数据库系统原理

邵蓥侠

课程内容

Chapter 1: Introduction

Chapter 2: Relation Model

Chapter 3: SQL

Chapter 4: Intermediate SQL

Chapter 5: Advanced SQL

Chapter 6: Database Design Using the E-R Model

Chapter 7: Relational Database Design

Chapter 8: Complex Data Types

Chapter 9: Application Development

Chapter 13: Data Storage Structures

Chapter 14: Indexing

Chapter 15: Query Processing

Chapter 16: Query Optimization

Chapter 17: Transactions

Chapter 18: Concurrency Control

Chapter 19: Recovery System

Chapter 2: Relation Model

- Relational data model
  - ***relational data structure***
    - basic elements in relational data model, i.e. tables and attributes, and relationships among them
  - ***integrity constraints***
    - constraints on attributes of schemas, e.g. value domain, type
    - constraints on dependencies among attributes of a schema
    - constraints on dependencies among attributes of different schemas
    - key, foreign key, etc.
  - ***operations on the model***
    - Relational algebra

Chapter 2: Relation Model

Symbol (Name)	Example of Use
σ (Selection)选择	$\sigma_{\text{salary} >= 85000}(\textit{instructor})$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)　投影	$\Pi_{\textit{ID, salary}}(\textit{instructor})$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output all pairs of rows from the two input relations whether or not they have the same value on all attributes that have the same name.
∪ (Union)并	$\Pi_{\textit{name}}(\textit{instructor}) \cup \Pi_{\textit{name}}(\textit{student})$
	Output the union of tuples from the *two* input relations.
– (Set Difference)集合差	$\Pi_{\textit{name}}(\textit{instructor}) - \Pi_{\textit{name}}(\textit{student})$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join) 自然连接	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Exercises

employee (person_name, street, city)
works (person_name, company_name, salary)
company (company_name, city)

给出关系代数表达式来表示下列每一个查询：

1）找出居住在“**Miami**”城市的所有员工姓名。

2）找出工资在**100000**美元以上的所有员工姓名。

3）找出居住在“**Miami**”并且工资在**100000**美元以上的所有员工姓名。

1) $\pi_{person\_name}\left(\sigma_{city='Miami'}(employee)\right)$

2) $\pi_{person\_name}\left(\sigma_{salary>100000}(works)\right)$

3) $\pi_{person\_name}\left(\sigma_{city='Miami' \wedge salary>100000}\ (employee \bowtie \text{works})\right)$

Chapter 3, 4, 5: SQL

- SQL includes DDL, DML, etc.
- DDL: Data Definition Language
  - Create, Drop, Alter (on schema)
- Objects of DDL
  - Data type
  - Table
  - Constraints
  - Index
  - … …

Chapter 3, 4, 5: SQL

- SQL includes DDL, DML, etc.
- DDL
  - Create, Drop, Alter
- Objects of DDL
  - Data type
  - Table
  - Constraints
  - Index
  - … …

create table *takes* (
ID **varchar**(5),
course_id **varchar**(8),
sec_id **varchar**(8),
semester **varchar**(6),
year **numeric**(4,0),
grade **varchar**(2),
primary key *(ID, course_id, sec_id, semester, year)* ,
foreign key (*ID*) **references** *student*,
foreign key (*course_id, sec_id, semester, year*)
references *section*);

Chapter 3, 4, 5: SQL

- DML: Data Manipulation Language
  - Select, Insert, Delete, Update
- ***Select***
  - Single Table
  - Join: Natural Join, Inner Join, [Left, Right, Full] Outer Join
  - Aggregate Function: Null Value
  - Subquery (**难点！！！**)
    - With clause
    - Nested clause
- Relational algebra vs. Select

Chapter 3, 4, 5 — Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries, to implement more complex query
- A subquery is a *select-from-group* expression that is nested within another query
  - The nesting can be done in the following SQL query

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

    as follows:
    - A_i can be replaced be a subquery that generates a single value.
    - r_i can be replaced by any valid subquery
    - P can be replaced with an expression of the form:

      B <operation> (subquery)

    Where B is an attribute and <operation> to be defined later.

Chapter 3, 4, 5 — Nested Subqueries

- The subquery is often nested in the *where clause/having clause* , *from clause*
- Subquery in the *where clause/having* *,* perform tests for
  - *set membership → in*
  - *set comparisons → some/all*
  - *set cardinality → empty relations, not exist*
  - *duplicate tuples → unique*
- The *with* clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs
- Subqueries in the *Select* clause
  - Scalar subquery is one which is used where a single value is expected

Use of “not exists” Clause

- Find all students who have taken all *courses* offered in the Biology *department*.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

- First, nested query lists all courses offered in Biology
- Second, nested query lists all courses a particular student took

Note that $X - Y = \emptyset \iff X \subseteq Y$

Note: Cannot write this query using = **all** and its variants

Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

Chapter 3, 4, 5: SQL

- Modification of the Database
  - Insert
  - Delete
  - Update
- Advanced SQL
  - Access from a programming language
    - JDBC, ODBC, ADO, …
    - Embedded SQL

Chapter 6: Database Design and E-R Model

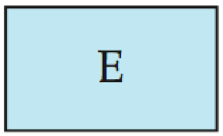
- DB/DBS/DBAS design process
  - DB design phases

    requirement analysis, conceptual design, logical design, physical design
- The Entity-Relationship Model
  - *basic* E-R model
    - modeling elements: entity sets, relationship sets, attributes
    - constraints: mapping cardinality, participation constraint, keys
    - weak entity sets
    - removing redundant attributes in entity sets
    - E-R diagram

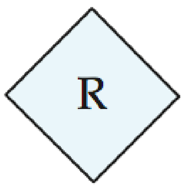
Chapter 6: Database Design and E-R Model

- *extended* E-R Features
  - OO features in E-R model, i.e specialization, generalization, attributes inheritance, constraints on generalization
  - aggregation: relationship among relationships
- Reduction to Relational Schemas
  - mapping elements in E-R model to that in relational models, i.e. conceptual schema → initial logical schema

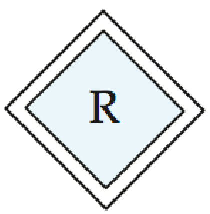
Symbols Frequently Used in E-R Notation



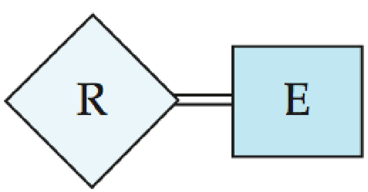
entity set



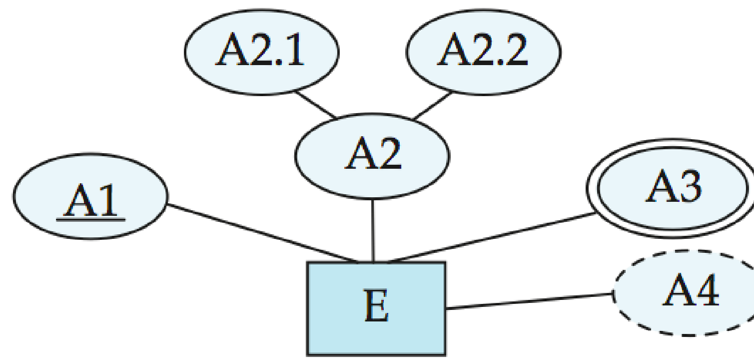
relationship set



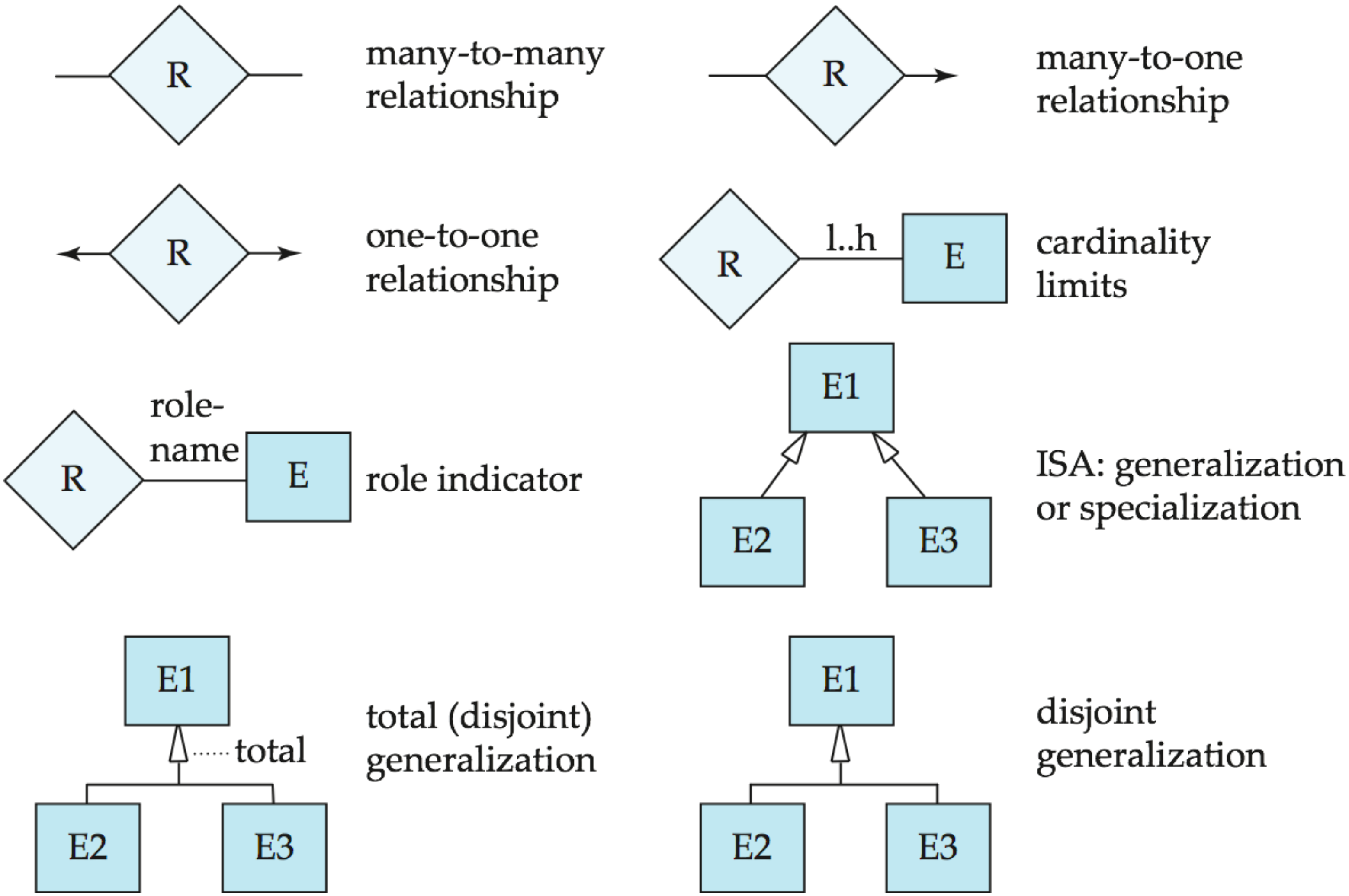
identifying
relationship set
for weak entity set



total participation
of entity set in
relationship



Symbols Frequently Used in E-R Notation



Chapter 6: Database Design and E-R Model

- For each *entity set* and *relationship set*, there is a unique table which is assigned the name of the corresponding entity set or relationship set
  - Strong entity set vs. Weak entity set
- Each table has a number of columns; the columns generally corresponding to entities' or relationships' attributes, and have unique names
  - Composite attributes, Multivalued attributes,

Chapter 6: Database Design and E-R Model

- Reduction of a relationship set into tables are ***strongly*** dependent on the *mapping cardinality* constraint and *total/partial constraints* related to this relationship set
- A *many-to-many* relationship set is represented as a table with columns for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set
- Many-to-one and one-to-many relationship sets *that are total on the many-side* can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side

Chapter 7, 8 — Schema Normalization

- Features of Good Relational Design
  - ***Inserting*** problem and ***information redundancy***
  - ***Deleting*** problem
  - ***Updating*** problem and ***information redundancy***
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies

函数依赖保持性

1NF

↓ 消除非主属性对键的部分函数依赖

2NF

↓ 消除非主属性对键的传递函数依赖

3NF

↓ 消除主属性对键的部分和传递函数依赖

BCNF

↓ 消除非平凡且非函数依赖的多值依赖

4NF

↓ 消除非平凡连接依赖

5NF

无损连接性

Chapter 7, 8 — Schema Normalization

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- A relation schema R is in **BCNF** with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
- A relation schema R is in **third normal form** (**3NF**) if for all: $\alpha \rightarrow \beta$ in F^+, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

Chapter 7, 8 — Schema Normalization

- Functional Dependency Theory
- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
- **Transitive dependency** (传递函数依赖)
- **Partial dependency** (部分函数依赖)
- Armstrong's Axioms
- Closure of a Set of Functional Dependencies
- Closure of Attribute Sets
- Canonical Cover

Chapter 7, 8 — Schema Normalization

- Algorithms
  - Closure of a Set of Functional Dependencies
  - Closure of Attribute Sets
  - Canonical Cover
  - Lossless-join Decomposition
  - Dependency Preservation Decomposition
  - BCNF vs. 3NF Decomposition
  - Computing of Candidate Keys

Chapter 13: Storage and File Structure

- File organization
  - FLC vs. VLC
  - Sequential file, Heap file, Hash file, clustering
- Data-dictionary Storage
  - The **Data dictionary** (also called **system catalog**) stores **metadata**
- Data Buffer
  - Buffer-Replacement Policies

Organization of Records in Files

- **Heap** – a record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- **Hashing** – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed
- Records of each relation may be stored in a separate file. In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O

Chapter 14: Indexing and Hashing

- Basic Concepts
- Ordered Indices
- B^+-Tree Index
- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Index Definition in SQL
- Multiple-Key Access

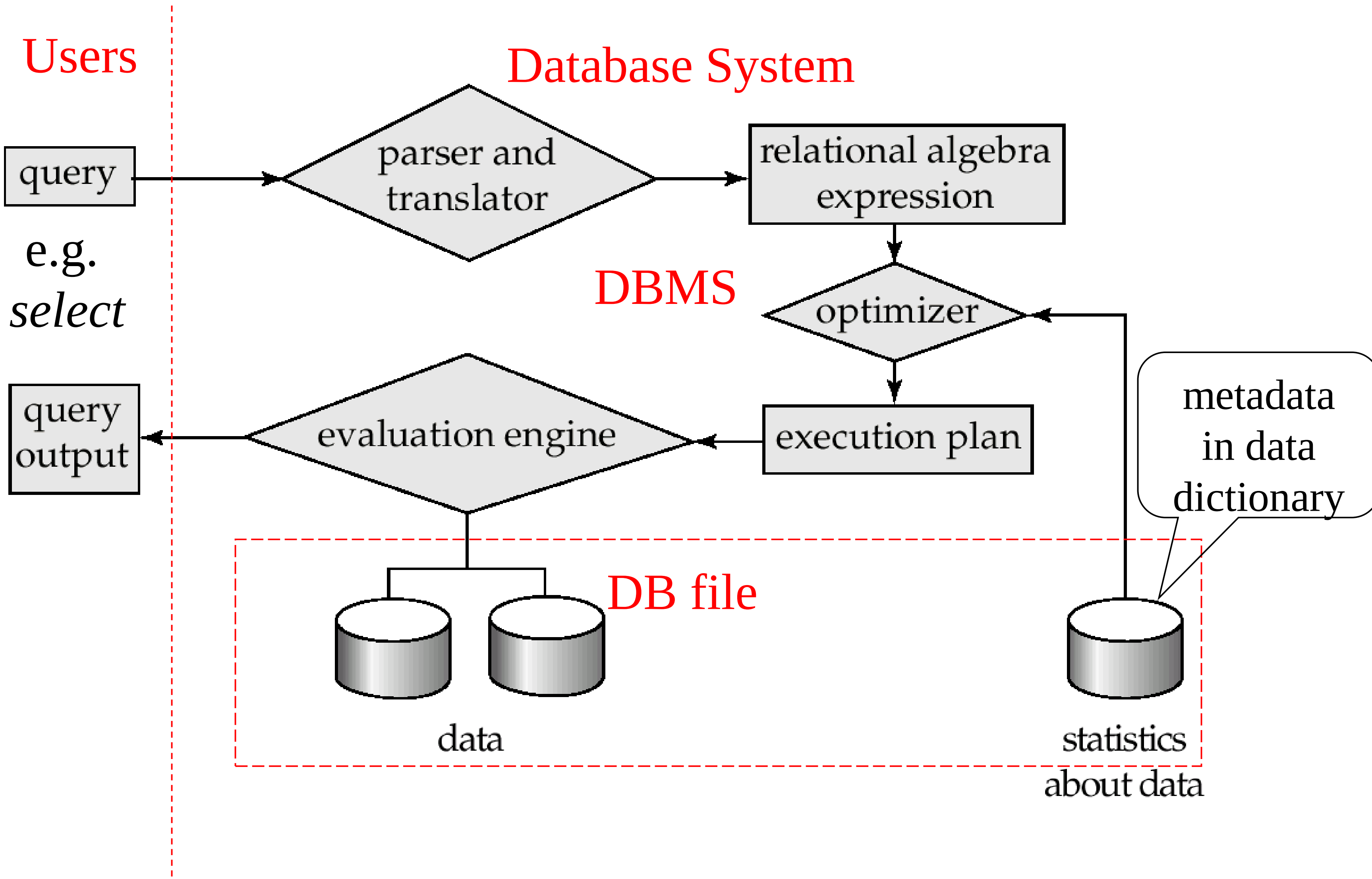
Chapter 14: Indexing and Hashing

- Basic concepts and classification of **indexing**
  - ordered indices, hash indices
- Properties/types of ***ordered*** indices
  - primary/clustering indices, secondary/non-clustering indices
  - dense indices, sparse indices
  - single-level indices, multi-level indices (e.g. B^+-tree, B-tree)
- ***Hash*** indices
  - hash functions
  - static hash, dynamic hash

Chapter 15: Query Processing

- Basic steps in query processing
- Measures of Query Cost
  - For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
- Selection Operation
  - A1-A10 file scan algorithms
- Sorting, Join Operation, Other Operations
- Evaluation of Expressions

1. Parsing and translation
2. Optimization
3. Evaluation



	Algorithm	Cost	Reason
A1	Linear Search	$t_S + b_r * t_T$	One initial seek plus b_r block transfers, where b_r denotes the number of blocks in the file.
A1	Linear Search, Equality on Key	Average case $t_S + (b_r/2) * t_T$	Since at most one record satisfies the condition, scan can be terminated as soon as the required record is found. In the worst case, b_r block transfers are still required.
A2	Clustering B$^+$-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	(Where h_i denotes the height of the index.) Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer.
A3	Clustering B$^+$-tree Index, Equality on Non-key	$h_i * (t_T + t_S) + t_S + b * t_T$	One seek for each level of the tree, one seek for the first block. Here b is the number of blocks containing records with the specified search key, all of which are read. These blocks are leaf blocks assumed to be stored sequentially (since it is a clustering index) and don't require additional seeks.
A4	Secondary B$^+$-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	This case is similar to clustering index.
A4	Secondary B$^+$-tree Index, Equality on Non-key	$(h_i + n) * (t_T + t_S)$	(Where n is the number of records fetched.) Here, cost of index traversal is the same as for A3, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if n is large.
A5	Clustering B$^+$-tree Index, Comparison	$h_i * (t_T + t_S) + t_S + b * t_T$	Identical to the case of A3, equality on non-key.
A6	Secondary B$^+$-tree Index, Comparison	$(h_i + n) * (t_T + t_S)$	Identical to the case of A4, equality on non-key.

Figure 15.3 Cost estimates for selection algorithms.

Chapter 16: Query Optimization

- Transformation of relational expressions
  - Rule 1- Rule 11: equivalence rules
- Choice of evaluation plans — Query optimization
  - cost-based optimization
  - ***heuristic optimization***

Example Three

- Consider the following relations in banking enterprise database, where the primary keys are underlined
  - *branch (branch-name, branch-city, assets),*
  - *loan (loan-number, branch-name , amount)*
  - *borrower(customer-name, loan-number, borrow-date)*
  - *customer (customer-name, customer-street, customer-city)*
  - *account (account-number, branch-name, balance)*
  - *depositor (customer-name, account-number , deposit-date)*

Example Three (cont.)

- For the query “ Find the ***names*** of all ***customers*** who have an ***loan*** at any ***branch*** that is located in ***Brooklyn*** and have ***assets*** more than $100,000, requiring that ***loan-amount*** is less than $1000”
  - give an SQL statement for this query
  - given a initial query tree for the query, and convert it into an optimized query tree by means of heuristic optimization

Example Three (cont.)

- SQL

  select *customer-name*

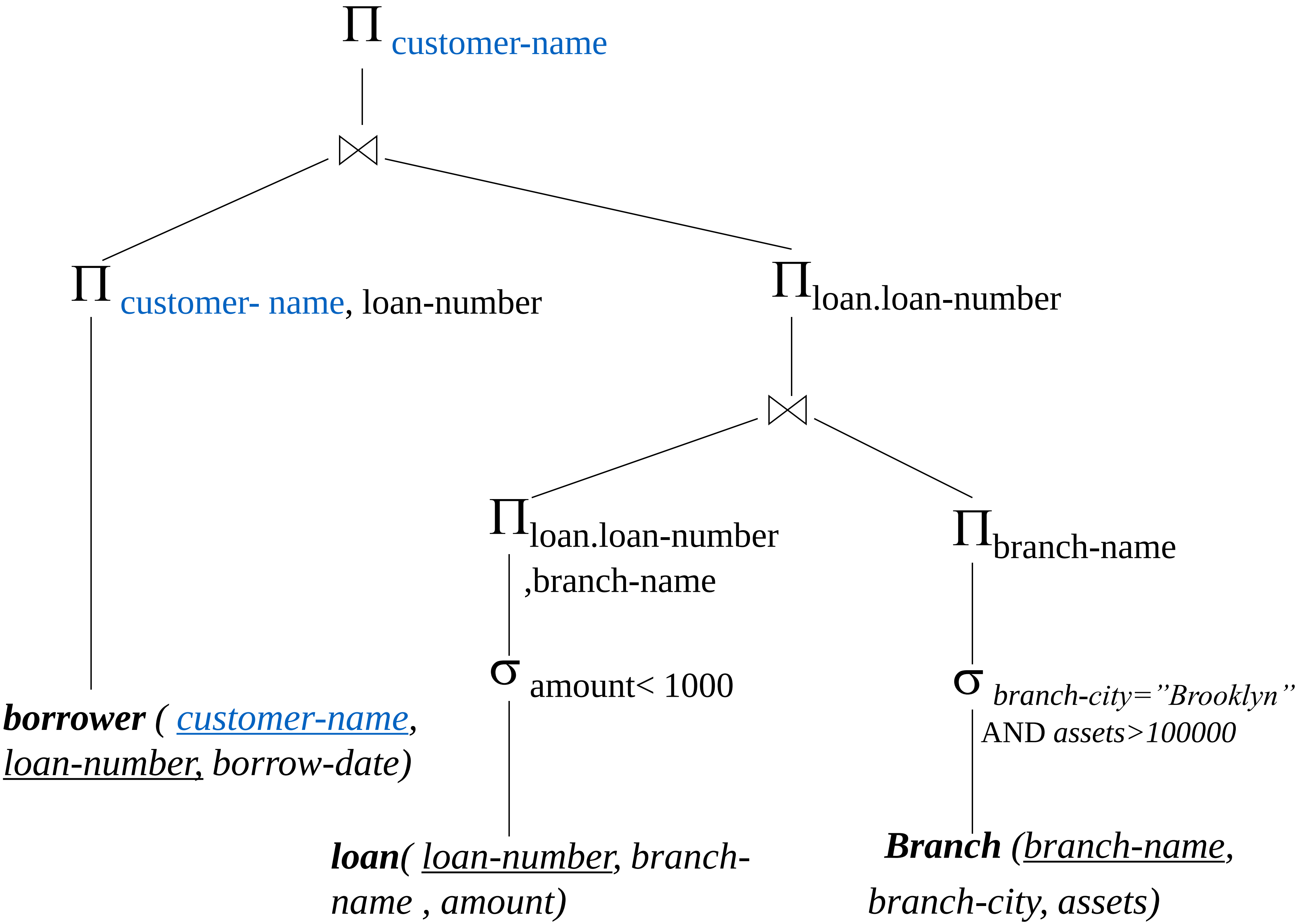
  from *borrower*, *loan*, *branch*

  where *loan.loan-number=borrower.loan-number*

  and *branch.branch-name=loan.branch-name*

  and *branch-city="Brooklyn"*

  and *assets>100000 and amount<1000*



Π customer-name
⋈
Π customer- name, loan-number
Π loan.loan-number
⋈
Π loan.loan-number ,branch-name
Π branch-name
σ amount< 1000
σ branch-city=”Brooklyn” AND assets>100000
borrower (customer-name, loan-number, borrow-date)
loan(loan-number, branch-name , amount)
Branch (branch-name, branch-city, assets)

Chapter 17: Transaction

- The concept of ***transaction***
- The properties of ***transaction* -- ACID**
  - **原子性** (atomicity)
  - **一致性** (consistency)
  - **独立性/隔离性** (isolation)
  - **永久性/ 持续性/操作结果永久保持性** (durability)
- The relationship between ACID and the DBS component
  - Concurrency control
  - Recovery management

Chapter 17: Transaction

- What is schedule of transaction?
- Serializability
  - Serializable schedule vs. serial schedule
  - Conflict serializability
  - Recoverable serializability
  - Cascadeless serializability
- Conflict equivalence and conflict serializability
  - How to test? → precedence graph
  - How to construct? → Topology sort

Chapter 18: Concurrency Control

- Lock-based Protocol
  - X-Lock, S-Lock
- 2 Phase Locking (2PL)
  - How to construct?
  - Strict 2PL → cascadeless and serializable schedules
  - Rigorous 2PL

复习建议

- 掌握基本概念
- 理解各类例子
- 运用算法进行推演和计算
- 复习作业题

预祝各位同学
期末顺利！

2021-12-27 19:30 教二401