File: {Data (D) Dependency, • D Redundancy • D Sharing • Development Times, Maintenance }

Relation: {Unique Names, Rows, Attrs • Atomic Attributes • Order is Irrelevant }

Abstraction Lvls:
- {External Schemas (Views) {} Logical (Conceptual) Schema {} Physical Schema }
- Logical & Physical Data Independence

Conceptual Database Design:
(Enhanced) ER [Generalisation/Abstraction]
- Unary/Binary/Tertiary Relationships
- ∂ Systems: ∂ Namings, ∂ Semantics • ISA:
  - Overlap Constraints:
    - ∆Disjoint (Max(1))
    - Overlapping (default: Max(∞))
  - Covering Constraints:
    - Total (Min(1) []=∆–[])
    - Partial (default: 0+)

UML • { ISA: Complete. Disjoint, Incomplete, Overlapping }

Relational Algebra:
- Union (U) • Intersection (∩) • Difference (–)
- Selection (σ) • Projection (π) • Cross-Product (X)
- Join (⋈) • Rename (ρ)
- $\pi_{name, uosCode}$ ($\sigma_{country='Aus', hid=sid}$ (Student x Hobbies))
- $\rho_{classLis(2\text{->}cid, 4\text{->}uos\_code)}$ (Enrolled x UnitOfStudy)
- $\pi_{address}(\sigma_{title='Database' \vee title='Data Management'}$(Publisher $\bowtie_{pname=publisher}$ Book))
- $Reaches_i = \pi_{frm, to}$(Flights $\bowtie_{to=frm}$ $Reaches_{i=i}$) U $Reaches_{i-1}$
- Division: via [»NOT EXISTS«, IN] and
  [(U), (∩),(–)] : [R(all) / S(filtered)]

INTEGRITY CONSTRAINTS:
Static ICs:
- Domain Cs: CHECK, DEFAULT, NOT NULL,
  CREATE DOMAIN Grade CHAR CHECK (value in ('A','B','C'))
- Key / Referential (/FK) Cs,
  Referential Integrity: • FK } [1]
  CON t_fk FK (sid) REFS Student ON [DELETE/UPDATE] [NO ACTION/CASCADE/SET NULL/SET DEFAULT]
- Semantic ICs (Assertions, Checks)
  ASSERTION: a predicate expressing a condition the db must always satisfy. Used for ICs over several tables. Are always checked, even if one table is empty. Are schema objects (~tables, views), may intro big Overhead. CREATE ASSERTION name CHECK condition.
- ALTER TABLE Person ADD CONSTRAINT LicenseChk CHECK (licenseValidTil > CURRENT_DATE);
  //+ { DROP C. / RENAME C. TO C. / MODIFY LicenseValidTil NOT NULL }
- DEFFERENCE:
  CON f FK (n) REFS n [NOT DEFERABLE] / [DEFERABLE [INITIALLY DEFFERED / INITIALLY IMMEDIATE]]

Dynamic IC:
- A TRIGGER is a statement that is executed automatically if specified modificiations occur. maintain FK and Semantic C.s, commonly used with ON DEL/ UPDATE. dynamic business rules. -monitoring, keep track of entries (e.g. sensor). simplified app design. [ON event IF precondition THEN action]. Traditionally for maintaining summary data (now materialized views' job), and replicated dbs by recording ∂s (now built-in support).
- CREATE TRIGGER name [AFTER/BEFORE] [INSRET/ DELETE/UPDATE OF attr] ON table REFERENCING [NEW/OLD] [TABLE/ROW] AS var-name FOR EACH [STATEMENT % / ROW, WHEN %]
- use BEFORE triggers ((row)row granuality)
  for checking ICs
- use AFTER triggers ((table)statement granularity)
  for integrity maintenance and update propagation

SQL:
Security (= View + GRANT/REVOKE/ROLE):
- SQL, No Authorization support at tuple level
- CREATE **VIEW** ageStudents AS || SELECT sid*10, name, extract (year from sysdate) - extract (year from birthdate) AS age
  FROM Student
- INSERT INTO --View-- ageStudents(sid, name, age) VALUES (123, 'Bill', 21)
- CREATE **ROLE** manager
  GRANT/REVOKE select, insert ON ageStudents TO manager
  GRANT manager TO mrjobs

GENERAL:
- DCL (Data Control Laanguage)
  [controls db, adminstering privelages to users] »
- DDL (Data Definition Language) [CREATE, DROP, ALTER, ICs: PK, FK REFS, NULL] »
- DML (Data Manipulation Language) [select, insert, delete, update]
- --
- SELECT (DISTINCT) »
  FROM »
  WHERE [AND (% (IN (%,%)) OR %)) »
  (GROUP BY ~π) »
  (HAVING ~ σ) »
  ORDER BY
- =,>,>=,<,<=,!=,<> [for NOTs, use Division]
- LIKE: [%]=substring, [_] = char, ||=concat
- --
- JOIN (common domain)
- EQUI-JOIN (equality, redundant columns)
- NATURAL-JOIN (x 2"" columns)
- OUTER JOIN (nulls) • L/R INNER JOIN (x nulls)
- Student LEFT OUTER JOIN Enrolled USING (sid)
- Student INNER JOIN Enrolled ON (sid=eid)
- Aggregate F()s:
  avg(), first(), last() max(), min(), sum(), count(*) //~~nulls~~, count(prereqUosCode) //+nulls
- Set Operations:
  UNION, INTERSECT, EXCEPT(Oracle=MINUS)
- NULL + 3ValuedLogic:
  unknown(u),True(T),False(F) • u||T=T • u||F=F • u||u=u • T&u=u • F&u=F • u&u=u • !u=u

SUBQUERIES:
- S_ F_ W
  _IN / _EXISTS / UNIQUE / NOT EXISTS (S_ F_ W_)
- IN:
  Compares value v with set of values V,
  True if ((v) in V)
- EXISTS:
  Checks if Nested is Empty (no tuples).
- --
- Correlated:
  Depends on Outer Query,
  executes Once for entire Outer
- Non-Correlated:
  Independent of Outer,
  executes once for Each Row of Outer

Extra:
- CREATE TABLE table (sid INTEGER, name CARCHAR(20), CONSTRAINT table_PK P.K. (sid) CON table_U UNIQUE (name));
- UPDATE s_table SET status='a' WHERE id='111'
- DELETE FROM table WHERE id>'111'
- ORDER BY year DESC, semester DESC
- ---
- //Find lecturer(s) that have already taught Every second year INFO subject ['for all']//
  Get Staff Name, WHERE NOT EXISTS ((All INFO2% subjects) MINUS (ALL INFO2 subjects taught by cur_staff))
- //Find all units of study with the very same set of pre-requisites.//
- A,B Set Comparison: A⋈B, WHERE NOT EXISTS (A-B) AND NOT EXISTS (B-A) AND A < B

DATA NORMALIZATION
Anomolies (Evils of Redundancy; ~storage+):
(PKs also prevent insertions, deletions, and nullifying.)
- Insertion: Adding new rows forces user to create duplicate data or to use null values.
- Deletion: Deleting rows may cause a loss of data that would be needed for other future rows.
- Update: Changing data in a row forces changes to other rows because of duplication.

FUNCIONAL DEPENDENCIES (FDs) (+ADV):
- X -> //(functionally) determines// Y
- Schema Normalization: The process of validating and improving a logical design so that it satisfies certain constraints (NFs) that avoid redundancy.
- Normalization maintains consistency and saves space. BUT, performance of querying can suffer because related info is now distributed over several Relations, requiring more JOINs (can ~compensate with indexes).
- Normalization = queries<updates, and vice-versa.
- --
- 1NF: no multivalued, or composite, attributes
- 2NF: 1NF+no partial dependencies, every non-key attr. fully dependent on PK
- 3NF: 2NF+no transitive dependencies,no FD between non-keys
- BCNF: 3NF+all FDs;
  X->Y, X is superkey (→ no 'loops'→)
- 4NF: BCNF+all 'non-trivial' multivalued dependencies are results of keys (basically backwards ⋈). Decomposed like -» BCNFs -».
- --
- Decomposing: Splitting (R) into Normalized (S),(T)
- - Dependency Preserving: FDs on R hold in S,T.
- - Lossless Join: Common Attributes of S and T are a key of Either S or T.
- - There is always a decomposition of any relation into BCNF which is Lossless Join and always a decomp. into 3NF with is lossless join AND dependency Preserving.
- ((CK) is the minimal SK)
- Armstrong's Axioms (Rules):
  Reflexivity: If X ⊆ Y, then Y→X
  Augmentation: If X→Y, then XZ→YZ for any Z
  Transitivity: If X→Y and Y→Z, then X→Z
  Union: If X→Y and X→Z, then X→YZ
  Decomposition: If X→YZ, then X→Y and X→Z
  Pseudotransitivity: If X→Y and YS→Z, then XS→Z
- [1NF - 3NF] less restrictions, achievable lossless and dependency-preserving, [BCNF+] less redundancy, some loss, non-d-p.

DB APPLICATIONS (+ADV)
- Interactive SQL: terminal
- Non-Interactive SQL is included in an app. program written in a host language like C, Java, Python.
- --
- Statement-Level v.s. Call-Level Interface: SLI uses embedded SQL (in C, SQLJ), CLI uses method calls from a special API to communicate with the db (JDBC, ODBC, PHP).
- Host Variables (Data Types;+Nulls): used for data transfer between DBMS and application and map the sql domain data types to the host language data types (+indicators).
- Impedence Mismatch (SQL=Sets, Cursor Concept)
- Error Handling: Important not to expose internal db error msgs to user; potential security issues, not user-friendly.
- Directives (DYNAMIC SQL): When SQL portion not known at compile time, more errors, + vice versa for Standard SQL Statements (STATIC embedded SQL).
- --
- Design Principles:
  - [**Presentation** Layer]
  - [Business **Logic**]
  - [Data **Access** Layer]
  - [Data Management **DBS**] :
  Separate Data Access Layer (xPHP), Error Handling, Validate.
- SQL Injection PROTECTION: Validate User Input, Use Dynamic SQL Statements with explicit, type-checked params (not Static SQL), Restrict user Priveleges, Error Handling.
- --
- Stored Procedures:
  Programs in DB server, callable by app
  (CREATE OR REPLACE F()) / CREATE PROCEDURE name (params) AS BEGIN IF ROLLBACK ELSE END
  +ves: *Correctness-Protection-Insulation, Throughput (dataTransfer)-I/O-Procedure-Only, Higher Abstraction-know-args-only, Efficiency, Authorization, Central Code Base for All Apps, Maintanability.*
  –ves: *Stored Procedures still writtin in Propietary Dialects, Non-Standard Development Ecos.*
- [Multiple APIs] [Automated Data Access Layer]:
  +ve *Reduces code complexity, more robust software*
  –ve *Adds overhead, less flexible; stored procedures may have better performance.*
- [Cursor Concept x?] Collection Support eg. Language Integrated Query (LINQ):
  1. create entity class,
  2. create data context to load from db,
  3. query
- --

Transaction Processing System Architectures:
- [Presentation Logic (GUI)]
  [Processing Logic (Procedures, F()s)]
  [Data Management (Storage Logic, DBMS)]
  - #-tier, # = residences
- 1-Tier: Centralised System: Single Computer, + Easy Maintenance + Central Administration – Bad Scalability for n users
- 2-Tier: Client-Server Systems: Client: PC that requests & uses a service, Server: system that satisfies requests of m client systems.
  [Thick/Thin Client: Data Presentation]>App. Logic<[Server:Data Management]
  +ves: *Flexibility, Scalability++, Data Integrity++, Fail Safety, Stored Procedures, ThinCents-system&vendor independence, long-term (LT) cost < + LT maintenance <,*
  –ves: *More Demanding on Server Performance, Thick Clients - no central place to update, large data transfers, servers need to Trust clients, more complex admin+maintenance (good tools needed). , ∂ in Security.*
- 3-Tier: Client-Server-Middleware / Internet App.s / Web DBs:
  - [Client, Pres.]
  - Split Server into 2 [App./Transaction Server]
  - [Database Server]
  App. Server acts as a workflow controller (router), transaction server does bulk of data processing.
  +ves ,*~2tier++s,* –ves; *high short term $, tools & training, incompatible standards.*

TRANSACTIONS (T)

- Transaction: a collection of 1/+ (r) & (w) operations on 1/+ dbs', which reflects a discrete unit of work. Mirror ∂s in real world. To maintain the relationship between enterprise state and db state, Ts' follow acid properties:
- - Atomicity (0 1)
  - Consistency (ICs Satisfied after each T, defferable during)
  - Isolation (Independent T Executions, serializability)
  - Durabilty (T have Permanent Effects, storage)
- [API for T: BEGIN T. – COMMIT – ROLLBACK]

Concurrency Control

- (Serializabililty Tests via Precedence Graph=Late)
- Two-Pase Locking (2PL):
  – Read shared(S) Lock
  – Write exclusive(X) lock»(!S):
- 2PL All locks acuired before any lock is released; may have Cascading Aborts (CA).
  - (strict)s2PL: T. holds all locks until completion, no CA.
- Lock Granularity(/size):
  - db > table/index > page > row > column
  - [waiting-time v.s. overhead]
- Deadlock:
  Cycle of transaction waiting for locks to be released by each other;
  SOLN: prevention – priorities(e.g. timestamps)
  detection – (algs / timeout).
- --
- Isolation Lvls:
  Lower Levels; may be adequage for some apps; better performance, may allow incorrect schedules.
- Anomolies in on-Serializable Schedules:
  Dirty Read, Non-Repeatable Read », Lost Update
- --
- Snapshot Isolation: Instead of writing over an old value of an object, a new version of it is created with the new value to give a 'snapshot' of the new db. snapshot requires complex maintenance due to multiversion db,
  (–ve) *serializability not guarantered.*
  (+ve) *Good Performance, 'readers never block'.*
- --
- Aborting Ts; Locks only released at commit time to avoid CA, DBMS mantains Log.

INDEXING

- access path: the algorithm and data structure (file-scan/index) used for retrieving and storing data in a table. it affects the efficiency with which queries are run.
- Index: an access path to efficiently locate row(s) via search key fields. no need search entire table, only specific field(s). indices commonly use either a B+tree or a hash as the data structure to locate rows.
- --
- **Main** Index (I.) (/ Primary I. / Integrated I.): for a sequentially ordered file, the index whose srchK. specifies the sequential order of the file.
- **2ndary** Index: an I. whose srchK specifies a dif. order of table entries to seq. order of the file. Sequential scans using Prime. I. is efficient, notso for 2ndary.
- **Clustered** Index: Index entries & rows r' sorted on the same searchK, a 2nd CI would have to be the same, - there can be Max(1) CI per table. When a table is created, a C.I. is generally created on the PK. Good for Range Searches.
- **Unclustered** Index: Index Entries & rows r' Not ordered in the same way. A 2ndary index may be CI or UCI, generally UCI. There can be multiple UCIs on a table.
- Type: **Tree**-Based Indexes:
  B+-Tree, very flexible, only indices to support point queries, range queries, & prefix searches.
- Type: **Hash**-Based Indexes:
  are the fest & fastest for Equality Searches.
- **Covering** Index: An Index that contains all attrs required to evaluate a particular SQL Query i.e. contains all attrs from SFWGH. NOTE: the prefix of the search key (the index) must be the attrs from the WHERE clause. This is the condition of the query.
- (–ves) of Indicies:
  Additional I/O to access index pages. index must be updated when table updates. Space. Indices on non-PKs may have to be ∂d on updates.
- [CREATE INDEX name
  ON table-name(<attribute-list>)]

XML

XML is a semistructured data model. semistructured data may contain missing or additional attrs, multiple attrs, ∂ types in ∂ objects, herogeneous collections. XML describes content & data whilst HTML describest style, presentation, layout.

- Document Type Definition (DTD): XML Grammar
  <!ELEMENT book (title+, author*, price?) >
  <!ATTLIST book genre CDATA #REQUIRED>
  <!ELEMENT genre (#PCDATA) >
- ?(0»1), +(1»∞), *(0»∞)
- --
  <a><des/>Well-Formed ,
  Valid - Follows DTD
- --
- XMLns (namespace): mechanism to include pre-defined element & attribute names
  <shelf xmlns:bs="www. b.com">
  <bs:book>...</bs:book>
  </shelf>
- XML Schema:
  Separates tags &, simple, & complex types. provides primitive data type, type construction, inheritence, value-based constraints, FKs
- --
- DTD vs XML-Schema: DTD uses grammar, 'part-of-relationships', PCDATA, Specified by XML prolog. XML Shema; Structure & typing, ~Inheritance, Data Types, Specified as attribute of the document element (+xmlns).
- --

(Query) XPath:

- / returns root node (first element),
- .=cur_node, ..=par_node
- * any element, *@ any attribute
- // wildcard, descending n levels
- [predicate] conditions
- *//Student[CrsTaken/@ucode='INFO2120']*
- *//Student[sum(.//@Grade) div cound(.//@Grade) > 3.5*

SQL/XML:

- Datatype: XML
- DML: XMLPARSE (CONTENT '<a>...'),
  SELECT A.id, XMLELEMENT (Name'Prof', XMLATTRIBUTES (A.deptId AS 'Dept'), A.name) AS Info From AcademicStaff A

OLAP, Data Warehousing

On Line Transaction Processing vs
On Line Analytic Processing:

- OLTP maintains a db that is an accurate model of real-world enterprise. Short simple Ts. frequend updates to db, access a small fraction of the db.
- OLAP is used to predict trend, complex queries, infrequent updates, Ts. access large fraction of db, historic data.
- --
- OLAP apps based on fact tables (visualizable as Data Cube, or Star Schema (SS)).
  - SS: 1 central fact table, n-dimension tables with FKs from fact table.
- Drilling Down executes a series of queries moving down a heirarchy, Rolling Up moves up.
- Slicing is WHERE clause in SFWG, Pivoting is GROUP BY.
- GROUP BY CUBE (A1, A2), = 4 Queries,
  - [time][time+A1][time+A2][time+A1+A2],
  vice versa for GROUP BY ROLLUP until []

Data Warehouse: stores data (often derived from OLTP) for OLAP & data mining apps. Read-Only, Periodically Refreshed, Often Several Gbs-Tbs, Efficiency for Complex Queries.

- ETL Process:
  Capture/Extract - Data Cleansing - Transform - Load
- data should be;
  detailed (not summarized)
  periodic - historic (metadata mngmnt) load-r5-purge,
  comprehensive,
  uniform format (semantics),
  quality controlled (integrity)
- Has Metadata Repository, Log. Undergoes Incremental Updates

ADV

Hierarchical SQL Data:

- [Adjacency][List Model]
- Materialised.Path.Model
- [L]Nested Set Model[R] (Depth1stTraversal)

Recursive SQL:

```
WITH RECURSIVE Reaches(frm, to) AS (
        SELECT    frm, to
        FROM      Flights
    UNION
        SELECT    F.frm, R.to
        FROM      Flights F, Reaches R
        WHERE     F.to=R.frm )
SELECT * FROM Reaches WHERE frm='SYD'
```

```
CREATE RECURSIVE VIEW Reaches (frm, to) AS
SELECT    frm, to FROM Flights
UNION     //non-recursive(^)//recursive(v)//
SELECT    F.frm, R.to
FROM      Flights F, Reaches R
WHERE     F.to = R.frm
```

Datalog:

Safe Rules:

- each variable; distinguished,
  in an arithmetic / negated subgoal, & must also appear in non-negated relational subgoal
- s(X,Y) :- arc(X,Z) AND arc(Z,Y) AND NOT Arc(X,Y)
- UNSAFE
  S(X) <- R(Y)
  S(X) <- R(Y) AND NOT R(X)
  S(X) <- R(Y) AND X < Y

Extensional Database (EDB):

- *par(c,p)*
- flight(ua450, syd, lax, 0630, 1845).

Intensional Database (IDB):

- *sib(X,Y) :- par(X,P), par(Y,P), X<>Y.*
- *cousin(X,Y) :- sib(X,Y).*
- *cousin(XlY) :- par(X,Xp), par(Y,Yp), cousin(Xp,Yp).*
- reaches(X,Y) :- flights(_,X,Y,_,_).
  reaches(X,Z) :- flights(_,X,Y,_,_), reaches(Y,Z).

Extra:

- uai(Sid,Year,Uai).
- advanced(S,Y,s1) :- uai (S,Y,UAI), UAI>85.
- advanced(S,Y,s2) :- advanced(S,Y,s1), (transcript (S,_,s1,Y,d); transcript(S,_s1,Y,hd)).
- advanced(S,Y,s1) :- adanced(S,YP,s2),(transcript (S,_,s2,YP,d); transcript(S,_s2,YP,hd)), Y is YP+1.

Indexing Heirarchies and Text

- R-Tree: a tree-structured index that remains balanced on inserts & deletes. (cf. Skyline)
- Boolean Retrieval: +ve: Efficient Implemention possible (eg. Inverted List/Index, maps words » docs), –ves: result set difficult to control, no weighting/order to terms
- Vector Space Model: Docs are represented as vectors in term space. all doc vectors together in Document-Term-Matrix. Vector distance = rank. Queries represented same as docs.

(Tuple) TRC & (Domain) DRC

- TRC ~ SQL, SFW :
  { (P.name) | ... / { P | Professor(P) AND ∃T ∈Teaching (P.Id = T.ProfId AND T.CrsCode='INFO2820') }
- DRC ~ Datalog : [∃=Extras] :
  { (Name,DeptId) | ∃Id ∀CrsCode (Professor (Id,Name,DeptId) AND NOT Teaching (Id,CrsCode,'S2002')) }
- QBE (Query By Eg), visual DRC,
  Negation:
  1. ∃ var, 2. (!∃), ∀ var_in_negated_table
- EXISTS(∃) – FORALL(∀) – AND(∧)
  OR(∨) – NOT(¬) – IN(∈)

Materialised Views (MVs)

- a MV is a virtual table that can be directly read. Can Be: partitioned & indexed, queried directly, DML applicable, r5 options, Best in read-intensive ecos.
- (+ves) *useful for summarizing, pre-computing, replicating & distributing data, faster access for expen$ive & complex joins, transparent to end-users*
- (–ves) *performance, & storage, co$ts of maintaining the views.*

Skyline Queries

Given a set of objects p1...pn, the skyline operator returns all objects pi such that pi is not dominated by another object pj. A point p Dominates another point q if the co-ordinate of p on any axis is not larger than the corresponding co-ordinate of q.

- (**Top-K** (or Ranked) Queries retrieve the best k objects that minimize a specific preference f(). Limits: Require ranking f()s, and the number of answers k from the user.)
- **Block Nested Loop**: Scans dataset, listing candidate skyline pts. Compares point p with every other point in list.
  (+ve) *wide applicability,*
  (–ve) *numerous comparisons, inadequacy for on-line processing.*
- **Divide-and-Conquer** (D&C): Recursively, Divide dataset into partions, until fits memory. Compute partial skyline per partiion. Combine.
- **Nearest Neighbour** (NN): Finds NN closest to origin, divides space into 2d non-disjoint regions, recursively search for NN. Skyline Forms. Number of unexplored regions grow rapidly.
  (+ves) *Efficient for finding result,*
  (–ves) *Redundant I/O computation, Explosive to-do list size.*
- **Branch & Bound Skyline** (BBS): NN Basis.
  - (Uses R-Tree (like B-Trees, but used for spatial access methods, i.e. for indexing multi-dimensional info).)
  - Assuming all points are indexed in R-Tree, takes a Top-Down Approach via minDist (lower-left corner » origin).
  - Data Structure: Heap by minDist, List to maintain Current Skyline (Improvable).

Computing Datalog Queries with –ve Cycles:
1. Partition (Split Dependency Graph into +vely-strogly connected components; called strata //No –ve arcs connecting any pair of nodes in the set // For every pair of nodes in the set, there is a +ve path connecting them)
2. Startify (Order the strata: if there is a path from some note in stratum S1 to a node in another stratum S2, then S1 must proceed S2. //Gives Partial Orer //Any total order of the strata that is consistent with this partial order is called stratification)
3. Evaluate the strata in the order of the stratification using the algorithm for computing +ve recursive queries //replace negated relations).