Luca Longinotti, 08-920-142

- Domain constraints (attribute domain)
- Key constraints (primary key)
- Entity constraints (not null)
- Referential integrity (foreign key)

select: σ_p, predicate $\wedge \vee \neg$

project π_a, keep only columns a

union $\cup$, same schema, union of rows

set difference $-$, union-compatible, row difference

cartesian product $\times$, disjoint attribute names, else rename!

rename $\rho_r(A,\ldots)$; rename rel. + attr.

set intersection $\cap$, equal to $r-(r-s)$

theta join $\bowtie_\theta$, θ is bool condition, joins all attrs. from both schemes

natural join $\bowtie$, all attrs. of R + new ones in S

division $\div$, "for all", R÷S

assignment $\leftarrow$, temp. variables

aggregate functions: $\mathcal{G}_{\text{group}}\ \theta\ F_{(A)}\ (E)$ (fcns, table)

outer join, join with no info loss (use null)

- Internal Schema — Physical DI
- Conceptual Schema — Logical DI
- External Schema

insertion $r \leftarrow r \cup E$

updating $r \leftarrow E$, $r \leftarrow \pi_{F_1,F_2,\ldots}(r)$

deletion $r \leftarrow r - E$

TRC / DRC | tables, columns, rows

PRIMARY KEY(X)
FOREIGN KEY (x) REFERENCES tab(Y)
LIKE % (0-n) or _ (1)
SELECT FROM WHERE GROUP HAVING (on groups)
AS to rename
FROM t_1 JOIN t_2 ON cond
NOT / AND / OR, GROUP BY
SELECT DISTINCT, *
UNION / INTERSECT / EXCEPT
IN, EXISTS, NOT EXISTS
UNIQUE, IS NULL, ORDER BY ASC/DESC
INSERT INTO x VALUES (...)
UPDATE x SET ..., CASE when then else END

Views: CREATE VIEW x AS ...
WITH clause for temporary views

CREATE DOMAIN, CAST x AS type

CREATE FUNCTION x(y)
RETURNS type AS $$
[DECLARE]
BEGIN stmt END;
$$ language plpgsql;

CREATE TRIGGER x AFTER/BEFORE INS/UPD/DEL ON table

- simplicity
- no redundancy
- no update anomalies
- avoid null values
- avoid spurious tuples

⇒ Functional Dependencies → minimal

Rules: $Y \subseteq X \Rightarrow X \to Y$
$X \to Y \Rightarrow XZ \to YZ$
$X \to Y,\ Y \to Z \Rightarrow X \to Z$

1NF: no composite, multival, nested
2NF: partial functionally dependent
3NF: BCNF + contained in candidate key
BCNF: trival or superkey

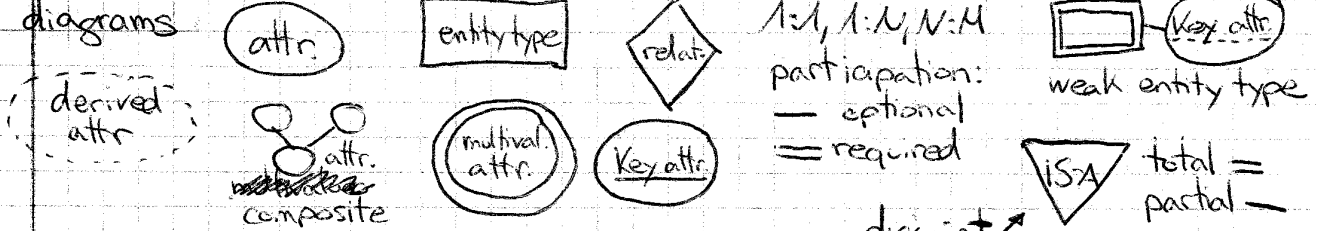
Join Evaluation

- nested loop
- block nested loop
- indexed nested loop
- merge join
- hash join (only equi)

cost-based → statistics

heuristical →
- selection early
- projection early
- most restrictive select/joins before others

ER diagrams



ER-to-Relational

1) Regular Entity types (PRIMARY KEYs)
2) Weak Entity types (FOREIGN KEYs)
3) 1:1 Relations (FOREIGN KEY)
4) 1:N Relations (FOREIGN KEY)
5) N:M Relations (rel. relation, 2 FOREIGN KEY)
6) Multivalued Attributes (relation, FOREIGN KEY)
7) N-ary Relationships (rel. relation, N FOREIGN KEY)
8) Specialization or Generalization

Static/Dynamic/Extendable Hashing

B+-tree multi-level index

- path length $\log_{\frac{m}{2}}$ (Keys)
- node: m-1 keys, m pointers
- nodes half to full
- internal nodes: between m/2 and m
- leaf nodes: (m-1)/2 and m-1 children
- root at least 2 children (or leaf)
- leaf nodes linked together (last ptr.)
- insertion: split
- deletion: coalesce, redistribute

Transaction States:

- active
- partially committed
- committed
- failed
- aborted (⇒ kill/restart)

ACID properties:

- Atomicity
- Consistency
- Isolation
- Durability

Schedule: seq. of instructions from a set of concurrent transactions

Serial schedule: sequential transactions

Serializable schedule: if equivalent to serial schedule

Conflict equivalence: transform into another using only non-conflicting swaps

Conflict serializable schedule: is conflict equivalent to a serial schedule

⇒ conflict if there is a **write!**

⇒ swaps can be made:
- both are reads
- they refer to different data
- one of them is not a Read

Concurrency control

Schedule: - serializable
- recoverable
- cascadeless

- Lock-based (X/S-locks)
  → ⚠ deadlocks, non-serializable
  - 2PL (obtain, release)
    - strict (X-only) } enf.
    - rigorous (both) }
  - undesirable phenomena
    - dirty read
    - nonrepeatable read
    - phantom read
  - ⇒ isolation levels
    - read uncommitted
    - read committed
    - repeatable read
    - serializable

Precedence Graph (if acyclic, conflict serializable)

A directed graph with a node T_i for each transaction and an edge $T_i \to T_j$ if:
- T_i executes write(Q) before T_j executes read(Q)
- T_i executes read(Q) before T_j executes write(Q)
- T_i executes write(Q) before T_j executes write(Q)