SQL 4: Joining Tables, Updating and Deleting Data, ACID, and Transactions

Databases and Interfaces

Matthew Pike & Yuan Yao

University of Nottingham Ningbo China (UNNC)

Overview

This Lecture

- In this lecture we will cover:
  - `JOIN`'ing tables in SQL
  - `UPDATE`'ing and `DELETE`'ing data
  - ACID Properties and Transactions

The Database Schema for this Lecture

```
CREATE TABLE Student(
    sID INTEGER PRIMARY KEY,
    firstName TEXT NOT NULL,
    lastName TEXT NOT NULL
);
```

```
CREATE TABLE Module(
    mCode CHAR(8) PRIMARY KEY,
    title TEXT NOT NULL,
    credits INTEGER NOT NULL
);
```

```
CREATE TABLE Grade(
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL,
    grade INTEGER NOT NULL,
    PRIMARY KEY (sID, mCode),
    FOREIGN KEY (sID)
        REFERENCES Student(sID),
    FOREIGN KEY (mCode)
        REFERENCES Module(mCode)
);
```

The Database Content for this Lecture

sID	firstName	lastName
1	John	Smith
2	Jane	Doe
3	Mary	Jones
4	Joe	Bloggs

Table 1: `Student` Table

mCode	title	credits
COMP1036	Fundamentals	20
COMP1048	Databases	10
COMP1038	Programming	20

Table 2: `Module` Table

sID	mCode	grade
1	COMP1036	35
1	COMP1048	50
2	COMP1048	65
2	COMP1038	70
3	COMP1036	35
3	COMP1038	65
6	COMP1038	55
6	COMP1099	68

Table 3: `Grade` Table

Joining Tables in SQL

JOINs

- JOINs can be used to combine rows from two or more tables, based on a related column between them.
- This is a very common operation in relational databases, as it allows you to link information between different tables.
- There are numerous types of JOINs, but the most common are:
    - CROSS JOIN
    - INNER JOIN
    - LEFT & RIGHT JOIN
    - NATURAL JOIN
    - FULL OUTER JOIN

CROSS JOIN

- The `CROSS JOIN` returns the Cartesian product of the sets of rows from the two tables.
    - This means that every row from the first table is combined with every row from the second table.
    - This also means that the resulting table will contain rows of data that are not related (nonsensical data).
- The syntax for a CROSS JOIN is:
    - `SELECT * FROM table1 CROSS JOIN table2;`
    - Which is equivalent to:
        - `SELECT * FROM table1, table2;`
- `CROSS JOIN` is rarely used in practice, as it can result in a very large number of rows.
    - We can constrain the number of rows returned by using a `WHERE` clause.

Example: CROSS JOIN

```
SELECT * FROM Student CROSS JOIN Module LIMIT 8;
```

sID	firstName	lastName	mCode	title	credits
1	John	Smith	COMP1036	Fundamentals	20
1	John	Smith	COMP1048	Databases	10
1	John	Smith	COMP1038	Programming	20
2	Jane	Doe	COMP1036	Fundamentals	20
2	Jane	Doe	COMP1048	Databases	10
2	Jane	Doe	COMP1038	Programming	20
3	Mary	Jones	COMP1036	Fundamentals	20
3	Mary	Jones	COMP1048	Databases	10

Table 4: The first 8 results of the `CROSS JOIN` of `Student` and `Module`

SELECT from Multiple Tables

- `SELECT` can be used with multiple tables, with table names separated by commas in the `FROM` clause.
  - `SELECT * FROM Student, Module;`
  - This is equivalent to a `CROSS JOIN` of the two tables.
- We can limit the columns returned by `SELECT` by specifying the table name before the column name.
  - `SELECT Student.sID, Module.mCode FROM Student, Module;`

Example: SELECT from Multiple Tables

```
SELECT
Student.sID,
Module.mCode,
grade --Not ambiguous
FROM
    Student, Grade, Module
WHERE
    Student.sID = Grade.sID
    AND
    Module.mCode = Grade.mCode;
```

sID	mCode	grade
1	COMP1036	35
1	COMP1048	50
2	COMP1048	65
2	COMP1038	70
3	COMP1036	35
3	COMP1038	65

Table 5: The SELECT from Multiple Tables

INNER JOIN

CROSS JOIN不可以用ON Clause

- The INNER JOIN returns only rows where the join condition is met.
- The join condition is specified in the ON clause.
    - `SELECT * FROM table1 INNER JOIN table2 ON table1.column1 = table2.column2;`

Example: INNER JOIN

```
SELECT
    Student.lastName,
    Grade.grade
FROM
    Student INNER JOIN Grade
    ON
    Student.sID = Grade.sID;
```

lastName	grade
Smith	35
Smith	50
Doe	65
Doe	70
Jones	35
Jones	65

Table 6: The INNER JOIN of Student and Grade

LEFT JOIN

- The `LEFT JOIN` returns all rows from the left table, and the matched rows from the right table.
- Any rows from the right table that do not have a match in the left table are returned with `NULL` values.
- Left joins are often used when you want to see all the rows from one table, even if there is no match in the other table.
- The syntax for a `LEFT JOIN` is:
    - `SELECT * FROM table1 LEFT JOIN table2 ON condition;`

LEFT JOIN Example

```sql
SELECT
    Student.sID,
    Student.lastName AS "Last",
    Module.mCode AS "Module",
    Grade.grade as 'Grade'
FROM
    Student LEFT JOIN Grade
    ON
    Student.sID = Grade.sID
    LEFT JOIN Module
    ON
    Grade.mCode = Module.mCode;
```

sID	Last	Module	Grade
1	Smith	COMP1036	35
1	Smith	COMP1048	50
2	Doe	COMP1038	70
2	Doe	COMP1048	65
3	Jones	COMP1036	35
3	Jones	COMP1038	65
4	Bloggs	NA	NA

Table 7: The `LEFT JOIN` of `Student` and `Grade`. Note the final row.

RIGHT JOIN

! `RIGHT JOIN` support in SQLite

Support for `RIGHT JOIN` is only available in SQLite version 3.39.0 and above.

- The `RIGHT JOIN` returns all rows from the right table, and the matched rows from the left table.
- Any rows from the left table that do not have a match in the right table are returned with `NULL` values.
- The syntax for a `RIGHT JOIN` is:
  - `SELECT * FROM table1 RIGHT JOIN table2 ON condition;`

RIGHT JOIN Example

```
SELECT
    Student.sID,
    Student.lastName AS "Last",
    Module.mCode AS "Module",
    Grade.grade as 'Grade'
FROM
    Student RIGHT JOIN Grade
    ON
    Student.sID = Grade.sID
    LEFT JOIN Module
    ON
    Grade.mCode = Module.mCode;
```

sID	Last	Module	Grade
1	Smith	COMP1036	35
1	Smith	COMP1048	50
2	Doe	COMP1038	70
2	Doe	COMP1048	65
3	Jones	COMP1036	35
3	Jones	COMP1038	65
NA	NA	COMP1038	55
NA	NA	NA	68

Table 8: The `RIGHT JOIN` of `Student` and `Grade`.

NATURAL JOIN

- The NATURAL JOIN returns all rows where the join condition is met.
- The syntax for a NATURAL JOIN is:
  - `SELECT * FROM table1 NATURAL JOIN table2;`
- The NATURAL JOIN can only be used if the columns to be joined have the same name in both tables.

Example: NATURAL JOIN

```
SELECT
    Student.sID,
    Student.lastName AS "Last",
    Module.mCode AS "Module",
    Grade.grade as 'Grade'
FROM
    Student
    NATURAL JOIN -- sID
    Grade
    NATURAL JOIN -- mCode
    Module;
```

sID	Last	Module	Grade
1	Smith	COMP1036	35
1	Smith	COMP1048	50
2	Doe	COMP1048	65
2	Doe	COMP1038	70
3	Jones	COMP1036	35
3	Jones	COMP1038	65

Table 9: The `NATURAL JOIN` of `Student`, `Grade`, and `Module`

FULL OUTER JOIN

! FULL OUTER JOIN support in SQLite

Support for **FULL OUTER JOIN** is only available in SQLite version 3.39.0 and above.

LEFT和RIGHT取交集，NULL都显示出来

- The **FULL OUTER JOIN** returns all rows from both tables, where the join condition is met.
- Any rows from the left table that do not have a match in the right table are returned with **NULL** values.
- Any rows from the right table that do not have a match in the left table are returned with **NULL** values.
- The syntax for a **FULL OUTER JOIN** is:
  - `SELECT * FROM table1 FULL OUTER JOIN table2 ON condition;`

Example: FULL OUTER JOIN

```
SELECT
    Student.sID,
    Student.lastName AS "Last",
    Module.mCode AS "Module",
    Grade.grade as 'Grade'
FROM
    Student FULL OUTER JOIN Grade
    ON
    Student.sID = Grade.sID
    FULL OUTER JOIN Module
    ON
    Grade.mCode = Module.mCode;
```

sID	Last	Module	Grade
1	Smith	COMP1036	35
1	Smith	COMP1048	50
2	Doe	COMP1038	70
2	Doe	COMP1048	65
3	Jones	COMP1036	35
3	Jones	COMP1038	65
4	Bloggs	NA	NA
NA	NA	COMP1038	55
NA	NA	NA	68

Table 10: The FULL OUTER JOIN of Student, Grade, and Module

Updating Data in SQL

UPDATE Statement

- The `UPDATE` statement is used to modify the existing records in a table.
- The syntax for the `UPDATE` statement is:
  - `UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;`
- The `WHERE` clause is optional.
  - If it is omitted, all records in the table will be updated.
- Within the `SET` clause, you can specify multiple columns and values.
- The `UPDATE` statement can reference column values from other columns in the same row.
  - For example, `UPDATE table SET column1 = column1 + 1;` will increment the value of `column1` by 1.

Example: UPDATE Statement

```sql
UPDATE Student
SET
    firstName = 'Johnathan',
    lastName = 'Creek'
WHERE sID = 1;
```

```sql
SELECT * FROM Student;
```

sID	firstName	lastName
1	Johnathan	Creek
2	Jane	Doe
3	Mary	Jones
4	Joe	Bloggs

Table 11: The Student table after UPDATE

Deleting Data with SQL

DELETE Statement

ℹ A Note on Refential Integrity

Remember, tables with foreign keys have a **refential integrity** constraint, which means that the DELETE statement may fail if the foreign key is referenced in another table, unless the CASCADE option is used. However, by default, SQLite does not enforce referential integrity. To enable it, we need to use the PRAGMA statement: `PRAGMA foreign_keys = ON;`.

- The DELETE statement is used to delete existing records in a table.
- The syntax for the DELETE statement is:
  - `DELETE FROM table_name WHERE condition;`
- The WHERE clause is optional.
  - If it is omitted, all records in the table will be deleted.
- The DELETE statement returns the number of rows that were deleted.

Example: DELETE Statement

```
DELETE FROM Student WHERE sID = 4;
```

```
DELETE FROM Grade WHERE sID = 6;
```

```
SELECT *
FROM Student
WHERE sID >= 3;
```

sID	firstName	lastName
3	Mary	Jones

Table 12: The Student table after DELETE

```
SELECT *
FROM Grade
WHERE sID >= 4;
```

sID	mCode	grade

Table 13: The Grade table after DELETE

Transactions

ACID Properties

- ACID is an acronym for the four properties of transactions:
    - **Atomicity**: All or nothing. Either all of the operations in a transaction are completed, or none of them are.
    - **Consistency**: The database is always in a valid state.
    - **Isolation**: Transactions are isolated from each other.
    - **Durability**: Once a transaction has been committed, it will remain so, even in the event of a system failure.
- SQLite is ACID compliant and supports transactions.
- SQLite guarantees that all transactions are ACID compliant even if the transaction is interrupted by a power failure or system crash.

Transactions

- A transaction is a sequence of SQL statements that are treated as a single unit.
    - Either all of the statements are executed, or none of them are.
- Transactions are used to ensure that the database is in a consistent state after the transaction is completed.
    - For example, if a transaction updates two tables, and one of the updates fails, the database should be left in the same state as before the transaction was started.

Transaction Syntax

- The syntax for a transaction is:
  - `BEGIN TRANSACTION;`
  - `-- SQL statements`
  - `COMMIT;`
- The `BEGIN TRANSACTION` statement starts a transaction.
- The `COMMIT` statement commits the transaction, which means that the changes are saved to the database.
- If any of the SQL statements in the transaction fail, the `ROLLBACK` statement can be used to undo the changes.
  - `ROLLBACK;`

Example: Transactions

```sql
BEGIN TRANSACTION;
    INSERT INTO Student VALUES (4, 'John', 'Doe');
    INSERT INTO Student VALUES (5, 'Jane', 'Smith');
    INSERT INTO Student VALUES (6, 'John', 'Smith');

-- Commit the changes to the database:
COMMIT;

-- If you do not want to save the changes:
ROLLBACK;
```

References

Learning Resources

Online Tutorials

These are clickable links to the online tutorials:

- Join Operators
- `Update`
- `Delete`
- Transactions
- A Visual Explanation of SQL Joins

Textbooks and Documentation

- Chapter 5 and 22 of the Databases textbook.
- SQLite Transactions
- SQLite Joins

Reference Materials

Mohan, Chandrasekaran, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. 1992. “ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.” *ACM Transactions on Database Systems (TODS)* 17 (1): 94–162.