Unit 5: Joins, Sub Queries Views and Transactions

Pratian Technologies (India) Pvt. Ltd.
www.pratian.com

PRATIAN
TECHNOLOGIES

Overview

- What are Joins?
- Types of Joins
  - Cross join
  - Inner join
  - Left outer join
  - Right outer join
  - Full Outer Join
- Unions
- SQL Functions
  - String
  - Date Time
  - Mathematical
  - Aggregate

PRATIAN
TECHNOLOGIES

Overview

- What are subqueries?
- Types of subqueries
  - Normal
  - Correlated
- Predicates with subqueries
- Restrictions with subqueries
- Views
- Transactions
  - What are transactions
  - Transaction control
  - Commit
  - Rollback

JOINS

- JOIN is a query that combines data from more than one table by means of a single statement
- Joining is done in SQL by specifying the tables to be joined in the FROM clause
- Most join queries contain WHERE conditions that compare two columns, each from a different table. Such a condition is called join condition

CROSS JOIN

- A cross-join between two tables takes the data from each row in table1 and joins it to the data from each row in table2.

- Example – To select students with their course information
  - mysql > SELECT s.Name, c.Course
    -> FROM Students s, Courses c

CROSS JOIN

- Students Table

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	2

- Courses Table

CourseId	Course
1	Basic SQL
2	Excel

Name	Course
Krishna S	Basic SQL
Krishna S	Excel
Raghav V	Basic SQL
Raghav V	Excel

EQUI JOIN

- An equi join is a join condition containing an equality operator
- Is also called inner join or simple join
- In the equi-join the comparison we are making between two columns is that they match the same value. We can use this method to select certain fields from both tables and only the correct rows will be joined together.
- Example – To select students with their course information
  - SELECT s.Name, c.Course
    FROM Students s, Courses c
    WHERE s.CourseId = c.CourseId
    Or
  - SELECT s.Name, c.Course
    FROM Students s INNER JOIN Courses c
    ON s.CourseId = c.CourseId

EQUI JOIN

- Students Table

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	2

- Courses Table

CourseId	Course
1	Basic SQL
2	Excel

- Result

Name	Course
Krishna S	Basic SQL
Raghav V	Excel

SELF JOIN

- A self join is a join of a table to itself
- The table appears twice in the FROM clause and is followed by table aliases, that qualify table names in the join condition
- The table rows are combined and the rows which satisfy the condition are returned
- To get names of all students with referred by student names
  - SELECT s.Name as StudentName, s.Age, r.Name as ReferredBy
    FROM Students s, Students r
    WHERE s.ReferredById = r.StudentId

SELF JOIN

- Students

StudentId	Name	Age	CourseId	ReferredById
1001	Krishna S	18	1	NULL
1002	Raghav V	19	2	1001

- Result

Student Name	Age	ReferredBy
Raghav V	19	Krishna

LEFT OUTER JOIN

- A left outer join extends the result of a simple join
- Returns all rows that satisfy the join conditions and those rows from left table for which no rows from the other satisfy the join condition
- Example: To get all students information, even those without course information
  - SELECT s.Name, s.Age, c.CourseId, c.Course
    FROM Students s
    LEFT OUTER JOIN Courses c
    ON s.CourseId = c.CourseId

LEFT OUTER JOIN

- Students

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	NULL
1003	Veena N	18	2

- Courses

CourseId	Course
1	Basic SQL
2	Excel

- Result

Name	Age	CourseId	Course
Krishna S	18	1	Basic SQL
Raghav V	19	NULL	NULL
Veena N	18	2	Excel

RIGHT OUTER JOIN

- A right outer join extends the result of a simple join
- Returns all rows that satisfy the join conditions and those rows from right table for which no rows from the other satisfy the join condition
- Example: To get all courses with their students, even those courses which doesn't have students assigned
  - SELECT s.Name, s.Age, c.CourseId, c.Course
    FROM Students s
    RIGHT OUTER JOIN Courses c
    ON s.CourseId = c.CourseId

RIGHT OUTER JOIN

- Students

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	NULL
1003	Veena N	18	1

- Courses

CourseId	Course
1	Basic SQL
2	Excel

- Result

Name	Age	CourseId	Course
Krishna S	18	1	Basic SQL
Veena N	18	1	Basic SQL
NULL	NULL	2	Excel

FULL OUTER JOIN

- A full outer join is union of LEFT and RIGHT outer join
- Returns all rows from right and left table. This includes both matching and non-matching data
- Example: To get all courses with their students with outer join
  - SELECT s.Name, s.Age, c.CourseId, c.Course
    FROM Students s
    FULL OUTER JOIN Courses c
    ON s.CourseId = c.CourseId

FULL OUTER JOIN

- Students

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	NULL
1003	Veena N	18	1

- Courses

CourseId	Course
1	Basic SQL
2	Excel

- Result

Name	Age	CourseId	Course
Krishna S	18	1	Basic SQL
Veena N	18	1	Basic SQL
Raghav V	19	NULL	NULL
NULL	NULL	2	Excel

UNION

- UNION is used to combine the result from multiple SELECT statements into a single result set
- There are few conditions to be kept in mind, when we use UNION
  - The number of columns in each SELECT statement has to be the same
  - The data type of the columns in the column list of the SELECT statement must be the same or at least convertible.

- Syntax:
  - **SELECT** statement
    UNION [**DISTINCT** | **ALL**]
    SELECT statement
    UNION [**DISTINCT** | **ALL]**

UNION

- By default the UNION removes all duplicated rows from the result set
- If you use UNION ALL explicitly, the duplicated rows will remain in the result set

- For Ex:
  - SELECT StudentId as Id, Name, Fees
    FROM Students
    WHERE Name LIKE '%K'
    UNION
    SELECT StudentId as Id, Name, Fees
    FROM Students
    WHERE Name LIKE '%A'

SQL FUNCTIONS

- There are few built in functions SQL Server provides
- We will be looking at few functions under the following categories
  - String
  - Mathematical
  - Date Time
  - Aggregate

SQL FUNCTIONS - String

- UPPER (string)
  - Converts all characters of a given string to upper case
  - SELECT UPPER(Name), JoinDate, Fees
    FROM Students
- LOWER(string)
  - Converts all characters of a given string to lower case
  - SELECT LOWER(Name), JoinDate, Fees
    FROM Students
- REVERSE(string)
  - Returns the reverse of a string
  - Select REVERSE('ABCD')
  - Output - DCBA

SQL FUNCTIONS - String

- SUBSTRING(expression, start, length)
  - Extracts substring from a given string considering the position and length provided
  - Select SUBSTRING('Krishna', 2, 5)
  - Output – rishn

- REPLACE(string expression, string pattern, string replacement)
  - Replaces a string after finding the pattern in the string provided
  - Arguments passed cannot be null
  - Select REPLAC('My first job', 'job', 'training')
  - Output – My first training

SQL FUNCTIONS - String

- LTRIM(string)
  - Returns string with leading space characters removed
  - SELECT LTRIM(' Krishna')
- RTRIM(string)
  - Similar to LTRIM. Removes trailing space characters
- LEN(string)
  - Returns the length of the string
  - *Note: This is only a partial list*

SQL FUNCTIONS - Mathematical

- ROUND(x, d) or ROUND(x)
  - Rounds the argument ***x*** to ***d*** decimal places
  - Select ROUND(150.222, 2)
  - Output – 150.22
- ABS(x)
  - Returns the absolute values
  - Select ABS(-32)
  - Output = 32
- SQRT(x)
  - Returns the square root a non-negative number
  - Select SQRT(4)
  - Output = 2

Note: This is only a partial list.

SQL FUNCTIONS – Date Time

- DATEPART(date part, date)
  - Returns the part of the date from a given date and time value
  - Select DATEPART(date, '11/08/2010 08:30:00')
  - Output : 11/08/2010
- GETDATE()
  - Returns the current date
  - Select GETDATE()
  - Output : 11/08/2010 08:30:00
- DAY(date)
  - Returns day of the month, in the range 1 – 31
  - SELECT DAY(GETDATE())
  - Output : 8

SQL FUNCTIONS - Date

- DATEADD(date part, number, date)
  - Function performs date arithmetic
  - SELECT DATEADD(DAY, 2, '11/08/2010 23:59:59')
  - Output - 11/10/2010 23:59:59

- MONTH(date)
  - To get the month of a given date
  - SELECT MONTH(GETDATE())
  - Output - 11

Note: This is only a partial list

SQL FUNCTIONS - Aggregate

- COUNT
  - Produces the number of rows query has selected
- AVG
  - Produces the average of all selected values of a given column
- MAX
  - Produces the largest of all selected values of a given column
- MIN
  - Produces the smallest of all selected values of a given column
- SUM
  - Produces the arithmetic sum of all selected values of a given column

SUBQUERIES

- A subquery is a SELECT statement within another statement
- Main advantages of subqueries are
  - Queries can be structured, so that it is possible to isolate each part of a statement
  - Provide alternate ways to perform operations that would otherwise require complex joins or unions
- Ex:
  - SELECT * FROM Table1 WHERE Column1 = (SELECT Column1 FROM Table2)
- A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns)

SUBQUERIES

- Used in either SELECT, WHERE or FROM clauses of an SQL statement
- In the WHERE clause subqueries can become a part of the following predicates
  - Comparison predicate
  - IN predicate
  - ANY or ALL predicate
  - EXISTS predicate
- Subqueries cannot have ORDER BY clause

NORMAL SUBQUERIES

- Does not need data from the outer query
- Subquery is evaluated only once
- Subquery generates values that are tested in the predicate of the outer query

- To list all students who have enrolled for Basic SQL course
  - SELECT Name, JoinDate

    FROM Students

    WHERE CourseId = (SELECT CourseId FROM Courses WHERE Course = 'BASIC SQL');

NORMAL SUBQUERIES

- Students

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	2
1003	Veena N	18	1

- Courses

CourseId	Course
1	Basic SQL
2	Excel

- Results

Name	Age	CourseId	Course
Krishna S	18	1	Basic SQL
Veena N	18	2	Basic SQL

CORRELATED SUBQUERY

- A correlated subquery uses any data from the FROM clause of the outer query
- The subquery is evaluated for each row of the outer query
- The subquery has to result in one value of the same data type as the left-hand side
- To get list of students and the number of courses they have joined

```
SELECT s.Name, s.Age, (Select count(*) FROM StudentCourses sc
WHERE sc.StudentId = s.StudentId) as NoOfCourses
FROM Students s
```

CORRELATED SUBQUERIES

- Students

StudentId	Name	Age	CourseId
1001	Krishna S	18	1
1002	Raghav V	19	2

- StudentCourses

StudentId	CourseId
1001	1
1001	2
1002	1

- Results

Name	Age	NoOfCourses
Krishna S	18	2
Raghav V	19	1

PREDICATES WITH SUBQUERIES

- Predicates that can be used with subqueries are
  - IN / NOT IN
  - EXISTS / NOT EXISTS
  - ALL / ANY

  - IN
    - SELECT DISTINCT(StudentId), Name, Fees

      FROM Students

      WHERE StudentId IN (Select st.StudentId FROM StudentCourses st)

PREDICATES WITH SUBQUERIES

- ALL
  - SELECT *

    FROM Students

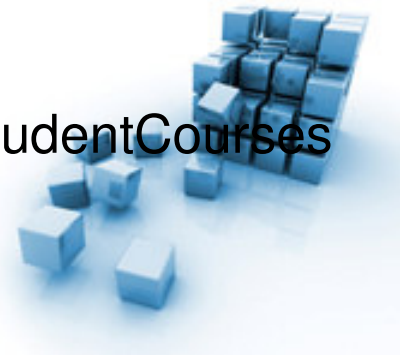
    WHERE Fees > ALL (SELECT Fees FROM Students WHERE CourseId = 1);
- ANY
  - SELECT *

    FROM Students

    WHERE Fess > ANY (SELECT Fees FROM Students WHERE CourseId = 1);
- EXISTS
  - SELECT s.Name, s.Fees

    FROM Students s

    WHERE EXISTS (SELECT st.StudentId FROM StudentCourses st WHERE st.StudentId = s.StudentId);



SUBQUERIES

- Can also be used in the FROM clause
- SELECT …FROM (subquery) [AS] name
  - The [AS] name clause is mandatory as every table in FROM clause should have a name

- Ex: Students who have paid more than Rs 1000 fees
  - SELECT StudentId, Name, Fees

    FROM (Select StudentId, Name, Fees From Students Where Fess >= 1000) as StudentsFeesGreater

SUBQUERIES - Restrictions

- A subquery's outer statement can be any one of: SELECT, INSERT, UPDATE, DELETE, SET, or DO.
- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms
  - DELETE FROM t WHERE ... (SELECT ... FROM t ...);
  - UPDATE t ... WHERE col = (SELECT ... FROM t ...);
  - {INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
- A subquery cannot have an ORDER BY statement
- *There are few more restrictions, but above mentioned are the most important*

QUESTION TIME

VIEWS

- Views are essentially saved SELECT queries that can themselves be queried
- It is stored as an object in the database
- They are used to provide easier access to normalized data
- Once you define a view then you can reference it like any other table in a database

VIEWS

- Views have the following benefits:
- *Security* -
  - Views can be made accessible to users while the underlying tables are not directly accessible. This allows the DBA to give users only the data they need, while protecting other data in the same table.
- *Simplicity* -
  - Views can be used to hide and reuse complex queries.
- *Column Name Simplification or Clarification* -
  - Views can be used to provide aliases on column names to make them more memorable and/or meaningful.
- *Stepping Stone* -
  - Views can provide a tiered approach in a "multi-level" query systematically.

VIEWS

- Syntax
  - CREATE VIEW view_name [(column_name[,column_name]….)]
    [WITH ENCRYPTION]
    AS select_statement [WITH CHECK OPTION]

- For Ex:
  - CREATE VIEW **StudentCourses** AS
    SELECT s.StudentId, s.Name, s.Fees, c.CourseId, c.Course
    FROM Students s, StudentCourses sc, Courses c
    WHERE s.StudentId = sc.StudentId and sc.CourseId = c.CourseId

- To query view
  - SELECT * FROM **StudentCourses**

VIEWS

- **Some restrictions imposed on views are given below :**
  - The view name must follow the rules for identifiers
  - The view name must not be the same as that of the base table
  - A view can be created if there is a SELECT permission on its base table.
  - A SELECT INTO statement cannot be used in view declaration statement.
  - The CREATE VIEW statement cannot be combined with other SQL statements in a single batch.

VIEWS

- ALTER VIEW
  - ALTER VIEW view_name [(column_name[,column_name]….)]
    [WITH ENCRYPTION]
    AS select_statement [WITH CHECK OPTION]

- For Ex:
  - ALTER VIEW **StudentCourses** AS

    SELECT s.StudentId, s.Name, s.Fees, c.Course

    FROM Students s, StudentCourses sc, Courses c

    WHERE s.StudentId = sc.StudentId

    and sc.CourseId = c.CourseId

- DROP VIEW
  - DROP VIEW [view_name]
  - DROP VIEW **StudentCourses**

TRANSACTIONS

- A transaction is a unit of work that may contain of one or more SQL statements
- A transaction is called atomic as the database modifications brought about by the SQL statements that constitute a transaction can be
  - Either made permanent to the database
  - Or, undone from the database
- The changes made to a table using INSERT, UPDATE or DELETE statements are not permanent till a transaction is marked complete

TRANSACTIONS

- A transaction will never be complete unless each individual operation within the group is successful
- If any operation within the transaction fails, the entire transaction will fail
- During a session, a transaction begins when the first SQL command (DDL or DML) is encountered and ends when one of the following occurs
  - A DDL is encountered
  - COMMIT/ROLLBACK statement is encountered
  - Logging off from the session
  - System failure

TRANSACTIONS – Properties [ACID]

- **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
- **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation:** enables transactions to operate independently of and transparent to each other.
- **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

TRANSACTIONS - BEGIN

- BEGIN TRANSACTION represents a point at which the data referenced by a connection is logically and physically consistent.

- If errors are encountered, all data modifications made after the BEGIN TRANSACTION can be rolled back to return the data to this known state of consistency

TRANSACTIONS - BEGIN

- BEGIN { TRAN | TRANSACTION }
  [{ transaction_name | @tran_name_variable }
  [WITH MARK ['description']]
  ]
  [;]
- transaction_name
  - Is the name assigned to the transaction
- @tran_name_variable
  - Is the name of a user-defined variable containing a valid transaction name
- WITH MARK ['description']
  - Specifies that the transaction is marked in the log. description is a string that describes the mark

TRANSACTIONS - BEGIN

- Example
  - DECLARE @TranName VARCHAR(20);
  - SET @TranName = 'My_Transaction';
  - BEGIN TRANSACTION @TranName;
    - DELETE FROM Students Where StudentId = 1001
    - COMMIT TRANSACTION @TranName;
  - GO
  - BEGIN TRANSACTION DeleteStudent
    - WITH MARK N'Deleting a Student';
    - DELETE FROM Students Where StudentId = 1001
    - COMMIT TRANSACTION DeleteStudent;
  - GO

TRANSACTIONS - COMMIT

- Committing a transaction makes permanent the changes resulting from all successful SQL statements in a transaction

- Syntax:
  - COMMIT { TRAN | TRANSACTION } [transaction_name | @tran_name_variable]]

    [;]

- AUTOCOMMIT option is available to execute COMMIT automatically whenever an INSERT, UPDATE or DELETE statement is executed
- Syntax:
  - SET IMPLICIT_TRANSACTIONS ON | OFF [Default]

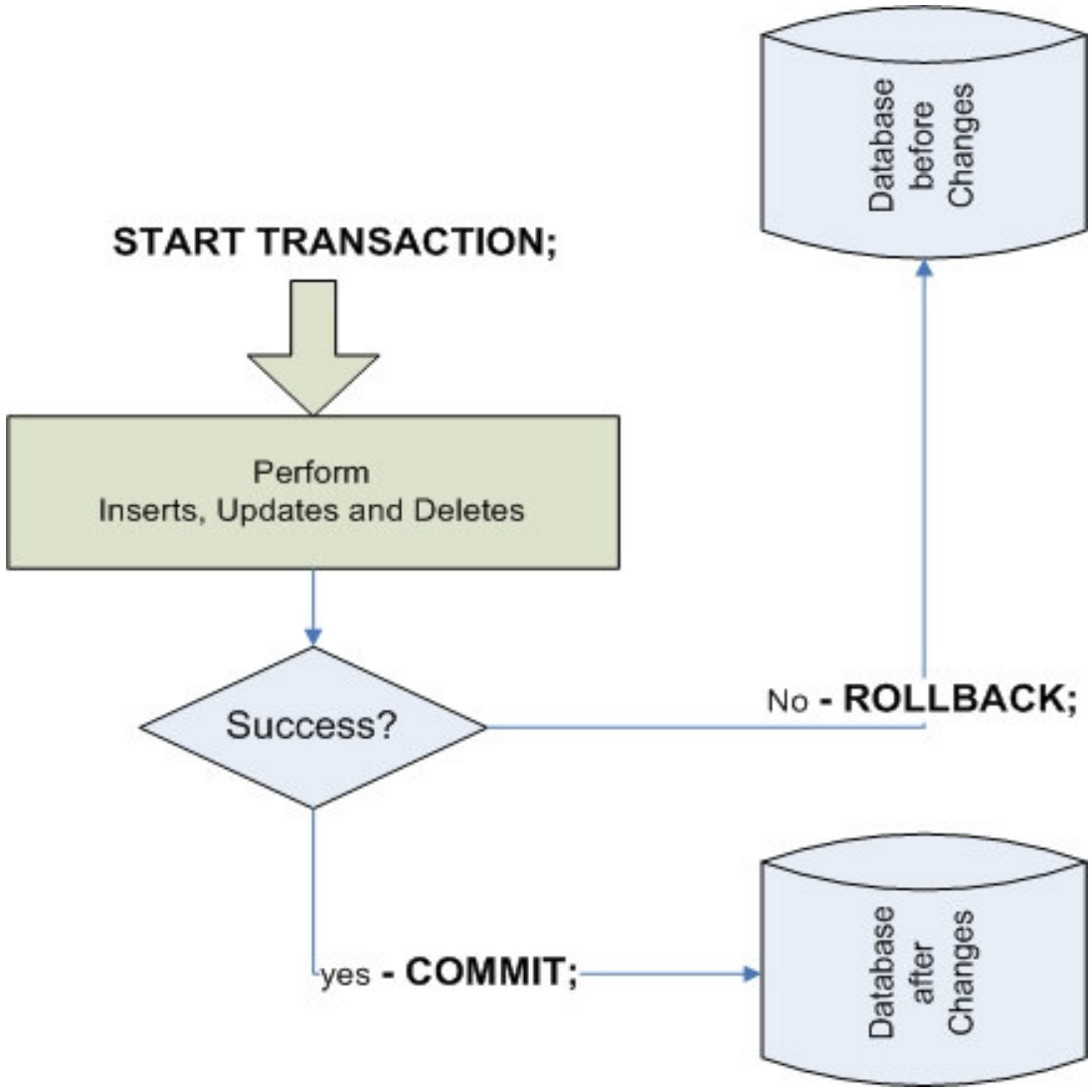
TRANSACTIONS - ROLLBACK

- Changes made to the database without COMMIT may be abandoned using the ROLLBACK statement
- When a transaction is rolled back, it is as if the transaction never occurred
- Syntax:
  - ROLLBACK { TRAN | TRANSACTION }

    [transaction_name | @tran_name_variable | savepoint_name | @savepoint_variable]

    [;]

TRANSACTIONS



TRANSACTIONS

- Syntax:
  - BEGIN TRANSACTION
    - Insert into table1…..
    - SAVE TRANSACTION firststep;
    - Update table2……..
    - IF @@TRANCOUNT= 0
      - ROLLBACK TRANSACTION firststep;
    - COMMIT TRANSACTION;

  GO

Question Time

Please try to limit the questions to the topics discussed during the session. Thank you

.