SQL DATA DEFINITION:
KEY CONSTRAINTS
Data Definition

- Covered most of SQL data manipulation operations
- Continue exploration of SQL data definition features
  - Specifying tables and their columns (lecture 4)
  - Declaring views of the logical-level schema (lecture 6)
  - Specifying constraints on individual columns, or entire tables
  - Providing stored procedures to manipulate data
  - Specifying security access constraints
  - …and more!
We will focus on the mechanics of data definition.

For now, ignoring a very important question:
- Exactly what is a “good” database schema, anyway?!!

General design goals:
- Should be able to fully represent all necessary details and relationships in the schema
- Try to eliminate the ability to store invalid data
- Many other design goals too (security, performance)
  - Sometimes these design goals conflict with each other…

DBMSes can enforce many different constraints
- Want to leverage this capability to ensure correctness
SQL provides hierarchical grouping capabilities for managing collections of tables

- Also separate namespaces for different collections of tables

Standard mechanism has three levels:

- Catalogs
- Schemas
- Tables

- Each level is assigned a name
- Within each container, names must be unique

Allows multiple applications to use the same server

- Even multiple instances of a particular application
Catalogs and Schemas (2)

- Every table has a full name:
  - `catalog.schema.table`

- Database systems vary **widely** on implementation of these features!
  - Catalog functionality not covered by SQL specification
  - Schema and table levels are specified
  - Most DBMSes offer some kind of grouping

- Common behaviors:
  - “Databases” generally correspond to catalogs
    - `CREATE DATABASE web_db;`
  - Schema-level grouping is usually provided
    - `CREATE SCHEMA blog_schema;`
Using a Database

- Normally, must connect to a database server to use it
  - Specify a username and password, among other things
- Each database connection has its own environment
  - “Session state” associated with that client
  - Can specify the catalog and schema to use
    - e.g. `USE bank;` to use the banking database
    - e.g. Specifying database `user_db` to the MySQL client
- All operations will use that catalog and schema by default
- Can frequently override using full names for tables, etc.
Creating Tables

- General form:
  
  ```sql
  CREATE TABLE name ( 
    attr1 type1, 
    attr2 type2, 
    ... 
  )
  ```

- SQL provides a variety of standard column types
  - INT, CHAR (N), VARCHAR (N), DATE, etc.
  - (see Lecture 4 for more details about basic column types)

- Table and column names must follow specific rules
- Table must have a unique name within schema
- All columns must have unique names within the table
Table Constraints

- By default, SQL tables have no constraints
  - Can insert multiple copies of a given row
  - Can insert rows with NULL values in any column
- Can specify columns that comprise primary key
  ```sql
  CREATE TABLE account (
    account_number CHAR(10),
    branch_name VARCHAR(20),
    balance NUMERIC(12, 2),
    PRIMARY KEY (account_number)
  );
  ```
  - No two rows can have same values for primary key
  - A table can have only one primary key
Primary Key Constraints

- Alternate syntax for primary keys
  ```sql
  CREATE TABLE account (
    account_number CHAR(10)       PRIMARY KEY,
    branch_name     VARCHAR(20),
    balance         NUMERIC(12, 2)
  );
  ```
  - Can only be used for single-column primary keys!

- For multi-column primary keys, must specify primary key after column specifications
  ```sql
  CREATE TABLE depositor (
    customer_name VARCHAR(30),
    account_number CHAR(10),
    PRIMARY KEY (customer_name, account_number)
  );
  ```
Null-Value Constraints

- Every attribute domain contains null by default
  - Same with SQL: every column can be set to NULL, if it isn’t part of a primary key
- Often, NULL is not an acceptable value!
  - e.g. bank accounts must always have a balance
- Can specify NOT NULL to exclude NULL values for particular columns
  - NOT NULL constraint specified in column declaration itself
- Stating NOT NULL for primary key columns is unnecessary and redundant
Account Relation

- Account number is a primary key
  - Already cannot be **NULL**
- Branch name and balance also should always be specified
  - Add **NOT NULL** constraints to those columns
- SQL:
  ```sql
  CREATE TABLE account (
      account_number CHAR(10)       PRIMARY KEY,
      branch_name    VARCHAR(20)    NOT NULL,
      balance        NUMERIC(12, 2) NOT NULL
  );
  ```
Other Candidate Keys

- Some relations have multiple candidate keys
- Can specify candidate keys with **UNIQUE** constraints
  - Like primary key constraints, can specify candidate keys in the column declaration, or after all columns
  - Can only specify multi-column candidate key after the column specifications
- Unlike primary keys, **UNIQUE** constraints **do not** exclude **NULL** values!
  - This constraint considers **NULL** values to be unequal!
  - If some attributes in the **UNIQUE** constraint allow **NULLs**, DB will allow multiple rows with the same values!
Example: An employee relation

```sql
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_ssn CHAR(9) NOT NULL UNIQUE,
    emp_name VARCHAR(40) NOT NULL,
    ...
);
```

- Employee’s ID is the primary key
- All employees need a SSN, but no two employees should have the same SSN
  - Don’t forget NOT NULL constraint too!
- All employees should have a name, but multiple employees might have same name
UNIQUE and NULL

Example:

```sql
CREATE TABLE customer (  
cust_name VARCHAR(30) NOT NULL,  
address VARCHAR(60),  
UNIQUE (cust_name, address)
);
```

Try inserting values:

```sql
INSERT INTO customer  
VALUES ('John Doe', '123 Spring Lane');
INSERT INTO customer  
VALUES ('John Doe', '123 Spring Lane');
```

Second insert fails, as expected:

```
Duplicate entry 'John Doe-123 Spring Lane' for key 'cust_name'
```
Example:

```sql
CREATE TABLE customer (  
cust_name VARCHAR(30) NOT NULL,  
address VARCHAR(60),  
UNIQUE (cust_name, address)  
);
```

Try inserting more values:

```sql
INSERT INTO customer VALUES ('Jane Doe', NULL);  
INSERT INTO customer VALUES ('Jane Doe', NULL);  
```

Both inserts succeed!

Be careful using nullable columns in UNIQUE constraints!

- Usually, you really want to specify `NOT NULL` for all columns that appear in UNIQUE constraints
CHECK Constraints

- Often want to specify other constraints on values
- Can require values in a table to satisfy some predicate, using a CHECK constraint
  - Very effective for constraining columns’ domains, and eliminating obviously bad inputs
- CHECK constraints must appear after the column specifications
- In theory, can specify any expression that generates a Boolean result
  - This includes nested subqueries!
  - In practice, DBMS support for CHECK constraints varies widely, and is often quite limited
CHECK Constraint Examples

- Can constrain values in a particular column:

```sql
CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  emp_ssn CHAR(9) NOT NULL UNIQUE,
  emp_name VARCHAR(40) NOT NULL,
  pay_rate NUMERIC(5,2) NOT NULL,
  CHECK (pay_rate > 5.25)
);
```

- Ensures that all employees have a minimum wage
CREATE TABLE employee (  
  emp_id INT PRIMARY KEY,  
  emp_ssn CHAR(9) NOT NULL UNIQUE,  
  emp_name VARCHAR(40) NOT NULL,  
  status VARCHAR(10) NOT NULL,  
  pay_rate NUMERIC(5,2) NOT NULL,  
  CHECK (pay_rate > 5.25),  
  CHECK (status IN  
    ('active', 'vacation', 'suspended'))  
);

- Employee status must be one of the specified values
  - Like an enumerated type
  - (Many DBs provide similar support for enumerated types)
Another **CHECK** Constraint

- **Depositor relation:**
  
  ```sql
  CREATE TABLE depositor (  
    customer_name  VARCHAR(30),
    account_number CHAR(10),
  PRIMARY KEY (customer_name, account_number),
  CHECK (account_number IN  
    (SELECT account_number FROM account))
  )
  ```

- **Rows in depositor table should only contain valid account numbers!**
  - The valid account numbers appear in account table
  - This is a **referential integrity** constraint
Another CHECK Constraint (2)

- Depositor relation:
  
  ```
  CREATE TABLE depositor (  
      customer_name VARCHAR(30),  
      account_number CHAR(10),  
      PRIMARY KEY (customer_name, account_number),  
      CHECK (account_number IN  
              (SELECT account_number FROM account))  
  );
  ```

- When does this constraint need to be checked?
  - When changes are made to depositor table
  - Also when changes are made to account table!
CHECK Constraints

- Easy to write very expensive CHECK constraints
- CHECK constraints aren’t used very often
  - Lack of widespread support; using them limits portability
  - When used, they are usually very simple
    - Enforce more specific constraints on data values, or enforce string format constraints using regular expressions, etc.
  - Avoid huge performance impacts!
- Don’t use CHECK constraints for referential integrity 😊
  - There’s a better way!
Referential Integrity Constraints

- Referential integrity constraints are very important!
  - These constraints span multiple tables
  - Allow us to associate data across multiple tables
  - One table’s values are constrained by another table’s values

- A relation can specify a primary key
  - A set of attributes that uniquely identifies each tuple in the relation

- A relation can also include attributes of another relation’s primary key
  - Called a foreign key
  - Referencing relation’s values for the foreign key must also appear in the referenced relation
Given a relation \( r(R) \)
- \( K \subseteq R \) is the primary key for \( R \)

Another relation \( s(S) \) references \( r \)
- \( K \subseteq S \) too
- \( \left\langle \forall t_s \in s : \exists t_r \in r : t_s[K] = t_r[K] \right\rangle \)

Also called a subset dependency
- \( \Pi_K(s) \subseteq \Pi_K(r) \)
- Foreign-key values in \( s \) must be a subset of primary-key values in \( r \)
SQL Foreign Key Constraints

- Like primary key constraints, can specify in multiple ways
- For a single-column foreign key, can specify in column declaration
- Example:
  ```sql
  CREATE TABLE depositor (  
    customer_name VARCHAR(30) REFERENCES customer,  
    account_number CHAR(10) REFERENCES account,  
    PRIMARY KEY (customer_name, account_number),  
  );
  ```
  - Foreign key refers to primary key of referenced relation
- Foreign-key constraint does **NOT** imply **NOT** **NULL**!
  - Must explicitly add this, if necessary
  - In this example, **PRIMARY KEY** constraint eliminates **NULLs**
Can also specify the column in the referenced relation

Especially useful when referenced column is a candidate key, but not the primary key

Example:
- Employees have both company-assigned IDs and social security numbers
- Health benefit information in another table, tied to social security numbers
Foreign Key Example

- Employee information:
  ```sql
  CREATE TABLE employee (  
    emp_id INT PRIMARY KEY,  
    emp_ssn CHAR(9) NOT NULL UNIQUE,  
    emp_name VARCHAR(40) NOT NULL,  
    ...  
  );
  ```

- Health plan information:
  ```sql
  CREATE TABLE healthplan (  
    emp_ssn CHAR(9) PRIMARY KEY REFERENCES employee (emp_ssn),  
    provider VARCHAR(20) NOT NULL,  
    pcp_id INT NOT NULL,  
    ...  
  );
  ```
Multiple Constraints

- Can combine several different constraints

```sql
emp_ssn CHAR(9) PRIMARY KEY
REFERENCES employee (emp_ssn)
```

- `emp_ssn` is primary key of `healthplan` relation
- `emp_ssn` is also a foreign key to `employee` relation
- Foreign key references the candidate-key `employee.emp_ssn`
Self-Referencing Foreign Keys

- A relation can have a foreign key reference to itself
  - Common for representing hierarchies or graphs
- Example:
  ```sql
  CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    emp_ssn CHAR(9) NOT NULL UNIQUE,
    emp_name VARCHAR(40) NOT NULL,
    ...,
    manager_id INT REFERENCES employee
  );
  ```
  - `manager_id` and `emp_id` have the same domain — the set of valid employee IDs
  - Allow **NULL** manager IDs for employees with no manager
Alternate Foreign Key Syntax

- Can also specify foreign key constraints after all column specifications
  - Required for multi-column foreign keys

- Example:

```sql
CREATE TABLE employee (
    emp_id INT,
    emp_ssn CHAR(9) NOT NULL,
    emp_name VARCHAR(40) NOT NULL,
    ...
    manager_id INT,

    PRIMARY KEY (emp_id),
    UNIQUE (emp_ssn),
    FOREIGN KEY (manager_id) REFERENCES employee
);
```
Multi-Column Foreign Keys

- Multi-column foreign keys can also be affected by NULL values
  - Individual columns may allow NULL values
- If all values in foreign key are non-NUL then the foreign key constraint is enforced
- If any value in foreign key is NULL then the constraint cannot be enforced!
  - Or, “the constraint is defined to hold” (lame...)
Example Bank Schema

- Account relation:
  ```sql
  CREATE TABLE account (
    account_number VARCHAR(15) NOT NULL,
    branch_name    VARCHAR(15) NOT NULL,
    balance        NUMERIC(12,2) NOT NULL,
    PRIMARY KEY (account_number)
  );
  ```

- Depositor relation:
  ```sql
  CREATE TABLE depositor (
    customer_name  VARCHAR(15) NOT NULL,
    account_number VARCHAR(15) NOT NULL,
    PRIMARY KEY (customer_name, account_number),
    FOREIGN KEY (account_number) REFERENCES account,
    FOREIGN KEY (customer_name) REFERENCES customer
  );
  ```
Foreign Key Violations

- Several ways to violate foreign key constraints
- If referencing relation gets a bad foreign-key value, the operation is simply forbidden
  - e.g. trying to insert a row into `depositor` relation, where the row contains an invalid account number
  - e.g. trying to update a row in `depositor` relation, trying to change customer name to an invalid value
- More subtle issues when the referenced relation is changed
  - What to do with `depositor` if a row is deleted from `account`?
Example Bank Data

- **account data:**

<table>
<thead>
<tr>
<th>account_number</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A-215</td>
<td>Mianus</td>
<td>700.00</td>
</tr>
<tr>
<td>A-217</td>
<td>Brighton</td>
<td>750.00</td>
</tr>
<tr>
<td>A-222</td>
<td>Redwood</td>
<td>700.00</td>
</tr>
<tr>
<td>A-305</td>
<td>Round Hill</td>
<td>350.00</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **depositor data:**

<table>
<thead>
<tr>
<th>customer_name</th>
<th>account_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Smith</td>
<td>A-215</td>
</tr>
<tr>
<td>Jones</td>
<td>A-217</td>
</tr>
<tr>
<td>Lindsay</td>
<td>A-222</td>
</tr>
<tr>
<td>Turner</td>
<td>A-305</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Try to delete A-222 from account. What should happen?
Foreign Key Violations

- **Option 1: Disallow the delete from account**
  - Force the user to remove all rows in depositor relation that refer to A-222
  - Then user may remove row A-222 in account relation
  - Default for SQL. Also a pain, but probably a good choice.

- **Option 2: Cascade the delete operation**
  - If user deletes A-222 from account relation, all referencing rows in depositor should also be deleted
  - Seems reasonable; rows in depositor only make sense in context of corresponding rows in account
Foreign Key Violations (2)

- Option 3: Set foreign key value to **NULL**
  - If primary key goes away, update referencing row to indicate this.
  - Foreign key column can’t specify **NOT NULL** constraint
  - Doesn’t make sense in every situation
    - Doesn’t make sense in account and depositor example!

- Option 4: Set foreign key value to some default
  - Can specify a default value for columns
  - (Haven’t talked about how to do this in SQL, yet.)
Cascading Changes

- Can specify behavior on foreign key constraint
  
  CREATE TABLE depositor (  
  ...  
  FOREIGN KEY (account_number) REFERENCES account  
  ON DELETE CASCADE,  
  FOREIGN KEY (customer_name) REFERENCES customer  
  ON DELETE CASCADE  
  );

- When account A-222 is deleted from account relation, corresponding rows in depositor will be deleted too

- Read: “When a row is deleted from referenced relation, corresponding rows are deleted from this relation.”

- Similar considerations for updates to primary key values in the referenced relation

- Can also specify ON UPDATE behaviors
Integrity constraints are a very powerful feature of the relational model

SQL provides many ways to specify and enforce constraints
- Actual support for different kinds of constraints varies among DBMSes

Allows a database to exclude all invalid values

Database can also resolve some integrity violations automatically
- e.g. cascade deletion of rows from referencing relations, or setting foreign key values to NULL