Relational Database Systems
RDBMS

Introduction to RDBMS

- RDBMS is the dominant data-processing software in use today
- This software represents the second generation of DBMSs
- Based on the relational data model proposed by E. F. Codd (1970)
- In the relational model,
  - all data is logically structured within relations (tables).
  - Each relation has a name
  - A Relation is made up of named attributes (columns) of data.
  - Each tuple (row) contains one value per attribute.

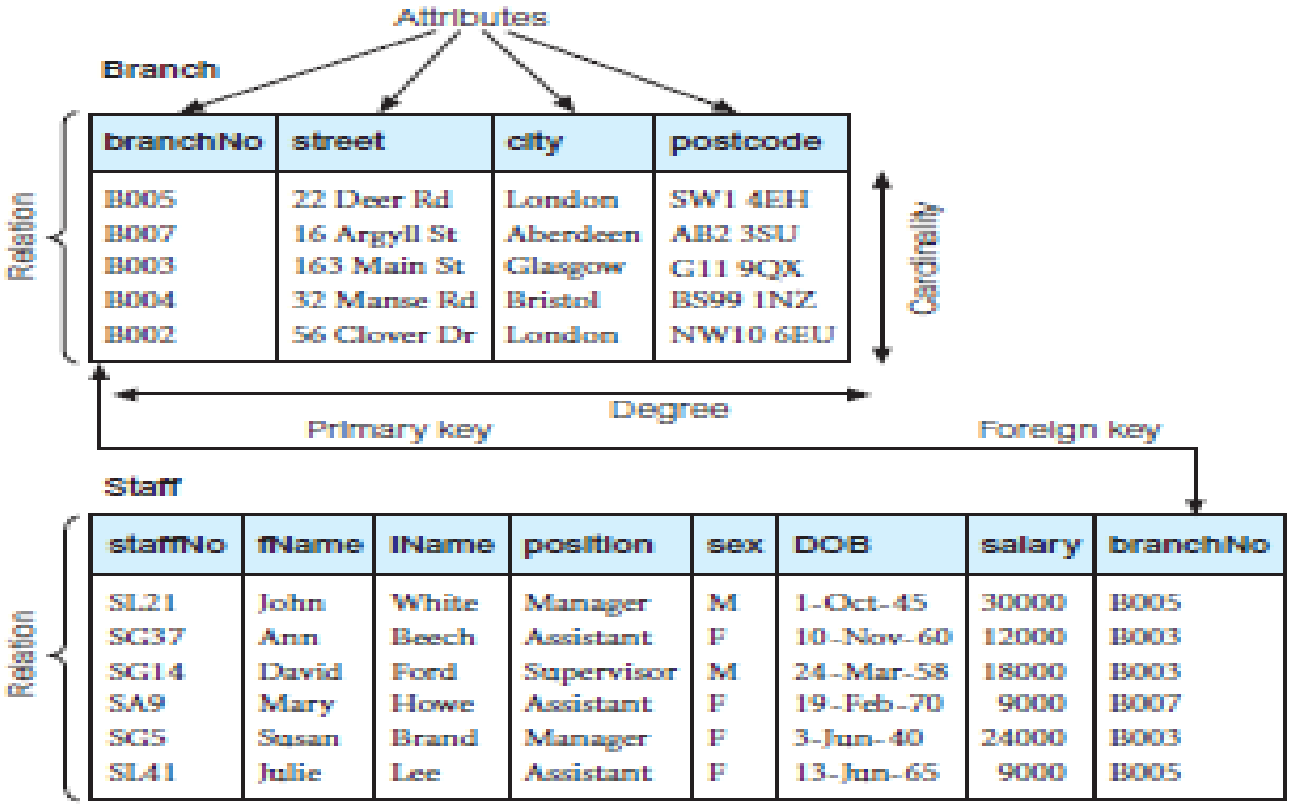
Objectives of the Relational Data Model

The relational model's objectives were specified as follows:

- To allow a high degree of data independence.
- To provide substantial grounds for dealing with
  - Data semantics
  - consistency, and
  - redundancy problems.
- introduced the concept of **normalized** relations
  - relations that have no repeating groups.
- To enable the expansion of set-oriented data manipulation languages
  - SQL

Terminologies

- **Relation**: A relation is a table with columns and rows.
- **Attribute**: An attribute is a named column of a relation.



Terminologies (contd.)

- **Domain**:
  - A domain is the set of allowable values for an attribute
  - Every attribute in a relation is defined on a domain.
  - Domains may be distinct for each attribute
    - or two or more attributes may be defined on the same domain.

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

Terminologies (contd.)

- **Tuple**
  - A tuple is a row of a relation.
- **Degree**
  - The degree of a relation is the number of attributes it contains.
- **Cardinality**
  - The cardinality of a relation is the number of tuples it contains.
- **Relational database**
  - A collection of normalized relations with distinct relation names.

Alternative Terminology

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Database Relations

- Relation schema
  - A named relation defined by a set of attribute and domain name pairs.
- A relation R defined by a relation schema S
  - is a set of mappings from the attribute names to their corresponding domains
- Let A1, A2, . . . , An be attributes with domains D1, D2, . . . , Dn
  - Then the set {A1:D1, A2:D2, . . . , An:Dn} is a relation schema.
  - Thus, relation R is a set of n-tuples:

  (A1:d1, A2:d2, . . . , An:dn)

  such that d1 $\in$ D1, d2 $\in$ D2, . . . , dn $\in$ Dn
  - Each element in the n-tuple consists of an attribute and a value for that attribute.

Database Relations (contd.)

- When we write out a relation as a table
  - we list the attribute names as column headings and
  - write out the tuples as rows having the form (d1, d2, . . . , dn), where each value is taken from the appropriate domain.
- The relation is any subset of the Cartesian product of the domains
- A table is simply a physical representation of such a relation.

Example

- The Branch relation is any subset of the Cartesian product of the domains
  - any set of four-tuples
  - the first element is from the domain BranchNumbers,
  - the second is from the domain StreetNames, and so on.
- One of the four-tuples is:

{(B005, 22 Deer Rd, London, SW1 4EH)} or more correctly:

{ (branchNo: B005, street: 22 Deer Rd, city: London, postcode: SW1 4EH), (branchNo: B007, street: 16 Argyll St, city: Aberdeen, postcode: AB2 3SU
}

- We refer to this as a relation instance
- Q: What is the difference between a Relation Schema and Relation Instance?

Relational Database Schema

- A set of relation schemas, each with a distinct name
- A relational database consists of any number of normalized relations
- The **conceptual model**, or **conceptual schema**, is the set of all such schemas for the database
- If R1, R2, . . . , Rn are a set of relation schemas, then we can write the relational database schema

  R = {R1, R2, . . . , Rn}

Example:

- Relational schema for HiLCoE can be represented as:

  Student (StudentCode, fName, lName, gender, DOB, batch)

  Course(CourseCode, CourseTitle, CreditHours)

  Registration (StudentCode, CourseCode, RegDate, Grade)

Properties of Relations

- The relation has a name that is distinct from all other relation names in the relational schema;
- Each cell of the 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;
- Order of attributes has no significance;
- Order of tuples has no significance

Relational Keys

- **Superkey**: An attribute, or set of attributes, that uniquely identifies a tuple within a relation.
- **Candidate key**: A superkey such that no proper subset is a superkey within the relation.
- A candidate key, K, for a relation R has two properties:
  - **uniqueness** – in each tuple of R, the values of K uniquely identify that tuple;
  - **irreducibility** – no proper subset of K has the uniqueness property.
- There may be several candidate keys for a relation.
- When a key consists of more than one attribute, we call it a **composite key**.

Relational Keys (cont.)

- **Primary key**: The candidate key that is selected to identify tuples uniquely within the relation.
- **Alternate keys**: The candidate keys that are not selected to be the primary key
- **Foreign key**: An attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation.

- When an attribute appears in more than one relation, its appearance usually represents a relationship between tuples of the two relations.

Integrity Rules

NULL

- NULL represents a value for an attribute that is currently unknown or is not applicable

- A NULL can be taken to mean the logical value ‘unknown’.
- It can mean that a value is not applicable to a particular tuple
- It could mean that no value has yet been supplied.
- Nulls are a way to deal with incomplete or exceptional data

Entity Integrity

- **Entity integrity:** In a base relation, no attribute of a primary key can be null.

- By definition, a primary key is a minimal identifier that is used to identify tuples uniquely.
  - This means that no subset of the primary key is sufficient to provide unique identification of tuples.
- If we allow a null for any part of a primary key,
  - we are implying that not all the attributes are needed to distinguish between tuples,
  - which contradicts the definition of the primary key.

Referential Integrity

- **Foreign key**: An attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation

- If a foreign key exists in a relation
  - either the foreign key value must match a candidate key value of some tuple in its home relation or
  - the foreign key value must be wholly NULL

.

General Constraints

- Additional rules specified by the users or database administrators of a database that define or constrain some aspect of the enterprise.
- It is also possible for users to specify additional constraints that the data must satisfy.

Relational Algebra

Relational Algebra

- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Relational Algebra is a query language which takes relations as an input and returns relation as an output.
- It uses operators to perform queries.
- Both operands and results are relations
  - Output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic.
- This property is called **closure**.

Relational Algebra (cont.)

- Relational algebra and relational calculus are formal languages associated with the relational model.
- Relational algebra is a procedural language
- Relational calculus a non-procedural language
  - However, formally both are equivalent to one another.
- A language that produces a relation that can be derived using relational calculus is relationally complete.

Relational Algebra (cont.)

- Relational operators can be either unary or binary
- Five basic operations in relational algebra:
  - Selection
  - Projection
  - Cartesian product
  - Union
  - Set Difference
  - Rename
- These perform most of the data retrieval operations needed.
- Also have Join, Intersection, and Division operations,
  - can be expressed in terms of the 5 basic operations.

Relational Calculus

- The purpose of relational calculus is to provide a formal basis for defining declarative query languages appropriate for relational databases.
- The basic difference between relational algebra and relational calculus is
  - Relational Algebra gives the procedure of how to evaluate the query whereas
  - Relational Calculus gives only the query without giving the procedure of how to evaluate the query

Relational Calculus (cont.)

- Relational calculus is nonoperational, and users define queries in terms of what they want
  - not in terms of how to compute it
- Relational calculus is Declarative
- Relational Calculus and Relational Algebra:
  - A relational calculus query specifies what information is retrieved
  - A relational algebra query specifies how information is retrieved

Selection (Restriction) Operator

- Selection operator is used to select tuples from a relation based on some condition
- Can be considered as row wise filtering
- Symbol: Greek letter σ [Sigma]
- $\sigma_{predicate}(R)$
  - Works on a single relation R
  - Defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (predicate).

Selection Operator - Example

- Extract students whose age is greater than 18 from STUDENT relation

  $\sigma_{(AGE>18)}(STUDENT)$

- List all staff with a salary greater than £10,000

  $\sigma_{salary > 10000}(Staff)$

Selection Operator (cont.)

Student

Roll. No	Name	GPA
001	Genet	3.50
002	Solomon	2.92
003	Hanna	3.50
004	Daniel	3.72

- $\sigma_{GPA > 3.2}$ (Student)

Roll. No	Name	GPA
001	Genet	3.50
003	Hanna	3.50
004	Daniel	3.72

Projection Operator

- Projection operator is used to project particular columns from a relation
- Symbol: Greek letter ∏ [Pi]
- $\prod_{col1, \ldots, coln}(R)$
  - Works on a single relation R (Unary Operator)
  - Defines a relation that contains a vertical subset of R
    - extracting the values of specified attributes and eliminating duplicates.
- Example
  - Produce a list of GPAs of students

    $\prod_{GPA}(Student)$
- Give the Role Numbers and GPA of Students

Rename Operator (cont.)

- Rename operator is used to give another name to a relation
- ρ(Relation2, Relation1)
- To rename STUDENT relation to STUDENT1, we can use rename operator like:
- ρ(STUDENT1, STUDENT)
- Examples:

  $\rho(\text{Student_Names}, \Pi_{\text{Name}}(\text{Student}))$

  $\rho(\text{Honor_List}, \Pi_{\text{Name}}(\sigma_{\text{GPA} > 3.0}(\text{Student})))$

  $\rho_{x}(E)$: returns the result of expression E under the name X

  $\rho_{x(A1, A2, A3)}(E)$

  - returns the result of expression E under the name X, and with the attributes renamed to A1 , A2 , …., An

Union Operator

- R ∪ S
  - Union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated.
  - R and S must be union-compatible.
- Two relation are **Union Compatible** if they have the same number of attributes and corresponding attributes in two relations are from the same domain
- If R and S have I and J tuples, respectively, their union will have a **maximum** of (I + J) tuples.
- Examples
  - Student ∪ Employee
  - $\Pi_{KKetema}(Student) \cup \Pi_{KKetema}(Employee)$

Set Difference

- R – S
- Defines a relation consisting of the tuples that are in relation R, but not in S.
- R and S must be **union-compatible**

- Example
- List all Kifle Ketema where there is a student but not an employee.

$$\Pi_{KKetema}(Student) - \Pi_{KKetema}(Employee)$$

Intersection

- $R \cap S$
- Defines a relation consisting of the set of all tuples that are in both R and S.
- R and S must be **union-compatible**.
- Expressed using basic operations:

  $R \cap S = R - (R - S)$
- **Example**
- List all Kifle Ketema where there is a student and also an employee.

  $\Pi_{KKetema}(Student) \cap \Pi_{KKetema}(Employee)$

Division Operator R ÷ S

- The division operation extracts records and fields from one table on the basis of data in the second table
  - Division operator A÷B can be applied if and only if the attributes of B is proper subset of Attributes of A.
  - The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The result of A ÷ B consists of
  - the attribute names unique to A, i.e., in the header of A but not in the header of B,
  - for which the tuples from A are associated to every B's tuple.

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

A

propertyNo
PG4
PG36

B

clientNo
CR56

A ÷ B

Cross Product Operator

- Also known as **Cartesian Product**
- For every row of Relation1, each row of Relation2 is concatenated (matched)
- If Relation1 has m tuples and Relation2 has n tuples, cross product of Relation1 and Relation2 will have m X n tuples
- Syntax:

  Relation1 X Relation2
- Cartesian product and Selection can be reduced to a single operation called a **JOIN**

Join Operations

- Various forms of join operation
  - Theta join
  - Equijoin (a particular type of Theta join)
  - Natural join
  - Outer join
  - Semijoin

Theta Join (θ -join)

- $R \bowtie_p S$
- Defines a relation that contains tuples satisfying the predicate p from the Cartesian product of R and S
- The predicate p is of the form

  $R.ai \ \theta \ S.bi$ where
  - θ may be one of the operators (<, $\leq$, >, $\geq$, =, $\neq$), and
  - ai & bi are attributes of R and S Respectively
- Can rewrite Theta join using basic Selection and Cartesian product operations.

  $R \bowtie_F S = \sigma_F(R \times S)$
- Degree of a Theta join is sum of degrees of the operand relations R and S.
- The cardinality?

Equi Join

- If predicate for a Theta Join contains only equality (=), the term Equijoin is used.
- Equi Join is a special case of condition joins where the condition C contains only equality.

Example

- Given the two relations Employee and Department, produce a list of all employees along with their departments
- Employee $\bowtie_{\text{E.DepID = Dep.DepID}}$ Department

Example

- A customer wants to buy a car and a boat, but she does not want to spend more for the boat than for the car
- The θ-join (⋈θ) on the relation CarPrice ≥ BoatPrice produces a table with all the possible options

Car ⋈ Boat

CarPrice >= BoatPrice

Car

CarModel	CarPrice
CarA	20,000
CarB	30,000
CarC	50,000

Boat

BoatModel	BoatPrice
Boat1	10,000
Boat2	40,000
Boat3	60,000

$Car \bowtie Boat$

$CarPrice \geq BoatPrice$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

Natural Join

- The natural join performs an equi join of the two relations R and S over all common attributes.
  - One occurrence of each common attribute is eliminated from the result.
  - In other words a natural join will remove duplicate attribute.
- In most systems a natural join will require that the attributes have the same name to identity the attributes to be used in the join

Natural Join (cont.)

- The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Outer Joins

- Outer joins are an extension of the join operation that avoids loss of information
- Outer joins compute the join and then add tuples form one relation that does not match tuples in the other relation to the result of the join.
- NULL values are used to represent unmatched tuples

Outer Joins [2]

- Outer joins are used to display rows that do not have matching values in the join column
- R ⟕ S Left Outer Join
  - Left Outer join is join in which tuples from R (left relation) that do not have matching values in common columns of S are also included in the result relation
- R ⟖ S Right Outer Join
  - Right outer join is join in which tuples from S (right relation) that do not have matching values in common columns of R are also included in result relation.
- R ⟗ S Full Outer
  - Full outer join is join in which tuples from both R and S that do not have matching values in common columns of S and R respectively are also included in result relation.

Left Outer Joins - Example

- Employee ⟕ Dept

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟕ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet
George	3401	Finance	NULL
Harriet	2202	Sales	Harriet
Tim	1123	Executive	NULL

- Dept ⟕ Employee ?

Right Outer Joins

- Employee ⟖ Dept
- The result of the right outer join is the set of all combinations of tuples in R and S
  - that are equal on their common attribute names,
  - in addition to tuples in S that have no matching tuples in R

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟖ Dept

Name	EmpId	DeptName	Manager
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
NULL	NULL	Production	Charles

Full Outer Joins

- R ⟗ S
- The result of the full outer join is the set of all combinations of tuples in R and S
  - that are equal on their common attribute names, and
  - tuples in S that have no matching tuples in R and
  - tuples in R that have no matching tuples in S in their common attribute names.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟗ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet
George	3401	Finance	NULL
Harriet	2202	Sales	Harriet
Tim	1123	Executive	NULL
NULL	NULL	Production	Charles

Semi Join

- Defines a relation that contains the tuples of R that participate in the join of R with S
- The result is the set of all tuples in R for which **there is a tuple in S** that is equal on their common attribute names.
- Can rewrite Semi join using Projection and Join:
- $R \ltimes S = \Pi_A(R \bowtie_F S)$
  - Where A is the set of attributes of R

- The difference from a natural join is that **other columns of S do not appear**.

Semi Join - Example

- Employee $\ltimes$ Dept

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

DeptName	Manager
Sales	Sally
Production	Harriet

Employee $\ltimes$ *Dept*

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

- Dept $\ltimes$ Employee ?