Relational Model Concepts

- It has become the industry standard in business applications.
- First proposed by E. F. Codd.
- The objectives:
  - High degree of data independence.
  - Dealing with data semantics, consistency and redundancy problems.
  - Use of set-oriented DML.

Terminology

- Relationship:
  - A relationship is a table with column and rows
  - A row is also called as a **tuple** or **record**.
  - A column is also called as an **attribute** or **field**.
  - A relationship is also called a **table** or **file**.
  - The **degree** of a relationship is the number of attributes it contains.
  - The **cardinality** of a relationship is the number of tuples it contains.
  - A **relational database** is a set of tables.

- Domain:
  - A domain D is a set of atomic values .
  - A method of specifying a domain is to specify a data type.
  - An integer value can be drawn from the set of integers.
  - E.g .
  - 1. Mobile number: A valid 10 Digit number
  - 2. Employee age: must be between 18 to 62.
  - 3. Landline number: a valid 7 digit number etc.

- Tuple:
  - A tuple is a row of relation.
  - Tuple in a relation can appear in any order and the relation will still be the same relation.

- Nulls:
  - Null represents a value of a data item that is currently not available or not applicable.
  - A zero value for integer should not be treated as a null value.

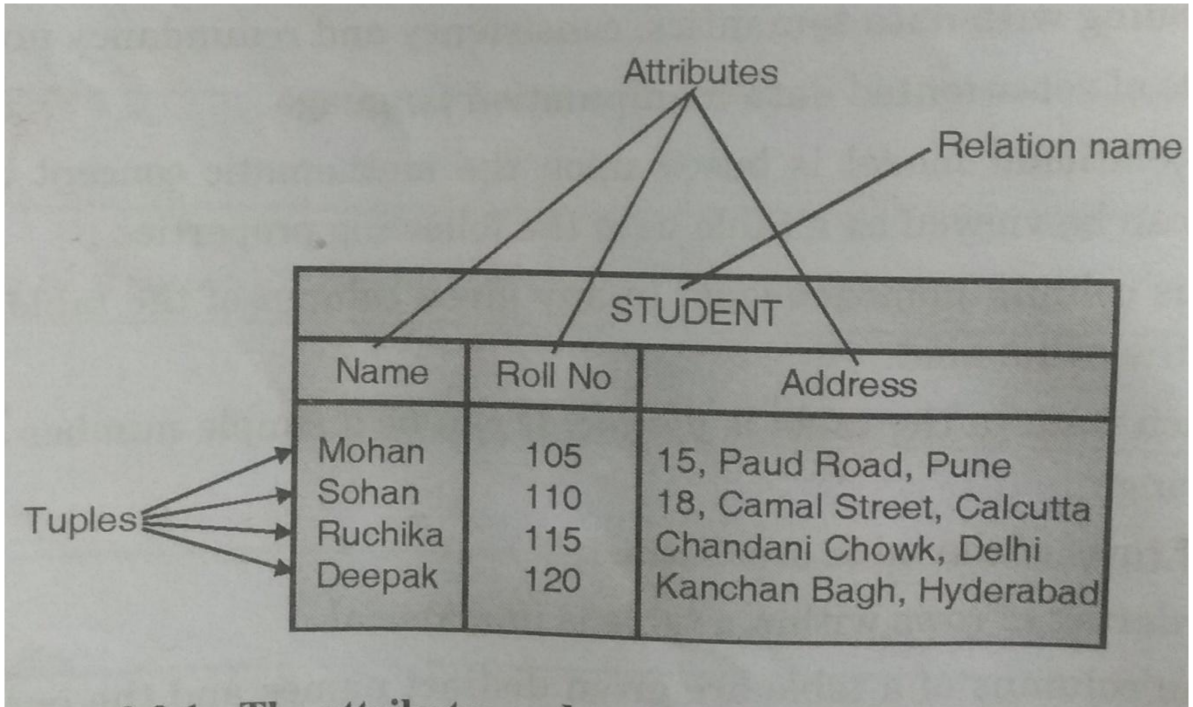
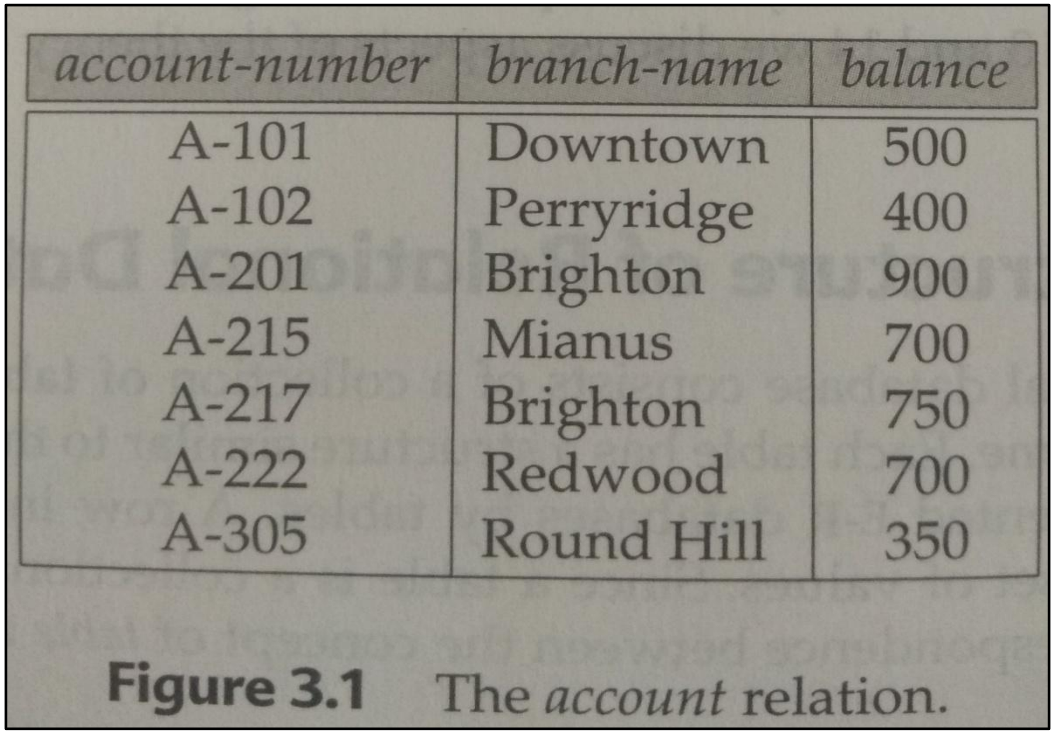


Fig. 3.2.1 : The attributes and tuples of a relation STUDENT

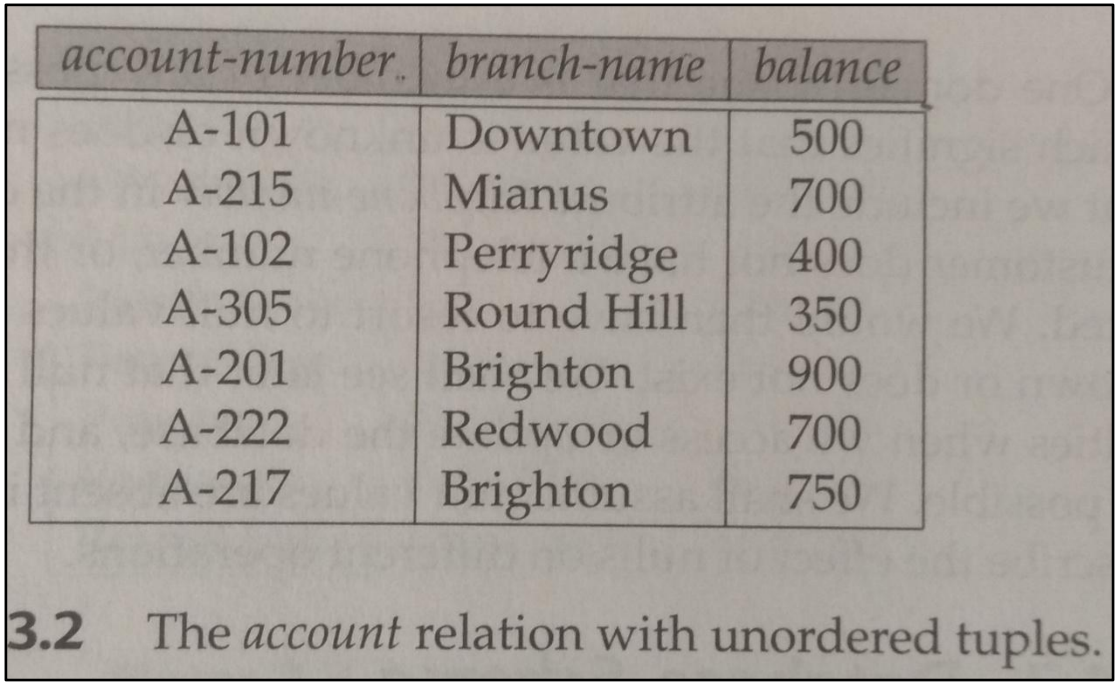
Basic Structure

- Consider the account table from fig. with three columns account-number, branch-name and balance.
- The columns are referred as attributes.
- For every attribute, there is a set of permitted values called a s domain of that attribute.
- E.g. the attribute branch-name, the domain set will be the names of all the branches of the bank.



account-number	*branch-name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Figure 3.1 The *account* relation.



account-number	*branch-name*	*balance*
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

3.2 The *account* relation with unordered tuples.

- Let D1 denote the set of all account numbers.
- D2 → set of all branch names
- D3 → set of all balances
- Any row of accounts must consists of 3 tuple (v1,v2,v3).
- Where v1 is account-number(v1 is in domain D1)
- Where v2 is branch-name (v2 is in domain D2)
- Where v3 is Balance (v3 is in domain D3)

- In general account will contain only a subset of the set of all possible rows. Therefore account is a subset of

  D1 X D2 X D3

- In general, a table of n attributes must be a subset of,

  D1 X D2 X D3 X………..X Dn-1 X Dn

- We shall use the mathematical terms **relation** and **tuple** in place of the terms **table** and **row**.
- A tuple variable is a variable that stands for a tuple.

  i.e. a tuple variable is a variable whose domain is the set of all tuples.

- In the account relation there are seven tuples.
- Let the tuple variable t refers to the first tuple of the relation.
- We use the notation t[account-number] to denote the value of t on account number attribute.
- Since the relation is a set of tuples , we use the mathematical notation t Є r to denote that tuple t is in relation r.
- The order in which the tuples appear in the relation is irrelevant. See fig. 3.1 & 3.2.

- We require for all relations r, the domains of all attributes of r be **atomic**.
- The domain is atomic, if elements of the domain are considered to be indivisible units.
- It is possible for several attributes to have the same domain.
- E.g. consider relations customer and employee.

  customer-name and employee-name may have same domain.

Database Schemas

- Database **Schemas** – Logical design of a database
- Database **Instance** - Snapshot of data in the database at a given instant in time.
- The concept of **relation** corresponds to the programming-language notion of a variable.
- The concept of **relation schema** corresponds to the programming-language notion of type definition.

- It is convenient to give name to a relation schema.
- We use lower case names for relations and names beginning with upper case letter for relation schemas.

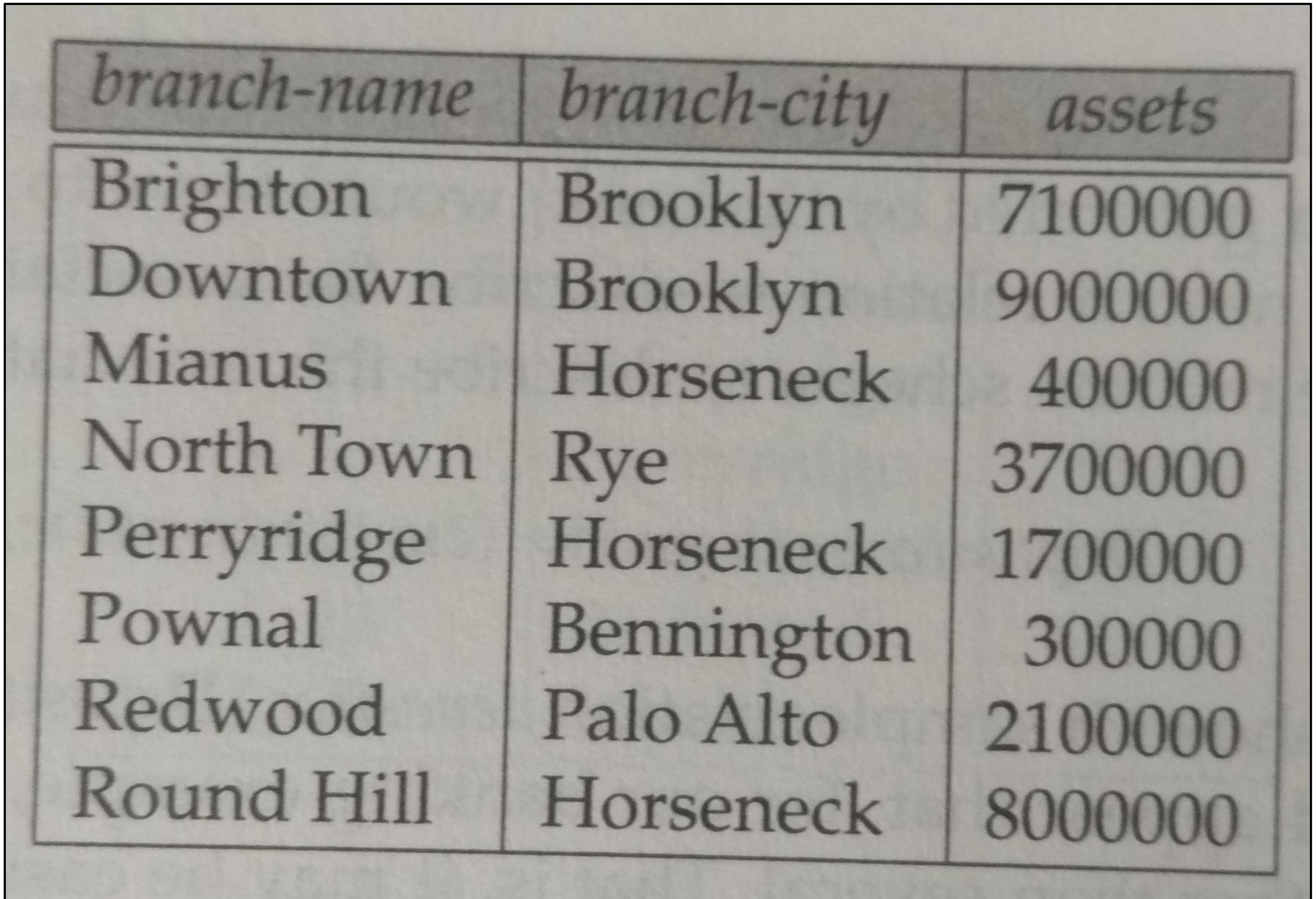
E.g. Account-schema = (account-number, branch-name, balance)

- We denote the fact that account is a relation on Account -schema by:

Account(Account-schema)

- The concept of a relation instance corresponds to the programming-language notion of value of a variable.
- The value of a given variable may change with time similarly the contents of a relation instance may change with time as the relation is updated.

  E.g. consider the branch relation in following figure:



branch-name	*branch-city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Figure 3.3 The *branch* relation.

Relational Modeling Constraints

- The integrity is the very essential property that data in the database should possess at any instance.
- The integrity constraints are the restrictions applied on the data so that the changes made to the database by authorized users do not result in loss of data consistency.
- Example: Consider the relations **Item** and **Transaction**.

Item			
ItemCd	**IName**	**Op_Stock**	**Cr_Stock**
I1	Sheets	100	105
I2	Nuts	50	35
I3	Bolts	75	75
I4	Panels	32	32

Transaction			
TNo	**TType**	**ItemCd**	**Qty**
101	Receipt	I1	50
102	Issue	I2	10
103	Issue	I1	110
104	Receipt	I1	65
105	Issue	I2	5

- ❑ At any point of time the current stock (Cur_Stock) of any item must be equal to the opening stock (Op_Stock) of that item plus sum of all the receipts of the item minus the sum of all the issue transactions of that item.
- ❑The database is said to be consistent if it agrees with above conditions at any instance.
- ❑There are many types of integrity constraints applicable at various situations as follows:

❑ Domain Constraints:

- ✓ For a given application an attribute is allowed to take a value from a set of permissible values. This set of allowable values for the attribute is called as domain of the attribute.

❑ Referential Integrity Constraint:

- ✓ It states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

  E.g. Department and Employee tables as follows:

  - ❖ **Department (deptno,deptname)**
  - ❖ **Employee (Empno, name, address, deptno)**

- ✓ deptno is foreign key n Employee table.
- ✓ So while entering tuple in Employee relation, department value is compaired Department tuples.
- ✓ If exsisting,   insertion in Employee is allowed.
- ✓ If the value of deptno I not present in the Department relation then system will not allow users to store the record in Employee relation.

❑ Entity Integrity Constraints:

- ✓ This is a specialization to the domain constraints for null values.
- ✓ It states that the primary key attribute(s) cannot have a null value in any tuple.
- ✓ In SQL when an attribute is defined as a primary key of a relation, the not null entity integrity constraint is automatically applied.

❑ Key Constraint:

- ✓ It states that the primary key value must be unique. It is not allowed to repeat.

Relational Algebra

- It is a procedural query language.
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- The fundamental operations in the relational algebra are:

1) Selection
2) Projection
3) Union
4) Set Difference
5) Cartesian Product
6) Insertion
7) Division
8) Assignment
9) Join
10) Rename

branch-name	*branch-city*	*assets*
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Figure 3.3 The *branch* relation.

customer-name	*account-number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Figure 3.5 The *depositor* relation.

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Figure 3.6 The *loan* relation.

customer-name	*loan-number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Figure 3.7 The *borrower* relation.

customer-name	*customer-street*	*customer-city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Figure 3.4 The *customer* relation.

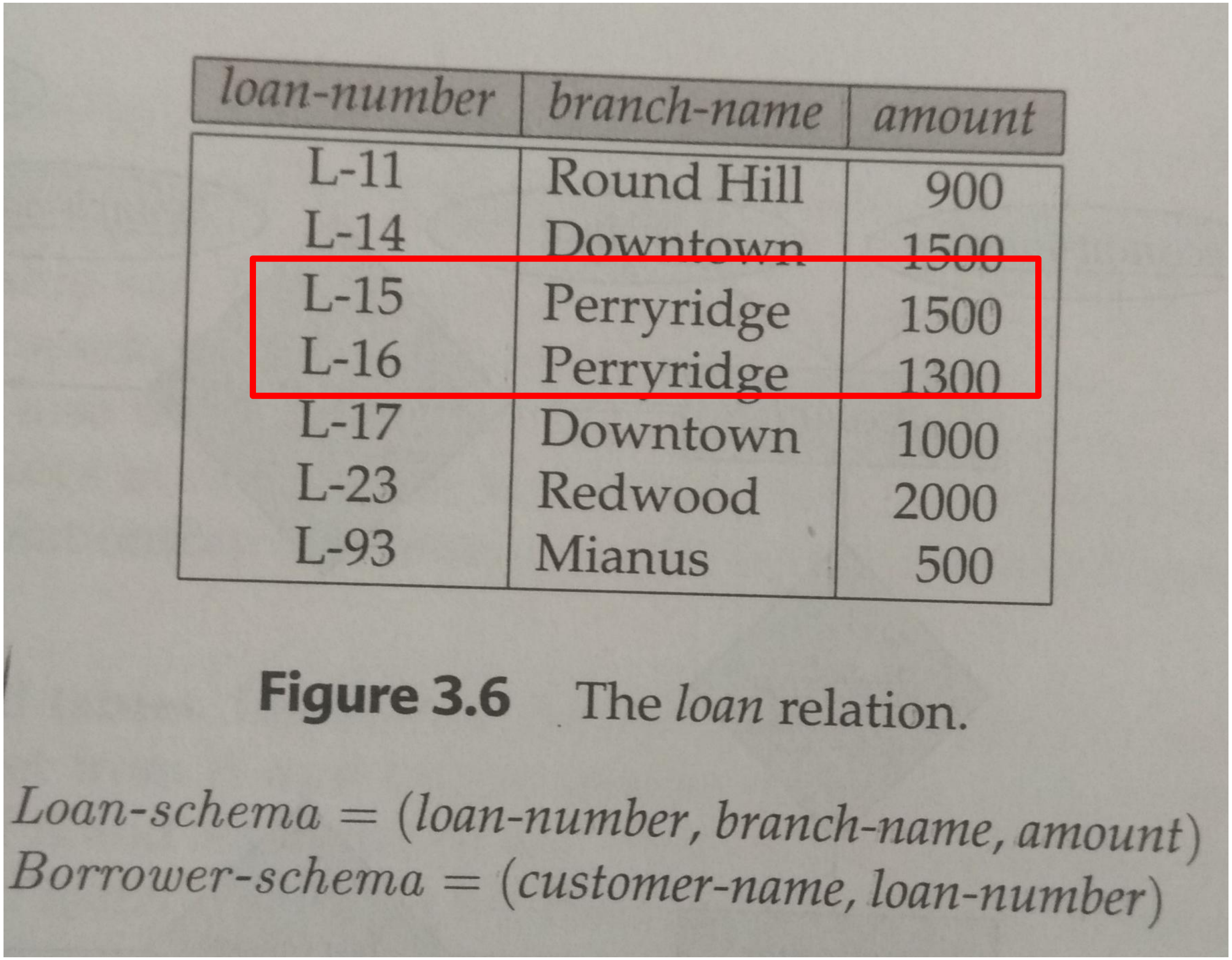
account-number	*branch-name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

Figure 3.1 The *account* relation.

The Select Operation

- It selects the tuples(rows) that satisfy the given predicate
- We use the lowerase Greek letter sigma(**σ**) to denote selection.
- The predicate appears as subscript to **σ.**
- The argument relation is in parenthesis after the σ.
- To select those tuples of the loan relation where the branch is "Perryridge", we write:

$$\sigma_{\text{branch-name} = \text{“Perryridge”}}\ (\text{loan})$$



loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Figure 3.6 The *loan* relation.

Loan-schema = (*loan-number*, *branch-name*, *amount*)
Borrower-schema = (*customer-name*, *loan-number*)

- We can find all tuples in which the amount is more than 1200 by writing:

$$\sigma_{amount>1200}\ (loan)$$

- We allow comparisons using =, ≠, <,>,≤,≥ in the selection predicate.
- We can also combine several predicates in to a larger predicate using the connectives and (▯), Or (▯) and not (¬).

- E.g. $\sigma_{amount>1200\ ▯\ branch=\text{“redwood”}}\ (loan)$

The Projection Operation

- It is a unary operation that returns its argument relation , with certain attributes left out. Duplicate rows are eliminated.
- Denoted by the Uppercase Greek letter pi (π).
- The argument relation follows in the parenthesis.
- We write the query to list all loan numbers and their amount as:

$$\pi_{\text{loan-number, amount}} (\text{loan})$$

Composition of Relational Operations

- To find those customers who live in “Harrison”, we write

$\pi_{\text{customer-name,}}(\sigma_{\text{customer-city=“Harrison”}}(\text{Customer}))$

The Union Operation

- Consider the query to find the name of all bank customers who have either an account or loan or both.
- Customer relation does not contain the information, since a customer does not need to have either an account or loan at the bank.
- To answer this query we need information from depositor relation and borrower relation.
- To find the names of all customers with a loan in the bank:

$$\pi_{\text{customer-name}}(\text{borrower})$$

- To find the names of all customers with an account:

$$\pi_{\text{customer-name}}(\text{depositor})$$

- To answer the query we need the **union** of these two sets.
- We need all customer names that appear in either or both of the two relations.
- We do this with the help of binary operation Union as follows:

$$\pi_{\text{customer-name}}(\text{borrower}) \cup \pi_{\text{customer-name}}(\text{depositor})$$

- For a union operation r U s to be valid we require two conditions to hold:

1) The relations r & s must be of the same arity. (i.e. they must have same number of attributes).

2)The domain of the i^{th} attribute of r and the i^{th} attribute of s must be the same for all i.

The Set Difference Operation

- The set difference operation , denoted by – allows us to find tuples that are in one relation but not in other relation.
- The expression r – s produces a relation containing those tuples that are in r but not in s.
- To find all customers of the bank who have an account but not a loan we can write:

$$\pi_{customer\text{-}name}(depositor) - \pi_{customer\text{-}name}(borrower)$$

➢ The relations r & s must be of the same arity. (i.e. they must have same number of attributes).

➢ The domain of the ith attribute of r and the ith attribute of s must be the same for all i.

The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (X) allows us to combine information from any two relations.
- We write the Cartesian product of the relations r1 and r2 as r1 X r2.
- Since the same attribute name may appear in both r1 and r2, we devise a naming schema to distinguish between these attributes.
- E.g. the relation schema for r = borrower X loan is (borrower.customer-name, borrower.loan-number, loan.loan-number, loan.branch-name, loan.amount)
- With this naming convention we can distinguish between borrower.loan-number, loan.loan-number.

customer-name	borrower. loan-number	loan. loan-number	branch-name	amount
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

Figure 3.14 Result of *borrower* × *loan*.

- Suppose we want to find the names of all customer who have a loan at perryridge branch , we need the information in both the relations loan and borrower.
- If we write :

$$\sigma_{\text{branch-name=“perriridge”}}\ (\text{borrower X loan})$$

The result of above would be:

customer-name	*borrower. loan-number*	*loan. loan-number*	*branch-name*	*amount*
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Hayes	L-15	L-15	Perryridge	1500
Hayes	L-15	L-16	Perryridge	1300
Jackson	L-14	L-15	Perryridge	1500
Jackson	L-14	L-16	Perryridge	1300
Jones	L-17	L-15	Perryridge	1500
Jones	L-17	L-16	Perryridge	1300
Smith	L-11	L-15	Perryridge	1500
Smith	L-11	L-16	Perryridge	1300
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300

Figure 3.15 Result of $\sigma_{branch\text{-}name = \text{“Perryridge”}}$ (*borrower* $\times$ *loan*).

- Since the cartecian product operation associates every tuple of loan with every tuple of borrower, we know that if a customer has a loan in the perryridge branch, then there is some tuple in borrower X loan that contains his name and borrwer.loan-number=loan.loan-number.
- So if we write:

$$\sigma_{\text{borrower.loan-number = loan.loan-number}} (\sigma_{\text{branch-name=“perriridge”}} (\text{borrower X loan}))$$

We get only those tuples of borrower X loan that pertain to customers who have a loan at the perriridge branch.

- Finally since we want only the customer-name, we do a projection:

$$\pi_{\text{customer-name}} (\sigma_{\text{borrower.loan-number = loan.loan-number}} (\sigma_{\text{branch-name=“perriridge”}} (\text{borrower X loan})))$$

- The result of this expression will be:

The Rename Operation

- Unlike relations in database, the results of relational algebra expressions do not have a name that we can refer.
- It is useful to be able to give them names; the rename operator, denoted by lowercase Greek letter rho(ρ) lets us do this.
- Given a relational algebra expression E, the expression

$$\rho_x(E)$$

returns the result of expression E under the name x.

- We can apply a rename operation to a relation r to get the same relation under a new name.
- Assume that a relational algebra expression E has arity n. Then the expression:

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

- Returns the result of expression E under the name x and with the attributes renamed to A1, A2, A3, ...An.

- Consider the query, “ Find the names of all customers who live on the same street and in the same city as smith.”
- We can obtain smith’s street and city by:

$$\pi_{\text{customer-street, customer-city}} (\sigma_{\text{customer-name=“smith”}}(\text{customer}))$$

- However to find the other customers with this street and city , we must refer the customer relation second time.
- In following query we use the rename operation on the preceding expression to give its result the name smith-addr and to rename its attributes to street and city, instead of customer-street and customer-city.

$\pi_{\text{customer.customer-name}}$

$(\sigma_{\text{customer.customer-street=smith-addr.street}} \quad \square$

$\text{customer.customer-city=smith-addr.city}$

$(\text{custmer X } \rho_{\textit{smith-addr(street,city)}}$

$(\pi_{\text{customer-street, customer-city}}$

$(\sigma_{\text{customer-name=“smith”}}(\text{customer})))))$

The Natural-Join operation

- The join operation combines two relations to form a new relation.
- The meaning of a join operation in terms of Cartesian product and selection operation is given below:
  - The join operation forms a Cartesian product of participating relations then performs a selection operation using equality of joining attributes.
  - The join operation allows the processing of relationships existing between the two relations.

- Example:
- Given the two relations EMPLOYEE and SALARY, we can join the tuples in the EMPLOYEE relation with those in SALARY such that the value of the attribute 'id' in EMPLOYEE is same as in SALARY.

EMPLOYEE	
ID	NAME
101	MOHAN
103	RITU
105	PREM

SALARY	
ID	SALARY
101	30000
103	35000
105	48000

Join of EMPLOYEE and SALARY			
EMPLOYEE.ID	NAME	SALARY.ID	SALARY
101	MOHAN	101	30000
103	RITU	103	35000
105	PREM	105	48000

- The general form of JOIN operation on two relations R(A1,A2,A3……,An and S(B1,B2,B3……….,Bm) is :

$$R \bowtie S$$

- The result of join is a relation T with n+m attributes.
- The result of natural join can be projected on required attributes.
- For finding the salary of employees by name, we can project the result of the natural join operation on the attributes name and salary.

$$\pi_{\text{name, salary}}(\text{EMPLOYEE} \bowtie \text{SALARY})$$

- The natural join operation can also be specified on multiple tables as:

$$(R \bowtie S) \bowtie T$$

The Division operation

- The division operation denoted by ÷, is suited to queries that include the phrase “ for all”.
- Suppose, we want to find all customers who have an account at all the branches located in brooklyn.
- We can obtain all the branches located in brooklyn by the expression:

$$r1 = \pi_{\text{branch-name}}(\sigma_{\text{branch-city=“brooklyn”}}(\text{branch}))$$

The result relation for this expression is :

branch-name
Brighton
Downtown

- We can find all (customer-name,branch-name) pairs for which the customer has an account at a branch by:

$$r2 = \pi_{\text{customer-name, branch-name}}(\text{ depositor } \bowtie \text{ account})$$

The result of above expression is:

Customer-name	**Branch-name**
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Londsay	Redwood
Smith	Mianus
Turner	Round Hill

- Now we need to find customers who appear in r2, with every branch-name in r1.
- The operation that provides exactly those customers is divide operation.
- The query will be:

$$\pi_{\text{customer-name, branch-name}}(\text{ depositor} \bowtie \text{account})$$

$$\div \; \pi_{\text{branch-name}}(\sigma_{\text{branch-city=“brooklyn”}}(\text{branch}))$$

- The result will be a relation customer-name with tuple “ Johnson”.

The Assignment operation

- It is convenient at times to write a relational algebra expression by assigning parts of it to temporary variables.
- The assignment operation denoted by $\leftarrow$, works like assignment in programming language.
- This relational variable can be used as a relation in subsequent expressions.

$$temp1 \leftarrow P \ \square\ Q$$

$$temp2 \leftarrow \pi_{A,B}\ (temp1)$$

Generalized Projection Operation

- The generalized projection operation extends the projection operation by allowing arithmetic functions to be used in projection list .
- Its form is:

$$\pi_{F1,F2,\ldots\ldots,Fn}(E)$$

- Where E is any relational algebra expression, and F1,F2.....are any arithmetic expression.
- Suppose we have a relation credit-info as:

Customer-name	**Limit**	**Credit-balance**
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

- Suppose we want to find how much more each person can spend, we can write following expression

$$\pi_{\text{customer-name, limit - credit-balance}}(\text{credit-info})$$

- The attribute resulting from ‘limit - credit-balance’ does not have a name.
- We can apply the rename operation to the result of generalized projection as:

$$\pi_{\text{customer-name, (limit - credit-balance) as credit-available}}(\text{credit-info})$$

Customer-name	**Credit-available**
Curry	250
Hayes	0
Jones	5300
Smith	1600

Aggregate Functions

- The symbol (calligraphic g) is used for aggregation.

$$\varsigma_{\text{sum(salary)}}(\text{EMPLOYEE})$$

- The above relational expression finds the sum of all the salaries of all employees .
- Some of the aggregate operations are:

1. Count ()
2. Sum ()
3. Average ()
4. Max ()
5. Min ()

The Outer-Join operation

- When we join two relations R and S, it is possible that, a tuple in one relation may not have a matching tuple in other relation.
- Outer join is used when we want a tuple from one of the relations to appear in the result when there is no matching value in other relation.
- There are three types of outer joins:

  1) Left Outer Join ⟕

  2) Right Outer Join ⟖

  3) Full Outer Join ⟗

- In Left Outer Join R⟕ S:

  Tuples from R may not have matching tuples in S, but they are still included in the result. Missing values in S are set to null

- In Right Outer Join R ⟖ S:

  Tuples from R may not have matching tuples in S, but they are still included in the result. Missing values in S are set to null

- In Full Outer Join R ⟗ S:

  Tuples from R and S are always included in the result. Missing values in R and S are set to null.

Employee

Id	Name
101	Mohan
102	Sohan
103	Ritu
105	Prem

Salary

Id	Name
101	30000
103	35000
104	32000
105	28000

Employee ⟕ Salary

Id	Name	Salary
101	Mohan	30000
102	Sohan	Null
103	Ritu	35000
105	Prem	28000

Employee ⟖ Salary

Id	Name	Salary
101	Mohan	30000
103	Ritu	35000
104	Null	32000
105	Prem	28000

Employee ⟗ Salary

Id	Name	Salary
101	Mohan	30000
102	Sohan	Null
103	Ritu	35000
105	Prem	28000
104	Null	32000

SQL (Structured Query Language)

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

Rules:

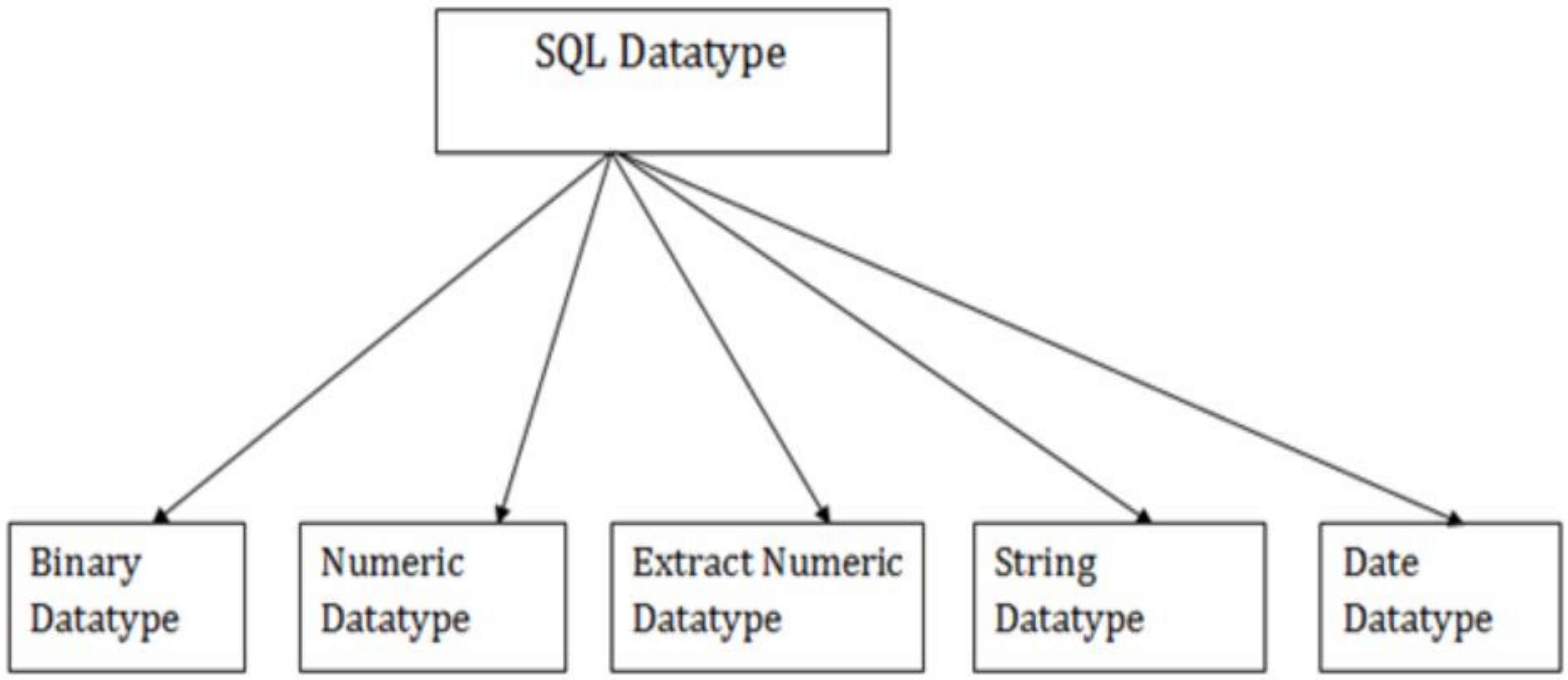
SQL follows the following rules:

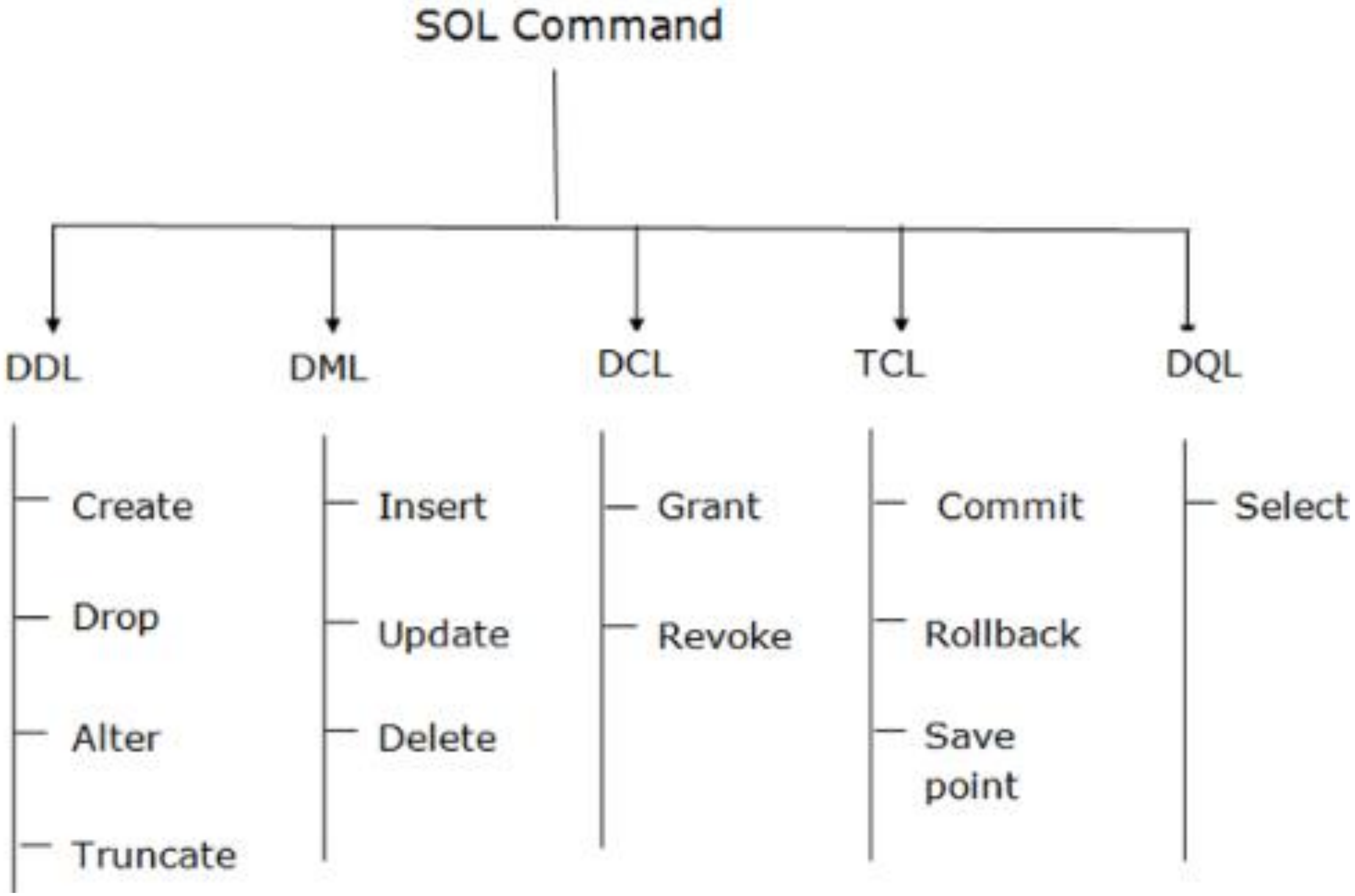
- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

Characteristics of SQL

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Datatype of SQL:





- **DCL: Data Control Language**
- **DQL: Data Query Language**
- **TCL: Transaction Control Language**

SQL Table

- Operation on Table

1) Create table
2) Drop table
3) Delete table
4) Rename table

Syntax

```
create table "table_name"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
-------------------------------------------------------
CREATE TABLE EMPLOYEE (
EMP_ID         INT                       NOT NULL,
EMP_NAME       VARCHAR (25)              NOT NULL,
PHONE_NO       INT                       NOT NULL,
ADDRESS        CHAR (30),
);
```

SQL SELECT Statement

- **Syntax**

```
SELECT column1, column2, ...
FROM table_name;
```

or

```
SELECT  *  FROM table_name;
```

e.g.

```
SELECT EMP_ID FROM EMPLOYEE;
SELECT EMP_NAME, SALARY FROM EMPLOYEE;
SELECT * FROM EMPLOYEE;
```