­ SUBCLASS

1. one table for each subclass w/ all attr. plus key from superclass

2. one big relation w/ all att. w/ null values for missing attributes

3. one table for every subtree (including root) w/ all attributes + all inherited attr.

**Normalization Theory**

Anomaly

1. UPDATE: only some info updated

- need to update all tuples

2. INSERTION: some info can't be represented

3. DELETION: info deletion may delete others

Functional Dependency

- some attributes determined by other attr.

- "no 2 tuples in R can have the same X values but different Y values"

Trivial FD

- when

Non-Trivial FD

- when

Completely Non-Trivial FD

- with no overlap b/t X and Y

Logical Implications

-

Closure of FD

- set of all FD's that are logically implied

Closure of Attribute Set

- set of all attr that are FD by X

FD and Key

- X is a Key of R iff

1. 2. X is minimal

Lossless-Join Decomposition

-

- is lossless IFF .

**Boyce-Codd Normal Form (BCNF)**

- *R* is in BCNF w/ regard to *F*, iff for every non-trivial , *X* contains a key

**Multi-Valued Dependency & 4NF**

Multivalued Dependency

- if u[X] = v[X], then there exists w such that:

1. w[X] = u[X] = v[X]

2. w[Y] = u[Y]

3. w[Z] = v[Z], Z is all att in R except X,Y

Complementation Rule

- X ->> Y, then X ->> Z where Z is all attr in R, except X and Y (swap Y is same as swap Z)

Trivial MVD

- X->>Y if

**Extendible Hashing**

- use *i* of *b* bits output by has func

- use directory that maintains points to hash buckets (indirection)

- no min. space guarantee (could waste)

Deletion

- no merging/ merging + shrink if need

Merging Condition

- bucket *i*'s are the same

- first *i-1* bits of hash key are same

Directory Shrink Condition

- all buckets *i*’s are smaller than dir *i*

Summary

- static: overflow + chaining

- extendible: can handle growing files

- indirection, dir doubles in size

**Join Algorithms**

|  |  |
| --- | --- |
| NLJ |  |
| SMJ | (already sorted) |
| Hash | or |
| Index |  |

Nested Loop Join

for each r in R do

for each s in S do

if r.C = s.C then output r, s pair

Index Join

for each r in R do

X <- index-lookup(S.C, r.C)

for each s in X do

output (r,s)

- lookup index to find matching tuples from S

Sort-Merge Join

idea: if tables have been sorted by join attribute, scan each table only once

1. if R & S not sorted, sort

2. i <- 1; j <- 1;

while (i <= |R|) AND (j <= |S|) do

if R[i].C=S[j].C then output tuples

else if R[i].C > S[j].C then j++

else if R[i].C < S[j].C then i++

*Cost of join stage*

1 avail. mem. buffers. disk blocks of tabl

2. need to read R table, S table and write

3. seq read R and S blocks 1 at a time

Hash Join

idea: if hash values are different, tuples will never join

join 2 tuples if hash values are same

**Entity-Relationship Model**

Terms

- entity: ‘thing’ or ‘object’ in real world

- entity set: a set of entities (object)

- rectangle in E/R

- attribute: property of entities “field”

- ellipsis in E/R

- key: set of attributes that uniquely identify an entity

- underline in E/R

- relationship: connection between entities

- "" set: set of relations of same kind

- diamond in E/R

- connections between entities

- can have ATTRIBUTES

- (add grade to Take, quarter to Teach)

Cardinality of Relationships

1. one-to-one

- each entity in E1 related to at most 1 in E2

- and vise-versa

- arrow at "one" end

2. many-to-one

- entity in E1 at most 1 entity in E2

- arrow at "one" end, line at other

3. many-to-many

- 0 or more entities in E2/ vice versa

- no arrow

4. total participation

- AT LEAST ONCE

- double lines

General Cardinality Notation

- l..h on an edge

- object participate in a 'ship l to h times

- \* means unlimited

- 0..\* corresponds to one, 0..1 to many

N-ary Relationship

- arrow in N-ary relationship: pick one entity from every other set w/o arrow

- these must be related to at most 1 entity with arrow

- do not put mult. arrows for non-binary relationship

Roles

- labels on edge in E/R

Subclasses

- similar to inheritance; ISA relationship

- generalization: subclass -> superclass

- specialization: superclass -> subclass

- subclass inherits all att. of superclass

- subclass participates in 'ships of super

- subclass may participate in own 'ship

Weak Entity Set

- entity sets without unique keys

- double rectangle/diamond in E/R

- discriminator: set of att. in WES that are part of the key

- dashed underline in E/R

Owner Entity Set

- entity set providing a part of the key

Identifying Relationship

- 'ship between a WES and OES

- must be Total participation

E/R to Relation

- entity set: one table w/ all attributes

- relationship set: one table w/ keys from linked ES and its own attributes

- weak entity set: 1 table w/ own att. and keys from owner ES

- subclass:

1.

|  |  |  |  |
| --- | --- | --- | --- |
| iso level | dirty read | nonrepeatable read | phantom |
| read uncommitted | Y | Y | Y |
| read committed | N | Y | Y |
| repeatable read | N | N | Y |
| serializable | N | N | N |

**Transaction**

ACID

1. atomicity

- "all-or-nothing" of each transaction executed

2. consistency

- consistent before, consistent after

3. isolation

- w/ multiple transactions, result is same as executing them in some sequential order

4. durability

- all changes remain permanently

Declaring in SQL

COMMIT - all changes is stored permanently

ROLLBACK - undo all changes by transaction

Autocommit

- ORACLE: SET AUTOCOMMIT ON/OFF

- MySQL: SET AUTOCOMMIT = {0|1}

SQL Isolation Levels

- may not need full ACID

- DIRTY READ = read before committed

- NON-REPEATABLE READ = when reads same row multiple times, may get diff values

- SELECT

- PHANTOM: when new tuples are inserted, once some of them are seen by statements, or only some statements see the newly inserted tuples

- INSERT

Mixed Isolation Levels

- only when all transactions are serializable, we guarantee ACID

- isolation level is in the eye of beholding trans.

SQL Isolation Level:

MySQL, DB2 = repeatable red

Oracle = read committed