted\_Instructor\_ID] and [Inserted\_Phone] to confirm success.
* **Smart Catch:** Catches specific SQL errors (e.g., 547 for FK, 2627 for PK, 515 for NOT NULL) and returns a clear, user-friendly error message.

**2. Instructor\_Phone\_Select**

**Purpose:** Retrieves one or more instructor phone records based on flexible filter criteria.

**Parameters:**

* @Instructor\_ID INT = NULL: The ID of the instructor to filter by.
* @Phone NVARCHAR(20) = NULL: The phone number to filter by.

**Logic & Error Handling:**

* **Consolidated Select:**
  + If both @Instructor\_ID and @Phone are provided, returns the single record for that specific link.
  + If only @Instructor\_ID is provided, returns all phone numbers for that instructor.
  + If only @Phone is provided, returns all instructors associated with that phone number.
  + If all parameters are NULL, returns all records from the table.
* **Error Handling:** Uses a standard TRY...CATCH block for any unexpected query errors.

**3. Instructor\_Phone\_Update**

**Purpose:** Updates an existing instructor phone number link. This is achieved by atomically deleting the old link and inserting a new one within a single transaction.

**Parameters:**

* @Old\_Instructor\_ID INT: The existing instructor ID of the link.
* @Old\_Phone NVARCHAR(20): The existing phone number of the link.
* @New\_Instructor\_ID INT: The (potentially new) instructor ID.
* @New\_Phone NVARCHAR(20): The (potentially new) phone number.

**Logic & Error Handling:**

* **Note:** This procedure uses the DELETE+INSERT pattern, which is necessary when updating any part of a composite primary key.
* **Pre-Checks:**
  + Verifies that all four parameters are non-NULL.
  + Verifies that the old record (@Old\_Instructor\_ID, @Old\_Phone) exists.
  + Verifies that the new record (@New\_Instructor\_ID, @New\_Phone) does *not* already exist (PK violation), unless the old and new are identical.
  + Verifies that the @New\_Instructor\_ID exists in the [Instructor] table.
* **Execution:**
  + Runs inside a TRANSACTION to ensure all-or-nothing execution.
  + DELETEs the old record, returning [Old\_...] columns via OUTPUT.
  + INSERTs the new record, returning [New\_...] columns via OUTPUT.
  + COMMITs the transaction.
* **Smart Catch:** Catches and reports specific SQL errors. If an error occurs, it ROLLBACKs the transaction.

**4. Instructor\_Phone\_Delete**

**Purpose:** Deletes a specific phone number record for an instructor.

**Parameters:**

* @Instructor\_ID INT = NULL: The instructor ID of the link.
* @Phone NVARCHAR(20) = NULL: The phone number of the link.

**Logic & Error Handling:**

* **Note on Current Logic:** As written, this procedure enforces a "Safe Delete" pattern.
  + The first IF check (IF @Instructor\_ID IS NULL OR @Phone IS NULL) returns an error if the full composite key is not provided.
  + **Effective Behavior:** The procedure will **only** delete a record if the full, non-NULL composite key is provided, and will return an error otherwise.
* **By Full PK:**
  + Checks if the record EXISTS first and returns an error if not found.
  + DELETEs the specific record.
  + Uses OUTPUT to return the [Deleted\_Instructor\_ID] and [Deleted\_Phone].
* **Smart Catch:** Catches and reports FK violations (Error 547) or other unexpected errors.