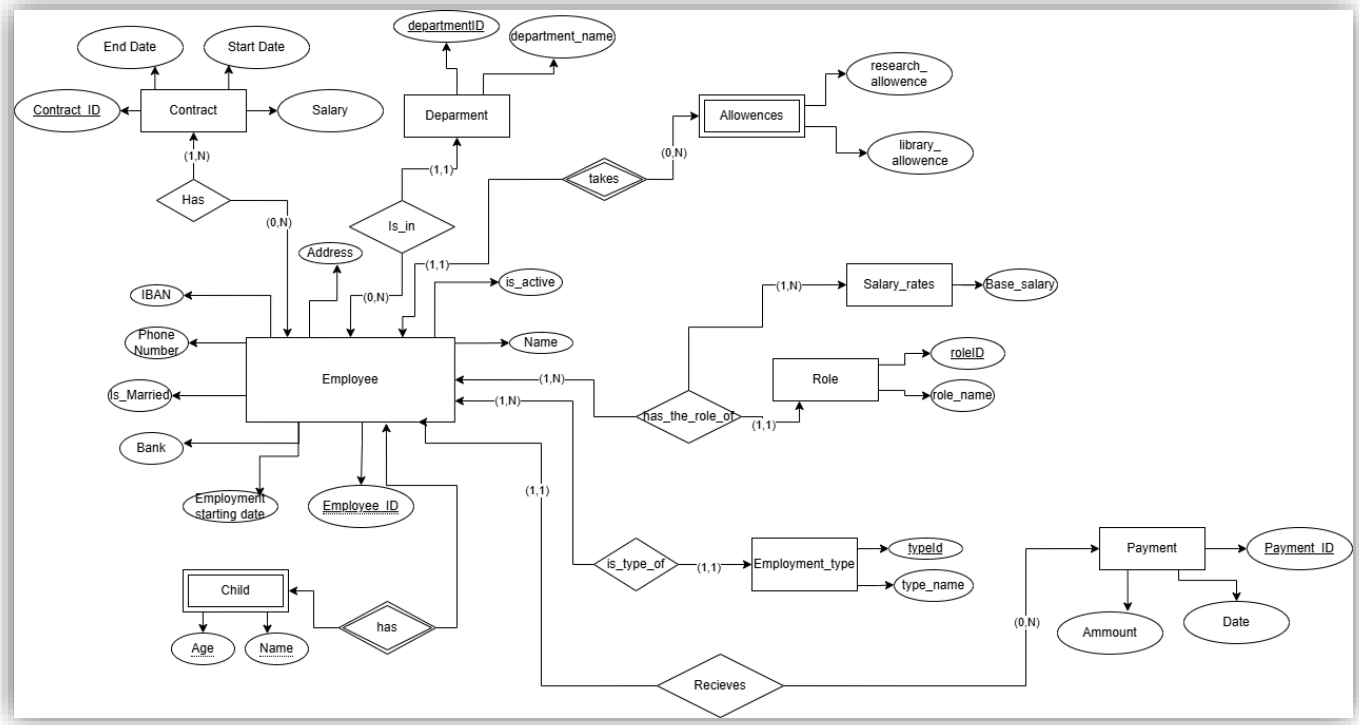
Αναφορά Project HY360

Ομάδα 5

csd5358 Αλτιντζής Λάζαρος Αλέξανδρος
csd5369 Βίτσος Γεώργιος Θεόδωρος
csd5452 Μεντόλλι Ματέο

Περιεχόμενα

Διάγραμμα Entity Relationship



Το παραπάνω διάγραμμα αναπαριστά την σχέση οντοτήτων της βάσης δεδομένων. Οι οντότητες και οι σχέσεις τους έχουν ως εξής

- **Employee**: Αναπαριστά τον γενικό τύπο υπαλλήλου, ο οποίος περιέχει όλα τα βασικά γνωρίσματα που κληρονομούνται από τις εξειδικευμένες οντότητες. Τα αναγνωριστικά αυτά είναι
    - **Employee_Id (INT):** Το πρωτεύων κλειδί του υπαλλήλου
    - **Name (VARCHAR):** Το όνομα του υπαλλήλου.
    - **Employment starting date (DATE):** Η ημερομηνία πρόσληψης του υπαλλήλου
    - **Bank (VARCHAR):** Η τράπεζα του υπαλλήλου.
    - **Is_married (BOOLEAN):** Δείχνει αν ο υπάλληλος είναι παντρεμένος η όχι.
    - **IBAN (VARCHAR):** Το IBAN του υπαλλήλου
    - **Phone Number (VARCHAR):** Το κινητό τηλέφωνο του υπαλλήλου.
    - **Address (VARCHAR):** Η διεύθυνση του υπαλλήλου.
    - **Is_active(BOOLEAN):** Δείχνει αν είναι ενεργός ο υπάλληλος.

    Οι σχέσεις του Employee είναι:

- **Is_In(1,1):** Ένας υπάλληλος είναι μόνο σε ένα τμήμα και ένα τμήμα περιέχει μόνο έναν υπαλλήλου με το συγκεκριμένο employee_Id.
- **Has(0,N):** Ένας υπάλληλος έχει από κανένα ως Ν παιδιά.
- **Receives(0,N):** Ένας υπάλληλος μπορεί να παίρνει διαφορετικούς μισθούς ανάλογα με ποσά συμβόλαια έχει.
- **HasTheRoleOf(1,1):** Ένας υπάλληλος έχει μόνο έναν ρόλο.
- **Is_type_of(1,1):** Ένας υπάλληλος έχει μόνο έναν τύπο.
- **Takes(0,N):** Ένας υπάλληλος μπορεί να πάρει κανένα η πολλαπλά επιδόματα.

- **Role:** Αναπαριστά τον ρόλο ενός υπαλλήλου. Τα γνωρίσματα του είναι:
  - **RoleID(INT):** το πρωτεύων κλειδί.
  - **Role_name(ENUM):** Ο ρόλος ενός υπαλλήλου μεταξύ Teaching και Administrator.

  Οι σχέσεις του ρολού είναι:
  - **Has_the_role_of(1,N):** Τον ρόλο τον μπορούν να μοιράζονται διάφορα salary_rates.
  - **Has_the_role_of(1,N):** Τον ρόλο τον μπορούν να μοιράζονται διάφοροι υπάλληλοι.

- **Employee_type:** Αναπαριστά τον τύπο ενός υπαλλήλου. Τα γνωρίσματα του είναι:
  - **TypeID(INT):** Το πρωτεύων κλειδί.
  - **Type_name(ENUM):** Τον τύπο υπαλλήλου του ανάμεσα σε Permanent και Contractor

  Η σχέση του Τύπου υπάλληλου είναι:
  - **Is_type_of(1,N):** Ο συγκεκριμένος τύπος υπάλληλου μπορεί να μοιράζεται από πολλούς υπάλληλους.

- **Department:** Αναπαριστά ένα τμήμα του πανεπιστήμιου στο οποίο ανήκει ένας υπάλληλος. Το γνώρισμα του είναι:
  - **Department_ID(INT):** Το πρωτεύων κλειδί του τμήματος.
  - **Department_name(VARCHAR):** Το όνομα του τμήματος.

  Η σχέση του τμήματος είναι:

  - **Is_In(0,N):** Ένα τμήμα μπορεί να έχει κανέναν η πολλούς υπαλλήλους που δουλεύουν εκεί.
- **Child:** Αναπαριστά ένα παιδί ενός υπάλληλου. Αυτή η οντότητα είναι ασθενής αφού δεν στέκει μόνη της στην βάση δεδομένων. Υπάρχει περιορισμός ολικής συμμετοχής του Child στη σχέση Has. Κάθε παιδί πρέπει υποχρεωτικά να συνδέεται με έναν γονέα-υπάλληλο. Τα γνωρίσματα της είναι:
  - **Name(VARCHAR):** Το όνομα του παιδιού π είναι μέλος του κλειδιού του μαζί με το Employee_Employee_Id.
  - **Age(INT):** Η ηλικία του παιδιού

  Η σχέση που έχει είναι:

- **Has(1,1):** Ένα παιδί μπορεί να έχει ένα και μόνο γονέα που είναι υπάλληλος.

- **Payment:** Αναπαριστά την πληρωμή ενός υπαλλήλου. Τα γνωρίσματα του είναι:
  - **Payment_Id(INT):** Το αναγνωριστικό της πληρωμής.
  - **Amount(DECIMAL(10,2)):** Ο μισθός πριν από κάποιο επίδομα.
  - **Date(DATE):** Η ημερομηνία που πραγματοποιήθηκε η πληρωμή.

  Η μοναδική σχέση της οντότητας είναι:

  - **Receives(1,1):** Η συγκεκριμένη πληρωμή είναι μοναδική για τον υπάλληλο.

- **Contract:** Αναπαριστά συμβόλαιο που έχει ένας συμβασιούχος υπάλληλος. Υπάρχει περιορισμός ολικής συμμετοχής του Contract στη σχέση Has. Κάθε συμβόλαιο πρέπει να ανήκει σε έναν συμβασιούχο.. Τα γνωρίσματα του είναι:
  - **Contract_ID(INT):** Το αναγνωριστικό του συμβολαίου.
  - **Start_date(DATE):** Η ημερομηνία έναρξης της συμβάσης.
  - **End_date(DATE):** Η ημερομηνία λήξης της συμβάσης.
  - **Salary(DECIMAL(10,2)):** Ο μισθός πριν από κάποιο επίδομα.

  Η μοναδική σχέση της οντότητας είναι:

  - **Has(1,1):** Το συγκεκριμένο συμβόλαιο ανήκει μόνο σε έναν συμβασιούχο του πανεπιστήμιου.
- **Salary_rates:** Αναπαριστά τον βασικό μισθό του κάθε υπάλληλου. Τα γνωρίσματα του είναι:
  - **RoleName(VARCHAR):** Ο ρόλος του υπάλληλου.
  - **BaseSalary(DECIMAL(10,2)):** Ο βασικός μισθός του υπάλληλου.

  Η μοναδική σχέση του είναι:
  - **HasTheRoleOf(0,N).** Πολλοί υπάλληλοι μπορούν να έχουν τον συγκεκριμένο ρολο.

Παρατηρήσεις

- **Υπολογισμός Μισθοδοσίας & SalaryRates:** Ο πίνακας SalaryRates συνδέεται με την οντότητα Role και καθορίζει τον Βασικό Μισθό για τους υπαλλήλους βάσει του ρόλου τους (π.χ. Teaching, Administrator). Ωστόσο, για τους υπαλλήλους με τύπο "Contractor", η εφαρμογή θα λαμβάνει υπόψη πρωτίστως τον μισθό που ορίζεται ρητά στο ενεργό Contract, επιτρέποντας την απαραίτητη ευελιξία στις συμβάσεις.
- **Κεντρική Διαχείριση Πληρωμών:** Η οντότητα Payment συνδέεται απευθείας με τον Employee. Αυτή η σχεδίαση επιτρέπει την ενιαία καταγραφή πληρωμών ανεξαρτήτως του Employee_type (Μόνιμος ή Συμβασιούχος), απλοποιώντας τα ερωτήματα (queries) που αφορούν το συνολικό κόστος μισθοδοσίας.
- **Ιστορικότητα Συμβάσεων**: Η σχέση Contract με τον συμβασιούχο υπάλληλο επιτρέπει πολλαπλά συμβόλαια (ιστορικό). Γίνεται η παραδοχή ότι σε επίπεδο επιχειρησιακής λογικής (Application

Logic), θα υπάρχει έλεγχος ώστε μόνο μία σύμβαση να είναι ενεργή (active) για κάθε υπάλληλο την τρέχουσα χρονική στιγμή, βάσει των ημερομηνιών Start_date και End_date.

- **Ασθενής Οντότητα Child:** Για την οντότητα Child, χρησιμοποιούμε το Name ως μερικό κλειδί μαζί με το Employee_Id. Κάνουμε την παραδοχή ότι στον ίδιο γονέα δεν καταχωρούνται δύο παιδιά με το ίδιο μικρό όνομα. Σε αντίθετη περίπτωση, θα απαιτούνταν χρήση τεχνητού κλειδιού.
- **Γονείς Υπάλληλοι:** Αγνοούμε την περίπτωση που και οι δύο γονείς ενός παιδιού εργάζονται στο πανεπιστήμιο, καθώς η μοντελοποίηση της σχέσης (1,2) θα αύξανε την πολυπλοκότητα χωρίς ουσιαστικό όφελος για το σύστημα μισθοδοσίας (το επίδομα τέκνων δίνεται συνήθως στον έναν εκ των δύο ή και στους δύο ανεξάρτητα).

Σχεσιακό Μοντέλο

Το σχεσιακό μοντέλο της βάσης δεδομένων ακολουθεί ως εξής:

- **Employee(**idEmployee, address, iban, phone_number, is_Married, is_active,bank, employment_starting_date, name, DepartmentID, RoleID, TypeID**)**
  - **FK1:** DepartmentID → Department(departmentID)
  - **FK2:** RoleID → Role(roleId)
  - **FK3:** TypeID → Employee_type(TypeID)
- **Role(**RoleID, role_name**)**
- **Employee_type(**TypeID, type_name**)**
- **Department(**departmentID, department_name**)**
- **Salary_rates(**roleID, base_salary**)**
  - **FK:** roleID → Role(roleID)
- **Payment(**idPayment, date, ammount**)**
  - **FK:** idEmployee → Employee(idEmployee)
- **Contract(**idContract, start_date, end_date, salary, idEmployee**)**
  - **FK:** idEmployee → Employee(idEmployee)
- **Child(**idEmployee, name, age**)**
  - **FK:** idEmployee → Employee(idEmployee)
- **Allowances(**idEmployee, research_allowence, library_allowence**)**
  - **FK:** idEmployee → Employee(idEmployee)

Γλώσσα Ορισμού Δεδομένων (DLL) μέσω SQL

Ακολουθεί ο κώδικας SQL που δημιουργεί όλες τις οντότητες που ανήκουν στην βάση δεδομένων

Role

```
-- -----------------------------------------------------
-- Table `Role`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Role` ;

CREATE TABLE IF NOT EXISTS `Role` (
  `roleID` INT NOT NULL AUTO_INCREMENT,
  `role_name` ENUM('TEACHING', 'ADMINISTRATIVE') NULL,
  PRIMARY KEY (`roleID`))
ENGINE = InnoDB;
```

Employment_type

```
-- -----------------------------------------------------
-- Table `Employment_type`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Employment_type` ;

CREATE TABLE IF NOT EXISTS `Employment_type` (
  `typeID` INT NOT NULL AUTO_INCREMENT,
  `type_name` ENUM('PERMANENT', 'CONTRACTOR') NULL,
  PRIMARY KEY (`typeID`))
ENGINE = InnoDB;
```

Department

```
-- -----------------------------------------------------
-- Table `Department`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Department` ;

CREATE TABLE IF NOT EXISTS `Department` (
  `departmentID` INT NOT NULL AUTO_INCREMENT,
  `department_name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`departmentID`),
  UNIQUE INDEX `department_name_UNIQUE` (`department_name` ASC) VISIBLE)
ENGINE = InnoDB;
```

Child

```
-- -----------------------------------------------------
-- Table `Child`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Child` ;

CREATE TABLE IF NOT EXISTS `Child` (
  `age` INT NULL,
  `name` VARCHAR(45) NOT NULL,
  `Employee_idEmployee` INT NOT NULL,
  PRIMARY KEY (`Employee_idEmployee`, `name`),
  CONSTRAINT `fk_Child_Employee1`
    FOREIGN KEY (`Employee_idEmployee`)
    REFERENCES `Employee` (`idEmployee`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Employee

```
-- -----------------------------------------------------
-- Table `Employee`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Employee` ;

CREATE TABLE IF NOT EXISTS `Employee` (
  `idEmployee` INT NOT NULL AUTO_INCREMENT,
  `address` VARCHAR(255) NULL,
  `iban` VARCHAR(45) NULL,
  `phone_number` VARCHAR(12) NULL,
  `is_Married` TINYINT NULL,
  `bank` VARCHAR(45) NULL,
  `employment_starting_date` DATE NULL,
  `name` VARCHAR(45) NULL,
  `Role_roleID` INT NOT NULL,
  `Employment_type_typeID` INT NOT NULL,
  `Department_departmentID` INT NOT NULL,
  PRIMARY KEY (`idEmployee`),
  INDEX `fk_Employee_Role1_idx` (`Role_roleID` ASC) VISIBLE,
  INDEX `fk_Employee_Employment_type1_idx` (`Employment_type_typeID` ASC) VISIBLE,
  INDEX `fk_Employee_Department1_idx` (`Department_departmentID` ASC) VISIBLE,
  CONSTRAINT `fk_Employee_Role1`
    FOREIGN KEY (`Role_roleID`)
    REFERENCES `Role` (`roleID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_Employment_type1`
    FOREIGN KEY (`Employment_type_typeID`)
    REFERENCES `Employment_type` (`typeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Employee_Department1`
    FOREIGN KEY (`Department_departmentID`)
    REFERENCES `Department` (`departmentID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Contract

```
-- -----------------------------------------------------
-- Table `Contract`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Contract` ;

CREATE TABLE IF NOT EXISTS `Contract` (
  `idContract` INT NOT NULL AUTO_INCREMENT,
  `start_Date` DATE NOT NULL,
  `end_Date` DATE NOT NULL,
  `salary` DECIMAL(10,2) NOT NULL,
  `Employee_idEmployee` INT NOT NULL,
  PRIMARY KEY (`idContract`),
  INDEX `fk_Contract_Employee1_idx` (`Employee_idEmployee` ASC) VISIBLE,
  CONSTRAINT `fk_Contract_Employee1`
    FOREIGN KEY (`Employee_idEmployee`)
    REFERENCES `Employee` (`idEmployee`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Payment

```
-- -----------------------------------------------------
-- Table `Payment`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Payment` ;

CREATE TABLE IF NOT EXISTS `Payment` (
  `idPayment` INT NOT NULL AUTO_INCREMENT,
  `date` DATE NULL,
  `amount` DECIMAL(10,2) NOT NULL,
  `Employee_idEmployee` INT NOT NULL,
  PRIMARY KEY (`idPayment`),
  INDEX `fk_Payment_Employee1_idx` (`Employee_idEmployee` ASC) VISIBLE,
  CONSTRAINT `fk_Payment_Employee1`
    FOREIGN KEY (`Employee_idEmployee`)
    REFERENCES `Employee` (`idEmployee`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Salary_rates

```
-- -----------------------------------------------------
-- Table `Salary_rates`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Salary_rates` ;

CREATE TABLE IF NOT EXISTS `Salary_rates` (
  `base_salary` DECIMAL(10,2) NULL,
  `Role_roleID` INT NOT NULL,
  PRIMARY KEY (`Role_roleID`),
  INDEX `fk_Salary_rates_Role1_idx` (`Role_roleID` ASC) VISIBLE,
  CONSTRAINT `fk_Salary_rates_Role1`
    FOREIGN KEY (`Role_roleID`)
    REFERENCES `Role` (`roleID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Allowances

```
-- -----------------------------------------------------
-- Table `Allowences`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Allowences` ;

CREATE TABLE IF NOT EXISTS `Allowences` (
  `research_allowence` DECIMAL(10,2) NULL,
  `library_allowence` DECIMAL(10,2) NULL,
  `Employee_idEmployee` INT NOT NULL,
  INDEX `fk_Allowences_Employee1_idx` (`Employee_idEmployee` ASC) VISIBLE,
  PRIMARY KEY (`Employee_idEmployee`),
  CONSTRAINT `fk_Allowences_Employee1`
    FOREIGN KEY (`Employee_idEmployee`)
    REFERENCES `Employee` (`idEmployee`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Περιορισμοί Ακεραιότητας και Συναρτησιακές Εξαρτήσεις

Περιορισμοί Ακεραιότητας

1. **Ακεραιότητα οντοτήτων:** Σε όλους τους πίνακες έχει οριστεί ένα πρωτεύων κλειδί, το οποίο διασφαλίζει την μοναδικότητα και αποτρέπει τις τιμές NULL
2. **Ακεραιότητα αναφοράς:** Έχουν οριστεί ξένα κλειδιά (FK) που συνδέουν τους πίνακες μεταξύ τους (Employee → Payment). Αυτοί οι περιορισμοί διασφαλίζουν ότι δεν θα εγγράφει υπάλληλος σε ανύπαρκτο τμήμα η πληρωμή σε ανύπαρκτο υπάλληλο κ.ο.κ.
3. **Περιορισμοί πεδίων:** Έχουν οριστεί κατάλληλοι τύποι δεδομένων (DATE, DECIMAL κλπ.) και περιορισμοί NOT NULL στα υποχρεωτικά πεδία

Συναρτησιακές εξαρτήσεις

Οι συναρτησιακές εξαρτήσεις των οντοτήτων, που εξηγούν και τα κλειδιά έχει ως εξής:

- **Employee**: idEmployee → {address, iban, phone_number, is_Married, bank, employment_starting_date, name, DepartmentID, RoleID, TypeID }
- **Role**: RoleID → {role_name}
- **Employee_type**: TypeID → {type_name}
- **Salary_rates**: roleID → {base _salary }
- **Department**: departmentID → deparpent_name
- **Payment**: idPayment → {date, amount, idEmployee }
- **Contract**: idContract → {start _date, end _date, salary, idEmployee }
- **Child**: {idEmployee, name } → age
- **Allowances:** idEmployee → {research_allowance, library_allowance}

Τρίτη κανονική μορφή (3NF)

Η σχεδίαση της βάσης δεδομένων καθιστά όλες τις οντότητες σε κατάσταση 3NF, οπότε δεν θα χρειαστεί να κάνομε κάποια κανονικοποίηση. Αναλυτικότερα:

- **Πρώτη κανονική μορφή (1NF):** Όλα τα πεδία των πινάκων περιέχουν ατομικές τιμές και δεν υπάρχουν επαναλαμβανόμενες ομάδες πεδίων.
- **Δεύτερή κανονική μορφή (2NF):** Εφόσον όλοι πίνακες είναι σε 1NF και κάθε γνώρισμα που δεν είναι κλειδί η μέρος κλειδιού εξαρτάται πλήρως από το κλειδί του
    - Το γνώρισμα age στον πίνακα child εξαρτάται και από τα 2 τμήματα το κλειδιού (idEmployee και name), καθιστώντας το 2NF.
- **Τρίτη κανονική μορφή (3NF):** Δεν υπάρχουν μεταβατικές εξαρτήσεις. Κάθε μη κλειδί γνώρισμα εξαρτάται αποκλειστικά από το πρωτεύων κλειδί και όχι από τα αλλά μη κλειδιά γνωρίσματα.

Διατήρηση εξαρτήσεων και μη απώλεια πληροφορίας

- **Διατήρηση συναρτησιακών σχέσεων.** Κάθε συναρτησιακή εξάρτηση (πχ role → salary) καλύπτεται από κλειδιά η ξένα κλειδιά πινάκων. Δεν χάθηκε καμία εξάρτηση κατά την δημιουργία των πινάκων.
- **Απώλεια Πληροφορίας:** Η διάσπαση των οντοτήτων (πχ διάσπαση Permanent από Employee) έγινε με τρόπο οπού επιτρέπει την ανακατασκευή της αρχικής πληροφορίας μέσω φυσικής συνένωσης των πινάκων, με την χρήση ξένων κλειδιών.

Ερωτήσεις προς την βάση δεδομένων με SQL

1. **Βρείτε τον μέσο μισθό των εργαζομένων:**

```sql
SELECT AVG(final_salary) AS average_university_salary
FROM (
    -- A. Υπολογισμός για Μόνιμους (Permanent)
    SELECT (sr.base_salary * (1 + (TIMESTAMPDIFF(YEAR, e.employment_starting_date, CURDATE()) * 0.15))) AS final_salary
    FROM Employee e
    JOIN Employment_type et ON e.Employment_type_typeID = et.typeID
    JOIN Salary_rates sr ON e.Role_roleID = sr.Role_roleID
    WHERE et.type_name = 'PERMANENT'

    UNION ALL

    -- B. Υπολογισμός για Συμβασιούχους (Contractors)
    SELECT c.salary
    FROM Contract c
    WHERE CURDATE() BETWEEN c.start_Date AND c.end_Date
) AS all_salaries;
```

2. **Βρείτε το τμήμα με τους περισσότερους υπαλλήλους:**

```sql
SELECT d.department_name, COUNT(e.idEmployee) AS num_employees
FROM Employee e
JOIN Department d ON e.Department_departmentID = d.departmentID
GROUP BY d.department_name
ORDER BY num_employees DESC
LIMIT 1;
```

3. **Βρείτε τους υπαλλήλους που έχουν περισσότερα από 2 παιδιά:**

```sql
SELECT e.name, COUNT(c.name) AS num_children
FROM Employee e
JOIN Child c ON e.idEmployee = c.Employee_idEmployee
GROUP BY e.idEmployee, e.name
HAVING COUNT(c.name) > 2;
```

4. **Υπολογίστε το συνολικό ποσό που δαπανά το πανεπιστήμιο για μισθοδοσία τον τρέχοντα μήνα:**

```sql
SELECT SUM(amount) AS total_monthly_payroll
FROM Payment
WHERE MONTH(date) = MONTH(CURDATE()) AND YEAR(date) = YEAR(CURDATE());
```

5. **Βρείτε τους συμβασιούχους των οποίων η σύμβαση λήγει τον επόμενο μήνα**

```sql
SELECT e.name, d.department_name, c.end_Date
FROM Employee e
JOIN Department d ON e.Department_departmentID = d.departmentID
JOIN Contract c ON e.idEmployee = c.Employee_idEmployee
WHERE c.end_Date BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 1 MONTH);
```

Περιγραφή διαδικασιών σε ψευδοκώδικα

1. Διαδικασία πρόσληψης υπάλληλου :

Η διαδικασία εγγραφής νέου υπαλλήλου στη βάση δεδομένων, διαχωρίζοντας τη λογική ανάλογα με τον τύπο (Permanent/Contractor) και τον ρόλο (Teaching/Administrative).

```
ALGORITHM Hire_Employee(Personal_Details, Role, Type, Salary_Info)
BEGIN
    1. INSERT basic details into Employee table:
       (Name, Address, Tax_ID, Phone, Bank, IBAN, Start_Date, RoleID, TypeID, DepartmentID)
    2. GET the generated idEmployee.

    3. CHECK Employment Type (TypeID):
       IF (Type == CONTRACTOR) THEN
           INSERT into Contract table:
           (Start_Date, End_Date, Salary, idEmployee)
       END_IF

    4. CHECK Job Role (RoleID):
       IF (Role == TEACHING) THEN
           IF (Type == PERMANENT) THEN
               INSERT into Allowances table: (Research_Allowance, idEmployee)
           ELSE IF (Type == CONTRACTOR) THEN
               INSERT into Allowances table: (Library_Allowance, idEmployee)
           END_IF
       END_IF

    5. PRINT "Employee hired successfully."
END
```

2. Διαδικασία υπολογισμού μισθοδοσίας:

Η διαδικασία υπολογισμού του τελικού μηνιαίου μισθού, λαμβάνοντας υπόψη τον βασικό μισθό, τα έτη προϋπηρεσίας, την οικογενειακή κατάσταση και τα επιδόματα.

```
ALGORITHM Calculate_Payroll(idEmployee)
BEGIN
    1. RETRIEVE Employee details from Database (Role, Type, Start_Date, Is_Married).
    2. IF (Employee.Is_Active == FALSE) THEN
           RETURN 0 (Inactive employees are not paid).

    3. INITIALIZE Final_Salary = 0
    4. INITIALIZE Base_Amount = 0

    // Step 1: Calculate Base Amount
    IF (Type == PERMANENT) THEN
        Base_Amount = RETRIEVE Base_Salary FROM Salary_Rates WHERE RoleID = Employee.RoleID
        Years_Of_Service = Current_Year - Start_Year

        // 15% increase for every year of service
        IF (Years_Of_Service > 0) THEN
            Bonus = Base_Amount * 0.15 * Years_Of_Service
            Final_Salary = Base_Amount + Bonus
        ELSE
            Final_Salary = Base_Amount
        END_IF

    ELSE IF (Type == CONTRACTOR) THEN
        Base_Amount = RETRIEVE Salary FROM Contract WHERE idEmployee = Employee.ID AND Active = True
        Final_Salary = Base_Amount
    END_IF

    // Step 2: Family Allowances (Percentage of Base Amount)
    IF (Employee.Is_Married == TRUE) THEN
        Final_Salary = Final_Salary + (Base_Amount * 0.05)
    END_IF

    Child_Count = COUNT records FROM Child WHERE idEmployee = Employee.ID
    IF (Child_Count > 0) THEN
        Final_Salary = Final_Salary + (Base_Amount * 0.05 * Child_Count)
    END_IF

    // Step 3: Specific Role Allowances
    IF (Record exists in Allowances for Employee) THEN
        Allowance_Val = RETRIEVE Amount FROM Allowances
        Final_Salary = Final_Salary + Allowance_Val
    END_IF

    // Step 4: Record Payment
    INSERT into Payment table: (Current_Date, Final_Salary, idEmployee)

    RETURN Final_Salary
END
```

3. Αλγόριθμος Μαζικής Εκτέλεσης Μισθοδοσίας

Η διαδικασία που εκτελείται στο τέλος του μήνα για την πληρωμή όλων των ενεργών υπαλλήλων.

```
ALGORITHM Run_Mass_Payroll()
BEGIN
    1. Employees_List = RETRIEVE ALL from Employee WHERE Is_Active = TRUE
    2. Total_Cost = 0

    3. FOR EACH Employee IN Employees_List DO:
           Salary = CALL Calculate_Payroll(Employee.ID)
           Total_Cost = Total_Cost + Salary
       END_FOR

    4. PRINT "Payroll completed. Total Cost: " + Total_Cost
END
```