WEEK2

wu

May 21, 2018

Contents

1 Chap1 — 5
- 1.1 Taught — 5
- 1.2 DBMS(database management system) — 5
  - 1.2.1 collection of interrelated data — 5
- 1.3 DBMS solution — 5
  - 1.3.1 Commercial/freeware DBMS — 5
  - 1.3.2 Application programs — 5
- 1.4 View of data — 5
  - 1.4.1 Levels of Abstraction — 5
  - 1.4.2 Instances() and schemas() — 5
  - 1.4.3 Data module — 6
- 1.5 Relational database — 6
  - 1.5.1 Relational model — 6
  - 1.5.2 Data definition language(DDL) — 6
  - 1.5.3 Data manipulation language(DML) — 6
- 1.6 SQL — 6
  - 1.6.1 Example — 6
  - 1.6.2 Database design — 7

2 Chap2 — 7
- 2.1 Structure of relational databases — 7
- 2.2 Database schema — 7
- 2.3 Keys — 7
- 2.4 Relational query language — 8
  - 2.4.1 procedural, non-procedural, or declarative — 8
  - 2.4.2 "Pure" language — 8
  - 2.4.3 relational algebra — 8

3 DONE **8**

4 Chap6 **9**

4.1 Relational Algebra 9
4.1.1 selection operation 9
4.1.2 project operation 9
4.1.3 union operation 9
4.1.4 set difference operation 9
4.1.5 Cartesian product operation 9
4.1.6 rename operation 10
4.1.7 Additional operations 10
4.1.8 Logical equivalence of RA plans 11
4.2 extend relational algebra operations 11
4.2.1 generalized projection 11
4.2.2 Aggregate functions and operations 11
4.2.3 Multiset relational algebra 11

5 DONE **11**

6 Chap3 **12**

6.1 SQL data definition 12
6.1.1 Basic types 12
6.1.2 Basic schema definition: table 12
6.1.3 update to table 12
6.2 Foreign key and primary key 12
6.3 Basic query structure 13

7 DONE **15**

8 Chap4 **15**

8.1 Joined relations 15
8.2 View 16
8.2.1 Materialized views 16
8.2.2 Transaction 16
8.2.3 Integrated constraints 16
8.2.4 Referential integrity 17
8.3 Complex check clause 17
8.4 DATATYPE 17
8.4.1 Built-in data types in SQL 17
8.4.2 index creation 17

8.4.3 User-defined 17
8.4.4 large-object types 17
8.5 Authorization 18
8.5.1 Authorization specification in SQL 18
8.5.2 privileges in SQL 18
8.5.3 revoking authorization in SQL 18
8.5.4 role 18
8.5.5 Authorization on views 19
8.5.6 other authorization fearture 19

9 Chap5 **19**
9.1 Trigger 19
9.1.1 example 19
9.1.2 triggering events and action in SQL 19
9.1.3 statement level triggers 19
9.2 Accessing SQL from a programming language 20

10 DONE **20**

11 Chap7: Entity-relationship model **20**
11.1 ER model 20
11.2 UML 22

12 DONE **22**

13 Chap8: Relational database design **22**
13.1 First normal form 22
13.2 Functional dependencies 23
13.3 Boyce-Codd normal form :: 24
13.4 Third normal form 24
13.5 How good is BCNF 24
13.6 Closure of attributes sets 25
13.7 Algorithms for decomposition 26
13.7.1 BCNF 26
13.7.2 3NF decomposition 27
13.8 Decomposition using multivalued dependencies 27

14 Chap10 **28**
14.1 Overview 28
14.2 Magnetic disk and flash storage 28
14.2.1 physical characteristics of disks 28

14.2.2 Performance measures of disks 28
14.2.3 optimization of disk-block access 29
14.2.4 flash storage 29
14.3 RAID(Redundant arrays of independent disks) 29
14.3.1 improvement in performance via parallelism 30
14.3.2 Raid level 30
14.4 File organization 30
14.4.1 fixed-length records 30
14.4.2 varaible-length record 30
14.4.3 organization of records in files 31
14.5 data dictionary storage 31

15 TODO **31**

16 Chap 11 **31**
16.1 Basic concepts 31
16.2 Ordered indices 32
16.3 B+-tree index files 32
16.4 B+-tree file organization 32
16.5 Bulk loading and bottom-up build 33
16.6 Multiple-key access 33

17 DONE **33**

18 chap12 **33**
18.1 Measure of query cost 33
18.2 selectoin operation 33
18.2.1 file scan 33
18.2.2 index scan 34
18.2.3 selection involving comparisons 34
18.2.4 implementation of complex selections 34
18.3 external sort-merge 34
18.4 Join 34
18.4.1 nested-loop join 34
18.4.2 block nested-loop join 35
18.4.3 indexed nested-loop join 35
18.4.4 merge join 35
18.4.5 hash join 35
18.5 evaluation of expression 35
18.5.1 materialization 35

18.5.2 pipeline 35
19 TODO **35**
20 chap13 **35**
20.1 overview 35
20.2 transformation 36

1 Chap1

1.1 Taught

1-8, 10, 11-16, 22-23

1.2 DBMS(database management system)

1.2.1 collection of interrelated data

1.3 DBMS solution

1.3.1 Commercial/freeware DBMS

1.3.2 Application programs

1.4 View of data

1.4.1 Levels of Abstraction

1. physical level how the data are stored
2. logical level relations among the data
3. view level

1.4.2 Instances() and schemas()

instance the collection of information stored in the database at a particular moment

schema the overal design of the database

1.4.3 Data module

1. a collections of tools for describing
   (a) Data
   (b) Data relationships
   (c) Data semantics
   (d) Data constraints

1.5 Relational database

1.5.1 Relational model

ID	name	$\text{Dept}_{\text{name}}$	Salary
22	112	131	13131

1.5.2 Data definition language(DDL)

create table instructor (ID char(5), name varchar(5))

DDL compiler generates a set of table templates stored in a data dictionary

Data dictionary contains metadata

1.5.3 Data manipulation language(DML)

Also known as query language() Types of access are

Procedural user specifyies what data is required and how to get the data

Declarative(nonprocedural) require a user to specify what data are needed without specifying how to get those data

query a statement requesting the retrieval of information

1.6 SQL

1.6.1 Example

Find the name of the instructor with ID 5 select name from instructor where instructor.ID = ’5’

1.6.2 Database design

2 Chap2

2.1 Structure of relational databases

domain the set of permitted values of each attribute

atomic a domain is atomic if elements of the domain are considered to be indivisible units

null a null value is a special value that signifies that the value is unknown or does not exist

2.2 Database schema

database schema logical design of the database

database instance snapshot of the data in the database at a given instant in time

relation schema consists of a list of attributs and their conrresponding domains

- A,..,A are attributes
- R = (A,. . . ,A) is a relation schema e.g. instructor$_{\text{schema}}$ = (ID, name, dept$_{\text{name}}$)
- r(R) denotes a relation r on the relation schema e.g. instructor(instructor$_{\text{schema}}$)
- fiven sets D,. . . ,D, a relation r is a subset of DŒD. . . ŒD Thus a relation is a set of n-tuples (a,. . . ,a) where each aD
- The current values (relation instance) of a relation are specified by a table
- An element t of r is a tuple, represented by a row in a table

relation instance corresponds to the programming language notion of a value of a varia

2.3 Keys

Let K R

superkey K is a superkey of R if values for K are suffient to identify a unique tuple of each possible relation r(R)

{ID} and {ID, name} are both superkeys of instrctor

candidate key s superkey K is minimal

{ID} is minimal

primary key one of the candidate key is chosen primary key

foreign key A relatoin schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a foreign key

2.4 Relational query language

2.4.1 procedural, non-procedural, or declarative

2.4.2 "Pure" language

- relational algebra
- tuple relational calculus
- domain relational calculus

2.4.3 relational algebra

- procedural language
- six basic operators
  - selection:
  - projection:
  - union:
  - natural join: //SHEN MI
  - set different: -
  - cartesian product: Œ

3 DONE

Ex 1.8, 1.9, 1.13, 1.15, 2.9, 2.13 Consider the bank database of Figure 2.15. a. What are the appropriate primary keys? b. Given your choice of primary keys, identify appropriate foreign keys.

branch(branch name, branch city, assets) customer (customer name, customer street, customer city) loan (loan number, branch name, amount) borrower (customer name, loan number) account (account number, branch name, balance) depositor (customer name, account number)

Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries: a. Find all loan numbers with a loan value greater than $10,000. b. Find the names of all depositors who have an account with a value greater than $6,000. c. Find the names of all depositors who have an account with a value greater than $6,000 at the Uptown branch

4 Chap6

4.1 Relational Algebra

4.1.1 selection operation

- notation: (r)

selection predicate

- (r) = {t | t r and p(t)}

  and

  or

4.1.2 project operation

4.1.3 union operation

must have the same arity the attribute domains must be compatible

4.1.4 set difference operation

have same arity attribute domains must be compatible

4.1.5 Cartesian product operation

- r Œ s = {t q | t r **and** q s}
- assume that r(R) and s(S) are disjoint
- If not disjoint, then **renaming** must be used

4.1.6 rename operation

(E) if E has arity n, then ,. . . ,(E)

4.1.7 Additional operations

set intersection r s = r - (r - s)

natural join (bowtie) r s R = (A,B,C,D) S = (B, E, D) R S = (A, B, C, D, E) Cartesian product is renaming projection of renaming of Cartesian product natural join is associative and commutative

theta join r s

equijoin

semijoin often used to compute natural joins in distributed databases

assignment

outer join
- extension of join operation avoids loss of information
- use **null** values: **null** signifies the value is unknown or doesn't exist instructor

ID	name	$dept_{name}$
1	A	Comp
2	B	Finance
3	C	Miao

teaches

ID	$course_{id}$
1	CS-101
2	FIN-201
4	BIO-201

left $outer_{join}$:

ID	name	$dept_{name}$	$course_{id}$
1	A	Comp	CS-101
2	B	Finance	FIN-201
3	C	Miao	**null**

right outer join

ID	name	$\text{dept}_{\text{name}}$	$\text{course}_{\text{id}}$
1	A	Comp	CS-101
2	B	Finance	FIN-201
4	**null**	**null**	BIO-201

full outer join

ID	name	$\text{dept}_{\text{name}}$	$\text{course}_{\text{id}}$
1	A	Comp	CS-101
2	B	Finance	FIN-201
3	C	Miao	**null**
4	**null**	**null**	BIO-201

null values three-ordered logic

or	null	true	false
null	null	null	null
true	null		
false	null		

division operator given relations r(R) and s(S) s.t. S

4.1.8 Logical equivalence of RA plans

- ((R)) = ((R))

4.2 extend relational algebra operations

4.2.1 generalized projection

- extends the projection operation by allowing arithmetic functions to be used in the projection list

4.2.2 Aggregate functions and operations

4.2.3 Multiset relational algebra

5 DONE

6.1 6.13 Write the following queries in relational algebra, using the university schema. a. Find the titles of courses in the Comp. Sci. department that have 3 credits. b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result. c. Find the highest salary of any instructor. d. Find all instructors earning the highest salary (there may be more e. Find the enrollment of each section that was offered in Autumn 2009. f. Find the maximum enrollment, across all sections, in Autumn 2009. g. Find the sections that had the maximum enrollment in Autumn 2009.

employee (person name, street, city) works (person name, company name, salary) company (company name, city) manages (person name, manager name) Consider the relational database of Figure 6.22. Give a relational-algebra expression for each of the following queries: a. Find the company with the most employees. b. Find the company with the smallest payroll. c. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

6 Chap3

6.1 SQL data definition

6.1.1 Basic types

char(n): fixed length varchar(n): variable-length

6.1.2 Basic schema definition: table

6.1.3 update to table

insert

delete

drop table

alter change the structure of the table

- **alter table r add A D** where A is the name of the attribute to be added to relation r and D is the domain of A
- **alter table r drop A** A is the name of an attribute

6.2 Foreign key and primary key

foreign keys are constraints

6.3 Basic query structure

select clause
- to force the elimination of duplicates, insert the keyword **distinct**
- **all** specifies that duplicates not to be removed
- select * denotes all attributes.
- An attribute can be a literal with no from clause *select '437'* results is a table with one column and a single row with value "437"
- *select 'A' from instructor result is a table with one column and N rows, each row with value "A"
- Can also contain arithmetic expressions

where clause
- Find the Cartesian product (select * from instructor, teaches) generates every possible **instructor-teacher** pair for common attribute, the attributes in the resulting table are renamed using the relation name

join
- the comma in from clause

natural join
- select * from instructor natural join teaches

rename operation
- **as**

string operation
- % matches any substring
- underscore _ matches any character
- e.g. where name like '%dar%'
- use %: %
- concatenation ||
- converting from upper to lower case
- finding string length, extracting substring

ordering the display of tuples
- select distinct name from instructor order by name
- **desc** for descending order **asc** for ascending. e.g. order by name desc
- can sort on multiple attributes e.g. order by dept$_{\text{name}}$, name

where clause predicates
- **between** e.g. select name from instructor where salary between 90000 and 10000

- tuple comparison $(a_1, a_2) \leq (b_1, b_2)$ iff $a_1 \leq b_1$ and $a_2 \leq b_2$

duplicate
- Multiset

Set operation
- union, intersect, except each of the above operations automatically eliminates duplicates
- to retain all duplicates, use union all, intersect all, except all

Null value
- arithmetic expression involving null is null
- is null can be used to check for null values select name from instructor where salary is null

Aggregate functions
- avg, min, max, sum, count
- Except count, all aggregations apply to a single attribute
- e.g. select avg(salary) from instructor where $\text{dept}_{\text{name}}$='Comp. Sci.';

avg(salary)
750000

- Find the number of tuples in the course relation select count(*) from course;

grouping and aggregation
- e.g. select product, sum from purchase group by product
- 1.compute the from and where clauses 2.group by the attributes in the group 3.Compute the select clause

Having clause
- predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before groups

al form of grouping and aggregation
- select S from R,...,R where C group by a,...,a having C

Nested subqueries
- SQL provides a mechanism for the nesting of subqueries. A subquery is a **select-from-where** expression that is nested within another query.

Set membership in tests for set membership

not in

set comparison-"some" clause **> some < some**

set comparison - "all" clasue **> all select** name **from** instructor **where** salary > **all** (**select** salary **from** instructor **where** $\text{dept}_{\text{name}}$ = 'Biology');

Test for empty relations The **exists** construct returns the value **true** if te argument subquery is nonempty

not exists

unique

- The **unique** construct tests whether a subquery has any duplicate tuples in its result
- The **unique** construct evaluates to "true" if a given subquery contains no duplicates

with

- provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs
- e.g **with** $\text{max}_{\text{budget}}$(value) **as** (**select max** (budget) **from** department) **select** department.name **from** department, $\text{max}_{\text{budget}}$ **where** department.budget = $\text{max}_{\text{budget}}$.value;
- is very useful for writing complex queries
- with dept _total ($\text{dept}_{\text{name}}$, value) as (select $\text{dept}_{\text{name}}$, sum(salary) from instructor group by $\text{dept}_{\text{name}}$), $\text{dept}_{\text{totalavg}}$(value) as (select avg(value) from $\text{dept}_{\text{total}}$) select $\text{dept}_{\text{name}}$ from $\text{dept}_{\text{total}}$, $\text{dept}_{\text{totalavg}}$ where $\text{dept}_{\text{total}}$.value > $\text{dept}_{\text{totalavg}}$.value;

insertion

- **insert into ... values** ....
- **insert into** student **select** ID, name, $\text{dept}_{\text{name}}$, 0 **from** instructor
- **insert into** table1 **select** * **from** table1

update

- **update** ... **set** .... **where** ...
- **update** ... **set** ... **case** ... **when** ... **then** ... **else** ... **end**

7 DONE

first 3.8 3.9 3.10 3.11 3.15 4.7 4.9 4.12

8 Chap4

8.1 Joined relations

Join operation

- takes two relations and return as a result another relation

outer join
- an extension of the join operation that avoids loss of information
- use **null** values
- To distinguish normal joins from outer joins, normal joins are called inter joins in SQL.

inner join
- equivalent to **join**

8.2 View

- In some cases, it's not desirable for all users to see the entire logical model
- A **view** provides a mechanism to hide certain data from the view of certain users
- **create view** v **as** <query expression>
- once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- view is not a table
- Most SQL implementations allow update only on simple views
- The **select** contains only attribute names of the relation and doesn't have any expressions, aggregates

8.2.1 Materialized views

Materializing a view create a physical table containing all the tuples in the result of the query defining the view

8.2.2 Transaction

- Atomic transaction Either fully executed or rolled back as if never occured

8.2.3 Integrated constraints

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database don't result in a loss of data consistency

8.2.4 Referential integrity

Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation

8.3 Complex check clause

check

8.4 DATATYPE

8.4.1 Built-in data types in SQL

date containing a (4 digit) year, month and data

time time of day, in hours, minutes **time** ’09:00:30’

timestamp date plus time of day

interval period of time interval ’1’ day

8.4.2 index creation

- **create index** $\text{studentID}_{\text{index}}$ **on** student(ID)
- indices are data structures used to **speed up** access to records with specified values for index attributes
- e.g. select * from student where ID = ’12345’ can be executed by using the index to find the required record without looking at all records of student

8.4.3 User-defined

- **create type** Dollars **as numeric** (12, 2) **final**
- **create domain** $\text{person}_{\text{name}}$ **char** (20) **not null**

8.4.4 large-object types

- large object are stored as large object

  blob binary large object object is a large collection of uninterpreted binary data

  clob character large object

8.5 Authorization

Forms of authorization on parts of the database
- read
- insert
- update
- delete

Forms of authorization to modify the database
- index
- resources allow creation of new relations
- alteration
- drop

8.5.1 Authorization specification in SQL

- **grant** <privilege list> **on** <relation name or view name> **to** <user list>
- <user list> is
  - a user-id
  - **public** which allows all valid users the privilege granted
  - a role
- granting on a privilege on a view doesn't imply granting any privileges on the underlying relation

8.5.2 privileges in SQL

select **grant select on** instrutor **to** U_1, U_2, U_3

insert

8.5.3 revoking authorization in SQL

- the revoke statement is used to revoke authorization **revoke** <privilege list> **on** <relation name or view name> **from** <user list>

8.5.4 role

- **create role** instructor
- **grant** instructor **to** Amil
- **grant select on** takes **to** instructor

8.5.5 Authorization on views

- **grant select on** $\text{geo}_{\text{instructor}}$ **to** $\text{geo}_{\text{staff}}$

8.5.6 other authorization fearture

- **grant reference** ($\text{dept}_{\text{name}}$) **on** department **to** Mariano

9 Chap5

9.1 Trigger

- a statement that the system executes automatically by the system as a side effect of a modification to the database
- to design, we must specify the condition specify the action

9.1.1 example

$\text{time}_{\text{slotid}}$ is not a primary key, so cannot create a foreign key constraint Alternative we can use trigger **create trigger** $\text{timeslot}_{\text{check1}}$ **after insert on** sectin **referencing new row as** nrow **for each row when** ($\text{nrow.time}_{\text{slotid}}$ **not in** (**select** $\text{time}_{\text{slotid}}$ **from** $\text{time}_{\text{slot}}$)) **begin rollback end**

9.1.2 triggering events and action in SQL

- triggering event can be **insert, delete** or **update**
- triggering on update can be restricted to specific attributes **after update of** takes **on** grade
- values of attributes **before and after** an udpate can be referenced **reference old row as reference new row as**
- triggers can be activated **before an event**

9.1.3 statement level triggers

- instead of executing a seperate action for each affectedrow a single action can be executed for all rows affected by a transaction
  - use **for each statement**
  - use **referencing old table**

9.2 Accessing SQL from a programming language

- To write an embedded SQL query, we use **declare**
- The **open** statment for our example is as follows: **EXEC SQL open c**; This statement causes the database system to execute the query and to save the results within a temporary relatoin

Dynamic SQL

Embedded SQL – the SQL statements are identified at compile time using a preprocessor

- Procedural extensions and stored procedures
- Functions

  returns indicates the variable-type taht is returned

  return specifies the values that are to be returned as result of invoking the function

10 DONE

5.15 5.17 5.21 employee (employee name, street, city) works (employee name, company name, salary)

11 Chap7: Entity-relationship model

11.1 ER model

model
- A database can be modeled as : a collection of entities relation among entities
- An **entity** is an object that is distinguishable from all other objects An **entity** ahs a set of properties, and the values for some set of properties may uniquely identify an entity

entity set – a set of entities of the same type that share the same properties, or attributes

relation **A mathematical definition**
- Let A = {1, 2, 3}, B = {a, b, c, d}
- We define a **relationship** of subset of $A \times B$

A **relationship** is an association among several entities.

relationship set

degree of relationship set
- binary relationship involve two entity sets most relationship sets in a database system are binary

Attributes

mapping cardinality

redundant attributes
- suppose we have entity sets instructor, with attributes including $\text{dept}_{\text{name}}$ department
- and a relationship $\text{inst}_{\text{dept}}$ relating instructor and department
- Attribute $\text{dept}_{\text{name}}$ in entity instructor is redundant since there is an explicit relationship $\text{inst}_{\text{dept}}$ which relates instructors to department

E-R diagram Roles
- Entity sets of a relationship needn't to be distinct Each occurence of an entity set plays a "role" in the relationship
- Such as requisite course
- The labels "$\text{course}_{\text{id}}$" and "$\text{prereq}_{\text{id}}$" are called roles

Cardinality constraint We express cardinality constraints by drawing either a directed line ->, signifying "one", or an undirected line (–), signifying "many" $\rightarrow$

one-to-one relationship, one-to-many, many-to-one, many-to-many

total participation every entity in the entity set participates in at least one relationship in the relationship set

Min and Max
- A line may have an associtaed minimum and maximum cardinality, shown in the form l..h, where l is the minimum and h the maximum cardinality
- A maximum value of * indicates no limit

Cardinality constraints on ternary relationship

Week entity set
- An entity set that doesn't have a primary key is referred to as **weak entity set**
- The existence of weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set **via a total**, **one-to-many** relationship set from the identifying to the weak entity set

- **identifying relationship** depicted using a double diamond

- The **discriminator** of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set
- We underline the discriminator of a weak entity set with a dashed lien

Binary vs non-binary relationship

- A ternary relationship parents, relating a child to his/her father and mother
- Any non-binary relationship can be represented using binary relationships by creating an artificial entity set

Extended ER features **Specialization** • top-down design process

Generalization • bottom-up design process

Aggregation • relation among relations

11.2 UML

- unified modeling language

12 DONE

7.1 7.2 7.20 Consider the E-R diagram in Figure 7.29, which models an online bookstore. a. List the entity sets and their primary keys. b. Suppose the bookstore adds Blu-ray discs and downloadable video to its collection. The same item may be present in one or both formats, with differing prices. Extend the E-R diagram to model this addition, ignoring the effect on shopping baskets. c. Now extend the E-R diagram, using generalization, to model the case where a shopping basket may contain any combination of books, Blu-ray discs, or downloadable video.

13 Chap8: Relational database design

13.1 First normal form

- domain is **atomic** if its elements are considered to be indivisible units

First normal form the domain of all attributes are atomic

- Atomicity is actually a property of how the elements of the domain are used

13.2 Functional dependencies

- constraints on the set of legal relations
- require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- Let R be a relation scheme $\alpha \subseteq R$ and $\beta \subseteq R$
- the **functional dependency** $\alpha \to \beta$ holds on R if and only if for any legal relations r(R), whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β that is $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

A	B
1	4
1	5
3	7

  here, $A \to B$ does **not** hold, but $B \to A$ hold

- K is a superkey for relation schema R if and only if $K \to R$ K is a candidate key for R if and only $K \to R$ and for no $a \subset K, a \to R$
- **Functional dependencies** allow us to express constraints that cannot be expressed using superkeys. E.g. inst$_{\text{dept}}$(ID__,name, salary, $\underline{\text{dept}_{\text{name}}}$, building, budget) We would expect $dept_{name} \to building$ and $ID \to building$ hold but $dept_{name} \to salary$ does not hold

usage
  - To test relations to see if they are legal under a given set of functional dependencies
    * If a relation r is legal under a set F of functional dependencies, we say r **satisfies** F
  - specify constraints on the set of legal relations
    * We say F **holds on** R if all legal relations on R satisfy the set of functional dependencies F

trivial it is satisfied by all instances of a relation set of functional dependencies

- given a set F of functional dependencies, there are certain other functional dependencies that are logically implies by F
  - e.g. If $A \to B$ and $B \to C$, then $A \to C$
- The set of **all** functional dependencies logically implied by F is the **closure** of F denoted by F^+

13.3 Boyce-Codd normal form ::

- A relation schema R is in BCNF w.r.t to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \to \beta$, at least one of the following holds: $\alpha \to \beta$ is trivial α is a superkey of R
- decomposing a schema into BCNF Supposing we have a schema R and a non-trivial dependency $\alpha \to \beta$ causes a violation of BCNF We decompose R into: $(\alpha \cup \beta)$ (R-$(\beta - \alpha)$)
- BCNF and dependency predervation

13.4 Third normal form

- for all $\alpha \to \beta$ in F^+, at least one of the following holds $\alpha \to \beta$ is trivial α is a superkey for R Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

Goals of normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation R is in "good" form
- In the case that a relation scheme R is not in "good" form, decompose it into the set of relation scheme $\{R_1, \ldots, R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - the decomposition should be denpendency preserving

- minimal relaxiation of BCNF

13.5 How good is BCNF

consider a relation

- inst$_{\text{info}}$(ID, child$_{\text{name}}$, phone)
-

ID	child$_{\text{name}}$	phone
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

problem
- there are no non-trivial functional dependencies and therefore the relation in BCNF
- insertion anomalies – if we add a phone 981-992-3443 to 99999, we need to add two tupples (99999, David, 981-992-3443) (99999, William, 981-992-3443)

solution decompose $inst_{info}$ into two tables

13.6 Closure of attributes sets

Armstrong's axioms **reflexivity** if $\beta \subseteq \alpha$, then $\alpha \to \beta$

augmentation if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$

transitivity if $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$

these rules are

- **sound** (generate only functional dependencies that actually hold)
- **complete** (generate all functional dependencies that hold)

Another rule **Union rule** if $\alpha \to \beta$ and $\alpha \to \gamma$, then $\alpha \to \beta\gamma$

Decomposition rule if $\alpha \to \beta\gamma$, then $\alpha \to \beta, \alpha \to \gamma$

Pseudotransitivity rule If $\alpha \to \beta$ and $\gamma\beta \to \delta$, then $\alpha\gamma \to \delta$

Canonical Cover

- If we perform a update on the relation, the system must ensure the update doesn't violate any functional dependencies. We need the reduce the check time

extraneous attributes
  - Consider a set F of functional dependencies and the functional dependency $\alpha \to \beta$ in F
  - A is extraneous in α if $A \in \alpha, F \to (F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$
  - A is extraneous in β if $A \in \beta, (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\} \to F$
  - For example, if we have $AB \to C, A \to C$, then B is extraneous in $AB \to C$
  - If $A \in \beta$, consider the set $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ check if $\alpha \to A$ can be inferred from F'. Compute α^+ under F', if α^+ includes A, then A is extraneous in β

- If $A \in \alpha$, let $\gamma = \alpha - \{A\}$ and check if $\gamma \to \beta$ can be inferred from F. To do so, compute γ^+ under F, if γ^+ includes all attributes in β then A is extraneous

Canonical cover F_c's properties
- $F_c \Leftrightarrow F$
- No functional dependency in F_c contains an extraneous attribute
- Each left side of a functional dependency in F_c is unique. $\forall \alpha_1 \to \beta_1, \alpha_2 \to \beta_2 \in F_c, \alpha_1 \neq \alpha_2$

Lossless decomposition
- We say that decomposition is a lossless decomposition if there is no loss of information by replacing $r(R)$ with two relation schemas $r_1(R_1), r_2(R_2)$ Or $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
- we can use functional dependencies to show when certain decompositions are lossless. If at least one of the following functional dependencies is in F^+: $R_1 \cap R_2 \to R_1$ $R_1 \cap R_2 \to R_2$

Dependency preservation
- Let F be a set of functional dependencies on a schema R, and let $R_1, \ldots, R_n$ be a decomposition of R. The **restriction** of F to R_i is the set F_i of all functional dependencies in F^+ that include *only* attributes of R_i
- Let $F' = F_1 \cup \cdots \cup F_n$, $F'^+ = F^+$ is a **dependency-preserving decomposition**

13.7 Algorithms for decomposition

13.7.1 BCNF

BCNF testing For every subset α of attributes in R_i, check that α^+ either includes no attribute of $R_i - \alpha$ or includes all attributes of R_i

BCNF decomposition
- pseudocode

```
result = {R}
done = False
F+=...
while not done:
    if Ri is not BCNF
        alphatobeta = nontrivial functional dependency hold on Ri a
        and alphacapbeta = None
        result = (result - Ri) cup (Ri - beta) cup (alpha, beta)
```

```
          else
               done = True
```

13.7.2 3NF decomposition

```
F_c = canonical cover for F
i = 0
for $\alpha\to\beta$ in F_c:
     i = i + 1
     R_i = $\alpha\beta$
if none of R_i contains a candidate keys:
     i = i + 1
     R_i = any candidate key of R
while R_j can be deleted(contains in another R):
     R_j = R_i
     i = i - 1
return (R_1, R_2, ..., R_i)
```

13.8 Decomposition using multivalued dependencies

- Multivalued dependencies
  - Multivalued dependencies, on the other hand, do not rule out the existence of certain tuples. Instead, theyrequire that other tuples of a certain form be present in the relation.
  - Let r(R) be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency** $\alpha \twoheadrightarrow \beta$ holds on R if in any legal instance of relation r(R), for pairs of tuples t_1, t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3, t_4 s.t. $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$ $t_3[\beta] = t_1[\beta]$ $t_3[R-\beta] = t_2[R-\beta]$ \$$\mathrm{t}_{4[\beta]}$=$\mathrm{t}_{2[\beta]}$ $t_4[R-\beta] = t_1[R-\beta]$

	α	β	R-α-β
$\mathrm{t_1}$	$\mathrm{a_1 \ldots a_i}$	$\mathrm{a_{i+1} \ldots a_j}$	$\mathrm{a_{j+1} \ldots a_n}$
$\mathrm{t_2}$	$\mathrm{a_1 \ldots a_i}$	$\mathrm{b_{i+1} \ldots b_j}$	$\mathrm{b_{j+1} \ldots b_n}$
$\mathrm{t_3}$	$\mathrm{a_1 \ldots a_i}$	$\mathrm{a_{i+1} \ldots a_j}$	$\mathrm{b_{j+1} \ldots b_n}$
$\mathrm{t_4}$	$\mathrm{a_1 \ldots a_i}$	$\mathrm{b_{i+1} \ldots b_j}$	$\mathrm{a_{j+1} \ldots a_n}$

  - If $\alpha \to \beta$, then $\alpha \twoheadrightarrow \beta$ If $\alpha \twoheadrightarrow \beta$, then $\alpha \twoheadrightarrow R - \alpha - \beta$
- Fourth normal form

- for all multivalued dependencies in D^+, at least one of the following holds
  - $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency
  - α is a superkey for R
- Every 4NF schema is in BCNF
- **restriction** of D to R_i is the set D_i consisting of:
  - All functional dependencies in D^+ that include only attributes of R_i
  - All multivalued dependencies of the form: $\alpha \twoheadrightarrow \beta \cap R_i$ where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+

Decomposition
  - really same to BCNF

14 Chap10

14.1 Overview

cache main memory flash memory magnetic disk optical disk magnetic tapes

14.2 Magnetic disk and flash storage

14.2.1 physical characteristics of disks

- a disk contains many **platter**
- each disk **platter** has a flat, circular shape
- the disk surface is logically divided into **tracks**, which are subdivided into **sectors**. A **sector** is the smalles unit of information
- ith tracks of all the platters together are called the ith **cylinder**
- **disk controller**

14.2.2 Performance measures of disks

Access time
- the time it takes **from** when a read or write request is issued **to** when data transfer begins. Consist of:

  Seek time() – time it takes to reposition the arm over the correct track

  Rotational latency – time it takes for the **sector** to be accessed to appear under the head

Data-transfer rate
- the rate at which data can be retrived from or stored to the disk

Mean time to failure(MTTF)()
- the average time the disk is expected to run continuously without any failure
-

14.2.3 optimization of disk-block access

Block
- a contiguous sequence of sectors from a single track
- data is transfered between disk and main memory in blocks

Disk-arm-scheduling
- algorithms order pending accesses to tracks so that disk arm movement is minimized
- elevator algorithm

File organization
- optimize block access time by organizing the blocks to correspond to how data will be accessed
- e.g. store related information on the same nearby cylinders

Nonvolatile write buffers
- speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
- controller then writes to disk whenever the disk has no other requests or request has been pending for some time

Log disk
- a disk devoted to writing a sequential log of block updates
- used like nonvolatile RAM

14.2.4 flash storage

NAND flash
- requires page-at-a-time read
- ssd (solid state disks) use multiple flash storage
- erase is slow

14.3 RAID(Redundant arrays of independent disks)

- high capacity and high speed by using multiple disks in parallel. high reliability by storing data redundantly

redundancy
  - store extra information that can be used to rebuild information lost in a disk failure

    - E.g. **Mirroring**

- Mean time to failure

14.3.1 improvement in performance via parallelism

- two main goals of parallelism in a disk system
    - load balance multiple small accesses to increase throughput
    - parallelize large accesses to reduce response time

bit-level striping split the bits of each byte across multiple disks

block-level striping with n disks, block i goes to disk (i mod n) + 1

14.3.2 Raid level

14.4 File organization

14.4.1 fixed-length records

- store record i starting from byte n * (i - 1), where n is the size of each record
- deletion
    - free list store the address of the first deleted record in the file header

14.4.2 varaible-length record

- arise in several ways
    - storage of multiple record types in a file
    - record types that allow variable lengths for one or more fields such as strings
    - record types that allow repeating fields
- variable length attributes represented by fixed szie, with actual data stored after all fixed length attributes
- null values represented by null-value bitmap

slotted page structure
    - contains number of record entries end of free space in the block location and size of each record

  - records can be moved around within a page to keep them contiguous with no empty space between them
  - pointers shouldn't point directly to record–instead they should point to the entry for the record in header

14.4.3 organization of records in files

- sequential file organization
- multitable clustering file organization

14.5 data dictionary storage

- data dictionary stores **metadata**
- information about relations
- user and accouting information
- statistical and descriptive data
- physical file organization

15 TODO

10.8 10.14 10.17

16 Chap 11

16.1 Basic concepts

index file
- consist of records of the form

  search-key pointer

- two kinds of indices
  - ordered indices
    - primary index
      - the index whose search key specifies the sequential order of the file
    - secondary index

        · the index whose search key specifies the sequential order of the file

  - hash indices

- index evaluation metrics
  - access types
  - access time
  - insertion time
  - deletion time
  - space overhead

16.2 Ordered indices

- index entities are stored sorted on the search key value

rimary index(clustering index)
  - not primary key
  - the index whose search key specifies the sequential order of the file

lary index(non-clustring index)
  - an index whose search key specifies an order different from the sequential order of the file
  - e.g.

dense index
  - index record appears for every search-key value

sparse index files
  - contains index records for only some search-key
  - compared to dense indices
    * less space and less maintenance
    * generally slower than dense index
  - good tradeoff
    * sparse index with an index entry for every block in file, corresponding to least search-key value in the block

multilevel index

16.3 B+-tree index files

16.4 B+-tree file organization

- leaf nodes store records

16.5 Bulk loading and bottom-up build

- problem inserting entries one-at-a-time into a B+-tree requires >= 1 IO per entry
- efficient alternative 1 sort entries first
- efficient alternative 2: bottom-up B+-tree construction starting with leaf level

16.6 Multiple-key access

-

```
select ID
from instructor
where dept_name = "Finance" and salary = 80000
```

17 DONE

11.3a 11.4

18 chap12

18.1 Measure of query cost

factors
- disk accesses
- CPU
- number of seeks
- number of blocks read
- number of blocks written

t_T time to transfer one block t_S time for one seek

use worst case estimates

18.2 selectoin operation

18.2.1 file scan

A1 (linear search)
- cost = b_r block transfers + 1 seek b_r denotes number of blocks containing records from relation r

18.2.2 index scan

A2 (primary index, equality on key)
- retrieve a single record taht satisfy the corresponding equality condition
- cost $= (h_i + 1) * (t_T + t_s)$

A3 (primary index, equality on nonkey)

A4 (secondary index, equality on nonkey)
- retrieve a single record if the search-key is a candidate key
- cost $= (h_i + 1) * (t_T + t_S)$
- retrieve multiple records if search-key is not a candidate key

18.2.3 selection involving comparisons

- can implement selections of the form $\sigma_{A \leq V}(r)$ or $\sigma_{A \geq V}(r)$

.5 (primary index, comparison)
- relation sorted on A

- A6 (secondary index

18.2.4 implementation of complex selections

- conjunction $\sigma_{\theta_1} \cap \ldots \sigma_{\theta_n}(r)$

ctive selection using one index)
- select a combination of θ_i and algorithms A1 through A7 that results in the least cost for $\sigma_{\theta_i}(r)$

eelction using composite index)

n by intersection of identifiers)

18.3 external sort-merge

18.4 Join

18.4.1 nested-loop join

- $r \bowtie_\theta s$
- worst $n_r \times b_s + b_r$

18.4.2 block nested-loop join

18.4.3 indexed nested-loop join

- for each tuple in the outer relation r, a lookup is performed on the index for s, and the relevant tuples are retrieved.
- in the worst case, b_r I/O operations are needed to read relation r $b_r(t_T + t_S) + n_r \times c$

18.4.4 merge join

- w
    1. sort both relations on their join attribute
    2. merge the sorted relations

18.4.5 hash join

18.5 evaluation of expression

18.5.1 materialization

- evaluate one operation at a time, starting at the lowest-level. Use intermediate results materialized into temporary relations to evaluate next-level operations

double buffering – use two output buffers for each operation, when one is ful

18.5.2 pipeline

- evaluate several operations simultaneously, passing the results of one operation on to the next

19 TODO

12.2 12.3b

20 chap13

20.1 overview

evalutaion plan defines exactly what algorithm is used for each operation and how the execution of the operations is coordinate.

cost-based query optimization

- generate logically equivalent expressions using **equivalence rules**
- annotate resultant expressions to get alternative query plans
- choose the cheapest plan based on **estimated cost**

20.2 transformation

equivalent rules

1. conjunctive selection operations $\sigma_{\theta_1 \cap \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
2. commutative $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
3. only the last in a sequence of projection operations is needed $\prod_{L_1}(\prod_{L_2}(\dots(\prod_{L_n}(E))\dots)) = \prod_{L_1}(E)$
4. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$ $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
5. theta-join operations are commutative
6. associative
7. the selection operation distributes over the theta-join operation under the following two conditions $\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$ $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$
8. $\prod_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\prod_{L_1}(E_1)) \bowtie_\theta (\prod_{L_2}(E_2))$
   - let L_3 are attributes of E_1 that are involved in join condition θ $\prod_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \prod_{L_1 \cup L_2}((\prod_{L_1 \cup L_3(E_1)})) \bowtie_\theta (\prod_{L_2 \cup L_4}(E_2))$
9. $\sigma_P(E_1 - E_2) = \sigma_P(E_1) - \sigma_P(E_2)$