Database Schema

- A relation schema is a relation name and a set of attributes
  R(a int, b varchar[20]);
- A relation instance for R is a set of records over the attributes in the schema for R.
Some Schema are better than others

- **Schema 1:**
  ```
  OnOrder(supplier_id, part_id, quantity, supplier_address)
  ```
- **Schema 2:**
  ```
  OnOrder2(supplier_id, part_id, quantity);
  Supplier(supplier_id, supplier_address);
  ```

- **Space**
  - Schema 2 saves space
- **Information preservation**
  - Some supplier addresses might get lost with schema 1.
- **Performance trade-off**
  - Frequent access to address of supplier given an ordered part, then schema 1 is good.
  - Many new orders, schema 1 is not good.

Functional Dependencies

- X is a set of attributes of relation R, and A is a single attribute of R.
  - X determines A (the functional dependency X → A holds for R) iff:
    - For any relation instance I of R, whenever there are two records r and r' in I with the same X values, they have the same A value as well.
- **OnOrder1(supplier_id, part_id, quantity, supplier_address)**
  - supplier_id → supplier_address is an interesting functional dependency

Key of a Relation

- Attributes X from R constitute a key of R if X determines every attribute in R and no proper subset of X determines every attribute in R.

- **OnOrder1(supplier_id, part_id, quantity, supplier_address)**
  - supplier_id, part_id is not a key
- **Supplier(supplier_id, supplier_address);**
  - Supplier(id) is a key
Normalization

• A relation is **normalized** if every interesting functional dependency \( X \rightarrow A \) involving attributes in \( R \) has the property that \( X \) is a key of \( R \).

• OnOrder1 is not normalized
• OnOrder2 and Supplier are normalized

Example #1

• Suppose that a bank associates each customer with his or her home branch. Each branch is in a specific legal jurisdiction.
  – Is the relation \( R(\text{customer}, \text{branch}, \text{jurisdiction}) \) normalized?

Example #1

• What are the functional dependencies?
  – customer \( \rightarrow \) branch
  – branch \( \rightarrow \) jurisdiction
  – customer \( \rightarrow \) jurisdiction

  – Customer is the key, but a functional dependency exists where customer is not involved.
  – \( R \) is not normalized.
Example #2

• Suppose that a doctor can work in several hospitals and receives a salary from each one. Is R(doctor, hospital, salary) normalized?

Example #2

• What are the functional dependencies?
  – doctor, hospital → salary
• The key is doctor, hospital
• The relation is normalized.

Example #3

• Same relation R(doctor, hospital, salary) and we add the attribute primary_home_address. Each doctor has a primary home address and several doctors can have the same primary home address. Is R(doctor, hospital, salary, primary_home_address) normalized?
Example #3

- What are the functional dependencies?
  - doctor, hospital → salary
  - doctor → primary_home_address
  - doctor, hospital → primary_home_address

- The key is no longer doctor, hospital because doctor (a subset) determines one attribute.
- A normalized decomposition would be:
  - R1(doctor, hospital, salary)
  - R2(doctor, primary_home_address)

Practical Schema Design

- Identify entities in the application (e.g., doctors, hospitals, suppliers).
- Each entity has attributes (an hospital has an address, a jurisdiction, …).
- There are two constraints on attributes:
  1. An attribute cannot have attribute of its own.
  2. The entity associated with an attribute must functionally determine that attribute.

Practical Schema Design

- Each entity becomes a relation
- To those relations, add relations that reflect relationships between entities:
  - WorksIn (doctor_ID, hospital_ID)

- Identify the functional dependencies among all attributes and check that the schema is normalized:
  - If functional dependency AB → C holds, then ABC should be part of the same relation.
Vertical Partitioning
Scenario #1

• Three attributes: account_ID, balance, address.
• Functional dependencies:
  – account_ID \(\rightarrow\) balance
  – account_ID \(\rightarrow\) address
• Two normalized schema design:
  – (account_ID, balance, address)
  or
  – (account_ID, balance)
  – (account_ID, address)
• Which design is better?

Vertical Partitioning
Scenario #1

• Which design is better depends on the query pattern:
  – The application that sends a monthly statement is the principal user of the address of the owner of an account
  – The balance is updated or examined several times a day

• The second schema might be better because the relation (account_ID, balance) can be made smaller:
  – An index on account_ID might be a level smaller
  – More account_ID, balance
  – A scan performs better because there are fewer pages.

Tuning Normalization

• A single normalized relation XYZ is better than two normalized relations XY and XZ if the single relation design allows queries to access X, Y and Z together without requiring a join.
• The two-relation design is better iff:
  – Users access tend to partition between the two sets Y and Z most of the time
  – Attributes Y or Z have large values
Vertical Antipartitioning

- Brokers base their bond-buying decisions on the price trends of those bonds. The database holds the closing price for the last 3000 trading days, however the 10 most recent trading days are especially important.
  - (bond_id, issue_date, maturity, ...)
  - (bond_id, date, price)
  - (bond_id, issue_date, maturity, today_price, ...10dayago_price)
  - (bond_id, date, price)

Vertical Partitioning and Scan

- R(X,Y,Z)
  - X is an integer
  - YZ are large strings
- Scan Query
- Vertical partitioning exhibits poor performance when all attributes are accessed.
- Vertical partitioning provides a sped up if only two of the attributes are accessed.

Vertical Partitioning and Point Queries

- R(X,Y,Z)
  - X is an integer
  - YZ are large strings
- A mix of point queries access either XYZ or XY.
- Vertical partitioning gives a performance advantage if the proportion of queries accessing only XY is greater than 20%.
- The join is not expensive compared to a simple look up.
Tuning Denormalization

- Denormalizing means violating normalization for the sake of performance:
  - Denormalization speeds up performance when attributes from different normalized relations are often accessed together
  - Denormalization hurts performance for relations that are often updated.

Denormalization

- TPC-H schema
- Query: find all lineitems whose supplier is in europe.
- With a normalized schema this query is a 4-way join.
- If we denormalize lineitem and introduce the name of the region for each lineitem we obtain a 30% throughput improvement.