Assignment 1

CCS 6344 Database And Cloud Security

Name	ID	Tutorial Section
Rohit A/L W.Sugathedasa	1211103400	TT6L
Akid Syazwan bin Nor Azman Shah	1211111238	TT2L
Teng Chay Xuan	1231300802	TT2L

Video Presentation Link: https://youtu.be/JCyT-Tpgfuc

GitHub Link: https://github.com/KidCuded/CCS6344_Assignment1_Group36

Task 1: Proposal for university sport equipment rental system

1.1 Objectives

- To allow MMU students an effortless interface to rent, browse, and return sports equipment.
- To enable admins to view rentals, manage returns, and update inventory in real-time.
- To enforce data confidentiality and security using SQL techniques such as prepared statements, hashing, and access control.
- To eliminate manual work by automating the administrative tasks of managing rental records, updating inventory, and tracking equipment status.

1.2 Proposed Design and Implementation

- User Register/Login
- User Dashboard (Browse, rent, return equipment's)
- Admin Dashboard (Add, Remove Equipment's, Manage rentals)

1.3 Proposed hardware and software to develop the application

1.3.1 Frontend: Html, CSS, JavaScript

- HTML will be used for the structure and organization of the web pages which includes login, registration, dashboards, and rental forms. The styling of such webpages is done with CSS which includes the arrangements, colors, fonts as well as the responsiveness of the user interface. The interactivity of the system is done by JavaScript.

1.3.2 Backend: Python, MSSQL

– Python, using the Flask framework, handles all server-side processing. This includes managing form submissions (such as login, registration, renting, returning, and adding equipment), maintaining user sessions to track login status and roles, performing CRUD (Create, Read, Update, Delete) operations on the Microsoft SQL Server database, and enforcing business rules—for example, preventing users from booking the same equipment at the same time. Flask serves as the backend web framework and works in coordination with a web server to respond to HTTP requests, manage dynamic

interactions, and deliver HTML, CSS, and JavaScript content to the client. Microsoft SQL Server is used to securely store and manage all application data.

1.4 System Design and Database Design

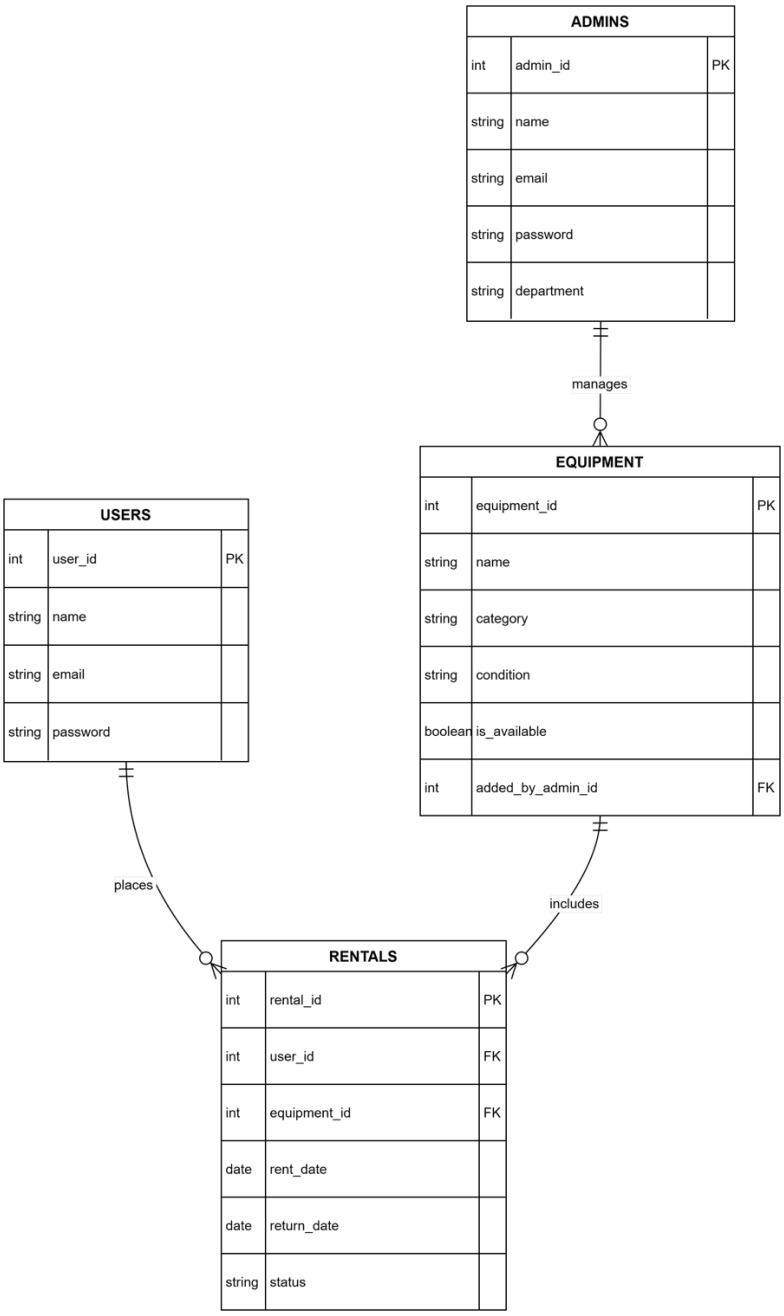

Figure 1.4.1 Database Design

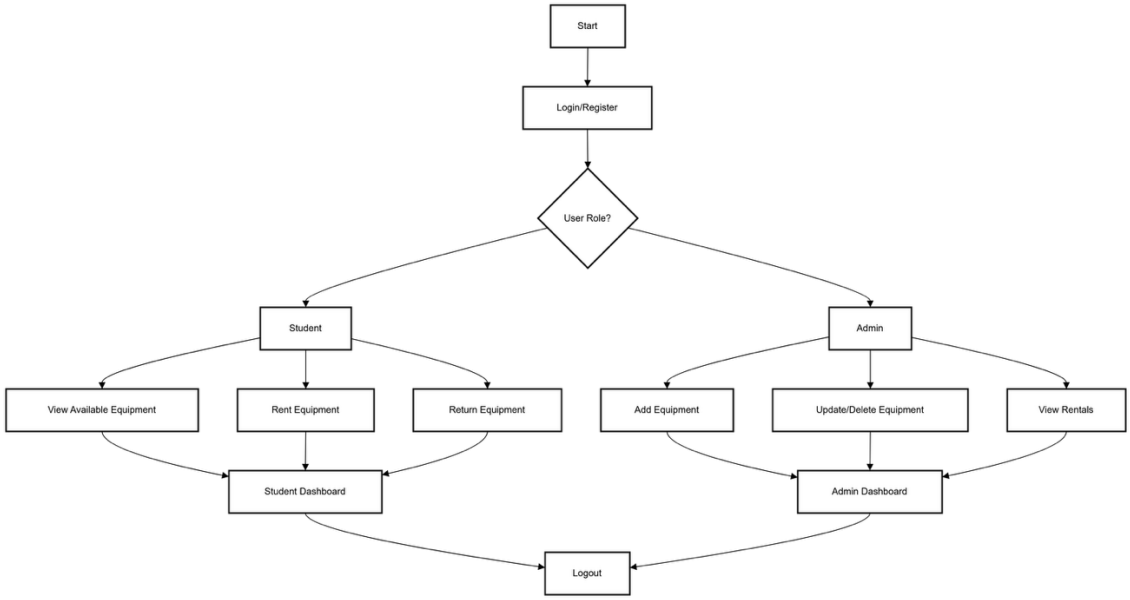

Figure 1.4.2 System Design

1.5 Database Security Using SQL

A.) Role Based Access Control

Students can only browse and rent, and admins can manage rentals and equipments which means students and admins have different roles

B.) Password Security

Passwords will be hashed, and login will be verified using the hash

C.) Input Validation

User inputs will be validated on both the client side using JavaScript and the server side using Python with Flask.

D.) Backup and Recovery

Data can be backed up using **Microsoft SQL Server's backup features** and transaction log backups to ensure data can be recovered in case of loss or failure.

Task 2: Implementation

2.1 Flask App Initiation

This code sets up a Flask app by importing necessary modules for web handling, database connection, password hashing, and input validation. It then creates the Flask application instance to manage web requests and routing.

```python
from flask import Flask,flash, render_template, request, redirect, url_for, session
import pyodbc, hashlib, re
from datetime import datetime, timedelta

app = Flask(__name__)
```

2.2 Database Connection

This code defines the connection string for connecting to a SQL Server database using the ODBC driver. It specifies the server, database name, and trusted connection settings. The get_db() function uses this string to establish and return a database connection with pyodbc, allowing the app to interact with the database.

```python
conn_str = (
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=DE_DRAGON\SQLEXPRESS;" # Change this if needed (e.g., IP address or hostname)
    "DATABASE=SportsInventory;"
    "Trusted_Connection=yes;"
)

def get_db():
    return pyodbc.connect(conn_str)

```

2.3 Login

This login route handles user authentication. When a POST request is made, it retrieves the email and password from the form, hashes the password using SHA-512, and checks the database for a matching user with the provided email and hashed password. If a user is found, their ID and role are saved in the session to track login status. Depending on the user's role, they are redirected to either the admin page or the user dashboard. If no match is found, an error message is shown, and the login page reloads. For GET requests, it simply renders the login form.

```python
@app.route('/login', methods=['GET', 'POST'])
def do_login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        hashed_password = hashlib.sha512(password.encode('utf-8')).digest()

        conn = get_db()
        cursor = conn.cursor()
        cursor.execute("SELECT id, role FROM Users WHERE email=? AND PasswordHash=?", (email, hashed_password))
        user = cursor.fetchone()
        conn.close()
        if user:
            user_id, role = user
            session['user_id'] = user_id
            session['role'] = role

            if role == 'admin':
                return redirect('/admin')
            else:
                return redirect('/dashboard')

        flash("Invalid email or password", "danger")
        return redirect('/login')
    return render_template('Login.html')

```

2.4 Register

This register route handles new user sign-ups. On a POST request, it collects and trims input data, then performs basic validations: ensuring the full name is provided, the student ID is alphanumeric, the email is in a valid format, and the password is at least 6 characters long. If any check fails, it flashes an error and redirects back to the registration form. The password is securely hashed using SHA-512 before storage. The route then checks the database to prevent duplicate email or student ID registrations. If the user is new, their details are inserted into the database, and they are redirected to the login page. For GET requests, it simply renders the registration form.

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        full_name = request.form['full_name'].strip()
        student_id = request.form['student_id'].strip()
        email = request.form['email'].strip()
        password = request.form['password']

        # Simple validations
        if not full_name:
            flash("Full name is required.")
            return redirect('/register')

        if not re.match(r'^[A-Za-z0-9]+$', student_id):
            flash("Student ID must be alphanumeric.")
            return redirect('/register')

        email_regex = r'^[^@]+@[^@]+\.[^@]+$'
        if not re.match(email_regex, email):
            flash("Invalid email address.")
            return redirect('/register')

        if len(password) < 6:
            flash("Password must be at least 6 characters.")
            return redirect('/register')
```

```python

        # Hash the password as binary
        hashed_password = hashlib.sha512(password.encode('utf-8')).digest()  # returns 64-byte binary

        conn = get_db()
        cursor = conn.cursor()

        # Check if user already exists by email or student_id
        cursor.execute("SELECT id FROM Users WHERE email = ? OR student_id = ?", (email, student_id))
        existing_user = cursor.fetchone()

        if existing_user:
            conn.close()
            return "Email or Student ID already registered"

        # Insert new user
        cursor.execute("""
            INSERT INTO Users (full_name, student_id, email, PasswordHash)
            VALUES (?, ?, ?, ?)
        """, (full_name, student_id, email, hashed_password))

        conn.commit()
        conn.close()

        return redirect('/login')

    return render_template('Register.html')
```

2.5 Admin and Add Equipment

This admin route controls access to the admin panel, allowing only logged-in users with a valid session. When an admin submits the form (POST), new equipment details (name, category, quantity) are inserted into the database, and a success message is shown. On every request, it retrieves all equipment and recent rental records by joining user, equipment, and rental tables, so the admin can view inventory and rental activity. The database connection is properly opened and closed to ensure smooth operation.

```python
@app.route('/admin', methods=['GET', 'POST'])
def admin_panel():
    # Only allow access if the user is logged in and is an admin
    if 'user_id' not in session:
        flash('Please log in to access the admin panel.', 'warning')
        return redirect(url_for('do_login'))

    conn = get_db()
    cursor = conn.cursor()

    # Adding Equipment
    if request.method == 'POST':
        name = request.form['EqName']
        category = request.form['category']
        quantity = request.form['quantity']

        cursor.execute(
            "INSERT INTO Equipment (name, category, quantity) VALUES (?, ?, ?)",
            (name, category, quantity)
        )
        conn.commit()
        flash('Equipment added successfully.', 'success')
```

```python
    # Fetch all equipment for display
    cursor.execute("SELECT * FROM Equipment")
    equipment = cursor.fetchall()

    # Fetch rental records
    cursor.execute("""
        SELECT TOP 20 U.full_name, U.student_id, E.name, R.quantity, R.date_rented, R.returned
        FROM Rentals R
        JOIN Users U ON R.user_id = U.id
        JOIN Equipment E ON R.equipment_id = E.id
        ORDER BY R.date_rented DESC
    """)
    records = cursor.fetchall()
    conn.close()

    return render_template('Admin.html', equipment=equipment, records=records)
```

2.6 Edit Equipment

This route handles updating existing equipment details. It receives the equipment ID and new values (name, category, quantity) from a form submission, then runs an SQL UPDATE query to modify the corresponding record in the database. After committing the changes, it closes the connection and redirects back to the admin panel. This lets admins efficiently edit inventory items.

```python
@app.route('/edit_equipment', methods=['POST'])
def edit_equipment():
    equipment_id = request.form['id']
    name = request.form['name']
    category = request.form['category']
    quantity = request.form['quantity']

    conn = get_db()
    cursor = conn.cursor()
    cursor.execute("""
        UPDATE Equipment
        SET name = ?, category = ?, quantity = ?
        WHERE id = ?
    """, (name, category, quantity, equipment_id))
    conn.commit()
    conn.close()
    return redirect('/admin')
```

2.7 Rent Equipment

This lets logged-in users rent equipment. It checks if the user is authenticated via session and retrieves the requested quantity. It then verifies there's enough stock available; if not, it returns an error. If sufficient, it inserts a new rental record with the rental and due dates, and updates the equipment quantity accordingly. After committing changes, it closes the connection and redirects the user to their dashboard. This ensures controlled, real-time tracking of equipment rentals and inventory.

```python
@app.route('/rent/<int:equipment_id>', methods=['POST'])
def rent_equipment(equipment_id):
    if 'user_id' not in session:
        return redirect('/')

    quantity = int(request.form.get('quantity', 1))
    user_id = session['user_id']
    today = datetime.now().date()
    due = today + timedelta(days=7)

    conn = get_db()
    cursor = conn.cursor()

    # Ensure enough stock is available
    cursor.execute("SELECT quantity FROM Equipment WHERE id = ?", (equipment_id,))
    available = cursor.fetchone()
    if not available or available[0] < quantity:
        conn.close()
        return "Not enough equipment available", 400

    # Add rental and update quantity
    cursor.execute("""
        INSERT INTO Rentals (user_id, equipment_id, date_rented, due_date, quantity)
        VALUES (?, ?, ?, ?, ?)
    """, (user_id, equipment_id, today, due, quantity))

    cursor.execute("UPDATE Equipment SET quantity = quantity - ? WHERE id = ?", (quantity, equipment_id))

    conn.commit()
    conn.close()
```

2.8 Return Equipment

This route handles returning rented equipment. It fetches the equipment_id for the given rental record, marks the rental as returned by updating the returned flag, and increments the equipment's available quantity by 1. After committing these changes, the database connection is closed and the user is redirected to their rentals page. This keeps rental records and inventory stock accurate.

```python
@app.route('/return/<int:rental_id>', methods=['POST'])
def return_rental(rental_id):
    conn = get_db()
    cursor = conn.cursor()

    # Get the equipment_id to increment quantity later
    cursor.execute("SELECT equipment_id FROM Rentals WHERE id = ?", (rental_id,))
    row = cursor.fetchone()
    if row:
        equip_id = row[0]
        cursor.execute("UPDATE Rentals SET returned = 1 WHERE id = ?", (rental_id,))
        cursor.execute("UPDATE Equipment SET quantity = quantity + 1 WHERE id = ?", (equip_id,))
        conn.commit()

    conn.close()
    return redirect('/myrentals')
```

Task 3: Threat Modelling

Stride Threat Modelling

Category	Threat	Description
Spoofing	Disguising oneself as a legitimate system user (student or admin).	The system verifies login details using hashed passwords ensuring authenticity and safety.
Tampering	An attacker tries to modify rental data or payment details in the system.	Role based access makes sure only admin can manage all this data
Repudiation	A user denies deleting any data	To ensure non-repudiation and simple user claim verification, actions are recorded in an audit database with timestamps and user IDs.
Information Disclosure	Unauthorized Access to sensitive data (Payment Info)	Only Admin can have access to all sensitive data

Denial Of Service (DOS)	The system becomes inaccessible because of constant dos attacks	Limit MSSQL connections
Elevation Of Privilege	Someone else gains access to admin level privilege	Only authorised users are able to carry out admin tasks thanks to role-based access control.

DREAD Threat Modelling

Risk	STRIDE(Category)	D	R	E	A	D	Score
Unauthorized login as admin	Spoofing	9	8	7	9	7	8
Modifying Rental/equipments record	Tampering	8	7	6	8	7	7
Denying a rental action	Repudiation	7	6	6	5	6	6
Leaking Personal Data	Information Disclosure	9	9	7	8	8	8
Overloading Server with Requests	Denial Of Service	7	7	5	8	6	7
Gaining Admin Privilege	Elevation of Privilege	9	8	7	9	8	8

Task 4: PDPA 2010

Categorization of personnel under PDPA 2010

- Data User: The organization is responsible for deciding how and why personal data is processed.

  Example: The admins oversee the rental records.

- Data Subject: The person whose personal data is captured and processed.

  Example: Students who borrow and return the equipment.

- Data Processor: The person who works on the information on behalf of the data user.

Example: The application backend which works with Python and MSSQL to store, retrieve, and process data.

Lifecycle Stage	PDPA Requirements	Compliance Measures	Responsible Person	Penalty for non compliance
Data Collection	Acquire legal consent and appropriately inform users of the intended use of the information.	- Show privacy policy notice during registration. - Gather only that which is necessary (name, email and password).	Admin and System Developer	Fine up to RM300,000 or imprisonment up to 2 years
Data Processing	Process data fairly and lawfully	Data is only processed after login checks.	Admin and Backend Developer	Fine up to RM300,000 or imprisonment up to 2 years
Data Storage	Providing safeguarding of data against unauthorized access, also to ensure data will withstand the passing of time.	-Stored Passwords are encrypted - Access and connection to the database are restricted to and from localhost only.	Database Admin	Fine up to RM300,000 or imprisonment up to 2 years
Data Sharing	Share data only with consent	-Mask Sensitive Data	Admin	Fine up to RM300,000 or

				imprisonment up to 3 years
Data Retention	Retain data only if deemed necessary	-A student's personal data is not kept when they are inactive.	Admin	Fine up to RM300,000 or imprisonment up to 2 years
Data Disposal	Dispose data securely	-Data deletion done through secure SQL queries	Database Admin	Fine up to RM300,000 or imprisonment up to 2 years

Task 5: Security Measure Implementation

5.1 Input Validation

The input validation in this Flask user registration route helps mitigate SQL injection risks by ensuring that only properly formatted and expected data reaches the database. First, it uses regular expressions to validate fields like the student ID and email, confirming that the input conforms to specific, safe patterns. This reduces the chance of malicious code being injected through these fields. More importantly, the route uses parameterized queries (? placeholders in the SQL INSERT statement), which is the most effective defense against SQL injection. With parameterized queries, the input values are bound to the SQL statement as parameters, rather than being concatenated directly into the SQL string. This means that even if a user attempts to input SQL commands into the form fields, those inputs are treated as plain data, not executable SQL code. As a result, this approach effectively neutralizes injection attempts and ensures secure interaction with the database.

```python
    if request.method == 'POST':
        full_name = request.form['full_name'].strip()
        student_id = request.form['student_id'].strip()
        email = request.form['email'].strip()
        password = request.form['password']

        # Simple validations
        if not full_name:
            flash("Full name is required.")
            return redirect('/register')

        if not re.match(r'^[A-Za-z0-9]+$', student_id):
            flash("Student ID must be alphanumeric.")
            return redirect('/register')

        email_regex = r'^[^@]+@[^@]+\.[^@]+$'
        if not re.match(email_regex, email):
            flash("Invalid email address.")
            return redirect('/register')

        if len(password) < 6:
            flash("Password must be at least 6 characters.")
            return redirect('/register')

```

```python
        # Insert new user
        cursor.execute("""
            INSERT INTO Users (full_name, student_id, email, PasswordHash)
            VALUES (?, ?, ?, ?)
        """, (full_name, student_id, email, hashed_password))
```

5.2 Hashing Password

The code hashes the password using SHA-512, converting it into a secure, fixed-length binary digest before storing it. This protects the password from being exposed if the database is compromised. While SHA-512 is strong, using specialized password hashing methods like bcrypt or Argon2 with salting is recommended for better security.

```python
        # Hash the password as binary
        hashed_password = hashlib.sha512(password.encode('utf-8')).digest()  # returns 64-byte binary

```

This is the how the password appears in its column in the database.

	id	full_name	student_id	email	PasswordHash	role
1	1	Akid Syazwan bin Nor Azman Shah	1211111238	1211111238@student.mmu.edu.my	0x41CB94C7C3CCFED627FC43BFF814B71BF8E5EF611DBF685...	NULL
2	3	admin	A001	admin@mmu.com	0xC7AD44CBAD762A5DA0A452F9E854FDC1E0E7A52A38015F...	admin

5.3 Database Audit

An audit that tracks database and schema access along with failed logins helps monitor and secures the SQL Server environment by recording who tried to access which database objects and when, as well as logging unsuccessful login attempts. This allows administrators to detect unauthorized access or suspicious activities, enforce compliance, and investigate security incidents.

```
USE [master]
GO

CREATE SERVER AUDIT [SportsInventoryAudit]
TO FILE
(   FILEPATH = N'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Audit\SportsInvetoryAuditLogs'
    ,MAXSIZE = 0 MB
    ,MAX_ROLLOVER_FILES = 2147483647
    ,RESERVE_DISK_SPACE = OFF
) WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
GO

ALTER SERVER AUDIT [SportsInventoryAudit] WITH (STATE = ON);
GO
```

```
USE [SportsInventory]
GO

CREATE DATABASE AUDIT SPECIFICATION [SportsInventoryAuditLogs]
FOR SERVER AUDIT [SportsInventoryAudit]
ADD (FAILED_DATABASE_AUTHENTICATION_GROUP),
ADD (SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP),
ADD (DATABASE_OBJECT_ACCESS_GROUP),
ADD (SCHEMA_OBJECT_ACCESS_GROUP)
GO
```

	event_time	sequence_number	action_id	succeeded	permission_bitmask	is_column_permission	session_id	server_principal_id	database_principal_id	target_server_principal_id	target_database_principal_id	object_id	class_type	session_server_principal_nam
1	2025-05-15 11:20:21.3848500	1	AUSC	1	0x00000000000000000000000000000000	0	64	259	0	0	0	0	A	MicrosoftAccount\akidsyazwa
2	2025-05-15 11:20:29.5353421	1	SL	1	0x00000000000000000000000000000001	1	67	259	1	0	0	-540	V	MicrosoftAccount\akidsyazwa
3	2025-05-15 11:20:29.5353421	1	SL	1	0x00000000000000000000000000000001	1	67	259	1	0	0	-548	V	MicrosoftAccount\akidsyazwa
4	2025-05-15 11:20:29.5353421	1	SL	1	0x00000000000000000000000000000001	1	67	259	1	0	0	-547	V	MicrosoftAccount\akidsyazwa
5	2025-05-15 11:20:29.5353421	1	SL	1	0x00000000000000000000000000000001	1	67	259	1	0	0	-645	V	MicrosoftAccount\akidsyazwa
6	2025-05-15 11:21:03.7751142	1	SL	1	0x00000000000000000000000000000001	1	67	259	1	0	0	-501	V	MicrosoftAccount\akidsyazwa
7	2025-05-15 11:21:05.4352200	1	SL	1	0x00000000000000000000000000000001	1	69	259	1	0	0	-501	V	MicrosoftAccount\akidsyazwa
8	2025-05-15 11:21:18.3397135	1	SL	1	0x00000000000000000000000000000001	1	70	259	1	0	0	-501	V	MicrosoftAccount\akidsyazwa
9	2025-05-15 11:29:32.4843367	1	SL	1	0x00000000000000000000000000000001	1	58	259	1	0	0	-540	V	MicrosoftAccount\akidsyazwa
10	2025-05-15 11:29:32.4843367	1	SL	1	0x00000000000000000000000000000001	1	58	259	1	0	0	-548	V	MicrosoftAccount\akidsyazwa
11	2025-05-15 11:29:32.4843367	1	SL	1	0x00000000000000000000000000000001	1	58	259	1	0	0	-547	V	MicrosoftAccount\akidsyazwa
12	2025-05-15 11:29:32.4843367	1	SL	1	0x00000000000000000000000000000001	1	58	259	1	0	0	-645	V	MicrosoftAccount\akidsyazwa
13	2025-05-15 11:29:40.2442804	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-426	V	MicrosoftAccount\akidsyazwa
14	2025-05-15 11:29:40.2505327	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-426	V	MicrosoftAccount\akidsyazwa
15	2025-05-15 11:29:40.2541714	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-425	V	MicrosoftAccount\akidsyazwa
16	2025-05-15 11:29:40.2744108	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-489	V	MicrosoftAccount\akidsyazwa
17	2025-05-15 11:29:40.2744108	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-389	V	MicrosoftAccount\akidsyazwa
18	2025-05-15 11:29:40.2744108	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-416	V	MicrosoftAccount\akidsyazwa
19	2025-05-15 11:29:40.2744108	1	SL	1	0x00000000000000000000000000000001	1	64	259	1	0	0	-412	V	MicrosoftAccount\akidsyazwa