SQL-3

Ν. ΠΟΛΥΖΩΤΗΣ, Π. ΞΗΡΟΣ, Π. ΒΑΣΙΛΕΙΑΔΗΣ

3.1 ΓΕΝΙΚΑ

Οι απαιτήσεις για τη διαχείριση των δεδομένων σε πολλές επιστημονικές αλλά και εμπορικές εφαρμογές πολλές φορές έχουν ξεπεράσει τις δυνατότητες των υπαρχόντων Συστημάτων Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ). Οι εφαρμογές αυτές συχνά απαιτούν μια ολοκληρωμένη λύση για δεδομένα διαφορετικής φύσης (π.χ. κείμενα, γραφικά, κλασικά αλφαριθμητικά δεδομένα, εικόνες, video κλπ.), τα οποία βρίσκονται αποθηκευμένα σε κατανεμημένες βάσεις δεδομένων. Οι ανάγκες των χρηστών επιβάλουν τη δυνατότητα για ομοιόμορφη πρόσβαση, ασφάλεια και αξιοπιστία των δεδομένων στα συστήματα αυτά.

Η SQL είναι η σχεσιακή γλώσσα που χρησιμοποιείται κυρίως στις εφαρμογές βάσεων δεδομένων (σε συνδυασμό με το ευρύτατα διαδεδομένο σχεσιακό μοντέλο) και είναι εδώ και πολύ καιρό διεθνές στάνταρτ. Το Remote Database Access (RDA) είναι το αντίστοιχο στάνταρτ για κατανεμημένη επεξεργασία. Ο συνδυασμός τους έχει χρησιμοποιηθεί ευρέως για την αντιμετώπιση των αναγκών των χρηστών. Η SQL, ειδικά, είναι ειδική στον ορισμό και τη διαχείριση δομημένων δεδομένων μέσω ενός interface υψηλού επιπέδου, το οποίο επιτρέπει την επερώτηση και ανανέωση των δεδομένων. Άλλα πλεονεκτήματα της SQL είναι η δυνατότητα διαχείρισης του σχήματος της βάσης (και των διαφορετικών όψεων που το σχήμα αυτό μπορεί να έχει για διαφορετικούς χρήστες), μέσα από ένα πλούσιο μοντέλο μετα-πληροφορίας, η ύπαρξη περιορισμών ακεραιότητας, καθώς και η διαχείριση της ταυτόχρονης πρόσβασης χρηστών, με διαφορετικά δικαιώματα στην προσπέλαση και διαχείριση των δεδομένων. Η SQL στηρίζεται σε ένα καλά θεμελιωμένο μαθηματικό υπόβαθρο που βασίζεται στον κατηγορικό λογισμό πρώτης τάξης.

Το 1991, τεχνικές επιτροπές για την τυποποίηση της SQL, λειτουργώντας κάτω από τη δικαιοδοσία του American National Standards Institute (ANSI) και του International

Organization for Standardization (ISO), ορίστηκαν για την αναβάθμιση της SQL σε μια υπολογιστικά πλήρη γλώσσα για τη διαχείριση σύνθετων αντικειμένων. Αποτέλεσμα αυτής της προσπάθειας ήταν το διεθνές στάνταρτ ISO/IEC 9075 που ετοιμάστηκε από την επιτροπή ISO/IEC JTC 1, Information Technology. Η τέταρτη έκδοση του στάνταρτ αυτού είναι αυτή που εμείς στη συνέχεια θα ονομάζουμε SQL-3 στάνταρτ (και που αντικαθιστά την προηγούμενη έκδοση, γνωστή και ως SQL92). Το ISO/IEC 9075 αποτελείται από τα παρακάτω τμήματα κάτω από το γενικό τίτλο "Information technology - Database languages - SQL":

- Part 1: Framework (SQL/Framework)
- Part 2: Foundation (SQL/Foundation)
- Part 3: Call-Level Interface (SQL/CLI)
- Part 4: Persistent Stored Modules (SQL/PSM)
- Part 5: Host Language Bindings (SQL/Bindings)
- Part 6: XA Specialization (SQL/Transaction)
- Part 7: Temporal (SQL/Temporal)
- Part 8: Extended Objects (SQL/Object)
- Part 9: Virtual Tables (SQL/VirtualTable)

3.2 ΤΑ ΒΑΣΙΚΑ ΤΜΗΜΑΤΑ ΤΟΥ SQL-3 ΣΤΑΝΤΑΡΤ

3.2.1 SQL/Framework

Το SQL/Framework είναι το τμήμα του στάνταρτ που παρέχει μια γενική εικόνα για τα υπόλοιπα τμήματα και πώς αυτά χρησιμοποιούνται. Το SQL/Framework χρησιμεύει σαν οδηγός για την κατανόηση του όλου στάνταρτ και παρουσιάζει πώς τα διάφορα τμήματα "δένουν" μεταξύ τους. Επιπλέον, το SQL/Framework παρέχει ένα κοινό λεξιλόγιο, την ορολογία, δηλαδή, του στάνταρτ.

3.2.2 SQL/Foundation

Το SQL/Foundation περιγράφει τις δομές δεδομένων και τις βασικές λειτουργίες που μπορούν να εφαρμοστούν σε δεδομένα. Παρέχει, δηλαδή, τις λειτουργικές ικανότητες για τη δημιουργία, προσπέλαση, συντήρηση, διαχείριση και προστασία των δεδομένων. Το SQL/Foundation καθορίζει τις δομές και τις βασικές πράξεις πάνω σε δεδομένα της SQL. Προδιαγράφει επίσης λειτουργίες για την δημιουργία, προσπέλαση, συντήρηση, έλεγχο και προστασία των δεδομένων. Ακόμα, καθορίζει την σύνταξη και την σημασιολογία μιας γλώσσας βάσεων δεδομένων με τις παρακάτω δυνατότητες:

- Καθορισμός και χειρισμός της δομής και των περιορισμών για την ακεραιότητα των δεδομένων.
- Δήλωση και εκτέλεση λειτουργιών πάνω σε δεδομένα και δείκτες

- Δήλωση ρουτινών.

Μαζί με τα προηγούμενα καθορίζεται επίσης και ένα σχήμα πληροφορίας το οποίο περιγράφει την δομή και τους περιορισμούς ακεραιότητας (integrity constraints) των δεδομένων. Το στάνταρτ επιχειρεί να δώσει μια κοινή πλατφόρμα για την ενοποίηση και λειτουργικότητα των διαφόρων υλοποιήσεων της SQL, χωρίς να προσπαθεί να καθορίσει πλήρως όλα τις λεπτομέρειες της υλοποίησης.

Ανάμεσα στα νέα στοιχεία που εισάγονται στην SQL και που περιγράφονται στο τμήμα SQL/Foundation είναι και τα παρακάτω:

- νέοι τύποι δεδομένων
- κατηγορήματα (predicates)
- σχεσιακός τελεστής αναδρομικής ερώτησης
- δυνατότητες δοσοληψιών
- ενεργοποιητές (triggers)
- user-defined types

3.2.3 SQL/CLI (Call Level Interface)

Το SQL/CLI στάνταρτ καθορίζει τις δομές και τις διαδικασίες που μπορούν να χρησιμοποιηθούν για την εκτέλεση εντολών της SQL μέσα από μια εφαρμογή που έχει γραφτεί σε μια από τις κλασικές γλώσσες προγραμματισμού. Η εκτέλεση των εντολών μπορεί να γίνει με τέτοιο τρόπο ώστε οι ρουτίνες της γλώσσας που θα χρησιμοποιηθούν να είναι ανεξάρτητες από τις SQL εντολές.

3.2.4 SQL/PSM (Persistent Stored Modules)

Το SQL/PSM στάνταρτ επιτρέπει το διαχωρισμό των εφαρμογών σε client και server και περιγράφει το συντακτικό και τη σημασιολογία μιας γλώσσας για τον ορισμό και τη διαχείριση persistent database language routines σε modules που τρέχουν σε σχεσιακά ΣΔΒΔ.

Η γλώσσα προγραμματισμού που ορίζει αυτό το τμήμα είναι υπολογιστικά πλήρης και αποτελεί ουσιαστικά ένα μίγμα των πιο δημοφιλών και πρακτικών χαρακτηριστικών διαφόρων γλωσσών προγραμματισμού. Περιέχει, έτσι, όλες τις κλασσικές προγραμματιστικές δομές, όπως έλεγχο ροής, μεταβλητές και εκφράσεις και χειρισμό εξαιρέσεων, επιτρέπει την κλήση συναρτήσεων που είναι γραμμένες σε κάποια άλλη γλώσσα 3ης γενιάς, ενώ παρέχει και την δυνατότητα ορισμού modules τα οποία υπάρχουν είτε στην βάση, είτε μόνο στην εφαρμογή-πελάτη.

3.2.5 SQL/Bindings

Το SQL/Bindings χρησιμοποιείται για δύο βασικά λόγους:

- Δυναμική (dynamic) SQL
- Βελτίωση της ενσωματωμένης (embedded) SQL

Με άλλα λόγια, το SQL/Bindings στάνταρτ πρακτικά τυποποιεί τον τρόπο με τον οποίο SQL εντολές θα ενσωματώνονται σε host γλώσσες για τη συγγραφή προγραμμάτων. Πιο αναλυτικά, οι δύο τομείς στους οποίους το SQL/Bindings επιλαμβάνεται της κατάστασης είναι:

- Την σύνταξη ενσωματωμένων SQL-δηλώσεων (statements) σε ένα πρόγραμμα το οποίο είναι γραμμένο σε μια συγκεκριμένη γλώσσα προγραμματισμού (host language).
- Τον τρόπο με τον οποίο παράγεται ένα ισοδύναμο πρόγραμμα έτσι ώστε να ταιριάζει στο συγκεκριμένο στάνταρτ της γλώσσας προγραμματισμού. Σε ένα τέτοιο ισοδύναμο πρόγραμμα, κάθε ενσωματωμένη SQL-δήλωση έχει αντικατασταθεί από μία ή περισσότερες

δηλώσεις της συγκεκριμένης γλώσσας προγραμματισμού. Μερικές από αυτές τις δηλώσεις μπορούν να καλούν εξωτερικές διαδικασίες SQL οι οποίες όταν εκτελούνται, επιστρέφουν τα ίδια αποτελέσματα όπως όταν εκτελούνται οι αρχικές SQL-δηλώσεις.

3.2.6 SQL/Transaction

Το SQL/Transaction καθορίζει τον τρόπο με τον οποίο ένα σύστημα συμβατό με SQL μπορεί να συμμετάσχει σε δοσοληψίες (transactions) στις οποίες εμπλέκονται διάφοροι διαχειριστές πόρων (multiple resource managers), οι οποίοι μπορεί να μην είναι καν σχεσιακά ΣΔΒΔ. Το SQL/Transaction αποτελεί εξειδίκευση του XA και τυποποιεί ένα XA interface για υλοποιήσεις SQL. Το SQL/Transaction ασχολείται με την τυποποίηση κατανεμημένων δοσοληψιών από πολλαπλούς διαχειριστές πόρων και δοσοληψιών.

3.2.7 SQL/Temporal

Το SQL/Temporal καθορίζει τον τρόπο με τον οποίο δεδομένα χρονικού τύπου μπορούν να υποστηριχθούν από την SQL.Η πρόσθεση χρονικών χαρακτηριστικών στα δεδομένα βασίζεται κυρίως στο TSQL-2 στάνταρτ. Η γλώσσα ορισμού και ερωτήσεων δεδομένων περιλαμβάνει την υποστήριξη *χρόνου εγκυρότητας (valid time)*, *χρόνου δοσοληψίας (transaction time)* και *διπλού χρόνου (bitemporal support)* καθώς και τη δυνατότητα διαχειρίσεως δεδομένων τύπου `PERIOD` μέσα από ειδικούς τελεστές διαχείρισης χρονικών διαστημάτων.

3.2.8 SQL/Object

Το SQL/Object αποτελεί την κορυφαία προσπάθεια προσθήκης αντικειμενοστρεφών χαρακτηριστικών στο σχεσιακό μοντέλο. Οι χρήστες μπορούν με το SQL/Object να ορίζουν *αφηρημένους τύπους δεδομένων (Abstract Data Types - ADTs)*, *επώνυμους τύπους γραμμής (named rows)*, *τύπους αναφοράς (reference types)* και όλα αυτά σε περιβάλλον που υποστηρίζεται η *κληρονομικότητα (inheritance)* και η *ενθυλάκωση (encapsulation)*.

3.3 SQL/Foundation

3.3.1 Τύποι δεδομένων

Το SQL/Foundation στάνταρτ εισάγει νέους *τύπους δεδομένων* και συγκεκριμένα τους εξής:

- `BOOLEAN`
- `LOB` (Large Object)
- `LOB LOCATOR` - για τη διαχείριση των `LOBs`

Ο τύπος `BOOLEAN` είναι ο γνωστός από τις γλώσσες προγραμματισμού τύπος δυαδικής λογικής. Ο τύπος `LOB` προορίζεται για την αποθήκευση και διαχείριση δεδομένων μεγάλου όγκου και ειδικής λειτουργικότητας, όπως video, image κλπ. Τα `LOBs` είναι είτε αντικείμενα *χαρακτήρων* (character) είτε *δυαδικά* (binary) αντικείμενα. Η διαχείριση των `LOBs` γίνεται μέσω `LOB LOCATORs` οι οποίοι επιτρέπουν την πρόσβαση σε τμήματα των `LOBs` (εν είδη pointers) καθώς και την πιο ευέλικτη εκτέλεση λειτουργιών σε αυτά. Σαν παράδειγμα μπορούμε να αναφέρουμε ένα `LOB` το οποίο αντιπροσωπεύει μια εικόνα, και ένα `LOB LOCATOR` το οποίο μας επιτρέπει να προσπελαύνουμε συγκεκριμένα τμήματα της εικόνας (χωρίς να είναι ανάγκη να διαβάσουμε όλη την εικόνα στην μνήμη).

3.3.2 Κατηγορήματα

Επιπλέον, το SQL/Foundation στάνταρτ εισάγει τρία νέα είδη ***κ****ατηγορημάτων*:

- SIMILAR (π.χ. x SIMILAR TO pattern)
- BOOLEAN (π.χ. expression IS FALSE/TRUE/UNKNOWN)
- Type predicate - που θα αναλυθεί εκτενώς στην ενότητα "Object -Relational επεκτάσεις στην SQL".

Το κατηγόρημα SIMILAR μοιάζει με το κατηγόρημα LIKE της SQL92. Η λειτουργία του είναι η σύγκριση μιας συμβολοσειράς με ένα πρότυπο που ακολουθεί την σύνταξη των κανονικών εκφράσεων που συναντάμε και στην εντολή grep του UNIX. Υπάρχουν δηλαδή οι ειδικοί μεταχαρακτήρες *, . , [ ] οι οποίοι "ταιριάζουν" (match) με ομάδες χαρακτήρων από την αρχική συμβολοσειρά. Στο σχήμα 3.1 φαίνεται ένα παράδειγμα χρήσης της SIMILAR, στο οποίο βρίσκουμε όλα τα ονόματα των υπαλλήλων που αρχίζουν από την συμβολοσειρά ASTE και τελειώνουν στην συμβολοσειρά X.

```
SELECT first_name, last_name
FROM employee e
WHERE e.last_name SIMILAR TO ASTE.*X
```

Σχήμα 3.1 Παράδειγμα χρήσης του κατηγορήματος SIMILAR

Το κατηγόρημα συντάσσεται επίσης μαζί με την επιλογή ESCAPE, με την οποία μπορούμε να αφαιρέσουμε από ορισμένους μετα-χαρακτήρες την ειδική τους λειτουργία (όταν δηλαδή θέλουμε να συμμετέχουν ως κανονικοί χαρακτήρες στο πρότυπο που συγκρίνουμε). Το σχήμα 3.2 παρουσιάζει ένα παράδειγμα αυτής της σύνταξης, όπου βρίσκουμε τα τμήματα της εταιρίας που στο όνομα τους έχουν την συμβολοσειρά [A].

```
SELECT name
FROM dept d
WHERE d.name SIMILAR TO .*[A].* ESCAPE []
```

Σχήμα 3.2 Παράδειγμα χρήσης του κατηγορήματος SIMILAR με την επιλογή ESCAPE

Το κατηγόρημα BOOLEAN χρησιμοποιούμενο στο WHERE clause μιας ερώτησης μας επιτρέπει να ελέγξουμε αν μια έκφραση τύπου BOOLEAN είναι αληθής (IS TRUE), ψευδής (IS FALSE) ή δεν ξέρουμε την τιμή της (IS UNKNOWN).

3.3.3 Αναδρομικός τελεστής

Η έλλειψη αναδρομής ήταν ένα από τα στοιχεία που χαρακτήριζαν την SQL σε όλες τις εκδόσεις της. Ήταν δε και ένας από τους λόγους για τους οποίους είχε κατηγορηθεί ως μη καθαρά σχεσιακή γλώσσα (έχουσα δηλαδή, σημαντική απόσταση από το καθαρό σχεσιακό μοντέλο). Για να ξεπεράσει τον περιορισμό αυτό, το SQL/Foundation στάνταρτ εισάγει και ένα καινούριο σχεσιακό τελεστή. Για να δείξουμε τη λειτουργικότητά του εν λόγω τελεστή (με όνομα WITH RECURSIVE) θα χρησιμοποιήσουμε το παράδειγμα του σχήματος 3.3.

Στο παράδειγμα αυτό υπολογίζουμε από τον πίνακα parents (ο οποίος έχει δύο πεδία name, και parent), το γενεαλογικό δέντρο ενός συγκεκριμένου προσώπου. Η λειτουργία του τελεστή στο συγκεκριμένο παράδειγμα προχωράει ως εξής:

- Στην αρχή της αναδρομής, ο πίνακας ancestors αρχικοποιείται από τα πεδία του πίνακα parents για τα οποία ισχύει ότι name = "Alkis"

- Στην συνέχεια, εκτελείται η ζεύξη (join) του πίνακα `ancestors` με τον πίνακα `parents`, βρίσκονται οι "παππούδες" του `"Alkis"` και οι καινούριες σειρές τοποθετούνται στον πίνακα `ancestors`.
- Μετά εκτελείται και πάλι η ζεύξη και βρίσκονται οι "προ-παππούδες" κ.ο.κ.

```
WITH RECURSIVE
    ancestors(name,parent) AS
    (SELECT name,parent
     FROM parents
     WHERE name="Alkis"
     UNION ALL
     SELECT parents(name), parents(parent)
     FROM ancestors a, parents p
     WHERE a.parent=p.name)
SELECT *
ORDER BY name,parent
```

Σχήμα 3.3 Παράδειγμα του τελεστή WITH RECURSIVE

Το ακριβές συντακτικό του τελεστή `WITH RECURSIVE` περιγράφεται στο Σχήμα 3.4.

```
WITH RECURSIVE
    query_name (col-list)
    AS (query expression)
    [search clause]
    [limit clause]
query expression
```

Σχήμα 3.4 Το συντακτικό του τελεστή WITH RECURSIVE

3.3.4 Δοσοληψίες

Το SQL/Foundation εισάγει και νέες εντολές για τη διαχείριση των δοσοληψιών. Πιο συγκεκριμένα, αυτές είναι:

- `START TRANSACTION (READ ONLY/READ WRITE)`
- `SET LOCAL TRANSACTION` (για transactions σε παραπάνω από ένα server)
- `CHAINED TRANSACTIONS (COMMIT/ROLLBACK AND CHAIN)`
- `SAVEPOINTS (ROLLBACK TO SAVEPOINT name)`

Η εντολή `START TRANSACTION` αρχίζει μια καινούρια δοσοληψία. Η επιλογή `READ ONLY (READ WRITE)` υποδηλώνει το είδος των πράξεων που θα εκτελέσει η δοσοληψία στην βάση, δηλαδή μόνο διαβάσματα ή διαβάσματα και ενημερώσεις. Το τελευταίο έχει να κάνει κυρίως με θέματα συγχρονισμού και κλειδώματος, και αποσκοπεί στην καλύτερη δρομολόγηση των δοσοληψιών από την βάση. Η εντολή αυτή έχει ορισμένες επιπλέον επιλογές, οι οποίες αφορούν κυρίως το επίπεδο απομόνωσης (isolation level) της δοσοληψίας στην βάση.

Η εντολή `SET LOCAL TRANSACTION` έχει τις ίδιες επιλογές όπως και η προηγούμενη εντολή αλλά επηρεάζει την συμπεριφορά δοσοληψιών που εκτελούνται τοπικά σε μία βάση. Έχει, δηλαδή, νόημα η χρήση της, όταν μια δοσοληψία εκτελείται σε παραπάνω από ένα εξυπηρετητές. Η εντολή επιτρέπει την βελτιστοποίηση τέτοιου είδους δοσοληψιών (που

εκτελούνται κατανεμημένα), όπως π.χ. να εκτελεστεί το κομμάτι της δοσοληψίας που έχει μόνο διαβάσματα τοπικά και όχι σε απομακρυσμένο εξυπηρετητή.

Το κομμάτι του στάνταρτ που αναφέρεται στα CHAINED TRANSACTIONS επιτρέπει την διαδοχική εκτέλεση δοσοληψιών, σαν να ήταν σε "αλυσίδα". Κάθε δοσοληψία αρχίζει μόλις τελειώσει η προηγούμενη, και κληρονομεί τα χαρακτηριστικά της. Δεν υπάρχει δηλαδή η δυνατότητα να χρησιμοποιήσουμε την εντολή SET TRANSACTION για να αλλάξουμε τις ιδιότητες της καινούριας δοσοληψίας. Μεταβατικά, αυτό σημαίνει ότι η πρώτη δοσοληψία της αλυσίδας καθορίζει και τα χαρακτηριστικά όλων των υπόλοιπων. Οι εντολές COMMIT AND CHAIN και ROLLBACK AND CHAIN είναι αυτές που "δημιουργούν" την αλυσίδα. Η διαφορά τους έγκειται στο τρόπο που τελειώνει η δοσοληψία, επιτυχώς δηλαδή ή ανεπιτυχώς (ABORT).

Τα SAVEPOINTS αποτελούν "σημεία" μέσα στην εκτέλεση μιας δοσοληψίας, τα οποία ουσιαστικά καταγράφουν την κατάσταση της βάσης (και της δοσοληψίας) εκείνη την χρονική στιγμή. Η δοσοληψία, μετά, έχει την δυνατότητα να κάνει ένα μερικό rollback και να "επιστρέψει" σε κάποιο από τα savepoints που είχε δημιουργήσει. Η εντολή SAVEPOINT name δημιουργεί ένα καινούριο savepoint, στο οποίο η δοσοληψία μπορεί να επιστρέψει με την εντολή ROLLBACK TO SAVEPOINT name. Η δοσοληψία μπορεί επίσης να καταργήσει ένα savepoint με την εντολή RELEASE SAVEPOINT name. Εσωτερικά στην βάση, τα savepoints μοντελοποιούνται ως φωλιασμένες υπο-δοσοληψίες της αρχικής, και το rollback στο savepoint υλοποιείται σαν το ABORT της αντίστοιχης υπο-δοσοληψίας.

3.3.5 Ασφάλεια

Η διαχείριση των δικαιωμάτων των χρηστών εμπλουτίζεται στο SQL/Foundation με την εισαγωγή της έννοιας των *ρόλων*. Οι ρόλοι λειτουργούν σαν τα δικαιώματα, με τη διαφορά ότι δεν απευθύνονται σε συγκεκριμένους χρήστες, αλλά καταγράφουν δραστηριότητες σε ένα οργανισμό. Η οργάνωση αυτή έχει το πλεονέκτημα ότι αποπροσωποποιεί τα δικαιώματα με αποτέλεσμα τη μεγαλύτερη ευελιξία στη διαχείρισή τους. Ένας ρόλος μπορεί να ανατεθεί σε ένα χρήστη της βάσης (ο οποίος θα αποκτήσει όλα τα δικαιώματα του ρόλου), ή μπορεί να ανατεθεί σε κάποιον άλλο ρόλο.

Στο παράδειγμα που ακολουθεί, ο ρόλος "payroll" μοντελοποιεί το πρόσωπο που διεξάγει τη διαδικασία πληρωμών για τους υπαλλήλους, και του ανατίθεται το δικαίωμα να εκτελεί επερωτήσεις μόνο στον αντίστοιχο πίνακα. Στην συνέχεια, ο ρόλος αυτός ανατίθεται σε κάποιο συγκεκριμένο χρήστη μέσα στον οργανισμό (χρήστης "john").

```
CREATE ROLE payroll
GRANT select ON emp TO payroll
GRANT payroll TO john
```

Αντίστοιχα με τις εντολές δημιουργίας και ανάθεσης ρόλων, υπάρχουν και οι εντολές ανάκλησης και κατάργησης:

```
REVOKE ROLE payroll FROM john
DROP ROLE payroll
```

3.3.6 Ενεργοποιητές (triggers)

Στο SQL/Foundation η χρήση των *ενεργοποιητών* (triggers) καθορίζεται αυστηρά. Οι ενεργοποιητές είναι διαδικασίες οι οποίες, δεδομένης μιας πράξης που επιτελείται στη βάση δεδομένων, και μιας συγκεκριμένης κατάστασης της βάσης, ενεργοποιούν μια συγκεκριμένη διαδικασία (εκφραζόμενη σαν μια SQL εντολή). Πιο συγκεκριμένα, ένας ενεργοποιητής εκφράζεται συναρτήσει ενός *γεγονότος*, του *χρόνου* που ο ενεργοποιητής πυροδοτείται σε σχέση με το γεγονός, μιας *συνθήκης* στη βάση δεδομένων και μιας *πράξης* που είναι το αποτέλεσμα της πυροδότησής του. Οι τιμές που μπορούν να πάρουν αυτές οι παράμετροι είναι οι εξής:

- *Γεγονότα*: INSERT/DELETE/UPDATE
- *Χρόνος που συμβαίνουν*: BEFORE/AFTER
- *Συνθήκη*: οποιαδήποτε SQL συνθήκη
- Πράξη: οποιαδήποτε SQL εντολή

Δύο επιπλέον χαρακτηριστικά στον ορισμό των ενεργοποιητών είναι αφενός η δυνατότητα προσπέλασης παλιών και νέων τιμών των επηρεαζόμενων rows και αφετέρου το γεγονός ότι οι κανόνες ελέγχονται είτε ανά γεγονός, είτε ανά row. Η εκτέλεση της πράξης ενός ενεργοποιητή μπορεί να προκαλέσει ένα γεγονός το οποίο θα πυροδοτήσει κάποιον άλλο ενεργοποιητή κ.ο.κ. Το φαινόμενο αυτό ονομάζεται "trigger cascading" και το πρόβλημα που μπορεί να δημιουργηθεί είναι ένας βρόγχος από triggers, που ο ένας πυροδοτεί τον άλλο και η διαδικασία δεν τελειώνει ποτέ.

Το παράδειγμα που ακολουθεί, παρουσιάζει την δημιουργία ενός trigger που "ελέγχει" την εισαγωγή στον πίνακα των υπαλλήλων μιας εταιρίας:

```
CREATE TRIGGER new_hire
AFTER INSERT ON emps
REFERENCING NEW AS n
UPDATE corp_status
    SET no_of_emps = no_of_emps + 1
    WHERE n.corp_id = corp.id;
```

Το γεγονός που θα πυροδοτήσει τον ενεργοποιητή `new_hire` είναι η εισαγωγή στον πίνακα `emps`, και μάλιστα ως χρονικό σημείο καθορίζεται η στιγμή μετά την εκτέλεσή της (`AFTER INSERT ON emps`). Στο συγκεκριμένο παράδειγμα, δεν έχουμε καθορίσει συνθήκη για την πυροδότηση του ενεργοποιητή, που σημαίνει ότι θέλουμε να εκτελείται η διαδικασία *κάθε* φορά, *αμέσως μετά* την εισαγωγή μιας καινούριας πλειάδας. Η πράξη του συγκεκριμένου ενεργοποιητή είναι η ενημέρωση ενός μετρητή στον πίνακα της εταιρίας (`UPDATE corp_status SET no_of_emps = no_of_emps + 1 WHERE n.corp_id = corp.id;`) όπου χρησιμοποιείται και ένα πεδίο της καινούριας πλειάδας (`n.corp_id`). Το όνομα της καινούριας πλειάδας έχει δοθεί πιο πριν με την εντολή `REFERENCING NEW AS n`. Με εντελώς ανάλογο τρόπο, ένας ενεργοποιητής μπορεί να αναφερθεί στην προηγούμενη τιμή μιας πλειάδας, χρησιμοποιώντας την εντολή `REFERENCING OLD AS n`.

Η μεγάλη χρησιμότητα των ενεργοποιητών έγκειται στο ότι βοηθούν στην διατήρηση της ακεραιότητας των δεδομένων, αφαιρώντας αυτήν την ευθύνη από την εφαρμογή και δίνοντας την στην ίδια την βάση (στην οποία και ανήκει).

3.3.7 SQL client modules

Στο SQL/Foundation επιτρέπεται ο ορισμός modules τα οποία βρίσκονται στους clients. Τα client modules περιέχουν ρουτίνες γραμμένες σε καθαρή SQL που μπορούν να κληθούν από άλλες γλώσσες (όπως για παράδειγμα C, Fortran, Cobol κλπ.). Σε αντίθεση με τα persistent stored modules (βλ. SQL/PSM), τα client modules δεν συμμετέχουν στο σχήμα της βάσης. Υπάρχουν μόνο στην "μεριά" του πελάτη.

Το στάνταρτ καθορίζει ορισμένες προϋποθέσεις για τις ρουτίνες αυτές. Πιο συγκεκριμένα, οι ρουτίνες πρέπει να έχουν μια παράμετρο κατάστασης με το όνομα `SQLCODE` και οι παράμετροι πρέπει να έχουν αντίστοιχο τύπο στις εξωτερικές γλώσσες που τις καλούν. Στα δύο σχήματα που ακολουθούν, φαίνεται ο ορισμός ενός client module που επιστρέφει το balance ενός λογαριασμού και ο τρόπος κλήσης του module μέσα από C.

```
MODULE trxn
LANGUAGE C
AUTHORIZATION admin
PROCEDURE get_balance
    (SQLCODE,
    :acct INTEGER,
    :bal REAL);
    SELECT balance INTO :bal
    FROM accounts
    WHERE account = :acct;
```

Σχήμα 3.5 Ορισμός client module

```
main ( )
{   long SQLCODE;
    long acct_no;
    real bal;
    ...
    get_balance (&SLQCODE,
    acct_no, bal);
}
```

Σχήμα 3.6 Κλήση client module μέσα από C

3.4 SQL/CLI (Call Level Interface)

Το στάνταρτ αυτό μοιάζει στην δομή (αλλά και στην σκοπιμότητα) με το ODBC και την Dynamic SQL. Προσδιορίζει ένα λειτουργικό interface για μια βάση, μέσα από το οποίο μια εφαρμογή μπορεί να εκτελέσει SQL εντολές, χωρίς όμως να είναι αναγκαίο να μεσολαβήσει ένα προεπεξεργαστής στον πηγαίο κώδικα. Πιο συγκεκριμένα, καθορίζονται συναρτήσεις για τις παρακάτω λειτουργίες:

- Σύνδεση στο server
- Παραχώρηση και αποδέσμευση πόρων
- Εκτέλεση SQL εντολών
- Διαγνωστικά
- Διαχείριση δοσοληψιών
- Διάφορες βοηθητικές λειτουργίες

Η βασική έννοια στο SQL/CLI είναι αυτή της *αναφοράς (handle)*. Για κάθε πόρο που δεσμεύει η εφαρμογή, της επιστρέφεται ένα handle το οποίο καθορίζει μοναδικά αυτόν τον πόρο στο περιεχόμενο (context) της εφαρμογής. Οι βασικοί πόροι τους οποίους χειρίζεται μια εφαρμογή μέσω του SQL/CLI είναι το *περιβάλλον (SQL-Environment)*, η *σύνδεση (SQL-Connection)*, η *εντολή (SQL-Statement)* και η *περιγραφική περιοχή (CLI descriptor area)*. Οι πόροι αυτοί είναι τοποθετημένοι σε ένα είδος ιεραρχίας, με την έννοια ότι μια σύνδεση μπορεί να υπάρξει μόνο μέσα σε ένα περιβάλλον, ενώ μία εντολή ή μια περιγραφική περιοχή μπορεί να υπάρξει μόνο μέσα σε μία σύνδεση. Για να χρησιμοποιηθεί λοιπόν το SQL/CLI είναι αναγκαίο να ακολουθηθούν τα παρακάτω βήματα:

1. Δέσμευση περιβάλλοντος
2. Δέσμευση σύνδεσης σε συγκεκριμένο περιβάλλον
3. Δέσμευση εντολών σε συγκεκριμένη σύνδεση και εκτέλεσή τους.
4. Αποδέσμευση πόρων με σειρά αντίστροφη από αυτή με την οποία παραχωρήθηκαν.

Το στάνταρτ καθορίζει την γενική συνάρτηση `AllocHandle` η οποία δεσμεύει πόρους και επιστρέφει τις αντίστοιχες αναφορές. Εκτός από αυτή τη γενική συνάρτηση, υπάρχουν "εξειδικευμένες" συναρτήσεις για την δέσμευση πόρων για περιβάλλον, συνδέσεις, εντολές και περιγραφικές περιοχές, όπως οι `AllocEnv, AllocConnect` κ.τ.λ. Αντίστοιχα υπάρχουν οι συναρτήσεις `FreeHandle, FreeEnv, FreeConnect` κ.τ.λ. οι οποίες αποδεσμεύουν πόρους που είχαν δεσμευτεί με κλήση των `Alloc` συναρτήσεων. Αφού δεσμευθούν οι αντίστοιχοι πόροι, υπάρχουν άλλες συναρτήσεις οι οποίες επιτρέπουν τον χειρισμό τους, όπως π.χ. τον καθορισμό της βάσης στην οποία αναφέρεται το SQL-Connection ή της εντολής που θα εκτελεστεί από ένα SQL-Statement.

Η εκτέλεση εντολών στο SQL/CLI είναι στενά συνδεμένη με την έννοια της περιγραφικής περιοχής. Οποιαδήποτε δυναμική πληροφορία που σχετίζεται με την εκτέλεση της εντολής, είτε αφορά δεδομένα εισόδου είτε δεδομένα εξόδου, βρίσκεται σε κάποια περιγραφική περιοχή που έχει συσχετισθεί με την εντολή. Σαν παράδειγμα, αναφέρουμε τις δυναμικές παραμέτρους μιας εντολής SQL, οι οποίες περιγράφονται και καθορίζονται, όταν έρθει η στιγμή της εκτέλεσης, μέσα στην αντίστοιχη περιγραφική περιοχή. Η μορφή των αποτελεσμάτων επίσης που επιστρέφονται από την εκτέλεση μιας εντολής SQL, δίνεται μέσα σε κάποια περιγραφική περιοχή. Για την προσπέλαση των αποτελεσμάτων χρησιμοποιούνται κέρσορες (cursors) οι οποίοι δημιουργούνται αυτόματα με την εκτέλεση κάποιας εντολής. Εκτός όμως από τους κέρσορες, το στάνταρτ παρέχει ειδικές συναρτήσεις με τις οποίες η εφαρμογή μπορεί να ανακτήσει τα δεδομένα κομμάτι-κομμάτι.

Παρακάτω παρουσιάζεται ενδεικτικά ο σκελετός ενός προγράμματος που χρησιμοποιεί το SQL/CLI, και στο οποίο φαίνεται καθαρά η ιεράρχιση των πόρων που αναφέρθηκε:

```
AllocEnv()
    AllocConnect()
        Connect()
            AllocStmt()
                build the statement…
                Execute() -repeat as required...
            Transact() -repeat as required...
            FreeStmt()
        Disconnect()
    FreeConnect()
FreeEnv()
```

3.5 SQL/PSM (Persistent Stored Modules)

Το τμήμα αυτό του στάνταρτ περιγράφει ουσιαστικά μια υπολογιστικά πλήρη γλώσσα προγραμματισμού, η οποία μπορεί να χρησιμοποιηθεί για την δημιουργία modules. Τα modules αυτά βρίσκονται αποθηκευμένα στον server και μπορούν να χρησιμοποιηθούν από οποιονδήποτε πελάτη.

Ο χειρισμός των modules γίνεται με τις εντολές `CREATE MODULE`, `ALTER MODULE` και `DROP MODULE`, με ακριβώς παρόμοιο τρόπο με τον χειρισμό των πινάκων. Παρέχεται επίσης η δυνατότητα της αποθήκευσης των modules μέσα στο σχήμα μιας συγκεκριμένης βάσης.

Ένα module, με την σειρά του, μπορεί να περιέχει ένα σύνολο από διαδικασίες και συναρτήσεις, τις οποίες μπορούν να καλούν οι πελάτες της βάσης. Ο χειρισμός των ρουτινών (ο όρος αναφέρεται και σε διαδικασίες και σε συναρτήσεις) γίνεται μέσα από παρόμοιες εντολές με αυτές για modules: `CREATE PROCEDURE, CREATE FUNCTION, DROP SPECIFIC ROUTINE`. Οι ρουτίνες, όπως είπαμε, μπορούν να ανήκουν σε κάποιο module, μπορούν όμως να αποθηκεύονται κατευθείαν και στο σχήμα μιας βάσης. Όσον αφορά το είδος τους, χωρίζονται σε δύο κατηγορίες:

- SQL routines
- external routines

3.5.1 SQL routines

Οι ρουτίνες αυτές είναι γραμμένες στην υπολογιστικά πλήρη γλώσσα που ορίζει το SQL/PSM. Εκτός από την δυνατότητα εκτέλεσης SQL εντολών, η γλώσσα αυτή παρέχει τις παρακάτω δομές:

```
compound statement  BEGIN…END;
variable declaration     DECLARE var CHAR (6)
if statement        IF subject (var) <> 'urgent' THEN … ELSE…;
case statement      CASE subject (var)
                              WHEN 'SQL' THEN…
                              WHEN …;
loop statement      LOOP <SQL statements list> END LOOP;
statement                WHILE i < 100 DO … END WHILE;
repeat statement    REPEAT … UNTIL i<100 END REPEAT;
of statement        FOR result AS … DO … END FOR;
leave statement          LEAVE …;
return statement    RETURN 'urgent';
call statement      CALL procedure_x (1, 3, 5,);
assignment statement     SET x = 'abc';
signal / resignal   SIGNAL division_by_zero
```

Σαν παράδειγμα παρουσιάζουμε μια ρουτίνα η οποία υπολογίζει τον αριθμό των πλειάδων ενός πίνακα με υπαλλήλους που έχουν το ίδιο όνομα, όπου η τελευταία πληροφορία περνάει σαν παράμετρος στην κλήση της ρουτίνας.

```
CREATE FUNCTION
    catalog1.schema2.num_dups
    (last CHAR(20),
     first CHAR(20),
     middle CHAR(20))
    RETURNS INTEGER;
BEGIN
    DECLARE number INTEGER;
    SELECT COUNT(*) INTO number
    FROM employee e
    WHERE e.last_name=last AND e.first_name=first AND
          e.middle_name=middle;
    RETURN number;
END
```

3.5.2 External routines

Οι ρουτίνες αυτές είναι γραμμένες σε κάποια host language με SQL bindings, και καλούνται μέσα από ένα wrapper το οποίο ορίζεται με την SQL/PSM. Το στάνταρτ καθορίζει, δηλαδή, ένα πρωτόκολλο επικοινωνίας μεταξύ αυτών των ρουτινών και της ρουτίνας που υπάρχει στο σχήμα της βάσης ή μέσα σε ένα module. Το πρωτόκολλο αυτό απαιτεί ότι κάθε τέτοια ρουτίνα πρέπει να χρησιμοποιεί μια παράμετρο τύπου `CHAR(5)` η οποία ονομάζεται `SQLSTATE` και η οποία κωδικοποιεί την έκβαση της ρουτίνας. Είναι απαραίτητο, επίσης, για κάθε μεταβλητή να καθορίζεται αν είναι μεταβλητή εισόδου ή εξόδου (ή και τα δύο) καθώς και ο τύπος επιστροφής της ρουτίνας αν είναι συνάρτηση.

Τα πλεονεκτήματα των εξωτερικών ρουτινών είναι η επαναχρησιμοποίηση υπάρχοντος λογισμικού, και η υπολογιστική πληρότητα (καθώς υπάρχει η δυνατότητα εκτέλεσης λειτουργιών εισόδου/εξόδου). Τα μειονεκτήματα είναι ότι οδηγούμαστε σε ένα ανομοιογενές περιβάλλον, υπάρχει η ανάγκη μετατροπής των τύπων μεταξύ βάσης και της εξωτερικής γλώσσας και κυρίως το αντιμετωπίζουμε πλέον θέμα ασφάλειας, καθώς δεν υπάρχει κανένας έλεγχος στον κώδικα που εκτελείται από την εξωτερική ρουτίνα.

Σαν παράδειγμα, δίνεται ο σκελετός του ορισμού μιας εξωτερικής ρουτίνας η οποία υπολογίζει το συνημίτονο ενός πραγματικού αριθμού, καλώντας μια εξωτερική ρουτίνα γραμμένη σε FORTRAN:

```
CREATE FUNCTION sin(FLOAT)
    RETURNS float
EXTERNAL NAME 'LIB%MATH-SUBS\SINE'
LANGUAGE FORTRAN...
```

3.6 SQL/BINDINGS

3.6.1 Εισαγωγή

Ο όρος "bindings" είναι δύσκολο να αποδοθεί στα ελληνικά. Πιστεύουμε ότι η κατανόηση του όρου θα γίνει μέσα από αυτά που θα αναφέρουμε παρακάτω. Το SQL/Bindings χρησιμοποιείται για δύο βασικά λόγους:

- Δυναμική (dynamic) SQL
- Βελτίωση της ενσωματωμένης (embedded) SQL

Το SQL/Bindings στάνταρτ καθορίζει:

- Την σύνταξη ενσωματωμένων SQL-δηλώσεων (statements) σε ένα πρόγραμμα το οποίο είναι γραμμένο σε μια συγκεκριμένη γλώσσα προγραμματισμού (host language).
- Τον τρόπο με τον οποίο παράγεται ένα ισοδύναμο πρόγραμμα έτσι ώστε να ταιριάζει στο συγκεκριμένο στάνταρτ της γλώσσας προγραμματισμού. Σε ένα τέτοιο ισοδύναμο πρόγραμμα, κάθε ενσωματωμένη SQL-δήλωση έχει αντικατασταθεί από μία ή περισσότερες δηλώσεις της συγκεκριμένης γλώσσας προγραμματισμού. Μερικές από αυτές τις δηλώσεις μπορούν να καλούν εξωτερικές διαδικασίες SQL οι οποίες όταν εκτελούνται, επιστρέφουν τα ίδια αποτελέσματα όπως όταν εκτελούνται οι αρχικές SQL-δηλώσεις.

3.6.2 Περιγραφή

Κέρσορες (Cursors)

Ένας *κέρσορας* καθορίζεται από μία δήλωση <dynamic declare cursor> ή από μία <allocate cursor statement>. Ένας κέρσορας που ορίζεται από μια δήλωση <dynamic declare cursor> δηλώνεται σαν ένας *δυναμικός κέρσορας*. Ένας κέρσορας που ορίζεται από μια <allocate cursor statement> είναι ένας *εκτεταμένος* (extended) δυναμικός κέρσορας. Ένας δυναμικός κέρσορας είναι είτε *δηλωμένος* (declared) είτε *εκτεταμένος* (extended). Για κάθε δήλωση <dynamic declare cursor> μέσα σε ένα <SQL-client module>, ο κέρσορας δημιουργείται όταν μια SQL-δοσοληψία (transaction) που αναφέρεται στο <SQL-client module> αρχικοποιείται. Ένας εκτεταμένος δυναμικός κέρσορας δημιουργείται όταν εκτελείται μια <allocate cursor statement> μέσα σε ένα SQL-session και καταστρέφεται όταν η SQL-session τερματίζεται. Αντίστοιχα, ένας δυναμικός κέρσορας καταστρέφεται όταν εκτελείται μια <deallocate prepared statement> η οποία απελευθερώνει την προετοιμασμένη δήλωση στην οποία είναι βασισμένος ο κέρσορας. Ένας κέρσορας δημιουργείται επίσης από μια <dynamic open statement> και επιστρέφει σε κατάσταση κλεισίματος με μια <dynamic close statement>. Η <dynamic fetch statement> τοποθετεί έναν ανοικτό κέρσορα σε μια συγκεκριμένη γραμμή (row) και κάνει ανάκτηση των πεδίων της συγκεκριμένης γραμμής. Μια <dynamic update statement: positioned> ενημερώνει την τρέχουσα γραμμή του κέρσορα. Μια <dynamic delete statement: positioned> διαγράφει την τρέχουσα γραμμή του κέρσορα.

Τύποι locators που ορίζονται από τον χρήστη (user-defined type locators)

Ένας *τύπος locator που ορίζεται από τον χρήστη (user-defined type, UDT)*, είναι μια τιμή η οποία παράγεται όταν μια UDT τιμή αποθηκεύεται σε μια <embedded variable name>. Ένας UDT locator αναφέρεται μοναδικά μια UDT θέση (site). Ένας UDT locator είναι πάντα περιορισμένος σε ένα συγκεκριμένο UDT, το οποίο αποκαλείται "συσχετισμένος ορισμένος από το χρήστη τύπος" του UDT locator. Ο τύπος των τιμών που αναγνωρίζονται από έναν UDT locator είναι ο συσχετισμένος user-defined type αυτού του UDT locator. Οι τιμές του UDT οι οποίες αποθηκεύονται σε έναν UDT locator πρέπει να είναι ένα στιγμιότυπο ενός υποτύπου του UDT. Όταν μια <embedded variable name> ενός UDT locator χρησιμοποιείται σε μια <value specification>, είναι ισοδύναμη με την τιμή του UDT value που της έχει ανατεθεί. Σε μια host μεταβλητή, ο UDT locator σχηματίζεται σαν μια τετραψήφια οκταδική τιμή η οποία μπορεί να αναπαρασταθεί με έναν ακέραιο τεσσάρων bytes. Ένας UDT locator δεν

μπορεί ποτέ να οριστεί σαν <data type> μιας στήλης ενός πίνακα, και κατά συνέπεια δεν αντιμετωπίζεται από το ΣΒΔ σαν ένας σταθερός τύπος δεδομένων.

Ένα χαρακτηριστικό ενός UDT locator είναι το αν είναι *holdable* ή όχι.

- Ένας UDT locator που δεν είναι holdable-UDT-locator ελευθερώνεται όταν η SQL-transaction στην οποία δημιουργήθηκε τερματίζεται.
- Ένας holdable-UDT-locator ο οποίος δεν έχει ελευθερωθεί με μια <free locator statement> μέσα σε μια SQL-transaction διατηρείται εάν η SQL-transaction τερματιστεί με μια <commit statement>.
- Ένας holdable-UDT-locator ελευθερώνεται αν η SQL-transaction τερματιστεί με μια <rollback-statement>.
- Ένας holdable-UDT-locator ελευθερώνεται αν η SQL-session στην οποία δημιουργήθηκε τερματιστεί.

SQL-δηλώσεις (statements)

Κλάσεις SQL-δηλώσεων

Οι περισσότερες SQL-statements μπορούν να *προετοιμαστούν* (prepared) για εκτέλεση (execution) και να *εκτελεστούν* με αντίστοιχους τρόπους. Αυτοί είναι:

- Σε ένα πρόγραμμα με ενσωματωμένη SQL, προετοιμάζονται όταν γίνεται προεπεξεργασία του προγράμματος.
- Προετοιμασία και εκτέλεση με τη χρήση δυναμικών SQL-δηλώσεων.
- Με απευθείας κλήση, όπου προετοιμάζονται αμέσως πριν από την εκτέλεση

Υπάρχουν τρεις τουλάχιστον πρόσθετοι τρόποι για να κατατάξουμε τις SQL-δηλώσεις:

- Σύμφωνα με το αν μπορούν να ενσωματωθούν
- Σύμφωνα με το αν μπορούν να προετοιμαστούν δυναμικά και να εκτελεστούν
- Σύμφωνα με το αν μπορούν να εκτελεστούν άμεσα.

Κατάλογοι (Catalogs)

Ο προκαθορισμένος κατάλογος για <preparable statement>s οι οποίες χρησιμοποιούνται δυναμικά σε ένα SQL-session εκτελώντας <prepare statement>s και <execute immediate statement>s, ορίζεται αρχικά αλλά μπορεί και να μεταβληθεί με τη χρήση του <set catalog statement>.

Κατάταξη SQL-δηλώσεων με βάση την λειτουργία

Οι ακόλουθες είναι άλλες κύριες κλάσεις από SQL-δηλώσεις:

- SQL-dynamic statements
- SQL embedded exception declaration

Οι ακόλουθες είναι πρόσθετες SQL-data δηλώσεις:

- <dynamic declare cursor>
- <allocate cursor statement>
- <dynamic select statement>

- <dynamic open statement>
- <dynamic close statement>
- <dynamic fetch statement>
- <direct select statement: multiple rows>
- <dynamic single row select statement>

Οι ακόλουθες είναι πρόσθετες SQL-data δηλώσεις αλλαγής:

- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

Οι ακόλουθες είναι πρόσθετες SQL-session δηλώσεις:

- <set catalog statement>
- <set schema statement>
- <set names statement>
- <set path statement>

Οι ακόλουθες είναι SQL-dynamic δηλώσεις:

- <execute immediate statement>
- <allocate descriptor statement>
- <deallocate descriptor statement>
- <get descriptor statement>
- <set descriptor statement>
- <prepare statement>
- <deallocate prepared statement>
- <describe input statement>
- <describe output statement>
- <execute statement>

Οι ακόλουθη είναι SQL embedded exception δήλωση:

- <embedded exception declaration>

SQL-δηλώσεις και καταστάσεις δοσοληψιών

Οι επόμενες πρόσθετες SQL-statements είναι δηλώσεις που ξεκινούν δοσοληψίες (transaction-initiating). Εάν δεν υπάρχει καμία τρέχουσα δοσοληψία και εκτελεστεί μια δήλωση αυτής της κλάσης, τότε αρχικοποιείται μια δοσοληψία:

- <allocate cursor statement>
- <dynamic open statement>
- <dynamic close statement>
- <dynamic fetch statement>
- <direct select statement: multiple rows>
- <dynamic single row select statement>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>

- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

Οι ακόλουθες είναι δυναμικές SQL δηλώσεις:

- <describe input statement>
- <describe output statement>
- <allocate descriptor statement>
- <deallocate descriptor statement>
- <get descriptor statement>
- <set descriptor statement>
- <prepare statement>
- <deallocate prepared statement>

Οι ακόλουθες πρόσθετες SQL-statements δεν αρχικοποιούν δοσοληψίες, δηλαδή αν δεν υπάρχει τρέχουσα δοσοληψία και εκτελεστεί κάποια από τις παρακάτω δηλώσεις, δεν θα αρχικοποιηθεί καμία δοσοληψία.

- SQL embedded exception declarations
- SQL-data statements:
  - <dynamic declare cursor>
  - <dynamic select statement>

SQL-δηλώσεις που μπορούν να ενσωματωθούν (embeddable)

Οι ακόλουθες δηλώσεις μπορούν να ενσωματωθούν σε ένα πρόγραμμα με ενσωματωμένη SQL. Αυτές μπορούν να είναι <SQL procedure statement> μέσα σε μια <externally-invoked procedure> σε ένα <SQL-client module>:

- Όλες οι SQL-schema δηλώσεις
- Όλες οι SQL-transaction δηλώσεις
- Όλες οι SQL-connection δηλώσεις
- Όλες οι SQL-session δηλώσεις
- Όλες οι SQL-control δηλώσεις
- Όλες οι SQL-dynamic δηλώσεις
- Όλες οι SQL-diagnostics δηλώσεις
- SQL-data δηλώσεις:
  - <allocate cursor statement>
  - <open statement>
  - <dynamic open statement>
  - <close statement>
  - <dynamic close statement>
  - <fetch statement>
  - <dynamic fetch statement>
  - <select statement: single row>
  - <insert statement>
  - <delete statement: searched>
  - <delete statement: positioned>
  - <dynamic delete statement: positioned>

- <update statement: searched>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <hold locator statement>
- <free locator statement>

Οι ακόλουθες SQL-δηλώσεις μπορούν να ενσωματωθούν σε ένα πρόγραμμα, και σε ένα <SQL-client module>, αλλά όχι σε μια <externally-invoked procedure>:

- <temporary table declaration>
- <declare cursor>
- <dynamic declare cursor>

Οι ακόλουθες SQL-δηλώσεις μπορούν να ενσωματωθούν σε ένα πρόγραμμα αλλά όχι σε ένα <SQL-client module>:

- SQL embedded exception declarations

Αντίθετα, οι ακόλουθες SQL-data δηλώσεις δεν ενσωματώνονται σε ένα πρόγραμμα, ούτε σε <SQL- client module>, ούτε σε <SQL procedure statement>:

- <dynamic select statement>
- <dynamic single row select statement>
- <direct select statement: multiple rows>
- <preparable dynamic delete statement: positioned>
- <preparable dynamic update statement: positioned>

SQL-δηλώσεις που μπορούν να προετοιμαστούν και να εκτελεστούν άμεσα (Preparable and immediately executable SQL-statements)

Οι ακόλουθες δηλώσεις μπορούν να προετοιμαστούν:

- Όλες οι SQL-schema δηλώσεις
- Όλες οι SQL-transaction δηλώσεις
- Όλες οι SQL-session δηλώσεις
- Όλες οι SQL-control δηλώσεις
- Οι ακόλουθες SQL-data δηλώσεις:
  - <delete statement: searched>
  - <dynamic select statement>
  - <dynamic single row select statement>
  - <insert statement>
  - <update statement: searched>
  - <preparable dynamic delete statement: positioned>
  - <preparable dynamic update statement: positioned>
  - <preparable implementation-defined statement>
  - <hold locator statement>
  - <free locator statement>

Αντίθετα, οι ακόλουθες SQL-δηλώσεις δεν μπορούν να προετοιμαστούν:

- Όλες οι SQL-connection δηλώσεις
- Όλες οι SQL-dynamic δηλώσεις
- Όλες οι SQL-diagnostics δηλώσεις
- SQL embedded exception δηλώσεις
- Οι ακόλουθες SQL-data δηλώσεις:
  - <allocate cursor statement>
  - <open statement>
  - <dynamic open statement>
  - <close statement>
  - <dynamic close statement>
  - <fetch statement>
  - <dynamic fetch statement>
  - <select statement: single row>
  - <delete statement: positioned>
  - <dynamic delete statement: positioned>
  - <update statement: positioned>
  - <dynamic update statement: positioned>
  - <direct select statement: multiple rows>
  - <temporary table declaration>
  - <declare cursor>
  - <dynamic declare cursor>

Οποιαδήποτε SQL-δήλωση που μπορεί να προετοιμαστεί μπορεί να εκτελεστεί άμεσα, με εξαίρεση τις ακόλουθες:

- <dynamic select statement>
- <dynamic single row select statement>

SQL-δηλώσεις απευθείας εκτέλεσης (directly executable SQL-statements)

- Όλες οι SQL-schema δηλώσεις
- Όλες οι SQL-transaction δηλώσεις
- Όλες οι SQL-connection δηλώσεις
- Όλες οι SQL-session δηλώσεις
- SQL-data δηλώσεις:
  - <temporary table declaration>
  - <direct select statement: multiple rows>
  - <insert statement>
  - <delete statement: searched>
  - <update statement: searched>

Αντίθετα, οι ακόλουθες SQL- δηλώσεις δεν μπορούν να εκτελεστούν απευθείας:

- Όλες οι SQL-dynamic δηλώσεις
- Όλες οι SQL-diagnostics δηλώσεις
- SQL embedded exception declarations
- SQL-data δηλώσεις:

- <declare cursor>
- <dynamic declare cursor>
- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <dynamic select statement>
- <dynamic single row select statement>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

Γλώσσες προγραμματισμού

Αυτό το μέρος του στάνταρτ περιγράφει τον μηχανισμό με τον οποίο μπορεί να ενσωματωθεί η SQL γλώσσα μέσα σε μια γλώσσα προγραμματισμού. Για λόγους επικοινωνίας με την γλώσσα προγραμματισμού, οι τύποι δεδομένων DATE, TIME, TIMESTAMP και INTERVAL πρέπει να μετατρέπονται σε σειρές χαρακτήρων (character strings) χρησιμοποιώντας την <cast specification>. Έχει γίνει πρόβλεψη ώστε οι μελλοντικές εκδόσεις των γλωσσών να υποστηρίζουν τύπους δεδομένων που να ανταποκρίνονται σε αυτούς τους τέσσερις τύπους. Ο τύπος δεδομένων CHARACTER μετατρέπεται επίσης σε σειρές χαρακτήρων στις γλώσσες προγραμματισμού. Για συγκεκριμένες γλώσσες προγραμματισμού όπως C, COBOL, FORTRAN και Pascal οι σειρές bit αντιστοιχίζονται σε μεταβλητές χαρακτήρων. Στην Ada και στην PL/I υποστηρίζονται μεταβλητές σειρών bit.

Σύνταξη ενσωματωμένης (embedded) γλώσσας

Κάθε <embedded SQL host program> είναι μια μεταγλωττισμένη μονάδα που αποτελείται από το κείμενο της γλώσσας προγραμματισμού και της SQL. Το κείμενο που περιέχει δηλώσεις της γλώσσας προγραμματισμού είναι προσαρμοσμένο στις απαιτήσεις της συγκεκριμένης γλώσσας. Το κείμενο που περιέχει SQL θα έχει μια ή περισσότερες <embedded SQL statement>s και κάποιες <embedded SQL declare statement>s όπως αυτές ορίζονται από το στάνταρτ. Αυτό επιτρέπει στις εφαρμογές βάσεων δεδομένων να έχουν μια υβριδική μορφή όπου οι SQL-δηλώσεις είναι ενσωματωμένες κατευθείαν σε μια μονάδα προγράμματος.

Μια υλοποίηση του στάνταρτ είναι πιθανό να δεσμεύσει ένα τμήμα από το σύνολο των ονομάτων μέσα στο <embedded SQL host program> για τα ονόματα των διαδικασιών και των υπορουτινών οι οποίες παράγονται για να αντικαταστήσουν τις SQL-δηλώσεις και τις μεταβλητές του προγράμματος.

Δυναμικές SQL δηλώσεις

Σε πολλές περιπτώσεις, μια SQL-δήλωση για να εκτελεστεί μπορεί να είναι μέσα σε ένα <SQL-client module> ή μέσα σε μια μονάδα προγράμματος σε embedded μορφή. Σε άλλες όμως περιπτώσεις, οι SQL-δηλώσεις δεν είναι γνωστές όταν γράφεται το πρόγραμμα, αλλά δημιουργούνται κατά την εκτέλεση του προγράμματος. Μια δήλωση <execute immediate> μπορεί να χρησιμοποιηθεί για να την προετοιμασία και εκτέλεση της δήλωσης για μία φορά. Μια δήλωση <prepare> χρησιμοποιείται για να προετοιμάσει μια SQL-δήλωση για εκτέλεση. Μια δήλωση <deallocate prepared> χρησιμοποιείται για να απελευθερώσει τις δηλώσεις που έχουν ήδη προετοιμαστεί με δήλωση <prepare>. Η περιγραφή των δυναμικών παραμέτρων εισόδου μιας προετοιμασμένης δήλωσης μπορεί να γίνει με τη χρήση της δήλωσης <describe input>. Η περιγραφή των στηλών εξόδου για μια δήλωση <dynamic select> μπορεί να γίνει με τη χρήση της <describe output>. Για τις άλλες δηλώσεις εκτός της <dynamic select>, χρησιμοποιείται η δήλωση <execute> για να συσχετίσει τις παραμέτρους με την προετοιμασμένη δήλωση και να την εκτελέσει σαν να ήταν από πριν γραμμένη στο πρόγραμμα. Για μια δήλωση <dynamic select>, χρησιμοποιείται η δήλωση <dynamic declare cursor> για να συσχετίσει αυτή με έναν προετοιμασμένο κέρσορα. Με τη χρήση της <dynamic open> δήλωσης γίνεται άνοιγμα του κέρσορα και συσχέτιση των δυναμικών παραμέτρων με αυτόν. Με τη δήλωση <dynamic fetch> γίνεται τοποθέτηση του ανοικτού κέρσορα στην συγκεκριμένη γραμμή και ανάκτηση των τιμών των στηλών αυτής της γραμμής. Μια δήλωση <dynamic close> κλείνει τον κέρσορα. Με τη δήλωση <dynamic delete: positioned> μπορούμε να διαγράψουμε γραμμές μέσω του δυναμικού κέρσορα. Με τη δήλωση <dynamic update: positioned> μπορούμε να διαγράψουμε γραμμές μέσω του δυναμικού κέρσορα.

Η διασύνδεση για τις δυναμικές παραμέτρους εισόδου και εξόδου για μια προετοιμασμένη δήλωση μπορεί να είναι μια λίστα από δυναμικές παραμέτρους ή ενσωματωμένες μεταβλητές ή μια SQL περιοχή περιγραφέων (descriptor area). Μια SQL περιοχή περιγραφέων αποτελείται από μηδέν ή περισσότερες περιοχές αντικειμένων και ένα COUNT για τον αριθμό αυτών των περιοχών αντικειμένων. Μια SQL περιοχή περιγραφέων μπορεί να οριστεί σαν *τοπική (local)* ή *καθολική (global)*. Παρόμοια, ένα εκτεταμένο όνομα δήλωσης ή κέρσορα μπορεί να είναι τοπικό ή καθολικό. Η εμβέλεια (scope) ενός καθολικού ονόματος είναι η SQL-session. Η εμβέλεια ενός τοπικού ονόματος είναι το <SQL-client module> στο οποίο εμφανίζεται. Η αναφορά σε μια οντότητα που καθορίζει καθολική εμβέλεια είναι έγκυρη μόνο αν η οντότητα έχει οριστεί σαν καθολική και αν η αναφορά είναι από την ίδια SQL-session που ορίστηκε. Η αναφορά σε τοπική οντότητα είναι έγκυρη μόνο αν η οντότητα έχει οριστεί σαν τοπική και μόνο μέσα από το <SQL-client module> στο οποίο ορίστηκε.

Πολλές SQL-δηλώσεις μπορούν να γραφούν έτσι ώστε να χρησιμοποιούν παραμέτρους, Στις δηλώσεις που εκτελούνται δυναμικά, οι παράμετροι αποκαλούνται "δυναμικές παράμετροι" και απεικονίζονται στην SQL με το ερωτηματικό (?). Σε πολλές περιπτώσεις, μια εφαρμογή που παράγει μια δυναμική δήλωση για εκτέλεση γνωρίζει σε λεπτομέρεια τα απαιτούμενα χαρακτηριστικά (τύπο δεδομένων, μήκος, ακρίβεια) της κάθε παραμέτρου που χρησιμοποιείται στη δήλωση. Παρόμοια, η εφαρμογή μπορεί να γνωρίζει σε λεπτομέρεια τα χαρακτηριστικά των τιμών που θα επιστραφούν από την εκτέλεση της δήλωσης. Όμως σε άλλες περιπτώσεις, η εφαρμογή μπορεί να μην γνωρίζει αυτή την πληροφορία στην απαραίτητη λεπτομέρεια. Είναι πιθανό σε κάποιες περιπτώσεις να βρεθεί αυτή η πληροφορία από το "σχήμα πληροφορίας" (information schema), αλλά σε άλλες δεν είναι γνωστή αυτή η πληροφορία παρά μόνο την στιγμή της προετοιμασίας για εκτέλεση. Για να παρέχεται η απαραίτητη πληροφορία στις

εφαρμογές, η SQL επιτρέπει την αίτηση στο σύστημα βάσης δεδομένων για την περιγραφή της προετοιμασμένης δήλωσης. Η περιγραφή της δήλωσης προσδιορίζει τον αριθμό των δυναμικών παραμέτρων (εισόδου) και τον τύπου δεδομένων τους ή προσδιορίζει τον αριθμό των τιμών που θα επιστραφούν (εξόδου) και τον τύπο δεδομένων τους. Η περιγραφή της δήλωσης τοποθετείται σε μια περιοχή περιγραφέων SQL για τους οποίους αναφερθήκαμε πριν.

Συντακτικά λάθη και παραβιάσεις κανόνων πρόσβασης προκαλούνται από την προετοιμασία ή την άμεση εκτέλεση των δηλώσεων. Τέτοιου είδους παραβιάσεις σημειώνονται με την εμφάνιση μιας κατάστασης εξαίρεσης (exception condition).

Απευθείας κλήση (direct invocation) SQL

Στην απευθείας κλήση SQL, η μέθοδος `<direct SQL>`, η μέθοδος για τον έλεγχο λαθών και εμφάνισης καταστάσεων εξαίρεσης, η μέθοδος πρόσβασης σε διαγνωστικές πληροφορίες και η μέθοδος για την επιστροφή των αποτελεσμάτων εξαρτάται από την συγκεκριμένη υλοποίηση.

3.7 SQL/TRANSACTION

3.7.1 Ορισμοί

- *Δοσοληψία (Transaction)*: Είναι μια ολοκληρωμένη μονάδα εργασίας, που πιθανά συνδυάζει πολλές υπολογιστικές υποεργασίες, οι οποίες περιλαμβάνουν ανάκτηση δεδομένων και επικοινωνία με τον χρήστη.
- *Κατανεμημένη επεξεργασία δοσοληψιών (Distributed transaction processing), ΚΕΔ*: Είναι η επεξεργασία η οποία γίνεται μεταξύ πολλαπλών διαχειριστών πόρων (resource managers).
- *Πρόγραμμα Εφαρμογής (Application Program, ΑΡ), ΠΕ*: Είναι ένα πρόγραμμα το οποίο ορίζει δοσοληψίες και πραγματοποιεί πρόσβαση σε πηγές δεδομένων.
- *Διαχειριστής Πόρων (Resource Manager, RM), ΔΠ*: Είναι ένα πρόγραμμα το οποίο διαχειρίζεται ένα συγκεκριμένο μέρος από τις διαθέσιμες πηγές ενός υπολογιστή, όπως δεδομένα SQL.
- *Καθολική Δοσοληψία (global transaction)*: Είναι μια δοσοληψία οι οποία μπορεί να συσχετίζει περισσότερους από έναν διαχειριστές πόρων.
- *Διαχειριστής Δοσοληψιών (transaction manager), ΔΔ*: Είναι ένα πρόγραμμα το οποίο διαχειρίζεται καθολικές δοσοληψίες και συντονίζει τις αποφάσεις για την οριστική καταχώρηση τους (commit) ή την ανάκληση τους (roll back).

3.7.2 Βασικές ιδέες

Ο όρος “Κατανεμημένη Επεξεργασία Δοσοληψιών” ή “ΚΕΔ” έχει χρησιμοποιηθεί για να ορίσει τη δυνατότητα πολλών διαχειριστών, να συμμετέχουν σε ένα μέρος μιας εργασίας, η οποία αποκαλείται *καθολική δοσοληψία*, και η οποία συντονίζεται από μια κεντρική αρχή, η οποία αποκαλείται *Διαχειριστής Δοσοληψιών (Transaction Manager)*. Αν και χρησιμοποιείται η λέξη “κατανεμημένη”, δεν σημαίνει ότι η δοσοληψία κατανέμεται σε πολλούς υπολογιστές, αλλά σημαίνει ότι η δοσοληψία κατανέμεται σε πολλούς διαχειριστές πηγών (resource managers). Για να αποφύγουμε λάθος συμπεράσματα, χρησιμοποιούμε συχνότερα τον όρο “καθολική δοσοληψία (global transaction)”.

Δοσοληψία (γενικά)

Οι δοσοληψίες πρέπει να έχουν την δυνατότητα της ανάκλησης (roll back). Ένας χρήστης μπορεί να κάνει ανάκληση μιας δοσοληψίας μετά από ένα γεγονός, όπως για παράδειγμα την απόφαση ενός πελάτη. Οι δοσοληψίες μπορούν επίσης να ανακληθούν σε περίπτωση αποτυχίας του συστήματος για ανάκτηση, επικοινωνία ή αποθήκευση δεδομένων. Κάθε κομμάτι προγράμματος που ασχολείται με κατανεμημένη επεξεργασία δοσοληψιών, πρέπει να μπορεί να ανακαλέσει όλη την εργασία που έχει κάνει σε περίπτωση που γίνει ανάκληση μιας δοσοληψίας.

Όταν το σύστημα αποφασίσει ότι η δοσοληψία μπορεί να ολοκληρωθεί χωρίς καμιά περίπτωση αποτυχίας, τότε καταχωρεί την δοσοληψία (commit). Αυτό σημαίνει ότι οι αλλαγές στις πηγές δεδομένων έχουν μόνιμα ενημερωθεί. Η καταχώρηση ή ανάκληση μιας δοσοληψίας οδηγεί το σύστημα πάντα σε μια σταθερή κατάσταση.

Κατανεμημένη επεξεργασία δοσοληψιών, ΚΕΔ

Κάθε σύστημα κατανεμημένης επεξεργασίας δοσοληψιών πρέπει να υποστηρίζει δοσοληψίες μεταξύ διαφορετικών διαχειριστών πηγών. Αυτό συνεπάγεται μερικά θέματα:

- Το σύστημα πρέπει να έχει έναν τρόπο να αναφέρεται σε μια δοσοληψία η οποία περικλείει όλη την εργασία που έγινε οπουδήποτε μέσα στο σύστημα.
- Η απόφαση της καταχώρησης ή της ανάκλησης μιας δοσοληψίας πρέπει να λάβει υπόψη την κατάσταση της εργασίας που έγινε οπουδήποτε από την δοσοληψία. Η απόφαση πρέπει να έχει ομοιόμορφη επίδραση σε όλο το σύστημα κατανεμημένης επεξεργασίας δοσοληψιών.

Μια SQL-υλοποίηση πρέπει να περιλαμβάνει αυτά τα δύο θέματα για να είναι ικανή να λάβει μέρος σε ένα περιβάλλον κατανεμημένης επεξεργασίας δοσοληψιών.

Διαχειριστής Πόρων (Resource Manager), ΔΠ

Ο διαχειριστής πόρων διαχειρίζεται ένα συγκεκριμένο μέρος από τις μοιραζόμενες πηγές του συστήματος. Πολλά άλλα προγράμματα μπορούν να ζητούν πρόσβαση στις ίδιες πηγές, χρησιμοποιώντας υπηρεσίες που παρέχει ο διαχειριστής πόρων. Μερικά παραδείγματα διαχειριστών πόρων ακολουθούν:

- Ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) είναι ένας διαχειριστής πόρων. Τα τυπικά ΣΔΒΔ είναι ικανά να ορίσουν δοσοληψίες και να καταχωρήσουν την εργασία τους ατομικά. Μια SQL-υλοποίηση είναι ένα ΣΔΒΔ.
- Μια μέθοδος προσπέλασης αρχείων όπως η Σειριακή με Δείκτη (ISAM) μπορεί να είναι η βάση για έναν διαχειριστή πόρων. Τυπικά, ένας ISAM διαχειριστής πόρων πρέπει να επεκταθεί για να υποστηρίζει δοσοληψίες, όπως τις αναφέραμε προηγουμένως.
- Ένας εξυπηρετητής εκτυπώσεων (printer server) μπορεί να υλοποιηθεί έτσι ώστε να αποτελεί έναν διαχειριστή πόρων.

Καθολική δοσοληψία (global transaction)

Κάθε διαχειριστής πόρων, σε ένα περιβάλλον κατανεμημένης επεξεργασίας δοσοληψιών, πρέπει να υποστηρίζει δοσοληψίες όπως αυτές έχουν ήδη περιγραφεί σε προηγούμενη υποενότητα. Σε ένα περιβάλλον κατανεμημένης επεξεργασίας δοσοληψιών, πολλοί διαχειριστές πόρων μπορούν να εργάζονται για να υποστηρίξουν την ίδια μονάδα εργασίας. Αυτή η μονάδα εργασίας είναι μια καθολική δοσοληψία. Για παράδειγμα, ένα πρόγραμμα εφαρμογής μπορεί να απαιτήσει ενημερώσεις σε δεδομένα που βρίσκονται σε διαφορετικές βάσεις δεδομένων. Κάθε

εργασία που γίνεται οπουδήποτε μέσα στο σύστημα πρέπει να καταχωρηθεί ατομικά. Κάθε διαχειριστής πόρων πρέπει να επιτρέπει στον διαχειριστή δοσοληψιών να συντονίζει τις δικές του μονάδες εργασίας οι οποίες αποτελούν μέρος την καθολικής δοσοληψίας.

Η καταχώρηση μιας εσωτερικής εργασίας σε έναν διαχειριστή πόρων δεν εξαρτάται μόνο από την επιτυχία των δικών του λειτουργιών, αλλά εξαρτάται επίσης και από τις λειτουργίες που συμβαίνουν σε έναν άλλο διαχειριστή πόρων. Αν οποιαδήποτε λειτουργία αποτύχει κάπου, κάθε διαχειριστής πόρων που συμμετέχει πρέπει να ανακαλέσει όλες τις λειτουργίες που πραγματοποίησε στη διάρκεια της καθολικής δοσοληψίας. Κάθε διαχειριστής πόρων δεν γνωρίζει την εργασία που πραγματοποιεί ένας άλλος διαχειριστής πόρων. Ο Διαχειριστής Δοσοληψιών ενημερώνει κάθε διαχειριστή πόρων για την ύπαρξη και την ολοκλήρωση μιας καθολικής δοσοληψίας. Κάθε διαχειριστής πόρων είναι υπεύθυνος για την διάθεση των μονάδων του που θα εργαστούν σε μια καθολική δοσοληψία.

Κλάδος δοσοληψίας (transaction branch)

Μια καθολική δοσοληψία έχει έναν ή περισσότερους κλάδους. Κάθε κλάδος είναι μέρος της εργασίας μιας καθολικής δοσοληψίας, με την οποία ασχολούνται ο διαχειριστής δοσοληψιών και ο διαχειριστής πόρων ξεχωριστά αλλά συντονισμένα όπως αυτό καθορίζεται με το πρωτόκολλο καταχώρησης δοσοληψίας (transaction commitment protocol). Κάθε μια από τις μονάδες εργασίας ενός διαχειριστή πόρων είναι μέρος από έναν ακριβώς κλάδο. Μετά την εκκίνηση του πρωτοκόλλου καταχώρησης στον διαχειριστή δοσοληψιών, ο διαχειριστής πόρων δεν λαμβάνει επιπλέον εργασία για τον συγκεκριμένο κλάδο της δοσοληψίας. Ένας διαχειριστής πόρων μπορεί να αναλάβει πρόσθετη εργασία στο ενδιάμεσο αυτής της δοσοληψίας από άλλους κλάδους. Οι διαφορετικοί κλάδοι που σχετίζονται με αυτή τη διαδικασία πρέπει να είναι εντελώς ανεξάρτητοι. Κάθε *αναγνωριστικό (identifier or XID)* ενός κλάδου δοσοληψίας που ο διαχειριστής δοσοληψιών δίνει σε έναν διαχειριστή πόρων, αναγνωρίζει την δοσοληψία και τον συγκεκριμένο κλάδο. Ο διαχειριστής πόρων μπορεί να χρησιμοποιήσει αυτή τη πληροφορία για να βελτιστοποιήσει τη χρήση των μοιραζόμενων πόρων και των κλειδωμάτων (locks).

Νήμα ελέγχου (thread of control)

Ένα νήμα ελέγχου είναι μια οντότητα, της οποίας όλο το περιβάλλον ελέγχεται από τον επεξεργαστή. Ένα νήμα ελέγχου είναι μια διαδικασία του λειτουργικού συστήματος. Το περιεχόμενο ενός νήματος μπορεί να περιλαμβάνει τα κλειδώματα της διαδικασίας στις διαθέσιμες πηγές και στα ανοικτά αρχεία. Για λόγους μεταφερσιμότητας, η έννοια "νήμα ελέγχου" πρέπει να συσχετίζεται με τα προγράμματα εφαρμογής, τους διαχειριστές δοσοληψιών και τους διαχειριστές πόρων.

Η ιδέα του νήματος είναι βασική στον συντονισμό των ΔΠ από τον διαχειριστή δοσοληψιών. Τα προγράμματα εφαρμογής καλούν τους διαχειριστές πόρων για να εκτελέσουν μια εργασία, ενώ οι διαχειριστές δοσοληψιών καλούν τους διαχειριστές πόρων για να αναθέσουν κλάδους δοσοληψιών. Ο τρόπος με τον οποίο ο διαχειριστής πόρων γνωρίζει ότι μια συγκεκριμένη εργασία ανήκει σε ένα συγκεκριμένο κλάδο δοσοληψίας, είναι η κλήση του προγράμματος εφαρμογής και του διαχειριστή δοσοληψιών μέσα από το ίδιο νήμα ελέγχου. Για παράδειγμα, ένα νήμα προγράμματος εφαρμογής καλεί τον διαχειριστή δοσοληψιών για να δηλώσει την αρχή μιας καθολικής δοσοληψίας. Ο διαχειριστής δοσοληψιών καταγράφει το γεγονός και ενημερώνει τους διαχειριστές πόρων. Αφού το πρόγραμμα εφαρμογής ανακτήσει τον έλεγχο, χρησιμοποιεί το φυσικό interface ενός ή περισσοτέρων ΔΠ για να εργαστεί. Ο

διαχειριστής πόρων λαμβάνει τις κλήσεις από το πρόγραμμα εφαρμογής και τον διαχειριστή δοσοληψιών μέσω του ίδιου νήματος ελέγχου. Όμως υπάρχουν συγκεκριμένες διαδικασίες οι οποίες πρέπει να κληθούν από ένα συγκεκριμένο νήμα. Σε μια καθολική δοσοληψία συμμετέχουν πολλά νήματα ελέγχου των εφαρμογών. Όλη η εργασία που γίνεται σε αυτά τα νήματα, ολοκληρώνεται ατομικά. Μέσα σε μια δοσοληψία, η σχέση μεταξύ δύο νημάτων που συμμετέχουν σε αυτή μπορεί να είναι είτε *στενά-συνδεδεμένη (tightly-coupled)*, είτε *χαλαρά-συνδεδεμένη (loosely-coupled)*.

Στενά-συνδεδεμένη σχέση νημάτων (tightly-coupled)

Μια στενά-συνδεδεμένη σχέση είναι αυτή όπου δύο νήματα είναι σχεδιασμένα για να μοιράζονται πόρους. Επιπλέον, λαμβάνοντας υπόψη την πολιτική απομόνωσης των διαχειριστών πόρων, κάθε τέτοιο ζευγάρι νημάτων μπορεί να αντιμετωπιστεί σαν μια οντότητα. Έτσι για ένα στενά-συνδεμένο ζευγάρι νημάτων, πρέπει να εξασφαλιστεί από τον διαχειριστή πόρων η αποφυγή αδιεξόδου (deadlock) μέσα σε έναν κλάδο δοσοληψίας. Σε μια απλή καθολική δοσοληψία, ένα σύνολο στενά-συνδεδεμένων νημάτων μπορεί να αποτελείται από περισσότερα του ενός ζευγαριού νήματα.

Χαλαρά-συνδεδεμένη σχέση νημάτων (loosely-coupled)

Λαμβάνοντας υπόψη την πολιτική απομόνωσης των διαχειριστών πόρων, κάθε ζευγάρι χαλαρά συνδεδεμένων νημάτων μπορεί να αντιμετωπιστεί σαν να συμμετέχει σε διαφορετικές δοσοληψίες. Πολλά σύνολα στενά-συνδεδεμένων νημάτων μπορούν να συμμετέχουν σε μια δοσοληψία και κάθε ένα από αυτά τα σύνολα να είναι χαλαρά-συνδεδεμένο σε σχέση με τα άλλα.

3.7.3 Δοσοληψία, ολοκλήρωση/ανάνηψη (completion/recovery)

Οι διαχειριστές δοσοληψιών και οι διαχειριστές πόρων χρησιμοποιούν τον αλγόριθμο ολοκλήρωσης δύο φάσεων (2-phase commit, 2PC), όπως αυτός ορίζεται στο OSI DTP specification.

- Στην πρώτη φάση του 2PC, ο διαχειριστής δοσοληψιών ζητάει από όλους τους διαχειριστές πόρων να προετοιμαστούν για καταχώρηση (commit). Αυτή η απαίτηση πρέπει να εξασφαλίσει την δυνατότητα καταχώρησης του μέρους της δοσοληψίας που αντιστοιχεί στον κάθε διαχειριστή πόρων. Για να απαντήσει ο διαχειριστής πόρων ίσως χρειαστεί να ρωτήσει εσωτερικά άλλες δικές του οντότητες. Εάν ο διαχειριστής πόρων μπορεί να ολοκληρώσει την εργασία του, τότε καταγράφει την πληροφορία που χρειάζεται για να το κάνει αυτό και απαντά. Κάθε αρνητική απάντηση αναφέρεται σαν αποτυχία για οποιονδήποτε λόγο. Αφού δώσει μια αρνητική απάντηση και ανακαλέσει την εργασία του, ο διαχειριστής πόρων μπορεί να διαγράψει όλη τη γνώση σχετικά με το μέρος της δοσοληψίας που του αντιστοιχεί.
- Στην δεύτερη φάση, ο διαχειριστής δοσοληψιών στέλνει σε όλους τους διαχειριστές πόρων μια αίτηση για ολοκλήρωση ή ανάκληση της εργασίας τους, ανάλογα με την περίπτωση. Πριν στείλει τις αιτήσεις για ολοκλήρωση της καταχώρησης (commit), ο διαχειριστής δοσοληψιών καταγράφει το γεγονός αυτής της απόφασης καθώς και την λίστα με τους διαχειριστές πόρων που εμπλέκονται. Όλοι οι διαχειριστές πόρων ολοκληρώνουν ή ανακαλούν τις αλλαγές τους και επιστρέφουν το αποτέλεσμα της κατάστασης στον διαχειριστή δοσοληψιών. Ο διαχειριστής δοσοληψιών τότε μπορεί να διαγράψει όλα όσα γνωρίζει για την καθολική δοσοληψία.

Ανακαλώντας μια καθολική δοσοληψία (Roll-back)

Ο διαχειριστής δοσοληψιών ανακαλεί μια καθολική δοσοληψία αν κάποιος από τους διαχειριστές πόρων απαντήσει αρνητικά στην αίτηση της πρώτης φάσης του 2PC, ή αν το πρόγραμμα εφαρμογής οδηγήσει τον διαχειριστή δοσοληψιών να ανακαλέσει την δοσοληψία. Ο διαχειριστής δοσοληψιών επιδρά στην δεύτερη φάση ζητώντας από τους διαχειριστές πόρων να ανακαλέσουν όλα τα μέρη της εργασίας τους. Δεν πρέπει να επιτρέψουν σε καμία αλλαγή να γίνει μόνιμη. Ο διαχειριστής δοσοληψιών δεν στέλνει καμιά αίτηση σε κανέναν από τους διαχειριστές πόρων που απάντησαν αρνητικά στην αίτηση της πρώτης φάσης. Ο διαχειριστής δοσοληψιών δεν χρειάζεται να καταγράψει την απόφαση για ανάκληση όπως επίσης δεν χρειάζεται να την καταγράψουν και οι υπόλοιποι διαχειριστές πόρων.

Βελτιστοποιήσεις πρωτοκόλλου (protocol optimizations)

Υπάρχουν δύο βελτιστοποιήσεις που επιτρέπουν σε διάφορες υλοποιήσεις να χρησιμοποιούνται για απομακρυσμένη επικοινωνία:

- *Βελτιστοποίηση μόνο-ανάγνωσης (Read-only).* Κάθε διαχειριστές πόρων μπορεί να απαντήσει στον διαχειριστή δοσοληψιών ότι δεν του ζητήθηκε να ενημερώσει κάποιες από τις πηγές του. Η δεύτερη φάση του 2PC δεν χρειάζεται να γίνει μεταξύ του διαχειριστή δοσοληψιών και του διαχειριστή πόρων. Όμως αν ο διαχειριστής πόρων επιστρέψει read-only βελτιστοποίηση πριν ολοκληρωθεί η προετοιμασία για την δοσοληψία, τότε δεν μπορεί να εξασφαλιστεί η σειριοποιησιμότητα (serializability). Αυτό συμβαίνει γιατί ο διαχειριστής πόρων μπορεί να απελευθερώσει τα περιεχόμενα της δοσοληψίας, όπως κλειδώματα ανάγνωσης, πριν τελειώσει όλη η διαδικασία της δοσοληψίας.
- *Βελτιστοποίηση μιας φάσης (One-phase commit).* Ένας διαχειριστής δοσοληψιών μπορεί να χρησιμοποιήσει ολοκλήρωση μιας φάσης, εάν γνωρίζει ότι υπάρχει μόνο ένας διαχειριστής πόρων σε όλο το περιβάλλον κατανεμημένης επεξεργασίας δοσοληψιών που πραγματοποιεί αλλαγές στις διαθέσιμες πηγές. Σε αυτή τη μέθοδο βελτιστοποίησης, ο διαχειριστής δοσοληψιών πραγματοποιεί την δεύτερη φάση χωρίς να έχει περάσει από την πρώτη φάση της προετοιμασίας. Αφού λοιπόν ο διαχειριστής πόρων αποφασίζει για το αποτέλεσμα της δοσοληψίας του και ξεχνά την πληροφορία για αυτή πριν επιστρέψει στον διαχειριστή δοσοληψιών, δεν υπάρχει λόγος για τον διαχειριστή δοσοληψιών να καταγράψει την καθολική δοσοληψία, και σε μερικές περιπτώσεις αποτυχίας να μην γνωρίζει το αποτέλεσμα της δοσοληψίας στον διαχειριστή πόρων.

Ευριστική (Heuristic) ολοκλήρωση μέρους της δοσοληψίας

Μερικοί διαχειριστές πόρων χρησιμοποιούν ευριστικούς τρόπους για λήψη αποφάσεων. Ένας διαχειριστής πόρων που έχει προετοιμαστεί για καταχώρηση ενός μέρους μιας δοσοληψίας μπορεί να αποφασίσει για την καταχώρηση ή την ανάκληση της δοσοληψίας ανεξάρτητα από τον διαχειριστή δοσοληψιών. Θα μπορούσε να ξεκλειδώσει τις μοιραζόμενες πηγές και να αφήσει το σύστημα σε μια ασταθή κατάσταση. Όταν ο διαχειριστής δοσοληψιών απαιτήσει από τον διαχειριστή πόρων να ολοκληρώσει την δοσοληψία, αυτός μπορεί να απαντήσει ότι το έχει ήδη κάνει. Ο διαχειριστής πόρων αναφέρει πότε ολοκλήρωσε την δοσοληψία, πότε την ανακάλεσε και πότε έκανε κάτι ανάμεσα στα δύο (ολοκλήρωσε ένα μέρος και ανακάλεσε το υπόλοιπο).

Αποτυχίες και ανάνηψη (failures and recovery)

Ένα χρήσιμο σύστημα κατανεμημένης επεξεργασίας δοσοληψιών πρέπει να έχει την δυνατότητα της ανάνηψης μετά από διάφορες αποτυχίες μιας δοσοληψίας. Αυτές οι αποτυχίες μπορούν να οφείλονται στο αποθηκευτικό μέσο, στο κανάλι επικοινωνίας σε έναν κόμβο του συστήματος και στο πρόγραμμα. Οι αποτυχίες που μπορεί να διορθώσει εσωτερικά ένας κόμβος δεν επηρεάζουν την συνολική δοσοληψία. Αποτυχίες που δεν διακόπτουν το πρωτόκολλο οριστικής καταχώρησης (commit) επιτρέπουν στο σύστημα κατανεμημένης επεξεργασίας δοσοληψιών να αντιδράσει ανακτώντας τις κατάλληλες καθολικές δοσοληψίες. Πιο σοβαρές αποτυχίες μπορούν να διαλύσουν το πρωτόκολλο οριστικής καταχώρησης (commit). Ο διαχειριστής δοσοληψιών ανιχνεύει την αποτυχία όταν δεν πάρει απάντηση σε μια συγκεκριμένη αίτηση.

3.7.4 XA interface

Το standard περιέχει interfaces επικοινωνίας μεταξύ των ΠΕ, ΔΠ, και ΔΔ. Το XA interface είναι αυτό που χρησιμοποιείται για την επικοινωνία του διαχειριστή δοσοληψιών με τον κάθε διαχειριστή πόρων. Γενικά, το XA interface χρησιμοποιείται από τα προγράμματα εφαρμογών που καλούν είτε τον διαχειριστή πόρων είτε τον διαχειριστή δοσοληψιών.

Υπηρεσίες στο XA interface

Το XA interface περιέχει δύο κλάσεις από ρουτίνες. Η μια κλάση η οποία αποκαλείται `ax_`, επιτρέπει σε έναν διαχειριστή πόρων να καλέσει έναν διαχειριστή δοσοληψιών. Όλοι οι διαχειριστές δοσοληψιών πρέπει να παρέχουν αυτές τις ρουτίνες. Αυτές οι ρουτίνες επιτρέπουν στον διαχειριστή πόρων να ελέγχει δυναμικά την συμμετοχή του σε ένα κλαδί μιας δοσοληψίας. Η άλλη κλάση καλείται `xa_`. Οι ρουτίνες της παρέχονται από τους διαχειριστές πόρων και καλούνται από τους διαχειριστές δοσοληψιών. Όταν ένα πρόγραμμα εφαρμογής καλεί έναν διαχειριστή δοσοληψιών να ξεκινήσει μια δοσοληψία, ο διαχειριστής δοσοληψιών μπορεί να χρησιμοποιήσει το `xa_` interface για να ενημερώσει τους διαχειριστές πόρων για την δοσοληψία. Αφού το πρόγραμμα εφαρμογής χρησιμοποιεί το interface του διαχειριστή πόρων για να υποστηρίξει την δοσοληψία, ο διαχειριστής δοσοληψιών καλεί άλλες `xa_` ρουτίνες για την ολοκλήρωση ή ανάκληση των κλαδιών της δοσοληψίας.

Ένας διαχειριστής δοσοληψιών πρέπει να καλεί τις `xa_` ρουτίνες με μια καθορισμένη σειρά. Όταν ένας διαχειριστής δοσοληψιών καλεί περισσότερους από έναν διαχειριστή πόρων με την ίδια `xa_` ρουτίνα, μπορεί να το κάνει με τυχαία σειρά. Στον παρακάτω πίνακα "Ρουτίνες στο XA interface", φαίνονται τα κανονικά ονόματα κάθε ρουτίνας. Αυτά είναι τα ονόματα που χρησιμοποιεί ο διαχειριστής δοσοληψιών για να καλέσει τις κατάλληλες ρουτίνες σε έναν διαχειριστή πόρων. Όμως τα πραγματικά ονόματα που ανάγει ο διαχειριστής πόρων σε αυτές τις ρουτίνες δεν είναι απαραίτητα τα ονόματα που χρησιμοποιεί ο διαχειριστής δοσοληψιών. Γι' αυτό το λόγο θα πρέπει να υπάρχει ένας εσωτερικός πίνακας που θα δείχνει την συσχέτιση του μεταξύ των ρουτινών που απαιτούνται από τον διαχειριστή δοσοληψιών και αυτών που παρέχονται από τον διαχειριστή πόρων.

`ax_reg`	Εγγραφή ενός Διαχειριστή Πόρων (ΔΠ) σε έναν Διαχειριστή Δοσοληψιών (ΔΔ).
`ax_unreg`	Διαγραφή ενός ΔΠ από έναν ΔΔ.
`xa_close`	Τερματίζει την χρήση ενός Προγράμματος εφαρμογής (ΠΕ) από

	έναν ΔΠ.
xa_commit	Ενημερώνει τον ΔΠ να καταχωρίσει μόνιμα τις αλλαγές της δοσοληψίας.
xa_complete	Ελέγχει την ολοκλήρωση μιας ασύγχρονης xa_ λειτουργίας .
xa_end	Απελευθερώνει ένα νήμα από τον κλάδο δοσοληψίας.
xa_forget	Επιτρέπει στον ΔΠ να διαγράψει όλη την πληροφορία σχετικά με έναν ευριστικά-ολοκληρωμένο κλάδο δοσοληψίας.
xa_open	Αρχικοποιεί έναν ΔΠ για χρήση από ένα ΠΕ.
xa_prepare	Ζητά από τον ΔΠ να ετοιμαστεί για commit ενός κλάδου δοσοληψίας.
xa_recover	Λαμβάνει μια λίστα από XIDs που έχει προετοιμάσει ή ολοκληρώσει ο ΔΠ.
xa_rollback	Ζητά από τον ΔΠ να ανακαλέσει το δικό του μέρος εργασίας.
xa_start	Ξεκινά ή επαναφέρει έναν κλάδο της δοσοληψίας-συσχετίζει ένα XID με την μελλοντική εργασία που το νήμα απαιτεί από τον ΔΠ.

Πίνακας : "Ρουτίνες στο XA_ interface"

Άνοιγμα/κλείσιμο ΔΠ

Σε κάθε νήμα ελέγχου, ο διαχειριστής δοσοληψιών πρέπει να καλέσει την `xa_open()` για κάθε διαχειριστή πόρων που είναι άμεσα προσπελάσιμος από αυτό το νήμα, πριν καλέσει άλλες xa_ ρουτίνες. Ο διαχειριστής δοσοληψιών πρέπει να καλέσει την `xa_close()` για να απελευθερώσει το πρόγραμμα εφαρμογής από τον διαχειριστή πόρων. Αν ένας διαχειριστής πόρων χρειάζεται ενέργειες εκκίνησης (άνοιγμα αρχείων, ορισμό μονοπατιών προς τον εξυπηρετητή, συγχρονισμός κόμβου με το δίκτυο) μπορεί να το κάνει καλώντας την `xa_open()`.

Σχέση νημάτων με τους κλάδους δοσοληψιών

Μερικά νήματα μπορούν να συμμετέχουν σε έναν μοναδικό κλάδο δοσοληψίας και μερικά περισσότερες από μια φορές. Οι ρουτίνες `xa_start()`, `xa_end()` περνούν ένα XID σε έναν διαχειριστή πόρων για να συσχετίσουν ή να χωρίσουν το νήμα με/από τον κλάδο δοσοληψίας. Η σχέση ενός νήματος με έναν κλάδο δοσοληψίας μπορεί να είναι *είτε ενεργή (active)* είτε *προσωρινά απενεργοποιημένη (suspended)*:

- Ένα νήμα είναι ενεργά συσχετισμένο με έναν κλάδο δοσοληψίας εάν έχει καλέσει την `xa_start()` και δεν έχει κάνει αντίστοιχη κλήση της `xa_end()`. Σε κάθε νήμα επιτρέπεται μόνο μια ταυτόχρονη ενεργή σχέση με κάθε διαχειριστή πόρων.
- Συγκεκριμένες κλήσεις της `xa_end()` απενεργοποιούν προσωρινά τη σχέση του νήματος με τον κλάδο δοσοληψίας. Η κλήση μπορεί να υποδείξει ότι η σχέση μπορεί να μεταφερθεί και οποιοδήποτε άλλο νήμα να επανακτήσει τη σχέση. Σε αυτή την περίπτωση το νήμα δεν είναι πλέον συσχετισμένο με τον κλάδο δοσοληψίας.

Περιβάλλον δοσοληψίας

Το περιβάλλον δοσοληψίας είναι μια ειδική πληροφορία του διαχειριστή πόρων η οποία είναι ορατή στο πρόγραμμα εφαρμογής. Ο διαχειριστής πόρων πρέπει να διατηρεί το περιβάλλον της δοσοληψίας μετά από μια `xa_end()` έτσι ώστε να μπορεί να το επαναφέρει σε

περιπτώσεις σύνδεσης (join) και συνέχισης μετά από διακοπή (resume). Στην περίπτωση σύνδεσης, ο διαχειριστής πόρων πρέπει να διαθέσει αρκετό από το περιβάλλον της δοσοληψίας έτσι ώστε τα στενά-συνδεμένα νήματα να μην είναι ευάλωτα σε καταστάσεις αδιεξόδου μέσα σε έναν κλάδο δοσοληψίας. Στην περίπτωση συνέχισης μετά από διακοπή, ο διαχειριστής πόρων πρέπει να διαθέσει τουλάχιστον το περιβάλλον που υπήρχε την στιγμή της διακοπής.

Εγγραφή των Διαχειριστών Πόρων

Κανονικά, ο διαχειριστής δοσοληψιών εμπλέκει όλους του σχετιζόμενους διαχειριστές πόρων σε έναν κλάδο μιας δοσοληψίας. Ο διαχειριστής δοσοληψιών καλεί όλους αυτούς τους διαχειριστές πόρων με `xa_start()`, `xa_end()` και `xa_prepare()`, αν και ένας ανενεργός διαχειριστής πόρων του κλάδου δεν χρειάζεται να συμμετέχει παραπέρα. Μια τεχνική για να μειωθεί ο φόρτος από αραιά χρησιμοποιούμενους διαχειριστές πόρων θα αναφερθεί παρακάτω.

Δυναμική εγγραφή

Συγκεκριμένοι διαχειριστές πόρων, ειδικά αυτοί που εμπλέκονται σε σχετικά λίγες δοσοληψίες, μπορούν να ζητήσουν από τον διαχειριστή δοσοληψιών να υποθέσει ότι δεν συμμετέχουν σε μια δοσοληψία. Αυτοί οι διαχειριστές πόρων πρέπει να καταγραφούν από τον διαχειριστή δοσοληψιών πριν αρχίσουν να κάνουν δουλειά για τις εφαρμογές, για να ορίσουν το πότε η εργασία μιας δοσοληψίας τους αφορά. Ο διαχειριστής δοσοληψιών ποτέ δεν καλεί τέτοιους διαχειριστές πόρων με ρουτίνες `xa_start()`. Ένας τέτοιος διαχειριστής πόρων δυναμικά καταγράφει την παρουσία του με δική του πρωτοβουλία και δεν αλλάζει την συμπεριφορά του διαχειριστή δοσοληψιών σε σχέση με τους άλλους διαχειριστές πόρων.

Ολοκλήρωση κλάδου δοσοληψίας

Ο διαχειριστής δοσοληψιών καλεί την `xa_prepare()` για να ζητήσει από τον διαχειριστή πόρων να προετοιμαστεί για commit ενός κλάδου δοσοληψίας. Ο διαχειριστής πόρων τοποθετεί όλους τους πόρους που έχει δεσμεύσει σε μια κατάσταση που να μπορεί να κάνει μόνιμες τις αλλαγές αν λάβει ένα `xa_commit()` ή να τις ακυρώσει αν λάβει ένα `xa_rollback()`. Μια καταφατική επιστροφή μετά από ένα `xa_prepare()` εγγυάται ότι ένα ακόλουθο `xa_commit()` ή `xa_rollback()` θα επιτύχει, ακόμα και αν ο διαχειριστής πόρων αποτύχει μετά την απάντηση στο `xa_prepare()`. Ο διαχειριστής δοσοληψιών καλεί την `xa_commit()` για να κατευθύνει το διαχειριστή πόρων να καταχωρήσει μόνιμα της αλλαγές ενός κλάδου δοσοληψίας. Ο διαχειριστής δοσοληψιών καλεί την `xa_rollback()` για να ζητήσει από τον διαχειριστή πόρων να ακυρώσει τις αλλαγές. Ο διαχειριστής πόρων ακυρώνει τις αλλαγές και απελευθερώνει τους πόρους που είχε κρατήσει.

Για να μπορέσει ο διαχειριστής δοσοληψιών να καλέσει μια `xa_prepare()`, θα πρέπει όλες οι σχέσεις να έχουν ολοκληρωθεί με την `xa_end()`. Τότε οποιοδήποτε νήμα μπορεί να αρχικοποιήσει την ολοκλήρωση του κλάδου. Ο διαχειριστής δοσοληψιών μπορεί να εποπτεύσει την ολοκλήρωση του κλάδου με ένα ξεχωριστό νήμα του προγράμματος εφαρμογής το οποίο πραγματοποίησε κάποια εργασία για την συνολική δοσοληψία. Αυτό το μοντέλο κατανεμημένης επεξεργασίας δοσοληψιών επιτρέπει στους διαχειριστές πόρων να ολοκληρώνουν κλάδους δοσοληψιών με ευριστικό τρόπο. Ο διαχειριστής πόρων δεν μπορεί να απορρίψει τη γνώση από έναν τέτοιο κλάδο μέχρι ο διαχειριστής δοσοληψιών να το επιτρέψει καλώντας την `xa_forget()` για κάθε κλάδο.

Καταστάσεις λειτουργίας ρουτινών: Σύγχρονα, χωρίς παρεμπόδιση (non-blocking), ασύγχρονα

Σύγχρονη λειτουργία ρουτινών

Οι ρουτίνες `xa_` τυπικά καλούνται σύγχρονα: ο έλεγχος δεν επιστρέφει σε αυτόν που την κάλεσε μέχρι να ολοκληρωθεί η λειτουργία τους. Μερικές ρουτίνες όπως η `xa_start()` μπορούν να μπλοκάρουν το νήμα που τις κάλεσε.

Λειτουργία ρουτινών χωρίς παρεμπόδιση

Συγκεκριμένες κλήσεις ρουτινών xa_ κατευθύνουν τον διαχειριστή πόρων να λειτουργήσει σύγχρονα με αυτόν που τις καλεί αλλά χωρίς να τον μπλοκάρουν. Εάν ο διαχειριστής πόρων δεν μπορεί να ολοκληρώσει την κλήση χωρίς μπλοκάρισμα, το αναφέρει αμέσως.

Ασύγχρονη λειτουργία ρουτινών

Οι περισσότερες ρουτίνες έχουν τη μορφή ασύγχρονης κλήσης. Οι ασύγχρονες κλήσεις πρέπει να επιστρέψουν αμέσως. Αυτός που καλεί μπορεί έπειτα να καλέσει την `xa_complete()` για να ελέγξει την ολοκλήρωση της ασύγχρονης λειτουργίας.

Ανάνηψη από αποτυχία (failure recovery)

Ο διαχειριστής δοσοληψιών πρέπει να βεβαιώνει την σωστή ολοκλήρωση όλων των κλάδων της δοσοληψίας. Ο διαχειριστής δοσοληψιών καλεί την `xa_recover()` κατά τη διάρκεια της ανάνηψης από αποτυχία για να πάρει μια λίστα από όλους τους κλάδους οι οποίοι προετοιμάστηκαν ή ολοκληρώθηκαν με ευριστικό τρόπο. Ένας διαχειριστής πόρων μπορεί να σημειώσει έναν κλάδο δοσοληψίας σαν rollback-only οποιαδήποτε στιγμή μετά από ένα επιτυχημένο prepare. Ο διαχειριστής δοσοληψιών το ανιχνεύει αυτό όταν επιστρέφει έναν rollback-only κωδικό. Ένας διαχειριστής πόρων μπορεί επίσης να μονομερώς να ανακαλέσει και να ξεχάσει έναν κλάδο οποιαδήποτε στιγμή εκτός μόνο μετά από ένα επιτυχημένο prepare. Ο διαχειριστής δοσοληψιών το ανιχνεύει αυτό όταν μια επόμενη κλήση δηλώνει ότι ο διαχειριστής πόρων δεν αναγνωρίζει το XID.

Αν ένα νήμα ελέγχου τερματιστεί, ο διαχειριστής πόρων πρέπει να διαχωρίσει και να ανακαλέσει όλους τους σχετιζόμενους κλάδους ελέγχου. Εάν ο διαχειριστής πόρων διαπιστώσει κάποια αποτυχία υλικού ή λογισμικού, θα πρέπει μονομερώς να ανακαλέσει όλους τους κλάδους που δεν έχουν προετοιμαστεί με επιτυχία.

3.8 OBJECT RELATIONAL ΕΠΕΚΤΑΣΕΙΣ ΣΤΗΝ SQL (SQL/Object, SQL/Foundation)

Όπως θα παρουσιάσουμε αναλυτικά στο κεφάλαιο 4, η ύπαρξη των αντικειμενοστρεφών συστημάτων βάσεων δεδομένων είχε ως αποτέλεσμα την η υιοθέτηση αντικειμενοστρεφών επεκτάσεων στο SQL-3. Πιο συγκεκριμένα, υπάρχουν τέσσερις κύριοι τομείς στους οποίους συνοψίζεται αδρά, η επίδραση αυτή. Οι τομείς αυτοί είναι:

- Εισαγωγή τύπων που ορίζονται από το χρήστη -UDTs
- Κατηγορήματα για την κατασκευή τους -Type Predicate
- Συναρτήσεις που μπορούν να οριστούν από το χρήστη - User Defined Functions & Procedures
- Κατάργηση της πρώτης κανονικής μορφής (1NF) - Collection Types

3.8.1 Τύποι Ορισμένοι από τον Χρήστη

Οι τύποι που μπορούν να οριστούν από το χρήστη (*User Defined Types -UDTs)* διαιρούνται με τη σειρά τους σε διάφορες κατηγορίες:

- Διακριτοί τύποι - Distinct Types
- Αφηρημένοι τύποι - Abstract Data Types (ADTs)
- Τύποι γραμμών / Τύποι αναφοράς - Row Types / Reference Types

User-defined Types - Distinct Types

Οι *διακριτοί τύποι* (distinct types) αποτελούν την πιο βασική έκφραση των ορισμένων από τον χρήστη τύπων. Ουσιαστικά, αποτελούν μετονομασία ενός βασικού τύπου (π.χ. `DECIMAL`, `REAL`, `INTEGER`), στον οποίο θα αναφερόμαστε ως πηγαίο τύπο (source type). Σαν παράδειγμα, στο σχήμα 3.7 δίνουμε την δήλωση δύο διακριτών τύπων, παράγωγα του `DECIMAL`, που θα αναπαριστούν το αμερικάνικο (`US_DOLLAR`) και το καναδικό δολάριο (`CND_DOLLAR`).

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9.2)
CREATE DISTINCT TYPE CDN_DOLLAR AS DECIMAL (9.2)
```

Σχήμα 3.7 Δήλωση διακριτών τύπων

Οι διακριτοί τύποι κληρονομούν τους συντελεστές σύγκρισης των αντίστοιχων πηγαίων τύπων. Αν και οι διακριτοί τύποι έχουν την ίδια εσωτερική δομή με τον πηγαίο τύπο, δεν είναι άμεσα συγκρίσιμοι με αυτόν. Για να γίνει δυνατή η σύγκριση μεταξύ διακριτών τύπων του ίδιου πηγαίου τύπου, καθώς και διακριτού τύπου με τον πηγαίο τύπο, πρέπει να χρησιμοποιηθούν συναρτήσεις μετατροπής (casting).

Αφηρημένοι τύποι δεδομένων

Εκτός από τους διακριτούς τύπους δεδομένων μπορούμε να ορίσουμε και *αφηρημένους τύπους δεδομένων* (Abstract Data Types, ADTs), οι οποίοι αποτελούν οντότητες με συμπεριφορά και συγκεκριμένη δομή. Στο σχήμα 3.8 φαίνεται ο ορισμός σε SQL ο ορισμός ενός ADT που αναπαριστά την διεύθυνση ενός ατόμου.

Τα χαρακτηριστικά των αφηρημένων τύπων δεδομένων είναι:

- *Ενθυλάκωση:* Η πρόσβαση στα πεδία του τύπου περιορίζεται σε συναρτήσεις που ορίζονται στο ορατό μέρος του κάθε αφηρημένου τύπου. Οι συναρτήσεις αυτές είτε επιστρέφουν την τιμή ενός πεδίου του τύπου (observer functions) ή αναθέτουν τιμή σε κάποιο πεδίο (mutator functions) και ορίζονται αυτόματα από το όνομα του πεδίου. Στο προηγούμενο σχήμα π.χ. ορίστηκαν αυτόματα δύο συναρτήσεις για το πεδίο `zip`: η `zip(address)` που επιστρέφει την τιμή του πεδίου, και η `zip(address, integer)` η οποία μεταβάλει την τιμή του πεδίου.

```
CREATE TYPE address
    (street char (30),
     city char (20),
     state char (2),
     zip integer);
```

Σχήμα 3.8 Ορισμός ADT

- *Δημιουργία Στιγμιοτύπων:* όπως στην προηγούμενη περίπτωση, έτσι και εδώ ορίζεται αυτόματα μια κατασκευαστική συνάρτηση (constructor function) η οποία επιτρέπει την δημιουργία στιγμιοτύπων (instance) του ADT. Η συνάρτηση έχει το ίδιο όνομα με το όνομα του τύπου. Για την δημιουργία π.χ. ενός στιγμιοτύπου του τύπου `address`, αρκεί να καλέσουμε την συνάρτηση `address()`.
- *Κληρονομικότητα:* Οι αφηρημένοι τύποι μπορούν να οριστούν και να κληρονομήσουν τα χαρακτηριστικά άλλων αφηρημένων τύπων. Αυτό σημαίνει επίσης ότι ένας υποτύπος (subtype) μπορεί να χρησιμοποιηθεί οπουδήποτε χρησιμοποιείται κάποιος από τους υπερτύπους του (substitutability). Στο σχήμα 3.9 παρουσιάζεται ο ορισμός ενός βασικού ADT που αναπαριστά σχήματα και ο ορισμός τριών συγκεκριμένων σχημάτων.

```
CREATE TYPE shape …
    CREATE TYPE point UNDER shape …
    CREATE TYPE line UNDER shape …
    CREATE TYPE polygon UNDER shape …
```

Σχήμα 3.9 Κληρονομικότητα ADT

- *Late binding:* Η SQL-3 δίνει την δυνατότητα ορισμού συναρτήσεων διαχείρισης πάνω σε αφηρημένους τύπους, χρησιμοποιώντας την SQL/PSM είτε κάποια γλώσσα τρίτης ή τέταρτης γενιάς. Οι υποτύποι ενός ADT έχουν την δυνατότητα να ξαναορίσουν ορισμένες συναρτήσεις, ώστε να έχουν νόημα στο συγκεκριμένο υποτύπο. Έστω π.χ., ότι για τον ADT shape έχει οριστεί η συνάρτηση surface, η οποία επιστρέφει το μέγεθος της επιφάνειας ενός στιγμιότυπου του shape. Κάθε υποτύπος του shape μπορεί να ξαναορίσει την συνάρτηση αυτή ώστε να λειτουργεί σωστά ανάλογα με τα χαρακτηριστικά του υποτύπου. Η καθυστέρηση στην κλήση (late binding) εξασφαλίζει ότι όταν κληθεί η surface, θα κληθεί η σωστή έκδοση της συνάρτησης ανάλογα με τον τύπο του shape στο οποίο αναφερόμαστε.

Οι ADTs χρησιμοποιούνται όπως ακριβώς και οι προκαθορισμένοι τύποι της SQL-3. Μπορούμε π.χ. να χρησιμοποιήσουμε τον τύπο `address` για τον ορισμό της σχέσης `employee` όπως φαίνεται στο παρακάτω σχήμα.

```
CREATE TABLE employee
    (fist_name char(20),
     last_name char(20),
     addr address)
```

Η εισαγωγή καινούριων τιμών γίνεται μέσω της κατασκευαστικής συνάρτησης, ενώ τα πεδία του τύπου σε μια επερώτηση προσπελαύνονται χρησιμοποιώντας δύο τελείες (dot notation), όπως φαίνεται στο σχήμα 3.10

```
INSERT employee
   VALUES ("Vassilis","Gkoysgkoynis",address("Ir.
        Polytechniou 9","Ath","GR",157 72))

SELECT fist_name,address..zip
FROM employee
WHERE address..city = "Ath"
```

Σχήμα 3.10: Χρήση ADT σε ερώτηση

Επώνυμοι Τύποι Γραμμής (Named Row Types)

Οι τύποι αυτοί αναφέρονται σε μια ολόκληρη πλειάδα μιας σχέσης. Είναι τύποι που ορίζονται από τον χρήστη, χωρίς όμως να υποστηρίζουν το χαρακτηριστικό της ενθυλάκωσης. Το σχήμα 3.11 δίνει ένα παράδειγμα ορισμού ενός row type.

```
CREATE ROW TYPE account_t
    (acctno    INTEGER,
     cust      REF(customer_t),
     type      CHAR (1),
     opened    DATE,
     rate      DOUBLE PRECISION,
     balance   DOUBLE PRECISION);
```

Σχήμα 3.11 Ορισμός named row type

Χρησιμοποιώντας τους επώνυμους τύπους γραμμής μπορούμε να ορίσουμε σχέσεις:

```
CREATE TABLE account OF account_t
    (PRIMARY KEY acctno);
```

Στο παράδειγμα ορισμού του row type χρησιμοποιήθηκε και μία αναφορά (`cust REF(customer_t)`). Οι τύποι αναφοράς παραπέμπουν σε δηλώσεις τύπων (στην συγκεκριμένη περίπτωση του τύπου `customer_t`) και χρησιμοποιούνται κυρίως για να μοντελοποιήσουν σχέσεις μεταξύ επώνυμων τύπων γραμμών. Οι τύποι αναφοράς μπορούν να έχουν εμβέλεια, και χρησιμοποιούνται σε ερωτήσεις με τον τελεστή `->`:

```
CREATE TABLE account OF account_t
    (PRIMARY KEY acctno,
     SCOPE FOR cust IS customer);

SELECT a.acctno, a.cust->name
FROM account a
WHERE a.cust->address..city = "Hollywood"
    AND a.balance > 1000000;
```

Οι επώνυμοι τύποι γραμμών υποστηρίζουν το χαρακτηριστικό της κληρονομικότητας. Έτσι, ένας row type μπορεί να είναι υποτύπος ενός ή περισσότερων row types, κληρονομώντας δομή και συμπεριφορά από αυτούς.

```
CREATE TYPE empoloyee (name CHAR (20), salary DECIMAL (10,2))
CREATE TYPE manager UNDER employee (bonus DEMICAL (10,2))
CREATE TYPE ssummer_stundent UNDER employee (school VARCHAR
    (30))
```

3.8.2 Type Predicate

Η SQL-3 παρέχει τον τελεστή TYPE με τον οποίο μπορούμε να ελέγχουμε τον τύπο ενός πεδίου όταν εκτελείται μια ερώτηση. Ο τελεστής αυτός έχει νόημα, βέβαια, όταν το συγκεκριμένο πεδίο που εξετάζεται έχει υποτύπους:

```
CREATE TABLE drawing
    (name char(20), s shape)

SELECT name
FROM drawing
WHERE TYPE(s) IN circle
```

3.8.3 Συναρτήσεις και Μέθοδοι ορισμένες από τον χρήστη

Όπως στους ADTs έτσι και στους επώνυμους τύπους γραμμής αλλά και στους διακριτούς τύπους, υπάρχει η δυνατότητα ορισμού συναρτήσεων διαχείρισης. Οι συναρτήσεις αυτές μπορούν να οριστούν είτε μέσω της SQL/PSM, είτε χρησιμοποιώντας μια γλώσσα 3GL ή 4GL. Και σε αυτή την περίπτωση ισχύει το χαρακτηριστικό του late binding, ώστε να καλείται η σωστή συνάρτηση ανάλογα με τον υποτύπο στον οποίο αναφερόμαστε.

3.8.4 Συγκεντρωτικοί Τύποι (Collection Types)

Η SQL-3 παρέχει προκαθορισμένους συγκεντρωτικούς τύπους δεδομένων, όπως *σύνολα (sets)*, *λίστες (lists), πίνακες (arrays)* και *πολυσύνολα (bags)*.

```
CREATE TYPE point
    (x integer,
     y integer)

CREATE ROW TYPE polygon
    (points SET(point))
```

Όταν χρησιμοποιούνται σε ερωτήσεις, οι τύποι αυτοί αντιμετωπίζονται σαν σχέσεις:

```
SELECT *
FROM polygon p
WHERE (SELECT count(*) FROM TABLE p.points) > 10
```

3.9 ΑΝΑΦΟΡΕΣ

[KMN97] K. Kulkarni, N. Mattos, A. K. Nori, “Object-Relational Database Systems - Principles, Products and Challenges” Tutorials of 23rd Int’l VLDB Conference, Athens, Greece, 1997

[MM96] J. Melton, N. Mattos, “An overview of SQL3 - the Emerging New Generation of the SQL Standard”, Tutorials of 22nd Int’l VLDB Conference, Mumbai, India, 1996

Διάφορα κείμενα σχετικά με το SQL-3 stantard μπορεί κανείς να βρεί στο ftp://jerry.ece.umassd.edu/isowg3/dbl/BASEdocs/public/