# ER Diagram

### Many to Many

A can have multiple B and B can have multiple A



### One to Many

A can have multiple B but B can only have one A



### One to One

A can have only one B and B can have only one B



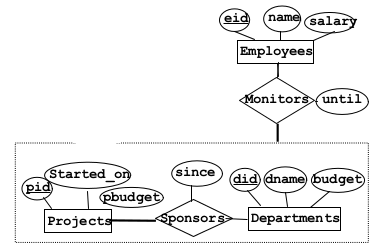
### At least one

Bold arrow specify there is at least one element.

|  |  |
| --- | --- |
|  | A have at least one |
|  | A have exactly one |

### Aggregation

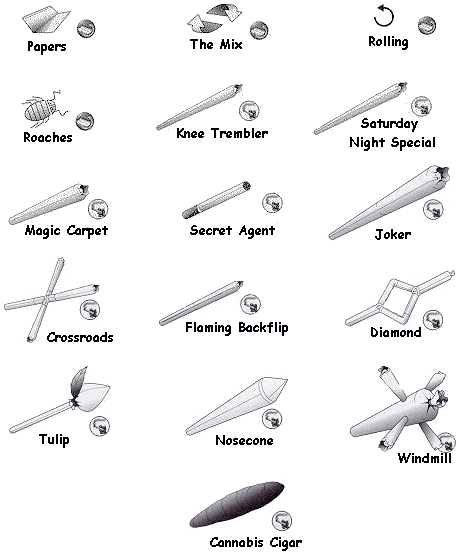
Allows us to treat a reltionship set R as an entity set so that R can participate in other relationships



# Relational algebra

|  |  |
| --- | --- |
| Selection |  |
| Projection |  |
| Renaming |  |
| Cross product |  |
| Join |  |
| Division |  |
| Intersection |  |
| Union |  |
| Set different |  |

## Type of joins



* Condition/Theta Join
* Equi Join: in2 Condition join where condition contains ONLY equalities
* Natural Join: Equijoin on all common attribute

# Sql

## Datatype

|  |  |
| --- | --- |
| Char(n) | A character string of fixed length n |
| VarChar(n) | Denotes a string of up to n charaters |
| INT or INTEGER | An integer |
| SHORTINT | Smaller integer |
| FLOAT or REAL | Float number |
| DOUBLE PRECISION | Double |
| DATE | Date format YYYY-MM-DD |
| TIME | Time format: hh:mm:ss |
| Bit strings |  |

## Table operations

|  |
| --- |
| --Create table  CREATE TABLE Students  (  id INT NOT NULL,  name VARCHAR(20),  login CHAR(10),  major VARCHAR(20) DEFAULT 'undefined',  school\_id INT,  PRIMARY KEY(id),  FOREIGN KEY(school\_id) REFERENCES School(id)  )  --Drop table  DROP TABLE Students  --Alter table  ALTER TABLE Students ADD COLUMN firstyear:integer |

## Row operation

|  |
| --- |
| --INSERT  INSERT INTO Students (id, name, faculty) VALUES (8908998, 'Dupont', 'Science')  --Delete  DELETE FROM Students WHERE id = 0894984  --Update  UPDATE Students SET faculty = 'Arts' WHERE id = 9849849 |

# XML

WHAT DAFUQ?!!@#!@#!?

## DTD

|  |
| --- |
| <!DOCTYPE people[  <!ELEMENT people(person\*)>  <!ELEEMNT person(name\*, (lastname|familyname)?)>  <!ATTLIST person PID ID #REQUIRED  age CDATA #IMPLIED  children IDREFS #IMPLIED  mother IDREF #IMPLIED  >  <!ELEMENT name(#PCDATA)>  <!ELEMENT lastname(#PCDATA)>  <!ELEMENT familyname (#PCDATA)>  ]> |

Data types: PCDATA (parsed character data) or CDATA (unparsed)

Attributes

* ID unique identifier (similar to primary key)
* IDREF: reference to single ID
* IDREFS: space-seperated list of references

Values

* can give a default value
* #REQUIRED must exist
* #IMPLIED optional

Specified in an XML file with <!DOCTYPE name SYSTEM "path/to/thing.dtd">

Can use regex style things too. \* is 0 or more. + is 1 or more, (a | b)? is one or the other

## XPATH

## /bibliography/book/author all author elements by root navigating through those elements

## /bibliography/book/@ISBN All ISBN attributes

## //title all title elements anywhere in the document

## /bibliography/\*/title titles of bibliography entries assuming that there could be books, journals, reports, etc...

## /bibliography/book[@year>1995] returns books where the year > 1995

## /bibliography/book[author='FooBar']/@Year returns the years of books written by FooBar

## /bibliography/book[count(author) <2]

* /bibliography/book/author[position()=1]/name position is the location of the node in the node set

## XQuery

For

|  |
| --- |
| for $b in document("bib.xml")/bib/book  return <result> $b</result> |

Returns

|  |
| --- |
| <result><book>…</book></result>  <result><book>…</book></result>  <result><book>…</book></result>  …  <result><book>…</book></result> |

Let

|  |
| --- |
| let $b in document("bib.xml")/bib/book  return <result> $b</result> |

Returns

|  |
| --- |
| <result>  <book>…</book>  <book>…</book>  ..  <book>…</book>  </result> |

## XML in DB@^%$^$#

|  |
| --- |
| INSERT INTO MyXML(id, INFO) VALUES (1000,  '<customerinfo cid="1000">  <name>Kathy Jones</name>  <addr country =Canada">  <street>123 fake</street>  <city>Ottawa</city>  <prov-state>Ontario</prov-state>  <pcode-zip>H0H 0H0</pcode-zip>  </addr>  </customerinfo>') |

# Indexing

## Linked, unordered list of all pages of the file

# scan retriving all records = good because all pages retrieved anyway

# equality search on primary key = bad have to read ~1/2 of pages

# range search = bad

# insert = good, low cost

# delete/update same as search

## Sorted Filesedit

# Records ordered according to one or more attributes of the relation

# scan retriving all records = good

# equality search on sort attribute = good about log2(number of pages)

# range search = good find first one, then scan along

# insert = bad, have to find proper page, need to reorganize

# delete/update same as search. update may be expensive

* sorted output: good if on sorted attribute

Let suppose we have a relation R (A, B, C, D, F) such that:

* A and B are int (6 byte)
* C-F are char [40] (10 byte per char).

Tuple = 172 bytes. 200,000 tuples

Each data page has 4000 bytes and is around 80% full.

.

B values are uniformly distributed

Non-clustered index B-tree with <k, list of rid>

Index pages are 4K and between 50%-100% fulll

rid = 10 bytes

size of pointer in intermediate page = 8 bytes

Index entry size in root/intermediate pages = size(key) + size(pointer) 6 + 8 = 14 bytes

Average number of rids per data entry = number of uples / different values (if uniform). eg: 200,000/20,000 = 10

Average length per data entry = size of key + (number of rids \* size of rid). eg: 6 + 10\*10 = 106

Average number of date entrier per page = fill-rate \* page size / length of entry. eg 0.75\*5000/106 = 28 entries per page

Estimate number of leaf page = number of different values / number of entrier per page. eg: 20,000 / 28 = 715

Number of entries in intermediate pages = fill-rate \* page size / lneght of index enty. eg: 0.5 \* 4000 / 15 = 143, 1\*4000/14 = 285

Height of tree = Number of leaf pages / (min | max)? number of entries in intermediate pages