DB Systems, Lecture 01

Data: facts that can be recorded
Information: data with meaning
Knowledge: information and its application
Mini-World: the part of the real world we are interested in
Database: a collection of related data
Database Management System: software to help create/maintain databases
Database System: DBMS + Dbs
Meta Data: structural information about a DB (usually a DB itself)

User → Queries → DB → Info

DBMS functionality:
- define a DB (data types, structures, constraints)
- construct the DB on secondary storage
- manipulate the DB (queries, insert, delete, update, …)
- share between concurrent users (keeping consistency)

Database Administrators
Database Designers
End Users (casual/naive/sophisticated/stand-alone)

Data Model: concepts to describe structure of a database, operations on it and constraints
Structure: elements and data types, groups of elements (entity, record, table) and relationships amongst them
Constraints: restrictions on valid data to be enforced at all times
Operations: retrievals and updates, including user-defined operations

Conceptual (high-level, semantic)/ Physical (low-level, internal)/ Implementation (representational)

Database Schema: description of a database
Database Instance: the actual data stored in a database at a particular moment in time

=> Avoid redundancy! (storing information multiple times)

Three Schema Architecture
- Internal schema (physical storage)
- Conceptual schema (structure and constraints)
- External schemas (user views)

Data Independence
- Logical Data Independence: capacity to change the conceptual schema without having to change the external schemas and their associated applications
- Physical Data Independence: capacity to change the internal schema without having to change the conceptual schema

- Insulation between programs and data (data independence)
- Control of redundancy
- Data abstraction
- Support of multiple views of the data
- Sharing of data, multi-user transaction processing
- Self-describing nature of a database systems (metadata)

DDL: Data Definition Language (conceptual schema)
DML: Data Manipulation Language (operations on the DB by users)

SQL (Structured Query Language) is both!

1-tier Architecture: combines everything into a single system, centralized site
2-tier Architecture: specialized servers with specialized functions (client – server)
3-tier Architecture: common for web apps (client – webserver - server)

Network Model
Hierarchical Data Model
Relational Model
Object-oriented Models
Data on the Web (mainly XML)

DB Systems, Lecture 02

Relational Model, Relation Schema R(A1, … An), R relation name, A1..n attributes
set of attribute values => domain, attributes are atomic

Domain: logical definition, data-type and format, NULL special value (atomic data values)
Attribute: role of a domain in a relation schema
Tuple: ordered set of values
Relational Instance: r(R) relation r is a subset of the Cartesian product of its domains
Database: set of multiple relations

Constraints: conditions that must be satisfied by all valid relation instances
- Domain constraints (values must be from attribute domain)
- Key constraints (superkey, primary key)
- Entity constraints (primary key not null)
- Referential integrity constraints (foreign key)

Superkey: if K is sufficient to uniquely identify a tuple of each possible relation r
Candidate key: if K is minimal (superkey and not subset of another superkey)
Primary Key: a candidate key chosen as the principal means of identifying tuples within a relation
Foreign Key: attribute that corresponds to the primary key of another relation

Query languages:
- Procedural: how to do it (can be optimized)
- Declarative: what to do (cannot be optimized)

Pure languages:

- Relational algebra
- Relational calculus (tuple, domain)

Relational Algebra

select σ

σp (r), p is selection predicate, conditions connected by ∧ (and), ∨ (or), ¬ (not)

project π

πA1 ,...,Ak (r), result is the relation of k columns obtained by deleting the columns that are not listed explicitly

union ∪

r ∪ s, r and s must have same schema, union of rows

set difference −

r − s, must be union-compatible relations (same arity, compatible), row difference

Cartesian product ×

r × s, attribute names must be disjoint, if not, rename!

rename ρ

rename results, correct name clashes
ρr (A1 ,...,An) (E) changes the relation name to r and the attribute names to A1, ..., Ak

Set intersection ∩

r ∩ s, must be union-compatible relations, equal to r − (r − s)

Theta Join |X|θ

r |X|θ s, θ is boolean condition on the attributes of r and s, join includes all attributes from schema R and all attributes from schema S

Equi Join |X|eq

r |X|eq s, eq is boolean condition that includes equality only

Natural Join |X|

r |X| s, attributes in r and s must be identical, includes all attributes from schema R and all attributes from schema S that do not occur in schema R

Division ÷

r ÷ s, suited for expressing “for all”, result is a relation on schema R − S (all attributes of schema R that are not in schema S), with t ◦ u is the concatenation of tuples t and u

Assignment ←

← provides a convenient way to express complex queries by breaking them up into smaller pieces, by assigning always to a temporary relation variable

Generalized Projection

extends the projection by allowing arithmetic functions to be used

Aggregate Functions and Operations

aggregation function: takes a collection of values and returns a single value as a result, such as avg, min, max, sum, count

aggregation operation: relational algebra
G1, G2 ,...,Gn ϑ F1 (A1), F2 (A2),...,Fn (An) (E) where G is a list of attributes on which to group (can be empty), F is an aggregate function, A is an attribute name

Outer Join (left, right, full)

extension of the join operation that avoids loss of information, computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join, uses null values (unknown/not existing)

Modifications to the database:
- Insertion r ← r ∪ E
- Updating r ← E, r ← π F1, F2, ..., Fi (r)
- Deletion r ← r − E

<u>Relational Calculus</u>

a relational calculus expression creates a new relation, which is specified in terms of variables that range over tuples (in tuple calculus) or attributes (in domain calculus), there is no order of operations, only what the result should contain is specified

Syntax of first order predicate logic:
- logical symbols: ∧, ∨, ¬, ⇒, ∃, ∀, ...
- constant: string, number, ...; ’abc’, 14, ...
- identifier: character sequence starting with a letter
- variable: identifier starting with capital letter; X , Y , ...
- predicate symbol: identifier starting with lower case letter
- build-in predicate symbol: =, <, >, ≤, ≥, =, ...
- term: constant, variable
- atom: predicate, built-in predicate; p(t1 , ..., tn); predicate symbol p
- formula: atom, A ∧ B, A ∨ B, ¬A, A ⇒ B, ∃XA, ∀XA, (A), ...

Domain Independence: only expressions that permit sensible answers are allowed

Tuple Relational Calculus

specify a number of tuple variables ranging over a particular relation, free tuple variables are bound successively

Example: { t.FName, t.LName | emp(t) ∧ t.Sal > 50000 }

Domain Relational Calculus

specify a number of variables that range over single values from domains of attributes, the position of attributes is relevant, names are not used, often anonymous variable _ is used

Example: { FN, LN | emp(FN, LN, Sal) ∧ Sal > 50000 }

DB Systems, Lecture 03

SQL; based on multisets (or bags) rather than sets. In a multiset an element may occur multiple times. Set { … }, Bag {{ … }}. Tables, columns, rows. Case sensitivity (~ quotes).

CHAR(n), VARCHAR(n), INTEGER, REAL, DOUBLE PRECISION, FLOAT(n)

CREATE TABLE
integrity constraints: not null, primary key (X), foreign key (X) references table(Y)

DROP TABLE
ALTER TABLE (add/remove columns)

LIKE with % (0-n chars) or _ (1 char)

SELECT FROM WHERE GROUP HAVING, UNION of those

AS to rename tables and columns, fully qualified column names are supported

FROM lists tables involved in query
FROM t1 JOIN t2 ON cond, LEFT/RIGHT OUTER JOIN, CROSS JOIN, INNER JOIN

WHERE conditions that must be satisfied by result tuples (filter)
AND, OR, NOT, LIKE

GROUP groups multiple tuples together (like small, multiple tables)
GROUP BY name

HAVING is applied to each group, only those fulfilling the condition are returned, always works on whole groups

SELECT lists the columns that shall be in the result of a query
SELECT DISTINCT to eliminate duplicates, * for “all columns”
aggregate functions avg, min, max, sum, count can be used

Derived Tables: use a query expression inside FROM clause

UNION is union, between two compatible queries
INTERSECT is intersection
EXCEPT is set difference

Subquery: a query expression nested within another one

SOME, ANY, ALL, IN
but preferred is
EXISTS, NOT EXISTS

UNIQUE to tests for duplicate tuples

Null values: unknown or not existing value, use IS NULL to check, if evaluates to unknown is treated as false and removed from WHERE clause, all aggregate operations ignore tuples with null values (except count(*))

ORDER BY X [DESC/ASC] to order results by a specific column

INSERT INTO table VALUES (a, b, c)
INSERT INTO table(A, B, C) VALUES (a, b, c)

DELETE FROM table WHERE cond

UPDATE table SET x WHERE cond

CASE statement:

```
CASE
        WHEN cond
        THEN x
        ELSE y
END
```

DB Systems, Lecture 04

Views: mechanism to hide data from user or give him direct access to computation results
CREATE VIEWS v AS <query expression>
view definition is stored in the metadata, updatable if reverse mapping defined

WITH clause: temporary views valid only for the associated query
WITH name(Return-Value) AS (query), multiple possible with , concatenation

Integrity constraints: guard against accidental damage by ensuring that changes do not result in loss of data consistency

On single relations:

- NOT NULL
- PRIMARY KEY (not null and unique)
- UNIQUE
- CHECK(p), where p is predicate

On multiple relations:
- FOREIGN KEY
- CHECK(p), where p is predicate
- ASSERTION

Domain constraints: most elementary integrity constraints, test values
CREATE DOMAIN x type(def)

to covert between types, use CAST x AS typeY

Referential integrity: by using FOREIGN KEY

Assertion: predicate expressing a condition that must be satisfied
CREATE ASSERTION name CHECK predicate
is tested on creation and on every update that might invalidate it (overhead!!!)

Authorization: how users can access, process, or alter data (Privileges, Roles)
GRANT privilege ON x TO user (user-id, public, role)
REVOKE to take away privileges

Accessing Databases: connect, send SQL commands, fetch result tuples back
- Embedded SQL: directly in program code, EXEC SQL
- ODBC (Open DataBase Connectivity): drivers for each database
- JDBC (Java Database Connectivity): for Java

User-defined Functions (UDF), PL/pgSQL for PostgreSQL
Value functions:
CREATE FUNCTION somefunc()
RETURNS <type> AS $$
[DECLARE <declarations>]
BEGIN
    <statements>
END;
$$ language plpgsql;

Example statements:
quantity := 4;
if quantity < 5 then
    quantity := 5;
end if;
while quantity < 10 loop
    quantity := quantity + 1;
end loop;

Can return value (integer, …) or TABLE (with definition)

Triggers: piece of code that is automatically executed in response to certain events
CREATE TRIGGER name AFTER/BEFORE INSERT/UPDATE/DELETE ON x …
FOR EACH ROW
WHEN (cond) <statements>

Recursive views: supported in SQL:1999

DB Systems, Lecture 05

Relational Database Design: find a good collection of relational schemas, with good attribute grouping, that ensure: simplicity, no redundancy, no update anomalies, as little null values as possible, as good coverage of original data as possible, no spurious (ie. wrong) tuples on joins

Functional Dependencies: specify formal measures of the goodness of relational designs, used to define normal forms, are constraints that represent meaning and interrelationships of data attributes, are unique

Armstrong's inference rules:
- Reflexivity: $Y \subseteq X \models X \rightarrow Y$
- Augmentation: $X \rightarrow Y \models XZ \rightarrow YZ$
- Transitivity: $X \rightarrow Y , Y \rightarrow Z \models X \rightarrow Z$
- Decomposition: $X \rightarrow YZ \models X \rightarrow Y , X \rightarrow Z$
- Union: $X \rightarrow Y , X \rightarrow Z \models X \rightarrow YZ$
- Pseudotransitivity: $X \rightarrow Y , WY \rightarrow Z \models WX \rightarrow Z$

Set of FD minimal if:
- no pair of FD has the same left-hand side
- it is not possible to remove any dependency from F and still have a set that is equivalent to F
- it is not possible to replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y subset of X and still have a set that is equivalent to F

Normal Forms

Normalization: the process of decomposing bad relations by breaking up their attributes into smaller relations that fulfill the normal forms

1NF: attribute values must be atomic
2NF, 3NF, BCNF : based on candidate keys and FD of a relation schema
4NF : based on candidate keys, multi-valued dependencies (MVDs)

First Normal Form (1NF)

Disallows:
- composite attributes
- multivalued attributes
- nested relations (attributes whose values for an individual tuple are relations)

Second Normal Form (2NF)

Each attribute not contained in a candidate key is not partially functional dependent on a candidate key. Partially functional dependent means it is functionally dependent on a proper subset of the candidate key.

Third Normal Form (3NF)

For all $X \rightarrow A \in F+$ at least one of the following holds:
- $X \rightarrow A$ is trivial
- X is a superkey for R
- A is contained in a candidate key of R

Forbids transitive dependencies, implies 2NF.

Boyce-Codd Normal Form (BCNF)

For all $X \rightarrow A \in F+$ at least one of the following holds:
- $X \rightarrow A$ is trivial
- X is a superkey for R

BCNF implies 3NF, the converse does not hold, since 3NF allows redundancies that BCNF does not allow. It is however more difficult to check BCNF as you have to consider multiple relations.

Relational database design by decomposition:
- Universal Relation Schema: includes all attributes
- Decomposition: decompose the URS into a set of relation schemas that will become the relational database schema by using functional dependencies
- Following conditions must be met:
  - Each attribute in URS must appear in at least one relation schema (no loss of attributes/data!)
  - Each individual relation in 3NF (or higher)
  - Lossless join decomposition: no wrong tuples if joins are performed (required!)
  - Dependency preservation: all functional dependencies can be checked by considering individual relations only (always possible in 3NF, not in BCNF)

Multivalued Dependencies

$X \rightarrow Y$ on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R:

> If two tuples t1 and t2 exist in r such that $t1[X] = t2[X]$, then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:
> - $t3[X] = t4[X] = t1[X] = t2[X]$.
> - $t3[Y] = t1[Y]$ and $t4[Y] = t2[Y]$.
> - $t3[Z] = t2[Z]$ and $t4[Z] = t1[Z]$.

A MVD $X \rightarrow Y$ in R is called a <u>trivial MVD</u> if $Y \subset X$ or $X \cup Y = R$.

Fourth Normal Form (4NF)

For every <u>multivalued</u> dependency $Y \rightarrow Y$ in F+ at least one of the following holds:
- $X \rightarrow Y$ is trivial
- X is a superkey for R

DB Systems, Lecture 06

Entities: specific objects in the mini-world that are represented in the database.
Attributes: properties used to describe an entity.

- Simple attribute
- Composite attribute
- Multi-valued attribute
- Derived attribute

Entity types: groupings of entities with the same basic attributes.
Entity set: collection of all entities of a particular entity type in a database.
Key attribute: attribute of an entity type for which each entity must have an unique value. Can be composite, an entity type may have several, it is underlined.

Relationships: connects two or more distinct entities with a specific meaning.
Relationship types: groupings of relationships of the same kind
Degree of a relationship: number of participating entities

ER diagrams:
entity type = rectangular box
attributes = oval
- Each attribute is connected to its entity type
- Components of a composite attribute are connected to the oval representing it
- Each key attribute is underlined
- Multivalued attributes are displayed in double ovals
- Derived attributes are displayed in dashed ovals

relationship type = diamond-shaped box (connected to entity with straight lines)

Relationships may have structural constraints:
- cardinality constraints: maximum participation
  - One-to-one (1:1)
  - One-to-many (1:N)
  - Many-to-one (N:1)
  - Many-to-many (M:N)
- participation constraints: minimum participation
  - zero (optional, not existence-dependent) (single-line)
  - one or more (required, existence-dependent) (double-line)

Recursive relationships are possible (like employee/supervisor).
In ER diagrams we need to display role names to distinguish.

Weak entity types: entity type that does not have a key attribute, must participate in an identifying relationship type with an owner entity type. This is a combination of both a partial key as well as the identifying entity type (parent/owner).
In ER diagrams we use double border for this and dashed underlining for the partial keys.

Higher degree relationships: can often not be inferred from binary relationships.
If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant.

Subclasses and Superclasses

ER diagrams do not model sub-groupings, which may be useful.
Those are modeled using superclass/subclass relationships, also called IS-A relationships.
Same entity but in a distinct, specific role!
Inherits everything from the superclass (parent).

Examples:

Employee/Secretary	Secretary IS-A Employee
Employee/Technician	Technician IS-A Employee

Modeled in Extended ER Diagrams using an “IS-A” triangle (point down).

Specialization VS Generalization

Disjointness constraint: the subclass of the specialization must be disjoint
Completeness constraint:

- Total: every entity of superclass must be a member of some subclass
- => double line
- Partial: allows an entity to not belong to any subclass
- => single line

ER-to-Relational Mapping:

- 1) Mapping of Regular Entity Types (decide PRIMARY KEY)
- 2) Mapping of Weak Entity Types (connect FOREIGN KEYs)
- 3) Mapping of Binary 1:1 Relation Types
- 4) Mapping of Binary 1:N Relation Types
- 5) Mapping of Binary M:N Relation Types
- 6) Mapping of Multivalued attributes
- 7) Mapping of N-ary Relationship Types
- 8) Mapping Specialization or Generalization

DB Systems, Lecture 07

Physical storage media: speed / cost / reliability / volatile VS persistent

- Cache
- Main Memory
- Flash Memory (SSDs)
- Magnetic Disk (Hds)
- Optical Disk
- Tape storage

Blocks / Buffers

Buffer replacement policies in DBMS can use various information :
- Queries have well-defined access patterns (sequential scans)
- Information in a query to predict future references
- Statistical information

File organization: sequence of records, either fixed length or variable length
Data dictionary (system catalog): stores metadata

Index Structures for Files

Speed up data access/lookup thanks to fast search; in index file.

Primary index: search key order corresponds to order of records, the search key is a primary key, thus the keys are unique.

Clustering index: cluster same keys together (may occur multiple times), point to first.

Secondary index: used to quickly find all records whose values in a certain field, which is not the primary key, satisfy some condition. Must be dense.

Sparse index: contains records only for some search key values.

Dense index: for each record an index record is present!

B+-Tree

Multi-level index, automatically maintains appropriate number of levels, reorganizes itself automatically, BUT overhead on insert/delete and space usage.
It's a rooted tree with the following properties:

- Balanced tree: all paths are same length
- A Node contains up to m-1 search keys values (sorted) and m pointers
- Nodes are half to full
- Internal nodes have between m/2 and m children
- Leaf nodes have between (m-1)/2 and m-1 children
- Root node has at least 2 children, unless it is leaf, then between 0 and m-1

Data pointers are only stored in leaf nodes!
Leaf nodes are linked together, the last pointer points to the next leaf node.
Path length at most log m/2 (K)
Insertion: if nodes are full, split upwards until everything is fine!
Deletion: if nodes are sparse, either coalesce siblings or, if too many entries for that, redistribute pointers between nodes.

Static Hashing

Hashing can provide a way to avoid index structures and access data directly, less I/O needed. We hash the search key and obtain a bucker, where multiple records are stored. The bucket then needs to be searched sequentially.

Ideal hash function: uniform and random distribution

Bucket overflow: either because insufficient buckets around or skew in distribution, the possibility cannot be eliminated completely! Overflow buckets (closed hashing, chaining).

Dynamic Hashing: allows the hash function to be changed, much more flexible
Extendable Hashing: hash function generates values over a large range, only a prefix of a certain length is used, and that length changes as the DB grows/shrinks

SQL: not directly supported, CREATE INDEX name ON relation (attributes)

DB Systems, Lecture 08

One of the most important tasks of a DBMS is figuring out an efficient **evaluation plan**. Especially for selections and joins.

Query-processing:
- Parsing and translation (from SQL to RA)
- Optimization (refine RA)
- Evaluation (execute RA)

Query cost: total elapsed time

Sorting is important, quick-sort for in-memory, merge-sort for external storage.

Selection Evaluation Strategies:
- linear search
- binary search
- primary index + equality on candidate key
- primary index + equality on non-candidate key
- secondary index + equality on search-key
- primary index on A + comparison condition
- secondary index on A + comparison condition

Join Evaluation:
- nested loop join: baseline, slooow
- block nested loop join
- indexed nested loop join
- merge join: robust, fast
- hash join: fastest, only for equality

Query optimization: alternative ways of evaluating a query because of equivalent expressions or different algorithms depending on the operation, chosen to minimize cost

A reasonably efficient execution tree is chosen using statistical / heuristic information.

Heuristic optimization uses rules to try to reduce the size of (intermediate) relations as early as possible:
- perform selection early
- perform projection early
- perform most restrictive selections and joins before others

Cost-based optimization uses statistical information to get a reasonably cheap plan.

DB Systems, Lecture 09

Transaction States:

- Active: during execution
- Partially committed: after final statement has been executed
- Committed: successfully completed and changes are permanent
- Failed: execution can no longer proceed normally
- Aborted: transaction has been rolled back, DB restored to previous state
  - restart the transaction
  - kill the transaction

ACID Properties:

- Atomicity: changes to state shall be atomic
- Consistency: no constraints are violated
- Isolation: even if multiple transactions concurrently, appear as though sequential
- Durability: after successful commit, change are persistent

Atomicity/Durability: implemented by recovery manager
Isolation: implemented by concurrency control system (better resource usage)
Consistency: task of the application programs!

Schedule: sequence of instructions from a set of concurrent transactions that indicate the chronological order in which these instructions are executed. Must include all instructions and preserve order within each individual transaction.

Serial schedule: transactions execute sequentially.

Serializable schedule: A schedule is serializable if it is equivalent to a serial schedule.

Conflict equivalence: if a schedule can be transformed into another one using only non-conflicting instruction swaps.

Conflict serializable schedule: a schedule that is conflict equivalent to a serial schedule.

Instructions A of transaction T and B of transaction U conflict if there exists a data item Q accessed by both and at least one of those instructions is a write. If both are reads, there is no conflict, as the order doesn't matter.

Instructions can be swapped if they are non-conflicting, meaning if:

- both are reads
- they refer to different data items
- one of them is not a DB operation (read/write)

Precedence graph (conflict graph)

A directed graph with a node Ti for each transaction and with an edge Ti → Tj if one of the following conditions hold:
- Ti executes write(Q) before Tj executes read(Q)
- Ti executes read(Q) before Tj executes write(Q)
- Ti executes write(Q) before Tj executes write(Q)

If precedence graph is acyclic, a schedule is conflict serializable.

Concurrency control protocols

Testing for serializability after execution is too late, and before, often, too slow.
So develop a protocol (set of rules) that avoids any non-serializable schedules.
Examples: lock-based protocols, multiversion concurrency control

Recoverable schedule: for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti must appear before the commit operation of Tj.

Cascading rollback: a single transaction failure leads to a series of rollbacks.

Cascadeless schedules: For each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti appears before the read operation of Tj. This avoids cascading rollbacks. It implies recoverability and is thus desirable.

Desirable properties of a schedule:
- serializable
- recoverable
- preferably cascadeless

Concurrency Control

- Lock-Based Protocols : ensure serializability by taking turns, Read/Write locks
- Pitfalls of Lock-Based Protocols : non-serializable schedules, deadlocks
- Two-Phase Locking (2PL) : a protocol that ensures conflict-serializable schedules, by working in two phases: 1) first obtain all locks 2) then release all locks. It does not ensure deadlock-freedom and cascading rollbacks are still possible. Variants:
  - Strict two-phase locking (S2PL): hold all X-locks until commit, avoid cascading rollback
  - Rigorous two-phase locking (SS2PL): hold all locks until commit, serialized order
- Transactions in SQL : three undesired phenomena: (see isolation levels)
  - dirty read: sees changes from other uncommitted transactions
  - nonrepeatable read: if retrieve same row twice, different answer
  - phantom read: if retrieve a range of rows twice, different answer
- Transaction in DBMSs: serializable not always default, COMMIT / ROLLBACK