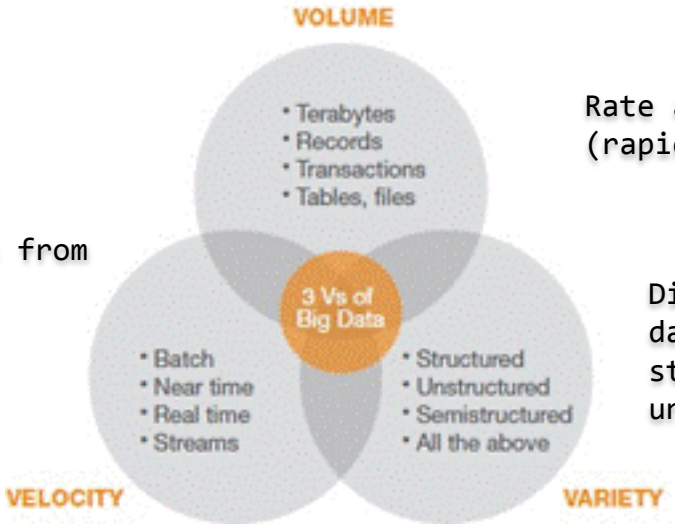
Databases
Relational Model

COMP 1531, 17s2
Aarthi Natarajan
Week 8

What is data?

- **Data** – facts that can be recorded and have implicit meaning (Elmasri & Navathe)
- Today data is being generated at an exponential rate
  - Financial market data, posts to social media sites, growing logs of web transactions, computation physics…BIG Data



Rate at which data is generated (rapid, exponential growth)

Rate at which data flows from different sources

Diversity in data types – structured data (financial market data), semi-structured (XML, web logs, email), unstructured (audio, video streams)

Why do we need a database ?

- Data by itself is not very useful.
- Give a context to data to transform data into information e.g., the numbers 45,55,67 do not mean much, but given a context such as these are the marks of students in COMP 1531, this is now information

  DATA -> INFORMATION -> DECISION

- This data needs to be:
  - Stored
  - Manipulated
  - Shared
  - Transmitted
- Red text handled by databases; green by networks.

- **Databases are everywhere…**
  - The Internet uses databases extensively
    - Google, Ebay, Amazon, iTunes Shop
    - Library catalogues, Train time tables, Airline bookings
    - Bank accounts, credit card, debit card
    - Medical records (Medicare), Tax Office
    - Facebook, Twitter, …
  - Every time you use a loyalty card, you're inputting information about your buying habits into the database of the company you are buying from
- **Challenges in building effective databases**
  - efficiency, security, scalability, maintainability, availability, integration, new media types (e.g. music), ...

What is a database ?

- A **database** is a logically coherent collection of related data
- A **database management system (DBMS)** is an application or collection of programs that allows users to:
  - create and maintain a database (DDL)
  - defining queries that causes data to be retrieved from the database
  - Perform transactions that cause some data to be written or deleted from the database (DML)
- A database and DBMS are collectively referred to as a **database system**

A DBMS provides:

- Persistence
- Concurrency
- Integrity
- Security

Examples of DBMS

- Open Source
  - SQLite
  - PostgreSQL
  - MySQL
- Commerical
  - Oracle
  - DB2 (IBM)
  - MS SQL Server

What is a relational DBMS?

A **Relational Database Management System** (RDBMS) is a special type of DBMS that:

- Stores data as *tuples* or *records* in *tables*
- Allows the user to create *relationships* between tables

Defining more database terminology

- A data model describes how the data is structured in the database
- A database schema adheres to a data model and provides a logical view of the database, defining how the data is organised and the relationships between them and is typically set up at the beginning
- A database schema instance is the state of the database at a particular instance of time

Data Models

- There are several types of data models
  - Relational model
    - a data structure where data is stored as a set of records known as tables
    - a table consists of rows of information (also called a tuple)
    - each row contains fields known as columns

StudentId	FirstName	LastName
213899	Joe	Bloggs
321456	Sam	Hunt
456789	John	Smith

  - Document model
    - data is stored in a hierarchical fashion e.g., XML
  - Object-oriented model
    - a data structure where data is stored as a collection of objects
  - Object-Relational model
    - a hybrid model that combines the relational and the object-oriented database models

Goals of this course

- Understand Relational Model
- Understand ER model and ER-to-relational mapping
- Database Application Development
  - SQLite: sqlite3 (SQL shell)
  - SQL (Structured Query Language, a standard that allows you to access different DBMS
  - Programming language access to databases (using Python, ORM)

Relational Data Model

The relational data model describes the world as a collection of inter-connected relations (or tables)

Goal of relational model:

- a simple, general data modelling formalism
- maps easily to file structures (i.e. implementable)

Relational model has two styles of terminology:

mathematical	relation	tuple	attribute
data-oriented	table	record (row)	field (column)

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

Relational Data Model

The relational model has one structuring mechanism ...

- a relation corresponds to a mathematical "relation"
- a relation can also be viewed as a "table"

Each relation (table) (denoted *R,S,T,...*) has:

- a name (unique within a given database)
- a set of attributes (or column headings)

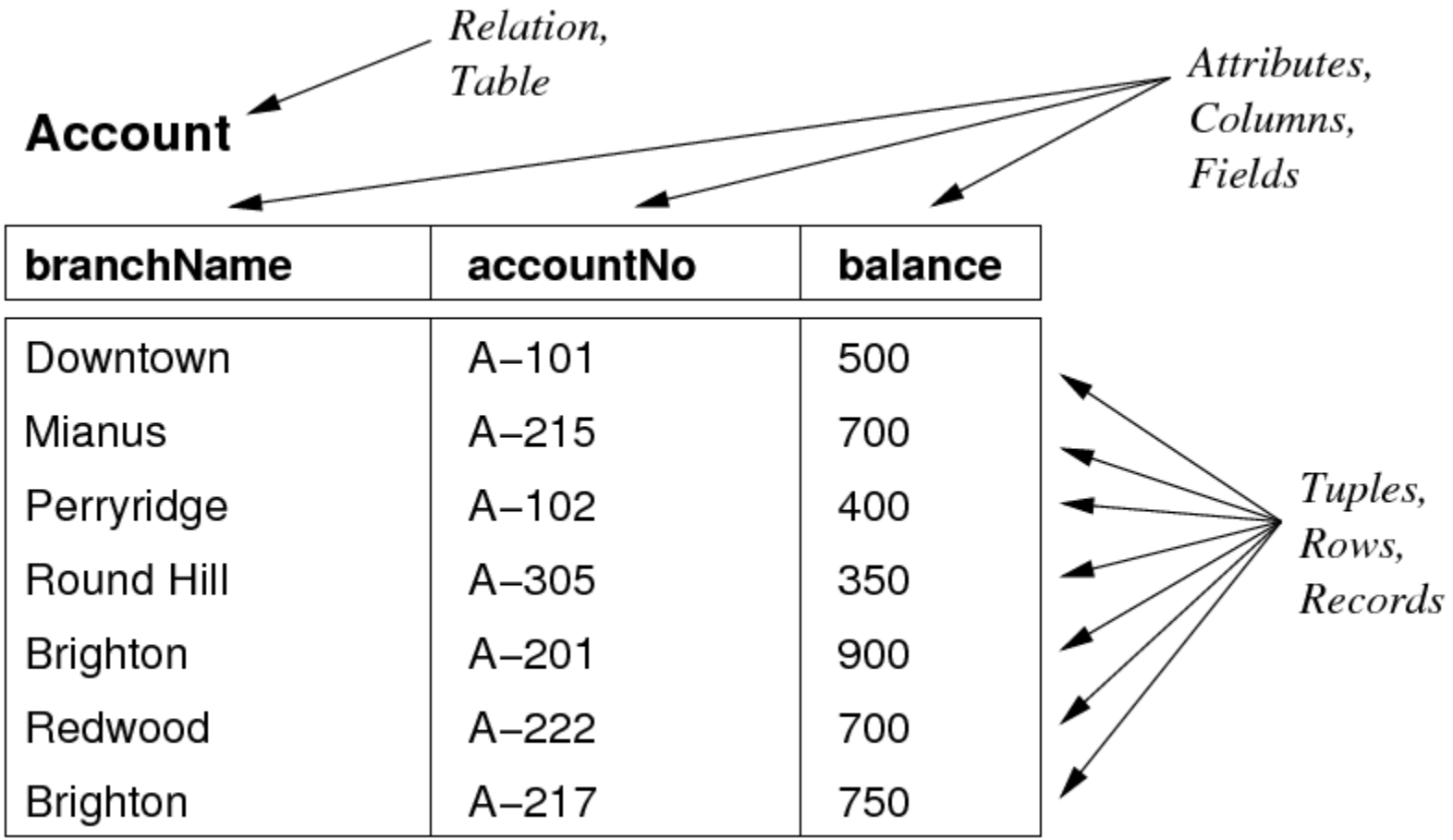
Each attribute (denoted *A,B,...* or $a_1,a_2,...$) has:

- a name (unique within a given relation)
- an associated domain (set of allowed values)

DB definitions also make extensive use of constraints

Relational Data Model

Example of a relation (table): Bank Account



branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Brighton	A–201	900
Redwood	A–222	700
Brighton	A–217	750

Relational Data Model

A ***tuple (row)*** is a **set** of **values (attribute or column values)**

Attribute values:

- Are atomic (no composite or multi-valued attributes).
- Belong to a domain; a domain has a **name**, **data type** and **format**
- A distinguished value **NULL** belongs to all domains.
- NULL has several interpretations: none, don't know, irrelevant

Column Header	**Domain Name**	**Domain Data Type, Format and Constrain**
phone_number	local_phone_numbers - (set of phone numbers valid in australia)	character string of the format *(dd) ddddddd*, where each *d* is a numeric (decimal) digit and the first two digits form a valid telephone area code.
age	employee_age (set of possible ages for employees in the company)	An integer value between 15 and 80

Relational Data Model

- A ***relation(table)*** is a **set** of ***tuples***.
- Since a relation is a set, there is **no ordering** on rows.
- Normally, **we define a standard ordering** on components of a tuple.
- The following are different presentations of the same relation:

branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Redwood	A–222	700

accountNo	branchName	balance
A–305	Round Hill	350
A–222	Redwood	700
A–215	Mianus	700
A–102	Perryridge	400
A–101	Downtown	500

- Each relation generally has a **primary key** (subset of attributes, unique over relation)

Relational Data Model

A ***database*** is a **set** of ***relations(tables)***

Account

branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Brighton	A–201	900
Redwood	A–222	700

Branch

branchName	address	assets
Downtown	Brooklyn	9000000
Redwood	Palo Alto	2100000
Perryridge	Horseneck	1700000
Mianus	Horseneck	400000
Round Hill	Horseneck	8000000
North Town	Rye	3700000
Brighton	Brooklyn	7100000

Customer

name	address	customerNo	homeBranch
Smith	Rye	1234567	Mianus
Jones	Palo Alto	9876543	Redwood
Smith	Brooklyn	1313131	Downtown
Curry	Rye	1111111	Mianus

Depositor

account	customer
A–101	1313131
A–215	1111111
A–102	1313131
A–305	1234567
A–201	9876543
A–222	1111111
A–102	1234567

Expressing Relational Data Model Mathematically

Given a relation *R* which has:

- *n* attributes $a_1, a_2, \ldots a_n$
- with corresponding domains $D_1, D_2, \ldots D_n$

We define:

- Relation schema of *R* as: $\boldsymbol{R}(a_1{:}D_1, a_2{:}D_2, \ldots a_n{:}D_n)$
- Tuple of *R* as: an element of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. list of values)
- Instance of *R* as: subset of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. set of tuples)
- Database schema : a collection of relation schemas.
- Database (instance) : a collection of relation instances.

Relation Schema

We often use *R* as a synonym for the relations schema: $R(a_1, a_1, \ldots a_n)$

e.g., the *Accounts* schema which has *3* attributes *branchName, accountNo, balance* with corresponding domains *string, string,int* can be defined as:

Account (branchName:string, accountNo:string, balance: int) OR

Account (branchName, accountNo, balance)

and a tuple of R (row of R)can be specified as:

(Downtown, A-101, 500)

and an account instance (set of tuples or rows)

* **No duplicates**

{ (Downtown, A-101, 500), (Mianus, A-215, 700),
(Perryridge, A-102, 400), (Round Hill, A-305, 350),
(Brighton, A-201, 900), (Redwood, A-222, 700)}

Relation Schema

- The **degree** (or **arity**) of a relation is the number of attributes *n* of its relation schema, so a relation schema *R* of degree *n* is denoted by $R(a_1, a_2, \ldots a_n)$.

  e.g. A relation of degree seven, which stores information about university students, would contain seven attributes describing each student. as follows:

  Student (name, ssn, home_phone, address, office_phone, age, gpa)

OR as

Student (name: string, ssn: string, home_phone: string, address: string, office_phone: string, age: integer, gpa: int)

Constraints

Relations are used to represent real-world *entities* and *relationships* between these *entities*

To represent real-world problems, need to describe

- what values are/are not allowed
- what combinations of values are/are not allowed

Constraints are logical statements that do this:

- Domain constraint
- Key constraint
- Entity constraint
- Integrity constraint
- Referential integrity

Constraints

Domain constraints example:

- Employee.age attribute is typically defined as integer
- better modelled by adding extra constraint (15<age<66)

Note: NULL satisfies all domain constraints (except (NOT NULL))

Key constraints example:

- Student(id, ...) is guaranteed unique
- Class(..., day, time, location,...) is unique

Entity integrity example:

- Class(...,Mon,2pm,Lyre,...) is well-defined
- Class(...,**NULL**,2pm,Lyre,...) is not well-defined

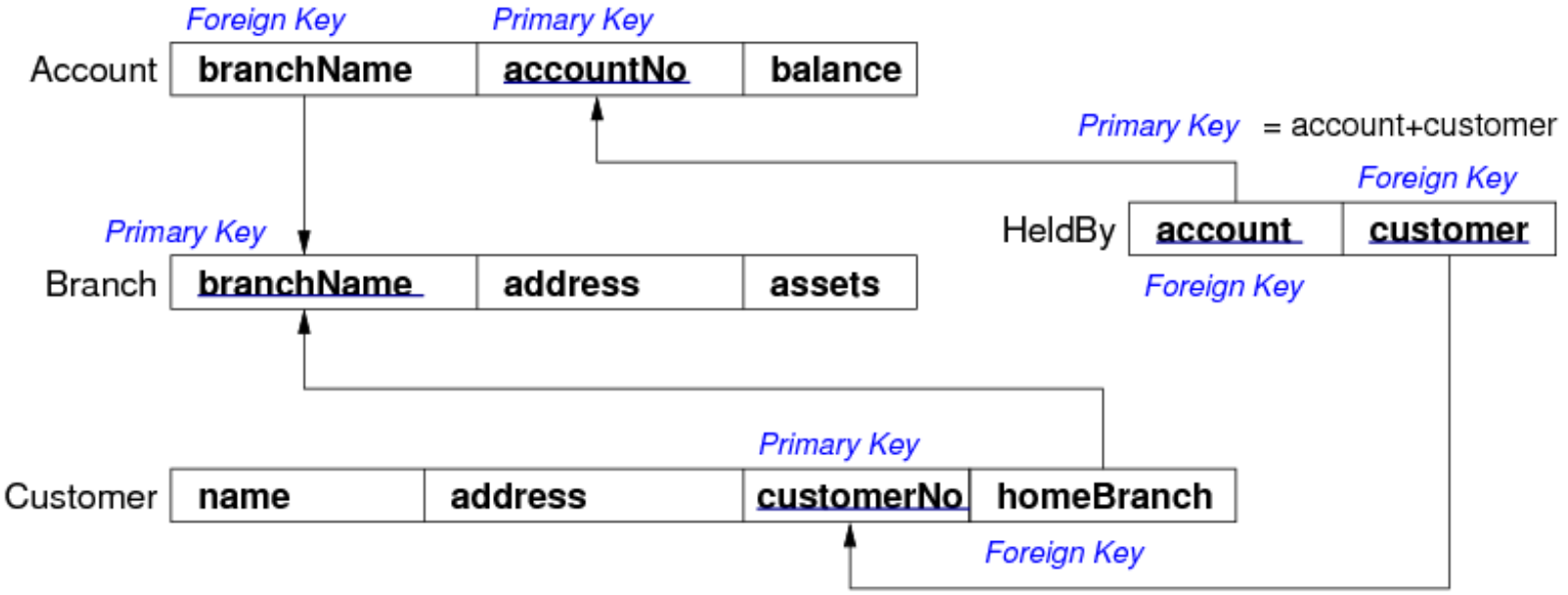
Referential Integrity Constraints

- Related to the notion of a foreign key
- A set of attributes *F* in relation R_1 is a foreign key if:
  - the attributes in *F* correspond to the attributes in the primary key of another relation R_2
  - the value for *F* in each tuple of R_1
    - either occurs as a primary key in R_2
    - or is entirely NULL
- Foreign keys are critical in relational DBs;
  - they provide ...the "glue" that links individual relations (tables)
  - the way to assemble query answers from multiple tables

Foreign Key Example

Primary key of the parent table linked to a foreign key in the child table

- the Account relation (child) needs to take note of the branch where each account is held
- implemented by storing the associated branch in each Account tuple
- it would not make sense to store a branchName that did not refer to one of the existing branches in the Branch relation (parent table)
- The notion that the branchName (in Account) ***must refer*** to a valid branchName (in Branch) is a **referential integrity constraint**.



Putting all together…

A relational database schema is viewed as:

- a set of relation schemas $\{ R_1, R_2, \ldots R_n \}$, and
- a set of integrity constraints

A relational database instance is

- a set of relation instances $\{ r_1(R_1), r_2(R_2), \ldots r_n(R_n) \}$
- where all of the integrity constraints are satisfied

One of the important functions of a relational DBMS:

- ensure that all data in the database satisfies constraints

Creating a database

- Choose a data model for the database
- Set up the structure of the database by defining a database schema for the database (e.g., for a relational data model, define the tables, rows and columns or field names and types of fields, constraints and relationships)
- Create the initial state of the data by loading data into the database
- After this, typically schema doesn't change much, but data changes rapidly as new data is loaded or existing data is updated

Changing Relation Schema and Relations

In making changes to relations, it is ...

- easy to add new tuples (rows) (relation insert)
- east to change attribute values in tuples (relation update)
- but, difficult to add new attributes (columns) (schema update)

The reasons:

- relation update ⇒ insertion of one new tuple into a set (in file terms: writing one record to the end of a data file)
- schema update ⇒ insertion of new data into every tuple (in file terms: re-writing the entire file to modify each record)

DBMS Terminology

To remember:

- DBMS-level ... database names must be unique
- database-level ... schema names must be unique
- schema-level ... table names must be unique
- table-level ... attribute names must be unique

Sometimes it is convenient to use same name in several tables

We distinguish which attribute we mean using qualified names

e.g. **Account.branchName** vs **Branch.branchName**

Relational Model vs DBMS

The relational model is a mathematical theory

- giving a representation for data structures
- with constraints on relations/tuples
- and an *algebra* for manipulating relations/tuples (union, intersect…)

Relational DBMSs

- provide an implementation of the relational model

Features of RDBMS

- Support large-scale data-intensive applications
- Provide efficient storage and retrieval (disk/memory management)
- Support multiple simultaneous users (privilege, protection)
- Support multiple simultaneous operations (transactions, concurrency)
- Maintain reliable access to the stored data (backup, recovery)
- Use SQL as language for:
  - data definition (creating, deleting relations i.e. tables)
  - relation query (selecting tuples)
  - relation update (changing relations)

RDBMS Operations

RDBMSs typically provide at least the following:

- create/remove a database or a schema
- create/remove/alter tables within a schema
- insert/delete/update tuples within a table
- queries on data, define named queries (views)
- transactional behaviour (ACID)

Most also provide mechanisms for:

- creating/managing users of the database
- defining/storing procedural code to manipulate data
- implementing complex constraints (triggers)
- defining new data types and operators (less common)

Describing Relational Schemas

- SQL (Structured Query Language) provides the formalism to express relational schemas

- SQL provides a Data Definition Language (DDL) for creating relations

```
CREATE TABLE TableName (
attrName1 domain1 constraints1 ,
attrName2 domain2 constraints2 , ...
PRIMARY KEY (attri,attrj,...)
FOREIGN KEY (attrx,attry,...)
REFERENCES OtherTable (attrm,attrn,...)
);
```

SQL Syntax in a NutShell

- Comments: everything after **--** *is a comment*
- Identifiers: alphanumeric (a la C), but also "An Identifier"
- Reserved words: many e.g. **CREATE**, **SELECT**, **TABLE**, ...
- Strings: e.g. 'a string', 'don''t ask', but no '\n'
- Numbers: like C, e.g. 1, -5, 3.14159, ...
- Identifiers and reserved words are case-insensitive:

  TableName = tablename = TaBLeNamE != "TableName"

- Types: integer, float, char(*n*), varchar(*n*), date, ...
- Operators: =, <>, <, <=, >, >=, AND, OR, NOT, ...

SQL Syntax in a NutShell

Defining tables:

CREATE TABLE *Name* (*Attributes*, *Constraints*)

Defining attributes:

Name Domain [*Constraint*] (constraint is optional)

Defining keys:

PRIMARY KEY (*AttrNames*)

FOREIGN KEY (*AttrNames*)
REFERENCES *Table* (*AttrNames*)

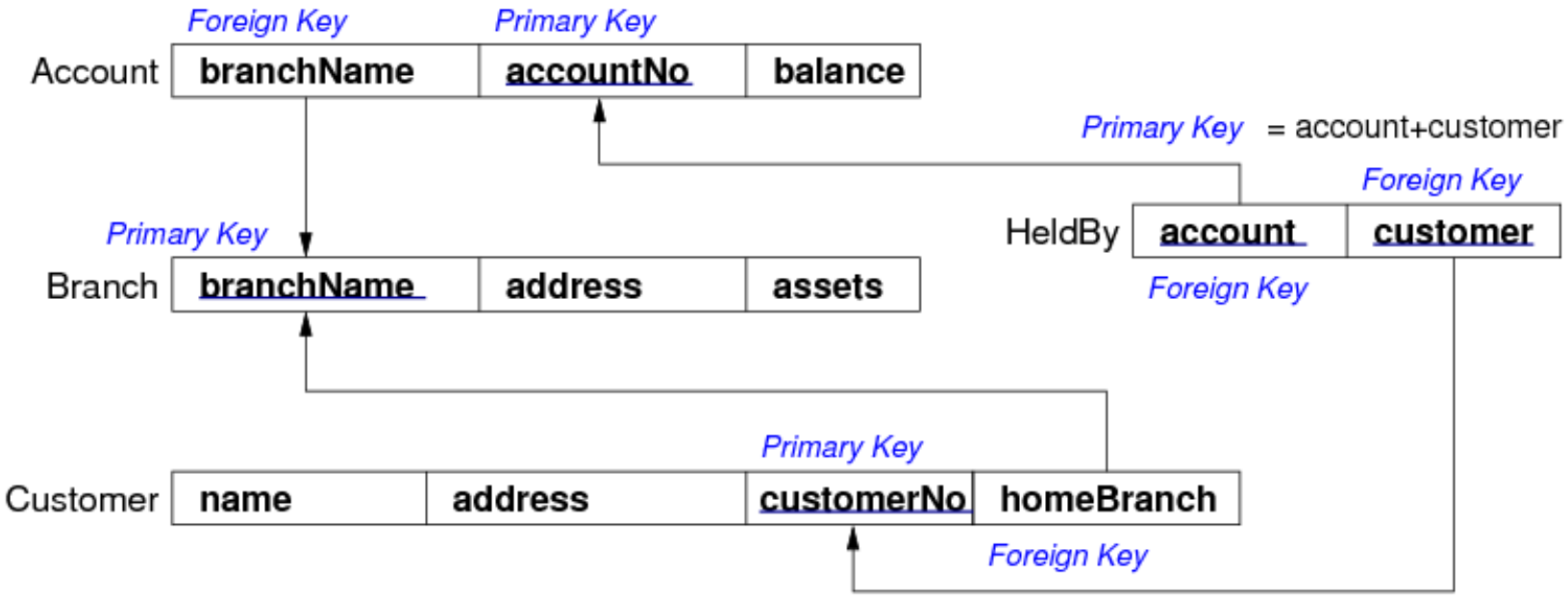
Defining constraints:

[CONSTRAINT *Name* **] CHECK** (*ExpressionOnAttributes*)

Exercise: Simple Relational Schema (i)

Express the following in SQL DDL:

Assume only two domains: string and integer



Exercise: Simple Relational Schema (Ii)

Augment the previous schema to enforce:

- no accounts can be overdrawn
- customer numbers are seven-digit integers
- account numbers look like **A-101, B-306** etc.

Assume that all standard SQL types (domains) are available

Add new domain to define account numbers and use it throughout

SQLite

- A very small and light-weight RDBMS
- An open source
- A complete database is simply stored in a single database file
- Available on UNIX and Windows
- SQLite Transactions are fully ACID-complaint
- Supports most of query language features in SQL92 (SQL2) standard

Managing Databases in SQLite3

To create a new SQLite database:

```
$sqlite3 database.db
```

After successfully creating a new database, then a **sqlite>** prompt will be provided, e.g.,

```
$sqlite3 testDB.db
SQLite version 3.20.1 2017-08-24 16:21:36
Enter ".help" for usage hints.
sqlite> .database
main: /home/sqlite/testDB.db
sqlite>
```

Use SQLite **.quit** to come out of the sqlite prompt

```
sqlite>.quit
```

Use SQLite **.dump** to export the complete database into a ASCII text file DB.sql

```
sqlite>testDB.db .dump > testDB.sql
```

To delete the database simply drop the testDB.db file from filesystem

Managing Tables in SQLite3

To create a new table in SQLite, use the CREATE TABLE SQL statement

```
CREATE TABLE database_name.table_name (
  column1 datatype PRIMARY KEY (one or more columns)
  column2 datatype,
  ...
  columnN datatype
);
```

Example:

```
sqlite>CREATE TABLE testDB.employee (
   emp_id INT PRIMARY KEY,
   department INT NOT NULL,
   salary REAL NOT NULL CHECK (salary > 10000));
```

Use **.tables** to list all the tables in an attached database and .schema to get complete information about a table

To drop a table in SQLite, use the DROP TABLE SQL statement

```
sqlite>.tables
Employee
sqlite> DROP TABLE database_name.table_name;
```

SQLite can be integrated with Python using sqlite3 module APIs

Basic Query Data in SQLite Database from Python

1. Establish a connection to the SQLite database by creating a Connection object passing in the name of the database to connect to

```
connection = sqlite3.connect('book.db')
```

2. Create a Cursor object using the cursor method of the Connection object.

```
cursorObj = connection.cursor()
```

3. Build the query

```
query = "SELECT title, author, year, genre  from BOOK
where title = ?"
```

4. Execute the query and fetch the results

```
rows = cursorObj.execute(query)
```

5. Finally, loop the cursor and process each row individually

```
for row in rows:
    print(row)
```