# 1 Relational algebra

**Relational data model**

* Relation
  + Header
* Schema
* Set of tuples
* Attribute
  + Domain
  + Null
* Keys
  + Super
  + Candidate
  + Foreign

**Operators**

* Select (restrict)
* Project PI ***(dømi)***
* Cartesian product X
* Rename p
* Union u

**Joins**

* Join is a cartesian producdt + a selection + a projection
* Theta join
* Equi join (special form of theta join
* Natural join (relations have common attributes)
* Semijoin (left/right)
  + Natural join with only left/right header

# 2 SQL

SQL’s data model

* Tables
* Rows
* Columns
* Types

DDL

* Create table ***(dømi Person)***
  + Name
  + Columns
    - Name
    - Type
    - constraints
  + Table constraints
    - PK
    - FK
* Create View
* Create index
* Drop table
* Alter table
  + add column
  + remove column

DML

* Select
* Update
* Delete
* Insert

**Subquery**

* Result of a query is a table -> relate to relational algebra
* Queries can be used in queries instead of table names
* Subquery in WHERE part ... operators

**Operators**

* any x < any [resulting table]
* all x < all [resulting table]
* in x in [resulting table]
* exists [resulting table].rows = 0 ? true : false

NULL

Views

# 3 ER modelling

What is ER model? W respect to databases?

Database = Collection of entities and relationships between them

ER model components

* Entity
  + Thing/object
  + Can be distinguished from other entities
  + Nouns
  + Have attributes
  + Weak entities (no key)
* Relationship
  + Can have attributes
  + Cardinality
    - 1:1
    - 1:m
    - m:n
  + Binary
  + Ternary
  + Participation
* Attribute
  + Stored vs. derived

Keys

* identifies an entity in the set

ER diagrams to tables ***dømi úr uppg***.

* 1 table pr. regular entity
* Weak entities merged to tables or get same key as strong entity
* 1:1 relations, extend one table with pk of the other
* 1:N
  + Extend one table
  + Create a new table for the relationship
* m:n
* Multi valued attributes

# 4 Integrity Constraints

Constraints

* Table/Column (Henvisning: mini projekt)
  + Unique
  + Primary key
  + Foreign key
  + Check
    - Address IN (Select address from...)
* Referential
  + Foreign key ***(vís dømi) (Suppliers and parts)***
    - On delete (cascade)
    - On update (Set null)
* Domain
* Assertion

Consistency not always best solution: tagging, very large databases (ebay)

**Suppliers**

|  |  |  |
| --- | --- | --- |
| **SNO** | **Name** | **City** |
| S1 | Smith | Paris |
| S2 | Blake | London |
| S3 | Adams | London |

**Parts**

|  |  |  |
| --- | --- | --- |
| PNO | Name | SNO |
| P1 | Screw | S1 |
| P2 | Bolt | S2 |

Update S1 -> S8

Delete S1

# Topic 5 Normalization

Redundant information

***Suppliers and parts***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SNO | SName | SCity | PNO | PName | PCity |
| S1 | Blake | London | P2 | Screw | Oslo |
| S2 | Clark | Paris | P2 | Screw | Paris |
| S2 | Clark | Paris | P4 | Bolt | Paris |
|  |  |  |  |  |  |

* Waste of space
* Update anomalies ***(dømi)***
  + Inconsistency
* Insert anomalies
* Delete anomalies

**Functional dependency**

{a} -> {b}

All tuples with the same a, also have the same b

Trivial if b is a subset of a

a is a candidate key if a -> R

**1NF**

Atomic values (non 1NF : Price table with part and price in same column)

**BCNF** X -> A

If A is part of X (trivial dependency)

If all attributes are functionally dependent on X

Example: R=(A, B, C)

F=(A->B, B->C)

R is not in BCNF, can be decomposed

R1 = (A, B) F=(A -> B)

R2 = (B, C) F=(B->C)

**3NF** Like BCNF

But allows A to be part of a candidate key

( Closure: F+ = all dependencies implied by F )

F = {a -> b, a -> c, cd -> a}

a -> bc

cd -> b cd -> a -> b (Transitive)

# Topic 6 Physical database design

Database = set of data files  
File = set of records  
Types of storage

* Disk (Slow)
* Main memory
* Cache (Fast)

I/O in blocks

Data file organization ***(dømi)***

* heap (unordered),
* sequential, (Ordered)
* hashing, (Record hashed to a block)
* clustering (Compute a key, similar keys stored close to each other)

Indexes

* Single level:
  + primary,
  + clustering,
  + secondary
* B-Plus tree index
* hash index

# 7 Query processing and optimization

**Evaluation strategies for**

* selections
  + Linear search
  + Binary search
    - Requires ordered file
  + Clustering index search
    - Block of records
  + Secondary index search, dense pointers to a single record
* joins
  + Nested loop join
    - Nested foreach loop
  + Index based join
    - At least one index on a join attribute
  + Sort merge join
    - Used for equi- and natural joins
    - 1. sort relations on join attributes
    - Has a pointer in each table
  + (Hash join)
    - Hash on the join attributes
    - Join the hashed values (buckets)

**External sorting**

* For relations too large for memory
* Sort in pieces
  + The relation gets split into multiple sorted pieces
* Merge the pieces to larger sorted pieces

**Query optimization**

* Heuristic
  + Decompose selects
  + Select (restrict) as soon as possible
  + Most restrictive selects first
  + Most restrictive joins first
  + Project early
* cost based
  + Statistics to estimate cost
  + Multiple query trees evaluated based on the statistics

Measures of cost

# 8 Transaction concept

Transaction: Unit of execution, group of commands

* Commit
* Rollback

***Transfer 10 kr. from account A to B***

***Read a  
a = a – 10  
write a  
read b  
b=b-10  
write b***

**ACID**

* Atomicity (log)
  + All or none operations reflected in the database
* Isolation (locks)
  + A transaction does not see the effects of another transactin untill it has committed
  + Necessary temproary inconsistencies are localized to each transaction
* Consistency
  + Consistency is preserved by isolation
* Durability (log)
  + Committed transactions persist

**Transactions run concurrently -> schedules**

**Serializability**

Schedules as if transactions ran in series

Results and

* conflicts
  + T1 write A, T2 read A
  + T1 read A, T2 wirte A
  + T1 wrte A, T2 read A
  + Ti ant Tj conflict only if there exists some item X accessed by both, and at least one wrote X
  + Conflict graph

Conflict equivalent schedules, contain no cycles in conflict graph

Equivealent to some serial schedule

**Recoverability**

**Ti Tj**  
read a  
write a  
 read a  
read b

Tj has to roll back if Ti fails, Tj has read a non consistent value

* Cascading rollback

Ti should commit before Tj reads

# 9 Concurrency control

Isolation

Protocols

Deadlocks

# 10 Recovery

When something goes wrong ***(dømi)***

* Logical error
  + Bad input
  + Owerflow
* System error
  + Deadlock
* System crash
  + Power outage
    - Volatile data lost
* Disk failure
  + Non volatile data lost

-------|

|--------|

|--------------- <crash>

**Atomicity**

* Operations/transactions complete fully or not at all

**Durability** of transactions

**Logging**

* Write every operation to a log
  + <T starts>
  + <T, X, 100, 90> <Transactionname, Data item name, old value, new value>
  + Then actually change the value
  + <T commits>

**Recovery algorithms**

* Undo/Redo
  + Undo
    - <start> no <commit>
    - not finished
  + Redo
    - <start> and <commit>
* No-undo/redo
  + Values written to disk after the transaction commits
  + nothing to undo
* Undo/no redo
  + Output all data before commit
* No-undo/no-redo
  + No-undo
    - Dont change the database during transaction
  + No-redo
    - On commit write all changes to database in a single action

**Checkpoints**

* Output log buffers
* Force database buffers to disk
* Output <checkpoint> to log
* Redo finished transactions after the checkpoint
* Undo not finished transactions after the checkpoint