# CSCD 327: Relational Database Systems

## Relational Model

Instructor: Dr. Dan Li

# Relational Databases

- Most common data model in modern
- Many commercial systems
  - Oracle, MS SQL Server, IBM DB2, more...
- Also open source
  - MySQL, PostgreSQL, more...

# Basic Structure

* Formally, given sets $A_1$, $A_2$, …. $A_n$ a **relation** *r* is a subset of
  $A_1$ x $A_2$ x … x $A_n$
  Thus, a relation is a set of *n*-tuples ($a_1$, $a_2$, …, $a_n$) where each $a_i \in A_i$
* Example: If
  * *customer_name* = {Jones, Smith, Curry, Lindsay, …}
    /* Set of all customer names */
  * *customer_street* = {Main, North, Park, …} /* set of all street names*/
  * *customer_city* = {Harrison, Rye, Pittsfield, …} /* set of all city names */

  Then *r* = { (Jones, Main, Harrison),
  (Smith, North, Rye),
  (Curry, North, Rye),
  (Lindsay, Park, Pittsfield) }

  is a relation over
  *customer_name x customer_street x customer_city*

# Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
  - E.g. the value of an attribute can be an account number,
    but cannot be a set of account numbers
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - We shall ignore the effect of null values in our main presentation and consider their effect later

# Relation Schema

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

  *Customer_schema* = (*customer_name, customer_street, customer_city*)

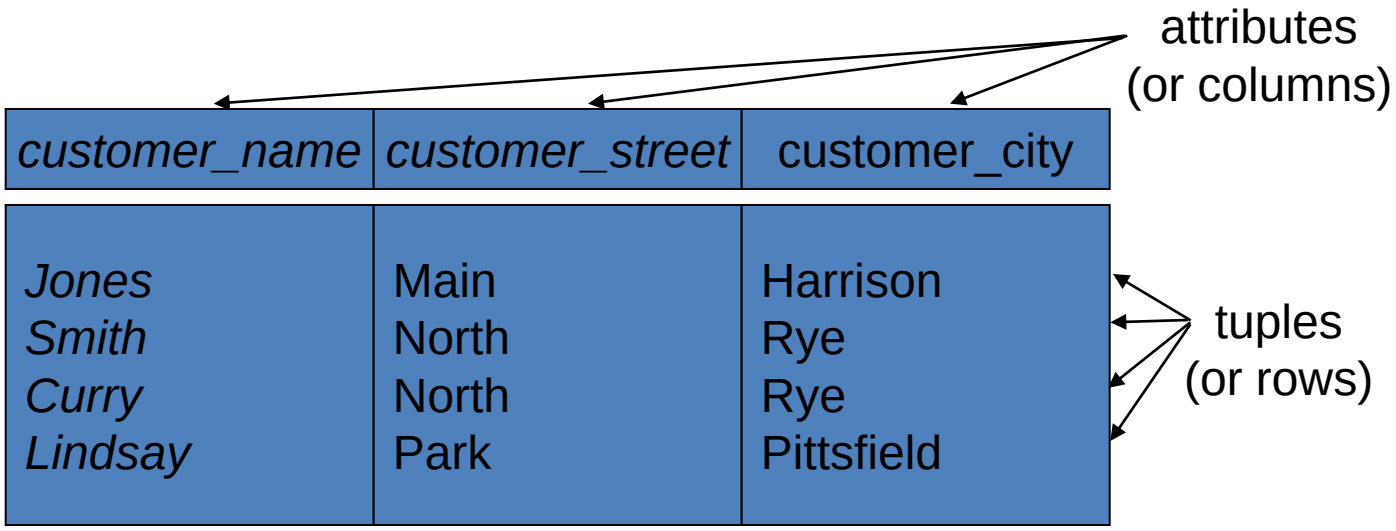
- *r*(*R*) denotes a *relation r* on the *relation schema R*

  Example:

  *customer (Customer_schema)*

# Relation Instance

* The current values (*relation instance*) of a relation are specified by a table
* An element *t* of *r* is a *tuple*, represented by a *row* in a table
* Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
* The degree of a relation is the number of attributes it contains.
* The cardinality of a relation is the number of tuples it contains.




*customer*

# Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  *account* : stores information about accounts
  *depositor* : stores information about which customer owns which account
  *customer* : stores information about customers

- Storing all information as a single relation such as
  *bank*(*account_number, balance, customer_name*, ..)
  results in
  - repetition of information
    - e.g.,if two customers own an account (What gets repeated?)
  - the need for null values
    - e.g., to represent a customer without an account
- Normalization theory deals with how to design relational schemas

# Keys

* Let K $\subseteq$ R
* *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  * by "possible *r* " we mean a relation *r* that could exist in the enterprise we are modeling.
  * Example: {*customer_name, customer_street*} and

    {*customer_name*}

    are both superkeys of *Customer*, if no two customers can possibly have the same name
    * In real life, an attribute such as *customer_id* would be used instead of *customer_name* to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

# Keys (Cont.)

- *K* is a **candidate key** if *K* is minimal
  Example: {*customer_name*} is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.
- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. email address is unique, but may change
- A relation schema may have an attribute that corresponds to the primary key of another relation. Such attribute is called a **foreign key**.
  - E.g. *customer_name* and *account_number* attributes of *depositor* are foreign keys to *customer* and *account* respectively.
  - Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**

# SQL Data Definition Language (DDL)

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.
- **CREATE TABLE** - creates a new database table
- **ALTER TABLE** - alters (changes) a database table
- **DROP TABLE** - deletes a database table

# Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

# Elements of Table Declarations

- Most basic element: an attribute and its type.
- The most common types are:
  - INT or INTEGER (synonyms) - integer
  - REAL or FLOAT- floating point numbers
  - CHAR(*n* ) = fixed-length string of *n* characters.
  - VARCHAR(*n* ) = variable-length string of up to *n* characters.

# DDL - Primitive Types

- numeric
  - INTEGER (or INT), SMALLINT are subsets of the integers (machine dependent)
  - REAL, DOUBLE PRECISION are floating-point and double-precision floating-point (machine dependent)
  - FLOAT(N) is floating-point with at least N digits
  - DECIMAL(P,D) (or DEC(P,D), or NUMERIC(P,D)):
    - P: The maximum total number of decimal digits that can be stored, both to the left and to the right of the decimal point.
    - D: The maximum number of decimal digits that can be stored to the right of the decimal point.

# DDL - Primitive Types (cont.)

- character-string
  - CHAR(N) (or CHARACTER(N)) is a fixed-length character string
  - VARCHAR(N) (or CHAR VARYING(N), or CHARACTER VARYING(N)) is a variable-length character string with at most N characters
- bit-strings
  - BIT(N) is a fixed-length bit string
  - VARBIT(N) (or BIT VARYING(N)) is a bit string with at most N bits

# DDL - Primitive Types (cont.)

- DATE and TIME are types in SQL.
- The form of a date value is:

  yyyy-mm-dd

  - Example: `DATE '2007-09-30'` for Sept. 30, 2007.
- The form of a time value is:

  hh:mm:ss

  with an optional decimal point and fractions of a second following.
  - Example: `TIME '15:30:02.5'` = two and a half seconds after 3:30PM.

# Example: Create Table

```
CREATE TABLE account
   (account_int    varchar(15),
    branch_name
   varchar(15),
    balance          int
    );
```

# Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE (more about unique from chapter 11).
- Either says that no two tuples of the relation may agree in all the attribute(s) on the list.
- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE account
  (account_int   varchar(15) unique,
   branch_name       varchar(15),
   balance           int,
    primary key(account_int));
```

# PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULLs, and there may be several tuples with NULL.

# Changing a Relation

- Simplest form is:

  ALTER TABLE <name>

  *action*;
- The SQL standard restricts each ALTER TABLE statement to a **single** table change.
- Add a column definition to a table
- Drop a column from a table
- Change the default value for a column
- Add or drop a primary key for a table
- Add or drop a new foreign key for a table
- Add or drop a uniqueness constraint for a table
- Add or drop a check constraint for a table

# Example: Alter Table

```
ALTER TABLE CUSTOMER
        ADD CONTACT_PHONE CHAR(10);
```

The new columns will have NULL values for existing customers.

```
ALTER TABLE CUSTOME
        ADD PRIMARY KEY (customer_name);
```