Databases 2022

Darko Bozhinoski,
Ph.D. in Computer Science

Agenda

- Database Normalization
- Query Optimization

Informal and Formal Design Guidelines for Relational Databases

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms
  - *1NF (First Normal Form)*
  - *2NF (Second Normal Form)*
  - *3NF (Third Normal Form)*
  - *BCNF (Boyce-Codd Normal Form)*

Informal Guidelines for Relational Databases

Guideline 1: Clear Semantics of the Relation Attributes
Guideline 2: Few redundant Information in Tuples and does not suffer Update Anomalies
Guideline 3: Few NULL Values in tuples
Guideline 4: Satisfy the lossless join condition (no spurious tuples)

A **lossless join decomposition is a decomposition of a relation R into relations R1 and R2 such that a natural join of the two smaller relations yields back the original relation**.

Functional Dependencies (1)

- Functional dependencies (FDs)
  - **A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y**
- There are two important properties of decompositions:
  - **Non-additive or losslessness of the corresponding join**
  - **Preservation of the functional dependencies.**

Inference Rules for FDs

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y
  - IR2. (**Augmentation**) If X -> Y, then XZ -> YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

Inference Rules for FDs (2)

- Some additional inference rules that are useful:
  - **Decomposition:** If X -> YZ, then X -> Y and X -> Z
  - **Union:** If X -> Y and X -> Z, then X -> YZ
  - **Psuedotransitivity:** If X -> Y and WY -> Z, then WX -> Z

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Inference Rules for FDs

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F

- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X

- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$
- Definition (**Covers**):
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if G^+ *subset-of* F^+)
- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Normalization of Relations

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Normalization of Relations (2)

- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- 4NF
  - based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF, BCNF or 4NF)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Keys and Attributes Participating in Keys

- A **superkey** of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

Keys and Attributes Participating in Keys (2)

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

First Normal Form

- Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**; attributes whose values for an *individual tuple* are non-atomic

Normalization into 1NF

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Normalization nested relations into 1NF

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

<u>Ssn</u>	Ename

EMP_PROJ2

<u>Ssn</u>	<u>Pnumber</u>	Hours

Figure 10.9

Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Second Normal Form

- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Full functional dependency:** a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold
  - {SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

Second Normal Form (2)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

- R can be decomposed into 2NF relations via the process of 2NF normalization

Third Normal Form

- Definition:
  - **Transitive functional dependency:** a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z
- Examples:
  - SSN -> DMGRSSN is a **transitive** FD
    - Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold
  - SSN -> ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN -> X and X -> ENAME

Third Normal Form (2)

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.
  - When Y is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary).
    - Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

Normalizing into 2NF and 3NF

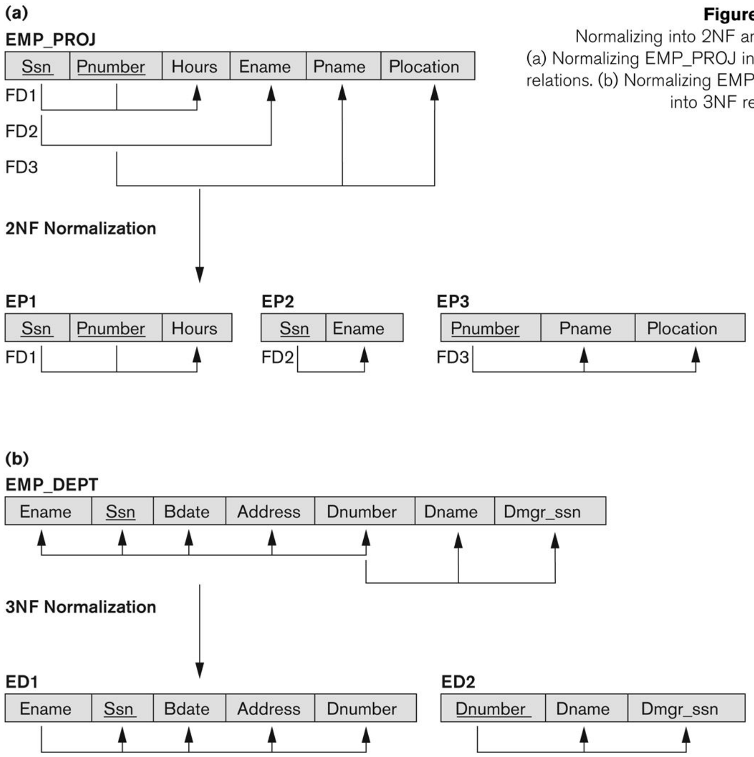


Figure 10.10
Normalizing into 2NF and 3NF.
(a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Normalization into 2NF and 3NF

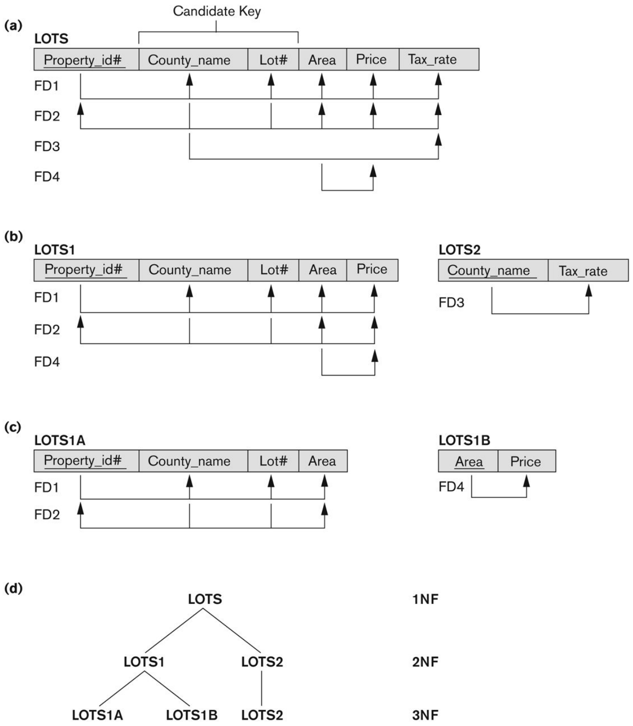


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

Table 14.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

General Normal Form Definitions (For Multiple Keys)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on *every* key of R

General Normal Form Definitions (2)

- Definition:
  - **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
  - A relation schema R is in **third normal form (3NF)** if whenever a FD X -> A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R
- NOTE: Boyce-Codd normal form disallows condition (b) above

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD X -> A** holds in R, then **X is a superkey** of R
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- The goal is to have each relation in BCNF (or 3NF)

Boyce-Codd normal form

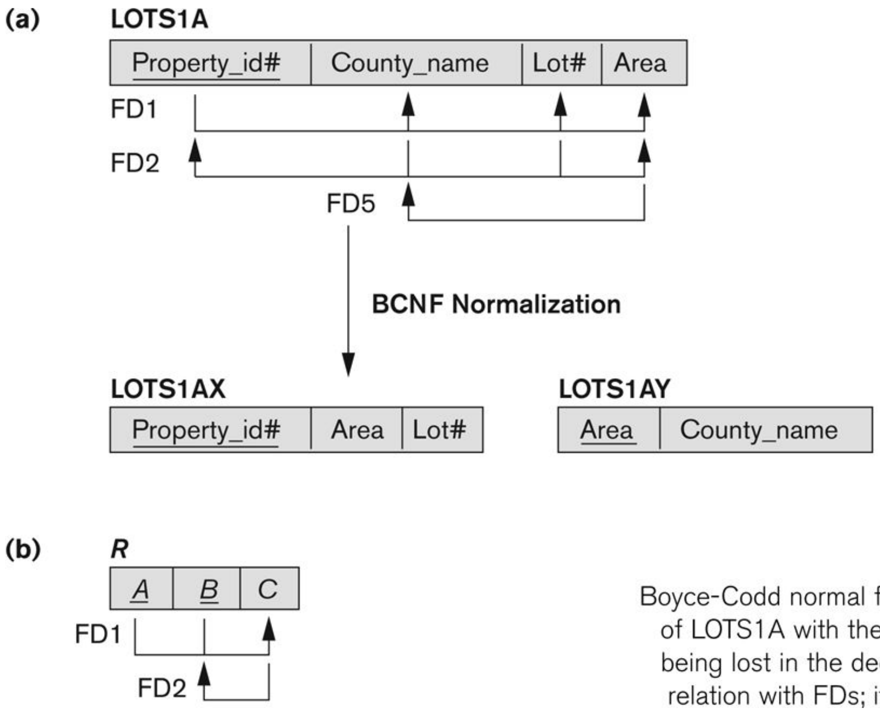


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

A relation TEACH that is in 3NF but not in BCNF

Achieving the BCNF by Decomposition

- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course
- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern.
  - So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
  - {student, instructor} and {student, course}
  - {course, instructor } and {course, student}
  - {instructor, course } and {instructor, student}
- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).

Achieving the BCNF by Decomposition (2)

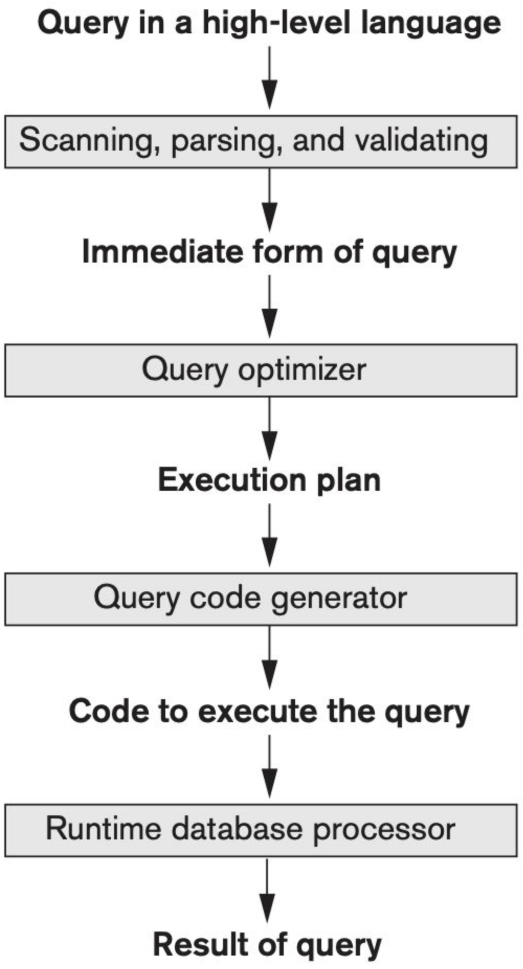
- Three possible decompositions for relation TEACH
  - {<u>student, instructor</u>} and {<u>student, course</u>}
  - {course, <u>instructor</u> } and {<u>course, student</u>}
  - {<u>instructor</u>, course } and {<u>instructor, student</u>}
- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).

Query Optimization

Basic Concepts

- ❖ **Query Processing** – activities involved in retrieving data from the database: – SQL query translation into low-level language implementing relational algebra – Query execution
- ❖ **Query Optimization** – selection of an efficient query execution plan

Query Processing Steps

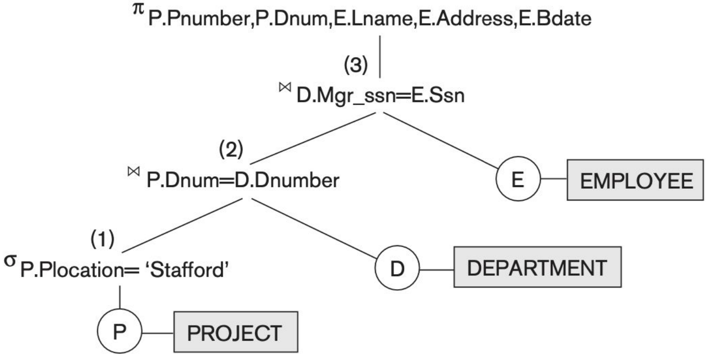


1. **Scanned**
   - It identifies the query tokens - such as SQL keywords, attribute names, and relation names - that appear in the text of the query
2. **Parsed**
   - It checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language
3. **Validated**
   - It checks whether all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried
4. **An internal representation of the query is then created**, usually as a tree data structure called a query tree
5. The DBMS must then **devise an execution strategy** (query optimization) or query plan

SQL Queries → Relational Algebra

In practice, SQL is the query language that is used in most commercial RDBMSs

- ❖ An SQL query is first translated into an equivalent extended relational algebra expression - represented as a query tree data structure - that is then optimized
- ❖ Typically, SQL queries are decomposed into query blocks, which form the basic units that can be translated into the algebraic operators and optimized



Query Decomposition

- ❖ Analysis:
  - ➢ Relational algebra tree
- ❖ Normalization
- ❖ Semantic analysis
- ❖ Simplification
- ❖ Query restructuring

Analysis

- Analyze query using compiler techniques
- Verify that relations and attributes exist
- Verify that operations are appropriate for object type
- Transform the query into some internal representation

Normalization

- ❖ Arbitrary complex qualification condition can be converted into one of the normal forms
- ❖ Conjunctive normal form – a sequence of boolean expressions connected by conjunction (AND):
  - – Each expression contains terms of comparison operators connected by disjunctions (OR)
- ❖ Disjunctive normal form – a sequence of boolean expressions connected by disjunction (OR):
  - – Each expression contains terms of comparison operators connected by conjunction (AND)

Semantic Analysis

- Applied to normalized queries
- Rejects contradictory queries:
  - – Qualification condition cannot be satisfied by any tuple Semantic Analysis
- Rejects incorrectly formulated queries:
  - – Condition components do not contribute to generation of the result.

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```

```
SELECT * FROM A WHERE 1 = 1;
```

Example - Semantic Analysis

Join Elimination

Ignoring Projections

```
SELECT * FROM A AS A1
 WHERE EXISTS(SELECT * FROM A AS A2
              WHERE A1.id = A2.id);
```

J

```
SELECT * FROM A;
```

Example - Semantic Analysis (2)

Ignoring Projections

```
SELECT * FROM A AS A1
 WHERE EXISTS(SELECT * FROM A AS A2
                  WHERE A1.id = A2.id);
```

Simplification

- Eliminates redundancy in qualification
- Transform query to equivalent efficiently computed form
- Main tool – rules of boolean algebra

Example

Ignoring Projections

```
SELECT * FROM A AS A1
 WHERE EXISTS(SELECT * FROM A AS A2
               WHERE A1.id = A2.id);
```

SELECT * FROM A

Rules of Boolean Algebra

$$p \wedge (p) \equiv p$$

$$p \vee (p) \equiv p$$

$$p \wedge false \equiv false$$

$$p \vee false \equiv p$$

$$p \wedge true \equiv p$$

$$p \vee true \equiv true$$

$$p \wedge (\neg p) \equiv false$$

$$p \vee (\neg p) \equiv true$$

$$p \wedge (p \vee q) \equiv p$$

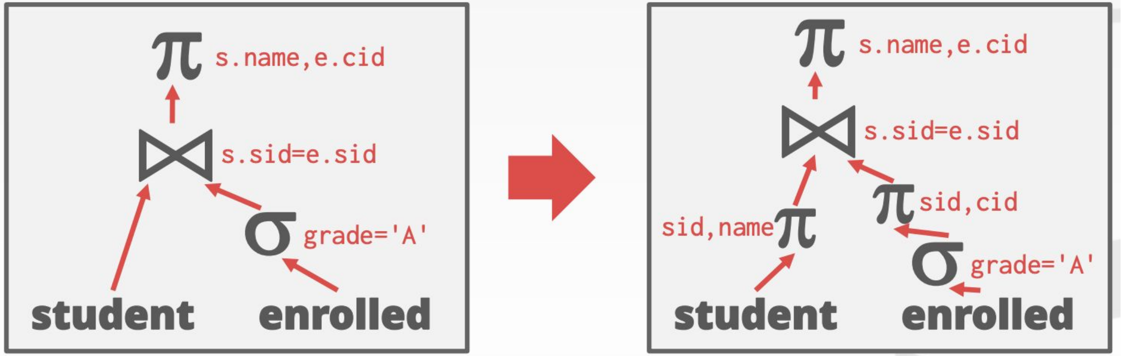
$$p \vee (p \wedge q) \equiv p$$

Query Restructuring

- Rewriting a query using relational algebra operations
- Modifying relational algebra expression to provide more efficient implementation
- Two relational algebra expressions are equivalent if they generate the same set of tuples.
- The DBMS can identify better query plans without a cost model.

Example: Projection Pushdown

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```



Query Optimization

- **Heuristics:**
  - Rewrite the query to remove stupid / inefficient things.
  - Does not require a cost model.

- **Cost-based Search:**
  - Use a cost model to evaluate multiple equivalent plans and pick the one with the lowest cost.

Optimization Criteria

- ❖ Reduce total execution time of the query:
  - Minimize the sum of the execution times of all individual operations
  - Reduce the number of disk accesses
- ❖ Reduce response time of the query:
  - Maximize parallel operations

Heuristic Approach

- ❖ Heuristic - problem-solving by experimental methods
- ❖ Applying general rules to choose the most appropriate internal query representation
- ❖ Based on transformation rules for relational algebra operations

Transformation Rules

1. **Cascade of σ.** A conjunctive selection condition can be broken up into a cascade (that is, a sequence) of individual σ operations:

$$\sigma_{c_1 \text{ AND } c_2 \text{ AND } \ldots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$$

2. **Commutativity of σ.** The σ operation is commutative:

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

3. **Cascade of π.** In a cascade (sequence) of π operations, all but the last one can be ignored:

$$\pi_{\text{List}_1}(\pi_{\text{List}_2}(\ldots(\pi_{\text{List}_n}(R))\ldots)) \equiv \pi_{\text{List}_1}(R)$$

4. **Commuting σ with π.** If the selection condition c involves only those attributes $A_1, \ldots, A_n$ in the projection list, the two operations can be commuted:

$$\pi_{A_1, A_2, \ldots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \ldots, A_n}(R))$$

5. **Commutativity of ⋈ (and ×).** The join operation is commutative, as is the × operation:

$$R \bowtie_c S \equiv S \bowtie_c R$$
$$R \times S \equiv S \times R$$

Notice that although the order of attributes may not be the same in the relations resulting from the two joins (or two Cartesian products), the *meaning* is the same because the order of attributes is not important in the alternative definition of relation.

Transformation Rules (2)

6. **Commuting σ with ⋈ (or ×).** If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, R—the two operations can be commuted as follows:

   $$\sigma_c (R \bowtie S) \equiv (\sigma_c (R)) \bowtie S$$

   Alternatively, if the selection condition c can be written as (c_1 AND c_2), where condition c_1 involves only the attributes of R and condition c_2 involves only the attributes of S, the operations commute as follows:

   $$\sigma_c (R \bowtie S) \equiv (\sigma_{c_1} (R)) \bowtie (\sigma_{c_2} (S))$$

   The same rules apply if the ⋈ is replaced by a × operation.

7. **Commuting π with ⋈ (or ×).** Suppose that the projection list is $L = \{A_1, \ldots, A_n, B_1, \ldots, B_m\}$, where $A_1, \ldots, A_n$ are attributes of R and $B_1, \ldots, B_m$ are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

   $$\pi_L (R \bowtie_c S) \equiv (\pi_{A_1, \ldots, A_n} (R)) \bowtie_c (\pi_{B_1, \ldots, B_m} (S))$$

Transformation Rules

8. **Commutativity of set operations.** The set operations $\cup$ and $\cap$ are commutative, but $-$ is not.

9. **Associativity of $\bowtie$, $\times$, $\cup$, and $\cap$.** These four operations are individually associative; that is, if both occurrences of θ stand for the same operation that is any one of these four operations (throughout the expression), we have:

   $(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$

10. **Commuting σ with set operations.** The σ operation commutes with $\cup$, $\cap$, and $-$. If θ stands for any one of these three operations (throughout the expression), we have:

    $\sigma_c\ (R\ \theta\ S) \equiv (\sigma_c\ (R))\ \theta\ (\sigma_c\ (S))$

11. **The π operation commutes with $\cup$.**

    $\pi_{\mathrm{L}}\ (R \cup S) \equiv (\pi_L\ (R)) \cup (\pi_L\ (S))$

12. **Converting a $(\sigma, \times)$ sequence into $\bowtie$.** If the condition c of a σ that follows a $\times$ corresponds to a join condition, convert the $(\sigma, \times)$ sequence into a $\bowtie$ as follows:

    $(\sigma_c\ (R \times S)) \equiv (R \bowtie_c S)$

Heuristic Rules

- ❖ Perform selection as early as possible
- ❖ Combine Cross product with a subsequent selection
- ❖ Rearrange base relations so that the most restrictive selection is executed first.
- ❖ Perform projection as early as possible
- ❖ Compute common expressions once.

Cost Estimation Components

- ❖ Cost of access to secondary storage
- ❖ Storage cost – cost of storing intermediate results
- ❖ Computation cost
- ❖ Memory usage cost – usage of RAM buffers

Cost Estimation for Relational Algebra Expressions

- ❖ Formulae for cost estimation of each operation
- ❖ Estimation of relational algebra expression
- ❖ Choosing the expression with the lowest cost

Cost Estimation in Query Optimization

- ❖ Based on relational algebra tree
- ❖ For each node in the tree the estimation is to be done for:

  – the cost of performing the operation;

  – the size of the result of the operation;

  – whether the result is sorted.

Indexes

- A data structure that allows the DBMS to locate particular records
- Index files are not required but very helpful
- Index files can be ordered by the values of indexing fields

<u>If no index is available, should we create one to improve performance on the query?</u>

Indexes

- A data structure that allows the DBMS to locate particular records
- Index files are not required but very helpful
- Index files can be ordered by the values of indexing fields

<u>If no index is available, should we create one to improve performance on the query?</u>

Is this query executed often enough to warrant this change? Indexes improve read speeds on queries, but will reduce write speeds, so we should add them with caution.

Retrieval Algorithms

- Files without indexes:
  - Records are selected by scanning data files
- Indexed files:
  - Matching selection condition
  - Records are selected by scanning index files and finding corresponding blocks in data files

Search Space

- Collection of possible execution strategies for a query:
- Strategies can use:
  - Different join ordering
  - Different selection methods
  - Different join methods
- Enumeration algorithm – an algorithm to determine an optimal strategy from the search space

Reading Material

- Fundamentals of Database Systems. Ramez Elmasri and Shamkant B. Navathe. Pearson. **Chapter 14, Chapter 15, Chapter 18, Chapter 19.**


Q & A