Relational Databases

Chapter No.	25
Status	Completed

- ▼ Structure of a Relational Database
    - A relational database has fixed schema
    - Record = Row
    - Field = Column

- ▼ Keys in a Relational Database
    - ▼ A candidate key is a minimal set of fields that can uniquely identify each record in a table
        - It should never be empty
    - ▼ A primary key is a candidate key that is most appropriate to become the main key for a table
        - Uniquely identifies each record in a table
        - Should not change over time
    - A secondary key is a candidate key that is not selected as a primary key
    - ▼ A composite key is a combination of two or more fields in a table that can be used to uniquely identify each record in a table
        - Uniqueness is only guaranteed when the fields are combined
        - When taken individually, the fields do not guarantee uniqueness
    - A foreign key is a field in one table that refers to the primary key in another table

- ▼ Data Redundancy
    - Data redundancy refers to the same data being stored more than once

- ▼ Data Dependency
    - ▼ Functional Dependency
        - Y is functionally dependent on X if for every valid instance of X, the value of X uniquely determines the value of Y
        - i.e. X → Y
    - ▼ Transitive Dependency
        - Z is transitively dependent on X if Y is functionally dependent on X, but X is not functionally dependent on Y, and Z is functionally dependent on Y
        - ▼ i.e. X → Z if:
            - X → Y
            - Y does not → X
            - Y → Z

- ▼ Normalisation
    - Normalisation is the process of organising the tables in a database to reduce data redundancy and prevent inconsistent data
    - ▼ Conditions for Normalisation
        - ▼ First Normal Form (1NF)
            - ▼ All columns must be atomic
                - i.e. the information cannot be broken down further
        - ▼ Second Normal Form (2NF)
            - The table should already be in 1NF
            - ▼ Every non-key attribute must be fully dependent on the entire primary key
                - i.e. no attribute can depend on part of the primary key only
        - ▼ Third Normal Form (3NF)
            - The table should already be in 2NF

        - The table should not have transitive dependencies

- ▼ Entity-Relationship (E-R) Diagram
    - ▼ An entity is a specific object of interest
        - Represented by rectangles
    - ▼ A relationship describes the link between two entities
        - Represented by the lines connecting two rectangles together
    - ▼ Types of Relationships
        - one-to-one
        - one-to-many
        - ▼ many-to-many
            - Usually decomposed into two (or more) one-to-many relationships
        - one (and only one)
        - zero or one
        - one or many
        - zero or many

- ▼ SQL Database Operations
    - Data Definition Language (DDL) defines database schemas
    - Data Manipulation Language (DML) is used to retrieve and modify data
    - Data Control Language (DCL) is used to control access to a database
    - Transaction Control Language (TCL) is used to manage changes to a database, usually at a transactional level
    - ▼ We only need to be able to understand the basic CRUD database operations

        CRUD in SQL

Operation	SQL Command

Operation	SQL Command
CREATE	INSERT
READ	SELECT
UPDATE	UPDATE
DELETE	DELETE

Creating & Manipulating a SQL Database

Data Definition Language (DDL)

CREATE

```
CREATE TABLE table_name(
  column1_name COLUMN1_TYPE COLUMN1_CONSTRAINTS,
  column2_name COLUMN2_TYPE COLUMN2_CONSTRAINTS,
  ...
  PRIMARY KEY (column1_name, column2_name, ...),
  FOREIGN KEY (column_name) REFERENCES table_name(column_name)
);
```

```
CREATE TABLE IF NOT EXISTS table_name(
  column1_name COLUMN1_TYPE COLUMN1_CONSTRAINTS,
  column2_name COLUMN2_TYPE COLUMN2_CONSTRAINTS,
  ...
  PRIMARY KEY (column1_name, column2_name, ...),
  FOREIGN KEY (column_name) REFERENCES table_name(column_name)
);
```

Field Types

- NULL
- REAL
- INTEGER
- TEXT

Field Constraints

- NOT NULL
- PRIMARY KEY
- AUTOINCREMENT

- UNIQUE

▼ DROP

```
DROP TABLE table_name;
```

▼ Data Manipulation Language (DML)

▼ INSERT

```
INSERT INTO table_name(column1_name, column2_name, ...)
VALUES(column1_value, column2_value, ...);
```

▼ SELECT

```
SELECT column1_name, column2_name, ...
FROM table_name
WHERE where_expression
ORDER BY order_expression <ASC/DESC>;
```

```
SELECT DISTINCT column1_name, column2_name, ...
FROM table_name
WHERE where_expression
ORDER BY order_expression <ASC/DESC>;
```

▼ UPDATE

```
UPDATE table_name SET
column1_name = column1_expression,
column2_name = column2_expression,
...
WHERE where_expression;
```

▼ DELETE

```
DELETE FROM table_name
WHERE where_expression;
```

▼ JOIN

▼ Inner join returns the Cartesian product of rows from the tables

    - i.e. it combines each row in the first table with each row in the second table

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table_name, table2_name
WHERE where_expression;
```

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table1_name
INNER JOIN table2_name ON join_expression;
```

- Left outer join takes into consideration all the records from one table and records from the other that meet the join conditions

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table1_name
LEFT OUTER JOIN table2_name ON join_expression;
```

- Aggregate Functions
    - COUNT( )
    - MAX( )
    - MIN( )
    - SUM( )
- Operators
    - Comparison Operators
        - =
        - !=
        - <
        - >
        - <=
        - >=
    - Logical Operators
        - AND

- OR
- IS
- IS NOT
- | | (string concatenation)

Arithmetic Operators

- +
- -
- *
- /
- %

Python & SQLite

Loading a Database

```
import sqlite3

connection = sqlite3.connect("database_name.db")

cursor = connection.cursor()

connection.close()
```

Executing SQL Statements

```
import sqlite3

connection = sqlite3.connect("database_name.db")

cursor = connection.cursor()

cursor.execute('''
                CREATE TABLE table_1 (
                column1_name INTEGER PRIMARY KEY AUTOINCREMENT
                column2_name TEXT NOT NULL)
                ''')

connection.commit()
connection.close()
```

Parameter Substitution

- Used to safely include data that is provided by the user
- The second argument of execute( ) is a tuple of values to fill in the placeholder "?" in SQL statement of execute( )
- The values in the tuple are substituted in the same order in which the placeholder "?" appear in the SQL statement

```
import sqlite3

connection = sqlite3.connect("database_name.db")

cursor = connection.cursor()

cursor.execute('''
                DELETE FROM table_1
                WHERE column1 > ? AND column_1 < ?
                ''', (2, 8))

connection.commit()
connection.close()
```

Retrieving Data from a Database

For Loop Method

- Each iteration of the cursor object returns a tuple of the columns in the current row

```
import sqlite3

connection = sqlite3.connect("database_name.db")

cursor = connection.cursor()

cursor.execute('''
              SELECT *
              FROM table_1
              ''')

for row in cursor:
  print(row[0])

connection.commit()
connection.close()
```

▼ Fetchone Method

- The fetchone( ) method advances the cursor to the next row
- Each iteration of the cursor object returns a tuple of the columns in the current row
- Calling it repeatedly will iterate through the selected rows until the cursor object reaches the end and returns None

```
import sqlite3

connection = sqlite3.connect("database_name.db")

cursor = connection.cursor()

cursor.execute('''
              SELECT *
              FROM table_1
              ''')

row = cursor.fetchone()
while row is not None:
  print(row[0])
  row = cursor.fetchone()

connection.commit()
connection.close()
```

▼ Fetchall Method

- The fetchall( ) method returns a list of tuples with each tuple containing the selected columns for a single row

```
import sqlite3

connection = sqlite3.connect("databse_name.db")

cursor = connection.cursor()

cursor.execute('''
              SELECT *
              FROM table_1
              ''')

rows = cursor.fetchall()
for row in rows:
  print(row[0])

connection.commit()
connection.close()
```

- .Row Class Method
    - Setting the connection object's row_factory attribute to the built-in sqlite3.Row class allows each row to be retrieved as a dictionary that maps column names to column values

```
import sqlite3

connection = sqlite3.connect("database_name")
connection.row_factory = sqlite3.Row

cursor = connection.cursor()

cursor.execute('''
              SELECT *
              FROM table_1
              ''')

for row in cursor:
  print(row["column_1"])

connection.commit()
connection.close()
```