Data Models

CS552- Chapter Three
Chapter Objectives

- What are the available options and techniques in modeling organizational data before implementation?
- Datamodels
- Relational Data model
- Relational Integrity
- Relational Algebra
Agenda (chapter three)

- Overview/Introduction
- The Hierarchical Model
- The Network Model
- The Relational Model
  - Terminologies, Relational Algebra, Properties of Relation, views and Relational Integrity Rules
Data Models: Overview

- A specific DBMS has its own specific Data Definition Language,
  - however this type of language is too low level to describe the data requirements of an organization in a way that is readily understandable by a variety of users.
- We need a higher-level language.
- Such a higher-level is called data-model.
• **Data Model**
  - A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
  - is a description of the way that data is stored in a database.
  - Data model helps to understand the relationship between entities and to create the most effective structure to hold data.
Cont...

- **Data Model** is a collection of tools or concepts for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Will be implemented on DBM Software
DBMS software Options

**Consumer**
- Microsoft Excel
  - Limit of 65,536 Rows
- Microsoft Access
- FileMaker Pro
- MySQL (Open Source)
- Postgres (Open Source)

**Enterprise RDMS**
- Oracle
- IBM/DB2
- MS SQL-server
- Sybase
- Informix
- Lotus Notes
- MySQL (Open Source)
- Postgres (Open Source)
Cont...
A data model is a collection of tools or concepts for describing data, the meaning of data, data relationships, and data constraints.

Within the history of database systems we have

- the first generation data models
  - Hierarchical Model
  - Network Model
- The second generation data models
  - Relational Model
- The third generation data models
  - Object Oriented Data Models
Cont...

- Data model Evolution
  - File Systems
  - Hierarchical Model (Tree-based)
  - Network Model (Graph-based)
  - Relational Model
  - Object Oriented Model
  - Object/Relational Model
Evolution of Data models Systems

- Flat files
- Hierarchical
- Network
- Relational
- Object-oriented
- Object-relational
- Data warehousing
- Web-enabled

1960 - 2000

- Under active development
- Legacy systems still used
The main **purpose** of Data Model is to represent the data in an understandable way.

Many data models available, falling in either of the following Categories

- Object-based data models
- Record-based data models
- Physical data models
• Object-based data models
  • Mostly used at higher level modeling (external and conceptual)
  • Uses concepts like entity, attribute and relationship
  • Some of the most common type are
    • Entity-relationship (most widely used)
    • Object Oriented (extends ER by adding behavior)
    • Semantic and Functional
Physical data models

- Used to describe data at the internal level
- Describe how data is stored in the computer representing information such as record structure, record ordering, and access paths
- Most common ones are
  - Unifying model (through Agile method)
Cont...

- **Record-based Data Models**
  - Consist of a number of fixed format records.
  - Each record type defines a fixed number of fields,
  - Each field is typically of a fixed length.
  - There are three major types
    - Hierarchical Data Model (old)
    - Network Data Model (old)
    - Relational Data Model
Hierarchical Model

- The simplest data model
- Record type is referred to as node or segment
- The top node is the root node
- Nodes are arranged in a hierarchical structure as sort of upside-down tree
- A parent node can have more than one child node
Hierarchical Data Model

- **ROOT**
  - **FIRST CHILD**
    - Compensation
    - Job Assignments
    - Benefits
    - Performance Ratings
    - Salary History
    - Pension
    - Life Insurance
    - Health
A child node can only have one parent node

The relationship between parent and child is one-to-many

Relation is established by creating physical link between stored records (each is stored with a predefined access path to other records)

To add new record type or relationship, the database must be redefined and then stored in a new form.
Cont...

Diagram:
- Department
  - Employee
    - Time Card
  - Job
    - Activity
Thus

- A hierarchical database consists of a collection of *records* which are connected to one another through *links*.
- A record is a collection of fields, each of which contains only one data value.
- A link is an association between precisely two records.
- The hierarchical model differs from the network model in that the records are organized as collections of trees rather than as arbitrary graphs.
A parent may have an arrow pointing to a child, but a child must have an arrow pointing to its parent.
Cont...

- Database schema is represented as a collection of tree-structure diagrams.
  - The root of this tree is a dummy node
  - The children of that node are actual instances of the appropriate record type
- When transforming E-R diagrams to corresponding tree-structure diagrams, we must ensure that the resulting diagrams are in the form of rooted trees.
Example: E-R diagram with two entity sets, *customer* and *account*, related through a binary, one-to-many relationship *depositor*.

Corresponding tree-structure diagram has

- the record type *customer* with three fields: *customer-name*, *customer-street*, and *customer-city*.
- the record type *account* with two fields: *account-number* and *balance*
- the link *depositor*, with an arrow pointing to *customer*
Cont...

(a) E-R diagram

(b) Tree-structure diagram
If the relationship *depositor* is one to one, then the link *depositor* has two arrows.

Only one-to-many and one-to-one relationships can be directly represented in the hierarchical mode.
• ADVANTAGES of Hierarchical Data Model:
  • Hierarchical Model is simple to construct and operate on
  • Corresponds to a number of natural hierarchically organized domains
    - e.g., assemblies in manufacturing, personnel organization in companies
  • Language is simple; uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT etc.

• DISADVANTAGES of Hierarchical Data Model:
  • Navigational and procedural nature of processing
  • Database is visualized as a linear arrangement of records
  • Little scope for "query optimization"
Network Data Model

- Allows record types to have more than one parent unlike hierarchical model
- A network data models sees records as set members
- Each set has an owner and one or more members
Cont...

- Allow no many to many relationship between entities
- Like hierarchical model network model is a collection of physically linked records.
- Allow member records to have more than one owner
Cont...

- For every E-R diagram, you can have a corresponding data-structure diagram.
Cont...

(a) E-R diagram

(b) Network diagram
Network Representation of Ternary Relationship
• ADVANTAGES of Network Data Model:
  • Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
  • Can handle most situations for modeling using record types and relationship types.
  • Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET etc. Programmers can do optimal navigation through the database.

• DISADVANTAGES of Network Data Model:
  • Navigational and procedural nature of processing
  • Database contains a complex array of pointers that thread through a set of records.
  • Little scope for automated "query optimization"
Relational Data Model

- Developed by **Dr. Edgar Frank Codd in 1970** (famous paper, 'A Relational Model for Large Shared Data Banks')
- Terminologies originates from the branch of mathematics called set theory and relation
- Can define more flexible and complex relationship
- Viewed as a collection of tables called “Relations” equivalent to collection of record types
Relational Data Model

Relation variable (Table name)

Attribute (Column) \{unordered\}

Heading

Body

Relation (Table)

Tupple (Row) \{unordered\}
A Relational model

(a) The `customer` table

<table>
<thead>
<tr>
<th>customer-id</th>
<th>customer-name</th>
<th>customer-street</th>
<th>customer-city</th>
</tr>
</thead>
<tbody>
<tr>
<td>192-83-7465</td>
<td>Johnson</td>
<td>12 Alma St.</td>
<td>Palo Alto</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>4 North St.</td>
<td>Rye</td>
</tr>
<tr>
<td>677-89-9011</td>
<td>Hayes</td>
<td>3 Main St.</td>
<td>Harrison</td>
</tr>
<tr>
<td>182-73-6091</td>
<td>Turner</td>
<td>123 Putnam Ave.</td>
<td>Stamford</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>Jones</td>
<td>100 Main St.</td>
<td>Harrison</td>
</tr>
<tr>
<td>336-66-9999</td>
<td>Lindsay</td>
<td>175 Park Ave.</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>Smith</td>
<td>72 North St.</td>
<td>Rye</td>
</tr>
</tbody>
</table>

(b) The `account` table

<table>
<thead>
<tr>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>A-215</td>
<td>700</td>
</tr>
<tr>
<td>A-102</td>
<td>400</td>
</tr>
<tr>
<td>A-305</td>
<td>350</td>
</tr>
<tr>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>A-217</td>
<td>750</td>
</tr>
<tr>
<td>A-222</td>
<td>700</td>
</tr>
</tbody>
</table>

(c) The `depositor` table

<table>
<thead>
<tr>
<th>customer-id</th>
<th>account-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>192-83-7465</td>
<td>A-101</td>
</tr>
<tr>
<td>192-83-7465</td>
<td>A-201</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>A-215</td>
</tr>
<tr>
<td>677-89-9011</td>
<td>A-102</td>
</tr>
<tr>
<td>182-73-6091</td>
<td>A-305</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>A-217</td>
</tr>
<tr>
<td>336-66-9999</td>
<td>A-222</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>A-201</td>
</tr>
</tbody>
</table>
Terminologies

- **Relation**
  - Two dimensional table
  - Stores information or data in the form of tables → **rows and columns**
    - A row of the table is called tuple → equivalent to **record**
    - A column of a table is called attribute → equivalent to **fields**
  - Data value is the value of the Attribute
Cont...

- Domain
  - A set of allowable values for one or more attributes
  - Example for an attribute we may define domain as “character: size 1, value M or F”

- Tuple
  - Is a raw of a relation
  - Are extension (state) of a relation which changes over time
Cont...

- **Degree**
  - The degree of a relation is the number of attribute it contains
  - Unary, Binary, Ternary, n-ary..
  - Is a property of the intension (schema) of the relation

- **Cardinality**
  - The cardinality of a relation is the number of tuples it contains
  - Is a property of the extension of the relation
• Relational database
  • A collection of normalized relations with distinct relation names
  • Consists of relations that are appropriate structured
    • This appropriateness is what we call normalization (later on this, chapter 5)
  • The rows represent records (collections of information about separate items)
  • The columns represent fields (particular attributes of a record)
**Alternative terminologies**

<table>
<thead>
<tr>
<th>Formal terms</th>
<th>Alternative 1</th>
<th>Alternative 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Table</td>
<td>File</td>
</tr>
<tr>
<td>Tuple</td>
<td>Row</td>
<td>Record</td>
</tr>
<tr>
<td>Attribute</td>
<td>Column</td>
<td>Field</td>
</tr>
</tbody>
</table>
• Records are related by the data stored jointly in the fields of records in two tables or files.
• The related tables contain information that creates the relation
• The tables seem to be independent but are related somehow.
• No physical consideration of the storage is required by the user
• Many tables are merged together to come up with a new virtual view of the relationship
Cont...

- Conducts searches by using data in specified columns of one table to find additional data in another table
- In conducting searches, a relational database matches information from a field in one table with information in a corresponding field of another table to produce a third table that combines requested data from both tables
Object-Relational data model

- Object-relational model:
  - Similar to a relational database model, but objects, classes and inheritance are directly supported in database schemas and in the query language.
More on Relational Model
Properties of a relation

• A relation has a unique name
• Each cell in a relation contains exactly one atomic (single) value
• Each attribute has a distinct name
• The values of an attribute are all from the same domain
• Each tuple is distinct; there are no duplicate tuples
• The order of attributes and tuples has no significance
Cont...

- each row of a table is uniquely identified by a **primary key** composed of one or more columns
- group of columns, that uniquely identifies a row in a table is called a **candidate key**
- **In relation to primary key; entity integrity rule** - states that no component of the primary key may contain a **null** value.
a column or combination of columns that matches the primary key of another table is called a foreign key.
used to cross-reference tables.
the referential integrity rule - states that, for every foreign key value in a table there must be a corresponding primary key value in another table in the database.
all tables are logical entities
Cont...

- A table is either a BASE TABLES (Named Relations) or VIEWS (Unnamed Relations)
- Only Base Tables are physically stores
- VIEWS are derived from BASE TABLES with SQL instructions like:
  
  \[
  \text{[SELECT .. FROM .. WHERE .. ORDER BY]}
  \]
- A relational database is the collection of tables
  - Each entity in one table
  - Attributes are fields (columns) in table
Cont...

- Order of rows and columns is immaterial
- Entries with repeating groups are said to be un-normalized
- Entries are single-valued
- Each column (field or attribute) has a distinct name
- All values in a column represent the same attribute and have the same data format
- The common convention for representing a relation schema is to give the name of the relation followed by the attribute names in parenthesis
  - Branch (branchNo, street, city, postcode)
- The conceptual model or conceptual schema (logical design), is the set of all such schema for the database.
Key constraints (Relational Key)

- If tuples are need to be unique in the database, then we need to make each tuple distinct.
- To do this we need to have relational keys that uniquely identify each relation.
  - **Super Key**: an attribute or set of attributes that uniquely identifies a tuple within a relation.
  - **Candidate Key**: a super key such that no proper subset of that collection is a Super Key within the relation.
    - (super key containing minimum number of attributes necessary for unique identification)
For example, Given an employee table, consisting of the columns employeeID, name, job, and departmentID, we could use the employeeID in combination with any or all other columns of this table to uniquely identify a row in the table.

Examples of superkeys in this table would be:
- \{employeeID, Name\},
- \{employeeID, Name, job\}, and
- \{employeeID, Name, job, departmentID\}.
In a real database we don't need values for all of those columns to identify a row. We only need, per our example, the set \{employeeID\}.

- This is a minimal superkey — that is, a minimal set of columns that can be used to identify a single row.

- So, employeeID is a candidate key.
A candidate key has two properties:
- **Uniqueness**
- **Irreducibility**

If a super key is having only one attribute, it is automatically a Candidate key.

If a candidate key consists of more than one attribute it is called Composite Key.
• **Primary Key:** the candidate key that is selected to identify tuples uniquely within the relation.
  • The entire set of attributes in a relation can be considered as a primary key in a worst case.

• **Foreign Key:** an attribute, or set of attributes, within one relation that matches the candidate key of some relation.
  • A foreign key is a link between different relations to create the view or the unnamed relation
Relational Integrity

- **Domain Integrity**: No value of the attribute should be beyond the allowable limits

- **Entity Integrity**: In a base relation, no attribute of a Primary Key can assume a value of NULL
  - Null – represents a value for an attribute that is currently unknown or is not applicable for this tuple.
    - Are ways to deal with incomplete or exceptional data

- **Domain Constraints**
  - Allowable values for an attribute. See the next Table

- **Entity Integrity**
  - No primary key attribute may be null. All primary key fields **MUST** have data
Domain definitions enforce domain integrity constraints

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Domain Name</th>
<th>Description</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_ID</td>
<td>Customer_IDs</td>
<td>Set of all possible customer IDs</td>
<td>character: size 5</td>
</tr>
<tr>
<td>Customer_Name</td>
<td>Customer_Names</td>
<td>Set of all possible customer names</td>
<td>character: size 25</td>
</tr>
<tr>
<td>Customer_Address</td>
<td>Customer_Addresses</td>
<td>Set of all possible customer addresses</td>
<td>character: size 30</td>
</tr>
<tr>
<td>City</td>
<td>Cities</td>
<td>Set of all possible cities</td>
<td>character: size 20</td>
</tr>
<tr>
<td>State</td>
<td>States</td>
<td>Set of all possible states</td>
<td>character: size 2</td>
</tr>
<tr>
<td>Postal_Code</td>
<td>Postal_Codes</td>
<td>Set of all possible postal zip codes</td>
<td>character: size 10</td>
</tr>
<tr>
<td>Order_ID</td>
<td>Order_IDs</td>
<td>Set of all possible order IDs</td>
<td>character: size 5</td>
</tr>
<tr>
<td>Order_Date</td>
<td>Order_Dates</td>
<td>Set of all possible order dates</td>
<td>date format mm/dd/yy</td>
</tr>
<tr>
<td>Product_ID</td>
<td>Product_IDs</td>
<td>Set of all possible product IDs</td>
<td>character: size 5</td>
</tr>
<tr>
<td>Product_Description</td>
<td>Product_Descriptions</td>
<td>Set of all possible product descriptions</td>
<td>character size 25</td>
</tr>
<tr>
<td>Product_Finish</td>
<td>Product_Finishes</td>
<td>Set of all possible product finishes</td>
<td>character: size 15</td>
</tr>
<tr>
<td>Standard_Price</td>
<td>Unit_Prices</td>
<td>Set of all possible unit prices</td>
<td>monetary: 6 digits</td>
</tr>
<tr>
<td>Product_Line_ID</td>
<td>Product_Line_IDs</td>
<td>Set of all possible product line IDs</td>
<td>integer: 3 digits</td>
</tr>
<tr>
<td>Ordered_Quantity</td>
<td>Quantities</td>
<td>Set of all possible ordered quantities</td>
<td>integer: 3 digits</td>
</tr>
</tbody>
</table>
Referential Integrity: If a Foreign Key exists in a relation, either the Foreign Key value must match a Candidate Key value in its home relation or the Foreign Key value must be NULL.

Enterprise Integrity: Additional rules specified by the users or database administrators of a database are incorporated.

Example restriction on a number of workers in a given branch office (branch as a relation)
Referential integrity constraints are drawn via arrows from dependent to parent table.
Examples

- **Referential Integrity**—rule states that any foreign key value (on the relation of the many side) MUST match a primary key value in the relation of the one side. (Or the foreign key can be null)

- For example: Delete Rules
  - **Restrict**—don’t allow delete of “parent” side if related rows exist in “dependent” side
  - **Cascade**—automatically delete “dependent” side rows that correspond with the “parent” side row to be deleted
  - **Set-to-Null**—set the foreign key in the dependent side to null if deleting from the parent side ➔ not allowed for weak entities
Example

- Keys are special fields that serve two main purposes:
  - **Primary keys** are unique identifiers of the relation in question. Examples include employee numbers, social security numbers, etc. *This is how we can guarantee that all rows are unique*
  - **Foreign keys** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)
- Keys can be **simple** (a single field) or **composite** (more than one field)
Primary Key

Foreign Key

(implies 1:N relationship between customer and order)

Combined, these are a composite primary key (uniquely identifies the order line)...individually they are foreign keys (implement M:N relationship between order and product)
Referential integrity constraints are implemented with foreign key to primary key references.
Relational Views

- Relations are perceived as a Table from the users’ perspective.
- Actually, there are two kinds of relation in relational database.
- The two categories or types of Relations are Named and Unnamed Relations.
- The basic difference is on how the relation is created, used and updated:
  - **Base Relation**
    - A *Named Relation* corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.
  - **View (Unnamed Relation)**
    - A View is the dynamic result of one or more relational operations operating on the base relations to produce another virtual relation that does not actually exist as presented.
Cont...

- So a view is **virtually derived relation** that does not necessarily exist in the database but can be produced upon request by a particular user at the time of request.
- The virtual table or relation can be created from single or different relations by extracting some attributes and records with or without conditions.
Cont...

- **Purpose of a view**
  - Hides unnecessary information from users:
    - since only part of the base relation (Some collection of attributes, not necessarily all) are to be included in the virtual table.
  - Provide powerful flexibility and security:
    - since unnecessary information will be hidden from the user there will be some sort of data security.
  - Provide customized view of the database for users:
    - each users are going to be interfaced with their own preferred data set and format by making use of the Views.
Cont...

- A view of one base relation can be updated.
- Update on views derived from various relations is not allowed since it may violate the integrity of the database.
- Update on view with aggregation and summary is not allowed.
  - Since aggregation and summary results are computed from a base relation and does not exist actually.
Relational Algebra: What? Why?

- Similar to normal algebra (as in $2 + 3x - y$), except we use relations as values instead of numbers, and the operations and operators are different.
- Not used as a query language in actual DBMSs. (SQL instead.)
- The inner, lower-level operations of a relational DBMS are, or are similar to, relational algebra operations.
- We need to know about relational algebra to understand query execution and optimization in a relational DBMS.
Cont...

- Some advanced SQL queries require explicit relational algebra operations, most commonly *outer join*.
- Relations are seen as *sets of tuples*, which means that no duplicates are allowed.
- SQL is *declarative*, which means that you tell the DBMS *what* you want, but not *how* it is to be calculated.
- A C++ or Java program is some how *procedural*, which means that you have to state, step by step, exactly how the result should be calculated.
Actually, relational algebra is mathematical expressions.
The basic set of operations for the relational model is known as the relational algebra.
These operations enable a user to specify basic retrieval requests.
The result of the retrieval is a new relation, which may have been formed from one or more relations.
• The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra.

• A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query (or retrieval request).

• Relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation.

• The output from one operation can become the input to another operation (nesting is possible).
Comparing RA and SQL

- Relational algebra:
  - is closed (the result of every expression is a relation)
  - has a rigorous foundation
  - has simple semantics
  - is used for reasoning, query optimisation, etc.

- SQL:
  - is a superset of relational algebra
  - has convenient formatting features, etc.
  - provides aggregate functions
  - has complicated semantics
  - is an end-user language.
There are different basic operations that could be applied on relations on a database based on the requirement

- **Selection** ($\sigma$) Selects a subset of rows from a relation.
- **Projection** ($\pi$) Deletes unwanted columns from a relation.
- **Renaming**: assigning intermediate relation for a single operation
- **Cross-Product** ($\times$) Allows us to combine two relations.
- **Join** Tuples joined from two relations based on a condition
### Cont...

<table>
<thead>
<tr>
<th>S.n</th>
<th>Operation</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Projection (pi)</td>
<td>( \pi )</td>
</tr>
<tr>
<td>2</td>
<td>Selection (sigma)</td>
<td>( \sigma )</td>
</tr>
<tr>
<td>3</td>
<td>Renaming (rho)</td>
<td>( \rho )</td>
</tr>
<tr>
<td>4</td>
<td>Cartesian product (times)</td>
<td>( \times )</td>
</tr>
<tr>
<td>S.n</td>
<td>Operation</td>
<td>Symbol</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>5</td>
<td>Join (bow-tie)</td>
<td>☝️</td>
</tr>
<tr>
<td>6</td>
<td>Left outer join</td>
<td>☞</td>
</tr>
<tr>
<td>7</td>
<td>Right outer join</td>
<td>☞</td>
</tr>
<tr>
<td>8</td>
<td>Full outer join</td>
<td>☞</td>
</tr>
<tr>
<td>9</td>
<td>Semijoin</td>
<td>☞</td>
</tr>
</tbody>
</table>
Example

Table 1: Sample table used to illustrate different kinds of relational operations.

The relation contains information about employees, IT skills they have and the school where they attend each skill.
<table>
<thead>
<tr>
<th>EmpID</th>
<th>FName</th>
<th>LName</th>
<th>SkillID</th>
<th>Skill</th>
<th>SkillType</th>
<th>School</th>
<th>SchoolAdd</th>
<th>SkillLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Abebe</td>
<td>Mekuria</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>Lemma</td>
<td>Alemu</td>
<td>5</td>
<td>C++</td>
<td>Programming</td>
<td>Unity</td>
<td>Gerji</td>
<td>6</td>
</tr>
<tr>
<td>28</td>
<td>Chane</td>
<td>Kebede</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>Abera</td>
<td>Taye</td>
<td>6</td>
<td>VB6</td>
<td>Programming</td>
<td>Helico</td>
<td>Piazza</td>
<td>8</td>
</tr>
<tr>
<td>65</td>
<td>Almaz</td>
<td>Belay</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>Helico</td>
<td>Piazza</td>
<td>9</td>
</tr>
<tr>
<td>24</td>
<td>Dereje</td>
<td>Tamiru</td>
<td>8</td>
<td>Oracle</td>
<td>Database</td>
<td>Unity</td>
<td>Gerji</td>
<td>5</td>
</tr>
<tr>
<td>51</td>
<td>Selam</td>
<td>Belay</td>
<td>4</td>
<td>Prolog</td>
<td>Programming</td>
<td>Jimma</td>
<td>Jimma City</td>
<td>8</td>
</tr>
<tr>
<td>94</td>
<td>Alem</td>
<td>Kebede</td>
<td>3</td>
<td>Cisco</td>
<td>Networking</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>7</td>
</tr>
<tr>
<td>18</td>
<td>Girma</td>
<td>Dereje</td>
<td>1</td>
<td>IP</td>
<td>Programming</td>
<td>Jimma</td>
<td>Jimma City</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>Yared</td>
<td>Gizaw</td>
<td>7</td>
<td>Java</td>
<td>Programming</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>6</td>
</tr>
</tbody>
</table>
Selection

- Selects subset of tuples/rows in a relation that satisfy *selection condition*.
- Selection operation is a unary operator (it is applied to a single relation)
- The Selection operation is applied to each tuple individually
- The degree of the resulting relation is the same as the original relation but the cardinality (no. of tuples) is less than or equal to the original relation.
Cont...

- Set of conditions can be combined using Boolean operations (\(\land (\text{AND})\), \(\lor (\text{OR})\), and \(\sim (\text{NOT})\))
- No duplicates in result!
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)
- It is a filter that keeps only those tuples that satisfy a qualifying condition (those satisfying the condition are selected while others are discarded.)
Cont..

- **Notation:**

\[ \sigma \langle \text{Selection Condition} \rangle \langle \text{Relation Name} \rangle \]

**Example:** Find all Employees with skill type of Database.

\[ \sigma \langle \text{SkillType ="Database"} \rangle (\text{Employee}) \]

- This query will extract every tuple from a relation called Employee with all the attributes where the SkillType attribute with a value of “Database”.
The resulting relation will be the following.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>FName</th>
<th>LName</th>
<th>SkillID</th>
<th>Skill</th>
<th>SkillType</th>
<th>School</th>
<th>SchoolAdd</th>
<th>SkillLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Abebe</td>
<td>Mekuria</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>5</td>
</tr>
<tr>
<td>28</td>
<td>Chane</td>
<td>Kebede</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>AAU</td>
<td>Sidist_Kilo</td>
<td>10</td>
</tr>
<tr>
<td>65</td>
<td>Almaz</td>
<td>Belay</td>
<td>2</td>
<td>SQL</td>
<td>Database</td>
<td>Helico</td>
<td>Piazza</td>
<td>9</td>
</tr>
<tr>
<td>24</td>
<td>Dereje</td>
<td>Tamiru</td>
<td>8</td>
<td>Oracle</td>
<td>Database</td>
<td>Unity</td>
<td>Gerji</td>
<td>5</td>
</tr>
</tbody>
</table>
Cont...

- If the query is all employees with a SkillType `Database` and School `Unity` the relational algebra operation and the resulting relation will be as follows.

  \[ \sigma < \text{SkillType} = \text{"Database"} \text{ AND School} = \text{"Unity"}> \ (\text{Employee}) \]

- The resulting relation will be the following.

<table>
<thead>
<tr>
<th>EmpID</th>
<th>FName</th>
<th>LName</th>
<th>SkillID</th>
<th>Skill</th>
<th>SkillType</th>
<th>School</th>
<th>SchoolAdd</th>
<th>SkillLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Dereje</td>
<td>Tamiru</td>
<td>8</td>
<td>Oracle</td>
<td>Database</td>
<td>Unity</td>
<td>Gerji</td>
<td>5</td>
</tr>
</tbody>
</table>
More on Selection

- **Notation:** $\sigma_c(R)$
- **Examples**
  - $\sigma_{\text{Salary} > 40000}$ (Employee)
  - $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)
- The condition $c$ can be $=$, $<$, $\leq$, $>$, $\geq$, $<>$

[in SQL:]
```
SELECT * FROM Employee
WHERE Salary > 40000
```
### Selection Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>DepartmentID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>John</td>
<td>1</td>
<td>30,000</td>
</tr>
<tr>
<td>7777777777</td>
<td>Tony</td>
<td>1</td>
<td>32,000</td>
</tr>
<tr>
<td>8888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>

Find all employees with salary more than $40,000.

\[ \sigma \text{ Salary} > 40000 \ (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>DepartmentID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>
**Projection**

- Selects certain attributes while discarding the other from the base relation.
- The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
- Deletes attributes that are not in projection list.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- If the Primary Key is in the projection list, then duplication will not occur
- Duplication removal is necessary to insure that the resulting table is also a relation.
Cont...

- **Notation:**

\[ \pi \langle \text{Selected Attributes} \rangle \langle \text{Relation Name} \rangle \]

- **Example:** To display Name, Skill, and Skill Level of an employee, the query and the resulting relation will be:

\[ \pi \langle \text{FName, LName, Skill, Skill\_Level} \rangle (\text{Employee}) \]
The resulting relation will be the following.

<table>
<thead>
<tr>
<th>FName</th>
<th>LName</th>
<th>Skill</th>
<th>SkillLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abebe</td>
<td>Mekuria</td>
<td>SQL</td>
<td>5</td>
</tr>
<tr>
<td>Lemma</td>
<td>Alemu</td>
<td>C++</td>
<td>6</td>
</tr>
<tr>
<td>Chane</td>
<td>Kebede</td>
<td>SQL</td>
<td>10</td>
</tr>
<tr>
<td>Abera</td>
<td>Taye</td>
<td>VB6</td>
<td>8</td>
</tr>
<tr>
<td>Almaz</td>
<td>Belay</td>
<td>SQL</td>
<td>9</td>
</tr>
<tr>
<td>Dereje</td>
<td>Tamiru</td>
<td>Oracle</td>
<td>5</td>
</tr>
<tr>
<td>Selam</td>
<td>Belay</td>
<td>Prolog</td>
<td>8</td>
</tr>
<tr>
<td>Alem</td>
<td>Kebede</td>
<td>Cisco</td>
<td>7</td>
</tr>
<tr>
<td>Girma</td>
<td>Dereje</td>
<td>IP</td>
<td>4</td>
</tr>
<tr>
<td>Yared</td>
<td>Gizaw</td>
<td>Java</td>
<td>6</td>
</tr>
</tbody>
</table>
Cont...

- If we want to have the Name, Skill, and Skill Level of an employee with Skill SQL and SkillLevel greater than 5 the query will be:

\[ \pi <FName, LName, Skill, Skill\_Level> (\sigma <Skill=“SQL” \land SkillLevel>5> (Employee)) \]

The resulting relation will be the following.

<table>
<thead>
<tr>
<th>FName</th>
<th>LName</th>
<th>Skill</th>
<th>SkillLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chane</td>
<td>Kebede</td>
<td>SQL</td>
<td>10</td>
</tr>
<tr>
<td>Almaz</td>
<td>Belay</td>
<td>SQL</td>
<td>9</td>
</tr>
</tbody>
</table>
More on Projection

- Eliminates columns, then removes duplicates
- Notation: \( \Pi_{A_1, \ldots, A_n}(R) \)
- Example: project to social-security number and names:
  - \( \Pi_{\text{SSN, Name}}(\text{Employee}) \)
  - Output schema: \( \text{Answer(\text{SSN, Name})} \)

[In SQL: SELECT DISTINCT SSN, Name FROM Employee]
## Projection Example

### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>DepartmentID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>John</td>
<td>1</td>
<td>30,000</td>
</tr>
<tr>
<td>7777777777</td>
<td>Tony</td>
<td>1</td>
<td>32,000</td>
</tr>
<tr>
<td>8888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>

\[ \Pi_{\text{SSN, Name}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>John</td>
</tr>
<tr>
<td>7777777777</td>
<td>Tony</td>
</tr>
<tr>
<td>8888888888</td>
<td>Alice</td>
</tr>
</tbody>
</table>
Find all employees who works in department 4 and whose salary is greater than 25000.

\[ \sigma \text{ dno}=4 \land \text{salary}>25000 \text{ (Employee)} \]
1. Find the employee names and department number of all employees

2. Find the sex and department number of all employees

\[ \Pi_{\text{lname, fname, dno}}(\text{Employee}) \]

\[ \Pi_{\text{sex, dno}}(\text{Employee}) \]

<table>
<thead>
<tr>
<th>f_name</th>
<th>l_name</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joseph</td>
<td>Chan</td>
<td>4</td>
</tr>
<tr>
<td>Victor</td>
<td>Wong</td>
<td>5</td>
</tr>
<tr>
<td>Carrie</td>
<td>Kwan</td>
<td>4</td>
</tr>
<tr>
<td>Joyce</td>
<td>Fong</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sex</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>4</td>
</tr>
<tr>
<td>M</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>4</td>
</tr>
</tbody>
</table>
**Rename Operation**

- In relational algebra, a **rename** is a unary operation written as $\rho_{a/b}(R)$ where:
  - $a$ and $b$ are attribute names
  - $R$ is a relation

- The result is identical to $R$ except that the $b$ field in all tuples is renamed to an $a$ field.

- For example, consider the following *Employee* relation and its renamed version:
Cont...

**Employee**

<table>
<thead>
<tr>
<th>Name</th>
<th>EmployeeId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>3415</td>
</tr>
<tr>
<td>Sally</td>
<td>2241</td>
</tr>
</tbody>
</table>

\[ \rho_{\text{EmployeeName} / \text{Name}}(\text{Employee}) \]

<table>
<thead>
<tr>
<th>EmployeeName</th>
<th>EmployeeId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry</td>
<td>3415</td>
</tr>
<tr>
<td>Sally</td>
<td>2241</td>
</tr>
</tbody>
</table>
Rename Operation (cont...) 

- We may also want to apply several relational algebra operations one after the other.
- The query could be written in two different forms:
  - Write the operations as a single relational algebra expression by nesting the operations.
  - Apply one operation at a time and create intermediate result relations.
    - In the latter case, we must give names to the relations that hold the intermediate results

- If we want to have the Name, Skill, and Skill Level of an employee with salary greater than 1500 and working for department 5, we can write the expression for this query using the two alternatives:
Cont...

- A single algebraic expression

- Using an intermediate relation by the Rename Operation

\[ \pi <\text{FName, LName, Skill, Skill\_Level}> (\sigma <\text{DeptNo}=5 \land \text{Salary}>1500> (\text{Employee})) \]

**Step 1:** \( \text{Result1} \leftarrow \sigma <\text{DeptNo}=5 \land \text{Salary}>1500> (\text{Employee}) \)

**Step 2:** \( \text{Result} \leftarrow \pi <\text{FName, LName, Skill, Skill\_Level}> (\text{Result1}) \)

Then Result will be equivalent with the relation we get using the first alternative.
More on Renaming

- Changes the schema, not the instance
- Schema: $R(A_1, \ldots, A_n)$
- Notation: $\rho_{B_1,\ldots,B_n}(R)$
- Example:
  - $\rho_{\text{Name, SSN}}(\text{Employee})$
  - Output schema: Answer(\text{Name, SSN})

[in SQL: SELECT Name AS \text{Name}, SSN AS \text{SSN} FROM \text{Employee}]

### Renaming Example

<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

\[ \rho_{\text{LastName}, \text{SocSocNo}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>LastName</th>
<th>SocSocNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>
CARTESIAN (cross product) Operation

- This operation is used to combine tuples from two relations in a combinatorial fashion.
- That means, every tuple in Relation1 (R) will be related with every other tuple in Relation2 (S).
- In general, the result of \( R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m) \) is a relation \( Q \) with degree \( n + m \) attributes
  - \( Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m) \), in that order.
  - Where \( R \) has \( n \) attributes and \( S \) has \( m \) attributes.
- The resulting relation \( Q \) has one tuple for each combination of tuples—one from \( R \) and one from \( S \).
- Hence, if \( R \) has \( n \) tuples, and \( S \) has \( m \) tuples, then \( | R \times S | \) will have \( n \times m \) tuples.
### Example

#### Employee

<table>
<thead>
<tr>
<th>ID</th>
<th>FName</th>
<th>LName</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Abebe</td>
<td>Lemma</td>
</tr>
<tr>
<td>567</td>
<td>Belay</td>
<td>Taye</td>
</tr>
<tr>
<td>822</td>
<td>Kefle</td>
<td>Kebede</td>
</tr>
</tbody>
</table>

#### Dept

<table>
<thead>
<tr>
<th>DeptID</th>
<th>DeptName</th>
<th>MangID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Finance</td>
<td>567</td>
</tr>
<tr>
<td>3</td>
<td>Personnel</td>
<td>123</td>
</tr>
</tbody>
</table>

**Then the Cartesian product between Employee and Dept relations will be of the form**

<table>
<thead>
<tr>
<th>ID</th>
<th>FName</th>
<th>LName</th>
<th>DeptID</th>
<th>DeptName</th>
<th>MangID</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Abebe</td>
<td>Lemma</td>
<td>2</td>
<td>Finance</td>
<td>567</td>
</tr>
<tr>
<td>123</td>
<td>Abebe</td>
<td>Lemma</td>
<td>3</td>
<td>Personnel</td>
<td>123</td>
</tr>
<tr>
<td>567</td>
<td>Belay</td>
<td>Taye</td>
<td>2</td>
<td>Finance</td>
<td>567</td>
</tr>
<tr>
<td>567</td>
<td>Belay</td>
<td>Taye</td>
<td>3</td>
<td>Personnel</td>
<td>123</td>
</tr>
<tr>
<td>822</td>
<td>Kefle</td>
<td>Kebede</td>
<td>2</td>
<td>Finance</td>
<td>567</td>
</tr>
<tr>
<td>822</td>
<td>Kefle</td>
<td>Kebede</td>
<td>3</td>
<td>Personnel</td>
<td>123</td>
</tr>
</tbody>
</table>
Cont...

- Basically, even though it is very important in query processing, the Cartesian Product is not useful by itself since it relates every tuple in the First Relation with every other tuple in the Second Relation.

- Thus, to make use of the Cartesian Product, one has to use it with the Selection Operation, which discriminate tuples of a relation by testing whether each will satisfy the selection condition.
• In our example, to extract employee information about managers of the departments (Managers of each department), the algebra query and the resulting relation will be.

\[ \pi \langle ID, FName, LName, DeptName \rangle (\sigma <ID=MangID>(Employee \land Dept)) \]

<table>
<thead>
<tr>
<th>ID</th>
<th>FName</th>
<th>LName</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Abebe</td>
<td>Lemma</td>
<td>Personnel</td>
</tr>
<tr>
<td>567</td>
<td>Belay</td>
<td>Taye</td>
<td>Finance</td>
</tr>
</tbody>
</table>
More on Cartesian Product

- Combine each tuple in R1 with each tuple in R2
- Notation: \( R1 \times R2 \)
- Example:
  - Employee \( \times \) Dependents
- Very rare in practice; mainly used to express joins

[In SQL: SELECT * FROM R1, R2]
**Cartesian Product Example**

<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dependents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EmployeeSSN</td>
<td>Dname</td>
</tr>
<tr>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee x Dependents</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
<td>EmployeeSSN</td>
<td>Dname</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Set difference

- The set difference operation: - finds tuples in one relation but not in other. It is denoted as \(-\) (minus)

- Example:
  Find the names of all customers who have an account but not a loan

\[ \pi \text{customer-name (Depositor)} - \pi \text{customer-name (Borrower)} \]
JOIN Operation

• The sequence of Cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called \textit{JOIN}.
• Thus in JOIN operation, the Cartesian Operation and the Selection Operations are used together.

• JOIN Operation is denoted by a \textbullet\textcircled{\textbullet} symbol. This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
• The general form of a join operation on two relations \(R(A_1, A_2, \ldots, A_n)\) and \(S(B_1, B_2, \ldots, B_m)\) is:
\textbf{Cont...}

\begin{itemize}
\item $R \bowtie <\text{join condition}> S$
\end{itemize}

\textit{is equivalent to}

\[ \sigma <\text{selection condition}> (R \times S) \]

\textit{where} $<\text{join condition}>$ \textit{and} $<\text{selection condition}>$ \textit{are the same}
Where, R and S can be any relation that results from general relational algebra expressions.

Since JOIN is an operation that needs two relation, it is a Binary operation.

This type of JOIN is called a **THETA JOIN (θ-JOIN)**. Where \( \theta \) is the logical operator used in the join condition.

\( \theta \) Could be \{ <, \leq, >, \geq, \neq, = \}
Example:

- Thus in the above example we want to extract employee information about managers of the departments, the algebra query using the JOIN operation will be.

\[ \text{Employee} \bowtie_{<ID=MangID>} \text{Dept} \]
EQUIJOIN Operation

- The most common use of join involves join conditions with equality comparisons only (\(=\)).
- Such a join, where the only comparison operator used is called an EQUIJOIN.
- In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple since we used the equality logical operator.
- For example, the above JOIN expression is an EQUIJOIN since the logical operator used is the equal to operator (\(=\)).
NATURAL JOIN Operation

- We have seen that in EQUIJOIN one of each pair of attributes with identical values is extra, a new operation called natural join was created to get rid of the second (or extra) attribute that we will have in the result of an EQUIJOIN condition.
- The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations.
- If this is not the case, a renaming operation on the attributes is applied first.
OUTER JOIN Operation

- OUTER JOIN is another version of the JOIN operation where non matching tuples from a relation are also included in the result with NULL values for attributes in the other relation.
- There are two major types of OUTER JOIN.
  - **RIGHT OUTER JOIN**: where non matching tuples from the second (Right) relation are included in the result with NULL value for attributes of the first (Left) relation.
  - **LEFT OUTER JOIN**: where non matching tuples from the first (Left) relation are included in the result with NULL value for attributes of the second (Right) relation.
Cont...

- When two relations are joined by a JOIN operator, there could be some tuples in the first relation not having a matching tuple from the second relation, and the query is interested to display these non matching tuples from the first or second relation.
- Such query is represented by the OUTER JOIN.
SEMIJOIN Operation

- SEMI JOIN is another version of the JOIN operation where the resulting Relation will contain those attributes of only one of the Relations that are related with tuples in the other Relation.
- The following notation depicts the inclusion of only the attributes form the first relation (R) in the result which are actually participating in the relationship.
End of Chapter Three