A Short Course for Relational Databases

Advanced Data Engineering

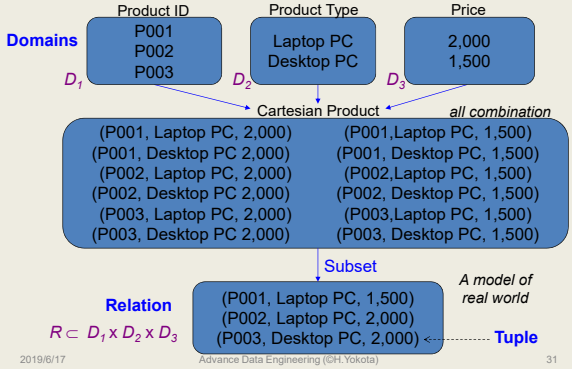
A Brief Introduction of Relational DB

- The **Relational Model**
  - Proposed by E.F. Codd (IBM) in 1970
    - 1981 ACM Turing Award
    - 2003.4.18 Died (78 years old)
  - Based on the Set Theory
- Basic Concept
  - A **relation** R is a subset of Cartesian product of **domain** D_j ($1 < j < n$)

$$R \subset D_1 \times D_2 \times \ldots \times D_j \times \ldots \times D_n$$

  - A **tuple** t is an element of a relation: $t \in R$

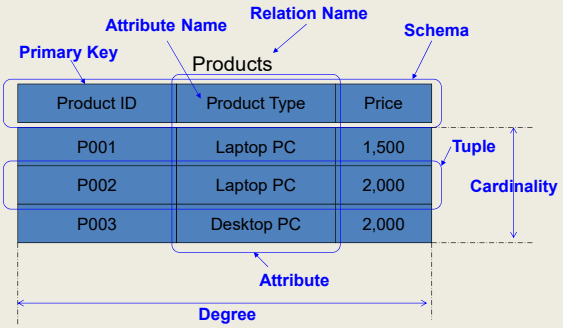
An Illustration of the Model



Terminologies of Relational DB

- The number of tuples is called **cardinality**
- A domain in a relation is called an **attribute**
  - Each attribute has an **attribute name**
  - The collection attribute names is called a **schema**
  - *n* is called a **degree** of the relation
- A set of attributes which uniquely identifies each tuple in a relation is called a **key** of the relation
  - There are some **candidate keys**
  - A **primary key** and other **alternate keys**
- A relation can be seen as a table.
  - A tuple is a row of the table.
  - An attribute is a column of the table.

A Table Image of RDB



Relational Database Operations

- Relational Algebra
  - A collection of operators
    - take relations as their operands and return relations as their results
- Relational Calculus
  - Specification of requiring results
    - An example: $\{ (t_r, t_s) \mid R(t_r) \wedge S(t_s) \wedge t_r[X] \; \theta \; t_s[Y] \}$
- Algebra is *prescriptive* while Calculus is *descriptive*
  - Both expressive power is identical

Relational Algebra Operations

- Set Operations
  - Union ($R \cup S$)
  - Intersection ($R \cap S$)
  - Difference ($R - S$)
  - Cartesian Product ($R \times S$)
- Dedicated Relational Algebra Operations
  - Projection ($\pi_{\text{attribute-list}}\ R$)
  - Selection ($\sigma_{\text{conditions}}\ R$)
  - Join (θ -Join: $R \bowtie_{r \theta s} S$, eq-Join: $R \bowtie_{r=s} S$)
  - Division ($R \div S$)

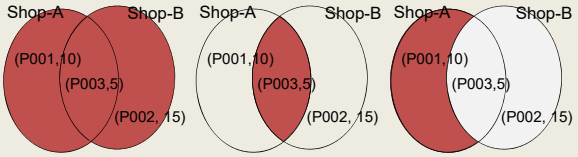
Set Operations on RDB

Shop-A

Product ID	Stocks
P001	10
P003	5

Shop-B

Product ID	Stocks
P002	15
P003	5



Shop-A ∪ Shop-B
Union

Shop-A ∩ Shop-B
Intersection

Shop-A – Shop-B
Difference

Projection π

- Derive specified attributes with eliminating duplication

Product ID	Product Type	Price
P001	Laptop PC	1,500
P002	Laptop PC	2,000
P003	Desktop PC	2,000

Product Type
Laptop PC
Desktop PC

$\pi_{\text{ProductType}}$ Products = { t[ProductsType] | Products(t) }

* Delta Projection: without eliminating duplication

Selection σ

- Derive all tuples satisfying specified conditions

Product ID	Product Type	Price
P001	Laptop PC	1,500
P002	Laptop PC	2,000
P003	Desktop PC	2,000

Product ID	Product Type	Price
P002	Laptop PC	2,000
P003	Desktop PC	2,000

$\sigma_{Price > 1,800}$ Products = { t | Products(t) ∧ t[Price] > 1,800}

Join ⋈

- Combine two relations with specified conditions

Product ID	Product Type	Price
P001	Laptop PC	1,500
P002	Laptop PC	2,000
P003	Desktop PC	2,000

⋈

Product ID	Stocks
P001	10
P003	5

Product ID	Product Type	Price	Stocks
P001	Laptop PC	1,500	10
P003	Desktop PC	2,000	5

Products $\bowtie_{ProductID = ProductID}$ ShopA

= { (t1, t2) | Products(t1) ∧ ShopA(t2) ∧ t1[ProductsID]=t2[ProductID]}

Query Languages

- SQL (Structured Query Language)
  - Developed in SystemR Project of IBM
  - Standardlized (ANSI/ISO, JIS)
  - Data Definition Language (DDL) + Data Manipulation Language (DML)
  - Directed or Embedded SQL
    - From some programming languages
- Basic Syntax (DML)

  SELECT List of Attribute Names
  FROM List of Relation Names
  WHERE Conditions

SQL Examples (1)

- SELECT DISTINCT Product-Type
  FROM Products
  - Projection for the Products Table
    - $\pi_{\text{Product-Type}}$ Products
    - { t[Product-Type] | Products(t) }
- SELECT Product-Type
  FROM Products
  - Delta Projection
    - Without eliminating duplication

SQL Examples (2)

- SELECT Product-ID, Product-Type, Price
  FROM Products
  WHERE Price > 1,800
  - Selection for the Products Table
    - $\sigma_{\text{Price} > 1,800}$ Products
    - { t | Products(t) $\wedge$ t[Price] > 1,800}
- SELECT *
  FROM Products
  WHERE Price > 1,800
  AND Product-Type = “Laptop PC”

SQL Examples (3)

- SELECT *
  FROM Products, Shop-A
  WHERE Products.Product-ID=Shop-A.Product-ID
  - Join between the Products and Shop-A Tables
    - Products $\bowtie_{\text{Product-ID = Product-ID}}$ Shop-A
    - { (t1, t2) | Products(t1) $\wedge$ ShopA(t2) $\wedge$ t1[ProductsID]=t2[ProductID]}
- SELECT *
  FROM Products
  WHERE Product-ID IN
  (SELECT Product-ID FROM Shop-A)

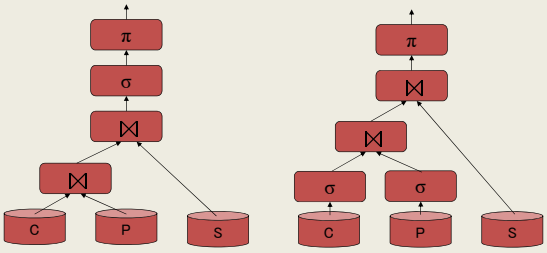
A Query Example

- Consider another relation "Categories" having two attributes: "Product-Type" and "Category"
  - Laptop PCs and Desktop PCs are categorized as PCs in the table
- Write an SQL query to derive stocks of PC sold in the Shop-A at the price higher than $1,800
- Consider Relational Algebra Expressions for the query

```
SELECT Stocks
FROM   Shop-A
WHERE Product-ID IN
 (SELECT Product-ID
  FROM   Products
  WHERE  Price > 1,800
  AND  Product-Type IN
   (SELECT  Product-Type
    FROM   Categories
    WHERE Category = PC))
```


Examples of Query Trees

$\pi(\sigma((C \bowtie P) \bowtie S))$ $\pi((\sigma(C) \bowtie \sigma(P)) \bowtie S)$



C: Categories, P: Products, S: Shop-A

Aggregate Functions (1)

- Count
  - SELECT COUNT(*)
    FROM Products
  - SELECT DISTINCT COUNT(Product-Type)
    FROM Products
- Summation
  - SELECT SUM(Stocks)
    FROM Shop-A, Products
    WHERE Shop-A.Product-ID=Products.Product-ID
    AND Products.Price > 1,800
- Other Functions: AVG(), MAX(), MIN()

Aggregate Functions (2)

- Group By
  - SELECT Product-Type, AVG(Price)
    FROM Products
    GROUP BY Product-Type

Product Type	AVG(Price)
Laptop PC	1,750
Desktop PC	2,000

- Having
  - SELECT Product-Type
    FROM Products
    GROUP BY Product-Type
    HAVING AVG(Price) > 1,800

Access Methods

- Sequential access vs. Direct access
- Indexing
  - To speed up retrieval
- Inverted File
  - Imagine index pages of a book
- Tree structured Index: B-trees
  - Common: most relational system support B-trees
  - B+tree support both direct & sequential accesses
- Hashing
  - Direct access by using a Hash function

Transaction

- A logic unit of work having ACID properties.
  - A: Atomicity
    - All-or-nothing (Commit: all, Rollback: nothing)
  - C: Consistency
    - A transaction transforms a consistent state of the database into another consistent state, without necessarily preserving consistency at all intermediate points.
  - I: Isolation
    - Transactions are isolated from one another. Any given transaction's update are concealed from all the rest transactions running concurrently, until that transaction commits.
  - D: Durability
    - Once a transaction commits, its update survive, even if there is a subsequent system crash.

Serializability

- Transactions should behave as if they were run **serially**
  - Even though their execution may overlap in time
- **Concurrency control** is required
  - Using Locks
    - Two Phase Lock (2PL) is essential
    - Deadlocks can be occurred
  - Using Timestamp
- We will consider distributed lock/commit

Recovery

- Transaction Recovery
  - Rollback by the user, or for serializability
  - States are kept on volatile memory
- System Recovery
  - Caused by system failures
  - States are kept on non volatile memory
  - Recovered by checkpoint and logs
- Media Recovery
  - Caused by media failures

Transaction control in SQL

- BEGIN WORK
  - Declaration of starting a transaction
- COMMIT WORK
  - Declaration of terminating a normal transaction
- ROLLBACK WORK
  - Abortion of a transaction
- We will also consider other controls for extended transaction models

OLTP / OLAP

- OLTP: On Line Transaction Processing
  - Typically handling simultaneous fixed form queries for data entry and retrieval
    - Banking system, Trading system, POS system, etc.
- OLAP: On Line Analytical Processing
  - Handling ad hoc complex queries for strategic decision support of managers

Recent Trend

- ACID properties are too strong for scalability
  - Some applications does not require strong consistency
    - Such as Twitter, Facebook, etc.
    - How about Amazon ?
- BASE transaction model (for KVS)
  - Basically Available
  - Soft state
  - Eventually consistent