Last Class

- Relational Database
  - Relation
  - Relation schema
  - Relation operator
  - Relation property
  - Relation language
    - SQL / Relational Algebra / Relational Calculus
  - Relation formal definition

Chapter 4 SQL

Related to text book chapter 4 & 7(version 7)

Related to text book chapter 4 (version 8)

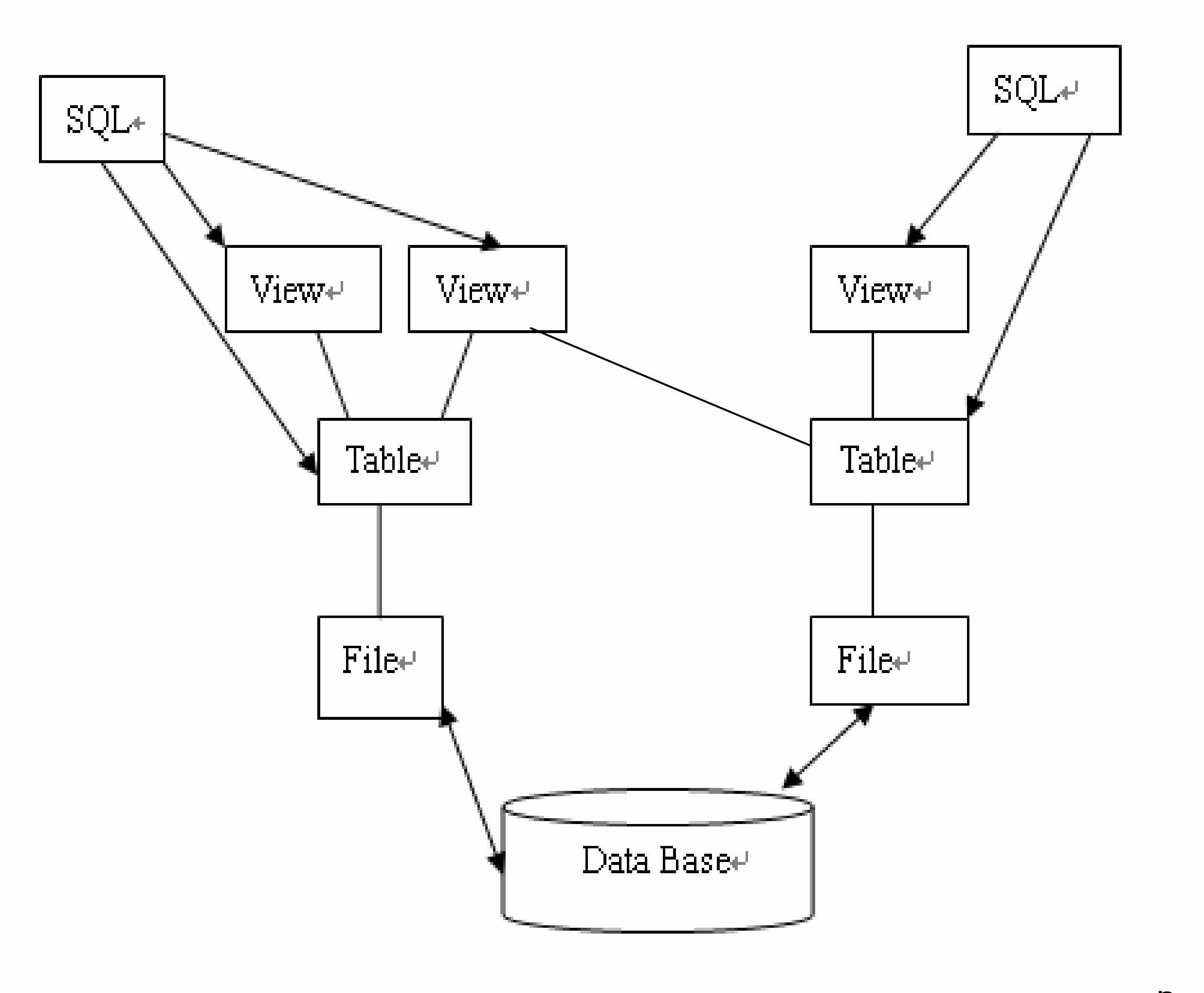
Contents

Introduction

- IBM SYSTEM R SEQUEL
  - 1974~~1979
- ANSI Standard SQL 1990
- ISO Standard SQL 1992
- SQL3 (SQL99)

Architecture

- View
- Table
- File



SQL Contents

- DDL
- DCL for integrity & security
  - Introduced later
  - Belong to DDL
- DML

SQL DDL

- Structure need to create
  - Table
  - View (chapter 8)
  - Index

SQL DDL-cont.

Create {table, view, index} < name > <description for that>

- **Attribute**
- **Type**

E.g.

```
Create Table DEPT ( DEPT#  Number,
                         DNAME  Char(5),
                        Budget    Number ( 7,2));
```

```
CREATE TABLE S
         ( S#       CHAR(5),
           SNAME    CHAR(20),
           STATUS   NUMERIC(5),
           CITY     CHAR(15),
       PRIMARY KEY ( S# ) );

CREATE TABLE P
         ( P#         CHAR(6),
           PNAME      CHAR(20),
           COLOR      CHAR(6),
           WEIGHT     NUMERIC(5,1),
           CITY       CHAR(15),
       PRIMARY KEY ( P# ) );

CREATE TABLE SP
         ( S#              CHAR(5),
           P#              CHAR(15),
           QTY            NUMERIC(9),
       PRIMARY KEY ( S#, P# ),
       FOREIGN KEY (S#) REFERENCES S,
       FOREIGN KEY (P#) REFERENCES P;
```

SQL DDL – cont.

- Index

Create index <name> on <table name>
(<index attr name list>)

E.g.

```
Create  index I1 on S (S#);
Create  index I2 on S (Sname);
```

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Index I1

S#	pointer
S1	Tuple1
S2	Tuple2
S3	Tuple3
S4	Tuple4
S5	Tuple5

Index I2

Sname	pointer
Adams	Tuple5
Blake	Tuple3
clark	Tuple4
Jones	Tuple2
Smith	Tuple1

SQL DDL – cont.

- Unique Index

E.g.

```
Create unique index I1 on S (S#);
```

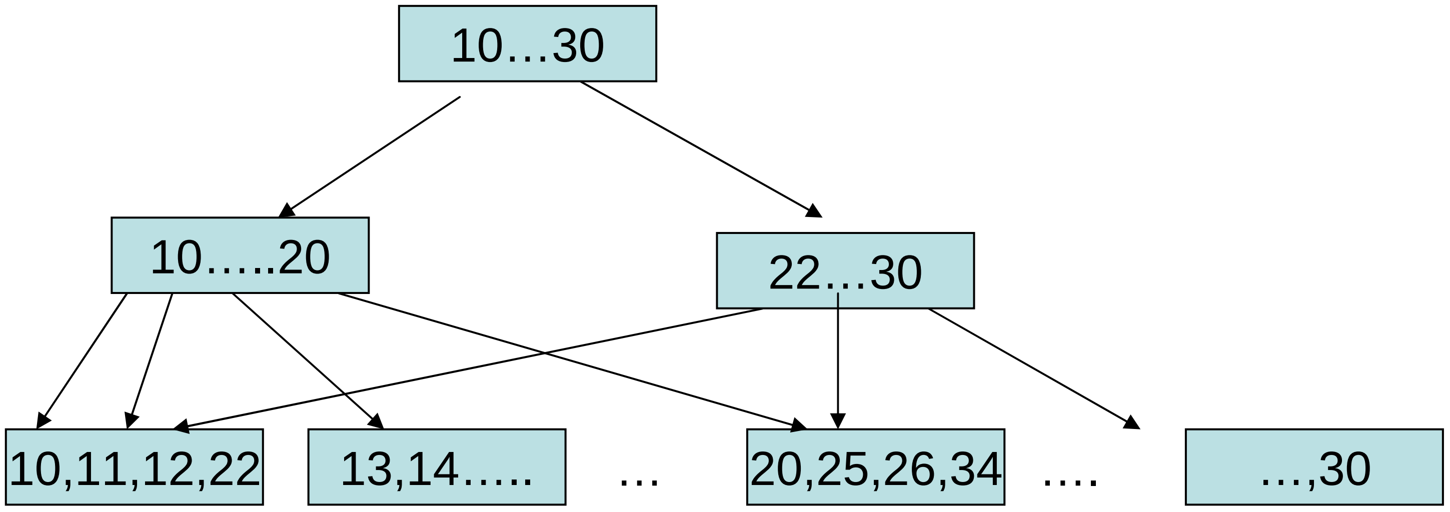
SQL DDL – cont.

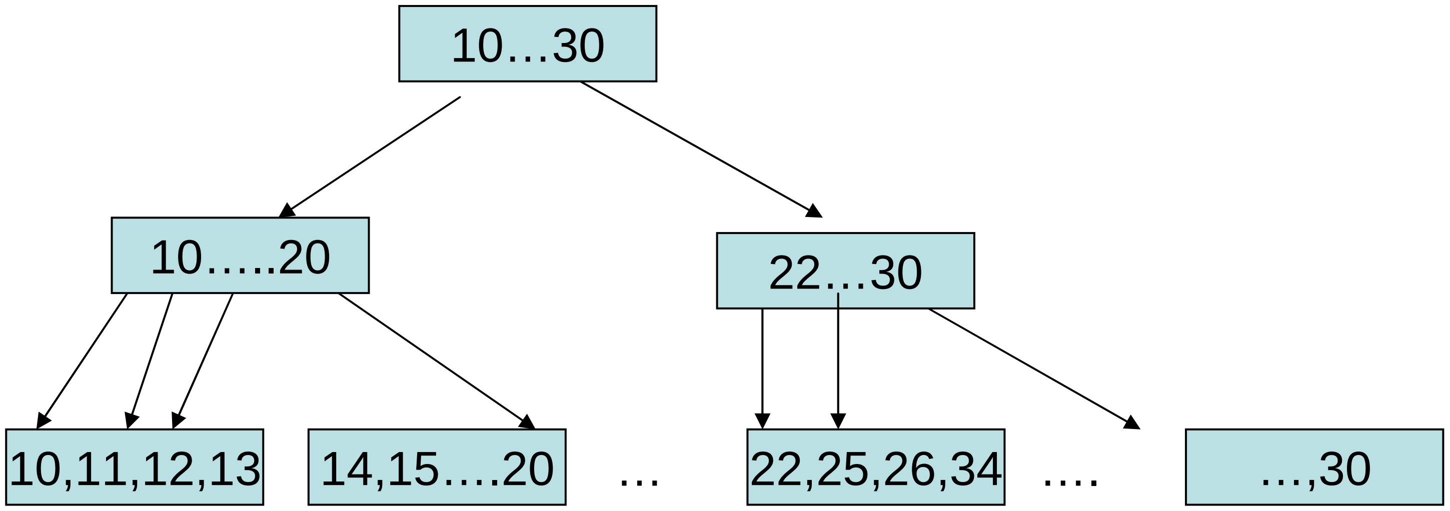
- Cluster Index

  Make tuples as much close as possible in storage according the index value’s order , it will reduce the I/O times when there are scan operate on the index value.

E.g.

Create cluster index CI1 on EMP (E#);


10…30
10…..20
22…30
10,11,12,22
13,14…..
…
20,25,26,34
….
…,30


10…30
10…..20
22…30
10,11,12,13
14,15….20
…
22,25,26,34
….
…,30

Query-Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements

Query-Basic Structure-cont.

- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  – A_is represent attributes

  – r_is represent relations

  – P is a predicate.

The select Clause

- The **select** clause is used to list the attributes desired in the result of a query.
- Find the names of all departments in the *DEPT* relation

select *dname*
from *DEPT*

The select Clause – cont.

- NOTE: SQL names are case insensitive, meaning you can use upper case or lower case.

The select Clause – cont.

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after **select.**

The select Clause – cont.

- Find the names of all departments in the DEPT relations, and remove duplicates

  select distinct d*name*
  from DEPT

The select Clause – cont.

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.

The select Clause – cont.

- The query:

```
select S#, Sname, Status ∗ 2
from   S
```

  would return a relation which is the same as the S relations, except that the attribute Status is multiplied by 2.

The where Clause

- The **where** clause consists of a predicate involving attributes of the relations that appear in the **from** clause.
- Find all supplier number for suppliers who lives in London with status greater than 20.

```
    select S#
     from   S
    where city='London' AND status> 20
```

The where Clause -Cont.

- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Comparisons can be applied to results of arithmetic expressions.

The where Clause -Cont.

- SQL Includes a **between** comparison operator in order to simplify **where** clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.

The where Clause -Cont.

- Find the supplier number of those suppliers with status between 20 and 30 (that is, ≥20 and ≤30)

```
select S#
from S
where status between 20 and 30
```

The from Clause

- The **from** clause lists the relations to be scanned in the evaluation of the expression.
- Find all employees and their department's information

```
select *
 from  EMP, DEPT
where  emp.D#=dept.D#
```

The Rename Operation

- The SQL allows renaming relations and attributes using alias name :

old-name   new-name

The Rename Operation-cont.

- Find the name, supplier number and supplier status of all suppliers; rename the column name S# as number *and sname as name*

select *sname name, s#* Snumber, *status*
from S

Tuple Variables

- Find the supplier names and part numbers for all suppliers having supplied that part.

```
select  sx.sname, spx.P#
 from    S sx, SP  Spx
 where sx.S#=spx.s#
```

Tuple Variables-cont.

- Tuple variables are defined in the **from** clause via the use of the alias.

String Operations

- SQL includes a string-matching operator for comparisons on character strings. Patterns are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.

String Operations-cont.

- Find the names of all suppliers whose city name includes the substring “Main”.

```
select  sname
from    s
where  city  like ‘%Main%’
```

String Operations – cont.

- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Exercise

1. Create following tables and some index, based on your understanding.

Movie(title, year, length, inColor, studioName, producerC#)

StarsIn(movieTitle, movieYear, strName)

MovieStar(name, address, gender, birthdate)

MovieExec(name, address, certification#, netWorth)

Studio(name, address, presidentC#)

Classes(class, type, country, numGuns, bore, displacement)

Ships(name, class, launched)

Battles(name, date)

Outcomes(ship, battle,result)

Next Class

- SQL continue