Continuing

- SQL (Chapter 3, 4)
  - Views (4.2)
  - Triggers (5.3)
  - Integrity Constraints (4.4)
  - Functions and Procedures (5.2), Recursive Queries (5.4), Authorization (4.6), Ranking (5.5)
  - Transactions
- Relational Algebra
- E/R Diagrams (7)

- Some Complex SQL Examples

Integrity Constraints

- Predicates on the database
- Must always be true (checked whenever db gets updated)

- There are the following 4 types of IC’s:
  - Key constraints (1 table)
    - e.g., 2 accts can’t share the same acct_no
  - Attribute constraints (1 table)
    - e.g., accts must have nonnegative balance
  - Referential Integrity constraints (2 tables)
    - E.g. bnames associated w/ loans must be names of real branches
  - Global Constraints (n tables)
    - E.g., all loans must be carried by at least 1 customer with a savings acct
Key Constraints

Idea: specifies that a relation is a set, not a bag

SQL examples:

1. **Primary Key:**

   ```sql
   CREATE TABLE branch(
       bname CHAR(15) PRIMARY KEY,
       bcity CHAR(20),
       assets INT);
   ``

   or

   ```sql
   CREATE TABLE depositor(
       cname CHAR(15),
       acct_no CHAR(5),
       PRIMARY KEY(cname, acct_no));
   ```

2. **Candidate Keys:**

   ```sql
   CREATE TABLE customer(
       ssn CHAR(9) PRIMARY KEY,
       cname CHAR(15),
       address CHAR(30),
       city CHAR(10),
       UNIQUE (cname, address, city));
   ```

Effect of SQL Key declarations

```
PRIMARY (A1, A2, .., An) or
UNIQUE (A1, A2, ..., An)
```

Insertions: check if any tuple has same values for A1, A2, .., An as any inserted tuple. If found, **reject insertion**

Updates to any of A1, A2, ..., An: treat as insertion of entire tuple

Primary vs Unique (candidate)

1. 1 primary key per table, several unique keys allowed.
2. Only primary key can be referenced by “foreign key” (ref integrity)
3. DBMS may treat primary key differently
   (e.g.: create an index on PK)
Attribute Constraints

- Idea:
  - Attach constraints to values of attributes
  - Enhances types system (e.g.: \(\geq 0\) rather than integer)

- In SQL:

  1. **NOT NULL**
     
     e.g.: CREATE TABLE branch(
           bname CHAR(15) NOT NULL,
           ....
        )
     
     Note: declaring `bname` as primary key also prevents null values

  2. **CHECK**
     
     e.g.: CREATE TABLE depositor(
           ....
           balance int NOT NULL,
           CHECK (balance >= 0),
           ....
        )
     
     affects insertions, updates in affected columns

Attribute Constraints

**Domains:** can associate constraints with DOMAINS rather than attributes

- e.g: instead of: CREATE TABLE depositor(

  ....
  balance INT NOT NULL,
  CHECK (balance >= 0)
  )

One can write:

CREATE DOMAIN bank-balance INT (  
CONSTRAINT not-overdrawn CHECK (value >= 0),  
CONSTRAINT not-null-value CHECK( value NOT NULL));

CREATE TABLE depositor (  
......  
balance  bank-balance,  
)

Advantages?
**Attribute Constraints**

Advantage of associating constraints with domains:

1. can avoid repeating specification of same constraint for multiple columns
2. can name constraints
e.g.: CREATE DOMAIN bank-balance INT (CONSTRAINT not-overdrawn CHECK (value >= 0), CONSTRAINT not-null-value CHECK( value NOT NULL));

allows one to:
1. add or remove:
   ALTER DOMAIN bank-balance
   ADD CONSTRAINT capped CHECK( value <= 10000)
2. report better errors (know which constraint violated)

**Referential Integrity Constraints**

Idea: prevent “dangling tuples” (e.g.: a loan with a bname, Kenmore, when no Kenmore tuple in branch)

Ref Integrity:
ensure that:
foreign key value  \(\rightarrow\) primary key value

(note: don’t need to ensure \(\leftarrow\), i.e., not all branches have to have loans)
Referential Integrity Constraints

CREATE TABLE A (    FOREIGN KEY c REFERENCES B action    .......... )

Action:
1) left blank (deletion/update rejected)
2) ON DELETE SET NULL/ ON UPDATE SET NULL
   sets ti[c] = NULL, tj[c] = NULL
3) ON DELETE CASCADE
   deletes ti, tj
   ON UPDATE CASCADE
   sets ti[c], tj[c] to new key values

Global Constraints

Idea: two kinds
1) single relation (constraints spans multiple columns)
   ◦ E.g.: CHECK (total = svngs + check) declared in the CREATE TABLE

SQL examples:
   All Bkln branches must have assets > 5M

   CREATE TABLE branch (    ...........    bcity CHAR(15),    assets INT,    CHECK (NOT(bcity = 'Bkln') OR assets > 5M))

Affects:
   insertions into branch
   updates of bcity or assets in branch
Global Constraints

2) Multiple relations: every loan has a borrower with a savings account

CHECK (NOT EXISTS (
    SELECT  *
    FROM loan AS L
    WHERE  NOT EXISTS(
        SELECT  *
        FROM borrower B, depositor D, account A
        WHERE B.cname = D.cname AND D.acct_no = A.acct_no AND L.lno = B.lno)
))

Problem: Where to put this constraint? At depositor? Loan? ....

Ans: None of the above:
CREATE ASSERTION loan-constraint
   CHECK(  ..... )

Summary: Integrity Constraints

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>Where declared</th>
<th>Affects...</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Constraints</td>
<td>CREATE TABLE</td>
<td>Insertions, Updates</td>
<td>Moderate</td>
</tr>
<tr>
<td></td>
<td>(PRIMARY KEY, UNIQUE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute Constraints</td>
<td>CREATE TABLE</td>
<td>Insertions, Updates</td>
<td>Cheap</td>
</tr>
<tr>
<td></td>
<td>CREATE DOMAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Not NULL, CHECK)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Referential Integrity | Table Tag
                       | (FOREIGN KEY ....    | 1. Insertions into        | 1,2: like key constraints.  |
|                       | REFERENCES ....)     | referencing rel’n    | Another reason to           |
|                       |                      | 2. Updates of       | index/sort on the primary   |
|                       |                      | referencing rel’n of | keys                        |
|                       |                      | relevant attrs       |                             |
|                       |                      | 3. Deletions from    |                             |
|                       |                      | referenced rel’n     |                             |
|                       |                      | 4. Update of         | 3,4: depends on             |
|                       |                      | referenced rel’n     |                             |
|                       |                      |                     | a. update/delete policy     |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     | b. existence of indexes     |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
|                       |                      |                     |                             |
| Global Constraints    | Table Tag (CHECK)    | 1. For single rel’n  | 1. cheap                    |
|                       | or outside table     | constraint, with     | 2. very expensive            |
|                       | (CREATE ASSERTION)   | insertion, deletion  |                             |
|                       |                      | of relevant attrs    |                             |
|                       |                      | 2. For assertions w/ every db modification | |
Today’s Plan

- SQL (Chapter 3, 4)
  - Views (4.2)
  - Triggers (5.3)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Functions and Procedures (5.2), Authorization (4.6), Ranking (5.5)
  - Return to / Finishing the Relational Algebra
  - E/R Diagrams

SQL Functions

- Function to count number of instructors in a department

```sql
create function dept_count (dept_name varchar(20)) returns integer
begin
    declare d_count integer;
    select count(*) into d_count
    from instructor
    where instructor.dept_name = dept_name
    return d_count;
end
```

- Can use in queries:

```sql
select dept_name, budget
from department
where dept_count (dept_name) > 12
```
**SQL Procedures**

- Same function as a procedure:
  ```sql
  create procedure dept_count_proc (in dept_name varchar(20), out d_count integer)
  begin
    select count(*) into d_count
    from instructor
    where instructor.dept_name = dept_count_proc.dept_name
  end
  ```

- But use differently:
  ```sql
  declare d_count integer;
  call dept_count_proc( 'Physics', d_count);
  ```

**HOWEVER:** Syntax can be wildly different across different systems

- Was put in place by DBMS systems before standardization
- Hard to change once customers are already using
- This example **NOT** valid in your version of postgresql

---

**Recursion in SQL**

- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

  ```sql
  with recursive rec_prereq(course_id, prereq_id) as ( 
    select course_id, prereq_id 
    from prereq 
    union 
    select rec_prereq.course_id, prereq.prereq_id, 
    from rec_prereq, prereq 
    where rec_prereq.prereq_id = prereq.course_id 
  ) 
  select * 
  from rec_prereq;
  ```

- Makes SQL Turing Complete (i.e., you can write any program in SQL)

  But: Just because you can, doesn’t mean you should
Ranking

- Ranking is done in conjunction with an order by specification.

- Consider: `student_grades(ID, GPA)`

- Find the rank of each student.
  
  ```sql
  select ID, rank() over (order by GPA desc) as s_rank
  from student_grades
  order by s_rank
  ```

- Equivalent to:
  
  ```sql
  select ID, (1 + (select count(*)
  from student_grades B
  where B.GPA > A.GPA)) as s_rank
  from student_grades A
  order by s_rank;
  ```

Authorization/Security

- GRANT and REVOKE keywords
  - grant select on instructor to U₁, U₂, U₃
  - revoke select on branch from U₁, U₂, U₃

- Can provide select, insert, update, delete privileges
- Can also create “Roles” and do security at the level of roles
- Some databases support doing this at the level of individual “tuples”
  - PostgreSQL: [https://www.postgresql.org/docs/10/ddl-rowsecurity.html](https://www.postgresql.org/docs/10/ddl-rowsecurity.html)
Transactions

A transaction is a sequence of queries and update statements executed as a single unit

- Transactions are started implicitly and terminated by one of
  - commit work: makes all updates of the transaction permanent in the database
  - rollback work: undoes all updates performed by the transaction.

Motivating example

- Transfer of money from one account to another involves two steps:
  - deduct from one account and credit to another
- If one steps succeeds and the other fails, database is in an inconsistent state
- Therefore, either both steps should succeed or neither should

If any step of a transaction fails, all work done by the transaction can be undone by rollback work.

Rollback of incomplete transactions is done automatically, in case of system failures

Transactions (Cont.)

- In most database systems, each SQL statement that executes successfully is automatically committed.
  - Each transaction would then consist of only a single statement
  - Automatic commit can usually be turned off, allowing multi-statement transactions, but how to do so depends on the database system
  - Another option in SQL:1999: enclose statements within
    - `begin atomic
      ...
      end`
Relational Data Model

Introduced by Ted Codd (late 60’s – early 70’s)

- Before = “Network Data Model” (Cobol as DDL, DML)
- Very contentious: Database Wars (Charlie Bachman vs. Ted Codd)

Relational data model contributes:

1. Separation of logical, physical data models (data independence)
2. Declarative query languages
3. Formal semantics
4. Query optimization (key to commercial success)

1st prototypes:

- Ingres → CA
- Postgres → Illustra → Informix → IBM
- System R → Oracle, DB2
**Key Abstraction: Relation**

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Terms:

- Tables (aka: Relations)

**Why called Relations?**

*Closely correspond to mathematical concept of a relation*

**Relations**

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Considered equivalent to…

\[
\{ \text{(Downtown, A-101, 500),} \\
\text{(Brighton, A-201, 900),} \\
\text{(Brighton, A-217, 500)} \}\]

*Relational database semantics defined in terms of mathematical relations*
Relations

Account =

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Considered equivalent to...

\[
\{(\text{Downtown}, A-101, 500), \\
(\text{Brighton}, A-201, 900), \\
(\text{Brighton}, A-217, 500)\}
\]

Terms:

- Tables (aka: Relations)
- Rows (aka: tuples)
- Columns (aka: attributes)
- Schema (e.g.: Acct_Schema = (bname, acct_no, balance))

Definitions

Relation Schema (or Schema)

A list of attributes and their domains (elided here for space)

E.g. account(account-number, branch-name, balance)

Programming language equivalent: A variable (e.g. x)

Relation Instance

A particular instantiation of a relation with actual values

Will change with time

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Programming language equivalent: Value of a variable
Definitions

Domains of an attribute/column
The set of permitted values
e.g., bname must be String, balance must be a positive real number
We typically assume domains are atomic, i.e., the values are treated as indivisible (specifically: you can’t store lists or arrays in them)

Null value
Used if attribute value is:
◦ unknown (e.g., don’t know address of a customer)
◦ inapplicable (e.g., “spouse name” attribute for a customer)
◦ withheld/hidden
Different interpretations all captured by a single concept – leads to major headaches and problems

Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building,

room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(Id, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)
Keys

- Let $K \subseteq R$ (R is a set of attributes)
- $K$ is a superkey of $R$ if values for $K$ are sufficient to identify a unique tuple of any possible relation $r(R)$
  - Example: \{ID\} and \{ID, name\} are both superkeys of instructor.
- Superkey $K$ is a candidate key if $K$ is minimal (i.e., no subset of it is a superkey)
  - Example: \{ID\} is a candidate key for Instructor
- One of the candidate keys is selected to be the primary key
  - Typically one that is small and immutable (doesn’t change often)
  - Chosen by app/user
- Primary key typically highlighted (e.g., underlined)

Tables in a University Database

takes(ID, course_id, sec_id, semester, year, grade)

Is ID, course_id a superkey?
- No. May repeat:
  - (“1011049”, “CMSC424”, “102”, “Fall”, 2015, null)

What about ID, course_id, sec_id?
- May repeat:
  - (“1011049”, “CMSC424”, “101”, “Fall”, 2015, null)

What about ID, course_id, sec_id, semester?
- Still no:
Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Keys

- **Foreign key**: Primary key of a relation that appears in another relation
  - {ID} from student appears in takes, advisor
  - student called referenced relation
  - takes is the referencing relation
  - Typically shown by an arrow from referencing to referenced

- **Foreign key constraint**: the tuple corresponding to that primary key must exist
  - Imagine:
    - Tuple: (‘student101’, ‘CMSC424’) in takes
    - But no tuple corresponding to ‘student101’ in student
  - Also called referential integrity constraint
Examples

- Married(person1_ssn, person2_ssn, date_married, date_divorced)
  - Married(person1_ssn, person2_ssn, date_married, date_divorced)

- Account(cust_ssn, account_number, cust_name, balance, cust_address)
  - If a single account per customer, then: cust_ssn
  - Else: (cust_ssn, account_number)
    - Not a good schema because it requires repeating information

- RA(student_id, project_id, supervisor_id, appt_time, appt_start_date, appt_end_date)
  - RA(student_id, project_id, supervisor_id, appt_time, appt_start_date, appt_end_date)
  - Could be smaller if there are some restrictions – requires some domain knowledge of the data being stored

- Person(Name, DOB, Born, Education, Religion, ...)
  - Information typically found on Wikipedia Pages
  - Unclear what could be a primary key here: you could in theory have two people who match on all of those