SQL

SQL

- SELECT-FROM-WHERE
- DISTINCT, ORDER BY, using AS to rename attributes
- INSERT, DELETE, UPDATE
- GROUP BY, HAVING, select only attributes in the GROUP BY clause or attributes in aggregate functions
- LEFT OUTER JOIN, RIGHT OUTER JOIN, etc., handling NULL
- WHERE EXISTS, IN, >= ALL, NOT EXISTS, etc.
- Finding witnesses

SQL

```
CREATE TABLE Class (
    dept VARCHAR(6),
    number INTEGER,
    title VARCHAR(75),
    PRIMARY KEY (dept, number)
);
```

Indexes
- Clustered vs. unclustered

Views
- Materialized: precomputed and stored on disk
- Virtual: recomputed when needed

Relational Algebra

- Union ∪, intersection ∩, difference –
- Selection σ
- Projection Π
- Cartesian product ×, join ⋈
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ

Query Plans

- Know basic ways of implementing query plans, particularly joins (nested loop, hash-join)

- Know how to compute costs of query plans (B(R), T(R), V(R,a), M)
  - B(R) - # of blocks for relation
  - T(R) - # of tuples
  - V(R, a) - # of distinct values of attribute a
  - M - # of memory pages

XML

Document Type Definitions (DTD):

```
    <!ELEMENT result (country)>
    <!ELEMENT country (name, city+)>
    <!ELEMENT city (name)>
    <!ELEMENT name (#PCDATA)>
```

- a+ = one or more
- a* = zero or more
- a? = zero or one
- (a|b) = a or b

XPath

bib   **matches a bib element**

***** **matches any element**

/ **matches the root element**

/bib **matches a bib element under root**

bib/paper **matches a paper in bib**

bib//paper **matches a paper in bib, at any depth**

//paper   **matches a paper at any depth**

paper|book **matches a paper or a book**

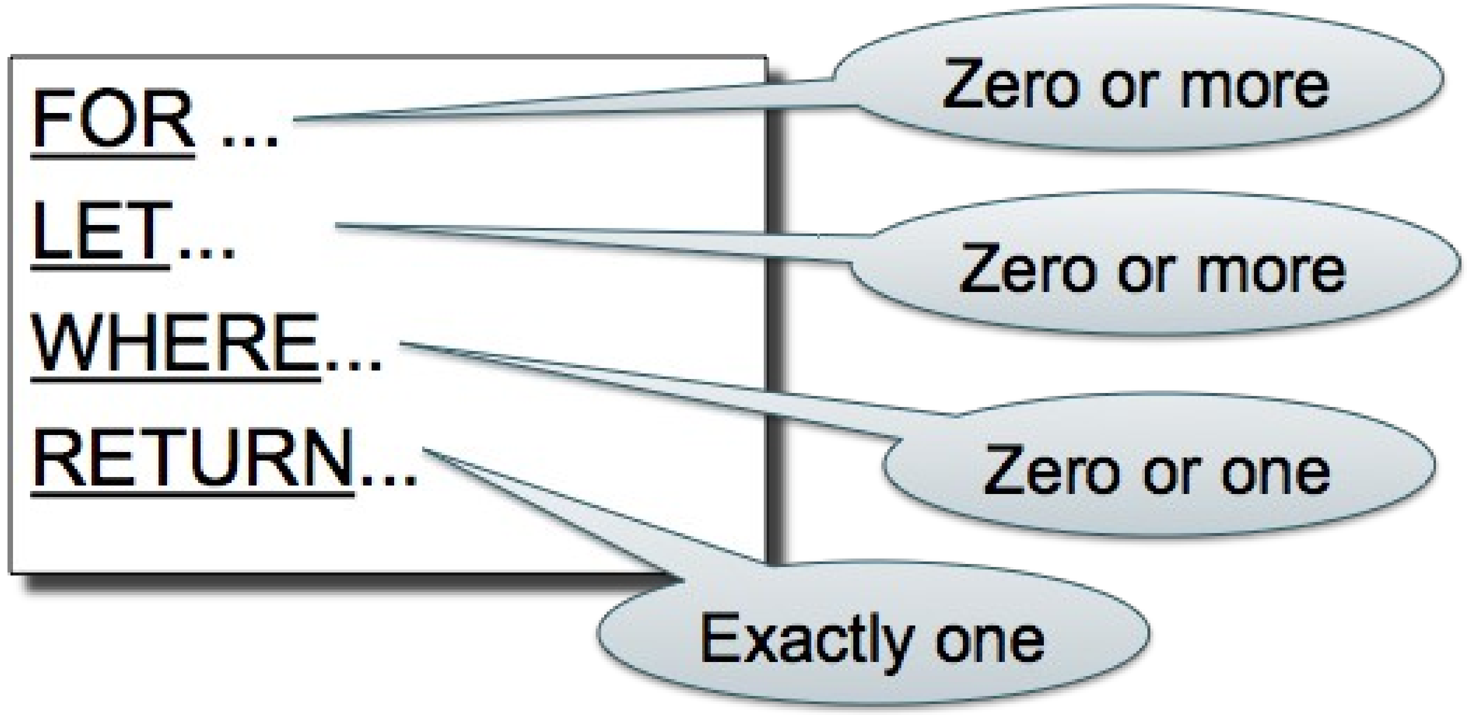
@price   **matches a price attribute**

bib/book/@price **matches price attribute in book, in bib**

bib/book[@price<"55"]/author/last-name **matches…**

bib/book[@price<"55" or @price>"99"]/author/last-name **matches…**

Xquery


FOR ...
LET...
WHERE...
RETURN...
Zero or more
Zero or more
Zero or one
Exactly one

Database Design

E/R diagrams:

- Entities, attributes
- Relationships:
  - Many-many, many-one, one-one, exactly one
  - Multi-way relationships
- Inheritance, weak entity sets, union types
- Constraints in E/R diagrams
- Translation to relations

Conceptual Design

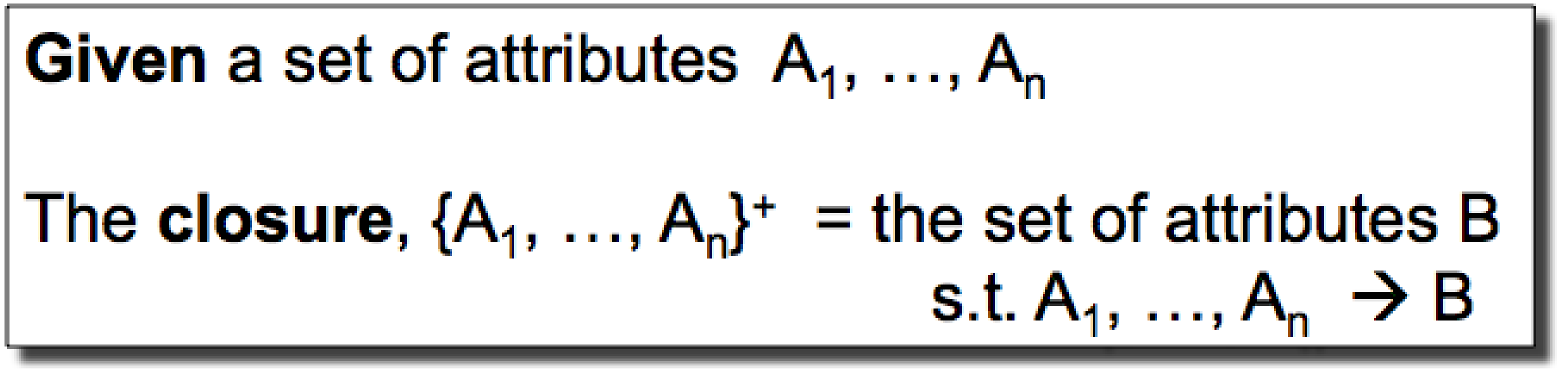
- Data anomalies
- Functional dependencies
  - Definition
  - Make sure you can check if a table satisfies a set of FDs
- Attribute closure
- Keys and Super keys
- Definition of BCNF
- Decomposition to BCNF

Functional Dependency

A1 -> A2:

If two tuples agree on the attribute A1

Then they must also agree on the attribute A2



Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

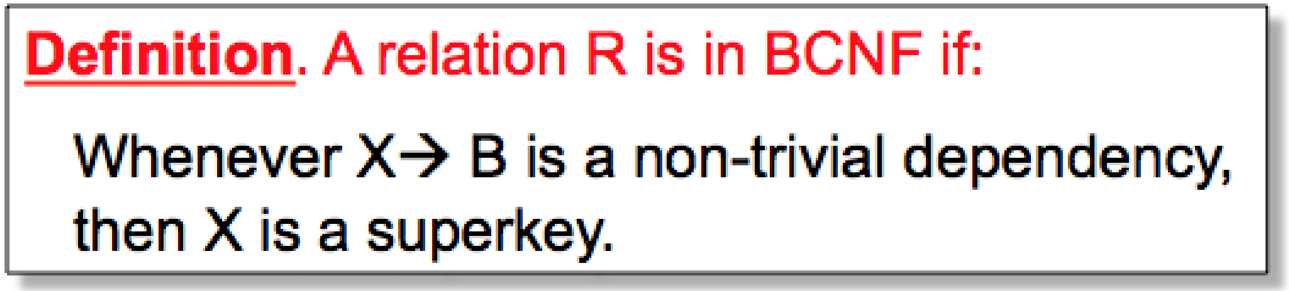
Superkeys

For all sets X, compute X+

- If X+ = [all attributes], then X is a superkey
- Keys are minimal superkeys

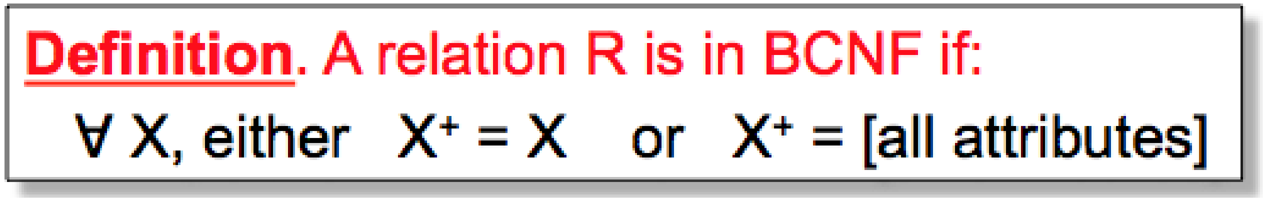
Boyce-Codd Normal Form

There are no "bad" FDs:


Definition. A relation R is in BCNF if:

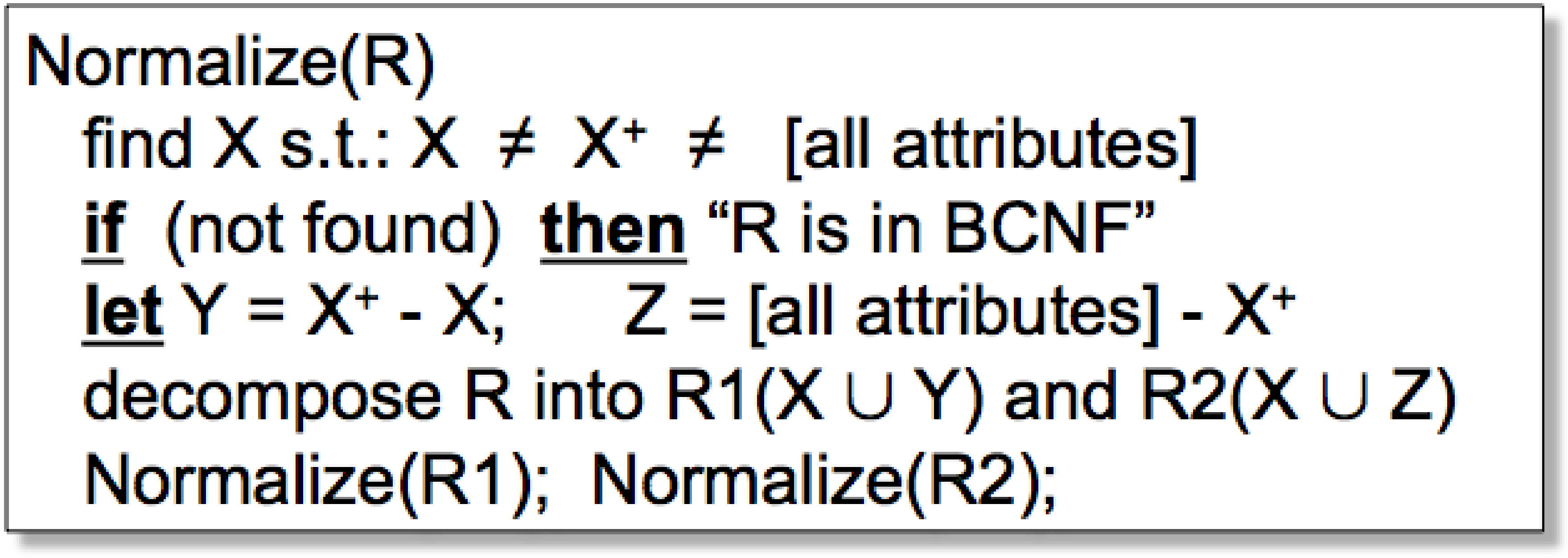
Whenever $X \rightarrow B$ is a non-trivial dependency, then X is a superkey.

Equivalently:


Definition. A relation R is in BCNF if:

$\forall$ X, either $X^{+} = X$ or $X^{+} =$ [all attributes]

BCNF Decomposition Algorithm



```
Normalize(R)
    find X s.t.: $X \neq X^+ \neq$ [all attributes]
    **if** (not found) **then** “R is in BCNF”
    **let** $Y = X^+ - X$;      $Z =$ [all attributes] $- X^+$
    decompose R into $R1(X \cup Y)$ and $R2(X \cup Z)$
    Normalize(R1);  Normalize(R2);
```

Transactions

A DBMS guarantees the following four properties of transactions:

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

Serial / Serializable

- A *serial* schedule is one in which transactions are executed one after the other, in (some) sequetial order

- A schedule is *serializable* if it is equivalent to a serial schedule

Conflicts

Conflicts: pair of actions (in order) in schedule such that if swapped, then behavior changes.

Two actions by same transaction Ti:	ri(X); wi(Y)
Two writes by Ti, Tj to same element	wi(X); wj(X)
Read/write by Ti, Tj to same element	wi(X); rj(X)
	ri(X); wj(X)

Note: any # of actions can appear between them

Conflict Serializability

- A schedule is *conflict serializable* if it can be transformed into a serial schedule by a series of swaps of adjacent non-conflicting actions
- Stronger condition than serializability
- How do we check for conflict serializability?
  - Using Precedence Graph

Locking

- Two Phase Locking (2PL): All lock requests precede all unlock requests

- Strict Two-Phase Locking (strict 2PL): All locks are held until the transaction commits or aborts. This guarantees that the schedule is recoverable (all transactions can be undone).

Isolation Levels in SQL

1. “Dirty reads”

   `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`

2. “Committed reads”

   `SET TRANSACTION ISOLATION LEVEL READ COMMITTED`

3. “Repeatable reads”

   `SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`

4. Serializable transactions

   `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`

Parallel Data Processing

- Speedup, scaleup
- Shared memory, shared disk, shared nothing
- Horizontal data partition: block, hash, range
- How to implement simple algorithms: group-by, join
- How to execute a complete query in parallel

MapReduce

- Functions: map, (combine,) reduce
- Terminology: chunk, map job / reduce job; map task / reduce task; server (instance); failed server
- Basic implementation of MR
- Dealing with server failures and stragglers

You will not be asked to write detailed Pig Latin code, but should have some basic understanding of how queries are implemented over MapReduce