Lecture 2. Relational Algebra

* Background context behind Relational Model

Remark. After it was discovered, it looked so obvious to everyone.

Q. What life was like before relational model?

A. No clean separation between logical & physical layer. (Tight coupling)
You had to know roughly what kind of queries your app have to execute.

* Relational Model Proposal by Edgar F. Codd.

Database abstraction to avoid the above problem.

i) Store database in simple data structure. → Robustness.

ii) Access data through high-level language.

Remark: Very controversial at the time, as many people believed that man-made code was better than generated code.

ex) C compiler generating assembly code more efficient compared to human coding?

iii) Physical storage left to implementation.

Remark. Can make decisions adaptively through time.
No need to change application co

* Data Models.

Def. A data model is a collection of concepts for describing data in a database.

Def. A schema is a description of a particular collection of data, using a given data model.

* Relational Model

i) Structure : The definition of relations & their contents.

ii) Integrity : Ensuring database's contents satisfy constraints

iii) Manipulation : How to access & modify database contents.

Def. A relation is an unordered set which contains ↗ n-ary relation = Tables with n columns.
the relationship of attributes that represent entities.

Def. A tuple is a sequence of attribute values in the relation.
"An instance of a relationship". ↳ "domain".

Remark. Values are (normally) atomic / scalar.

Remark. The special value NULL is a member of every domain.

Def. Every relation has at least one candidate key
that uniquely identifies a single tuple.

↳ Manual injection of "id" solves this problem.

Def. A relation's primary key is a candidate key
that is deemed more "important" than other candidate keys.

Def. A foreign key specifies that
an attribute from one relation has to map to a tuple in another relation.

* Data Manipulation Languages (DML)

i) Procedural (Navigation) ⇝ Relational Algebra. SQL.

Query specifies the high-level strategy the DBMS should use.

↳ Not algorithms or data structure.
"Steps to take to find the results"

ii) Non-procedural. ⇝ Relational Calculus.

Query only specifies what data is wanted.

* Relational Algebra.

: fundamental operations to retrieve and manipulate tuples in a relation.
Each operator takes one or more relations as its inputs,
and outputs a new relation. ⇝ "Chaining" operators.

i) Select. $\sigma_p(R)$ WHERE p

Choose a subset of tuples from a relation R
that satisfies a selection predicate p.

Remark. Can combine multiple predicates using conjunctions or disjunctions.

ii) Projection $\Pi_{A_1, A_2, \ldots}(R)$. SELECT $A_1, A_2, \ldots$

Generate a new relation from a relation R
with tuples that contains specified attributes $A_1, A_2, \ldots$.

Remark. Can rearrange attributes' ordering. Can manipulate values arithmetically.

iii) Union $R \cup S$ R UNION S

Generate a new relation that contains all tuples in either or both
of the input relation R and S.

Remark. Same arity & domain needed.

iv) Intersection $R \cap S$ R INTERSECT S

Generate a new relation that contains all tuples in both of the input relation R and S.

Remark. Same arity & domain needed.

v). Difference $R - S$. R EXCEPT S

Generate a new relation that contains only tuples that appear in relation R and not S.

Remark. Same arity & domain needed.

vi). (Cartesian) Product $R \times S$ R CROSS JOIN S.

Generate a relation that contain all possible combinations of tuples from the input relation R and S.

vii) Join $R \bowtie S$. R NATURAC JOIN S

Generate a relation that contains all the tuples that a combination of two tuples, one from each input relation R and S, with a common value(s) for one or more attributes.

* Join Types.

i) CROSS JOIN : Same as cartesian product

ii) INNER JOIN : Must have a corresponding match.

- Natural Join $R \bowtie S$
- Theta Join $R \bowtie_\theta S$: Match tuples with some arbitrary join predicate θ.
- Equi Join $R \bowtie_\theta S$: Theta join when θ is an equality predicate.
- Semi Join $R \ltimes S$: Same with natural join, but output relation only contain tuples of R.
- Anti Join $R \triangleright S$: Contains tuple of R that do not match with any tuple of S.

→ Important in distributed database systems.

```
SELECT R.* FROM R
    WHERE EXISTS (
        SELECT S.* FROM S ...
```

→ Allow DBMS to only send the data from R not S.
= Amount of data moving is reduced.

iii) OUTER JOIN : Do not need to have a corresponding match.

- Left Outer join $R \bowtie S$: Generate all the combinations of tuple in R and S that are equal on their shared attributes, in addition to tuples in R that have no matching tuple in S.
- Right outer join $R \bowtie S$: $S \bowtie R$.
- Full outer join $R \bowtie S$: $(R \bowtie S) \cup (R \bowtie S)$.

* Additional Operators

i) Rename ρ ex) AS

ii) Assignment $R \leftarrow S$ ex) Store output relation

iii) Duplicate elimination δ ex) DISTINCT.

iv) Aggregation γ ex) avg. cnt. min. max.

v) Sorting τ Remark. Cannot be done in a strict manner. "unordered sets"

vi) Division $R \div S$ Remark. Doesn't actually appear in the real world.

Remark. Relational algebra does define high level steps of how to compute.

ex) $\sigma_{b\_id=1}(R \bowtie S)$ vs. $R \bowtie \sigma_{bid=1}(S)$ → More efficient!

↝ Better approach: Just state high-level query, namely, SQL.
Let the DBMS does the optimization!

* Relational Calculus. (equivalent to relational algebra).

i) Tuple relational calculus: Bound tuples with predicates.

ii) Domain relational calculus: Bound attributes with predicates.

* Non-relational Data Model.

i) Key-Value. ex) Redis

Store records in an associative array that maps a key to a value.
So it's sometimes called a dictionary or a hash table.
Remark. Value is often opaque to DBMS, so it's up to application to interpret.

ii) Graph ex) Neo4j

Represent the database as a collection of nodes & edges.
Each node & edge are annotated with additional properties. (metadata).

Remark. You can do all the things with RDB, but it is not trivial.

iii) Document. ex) MongoDB

A document is a self-contained record that contains
the description of its attributes and their corresponding values.

Remark. "Self-contained" → Schemaless. Attribute is not fixed for every document.

Remark. Values aren't scalar. Values are also documents. "nesting".
⇒ "Denormalization" in relational model. (very arguable.)

iv) Column-family. ex) Google Bigtable.

Hybrid data model that maps a key to a column family.
Each column family contains any number of rows
that each has one or more column names and values.

Remark. Data per row → RDB-like
Column family is self-contained → Document-like.

Remark. All of these is representable in the relational model.
Now some of them support dialects of SQL.
They are all dependent in physical storage.