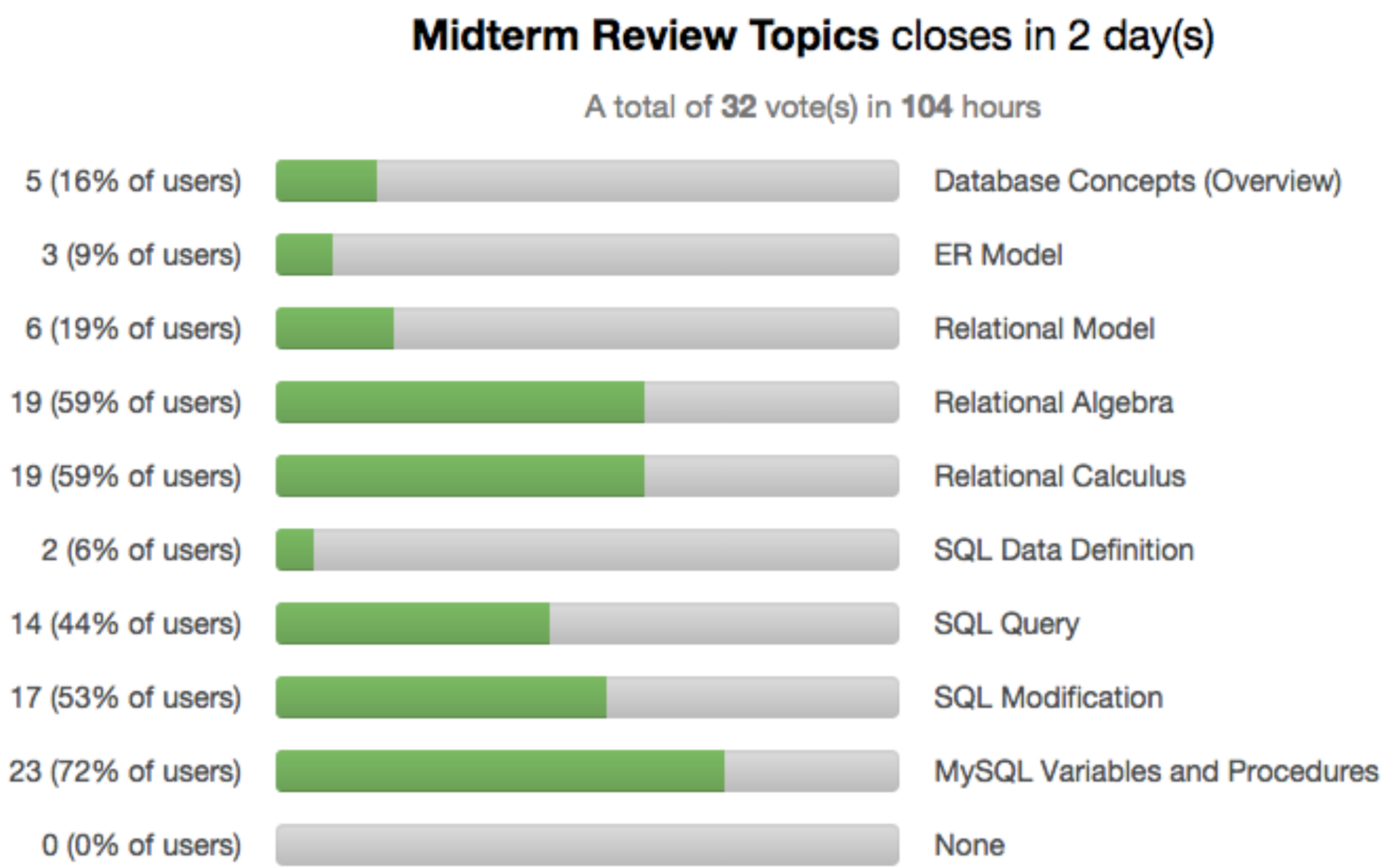
Midterm Review

CS 377: Database Systems

Piazza Poll Results



Database Concepts

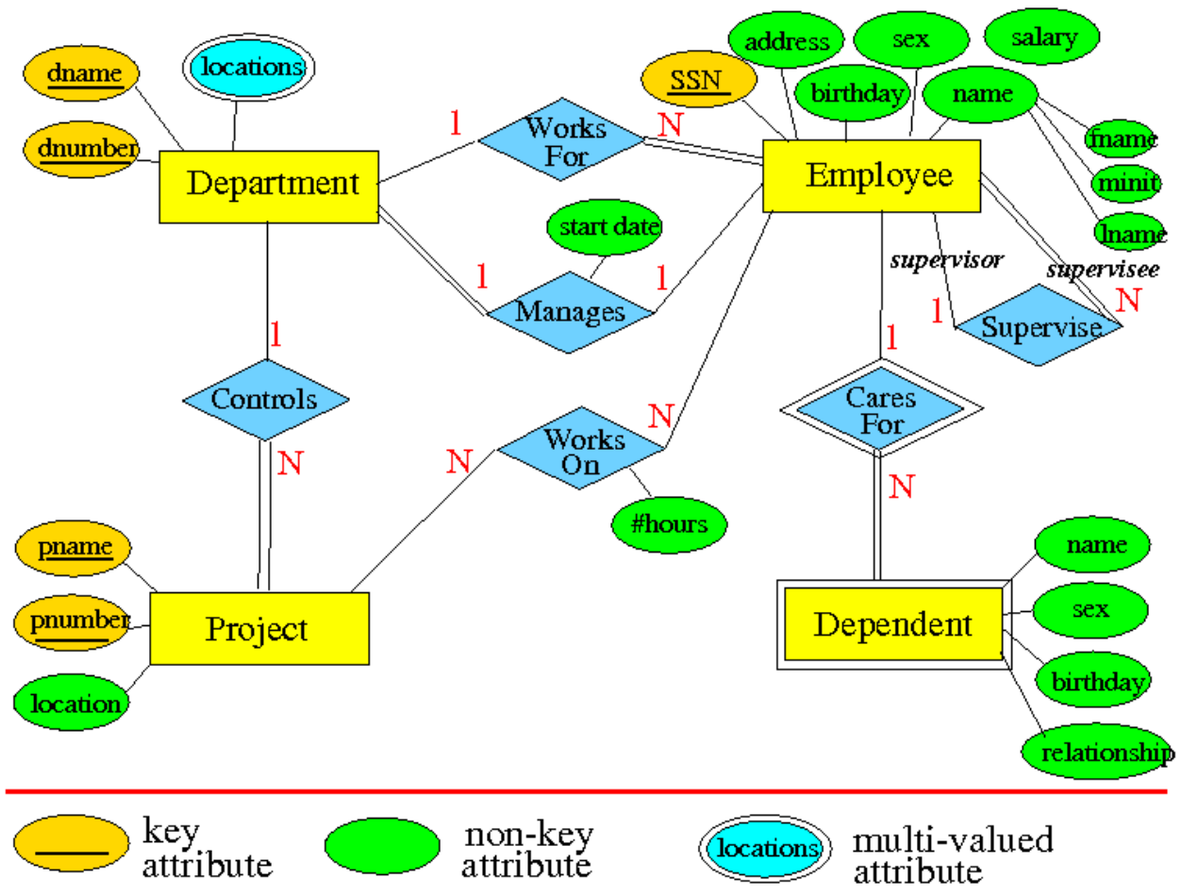
- Data model categories: high-level or conceptual data models, low-level or physical data models, and representational or implementation data models

- Physical data and logical data independence
    - How metadata fits into the picture

- Three schema architecture

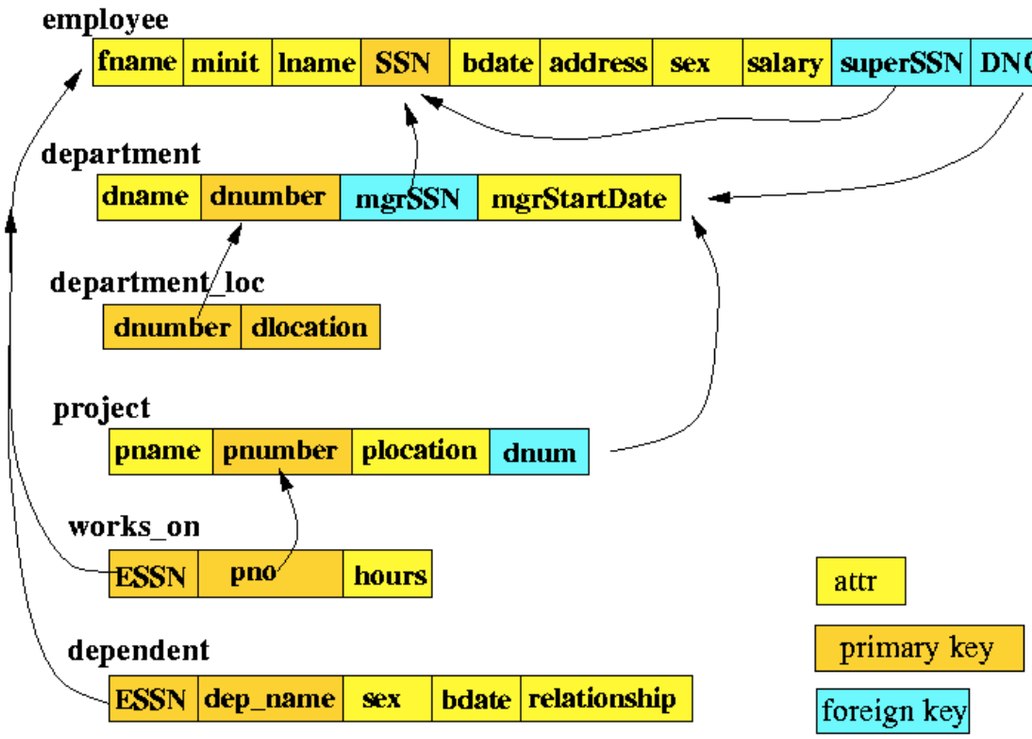
Entity Relationship (ER) Model

- Entity
  - Attributes
  - Weak Entity
- Relationship
  - Degree
  - Cardinality ratio constraint
  - Participation constraint



Relation Model

- Relation, attributes
- Schema vs instance
- Relational model constraints
  - Domain constraint
  - Key constraint
  - Referential integrity constraint



ER to Relational Model

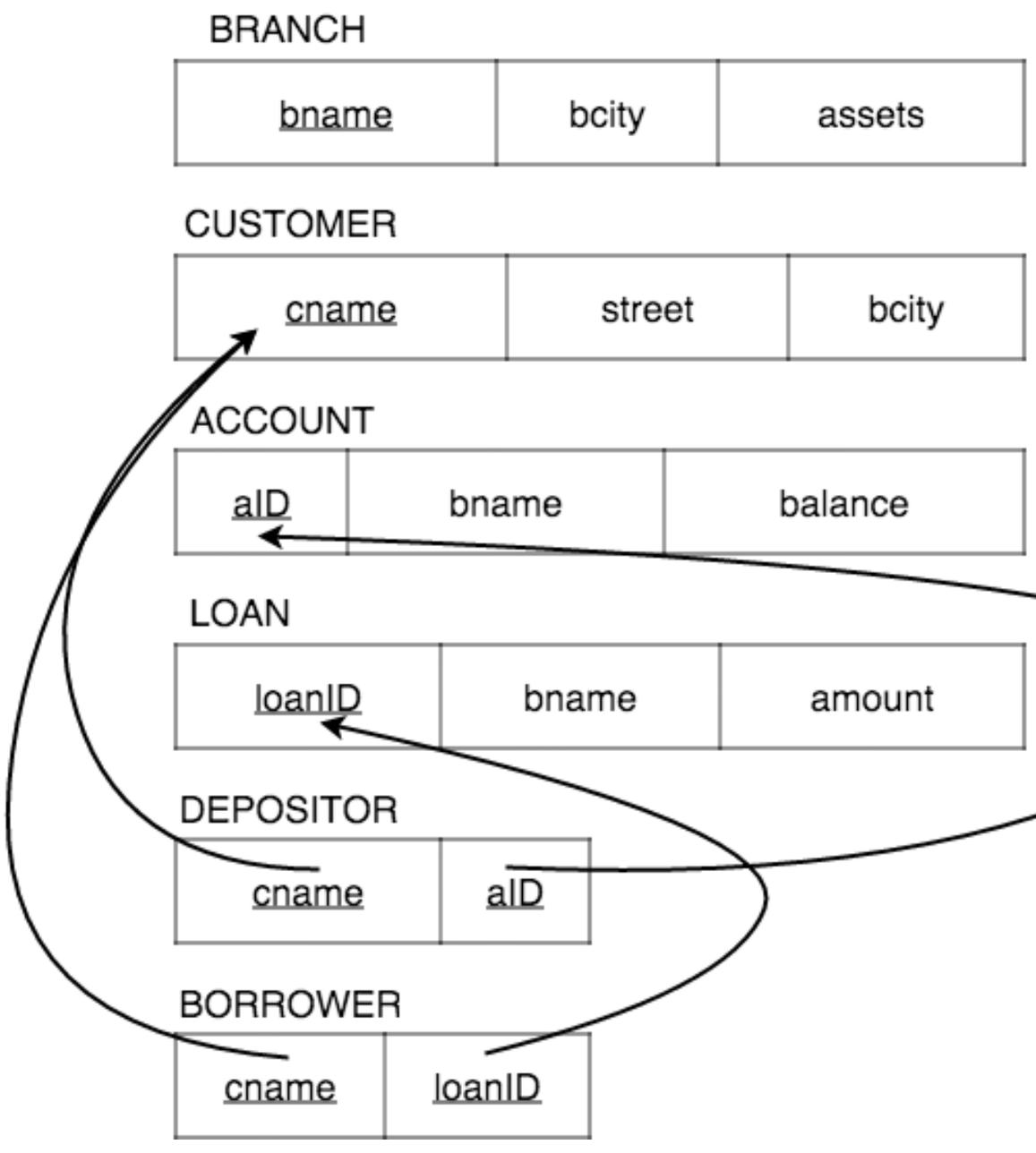
Entity set and relationships and convert them to relation

ER Model	Relational model
Entity type	Entity relation
1:1 or 1:N relationship	Expand (or create R relation)
M:N relationship	Create R relation with two foreign keys
n-ary relationship type	Create R relation with n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Key attribute	Primary (or secondary) key

Relational Algebra

Operation	Notation	Purpose
SELECT	$\sigma_{<\text{selection condition}>}(R)$	Selects all tuples that satisfy the selection condition from a relation R
PROJECT	$\pi_{<\text{atttribute list}>}(R)$	New relation with subset of attributes of R and removes duplicate tuples
THETA_JOIN	$R_1 \bowtie_{<\text{join condition}>} R_2$	All combinations of tuples from R_1 and R_2 that satisfy the join condition
EQUIJOIN	$R_1 \bowtie_{<\text{join condition}>} R_2$	Theta join with only equality join comparisons
NATURAL JOIN	$R_1 *_{<\text{join condition}>} R_2$	Equijoin except join attributes of R_2 are not included in the resulting relation
UNION	$R_1 \cup R_2$	Relation that includes all tuples in R_1 or R_2
INTERSECTION	$R_1 \cap R_2$	Relation that includes all tuples in both R_1 and R_2
DIFFERENCE	$R_1 - R_2$	Relation that includes all tuples in R_1 that are not in R_2
CARTESIAN PRODUCT	$R_1 \times R_2$	Relation with attributes of R_1 and R_2 and includes tuples with all possible combinations of tuples of R_1 and R_2
DIVISION	$R_1(Z) \div R_2(Y)$	Relation that includes all tuples t[X] in R_1(Z) that appear in R_1 in combination with every tuple from R_2(Y) where $Z = X \cup Y$
GROUP BY AGGREGATE	$<\text{group attrs}>\mathcal{F}_{<\text{set funcs}>}$	Relation that includes the grouping attributes and the set function values

Banking Example



- Find the names of all customers who have a loan and a savings account at the bank

- Find the names of all customers who have a loan at the Decatur branch but do not have a savings account at any branch of the bank

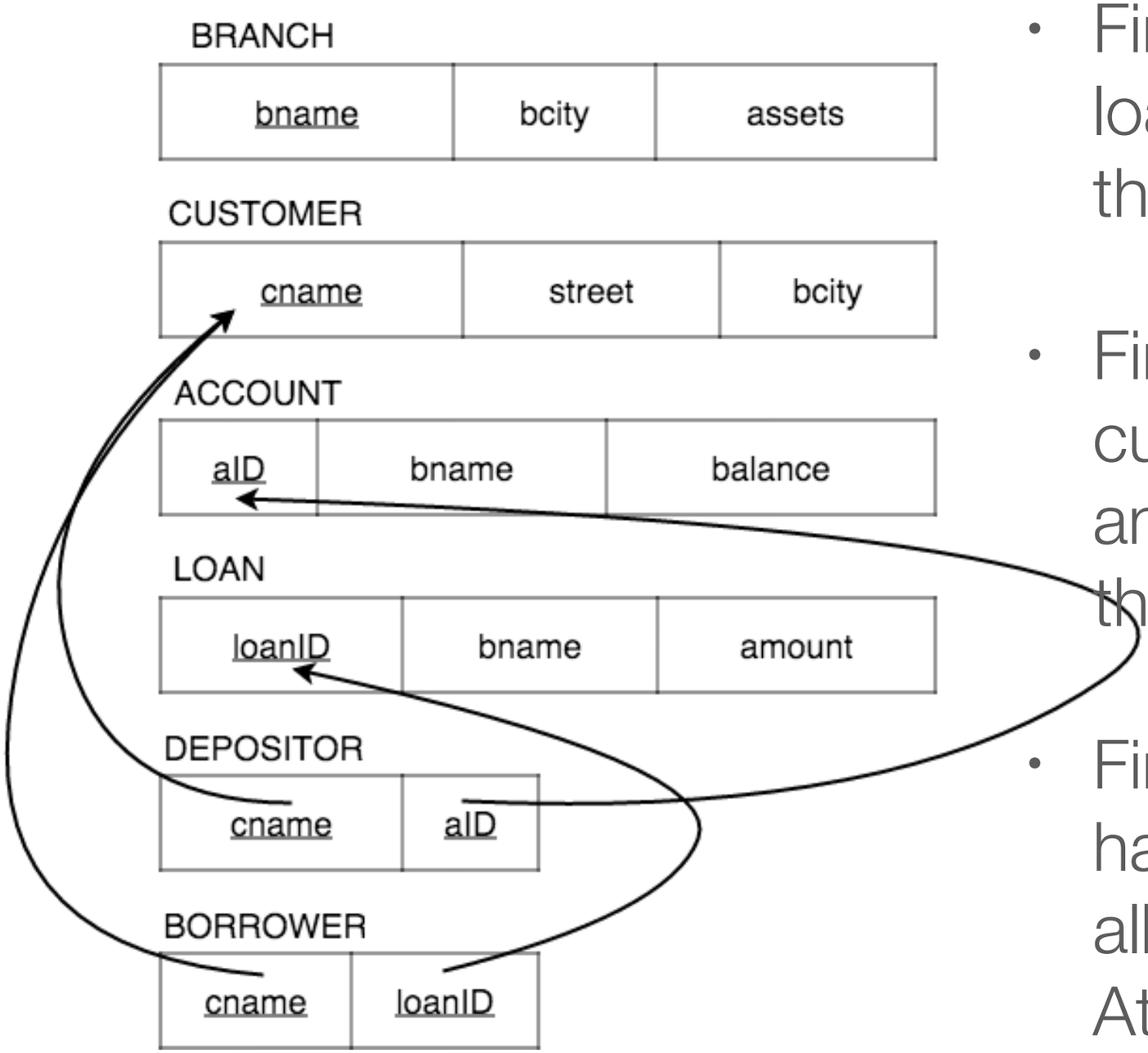
- Find all customers who have a savings account at all branches located in Atlanta city

Relational Calculus (Tuple Relational Calculus)

Query of the form: {t | CONDITION(t) }

- Conditions are formulas and are recursively defined
- Atomic formula (Relation(t), R.a *op* S.b / constant)
- Special formula quantifiers
  - Universal quantifier $(\forall t)\ (\mathrm{Condition}(t))$
  - Existential quantifier $(\exists t)\ (\mathrm{Condition}(t))$

Banking Example



- Find loan number for each loan of an amount greater than $1200
- Find the names of all customers who have a loan and a savings account at the bank
- Find all customers who have a savings account at all branches located in Atlanta city

SQL Data Definition

- Create database
- Create table
  - Attribute datatypes and constraints
  - Key constraints (primary and foreign key)
    - Circular integrity constraints
- Alter tables
  - Add/remove attributes
  - Add/remove constraints
- Drop tables & databases

SQL Query

- Basic SQL query

```
SELECT      [DISTINCT] <attribute list>
FROM         <table list>
[WHERE       <condition on the tables>]
[GROUP BY <grouping attributes>]
[HAVING      <group condition>]
[ORDER BY  <attribute list>   ASC | DESC]
[LIMIT       <number of tuples>]
```

- Nested queries and temporal relations

- Advanced query formulations

SQL Data Modification

Data modification does not return a result but changes the database

- INSERT (add new tuples)

- Literal values (constant or known values):
  INSERT INTO <table name>[(<attr names>)]
  VALUES (<list of values>);

- Result from a **SELECT** command:
  INSERT INTO <table name>[(<attr names>)]
  (<SELECT subquery>)

SQL Data Modification (2)

Data modification does not return a result but changes the database

- DELETE (remove tuples)
  DELETE FROM <relation> WHERE <condition>;
  - Tuples are deleted from only one table at a time unless **CASCADE** is specified on a referential integrity constraint
- UPDATE (change value(s) of existing tuples)

SQL Data Modification (3)

Data modification does not return a result but changes the database

- UPDATE (change value(s) of existing tuples)
  UPDATE <relation>
  SET <list of attribute assignments>
  WHERE <condition>;

- Modify/change certain attributes in certain tuples of a relation

- Only changes tuples that match the WHERE condition

SQL Views

- View is a virtual table that does not exist in physical form
  - Allows ability to present information in different ways to different users
  - Can be used like an ordinary relation and simplifies complex queries
  - Limits data access to specific users (sensitive data can be hidden)
  - If conceptual schema changes, only the **SELECT** query needed to construct view needs to change
- Syntax:
  CREATE VIEW <name> AS <query>;

MySQL Session Variables

- A session starts with a connection to the SQL server and ends when the connection is closed

- Session variables can be created anytime during a SQL session and exists for the remainder of the SQL session

- Always begins with the symbol “@“
  (e.g, @x, @count)

- Syntax:
  SET <varName> = express;
  SELECT … INTO @varname FROM … WHERE …;

MySQL Temporary Tables

- Temporary tables are used to store and process intermediate results:
  CREATE TEMPORARY TABLE
  ...
  - Same selection, update, and join capabilities in typical SQL tables
  - Deleted when the current client session terminates
- Stored functions

MySQL Stored Procedures

- Generalization of SQL by adding programming language-like structure to the SQL language

- Syntax:
  DELIMITER <DL>
  CREATE PROCEDURE <procedure name> (parameters)
  BEGIN
  <statements of the procedure>
  END <DL>

- A stored procedure can only be used within the database where the stored procedure was defined

MySQL Stored Procedures (2)

- Stored procedure can have local variables
  - **BEGIN** and **END** keywords defines the scopes of local variables
    - Inner levels can access variables from outer levels but not vice-versa
- Stored procedure can have parameters (similar to methods in programming language)
  - 3 modes to pass in parameters (IN, OUT, INOUT)

MySQL Stored Procedures (3)

Control structures are available similar to traditional programming languages

- IF statements (conditional)
  - **IF <condition> THEN <command> END IF;**
  - **IF <condition> THEN <command1>**
    ELSE <command2> END IF;

- CASE statements (alternative conditional structure)

CASE <case expression>
  WHEN <expression1> THEN <command1>
  …
  ELSE <commandN>
END CASE;

MySQL Stored Procedures (4)

Control structures are available similar to traditional programming languages

- LOOP statements (repeated execution)

  - **WHILE <condition> DO <commands>**
    END WHILE;

  - REPEAT <commands> UNTIL <condition>
    END REPEAT;

  - <LoopLabel>:
    LOOP
      <commands>
     IF <condition1> THEN LEAVE <LoopLabel>;
     IF <condition2> THEN ITERATE <LoopLabel>;
    END LOOP;

MySQL Stored Functions

User-defined functions

```
CREATE FUNCTION <function_name>(parameter)
    RETURNS datatype
    [NOT] DETERMINISTIC
<statements>;
```

- Returns a single value (similar to aggregate functions)
- Meant to encapsulate common formulas or business rules that are reusable
- Can be used in SQL SELECT statements

JDBC Program Steps

- Import JDBC library (java.sql.*)
- Load appropriate JDBC driver
- Create a connection object
- Create a statement object
- Submit SQL statement
- Process query results
- Close connections