Prevent User Deletion If Assigned To An Incident

1. Enforce It Where It Matters Most

Layer	Why it helps	Typical implementation
Database	Absolute last line of defence; protects from rogue scripts	Foreign key with ON DELETE RESTRICT or trigger to raise exception
Service layer/ORM	Clean rule in business logic, return friendly errors	Check via query (e.g. exists check)
API/Controller	Catch and return appropriate API response	Wrap delete call, return 409 or 422
UI	Prevents user attempt visually	Grey-out button or show message

Best practice: Implement checks in every layer, but DB constraint is essential.

2. Quick Reference Snippets

SQL (PostgreSQL):

```
ALTER TABLE incidents
  ADD CONSTRAINT fk_incident_user
  FOREIGN KEY (assigned_user_id)
  REFERENCES users(id)
  ON DELETE RESTRICT;
```

SQL Trigger (PostgreSQL):

```
CREATE OR REPLACE FUNCTION prevent_user_delete()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (SELECT 1 FROM incidents WHERE assigned_user_id = OLD.id) THEN
    RAISE EXCEPTION 'Cannot delete user %: still assigned to incident(s)', OLD.id;
  END IF;
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_block_user_delete
BEFORE DELETE ON users
FOR EACH ROW EXECUTE FUNCTION prevent_user_delete();
```

Django ORM:

```
from django.db import IntegrityError
from incidents.models import Incident
from users.models import User

def delete_user(user_id):
    if Incident.objects.filter(assigned_user_id=user_id).exists():
        raise IntegrityError("User cannot be deleted: assigned to incident(s).")
    User.objects.filter(id=user_id).delete()
```

Node.js (Express + Sequelize):

```
async function deleteUser(req, res) {
  const { userId } = req.params;
  const count = await Incident.count({ where: { assignedUserId: userId } });

  if (count) {
    return res.status(409).json({
      error: "User is assigned to one or more incidents and cannot be deleted.",
    });
  }
  await User.destroy({ where: { id: userId } });
  res.status(204).end();
}
```

ServiceNow Script Include:

```
var UserValidator = Class.create();
UserValidator.prototype = {
  preventDeletion: function(userSysId) {
    var incGr = new GlideRecord('incident');
    incGr.addQuery('assigned_to', userSysId);
    incGr.query();
    if (incGr.hasNext()) {
      gs.addErrorMessage('User is assigned to an Incident and cannot be deleted.');
      return false;
    }
    return true;
  },
  type: 'UserValidator'
};
```

3. Optional Extras

- Soft-delete (is_deleted flag) for historical tracking.
- Nightly cleanup jobs to notify stale assignments.
- Unit & integration tests to protect logic.

Next Step: Adapt code for your stack (Laravel, Spring Boot, ServiceNow, etc.) as needed.