CS 4347 — Database Systems, Exam 1 Review

https://iniyanjoseph.com | iniyanijoseph@gmail.com

Databases are needed for storing data

Database System

DBMS A software package to help create and manage databases
Database A collection of related data
Data Known facts which can be recorded and have implicit meaning

Database System Environment

Users/Programmers ⇒ Application Programs / Queries ⇒ Software to Process Queries ⇒ Software to access stored data ⇒ Stored DB Definition (Meta-Data) + Stored Database Data
A DBMS deals with definition, construction, manipulation, processing of databases
Typical DBMS functionality:

- Define a database (data)
- Construct/load a databases initial data or a secondary storage medium
- Manipulating the database (i.e. query, delete, add, generate reports)
- Retrieval of data - Querying, generating reports
- Modification of data - inserting, deleting, updating
- Access of data through applications
- Processing user requests and sharing data while keeping it consistent

Common activities against a database

- Applications will try to generate query results (by tabulating data together)
- Applications will try to generate transactions (changing current data)
- Preveting unauthorized users from accessing restricted data
- Keeping up with changing user requests

Additional functionailities a DBMS may provide

- Security measures
- Presentation & visualization of data
- Database maintainance

Characteristics of the Database Approach

Meta-data Description of data types, constraints, etc
Program-Data independence Insulation between programs and data (Leaving data accessible without being able to change structures)
Data Abstraction Allows users to interact with data without having to understand underlying details
Virtual Tables Each is a users' view of the data of interest for the user (multiple views supported)
Concurrency Control Ensures each transaction is either fully executed or fully aborted
Recovery Subsystem Ensures completed transactions are permanently recorded in the database (failsafe)

Database users:

- Actors on the scene
  - Those who use/control content
  - Those who design/develop/maintain DB applications
  - End users - Using data

  - **Casual** Occasionally access DB when needed
  - **Naive/Parametric** Regular use of "canned" transactions
  - **Sophisticated use** use software packages, etc w/ Database
  - **Stand-alone** Maintain personal Database
- Actors behind the scene
  - Those who design and implement DBMS modules and interfaces

Advantages of the Database Approach

- Ability to control redundancy (normalization has more tables + less redundancy. denormalization has less tables + more redundancy)
- Restricting unauthorized access to data
- Providing structures that allow for efficient processing
- Backup & recovers services
- Multiple interfaces for multiple user types
- Integrity of data (constraints) is not compromised

Implications of using the DB approach:

- Potential for enforcing standards
- Reducide application development time
- Flexibility to change data structures
- Availability of current information

When NOT to use a DBMS

- Insufficient initial investment or hardware/storage
- Simple, well defined applications where multiple-user access is not required
- Data Model: Collection of concepts to describe the structure, operations, and constraints of a DB
  - Conceptual Schema (how users perceive the data)
  - Physical Schema (how computers store data)
  - Implementation Schema (Falls between the two, need by DBMS)
  - Self-describing (Combines descriptions with values)
- **Data schema** description of data structures, operations, and constraints (infrequently changes)
  - Described by a schema diagram
  - Has components (schema constructs)
  - There are table headers, each with columns
- **Data state/Instance** Actual data being stored at a particular moment in time (frequently changes)

Data Independence:

- **Physical** Capacity to change internal schema without affecting the conceptual schema
- **Logical** Same, but change external schema without affecting the logical schema

DBMS Languages

- Data definition languages need by DBAs (used for all levels)
  - Alter schema objects
  - Grant/revoke roles & privileges
  - Collect statistics
  - Establish schema auditing
- Data manipulation language used by application developers
  - (Statements do not implicity commit the current action)

- ▸ High Level Languages (SQL) Set-oriented, specify what to retrieve (but not how)
- ▸ Low-level/procedural languages
  - – Require constructs such as looping to retrieve many records at once, pointers are used as well

DBMS interfaces:

- Stand-alone query language interfaces (e.q. SQL*Plus)
- Programmer-friendly interfaces
- User-friendly interfaces (menu based, graphics based, etc)

DBMS architecture how DBMs are organized (centralized, two-tier, three-tier)

Two-tier structure

- Server handles query + transaction functionality related to SQL
- User interface program & apps

Three-tier structure

- Add a third layer between the two: i.e. the app and the web server
- Runs apps and store business rules
- N-tier divides this into more layers
- MVC

N-tier architecture

- Divides this into more layers

DBMS classification

- Data Model
  - ▸ Relational
  - ▸ Heirarchical & network
  - ▸ Objects
  - ▸ Native XML
- Number of users
- Number of Sites
  - ▸ Centralized
  - ▸ Distributed

When designing a database, both the database itself and applications which use it have to be made. First let's design the database.

Database

- Requirements collection and analysis
- Creating a conceptual schema
- Logical design (create a DB schema)
- Physical design (storage, file structure)

DB Design:

Mini-World

Requirements collection Data requirements

Conceptual Design Conceptual Schema

Logical Design Logical Schema

Physical Design Internal Schema

Transaction Implementation App program

ER Diagram

Entities (objects), relationships, and attributes (data)

- Attributes can be simple (one value), composite (many components), multi-valued (many values), complex(both)
- Derived attributes are derived from other attributes
- Null attributes don't have value (missing/no need)

Entity types

- Same basic attributes ⇒ becomes an entity types
- Entity type has a collection, that is entity set
- Box = Entity
- ellipse = Attribute
- doublecircle = Multivalued attribute
- <u>ellipse</u> = key attribute
- Note that each simple attribute has an associated value set (e.g. DoB has to be of the form XX-XX-XXXX). This gives a restriction on what kinds of data are allowed

Relationship types and relationship sets

- A relationship relates two or more distinct entities with a specific meaning, same type → type

Degree is the # of entities connected (mostly 2)

- doublebox = Weak entity
- diamond = relationship
  - More than one relationship can exist between the same entity types
- doublediamond = identifying relationship
- Dotted ellipse = derived attribute
- Participation constraint - Partial or total
  - "= diamond" = Partial participation
  - "= diamond" = Partial participation
- "$-_1$" = Cardinality of 1
- "$-_N$" = Cardinality of N
- "$-_{\min,\max}$" = min/max constraints
  - Cardinality ratio - 1:1, 1:N, N:1, or M:N
- Recursive relationship type - same entity, different roles
- Relationships can have attributes as well (Things which are generated by the interaction)
- Weak entity types have an identifying relationship with an owner. If the owner is the deleted, so are these
- Nouns are entities, verbs are relationships
- Ternary relationships are equivalent to 3 binary identifying binary relationships to a weak entity

Enhanced ER diagram

The enhanced entity-relationship model is used to include classes and generalizations

EER model additions

- Superclasses/subclasses
- Specialization/generalization + constraints
- Categories (union types)
- Attribute and relationship inheritance

Superclasses/subclasses: Specialization - defining a set of subclasses for a superclass
Double line = total participation
d for distinct o for overlapping (Can the object participate in multiple?)

Specialization Defining a set of subclasses for a superclass
- Generally done because attributes/relationships are subclass specific
  - Subclass attributes are called local attributes

Generalization Defining a superclass for several subclasses
- Generally done becuase A/R are common features

Types of specialization:
Predicate defined Based on a condition/predicate
Attribute defined Based on an attribute
User defined Membership is determined on a case-by-case basis

Notes about certain ways of inheritance:
- In a lattice/heirarchy, a subclass inherits from all superclasses above it
- A subclass w/ more than one supercalss is called a shared subclass

Union Types:
- A subclass which has more than one superclass (these are different entity types) is a category aka

A shared subclass is a subset of the intersection of its superclass (some properties of each type), while a union type uses a union instead (all properties from both types).

UML Diagrams

- Blank triangle is disjoint, filled is overlapping
- Root is base class, leaf is uflly specialized (subclass)
- Diamonds indicate posession (filled means required possesion)

Informal definitions

A relation looks like a table of values
- Set of rows (tuples)
- With a number of columns (attributes)

Formal definitions

- Relational schema $R(A_1, A_2, ..., A_n)$
  - Relation + Attributes
- Relation (state)
- r(R) $\subseteq$ (dom(A_1) $\times$ dom(A_2), $\times$ ... $\times$ dom(A_n)) is a selection of rows
- Each dom(A_i) has cardinality |dom(A_i)| - set size
- Each value in a tuple is atomic (can be NULL)
  - Multivalued attributes get seperate relations
  - Composite attributes are represented by simple component attributes λ

Database constraint types

- Inherent/implicit - based on the relational model
- Schema-based/explicit - Specified by the DDL
- App-based/semantic - enforced by applications

Schema Based Constraints

- Domain constraints
  - Specify some data type (int, string, char, etc)
- Key constraints
  - No two 5-tuples can have the same attributes
  - **key** a set of attributes that uniquely identifies a row in a table
    - **Super key** $\forall t_1, t_2 \in r(R)$, $t_1[SK] = t_2[SK] \Rightarrow t_1 = t_2$
    - In other words, a single uniquely identifying element

- **Entity integrity** No primary key can be NULL
- **Referential integrity** Violation of consistency between relations

Foreing keys are used to reference rows in other relations. ($t_1 \in R_1$ references $t_2 \in R_2$ if $t_1[FK] = t_2[PK]$). This is drawn as an arrow

A foreign key must be

- Either a reference to another primary key, or ...
- NULL (can not be part of a primary key)
- Insertion
  - Reject with a reason (domain, key, reference, or entity violations)
  - Offer an opportunity to correct the violation
- Deletion
  - Deleting a PK can cascade if it is being referenced (reject, let it cascade, set it to null)
- Update
  - Some steps as for insertion (only domain/NULL)

Mapping from EER to relational schema

1. For each entity E, create a relation R with all simple attributes, then choose one of the key attributes as the primary key
   - Even if it's composite , it is not a problem
2. For each weak entity W, create a relation R with all simple attributes
   - PK of owner, E, should become FK for R
   - PK of R is E's PK + W's partial key (if any)
   - Determine PKs in order from closest to farthest from a (strong) entity
3. For each 1:1 relationship, create a mapping
   - **Foreign key approach** choose one relation (from total participation is better) and include the other's PK and this one's FK
   - **Merged relationship approach** if both participations are total, merge into a single relation (works b/c it's 1:1)
   - **Cross-reference approach** set up a third relation that relates PKs of both (song, S&T); this is required for M:N binary relationships
4. Same as 3, but 1:N
   - **Foreign key approach** the N-side entity has a FK which is the 1-side entity's PK
   - Cross-reference approach - Same as before
5. Same as 4, but M:N
   - Cross reference appoarch - Same as before
6. For each multivalued attribute, create a new relation with PK as a combination of PK of entity and PK of attribute (if possible, include the single components)
7. For each N-ary relationship, include all the PKs of all N entities of FKs in the relation
   - Multiple relations, subclass & superclass approach - create a relation with PK k; Then, for each subclass S_i, create a relation L_i with PK k. (Works for any specialization)
   - Multiple relations, subclass only approach - like above, but omit the superclas relation (works if specialization has subclasses whose union is all of the superclasses of λ)
   - Single relation, one type attribute approach - have one relation with attributes of all S_i, PK k, and a type attribute t discriminating attribute
   - Single relation, muliple type attribute approach - similar to above but have $t_1, ..., t_n$ as a boolean attribute indicating belonging to $S_i, ..., S_n$ (handles overlapping subclasses)
8. For each union type, create a new key attribute (a surrogate key, similar to a foreign key) and insert it in λ all of the classes in the union

Relational Algebra

- **Relational Algebra** Basic set of operations on relational model
- **Relational Calculus** Higher level language of relational queries
- **Unary relational operators**

- **SELECT** $\sigma_{\text{condition}}(R)$, selects tuples from R
- **PROJECT** $\pi_{\text{condition}}(R)$, selects just the information from the given attributes from all tuples + removes duplicates if necessary
- **RENAME** $p_{s(B_1,\ldots,B_n)} = \pi_{B_1,\ldots,B_n}(\sigma_s(R))$

- Set theory relational operators:
- **UNION** $R \cup S$ without duplicates
- **MINUS** $R - S$
- **INTERSECTION** $R \cap S = (R \cap S - (R - S)) - (S - R)$
- **PRODUCT** $R \times S$
- **JOIN** $\bowtie$ Combine the two tables using pairs of rows (a, b) if the key specified is equal
  - **EQUIJOIN** THETA JOIN $(R.A \stackrel{?}{=} S.B)$ where $=$ is the only symbol allowed in place of $\stackrel{?}{=}$
  - **NATURAL/INNER JOIN *noted by** Removes the second attribute in an EQUIJOIN condition by using only one join attribute (renaming allowed)
  - **OUTER JOIN**
    - **Left, Right, Outer** $(\bowtie, \ltimes, \rtimes)$
- Aggregate functions
- Writing Queries

SQL

- table, row, column $\Leftrightarrow$ relation, tuple, attribute
- Main command for data definition in `CREATE`
- Schemas
  - Tables, constrains, views, domains, constructs
  - Schema name, authorization, descriptions
- Catalog
  - Named collection of schemas (in SQL)
  - Used for uniform referential integrity
  - Can go higher (cluster of catalogs)

`CREATE`

- `SCHEMA` Schema_name, `AUTHORIZATION` 'Identifier';
- `TABLE` (Schema).Table.Name (...)
  - Data types : `INT, SMALLINT, FLOAT, DOUBLE PRECISION, CHAR(N), VARCHAR(N), BIT(N), DATE`
  - Attribute Data_type < if needed, NOT NULL>
  - PRIMARY Key (PK)
  - UNIQUE
  - FOREIGN KEY (FK) REFERENCES Table (Attr)
- `VIEW`

Constraints In SQL:

- By default, no PK duplicates, no NULL Pks, and FK is either NULL or has a value which is a PK
- NOT NULL
- `DEFAULT` < value >
- For FKs...
  - `ON DELETE <SET NULL, SET DEFAULT, CASCADE>`
  - `ON UPDATE <SET NULL, SET DEFAULT, CASCADE>`
- `CHECK` < condition >, (after `CREATE TABLE`'s other stuff

Retrieval query:

```
SELECT # attr and func list
FROM # table list
GROUP BY # condition
HAVING # grouping attr(s)
ORDER BY # group condition
```

Ambiguous names and aliases:
- If attribute names are shared across multiple relations, the relation name must be stated (EMPLOYEE.Dno)
- Since repeatedly using these names can get long, the `AS` keyword can be used to specify an alias

Table as sets:
- `SELECT ALL` keeps duplicates (the default) for query results while `SELECT DISTINCT` removes them
- `UNION (ALL), INTERSECT (ALL), EXCEPT (ALL)` by default removes duplicates, `ALL` keeps them

Extra specifications:
- `LIKE` for string matching
  - `%` for any # of characters
  - `_` matches a single character
- `BETWEEN #1 AND #2` for values in [#1, #2]
- Standard arithmetic operators are fine
- `ORDER BY A1 <DESC, ASC>,...` for ordering by d/a

Modifications:
- `INSERT INTO table (attr, ...) SELECT ...` loads from a query result
- `INSERT INTO table VALUES` loads from a query result
- `CREATE TABLE NewTable LIKE OldTable (SELECT ...) WITH DATA` for bulk loading
- `DELETE FROM Table WHERE <condition>`
- `UPDATE Table SET <setting statements> WHERE <condition>`
- For all modifications, the appropriate conditions will be verified

`AND, OR, NOT` for logic

SQL can check for `NULL` by using `IS <NOT> NULL`

Queries can be nested with `IN`

```
SELECT DISTINCT essn
FROM WORK_ON
WHERE (Pro, Hours) IN (SELECT Pno, Hours FROM WORK_ON WHERE Essn='11222333')
```

Nested queries operations:
- `NOT IN (...), ~ IN (...), OR ~IN(...), ~ IN (...) AND ~IN (...)`
- `~= ANY(...)` (can use other comparison operators)
- `~>ALL(...)` (can use other comparison operators)
- `EXISTS (...), NOT EXISTS (...)` (checks for ≥ 1 or 0 tuples)
- `UNIQUE (...)` (checks for no duplicate tuples)

Sets can be renamed explicitly: `WHERE Pno IN (1, 2, 3);`

Joins:
- Specified in the `FROM` clause
- Five types: `EQUIJOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN`
  - `FROM T_A <EQUIJOIN, NATURAL JOIN, ...> T_B` in general
  - Joins can be nested to become multiway

Aggregate functions in SQL:
- Built in are `COUNT, SUM, AVG, MIN, MAX`
- `SELECT FUNC(attr) FROM Table;`
  - `NULL` values are ignored
- `GROUP BY Attr` seperates the tuples before applying functions
- `HAVING <condition>` allows for rejecting some groups

`CASE` does grouping for updates

```
UPDATE EMPLOYEE
SET SALARY =
CASE WHEN Dno = 5 THEN Salary*0.5
  WHEN Dno = 1 THEN Salary *1.5
```

SQL also can execute recursive queries - queries that continue to run until a base case is reached & they terminate

Outside of EER/relation model, we have several λ main semantic constraints: assertions and triggers.

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
  FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D WHERE E.Salary > M.Salary AND E.Dno =
D.Dnumber AND D.Mgr_ssn = M.ssn));
```

Any tuple inserted pass this assertion

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF Salary, Supervisor, ssn ON EMPLOYEE
FOR EACH ROW
WHEN (New.Salary > SELECT Salary FROM EMPLOYEE
WHERE Ssn = New.Supervisor.SSn)

INFORM_SUPERVISOR (New.Supervisor, Ssn, New.SSn)
```

This trigger monitors the database and detects when the employee has a higher salary than their supervisor.

Views on SQL:

- `CREATE VIEW View_name AS SELECT (...) FROM (...) WHERE (...) ...`
- This view is a virtual table and is always up-to-date (responsibility of DBMS)
- `DROP VIEW` disposes of a view

Admin & schema evaluation:

- If a DBA wants to change the schema while the DB is still in use, it can be done w/o recompilation
- `DROP SCHEMA Schema_name <CASCADE, RESTRICT>`
- `ALTER TABLE Schema.Table ADD_COLUMN Col_name ...`
- `"                    " DROP COLUMN <CASCADE, RESTRICT>`
- `ALTER TABLE <CASCADE, RESTRICT> ALTER COLUMN <CASCADE, RESTRICT>`
  - `DROP DEFAULT;`
  - `SET DEFAULT ~;`