Relational Model Definitions

- A **relation** is a table with columns and rows.
- An **attribute** is a named column of a relation.
- A **tuple** is a row of a relation.
- A **domain** is a set of allowable values for one or more attributes.
- The **degree** of a relation is the number of attributes it contains.
- The **cardinality** of a relation is the number of tuples it contains.
- A **relational database** is a collection of normalized relations with distinct relation names.

Integrity Constraints (ICs)

- **IC**: condition that must be true for any instance of the database; e.g., **domain constraints**.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A **legal** instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- Why is this useful
  - If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!
- Think of the constraints as the business rules derived from the application

Relational Integrity

- Integrity rules are used to insure the data is accurate.
- **Constraints** are rules or restrictions that apply to the database and limit the data values it may store.
- Types of constraints:
  - **Domain constraint** - Every value for an attribute must be an element of the attribute's domain or be null. null represents a value that is currently unknown or not applicable. null is not the same as zero or an empty string.
  - **Entity integrity constraint** - In a base relation, no attribute of a primary key can be null.
  - **Key constraint** – every relation must have a key
  - **Referential integrity constraint** - If a foreign key exists in a relation, then the foreign key value must match a primary key value of a tuple in the referenced relation or be null.
Relational Query Languages

- **Query languages:**
  - Allow specification of schemas and constraints
  - Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages ≠ programming languages!
  - QLs not expected to be “Turing complete”.
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Query Languages

- Formal query languages are defined as mathematical operators over the set
  - What is the advantage of a formal language?
  - Relational algebra, Relational calculus are examples
- Procedural vs Non-procedural languages
  - Procedural: what data to fetch from DB and how/where to get the data
  - Non-procedural: what data to fetch from DB
    - System/DBMS needs to figure out the “how”
  - Can have a mix in practice
    - **Relational algebra:** procedural language
    - **Relational calculus:** non-procedural (declarative)

SQL: Structured Query Language

- The standard language for relational data
  - Invented by folks at IBM, esp. Don Chamberlin
  - Actually not a great language...
  - Beat a more elegant competing standard, QUEL, from Berkeley
- Separated into a DML & DDL
- SQL DML component based on relational algebra & calculus
  - Data definition (DDL) – to define schema/tables
    - Define Schema
    - Define Constraints

SQL Overview

- Structured Query Language or SQL is the standard query language for relational databases.
  - It first became an official standard in 1986 as defined by the American National Standards Institute (ANSI).
  - All major database vendors conform to the SQL standard with minor variations in syntax (different dialects).
  - SQL consists of both a Data Definition Language (DDL) and a Data Manipulation Language (DML).
- SQL is a declarative language (non-procedural). A SQL query specifies what to retrieve but not how to retrieve it.
  - Basic SQL is not a complete programming language as it does not have control or iteration commands.
    - Procedural extensions: PL/SQL (Oracle), T-SQL (SQL Server)
SQL History

- 1970 - Codd invents relational model and relational algebra
- 1974 - D. Chamberlin (also at IBM) defined Structured English Query Language (SEQUEL)
- 1976 - SEQUEL/2 defined and renamed SQL for legal reasons. Origin of pronunciation 'See-QueI' but official pronunciation is 'S-Q-L'.
- Late 1970s - System R, Oracle, INGRES implement variations of SQL-like query languages.
- 1982 - standardization effort on SQL begins
- 1986 - became ANSI official standard
- 1987 - became ISO standard
- 1992 - SQL2 (SQL92) revision
- 1999 - SQL3 (supports recursion, object-relational)

SQL Query Language: DML

- The most widely used relational query language. Current standard is SQL-92.
- SELECT clause
  - What attributes you want
  - What relations/tables to search
  - What condition/predicate to apply

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT * from Students S
WHERE S.age=18
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
</tbody>
</table>

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

SQL Basic Rules

- Some basic rules for SQL statements:
  1) There is a set of reserved words that cannot be used as names for database objects. (e.g. SELECT, FROM, WHERE)
  2) SQL is case-insensitive. Only exception is string constants. 'FRED' not the same as 'fred'.
  3) SQL is free-format and white-space is ignored.
  4) The semi-colon is often used as a statement terminator, although that is not always required.
  5) Date and time constants have defined format:
      Dates: 'YYYY-MM-DD' e.g. '1975-05-17'
      Times: 'hh:mm:ss[.f]' e.g. '15:00:00'
      Timestamp: 'YYYY-MM-DD hh:mm:ss[.f]' e.g. '1975-05-17 15:00:00'
  6) Two single quotes " are used to represent a single quote character in a character constant. e.g. "Master"s".
Next: DDL and Constraint specifications

- Specifying schema/table
- Specifying constraints

SQL DDL Overview

- SQL contains a data definition language (DDL) that allows users to:
  - add, modify, and drop tables
  - create views
  - define and enforce integrity constraints
  - enforce security restrictions

SQL Identifiers

- Identifiers are used to identify objects in the database such as tables, views, and columns.
  - The identifier is the name of the database object.

- An SQL identifier (name) must follow these rules:
  - only contain upper or lower case characters, digits, and underscore ("_") character
  - be no longer than 128 characters
    - DB vendors may impose stricter limits than this.
  - must start with a letter (or underscore)
  - cannot contain spaces
  - Note: Quoted or delimited identifiers enclosed in double quotes allow support for spaces and other characters. E.g. "select"

SQL Data Types

- In the relational model, each attribute has an associated domain of values.

- In SQL, each column (attribute) has a data type that limits the values that it may store. The standard SQL data types are similar to their programming language equivalents.

- The database will perform (implicit) data type conversion when necessary.

- Explicit data type conversion using functions such as CAST and CONVERT.
### SQL Data Types (2)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>CHAR</td>
<td>Fixed length string (padded with blanks) e.g. CHAR(10)</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>Variable length string e.g. 'VARCHAR(50)'</td>
</tr>
<tr>
<td>BIT</td>
<td>Bit string e.g. 'BIT(4)' can store '0101'</td>
</tr>
<tr>
<td>NUMERIC or DECIMAL</td>
<td>Exact numeric data type e.g. NUMERIC(7,2) has a precision of 7 digits and 2 scale</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer data only</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Smaller space than INTEGER</td>
</tr>
<tr>
<td>FLOAT or REAL</td>
<td>Approximate numeric data types.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Precision dependent on implementation.</td>
</tr>
<tr>
<td>DATE</td>
<td>Stores YEAR, MONTH, DAY</td>
</tr>
<tr>
<td>TIME</td>
<td>Stores HOUR, MINUTE, SECOND</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Stores date and time data</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>Time interval.</td>
</tr>
<tr>
<td>CHARACTER LARGE OBJECT</td>
<td>Stores a character array (e.g. for a document)</td>
</tr>
<tr>
<td>BINARY LARGE OBJECT</td>
<td>Stores a binary array (e.g. for a picture, movie)</td>
</tr>
</tbody>
</table>

### SQL User Defined Data Types
- Analogous to programming languages (like C)
- The `CREATE DOMAIN` command allows you to define your own types that are subsets of built-in types:

```
CREATE DOMAIN domainName AS dataType
    [DEFAULT defaultValue]
    [CHECK (condition)]
```

- Example: Create user-defined domain for `Emp.title`:

```
CREATE DOMAIN titleType AS CHAR(2)
    DEFAULT 'EE'
    CHECK (VALUE IN ('EE', 'SA', 'PR', 'ME'));
```

### SQL User Defined Data Types (2)
- The `CHECK` clause can use a nested select statement to retrieve values from the database:

```
CREATE DOMAIN mgrType AS CHAR(5)
    DEFAULT NULL
    CHECK (VALUE IN (SELECT eno FROM emp
                        WHERE title = 'ME' OR title = 'SA'));
```

- Domains can be removed from the system using `DROP`:

```
DROP DOMAIN domainName [RESTRICT | CASCADE]
```
- `RESTRICT` - if domain is currently used, drop fails.
- `CASCADE` - if domain is currently used, domain dropped and fields using domain defaulted to base type.

- Example:

```
DROP DOMAIN mgrType;
```

### Example Schema
- Relational database schema:

```
emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dep (dno, dname, mgreno)
workson (eno, pno, resp, hours)
```
COMPANY Database Schema

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>DEPARTMENT</th>
<th>DEPT_LOCATIONS</th>
<th>PROJECT</th>
<th>WORKS_ON</th>
<th>DEPENDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td>Mnt</td>
<td>Lname</td>
<td>Str</td>
<td>Bldno</td>
<td>Address</td>
</tr>
<tr>
<td>Salary</td>
<td>Super</td>
<td>Dno</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CREATE TABLE

The CREATE TABLE command is used to create a table in the database. A table consists of a table name, a set of fields with their names and data types, and specified constraints.

The general form is:

```
CREATE TABLE tableName (
  attr1Name attr1Type  [attr1_constraints],
  attr2Name attr2Type  [attr2_constraints],
  ...
  attrMName attrMType  [attrM_constraints],
  [primary and foreign key constraints]
);
```

Example

The CREATE TABLE command for the Emp relation:

```
CREATE TABLE emp (
  eno CHAR(5),
  ename VARCHAR(30) NOT NULL,
  bdate DATE,
  title CHAR(2),
  salary DECIMAL(9,2),
  supereno CHAR(5),
  dno CHAR(5),
  PRIMARY KEY (eno),
  FOREIGN KEY (dno) REFERENCES dept(dno)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```
**SQL Constraints - Entity Integrity**

- **Entity Integrity constraint** - The primary key of a table must contain a unique, non-null value for each row. The primary key is specified using the `PRIMARY KEY` clause.
  - e.g. `PRIMARY KEY (eno)` (for Emp relation)
  - e.g. `PRIMARY KEY (eno,pno)` (for WorksOn relation)
  - It is also possible to use `PRIMARY KEY` right after defining the attribute in the `CREATE TABLE` statement.

- There can only be one primary key per relation, other candidate keys can be specified using `UNIQUE`:
  - e.g. `UNIQUE (ename)`

**SQL Constraints - Referential Integrity**

- **Referential integrity constraint** - Defines a foreign key that references the primary key of another table.
  - If a foreign key contains a value that is not `NULL`, that value must be present in some tuple in the relation containing the referenced primary key.

- **Example:** Workson contains two foreign keys:
  - `workson.eno` references `emp.eno`
  - `workson.pno` references `proj.pno`

- Specify foreign keys using `FOREIGN KEY` syntax:
  
  ```
  FOREIGN KEY (eno) REFERENCES emp(eno)
  ```

**SQL Referential Integrity Example**

- The `CREATE TABLE` command for the `workson` relation:
  ```
  CREATE TABLE workson {
    eno CHAR(5),
    pno CHAR(5),
    resp VARCHAR(20),
    hours SMALLINT,
    PRIMARY KEY (eno,pno),
    FOREIGN KEY (eno) REFERENCES emp(eno),
    FOREIGN KEY (pno) REFERENCES proj(pno)
  };
  ```

**SQL Referential Integrity and Updates**

- When you try to `INSERT` or `UPDATE` a row in a relation containing a foreign key (e.g. `workson`) that operation is rejected if it violates referential integrity.

- When you `UPDATE` or `DELETE` a row in the primary key relation (e.g. `emp` or `proj`), you have the option on what happens to the values in the foreign key relation (`workson`):
  - 1) `CASCADE` - Delete (update) values in foreign key relation when primary key relation has rows deleted (updated).
  - 2) `SET NULL` - Set foreign key fields to `NULL` when corresponding primary key relation row is deleted.
  - 3) `SET DEFAULT` - Set foreign key values to their default value (if defined).
  - 4) `NO ACTION` - Reject the request on the parent table.
**SQL Referential Integrity Example (2)**

```sql
CREATE TABLE workson (
    eno CHAR(5),
    pno CHAR(5),
    resp VARCHAR(20),
    hours SMALLINT,
    PRIMARY KEY (eno,pno),
    FOREIGN KEY (eno) REFERENCES emp(eno)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (pno) REFERENCES proj(pno)
        ON DELETE NO ACTION ON UPDATE CASCADE
);```

---

**SQL CREATE TABLE Full Syntax**

- Full syntax of CREATE TABLE statement:

```sql
CREATE TABLE tableName (  
    attrName attrType [NOT NULL] [UNIQUE] [PRIMARY KEY]  
        [DEFAULT value] [CHECK (condition)] }  
    [FOREIGN KEY (colList) REFERENCES tbl [(colList)],  
        [ON UPDATE action]  
        [ON DELETE action] ] }  
{[CHECK (condition)] }  
);```

---

**Creating the Example Database**

```
CREATE DOMAIN T_eno AS CHAR(5);
CREATE DOMAIN T_pno AS CHAR(5);
CREATE DOMAIN T_dno AS CHAR(5);

CREATE TABLE emp (  
    eno T_eno,
    ename VARCHAR(30) NOT NULL,
    bdate DATE, title CHAR(2),
    salary DECIMAL(9,2),
    supereno T_eno,
    dno T_dno
    PRIMARY KEY (eno),
    FOREIGN KEY (dno) REFERENCES dept(dno)  
        ON DELETE SET NULL ON UPDATE CASCADE
    );
```

---

**Creating the Example Database (2)**

```
CREATE TABLE workson (  
    eno T_eno,
    pno T_pno,
    resp VARCHAR(20),
    hours SMALLINT,
    PRIMARY KEY (eno,pno),
    FOREIGN KEY (eno) REFERENCES emp(eno)
        ON DELETE NO ACTION ON UPDATE CASCADE,
    FOREIGN KEY (pno) REFERENCES proj(pno)
        ON DELETE NO ACTION ON UPDATE CASCADE
);```

---

**Question:**
Write CREATE TABLE statements to build the proj and dept relations:
- dept(dno, dname, mgreno)
- proj(pno, pname, budget, dno)
Defining a Database

- There is typically a hierarchy of database objects that you can create, alter, and destroy.
  
  
  Database \(\rightarrow\) database instance
  
  Schema \(\rightarrow\) collection of objects (same user)
  
  Table \(\rightarrow\) relation containing columns

- SQL does not standardize how to create a database. A database often contains one or more catalogs, each of which contains a set of schemas.

  To make things more complicated, many DBMSs do not implement everything and rename things. e.g. A database IS a schema for MySQL (there is no CREATE SCHEMA command).

Creating Schemas

- A schema is a collection of database objects (tables, views, domains, etc.) usually associated with a single user.

- Creating a schema: (User Joe creates the schema)

  ```sql
  CREATE SCHEMA employeeSchema AUTHORIZATION Joe;
  ```

- Dropping a schema:

  ```sql
  DROP SCHEMA employeeSchema;
  ```

ALTER TABLE

- The ALTER TABLE command can be used to change an existing table. This is useful when the table already contains data and you want to add or remove a column or constraint.

  DB vendors may support only parts of ALTER TABLE or may allow additional changes including changing the data type of a column.

- General form:

  ```sql
  ALTER TABLE tableName
  [ADD [COLUMN] colName dataType [NOT NULL] [UNIQUE] [DEFAULT value] [CHECK (condition)]]
  [DROP [COLUMN] colName [RESTRICT | CASCADE]]
  [ADD [CONSTRAINT (constraintName)] constraintDef]
  [DROP CONSTRAINT constraintName [RESTRICT | CASCADE]]
  [ALTER [COLUMN] SET DEFAULT defValue]
  [ALTER [COLUMN] DROP DEFAULT]
  ```

ALTER TABLE Examples

Add column location to dept relation:

```sql
ALTER TABLE dept
ADD location VARCHAR(50);
```  

Add field SSN to Emp relation:

```sql
ALTER TABLE emp
ADD SSN CHAR(10);
```  

Indicate that SSN is UNIQUE in emp:

```sql
ALTER TABLE emp
ADD CONSTRAINT ssnConst UNIQUE(SSN);
```
**DROP TABLE**

- The command **DROP TABLE** is used to delete the table definition and all data from the database:

  ```sql
  DROP TABLE tableName [RESTRICT | CASCADE];
  ```

- Example:

  ```sql
  DROP TABLE emp;
  ```

  Question: What would be the effect of the command:

  ```sql
  DROP TABLE emp CASCADE;
  ```

**Database Updates**

- Database updates such as inserting rows, deleting rows, and updating rows are performed using their own statements.

- Insert is performed using the **INSERT** command:

  ```sql
  INSERT INTO tableName [(column list)]
  VALUES (data value list)
  ```

- Examples:

  ```sql
  INSERT INTO emp VALUES ('E9', 'S. Smith', DATE '1975-03-05', 'SA', '60000', 'E8', 'D1');
  INSERT INTO proj (pno, pname) VALUES ('P6', 'Programming');
  ```

  Note: If column list is omitted, values must be specified in order they were created in the table. If any columns are omitted from the list, they are set to NULL.

**INSERT Multiple Rows**

- INSERT statement extended by many databases to take multiple rows:

  ```sql
  INSERT INTO tableName [(column list)]
  VALUES (data value list) [, (values) ]
  ```

- Example:

  ```sql
  INSERT INTO emp (eno, ename) VALUES
  ('E10', 'Fred'), ('E11', 'Jane'), ('E12', 'Joe')
  ```

**INSERT rows from SELECT**

- Insert multiple rows that are the result of a **SELECT** statement:

  ```sql
  INSERT INTO tableName [(column list)]
  SELECT ...
  ```

- Example: Add rows to a temporary table that contains only employees with title = 'EE'.

  ```sql
  INSERT INTO tmpTable
  SELECT eno, ename
  FROM emp
  WHERE title = 'EE'
  ```
**UPDATE Statement**

- Updating existing rows is performed using `UPDATE` statement:
  
  ```sql
  UPDATE tableName
  SET col1 = val1 [,col2=val2...] 
  [WHERE condition]
  ```

- Examples:
  1. Increase all employee salaries by 10%.
     ```sql
     UPDATE emp SET salary = salary*1.10;
     ```
  2. Decrease salaries of employees in department 'D1' by 8% and put their title after their name.
     ```sql
     UPDATE emp
     SET salary = salary*0.92, ename=concat(ename, ' ', title) 
     WHERE dno = 'D1';
     ```

**DELETE Statement**

- Rows are deleted using the `DELETE` statement:
  
  ```sql
  DELETE FROM tableName 
  [WHERE condition]
  ```

- Examples:
  1. Fire everyone in the company.
     ```sql
     DELETE FROM workson;
     DELETE FROM emp;
     ```
  2. Fire everyone making over $35,000.
     ```sql
     DELETE FROM emp 
     WHERE salary > 35000;
     ```

**Practice Questions**

- Relational database schema:
  ```sql
  emp (eno, ename, bdate, title, salary, supereno, dno) 
  proj (pno, pname, budget, dno) 
  dept (dno, dname, mgreno) 
  workson (eno, pno, resp, hours)
  ```

  1) Insert a department with number 'D5', name 'Useless', and no manager.
  2) Insert a `workson` record with eno='E1' and pno='P3'.
  3) Delete all records from `emp`.
  4) Delete only the records in `workson` with more than 20 hours.
  5) Update all employees to give them a 20% pay cut.
  6) Update the projects for `dno='D3'` to increase their budget by 10%.

**DDL Summary**

- SQL contains a data definition language that allows you to `CREATE`, `ALTER`, and `DROP` database objects such as tables, triggers, indexes, schemas, and views.

- Constraints are used to preserve the integrity of the database:
  ```
  • CHECK can be used to validate attribute values.
  • Entity integrity constraint - The primary key of a table must contain a unique, non-null value for each row.
  • Referential integrity constraint - Defines a foreign key that references a unique key of another table.
  ```

- `INSERT`, `DELETE`, and `UPDATE` commands modify the data stored within the database.
Summary… and next

- SQL Intro… DDL
  - Schema definition
  - Constraints

Next:

- Formal Query Languages:
  - Relational Algebra
  - Relational tuple calculus