W4111 – Introduction to Databases
Section 003, V03, Fall 2024
Lecture 6: ER(5), Relational(5), SQL(5)

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

*W4111 – Introduction to Databases
Section 003, V03, Fall 2024
Lecture 6: ER(5), Relational(5), SQL(5)*

We will start in a few minutes.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Today's Contents

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

- Introduction and course updates.
  - Midterm
  - Questions
- Codd's 12 Rules – Metadata.
- Relational model and algebra (5).
- SQL (5).
- Some complete examples of modeling → implementation.

Columbia Engineering
The Fu Foundation School of Engineering and Applied Science

Introduction
and
course updates

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Course Updates

- Midterm is 18-OCT-2024
  - CVN students will take online in a proctored environment.
    - Students have flexibility on day and time but must complete by 23-OCT at 11:59.
    - We will provide details and instructions on Ed.
  - Students registered with their school's disability services will coordinate and follow the services instructions for taking the test.
  - All other students will take the exam in the classroom 309 Havemeyer from 10:10 AM until 11:30 AM. We will be strict with enforcing start and stop times.
  - A continuation lecture will be from 11:45 AM to 12:40 PM.
  - You may use one 8.5" by 11" double side "cheat sheet" that you create.
    You can choose the content that you put on your sheet.
  - <u>The exam will be difficult. Will "curve" the scores. The exam will be long ➔ manage your time.</u>
- You are are responsible for:
  - Any topic covered in a lecture, even if covered verbally, answering a question, etc.
  - ANY material from ANY slide in the slide decks for lectures 1 through 6.
  - Any material in the slides for Ch1, Ch2, Ch3, Ch4, Ch5 (except for JDBC).
  - Ch6: You are responsible for understanding the concepts. The only modeling notation language for which you are responsible is Crow's Foot Notation.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Codd's 12 Rules
Metadata

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Codd's 12 Rules

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Columbia ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Codd's 12 Rules

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *instructor* (
                                         ID   **char**(5),
        name   **varchar**(20)**,**
        dept_name **varchar**(20),
        salary   **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***

- Data dictionary contains metadata (i.e., data about data)

  - Database schema

  - Integrity constraints

    - Primary key (ID uniquely identifies instructors)

  - Authorization

    - Who can access what

Metadata and Catalog

- 'Metadata is "data that provides information about other data". In other words, it is "data about data". Many distinct types of metadata exist, including descriptive metadata, structural metadata, administrative metadata, reference metadata and statistical metadata.' (https://en.wikipedia.org/wiki/Metadata)

- "The database catalog of a database instance consists of metadata in which definitions of database objects such as base tables, views (virtual tables), synonyms, value ranges, indexes, users, and user groups are stored. ... ...

  The SQL standard specifies a uniform means to access the catalog, called the INFORMATION_SCHEMA, but not all databases follow this ..." (https://en.wikipedia.org/wiki/Database_catalog)

- Codd's Rule 4: Dynamic online catalog based on the relational model:
  - The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

Columbia Engineering
The Fu Foundation School of Engineering and Applied Science
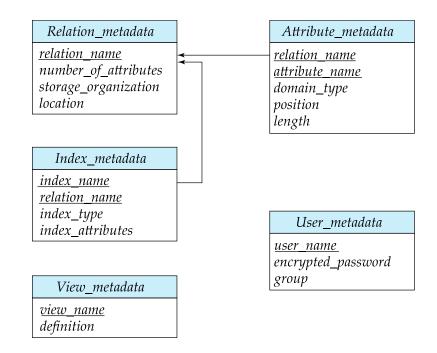
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

- Information about relations
    - names of relations
    - names, types and lengths of attributes of each relation
    - names and definitions of views
    - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
    - number of tuples in each relation
- Physical file organization information
    - How relation is stored (sequential/hash/…)
    - Physical location of relation
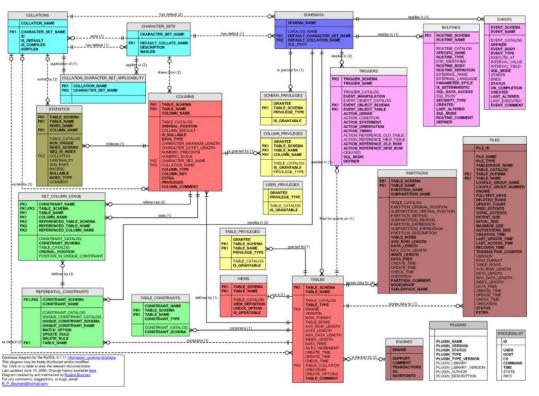- Information about indices (Chapter 14)

Relational Representation of System Metadata

- Relational representation on disk

- Specialized data structures designed for efficient access, in memory

Relation_metadata

_relation_name_
_number_of_attributes_
_storage_organization_
location

Attribute_metadata

_relation_name_
_attribute_name_
_domain_type_
position
length

Index_metadata

_index_name_
_relation_name_
_index_type_
_index_attributes_

User_metadata

_user_name_
_encrypted_password_
group

View_metadata

_view_name_
definition

MySQL Catalog (Information_Schema)

Some of the MySQL Information Schema Tables:

- 'ADMINISTRABLE_ROLE_AUTHORIZATIONS'
- 'APPLICABLE_ROLES'
- 'CHARACTER_SETS'
- 'CHECK_CONSTRAINTS'
- 'COLUMN_PRIVILEGES'
- 'COLUMN_STATISTICS'
- 'COLUMNS'
- 'ENABLED_ROLES'
- 'ENGINES'
- 'EVENTS'
- 'FILES'
- 'KEY_COLUMN_USAGE'
- 'PARAMETERS'
- 'REFERENTIAL_CONSTRAINTS'
- 'RESOURCE_GROUPS'
- 'ROLE_COLUMN_GRANTS'
- 'ROLE_ROUTINE_GRANTS'
- 'ROLE_TABLE_GRANTS'
- 'ROUTINES'
- 'SCHEMA_PRIVILEGES'
- 'STATISTICS'
- 'TABLE_CONSTRAINTS'
- 'TABLE_PRIVILEGES'
- 'TABLES'
- 'TABLESPACES'
- 'TRIGGERS'
- 'USER_PRIVILEGES'
- 'VIEW_ROUTINE_USAGE'
- 'VIEW_TABLE_USAGE'
- 'VIEWS'

- CREATE and ALTER statements modify the data.
- DBMS reads information:
  - Parsing
  - Optimizer
  - etc.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Relational Algebra

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Semi-Join

Semi-join is a type of join that is applied to relations to join them based on the related columns. When semi-join is applied, it returns the rows from one table for which there are matching records in another related table.

course ⋈ prereq
Execution time: 1 ms

course.course_id	course.title	course.dept_name	course.credits	prereq.prereq_id
'BIO-301'	'Genetics'	'Biology'	4	'BIO-101'
'BIO-399'	'Computational Biology'	'Biology'	3	'BIO-101'
'CS-190'	'Game Design'	'Comp. Sci.'	4	'CS-101'
'CS-315'	'Robotics'	'Comp. Sci.'	3	'CS-101'
'CS-319'	'Image Processing'	'Comp. Sci.'	3	'CS-101'
'CS-347'	'Database System Concepts'	'Comp. Sci.'	3	'CS-101'
'EE-181'	'Intro. to Digital Systems'	'Elec. Eng.'	3	'PHY-101'

course ⋉ prereq
Execution time: 1 ms

course.course_id	course.title	course.dept_name	course.credits
'BIO-301'	'Genetics'	'Biology'	4
'BIO-399'	'Computational Biology'	'Biology'	3
'CS-190'	'Game Design'	'Comp. Sci.'	4
'CS-315'	'Robotics'	'Comp. Sci.'	3
'CS-319'	'Image Processing'	'Comp. Sci.'	3
'CS-347'	'Database System Concepts'	'Comp. Sci.'	3
'EE-181'	'Intro. to Digital Systems'	'Elec. Eng.'	3

Columbia ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Division

Division [edit]

The division (÷) is a binary operation that is written as $R \div S$. Division is not implemented directly in SQL. The result consists of the restrictions of tuples in R to the attribute names unique to R, i.e., in the header of R but not in the header of S, for which it holds that all their combinations with tuples in S are present in R.

Example [edit]

Relax Example

X = π course_id←course_id (σ dept_name='History' (course))

Y = π ID←ID, course_id←course_id (student ⋈ takes)

Y ÷ X

Completed	
Student	**Task**
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject
Task
Database1
Database2

Completed
÷
DBProject

Student
Fred
Sarah

If *DBProject* contains all the tasks of the Database project, then the result of the division above contains exactly the students who have completed both of the tasks in the Database project. More formally the semantics of the division is defined as follows:

$$R \div S = \{\, t[a_1,...,a_n] : t \in R \,\wedge\, \forall s \in S\, (\, (t[a_1,...,a_n] \cup s) \in R)\,\}$$ (6)

where $\{a_1,...,a_n\}$ is the set of attribute names unique to R and $t[a_1,...,a_n]$ is the restriction of t to this set. It is usually required that the attribute names in the header of S are a subset of those of R because otherwise the result of the operation will always be empty.

Indexes

© Donald F. Ferguson, 2024

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Index Creation

- Many queries reference only a small proportion of the records in a table.

- It is inefficient for the system to read every record to find a record with particular value

- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.

- We create an index with the **create index** command

    create index <name> **on** <relation-name> (attribute);

Index Creation Example

- **create table** *student*
  (*ID* **varchar** (5),
  name **varchar** (20) **not null**,
  dept_name **varchar** (20),
  tot_cred **numeric** (3,0) **default** 0,
  primary key (*ID*))

- **create index** *studentID_index* **on** *student*(*ID*)

- The query:

      select *
       from *student*
       where *ID* = '12345'

  can be executed by using the index to find the required record, without
  looking at all records of *student*

DFF:

- An index on (column1, column2, column3)
- Is also an index on (column1) and (column1, column2)
- But not (column2), (column3), (column2, column3).
- We will see "why" later in the semester.

Functions, Procedures, Triggers

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Some Concepts

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Functions and Procedures

- Functions and procedures allow "business logic" to be stored in the database and executed from SQL statements.

- These can be defined either by the procedural component of SQL or by an external programming language such as Java, C, or C++.

- The syntax we present here is defined by the SQL standard.
  - Most databases implement nonstandard versions of this syntax.

Note:
- The programming language, runtime and tools for functions, procedures and triggers are not easy to use.
- My view is that calling external functions is an anti-pattern (bad idea).
  - External code degrades the reliability, security and performance of the database.
  - Databases are often mission critical and the heart of environments.

Language Constructs for Procedures & Functions

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
  - Warning: most database systems implement their own variant of the standard syntax below.
- Compound statement: **begin** … **end,** {}
  - May contain multiple SQL statements between **begin** and **end**.
  - Local variables can be declared within a compound statements
- While and repeat statements:
  - **while** boolean expression **do**
          sequence of statements ;
      end while

  - **repeat**
         sequence of statements ;
      until boolean expression
      end repeat

(Core) Language Constructs (Cont.)

- **For** loop
  - Permits iteration over all results of a query
- Example: Find the budget of all departments

```
declare n  integer default 0;
for r  as
     select budget from department
          where dept_name = 'Music'
do
              set n = n + r.budget
end for
```

Note:
- There are various other looping constructs.

(Core) Language Constructs – if-then-else

- Conditional statements (**if-then-else**)

    if *boolean expression*
        then *statement or compound statement*
      elseif *boolean expression*
        then *statement or compound statement*
    else *statement or compound statement*
      end if

Note:
- We will not spend a lot of time writing functions, procedures, or triggers.
- The language and development environment are not easy to use.

Functions

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Declaring SQL Functions

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

  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

- The function *dept_*count can be used to find the department names and budget of all departments with more that 12 instructors.

  select *dept_name, budget*
  from *department*
  where *dept_*count (*dept_name*) > 12

Table Functions

- The SQL standard supports functions that can return tables as results; such functions are called **table functions**

- Example: Return all instructors in a given department

  create function *instructor_of* (*dept_name* **char**(20))

          returns table (

        ID **varchar**(5),

           name **varchar**(20),

        dept_name **varchar**(20),

           salary **numeric**(8,2))

      return table

          (**select** *ID, name, dept_name, salary*

          from *instructor*

          where *instructor.dept_name = instructor_of.dept_name*)

- Usage

      select *

      from table (*instructor_of* ('Music'))

Procedures

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

SQL Procedures

- The *dept_count* function could instead be written as procedure:

  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

- The keywords **in** and **out** are parameters that are expected to have values assigned to them and parameters whose values are set in the procedure in order to return results.

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

          declare *d_count* **integer**;
          call *dept_count_proc*('Physics', *d_count*);

SQL Procedures (Cont.)

- Procedures and functions can be invoked also from dynamic SQL
- SQL allows more than one procedure of the so long as the number of arguments of the procedures with the same name is different.
- The name, along with the number of arguments, is used to identify the procedure.

Triggers

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.

- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.

- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Syntax illustrated here may not work exactly on your database system; check the system manuals

X on T

    before insert

    before update

    before delete

    after insert

    after update

    after delete

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - For example, **after update of** *takes* **on** *grade*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** **:** for deletes and updates
  - **referencing new row as** **:** for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. For example, convert blank grades to null.

> **create trigger** *setnull_trigger* **before update of** *takes*
> **referencing new row as** *nrow*
> **for each row**
>   **when (***nrow.grade* = ' ')
> **begin atomic**
>   **set** *nrow.grade* = **null**;
> **end;**

Trigger to Maintain credits_earned value

- **create trigger** *credits_earned* **after update of** *takes* **on** (*grade*)
  referencing new row as *nrow*
  referencing old row as *orow*
  for each row
  when *nrow.grade* <> 'F' **and** *nrow.grade* **is not null**
     and (*orow.grade* = 'F' **or** *orow.grade* **is null**)
  begin atomic
     update *student*
     set *tot_cred*= *tot_cred* +
        (**select** *credits*
         from *course*
         where *course.course_id*= *nrow.course_id*)
      where *student.id* = *nrow.id*;
  end;

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction

    - Use **for each statement** instead of **for each row**

    - Use **referencing old table** or **referencing new table** to refer to temporary tables (called *transition tables*) containing the affected rows

    - Can be more efficient when dealing with SQL statements that update a large number of rows

When Not To Use Triggers

- Triggers were used earlier for tasks such as
  - Maintaining summary data (e.g., total salary of each department)
  - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

When Not To Use Triggers (Cont.)

- Risk of unintended execution of triggers, for example, when
  - Loading data from a backup copy
  - Replicating updates at a remote site
  - Trigger execution can be disabled before such actions.
- Other risks with triggers:
  - Error leading to failure of critical transactions that set off the trigger
  - Cascading execution

Summary

© Donald F. Ferguson,

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Comparison

comparing triggers, functions, and procedures

	triggers	functions	stored procedures
change data	yes	no	yes
return value	never	always	sometimes
how they are called	reaction	in a statement	exec

lynda.com

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Comparison – Some Details

A *trigger* has capabilities like a procedure, except …

- You do not call it. The DB engine calls it before or after an INSERT, UPDATE, DELETE.
- The inputs are the list of incoming new, modified rows.
- The outputs are the modified versions of the new or modified rows.

Sr.No.	User Defined Function	Stored Procedure
1	Function must return a value.	Stored Procedure may or not return values.
2	Will allow only Select statements, it will not allow us to use DML statements.	Can have select statements as well as DML statements such as insert, update, delete and so on
3	It will allow only input parameters, doesn't support output parameters.	It can have both input and output parameters.
4	It will not allow us to use try-catch blocks.	For exception handling we can use try catch blocks.
5	Transactions are not allowed within functions.	Can use transactions within Stored Procedures.
6	We can use only table variables, it will not allow using temporary tables.	Can use both table variables as well as temporary table in it.
7	Stored Procedures can't be called from a function.	Stored Procedures can call functions.
8	Functions can be called from a select statement.	Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure.
9	A UDF can be used in join clause as a result set.	Procedures can't be used in Join clause

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Authorization

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Security Concepts (Terms from Wikipedia)

- Definitions:
  - "A (digital) identity is information on an entity used by computer systems to represent an external agent. That agent may be a person, organization, application, or device."
  - "Authentication is the act of proving an assertion, such as the identity of a computer system user. In contrast with identification, the act of indicating a person or thing's identity, authentication is the process of verifying that identity."
  - "Authorization is the function of specifying access rights/privileges to resources, … More formally, "to authorize" is to define an access policy. … During operation, the system uses the access control rules to decide whether access requests from (authenticated) consumers shall be approved (granted) or disapproved.
  - "Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles, and through those role assignments acquire the permissions needed to perform particular system functions."
  - "In computing, privilege is defined as the delegation of authority to perform security-relevant functions on a computer system. A privilege allows a user to perform an action with security consequences. Examples of various privileges include the ability to create a new user, install software, or change kernel functions."

- SQL and relational database management systems implementing security by:
  - Creating identities and authentication policies.
  - Creating roles and assigning identities to roles.
  - Granting and revoking privileges to/from roles and identities.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Authorization

- We may assign a user several forms of authorizations on parts of the database.

  - **Read** - allows reading, but not modification of data.

  - **Insert** - allows insertion of new data, but not modification of existing data.

  - **Update** - allows modification, but not deletion of data.

  - **Delete** - allows deletion of data.

- Each of these types of authorizations is called a **privilege**. We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.

Authorization (Cont.)

- Forms of authorization to modify the database schema
    - **Index** - allows creation and deletion of indices.
    - **Resources** - allows creation of new relations.
    - **Alteration** - allows addition or deletion of attributes in a relation.
    - **Drop** - allows deletion of relations.

Authorization Specification in SQL

- The **grant** statement is used to confer authorization

    grant <privilege list> **on** <relation or view > **to** <user list>

- <user list> is:

    - a user-id

    - **public**, which allows all valid users the privilege granted

    - A role (more on this later)

- Example:

    - **grant select on** *department* **to** Amit, Satoshi

- Granting a privilege on a view does not imply granting any privileges on the underlying relations.

- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view

  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

    grant select on *instructor* **to** U_1, U_2, U_3

- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list> **on** <relation or view> **from** <user list>

- Example:

  revoke select on *student* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.

- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.

- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.

- All privileges that depend on the privilege being revoked are also revoked.

Roles

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.

- To create a role we use:

  create a role <name>

- Example:
  - **create role** instructor

- Once a role is created we can assign "users" to the role using:
  - **grant** <role> **to** <users>

Roles Example

- **create role** instructor;

- **grant** *instructor* **to** Amit**;**

- Privileges can be granted to roles:

  - **grant select on** *takes* **to** *instructor*;

- Roles can be granted to users, as well as to other roles

  - **create role** *teaching_assistant*

  - **grant** *teaching_assistant* **to** *instructor*;

    - *Instructor* inherits all privileges of *teaching_assistant*

- Chain of roles

  - **create role** *dean*;

  - **grant** *instructor* **to** *dean*;

  - **grant** *dean* **to** Satoshi;

Authorization on Views

- **create view** *geo_instructor* **as**
  (**select** *
  from *instructor*
  where *dept_name* = 'Geology');

- **grant select on** *geo_instructor* **to** *geo_staff*

- Suppose that a *geo_staff* member issues

  - **select** *
    from *geo_instructor*;

- What if

  - *geo_staff* does not have permissions on *instructor?*

  - Creator of view did not have some permissions on *instructor?*

Other Authorization Features

- **references** privilege to create foreign key
  - **grant reference** (*dept_name*) **on** *department* **to** Mariano;
  - Why is this required?
- transfer of privileges
  - **grant select on** *department* **to** Amit **with grant option**;
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;
  - And more!

Note:
- Like in many other cases, SQL DBMS have product specific variations.

Switch to notebook.

Some Advanced SQL

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

WITH Clause
Common Table Expressions

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

With Clause

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.

- Find all departments with the maximum budget

  with *max_budget* (*value*) **as**
     (**select max**(*budget*)
      from *department*)
  select *department.name*
  from *department*, *max_budget*
  where *department.budget* = *max_budget.value*;

Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

    with *dept _total* (*dept_name*, *value*) **as**
        (**select** *dept_name*, **sum**(*salary*)
         from *instructor*
         group by *dept_name*),
    dept_total_avg(*value*) **as**
        (**select avg**(*value*)
         from *dept_total*)
    select *dept_name*
    from *dept_total*, *dept_total_avg*
    where *dept_total.value* > *dept_total_avg.value*;

Common Table Expressions

- CTE: (https://www.essentialsql.com/introduction-common-table-expressions-ctes/)
  - "The Common Table Expressions or CTE's for short are used within SQL (…) to simplify complex joins and subqueries, …"
  - "A CTE (Common Table Expression) defines a temporary result set which you can then use in a SELECT statement. It becomes a convenient way to manage complicated queries."
  - There are two types of CTEs:
    - Non-recursive.
    - Recursive (note, recursive SQL weirds me out).
  - The benefits are clarity and improved quality through incremental development.

- Basic syntax by example.
  - There may be several CTEs.
  - A CTE can reference other CTEs.

```
With Employee_CTE (EmployeeNumber, Title)
AS
(
SELECT  NationalIDNumber,
        JobTitle
FROM    HumanResources.Employee
)
SELECT  EmployeeNumber,
        Title
FROM    Employee_CTE
```

CTE
(Common Table Expression)

Query Using CTE

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Common Table Expressions – Example

```sql
WITH career_batting (playerid, h, ab, bb, hr, rbi) AS
(
    SELECT playerid, sum(h) AS h, sum(ab) AS ab, sum(bb) AS bb,
        sum(hr) AS hr, sum(rbi) AS rbi
    FROM batting GROUP BY playerid
),
career_pitching (playerid, w, l, ipouts, er) AS
(
    SELECT playerid, sum(w) AS w, sum(l) AS l, sum(ipouts) as ipouts, sum(er) AS er
    FROM pitching GROUP BY playerid
),
career_summary (playerid, h, ab, bb, hr, rbi, bavg, obp, w, l, ipouts, er, era) AS
(
    SELECT career_batting.playerid, h, ab, bb, hr, rbi,
        IF(ab<500, NULL, round(h/ab, 3)) AS bavg,
        IF(ab<500, NULL, round((h+bb)/(ab + bb), 3)) AS obp,
        w, l, ipouts, er, round((er/(ipouts/3))*9, 3) AS era
    FROM
        career_batting join career_pitching using(playerid)
)
SELECT
    playerid, nameLast, nameFirst, career_summary.*
FROM
    people JOIN career_summary using(playerid);
```

- Producing a career summary requires several tasks:
    1. Computing batting totals
    2. Computing pitching totals
    3. Applying formulas to produce derived averages and metrics.
    4. Joining (1), (2) and (3) into a career summary.
    5. Joining with people to bring in personal information.

- Producing the table is possible with a single SELECT statement, but … …
    - Developing incrementally one simpler query at a time is easier.
    - The resulting query is easier to understand and maintain.

© Donald F. Ferguson, 2020

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Common Table Expressions – Example

```sql
WITH career_batting (playerid, h, ab, bb, hr, rbi) AS
(
    SELECT playerid, sum(h) AS h, sum(ab) AS ab, sum(bb) AS bb,
        sum(hr) AS hr, sum(rbi) AS rbi
    FROM batting GROUP BY playerid
),
career_p
(
    SELE
        FR
),
career_s
(
    SELE
        IF(
        IF(
        w, l, ipouts, sr, round((sr(ipouts * 9), s) AS era
    FROM
        career_batting join career_pitching using(playerid)
)
SELECT
    playerid, nameLast, nameFirst, career_summary.*
FROM
    people JOIN career_summary using(playerid);
```

Overlay box:
- I find it difficult to impossible to write complex queries without using CTEs.
- I think about incremental steps to produce the result.
- Write and test the CTE by adding one step at a time.
- Please, please, please:
  - Use CTEs.
  - Use DataGrip

- Producing a career summary requires several tasks:
  1. Computing batting totals

  ... (obscured) ... ls

  ... (obscured) ... duce
  ... (obscured) ... etrics.

  ... (obscured) ... to

  ... (obscured) ... ring in

  ... (obscured) ... ssible with a single SELECT statement, but ... ...
    – Developing incrementally one simpler query at a time is easier.
    – The resulting query is easier to understand and maintain.

Columbia | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Recursive Queries

© Donald F. Ferguson, 2024

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Recursion in SQL

- SQL:1999 permits recursive view definition

- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course
  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_rereq*, *prereq*
      where *rec_prereq.prereq_id = prereq.course_id*
)
  select *

  from *rec_prereq*;

  This example view, *rec_prereq,* is called the *transitive closure* of the *prereq* relation
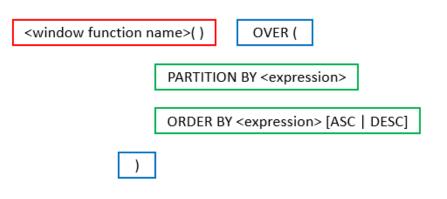
The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.

  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *prereq* with itself

    - This can give only a fixed number of levels of managers

    - Given a fixed non-recursive query, we can construct a database with a greater number of levels of prerequisites on which the query will not work

    - Alternative: write a procedure to iterate as many times as required

      - See procedure *findAllPrereqs* in book

- Since SQL is not Turing complete, there are many computations it cannot perform.
- Recursion enables some scenarios, but recursion in general is hard to understand.
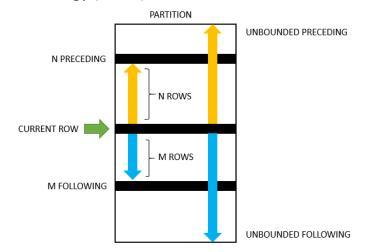- Switch to notebook

Advanced Aggregation
Window Functions

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Advanced Aggregates and Window Functions

- The aggregation functions in SQL are powerful.

- There are some scenarios that are very difficult. SQL and databases add support for more advanced capabilities:

  – Ranking

  – Windows

  – Pivot, Slice and Dice, but we will cover these with a different technology (OLAP) later in the semester.

  – This is powerful, a little complex and requires trial and error.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

```sql
SELECT
    country,
    YEAR(orderDate) AS year,
    ROUND(SUM(priceEach * quantityOrdered), 2) AS annual_revenue,
    ROUND(
        (SUM(priceEach * quantityOrdered) -
        LAG(SUM(priceEach * quantityOrdered)) OVER (PARTITION BY country ORDER BY YEAR(orderDate))) /
        LAG(SUM(priceEach * quantityOrdered)) OVER (PARTITION BY country ORDER BY YEAR(orderDate)) * 100, 2
    ) AS yoy_growth_percentage
FROM
    customers
    JOIN orders ON customers.customerNumber = orders.customerNumber
    JOIN orderdetails ON orders.orderNumber = orderdetails.orderNumber
GROUP BY
    country, year
ORDER BY
    country, year;
```
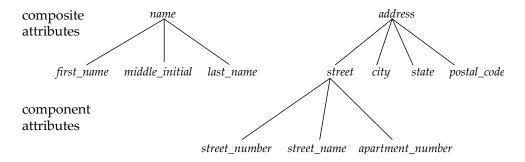
ER Modeling and Realization

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Some Concepts

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Complex Attributes

- Attribute types:
  - **Simple** and **composite** attributes.
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth
- **Domain** – the set of permitted values for each attribute

Composite Attributes

- Composite attributes allow us to divided attributes into subparts (other attributes).

composite
attributes

name

first_name　*middle_initial*　*last_name*

address

street　*city*　*state*　*postal_code*

component
attributes

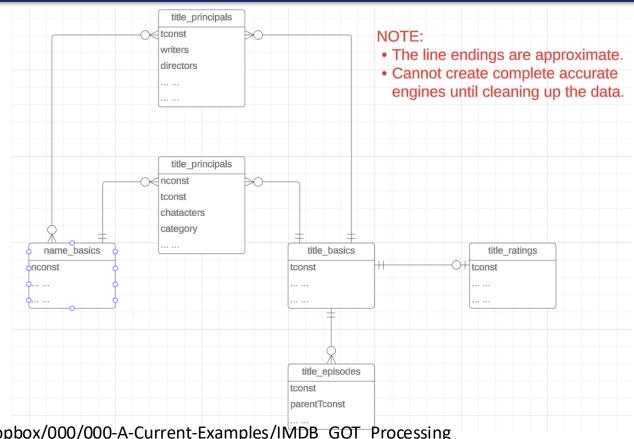*street_number*　*street_name*　*apartment_number*

Entity Attributes

- The relational model and well-designed SQL schema have *atomic attributes.*

- There are several ways to think about attributes:

  - Simple (Atomic) vs Composite

  - Single Valued versus Multi-valued

  - Derived or Not Derived

- And, there can be combinations, for example a Composite, Multi-Value Attribute. Phone number is an example:

  - A phone number is a composite (+1, 914-555-1212)

  - A customer may have several: work, home, mobile, … …

- Examining *name_basics* from the IMDB dataset is interesting.

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

IMDB Free Data

- [IMDB Free Dataset](#) is a set of TSV files:
  - Title Basics
  - Title AKAS
  - Title Crew
  - Title Episodes
  - Title Principals
  - Title Ratings
  - Name Basics
- The data needs a lot of "clean up."

See /Users/donald.ferguson/Dropbox/000/000-A-Current-Examples/IMDB_GOT_Processing

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Example from IMDB

- Consider name_basics

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
1	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0050419,tt0031983,tt0072308,tt0053137
2	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0071877,tt0117057,tt0037382,tt0038355
3	nm0000003	Brigitte Bardot	1934	<null>	actress,soundtrack,music_department	tt0049189,tt0056404,tt0057345,tt0054452
4	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0077975,tt0072562,tt0080455,tt0078723
5	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0050986,tt0060827,tt0069467,tt0050976
6	nm0000006	Ingrid Bergman	1915	1982	actress,soundtrack,producer	tt0077711,tt0038109,tt0034583,tt0036855
7	nm0000007	Humphrey Bogart	1899	1957	actor,soundtrack,producer	tt0043265,tt0034583,tt0042593,tt0037382
8	nm0000008	Marlon Brando	1924	2004	actor,soundtrack,director	tt0078788,tt0068646,tt0070849,tt0047296
9	nm0000009	Richard Burton	1925	1984	actor,soundtrack,producer	tt0061184,tt0087803,tt0057877,tt0059749
10	nm0000010	James Cagney	1899	1986	actor,soundtrack,director	tt0029870,tt0035575,tt0042041,tt0055256

- There
  - Is one composite attribute, primaryName.
  - Are two multivalued attributes: primaryProfession, knownForTitles
  - knownForTitles is also tricky, which we will see.
  - Names are also a little tricky

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

Worked Example

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Worked Example

- Person, Student, Faculty, Course, Section entity types.

- Student and Faculty specialize Person.

- Person, Student, Faculty:
  - Overlapping
  - Complete

- Functions and encapsulation:
  - Hide base tables and protect with authorization rules.
  - get_next_uni() function.
  - Use a trigger to make the UNI *idempotent.*
  - Stored procedures for *insert, update, delete.*

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science