VI. Adding / Deleting Tables & Relationships

- creating a table is like normal; just don't forget Fk and Fk constraints
- deleting tables means dropping Fk constraints, then dropping the tables
- construct Dependency Graphs once removed to understand which programs will be affected by the deletion

• this is all for the purpose of either normalization or denormalization

VII. Forward Engineering

- many skeptics don't use redesign tools bc its automated & the importance level to make the correct changes is very high
- automation may not be the best idea when dealing w/ lots of data (write my own SQL)

# Database Ch. 9 notes - Managing Multiuser Databases

I. Importance of Working With an Installed DBMS Product

- hands-on practice is the best!

II. DB Administration

- data Admin: mgmt-oriented function that concerns corporate data privacy / security issues
- db Admin: technical term referring to a particular db or apps tied to it
  - vary in size from personal db to large org db
  - large companies will have a DBA whereas in personal db's, the DBA is the owner

DBA: facilitate development & use of the db

Managing the Database Structure

- DBA's need to be a part of it all: design, modeling, requirements, implementation, seeding, backups, maintenance, et

Config Control

- change to db design is inevitable in the long-run
* - the db is most vulnerable to failure after it's structure has been changed
- Figure 9-1 (pg426) - Summary of DBA tasks

Documentation

- know what changes have been made, how they were made, and when ~~they~~ they were made
- CASE tools; Versioning software; Data dictionaries
  - like when you find a fix to a helpdesk issue, you document it!
- important when talking about historical data that's been archived
- Figure 9-2 (pg427) - Summary of DBA's resp. for Managing Db structure

III. Concurrency Control

- ensuring one user's work doesn't affect another user's work
  - processing should be the same if 1 entry or 100 entries in the table

The need for Atomic Transactions (All or none)

- transaction aka "Logical Unit of Work"; series of actions on the db so all are performed successfully, or none at all
  - 'atomic' b/c it's performed as a unit

Ex: an order makes changes to 2 columns & adds a row to the ORDER table
  - Either all of these proces pass, or none do.

* Figure 9-3 (6)

Concurrent Transaction Processing

- interleave them; have the CPU portion each task, so they'll both gradually process together
  - Figure 9-4 (pg430)

## The Lost Update Problem

Ex. 2 user's want to order different qty's of the same item
- Cust A reads Item's qty
- Cust B reads Item's qty
- set Item's qty decr. by 5 by Cust A
- write new qty for Item to db

ISSUE: Cust B has already read the Item's qty
- set Item's qty decr by 3 by Cust B
- write new qty for Item to db

= lost update problem

- dont give incorrect data to read
  - instead, practice resource locking: prevent from obtaining the same record while record is about to be changed

## Resource Locking

- prevents concurrent processing problems by locking the resource using a 'lock' command

### Lock Terminology

*Figure 9-6

implicit locks: lock performed by the DBMS (most common)
explicit locks: locked by the cmd
lock granularity: the size (scope) of the lock
- large gran = easy to administer, more conflicts

exclusive lock: locks item from any other access
shared lock: locks from changing, but not reading

### Serializable Transactions

- a scheme (method) to process concurrent trans.
- 2phase locking: transactions can get locks. But, once the first lock is released, can't get another lock
  - locks may not be released until commit() or rollback() commands

## Deadlock

- each user is waiting for a resource the other has locked
  - can prevent or allow, then breaking it
  - require users to lock all requests (Items) at once

## Optimistic vs Pessimistic Looking

- invoking locks

Fig 9-8: Optimistic: assume no conflict will occur; if conflict, then transaction repeats until no conflict

Fig 9-9: Pessimistic: assume conflict; issue locks → process → free locks

- if user is taking long time, optimistic is better (improves throughput)
  - but, on flipside, optimistic transaction could repeat many times if a large process
- book says optimistic is a better choice

## SQL Transactions Control Lang & Declaring Lock Characteristics

- can declare the type of locking behavior & placing & removing locks dynamically
- Standard commands:
  - BeginTransaction
  - CommitTransaction: makes changes to Db permanent
  - RollbackTransaction: undo

* Figure 9-10: how you want to do it (pg435)

## Implicit & Explicit COMMIT Transaction

- Oracle Db doesn't provide implicit Commits

## Consistent Transactions

- may see ACID applied to transactions
  - ACID: atomic, consistent, isolated, durable
    - atomic: all or none
    - durable: all commits are permanent

Consistent:

1.) Stmt-level: doesn't matter how long the stmt takes, it'll complete before moving on
2.) transaction-level: all rows affected by either of SQL Stmts are protected from changes

isolated: refer to:
* 9-11 (pg 437) - Summary of Data Read Problems (i.e. dirty, nonrepeatable, phantom)
  - to declare an isolation level, refer to:
* Figure 9-12 (pg 438) - Summary of Transaction Isolation Levels
  - the more restrictive, the less throughput

SQL Cursors
- a pointer into a set of rows in a table
* Figure 9-13 - Summary of SQL Cursor Types (4)

## IV. Database Security
- ensure only auth users can perform auth tasks @ auth times

Processing Rights & Responsibilities
- assign roles so permissions can be assigned to groups of people
- responsibilities go w/ rights!
- make sure these are easy to be changed

DBMS Security
- Permissions are managed by SQL Data Control Language (DCL) using: Grant / Revoke stmts
- all DBMS come w/ a username/password verification
- Internet apps could have an 'Unknown Public' group to handle guest users

DBMS Security Guidelines
* Figure 9-16 (pg 442) - Summary of ↩
  - Security mindset to least privilege!
  - Figure contains info on user/pw, security protection, ports

strong password: $x \geq 15$ in length; upper & lowercase; numbers; specials

Application Security
- provided by Web Server if using Web App
- could get/setup a security db w/ specific checks on roled user

SQL Injection

- when data from user are used to modify a SQL Stmt
  - be careful of raw SQL queries in code, as they are passed as strings & susceptible to MitM attacks

## V. Database Backup & Recovery

- if it ever breaks, it's gotta catch up once back up & working
- it's not simple to fix if something breaks as configurations could never be the same. Instead, we can:
  1.) Recover via reprocessing
  2.) Recover via rollback / rollforward

Recovery via Reprocessing

- go back to the last known point it was working right & start playing catchup from there
- not a good option when dealing w/ a concurrent processing system

Recovery via Rollback / Rollforward

* Figure 9-17 (pg 496) - Undo & Redo Transactions
- periodically make a copy of the db (makeshift db versioning control)
  - 'database save': ↻
  - if a failure occurs, restore the db save (rollforward) redo
  - or, we could undo the changes that caused the errors (roll back) ↩
- both methods require a log to be kept (event log)
- undo = before images / redo = after images
- logs are where 'pointers' can be utilized
- checkpoints: point of sync b/w db & transaction log (no processes waiting)
  - it's a recovery point
  - feasible option

## VI. Managing the DBMS

- analyze / monitor system performance, track & control issues, generate reports, improvement ideas (keys, relationships, etc.), tuning / optimization

* Figure 9-20 (pg 499) - Summary of DBA's Resp. for Managing the DBMS

## Managing the Data Repository

- Data Repository: collections of metadata about db's, db apps, Web pages, users, & other app components
  - very useful for when senior mgmt comes to you asking to expand the db bc they're selling a new line of clothing
  - should be considered an important part of system deliverables when designing the db.
  - active repos: metadata is auto-created as system components are created
  - passive repos: metadata is manually created

- often, building a new system is easier than adapting a current one