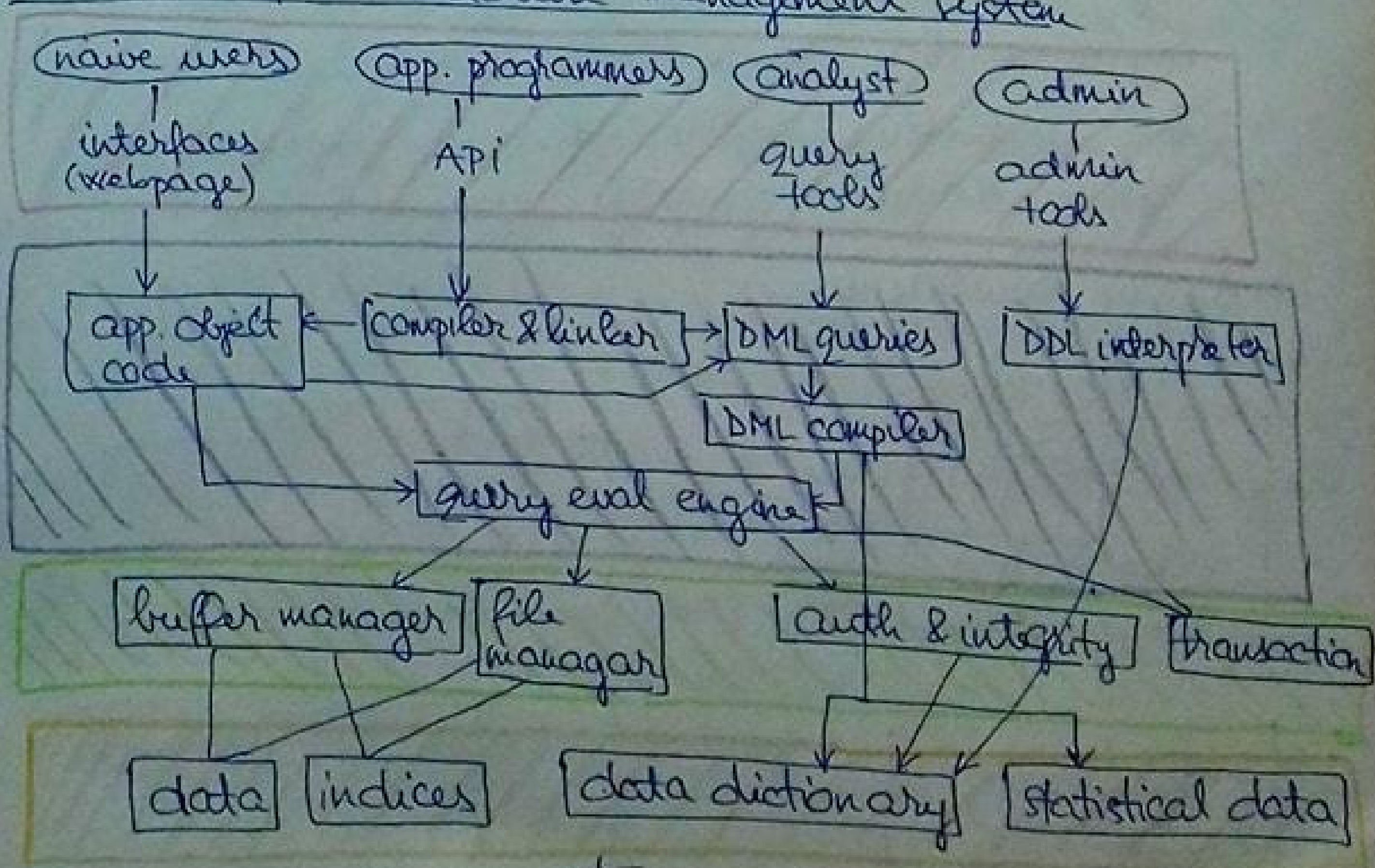
Database systems

FÉLÈVES ÖSSZEFOGLALÓ VIZSGÁRA

1. Introduction, course outline
    - What is a DBMS?
    - What is data modelling?
    - Abstraction levels of a database
    - Creating databases (abstraction levels, logical struct., DDL)
    - Getting data from database : QL → RELATIONAL ALGEBRA
    - Manipulating data in database : DML → SQL
    - Storing data : authorization, recovery → UNDO/REDO LOGGING SHADOWING
    - Multiuser environment : transaction management, concurrency control → LOCKING MECHANISMS TIMESTAMPING
    - Developement lifecycle
    - Active elements : constraints, triggers
    - Database architectures
    - Functional dependencies and normalisation
2. Structure of a database management system



What is a database :

- structured data in relational storage (properties)
- serves multiple users :
  - access, insert, modify data → QL, DML → authority management
  - simultaneous access → transaction manager
- history (recovery)
- File Management (physical layer)

Data modelling, abstraction levels

WORLD → abstraction : no values, just relations
database INSTANCE → satisfying model

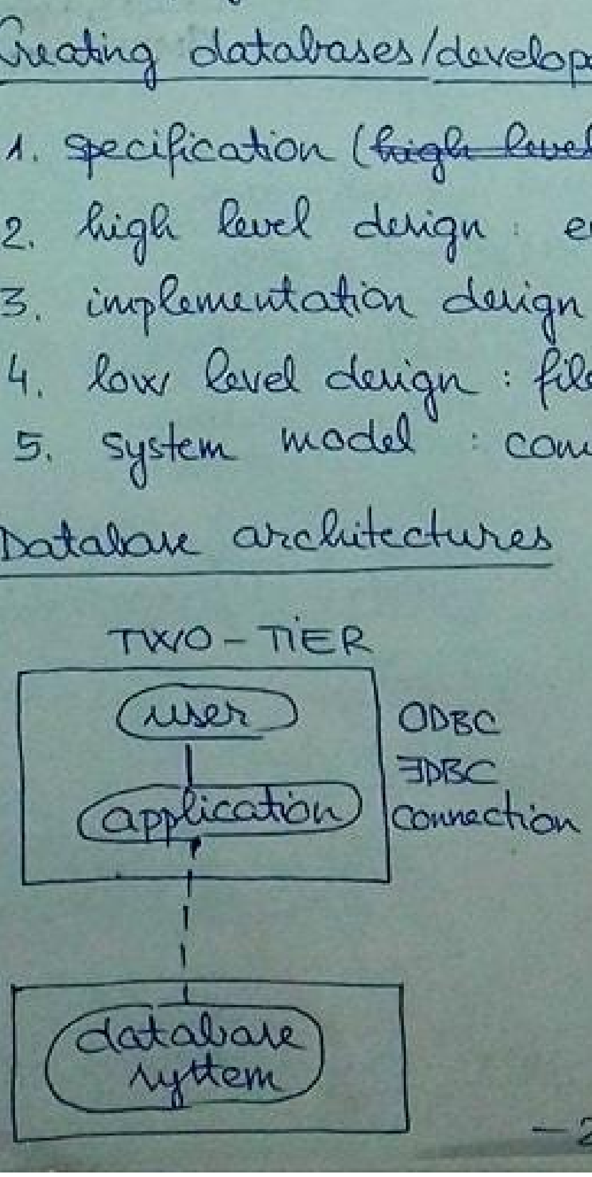
Levels :
- view level : accessible by naive users and apps
- logical / conceptual level : DB admins
- physical level : DBMS programmers :P

Creating databases / developement lifecycle

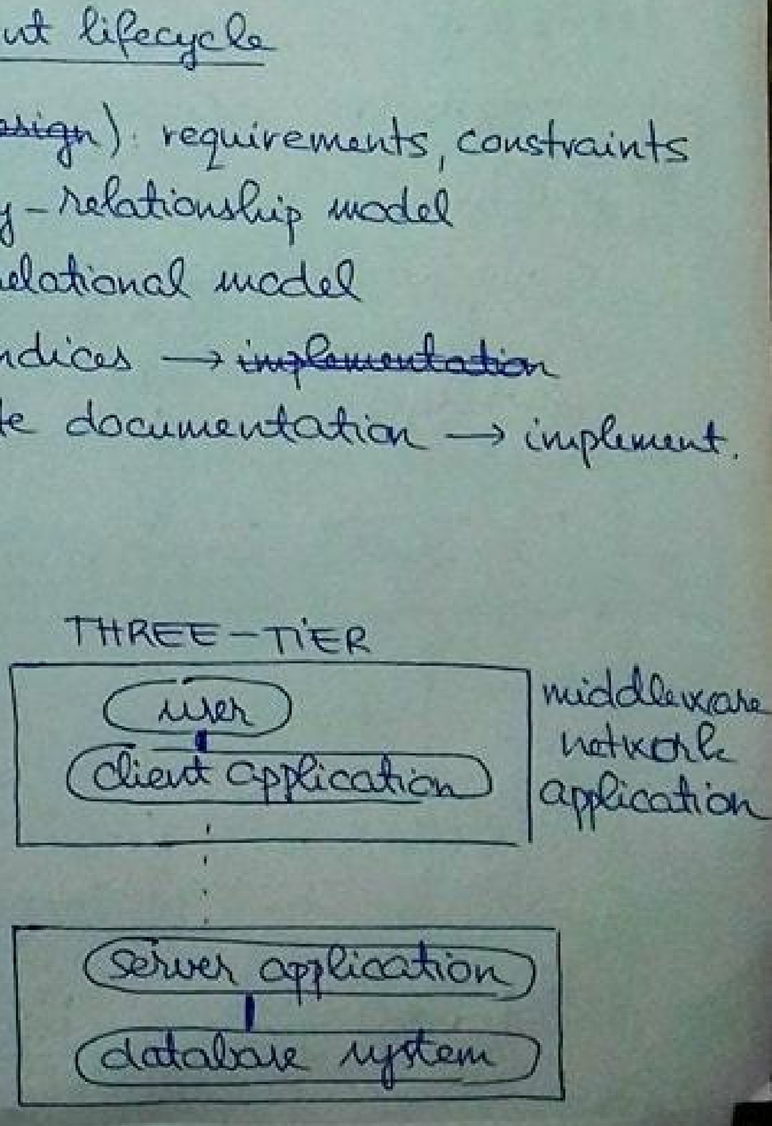
1. specification (~~high level design~~) : requirements, constraints
2. high level design : entity-relationship model
3. implementation design : relational model
4. low level design : files, indices → ~~implementation~~
5. system model : complete documentation → implement.

Database architectures

TWO-TIER



THREE-TIER



Creating databases, modeling data (E-R)

HIGH LEVEL DESIGN

Entity: object distinguishable from others in the microworld, through its attributes

Entity set: group of similar objects, sharing the same properties called attributes

Relationship: logical connection between two or more entity sets (association among entities)

Relationship (set): mathematical relation:

$$R = \{(e_1, e_2, \ldots, e_N) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_N \in E_N\}$$

where $(e_1, e_2, \ldots, e_N)$ is a relationship (n-ary tuple)

Relation: $r(R) \subseteq E_1 \times E_2 \times \ldots \times E_N \rightarrow$ table

Relation schema: $R(E_1, E_2, \ldots, E_N) \rightarrow$ table header

Types of relationship (based on degree – number of elements)

- one to one relationship
- one to many relationship
- many to many relationship
  - total participation
  - partial participation

FOREIGN KEY:
- PK of another entity set
- must exist in the other (constraint)

Keys:

SUPERKEY: set of attributes uniquely identifying the entity

CANDIDATE KEY: minimal set of attributes, forming an SK

PRIMARY KEY: Chosen CK.

Weak entity set: has not enough attributes for identification

→ joined by so-called weak one to many r. to a strong set.

→ discriminator: unique in context.

Generalization, specialization (ISA): the sub entity inherits all properties of its anchestor

- may or may not be a partition of the superset

Domain constraint: set of ~~all~~ values an attribute can take

NULL values and three-valued logic

NULL value: data is not known, not available (permission contr.) or has no meaning in the given context.

Arithmetic interpretation: operations are undefined

Logical operations:

TRUE — 1		AND:	min(...)
NULL — 1/2		OR:	max(...)
FALSE — 0			

Relational model: IMPLEMENTATION DESIGN

- transformation of the high-level design
- entity sets ⇒ tables, attributes ⇒ columns
- binary relationships ⇒
  - one-to-one: foreign key on any of the sides
  - one-to-many: foreign key on the many side
  - many-to-many: separate table with two foreign keys
- non-binary relationship ⇒ artificial (weak) entity set and multiple binary relations
- ISA relationship types:
  - condition-defined vs. user-defined
  - disjoint vs. overlapping
  - total or partial
  - aggregated

Relational algebra. Getting data from database

Relation schema: $R(A_1, A_2, \ldots, A_n)$ where A_n are attributes
Relation: $r(R) \subseteq D_1 \times D_2 \times \ldots \times D_n$
Relation instance: current values of a relation (table)
Keys (revision):
SK: K is SK of R if values of K identify a unique tuple of each possible relation in $r(R)$
CK: $\forall$ subset of CK does not identify + is an SK
Strong, weak and relationship entity set.

BASIC OPERATORS: SELECT, PROJECT, UNION, DIFF, PROD, RENAME

1. SELECT: $\sigma_p(r) = \{t \mid t \in r \text{ and } p(t) \text{ is true}\}$
   p: selection predicate, propositional calculus
2. PROJECT: $\Pi_{A_1, A_2, \ldots, A_k}(r)$: deleted columns, removed dupplicates
3. UNION: $r \cup s = \{t \mid t \in s \text{ or } t \in r\}$
   constraint: r, s compatible — same arity, convertible data type
4. DIFFERENCE: $r - s = \{t \mid t \in r \text{ and } t \notin s\}$
   constraint: r, s compatible
5. CARTESIAN: $r \times s = \{(t, q) \mid t \in r \text{ and } q \in s\}$
   Constraint: disjoint attributes
6. RENAME: $\rho_{X(B_1, B_2, \ldots, B_n)}(E) \longrightarrow$ returns $E(A_1, \ldots, A_n)$ under the name/schema $X(B_1, B_2, \ldots)$

ADDITIONAL OPERATORS

7. SET INTERSECT: $r \cap s = r - (r - s)$

8. NATURAL JOIN

$$r \bowtie s = \pi_{r.A_1, r.A_2, \ldots, r.A_n, s.B_{n+1}, s.B_{n+2}, \ldots s.B_k}(\sigma_{\ldots}(r \times s))$$

9. DIVISION

$R = (A_1, \ldots, A_n, B_1, \ldots, B_m)$

$S = (B_1, \ldots, B_m)$

$r \div s = \{t \mid t \in r \text{ and } \forall u \in s \; tu \in r\}$

$r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - r)$

10. ASSIGNMENT : convenience, nothing else

$\boxed{\text{EXTENDED OPERATIONS}}$

11. GENERALIZED PROJECTION : operations on attributes are also permitted (arithmetical, string)

12. AGGREGATE FUNCTIONS

$(G_1 G_2 \ldots G_n \; g \; F_1(A_1), F_2(A_2), \ldots, F_n(A_n))(E)$

$G_1, G_2, \ldots, G_n$: attributes on which to group

$F_1, F_2, \ldots, F_n$: operations (avg, min, max, sum, count)

$A_1, A_2, \ldots, A_n$: attributes

Convenience: renaming with "as" of $F(A_n)$ is permitted.

13. OUTER JOIN (~~inner~~, left, right, full)

- nonmatching ~~relatio~~ tuples filled with null values

$\boxed{\text{Manipulating data with relational algebra}}$

1. DELETE : $r \leftarrow r - E$
2. INSERT : $r \leftarrow r \cup E$
3. UPDATE : $r \leftarrow \pi_{F_1, F_2, \ldots, F_k}(r)$

Functional dependencies

Outline

- functional dependency is a semantic concept, used to define formal measures on the goodness of a relational design
  (semantic → defined by the world, being modeled)
- can be defined as a constraint on legal relation instances
- used to define normal forms

Formal definition

"Tuple function": $t: \{A_1, A_2, \dots, A_n\} \rightarrow D_{A_1} \cup D_{A_2} \cup \dots \cup D_{A_n}$

↳ example: $t[ID] =$ "Bus123". ← value in the tuple

Functional dependency:

$\alpha = \{A_{l_1}, A_{l_2}, \dots, A_{l_n}\}$ $\beta = \{A_{l_1}, A_{l_2}, \dots, A_{l_n}\}$

$\alpha \rightarrow \beta$ means that $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

Trivial dependency: satisfied by all tuples.

Partial dependency: attributes depend on part of the key, not all attributes of it. [this should be avoided]

Armstrong axioms

3. Transitivity : $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma \Rightarrow \alpha \rightarrow \gamma$
2. Augmentation : $\alpha \rightarrow \beta \Rightarrow \alpha\gamma \rightarrow \beta$
1. Reflexivity : $\alpha \rightarrow \beta$ if $\beta \subseteq \alpha$ Proof: trivial

Rules derived from this:

4. Pseudotransitivity : $\alpha \rightarrow \beta$ and $\beta\gamma \rightarrow \delta \Rightarrow \alpha\gamma \rightarrow \delta$
5. Union : $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma \Rightarrow \alpha \rightarrow \beta\gamma$
5. Decomposition : ~~a~~ $\alpha \rightarrow \beta\gamma \Rightarrow \alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$

$\boxed{\text{Derivation}}$ (bizonyítás, levezetés, bővítés – nem $\frac{d}{dx}$:P)

<u>Implication</u>: $FD_1 \models FD_2$ if every instance legal under FD_1 is legal under FD_2 as well

<u>Derivation</u>: az $FD_1, FD_2, \dots, FD_n$ sorozat az $\alpha \to \beta$ levezetése az FD függőségi halmazból, ha $FD_n = \alpha \to \beta$ (azaz előáll a cél) és $FD_k \in FD$ (már benne volt) vagy $\exists FD_i, i<k : FD_i \models FD_k$, az Armstrong axiómákkal bizonyíthatóan.

thu
↓
trans. needed

<u>Armstrong axioms</u>:

- sound: every FD obtained by derivation is consequence of the original one
- complete: if it is consequence, a derivation exists.

<u>Closure of FD set</u>:

F^+ contains all FD's implied by F. (derivable)

<u>Keys as FD's</u>:

$\beta \subseteq R$ is SUPERKEY, if $\beta \to R$

$\alpha \subseteq R$ is CANDIDATEKEY, if α is super, but $\nexists \gamma \subset \alpha$: $\gamma \to R$ is true.

<u>Closure of attribute set</u>: $\alpha^+ = \{A \mid \exists FD \in F^+ : FD = \alpha \to A\}$

thu "azok az attribútumok, amit α meghatároz"

"Algorithm": $\alpha^+(0) = \alpha$; $\alpha^+(i+1) = \alpha^+(i) \cup A$ if $\exists Y \in \alpha^+(i)$ $Y \to A$.

<u>Theorem</u>: $\alpha \to \beta$ is derivable from F iff $\beta \in \alpha^+$

"$\Rightarrow$" if $\beta \in \alpha^+$: algorithm constructively gives derivation

"$\Leftarrow$" if $\alpha \to \beta$ is derivable: by definition $\beta \in \alpha^+$

Normalization

<u>Normalization</u>: process of
a.) determining the normal form of a relation
b.) applying decomposition methods to achieve that.

<u>Anomalies</u> (of poorly designed systems):
a) update anomaly: caused by uncontrolled redundancy
b.) insert anomaly: not possible to stroe information unless another information is stored as well.
c) delete anomaly: cannot delete information without deleting another information as well.

<u>Normal forms</u> (shortly):

1NF: all attributes must have atomic domain

2NF: nonprime attribútum parciálisan nem függhet a CK-tól + 1NF teljesül

3NF: nonprime attr. nem függhet tranzitív módon CK-tól + 2NF

BCNF: az attribútumok csak SK-tól függhetnek

<u>Total dependency</u>: in the FD $\alpha \to \beta$ attribute set β is totally dependen on α if $\nexists \delta \subset \alpha : \delta \to \beta$

<u>Partial dependency</u>: in the FD $\alpha \to \beta$ attribute set β is partially dependent on α if $\exists \delta \subset \alpha : \delta \to \beta$

<u>Prime attribute</u>: attribute involved in (at least one) CK.

<u>Nonprime attribute</u>: no CK contains it.

$\boxed{\text{2NF}}$: all ~~maximal~~ nonprime attributes totally depend on any CK.

$\alpha \to \beta$ violates the 2NF, if β is nonprime, and $\exists \delta \subset \alpha : \delta \to \beta$.

Transitive dependency: γ depends transitively on a set of attribute α if $\exists \beta : \alpha \to \beta$ and $\beta \to \gamma$.

$\boxed{\text{3NF}}$:

DEF 1: A relation is in 3NF, if it is in 2NF and non-prime attributes do not depend ~~partially~~ transitively on any CK.

$\forall \alpha$ nonprime $\nexists \beta$ CK, γ : $\beta \to \gamma$ and $\gamma \to \alpha$.

DEF 2: A relation is in 3NF, if for each nontrivial FD in F^+ in form of $\alpha \to \beta$:

a) α is superkey OR:

b) β is prime attribute

Definition equivalency:

DEF 1 $\to$ DEF 2:

if $\alpha \to \beta$ violates DEF 2, ~~two~~ one possible case:

a) α not superkey, β nonprime

if α is not superkey, then:

i) $\alpha \subset$ CKey $\Rightarrow$ $\alpha \to \beta$ is partial dependency of nonprime attribute $\Rightarrow$ not 2NF

ii) $\alpha \not\subset$ CKey $\Rightarrow$ CKey $\to \alpha \to \beta$ is transitive dependency

DEF 2 $\mapsto$ DEF 1 :

if $\alpha \to \beta$ violates DEF 1 :

1.) $\alpha \to \beta$ violates 2NF, meaning β is nonprime and α is not a CK $\Rightarrow$ β nonprime, α not superkey

2.) $\alpha \to \beta$ violates transitivity, meaning

i.) $\beta \subset \alpha \Rightarrow$ β not superkey & β nonprime

ii) $\beta \not\subset \alpha \Rightarrow$ β not superkey (??)

BCNF: attributes may depend just on superkeys!

$\boxed{\text{Decomposition methods and conditions}}$

Decomposition of $R = (A_1, A_2, \ldots, A_n)$ is a set of relations $R_1(A_{11}, A_{12}, \ldots, A_{1i})$

$$R_2(A_{21}, A_{22}, \ldots, A_{2j})$$

$$\vdots$$

$$R_k(A_{k1}, A_{k2}, \ldots, A_{kn})$$

such that $\bigcup A_{ij} = \{A_1, A_2, \ldots, A_n\}$

and $r_k = \pi_{A_{k1}, A_{k2}, \ldots, A_{kn}}(r)$.

Lossy decomposition $r \subset r^* = r_1 \bowtie r_2$

Lossless decomposition.

Theorem: if $R_1, R_2, \ldots, R_k$ is a decomposition of R, then $r \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$ is true.

Definition: decomposition is lossless, if $r = r_1 \bowtie \ldots \bowtie r_k$

Theorem of lossless decomposition: decomposition is lossless iff at least one holds:

$$\left.\begin{array}{l} R_1 \cap R_2 \rightarrow R_1 \\ R_1 \cap R_2 \rightarrow R_2 \end{array}\right\}$$ the common attribute is SK in one of the parts

Lossless 2NF decomposition: if $\alpha \rightarrow \beta$ violates the 2NF, decompose to: $R_1(\alpha, \beta)$

$R_2(R - \beta)$

What ensures its really lossless: α is SK in R_1.

Lossless 3NF / BCNF decomposition: always exists

1. Find CK's for R
2. If $\alpha \rightarrow \beta$ in F^+ corresponds to 3NF / BCNF
3. Decompose relation. (??)

... details later...

Projection of FD sets:

$$\pi_{R_i}(F) = \{ X \rightarrow Y \mid X \rightarrow Y \in F^+ \text{ és } X \cup Y \subseteq R_i \}$$

Decomposition of schemas:

$S = (R; F)$ can be decomposed into

$S_1 = (R_1; F_1)$ $\quad S_2 = (R_2; F_2)$ ... $\quad S_k = (R_k, F_n)$

such that:

$$R = \bigcup_{i=1}^{k} R_i$$

$$F_i = \pi_{R_i}(F)$$

Equivalency of FD sets: E and F are equivalent iff $E^+ = F^+$, meaning $E \models F$ and $F \models E$.

Dependency preserving decomposition:

$S \rightarrow S_1, S_2, \ldots, S_k$ is dependency preserving, if F is equivalent to $\bigcup_{i=1}^{k} F_i$

Ullmans minimal cover (FD set)

1.) single attribute on the right side (decompos. rule)
2.) left-reduced: no partial dependency
3.) non-redundant: removing $X \rightarrow Y$ ruins equivalency

Lossless dependency-preserving 3NF decomposition

1.) group dependencies based on the left side
2.) decompose everything based on groups
3.) add relation for the candidate keys
4.) decomposition ok.

What ensures losslessness?

$\cancel{R_1}$ $R_1 \cap R_2 \rightarrow R_2$

What ensures dependency preservation: definition of the minimal cover.

Why does lossless dep. pres BCNF always exist?

R(City, Street, Zip code) $F = \{CS \rightarrow Z;\ Z \rightarrow C\}$

Nonprime: — Prime: C, S, Z.

1NF, 2NF, 3NF ok.

BCNF: Z is not a superkey.

$F^- = \{CS \rightarrow Z,\ Z \rightarrow C\}$

$R_1 = (Z, C)$ $Z \rightarrow C$ OK

$R_2 = (C, S)$ trivial

Database recovery systems

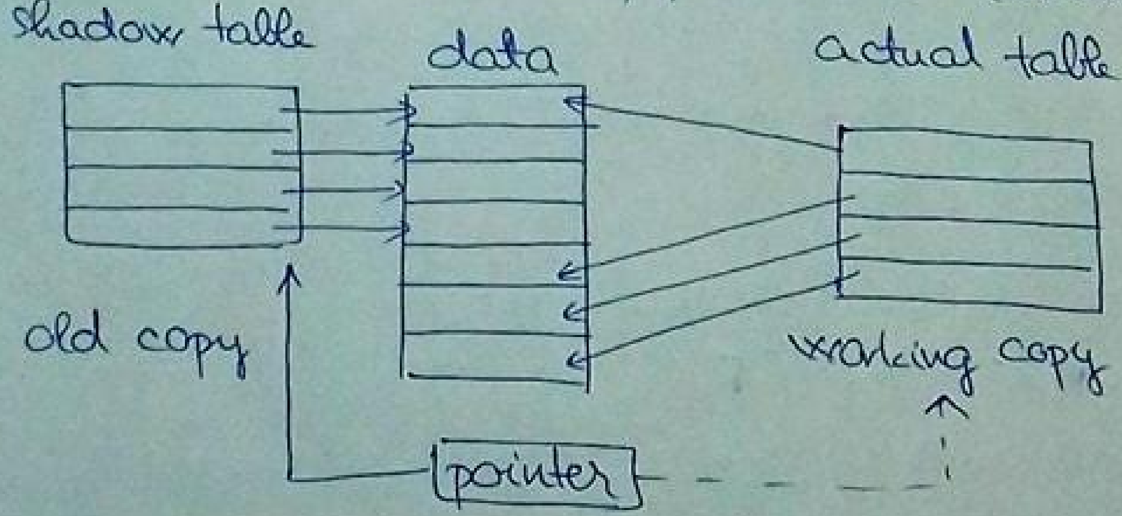
Aim : restore database to a known consistent state.

Properties of database :

Atomicity : transaction completes or does not touch data
Consistency :
isolation : } later in transaction manager
Durability : data is not lost in case of system failure (like hardware error, power loss, ...)

Types of recovery: LOG based or SHADOW based

SHADOW based : transactions work on a copy
on commit, pointer is set (atomic op).



LOG based :

Deferred modification model (~~UNDO~~ REDO) : uncommitted transactions are not allowed to touch database
Immediate modification (UNDO)
REDO / UNDO : no idea when storage is written
CHECKPOINT : data of committed transactions is safe

Cascading rollback caused by dirty reading
Solution : immediate modification
deferred + RIGOROUS 2-PHASE PROTOCOL