Course Introduction

COSC 304 – Introduction to Database Systems

Introductions – Dr. Ramon Lawrence

Professor, Computer Science, MDS Program Director- Okanagan

Research area: database systems, Internet of Things, software development

Teaching experience:

- 2020 Killam Teaching Prize (top teaching award at UBCO – one per year)
- 2017 UBCO Teaching Excellence Award Winner (two per year)
- 9-time member of teaching honour roll (top 10% instructors)

Industry experience: GE Big Data, UnityJDBC company/consulting

Note: May address me as "Dr. Lawrence", "Professor", or "Ramon" (pronounced RAY-MUN).

The Essence of the Course

The overall goal of this course is for you to:

Create, query, and program with databases to develop applications, web sites, and perform data analysis.

Course covers database techniques and software including relational and NoSQL databases, SQL, JSON, and XML.

This is an important, ***required*** course as almost all systems require a database. Developers and data analysts must have fluency in SQL to process data, and the ability to integrate databases into their development.

My Course Goals

1) Provide the information in an effective way for learning.
2) Inspire and motivate students to learn and appreciate benefits of the course.
3) Strive for ***all*** students to understand the material and excel.
4) Be available for questions during scheduled times, office hours, and at other times as needed.
5) Teach students how to be a sophisticated database user (by understanding SQL), a database application programmer, and a database designer.

Course Objectives

1) Understand the use case for databases and the relational model for data storage.
2) Fluency in SQL including SQL DDL (`CREATE`, `DROP`, `INSERT`, `UPDATE`, `DELETE`) and SQL queries using `SELECT`.
3) Read existing database designs, design new databases using ER/UML modeling, and convert to the relational model.
4) Construct programs that access a database to read data, perform analysis, and output results.
5) Exposure to database technologies like NoSQL, JSON, XML, and cloud databases.

Academic Dishonesty

Cheating is strictly prohibited and is taken very seriously by UBC.

A guideline to what constitutes cheating:

- Labs
  - Submitting code produced by others.
  - Working in groups to solve questions and/or comparing answers to questions once they have been solved (except for group assignments).
  - Discussing HOW to solve a particular question instead of WHAT the question involves.
- Exams
  - Only materials permitted by instructor should be used in an exam.

Academic dishonesty may result in a "F" for the course and further actions by the Dean's office.

How to Excel in This Course

Attend ***every*** class:

- Read notes ***before*** class as preparation and complete the questions.
- Participate in class exercises and questions.

Attend and complete all lab assignments:

- Labs practice the fundamental employable skills as well as being for marks.

Practice on your own. Practice makes perfect.

- Do more questions than in the labs.
- Read the additional reference material and perform practice questions.

The Database Implementation Project

The course project uses a database to build an e-commerce web site.

- Database and web development skills are very valuable to employers.
- The project is integrated in the assignments, and is a significant part of the mark.

During the project you will:

- Design a database using ER/UML diagrams.
- Write SQL to query and update the database.
- Develop a web user interface to your database.
  - In the process you will learn HTML and JSP/PHP.
  - Note that limited background will be given on web programming.

The project assignments will be done in groups of **four**.

The Lab Assignments

The lab assignments are critical to learning the material and are designed to build up your skills and prepare you for the exams.

The weekly lab assignments are worth ***25%*** of your overall grade.

You have until the week after the lab is assigned to complete it.

- No late assignments will be accepted.
- An assignment may be handed in any time before the due date and may be marked immediately by the TA.
- Lab assignments may take between 2 and 10 hours depending on the lab.

Lab assignments are done in ***pairs***. Project assignments are done in groups of ***four***.

There is no scheduled lab time. TA help desk hours will be posted.

The Online Questions

To promote understanding, 10% of your overall grade is allocated to answering questions online and during class.

These questions may be multiple choice, short answer, or programming questions.

- All questions will be able to be answered both asynchronously (outside of class time) and synchronously (during class time).
- **No make-ups for forgetting to answer questions. Questions will have posted deadline for when they must be completed.**
- Canvas quizzes will be used as well as real-time polling questions.

A student must only get 80% of questions right to get full 10%.

- That is, if there was a total of 100 marks of online questions, 80 out of 100 will give you 10%. 40 out of 100 would give you 5%.

Systems and Tools

Course material, online quizzes, discussion forums and feedback, and marks are on Canvas.

Access to both MySQL and SQL Server will be provided as well as access to a web server for web development. These systems will have separate user ids and passwords.

All software used will be open source or free to install on your computer. You do not need to access hardware/software at UBCO, but can if you wish.

My Expectations

My goal is for you to learn the material and walk out of this course confident in your abilities:

- To query and update an existing database
- Write code interfacing with a database to build standalone and web-based applications
- To design and model a database using UML

I have high standards on the amount and difficulty of material that we cover. I expect a strong, continual effort in keeping up with readings, doing assignments, and working on projects.

The course will be very straightforward – if you do the work, you will do well.

Your mark is 80% perspiration and 20% inspiration.

Why are you here?

A) It is a required course for the COSC/DATA major.

B) This is an optional elective for my program.

C) Engineering Major taking a minor in Computer Science.

D) Taking a minor in Data Science.

E) Other

What Topic are You Most Interested In?

A) What is a database and how do you use them?

B) Querying using SQL

C) Designing databases

D) Using databases with programs (stand-alone, web, mobile)

E) None of the above

Database Survey Question

Question: Have you used any of these database systems?

A) MySQL

B) Microsoft Access or SQL Server

C) PostgreSQL

D) Used more than two different databases

E) Used no databases

What Grade are You Expecting to Get?

A) A

B) B

C) C

D) D

F) F

The Essence of the Course

Essence of the course is to appreciate that:

Databases are the best way for storing and manipulating *persistent* information. You will learn the skills to exploit the full power of database systems.

The skills you will acquire are in high demand for many software development jobs. Database skills make you more marketable and allow you to construct more sophisticated systems.

- Note: This is a course on how to use/program with databases. It is a very applied course with specific skills.
- If you want to learn how to build database systems and what is "inside the box", that is the subject of COSC 404!


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Database Introduction

COSC 304 – Introduction to Database Systems

What is a database?

A ***database*** is a collection of logically related data for a particular domain.

A ***database management system*** (***DBMS***) is software designed for the creation and management of databases.

- e.g. Oracle, DB2, Microsoft Access, MySQL, SQL Server, MongoDB

Bottom line: A ***database*** is the *data* stored and a ***database system*** is the *software* that manages the data.

Databases in the Real-World

Databases are everywhere in the real-world even though you do not often interact with them directly.

- $40 billion dollar annual industry

Examples:

- Retailers manage their products and sales using a database.
  - Wal-Mart has one of the largest databases in the world!
- Online web sites such as Amazon, eBay, and Expedia track orders, shipments, and customers using databases.
- The university maintains all your registration information and marks in a database that is accessible over the Internet.

Can you think of other examples?

What data do you have?

Developing without a Database

Without a database, applications use files to store data *persistently*. A ***file-based system*** has several problems: code and data duplication, high maintenance costs, and difficulty in supporting multiple users.

- There is no ***program-data independence*** separating the application from the data it is manipulating. If the data file changes, the code must be changed.

Example: You have been asked to build a simple inventory/sales program for a store. What challenges would you face without a DBMS?

Data Independence and Abstraction

Databases provide ***data abstraction*** allowing the internal representation of the data to change without affecting programs that use the object through an external definition.

- The DBMS takes the description of the data and handles the low-level details of how to store it, retrieve it, and provide concurrent access to it.

Database System Properties

A database system provides *efficient*, *convenient*, and *safe multi-user* storage and access to *massive* amounts of *persistent* data.

Efficient - Able to handle large data sets and complex queries without searching all files and data items.

Convenient - Easy to write queries to retrieve data.

Safe - Protects data from system failures and hackers.

Massive - Database sizes in gigabytes, terabytes and petabytes.

Persistent - Data exists even if have a power failure.

Multi-user - More than one user can access and update data at the same time while preserving consistency.

Database Terminology

A ***data model*** is a collection of concepts that is used to describe the structure of a database. E.g. relational, XML, graphs, objects, JSON

- In the relational model, data is represented as tables and fields.

Data Definition Language (***DDL***) allows the user to create data structures in the data model used by the database. A ***schema*** is a description of the structure of the database and is maintained and stored in the ***system catalog***. The schema is ***metadata***.

- A schema contains structures, names, and types of data stored.

Once a database has been created using DDL, the user accesses data using a ***Data Manipulation Language*** (***DML***).

- The DML allows for the insertion, modification, retrieval, and deletion of data.

SQL is a standard DDL and DML for the relational model.

SQL DDL and DML Examples

Create a table to store data on products:

```
CREATE TABLE product(
    sku as VARCHAR(10),
    name as VARCHAR(40),
    desc as VARCHAR(50),
    inventory as INTEGER,
    PRIMARY KEY (sku));
```

Insert product into database:

```
INSERT INTO product VALUES ('1234','Soap','Ivory',100);
```

Retrieve all products where inventory < 10:

```
SELECT name, inventory FROM product WHERE inventory < 10;
```

Database Properties Question

Question: **True or False**: The data in a database is lost when the power to the computer is turned off.

A) true

B) false

Database Abstraction Question

Question: Defining how data is stored using DDL is similar to what in object-oriented programming?

A) Objects

B) Classes

C) Inheritance

D) Polymorphism

DDL vs. DML Question

Question: If you are querying data in a database, which language are you using:

A) DML

B) DDL

C) schemas

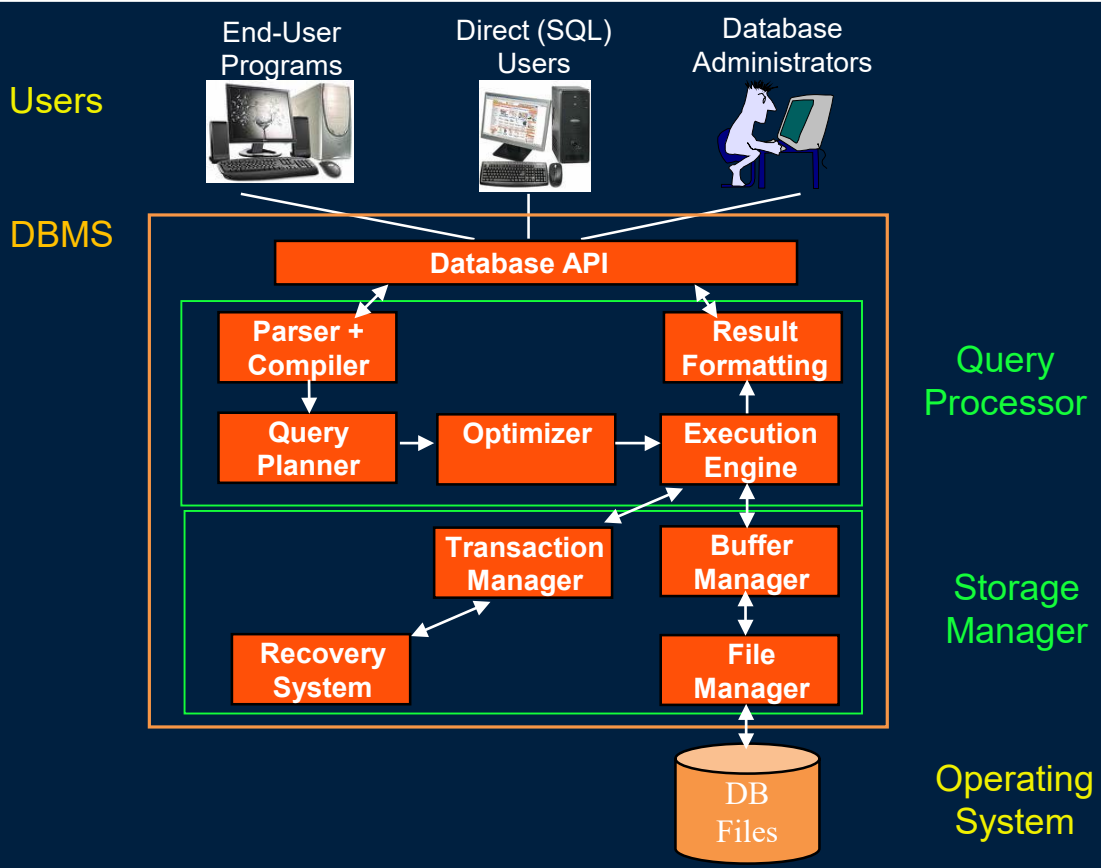
D) Java

Components of a DBMS

A DBMS is a complicated software system containing many components:

- ***Query processor*** - translates user/application queries into low-level data manipulation actions.
  - Sub-components: query parser, query optimizer
- ***Storage manager*** - maintains storage information including memory allocation, buffer management, and file storage.
  - Sub-components: buffer manager, file manager
- ***Transaction manager*** - performs scheduling of operations and implements concurrency control algorithms.

DBMS Architecture



Database Architectures

There are several different database architectures:

- **File-server (embedded) architecture** - files are shared but DBMS processing occurs at the clients (e.g. Microsoft Access or SQLite)
- **Two-Tier client-server architecture** - dedicated machine running DBMS accessed by clients (e.g. SQL Server)
- **Three-Tier client-server architecture** - DBMS is bottom tier, second tier is an application server containing business logic, top tier is clients (e.g. Web browser-Apache/Tomcat-Oracle)

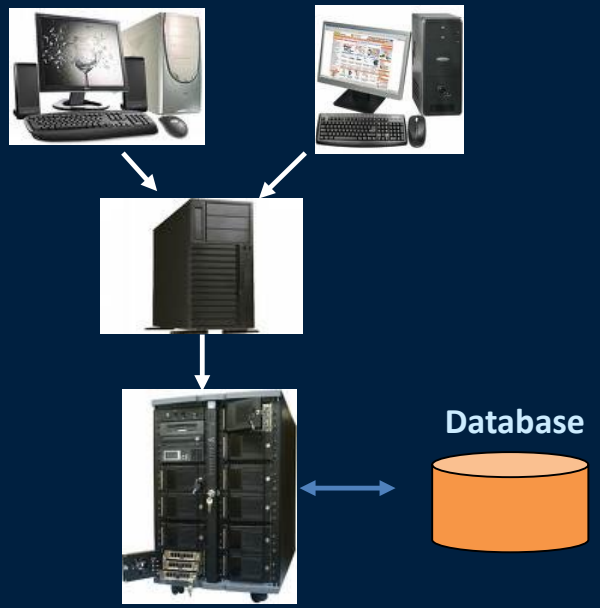
Two-Tier Client-Server Architecture

Tier 1: Client
- User Interface
- Business and Data Logic

Tier 2: Database Server
- Server-side validation
- Data storage/management

Advantages:
- Only one copy of DBMS software on dedicated machine.
- Increased performance.
- Reduced hardware and communication costs.
- Easier to maintain consistency and manage concurrency.

Three-Tier Client-Server Architecture



Tier 1: Client (Web/mobile)
- User Interface

Tier 2: Application Server
- Business logic
- Data processing logic

Tier 3: Database Server
- Data validation
- Data storage/management

Advantages:
- Reduced client administration and cost using thin web clients.
- Easy to scale architecture and perform load balancing.

Database People

There are several different types of database personnel:

- ***Database administrator*** (***DBA***) - responsible for installing, maintaining, and configuring the DBMS software.
- ***Data administrator*** (***DA***) - responsible for organizational policies on data creation, security, and planning.
- ***Database designer*** - defines and implements a schema for a database and associated applications.
  - ***Logical/Conceptual database designer*** - interacts with users to determine data requirements, constraints, and business rules.
  - ***Physical database designer*** - implements the logical design for a data model on a DBMS. Defines indexes, security, and constraints.
- ***DBMS developer*** - writes the DBMS software code.
- ***Application developer*** - writes code that uses the DBMS.
- ***User*** - uses the database directly or through applications.

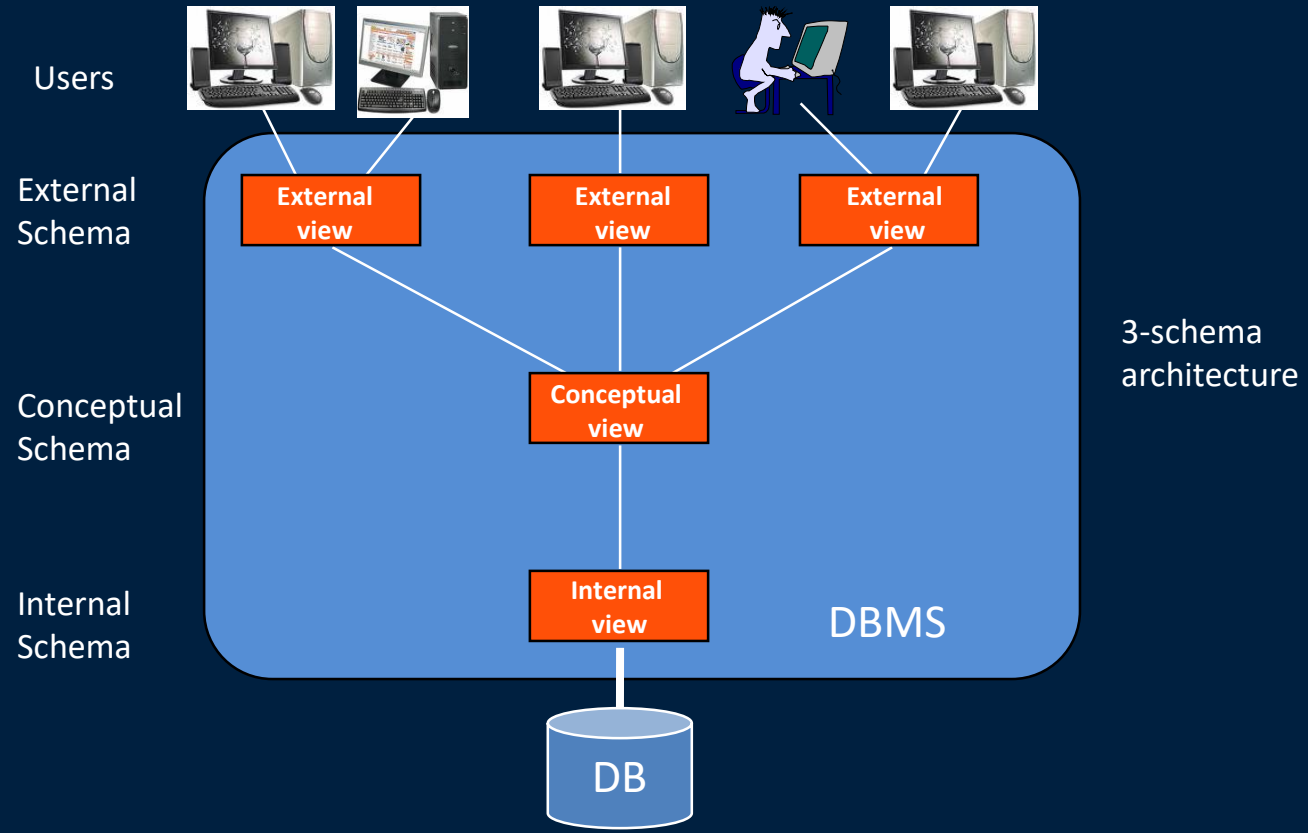
ANSI/SPARC Architecture

One of the major advantages of database systems is data abstraction. Data abstraction is achieved by defining different views of the data. Each view isolates higher-level views from data representation details.

The ANSI/SPARC architecture consists of three views:

- ***Internal View*** - The physical representation of the database on the computer. *How* the data is stored.
- ***Conceptual View*** - The logical structure of the database that describes *what* data is stored and its relationships.
- ***External View*** - The *user's view* of the database that provides the part of the database relevant to the user.

ANSI/SPARC Architecture



Benefits of Three Schema Architecture

External Level:

- Each user can access the data, but have their own view of the data independent of other users.
  - *Logical data independence* - conceptual schema changes do not affect external views.

Conceptual Level:

- Single shared data representation for all applications and users which is independent of physical data storage.
  - Users do not have to understand physical data representation details.
  - The DBA can change the storage structures without affecting users or applications. *Physical data independence* - conceptual schema not affected by physical changes such as adding indexes or distributing data.

Internal (Physical) Level:

- Provides standard facilities for interacting with operating system for space allocation and file manipulation.

Microsoft Access and the Three Schema Architecture

External Level:

- Microsoft Access does not call them views, but you can store queries and use the results in other queries (like a view).
  - External schema is the query (view) name and the attribute metadata.

Conceptual Level:

- All tables and field definitions are in the schema (accessible from the Tables tab).
  - Note that conceptual **schema** is not the data but the metadata.

Physical Level:

- Access represents all data in a single file whose layout it controls.
- The system processes this raw data file by knowing locations and offsets of relations and fields.

ANSI/SPARC Architecture - Three Levels of Views

Question: What are the three levels of views in the ANSI/SPARC architecture starting with the view closest to the user?

A) Internal, Conceptual, External

B) External, Internal, Conceptual

C) Internal, External, Conceptual

D) External, Conceptual, Internal

E) User, Logical, System

ANSI/SPARC Architecture - Abstraction with Views

Question: Assume you have a Java program accessing data stored in a file. Select ***one*** true statement.

A) The file organization is changed. The internal view is where this change is made.

B) A field is added to the database. The conceptual view is changed.

C) A user account has restricted access to the file. The external view must be changed.

D) More than one of the above

Conclusion

A ***database*** is a collection of logically related data managed by a ***database management system*** (DBMS).

- Provides data independence and abstraction
- Data definition and manipulation languages (DDL and DML)

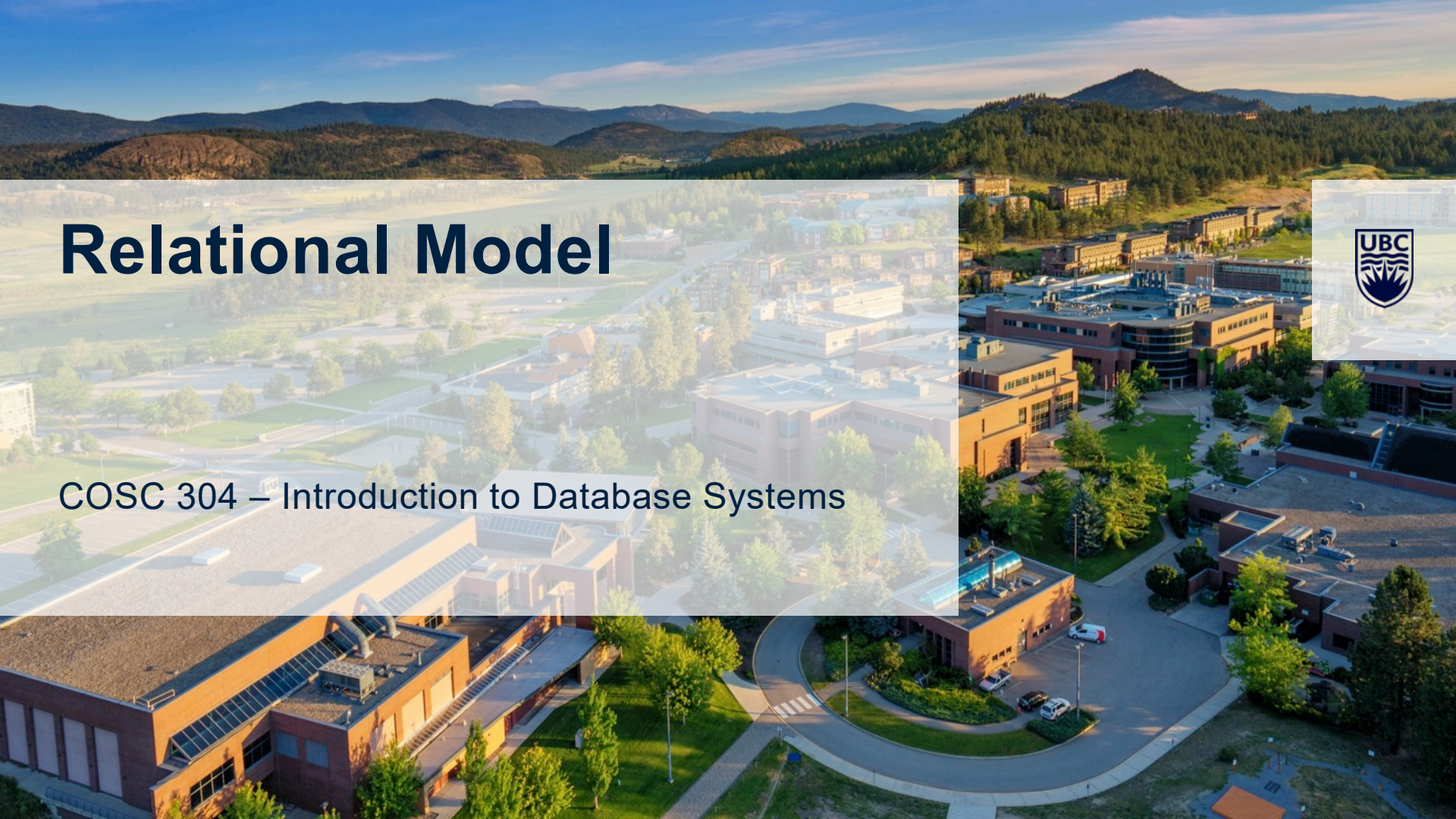
A DBMS has advantages over traditional file systems by supporting data independence and standard implementations for data management tasks.

The three schema architecture consists of external, conceptual, and internal schemas. Each view provides data abstraction and isolates the layer above from certain data manipulation details.

Objectives

- Define: database, DBMS, schema, metadata
- Describe the features of a file-based system and some limitations inherent in that architecture.
- Define program-data independence/data abstraction and explain how it is achieved by databases but not by file systems.
- Define DDL and DML. What is the difference?
- List some modules of a DBMS.
- List different people associated with a DBMS and their roles.
- Explain how a schema differs from data.
- Draw a diagram of the three schema architecture and explain what each level provides. List benefits of the architecture.
- How does a schema provide data independence?
- Compare/contrast two-tier and three-tier architectures.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA



Relational Model

COSC 304 – Introduction to Database Systems

Relational Model History

The relational model was proposed by E. F. Codd in 1970. Commercial implementations appeared in the late 1970s and early 1980s.

One of the first relational database systems, System R, developed at IBM led to several important breakthroughs:

- the first version of SQL
- various commercial products such as Oracle and DB2
- extensive research on concurrency control, transaction management, and query processing and optimization

Currently, the relational model is the foundation of the majority of commercial database systems.

The Relational Model: Terminology

The ***relational model*** organizes data into tables called relations.

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.

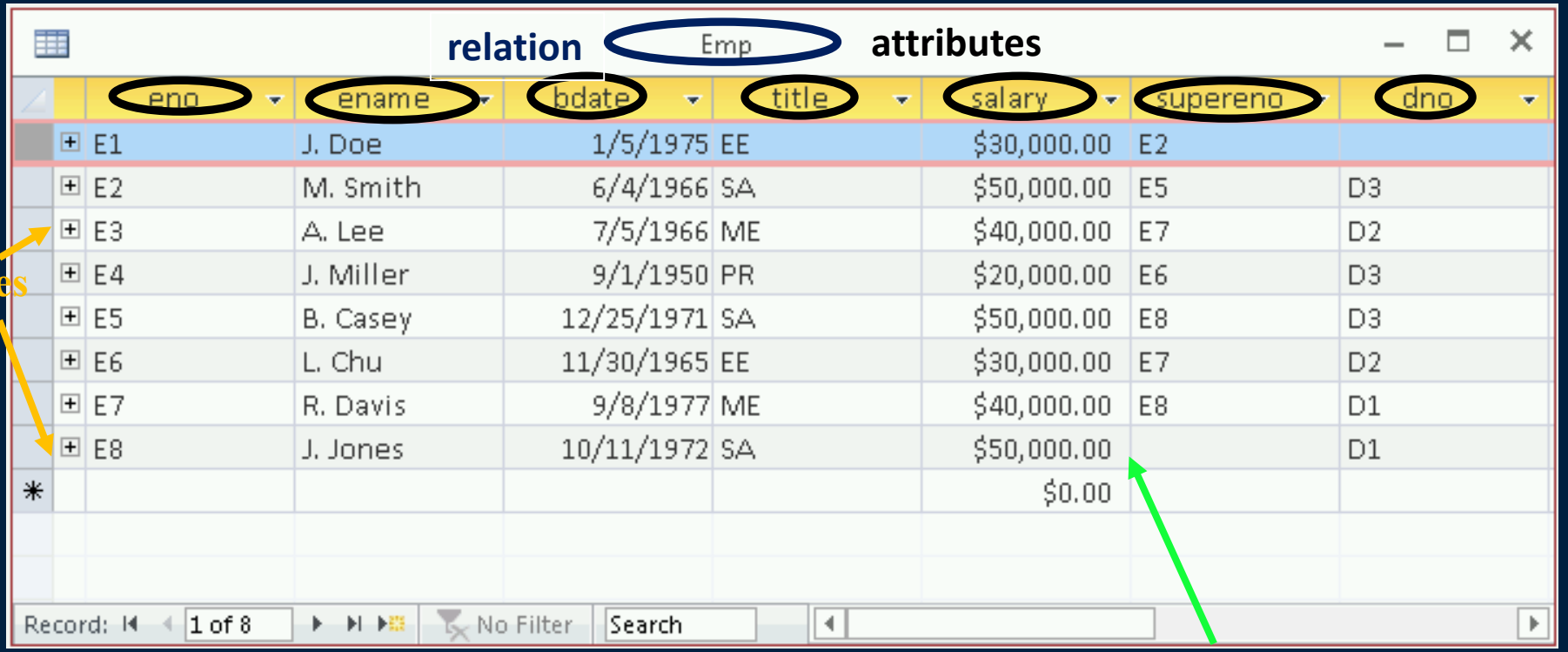
The ***degree*** of a relation is the number of attributes it contains.

The ***cardinality*** of a relation is the number of tuples it contains.

The ***intension*** is the structure of the relation including its domains.

The ***extension*** is the set of tuples currently in the relation.

Relation Example



eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1/5/1975	EE	$30,000.00	E2	
E2	M. Smith	6/4/1966	SA	$50,000.00	E5	D3
E3	A. Lee	7/5/1966	ME	$40,000.00	E7	D2
E4	J. Miller	9/1/1950	PR	$20,000.00	E6	D3
E5	B. Casey	12/25/1971	SA	$50,000.00	E8	D3
E6	L. Chu	11/30/1965	EE	$30,000.00	E7	D2
E7	R. Davis	9/8/1977	ME	$40,000.00	E8	D1
E8	J. Jones	10/11/1972	SA	$50,000.00		D1
*				$0.00		

Degree =7

Cardinality = 8

Domain of salary is *currency*

Relational Model Formal Definition

The relational model is formally defined in terms of sets and set operations.

A ***relation schema*** ***R*** (A_1, A_2, …, A_n) has each attribute A_i with a name and a domain $dom(A_i)$.

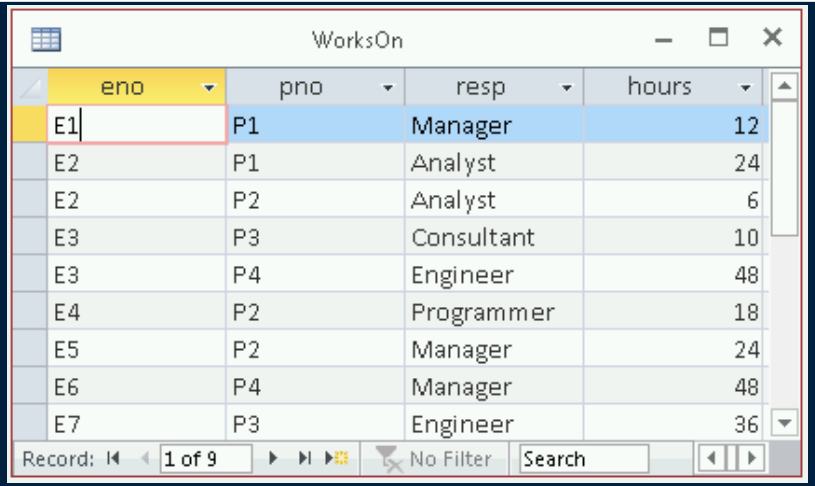
A ***relation instance*** denoted $r(R)$ is a set of n-tuples $<d_1, d_2, ..., d_n>$ where each d_i is an element of $dom(A_i)$ or is **null**.

- The relation instance is the *extension* of the relation.
- A value of **null** represents a missing or unknown value.

Example: Product (id, name, supplierId, categoryId, price)

- ***R*** = Product (relation name)
- Set ***A*** = {id, name, supplierId, categoryId, price}
- dom(price) is set of all possible positive currency values
- dom(name) is set of all possible strings that represent people's names

Relation Practice Questions



WorksOn

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

Record: 1 of 9 No Filter Search

1) What is the name of the relation?
2) What is the cardinality of the relation?
3) What is the degree of the relation?
4) What is the domain of `resp`? What is the domain of `hours`?
5) What is larger the size of the intension or extension?
6) Is a relation's cardinality always bigger than its degree?

Database Definition Matching Question

Question: Given the three definitions, select the ordering that contains their related definitions.

1) relation

2) tuple

3) attribute

A) column, row, table

B) row, column, table

C) table, row, column

D) table, column, row

Cardinality and Degree Question

Question: A database table has 5 rows and 10 columns. Select ***one*** true statement.

A) The table's degree is 50.

B) The table's cardinality is 5.

C) The table's degree is 5.

D) The table's cardinality is 10.

Relation Properties

1) No two relations have the same name.
2) Each attribute of a relation has a distinct name.
3) Each tuple is distinct. There are no duplicate tuples.
4) The order of attributes is not important.
5) The order of tuples has no significance.

Relational Keys

Keys are used to uniquely identify a tuple in a relation.

- Note that keys apply to the schema not to the data. That is, looking at the current data cannot tell you for sure if the set of attributes is a key.

A ***superkey*** is a set of attributes that uniquely identifies a tuple in a relation.

A (***candidate***) ***key*** is a *minimal* set of attributes that uniquely identifies a tuple in a relation.

- There may be more than 1 candidate key for a relation with different # of attributes.

A ***primary key*** is the candidate key designated as the distinguishing key of a relation.

A ***foreign key*** is a set of attributes in one relation referring to the primary key of a relation.

- Foreign keys enforce referential integrity. Note: A FK may refer to its own relation.

Keys and Superkeys Question

Question: **True or false:** A key is always a superkey.

A) true

B) false

Keys and Superkeys Question (2)

Question: **True or false:** It is possible to have more than one key for a table and the keys may have different numbers of attributes.

A) true

B) false

Keys and Superkeys Question (3)

Question: **True or false:** It is possible to always determine if a field is a key by looking at the data in the table.

A) true

B) false

Example Relational Data Questions

Keys are underlined

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Questions:

1) Is *ename* a key for *emp*?
2) Is *eno* a key for *WorksOn*?
3) List all the superkeys for *WorksOn*.

Keys Practice Question

Consider a relation storing driver information including:

- SSN, name, license plate number and state (unique together)

Driver Relation

SSN	name	LicNum	LicState
123-45-6789	S. Smith	123-456	IA
111-11-1111	A. Lee	123-456	NY
222-22-2222	J. Miller	555-111	MT
333-33-3333	B. Casey	678-123	OH
444-44-4444	A. Adler	456-345	IA

Questions:

1) List the candidate keys for the relation.
2) Pick a primary key for the relation.
3) Is *name* a candidate key for *Person*?
4) List all the superkeys for *Person*.

Assumptions:

1) A person has only one license plate.
2) A license plate uniquely identifies a person.

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*** - No attribute of a primary key can be null.
- ***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.

Foreign Keys Example

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

WorksOn.eno is FK to Emp.eno

WorksOn.pno is FK to Proj.pno

Foreign Keys Example (2)

Proj Relation

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	CAD/CAM	250000	D3
P4	Maintenance	310000	null
P5	CAD/CAM	500000	D1

Proj.dno is FK to Dept.dno

Department Relation

dno	dname
D1	Management
D2	Consulting
D3	Accounting
D4	Development

Integrity Constraints Question

Question: What constraint says that a primary key field cannot be null?

A) domain constraint

B) referential integrity constraint

C) entity integrity constraint

Entity Integrity Constraint Question

Question: A primary key has three fields. Only one field is `null`. Is the entity integrity constraint violated?

A) Yes

B) No

Referential Integrity Constraint Question

Question: A foreign key has a `null` value in the table that contains the foreign key fields. Is the referential integrity constraint violated?

A) Yes

B) No

Integrity Questions

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	AS
E2	null	SA	50000
E3	A. Lee	null	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
null	L. Chu	EE	30000
E7	R. Davis	ME	null
E8	J. Jones	SA	50000

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	null	null

WorksOn Relation

eno	pno	resp	dur
E1	P0	null	12
E2	P1	Analyst	null
null	P2	Analyst	6
E3	P3	Consultant	10
E9	P4	Engineer	48
E4	P2	Programmer	18
E5	null	Manager	24
E6	P4	Manager	48
E7	P6	Engineer	36
E7	P4	Engineer	23
null	null	Manager	40

Question: How many rows have violations of integrity constraints? Note: salary, budget, dur are number fields.

A) 8 B) 9 C) 10 D) 11 E) 12

Conclusion

A ***database*** is a collection of logically related data managed by a ***database management system*** (DBMS).

- Provides data independence and abstraction
- Data definition and manipulation languages (DDL and DML)

The ***relational model*** represents data as relations which are sets of tuples. Each relational schema is a set of attributes with domains.

The relational model has ***constraints*** to guarantee data integrity including: domain, entity integrity and referential integrity constraints. ***Keys*** are used to uniquely identify tuples in relations.

Objectives

- Define: relation, attribute, tuple, domain, degree, cardinality, intension, extension, relation schema, relation instance, null
- List the properties of relations.
- Define: superkey, key, candidate key, primary key, foreign key
- Define: integrity, constraints, domain constraint, entity integrity constraint, referential integrity constraint
- Given a relation be able to:
  - identify its cardinality, degree, domains, keys, and superkeys
  - determine if constraints are being violated


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

Relational Algebra

COSC 304 – Introduction to Database Systems

Relational Algebra Query Language

A ***query language*** is used to update and retrieve data that is stored in a data model.

Relational algebra is a set of relational operations for retrieving data.

- Just like algebra with numbers, relational algebra consists of operands (which are relations) and a set of operators.

Every relational operator takes as input one or more relations and produces a relation as output.

A sequence of relational algebra operators is called a ***relational algebra expression***.

Relational algebra is the foundation of all relational database systems. SQL gets translated into relational algebra.

Relational Algebra Operators

Operator	Symbol	Purpose
Selection	σ	Filter rows
Projection	Π	Keep only certain columns
Cartesian Product	$\times$	Combine two tables in all possible ways
Join	$\bowtie$	Combine two tables based on a condition
Union	$\cup$	Keep rows in either of two tables
Difference	-	Keep rows in first table that are not in second
Intersection	$\cap$	Keep rows that are in both tables

Relational algebra operators are fundamental to data processing and occur in other data systems (even non-relational).

Selection Operation

The ***selection operation*** takes a relation as input and returns a new relation as output that contains tuples that satisfy a condition, called a ***predicate***.

- The predicate is similar to a condition in an `if` statement.
- The output relation has the same number of columns as the input relation, but may have less rows.

Selection operation on relation *R* with predicate *F* is denoted by $\sigma_F(R)$.

- The predicate uses operands that are attributes or constants/expressions, comparison operators (<, >, =, ≠, ≤, ≥), and logical operators (AND, OR, NOT).

Selection Example

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

$\sigma_{title = 'EE'}(Emp)$

eno	ename	title	salary
E1	J. Doe	EE	30000
E6	L. Chu	EE	30000

$\sigma_{salary > 35000\ OR\ title = 'PR'}(Emp)$

eno	ename	title	salary
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question

Question: Given this table and the query:

$$\sigma_{salary > 50000 \text{ or } title='PR'}(Emp)$$

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question (2)

Question: Given this table and the query:

$$\sigma_{salary > 50000 \text{ or } title='PR'}(Emp)$$

How many columns are returned?

A) 0

B) 2

C) 3

D) 4

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Projection Operation

The ***projection operation*** takes a relation as input and returns a new relation as output that contains a subset of the attributes of the input relation and all non-duplicate tuples.

- The output relation has the same number of tuples as the input relation unless removing the attributes caused duplicates to be present.
- Question: When are we guaranteed to never have duplicates when performing a projection operation?

Projection on relation R with output attributes $A_1,\ldots,A_m$ is denoted by $\boldsymbol{\Pi_{A_1,\ldots,A_m}(R)}$.

- Order of $A_1,\ldots, A_m$ is significant in the result.
- Cardinality of $\Pi_{A_1,\ldots,A_m}(R)$ may not be the same as R due to duplicate removal.

Projection Example

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

$\Pi_{eno,ename}(Emp)$

eno	ename
E1	J. Doe
E2	M. Smith
E3	A. Lee
E4	J. Miller
E5	B. Casey
E6	L. Chu
E7	R. Davis
E8	J. Jones

$\Pi_{title}(Emp)$

title
EE
SA
ME
PR

Projection Question

Question: Given this table and the query:

$$\Pi_{title}(\text{Emp})$$

How many rows are returned?

A) 0

B) 2

C) 4

D) 8

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection and Projection Questions

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

Write the relational algebra expression that:

1) Returns all rows with an employee working on project P2.
2) Returns all rows with an employee who is working as a manager on a project.
3) Returns all rows with an employee working as a manager for more than 40 months.
4) Returns only attributes *resp* and *dur*.
5) Returns only *eno*.
6) Returns only *pno*.

How many tuples in each output relation?

Union

Union takes two relations *R* and *S* as input and produces an output relation that includes all tuples that are either in *R*, or in *S*, or in both *R* and *S*. Duplicate tuples are eliminated. Syntax: $R \cup S$

R and *S* must be ***union-compatible***:

- 1) Both relations have same number of attributes.
- 2) Each attribute pair, R_i and S_i, have compatible data types for all attribute indexes *i*.
- Note that attributes do not need to have the same name.
- Result has attribute names of first relation.

Union Example

Emp

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

WorksOn

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23

$\Pi_{eno}(\text{Emp}) \cup \Pi_{eno}(\text{WorksOn})$

eno
E1
E2
E3
E4
E5
E6
E7
E8

Set Difference

Set difference takes two relations *R* and *S* as input and produces an output relation that contains all the tuples of *R* that are not in *S*.

Syntax: ***R – S***

Note:

- R – S ≠ S – R
- R and S must be union compatible.

Set Difference Example

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23

$\Pi_{eno}(\text{Emp}) - \Pi_{eno}(\text{WorksOn})$

eno
E4
E8

Question: What is the meaning of this query?

Question: What is $\Pi_{eno}(\text{WorksOn}) - \Pi_{eno}(\text{Emp})$?

Intersection

Intersection takes two relations R and S as input and produces an output relation which contains all tuples that are in both R and S.

- R and S must be union-compatible.

Syntax: $\boldsymbol{R \cap S}$

Note that $R \cap S = R - (R - S) = S - (S - R)$.

Intersection Example

Emp

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

WorksOn

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23

$\Pi_{eno}(\text{Emp}) \cap \Pi_{eno}(\text{WorksOn})$

eno
E1
E2
E3
E5
E6
E7

Set Operations: Union-compatible Question

Question: Two tables have the same number of fields in the same order with the same types, but the names of some fields are different.
True or false: The two tables are union-compatible.

A) true

B) false

Cartesian Product

Cartesian product of relations R (of degree k_1) and S (of degree k_2) is a relation of degree $(k_1 + k_2)$ that consists of all $(k_1 + k_2)$-tuples where each tuple is a concatenation of one tuple of R with one tuple of S.
Syntax: $R \times S$

The cardinality of $R \times S$ is $|R| * |S|$.

The Cartesian product is also known as ***cross product***.

Cartesian Product Example

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000

Emp × *Proj*

eno	ename	title	salary	pno	pname	budget
E1	J. Doe	EE	30000	P1	Instruments	150000
E2	M. Smith	SA	50000	P1	Instruments	150000
E3	A. Lee	ME	40000	P1	Instruments	150000
E4	J. Miller	PR	20000	P1	Instruments	150000
E1	J. Doe	EE	30000	P2	DB Develop	135000
E2	M. Smith	SA	50000	P2	DB Develop	135000
E3	A. Lee	ME	40000	P2	DB Develop	135000
E4	J. Miller	PR	20000	P2	DB Develop	135000
E1	J. Doe	EE	30000	P3	CAD/CAM	250000
E2	M. Smith	SA	50000	P3	CAD/CAM	250000
E3	A. Lee	ME	40000	P3	CAD/CAM	250000
E4	J. Miller	PR	20000	P3	CAD/CAM	250000

Cartesian Product Question

Question: *R* is a relation with 10 rows and 5 columns. *S* is a relation with 8 rows and 3 columns.

What is the degree and cardinality of the Cartesian product?

A) degree = 8, cardinality = 80

B) degree = 80, cardinality = 8

C) degree = 15, cardinality = 80

D) degree = 8, cardinality = 18

θ-Join

Theta (θ) join is a derivative of the Cartesian product. Instead of taking all combinations of tuples from *R* and *S*, we only take a subset of those tuples that match a given condition *F*. Syntax: $\boldsymbol{R \bowtie_F S}$

Note that $R \bowtie_F S = \sigma_F(R \times S)$.

θ-Join Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn $\bowtie_{dur*10000 > budget}$ Proj

eno	pno	resp	dur	P.pno	pname	budget
E2	P1	Analyst	24	P1	Instruments	150000
E2	P1	Analyst	24	P2	DB Develop	135000
E3	P4	Engineer	48	P1	Instruments	150000
E3	P4	Engineer	48	P2	DB Develop	135000
E3	P4	Engineer	48	P3	CAD/CAM	250000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P1	Instruments	150000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P1	Instruments	150000
E6	P4	Manager	48	P2	DB Develop	135000
E6	P4	Manager	48	P3	CAD/CAM	250000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P1	Instruments	150000
E7	P3	Engineer	36	P2	DB Develop	135000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P1	Instruments	150000
E7	P4	Engineer	23	P2	DB Develop	135000

Types of Joins

The θ-Join is a general join that allows any expression for condition *F*. However, there are more specialized joins that are frequently used.

A ***equijoin*** only contains the equality operator (=) in formula *F*.

- e.g. WorksOn $\bowtie_{WorksOn.pno = Proj.pno}$ Proj

A ***natural join*** over two relations *R* and *S* denoted by $R \bowtie S$ is the equijoin of *R* and *S* over a set of attributes common to both *R* and *S*.

- It removes the "extra copies" of the join attributes.
- The attributes must have the same name in both relations.

Equijoin Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn $\bowtie_{WorksOn.pno = Proj.pno}$ Proj

eno	pno	resp	dur	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000

What is the meaning of this join?

Natural join Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn ⋈ Proj

eno	pno	resp	dur	pname	budget
E1	P1	Manager	12	Instruments	150000
E2	P1	Analyst	24	Instruments	150000
E2	P2	Analyst	6	DB Develop	135000
E3	P4	Engineer	48	Maintenance	310000
E5	P2	Manager	24	DB Develop	135000
E6	P4	Manager	48	Maintenance	310000
E7	P3	Engineer	36	CAD/CAM	250000
E7	P4	Engineer	23	Maintenance	310000

Natural join is performed by comparing *pno* in both relations.

Join Practice Questions

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

Compute the following joins (counts only):

1) Emp $\bowtie_{\text{title='EE' and budget > 400000}}$ Proj
2) Emp $\bowtie$ WorksOn
3) Emp $\bowtie$ WorksOn $\bowtie$ Proj
4) $\text{Proj}_1 \bowtie_{\text{Proj1.budget > Proj2.budget}} \text{Proj}_2$

Outer Joins

Outer joins are used in cases where performing a join "loses" some tuples of the relations. These are called *dangling tuples*.

There are three types of outer joins:

- 1) ***Left outer join*** - $R \mathbin{⟕} S$ - The output contains all tuples of *R* that match with tuples of *S*. If there is a tuple in *R* that matches with no tuple in *S*, the tuple is included in the final result and is padded with nulls for the attributes of *S*.
- 2) ***Right outer join*** - $R \mathbin{⟖} S$ - The output contains all tuples of *S* that match with tuples of *R*. If there is a tuple in *S* that matches with no tuple in *R*, the tuple is included in the final result and is padded with nulls for the attributes of *R*.
- 3) ***Full outer join*** - $R \mathbin{⟗} S$ - All tuples of *R* and *S* are included in the result whether or not they have a matching tuple in the other relation.

Right Outer Join Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

WorksOn $\overset{\subset}{\bowtie}_{WorksOn.pno = Proj.pno}$ Proj

eno	pno	resp	dur	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000
null	null	null	null	P5	CAD/CAM	500000

Outer Join Question

Question: Given this table and the query:

$$\text{WorksOn} \overset{}{\supset\!\!\bowtie}_{\text{WorksOn.pno = Proj.pno}} \text{Proj}$$

How many rows are returned?

A) 10

B) 9

C) 8

D) 7

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P4	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Semi-Join and Anti-Join

A ***semi-join*** between tables returns rows from the first table where one or more matches are found in the second table.

- Semi-joins are used in `EXISTS` and `IN` constructs in SQL.

An ***anti-join*** between two tables returns rows from the first table where ***no*** matches are found in the second table.

- Anti-joins are used with `NOT EXISTS`, `NOT IN`, and `FOR ALL`.
- Anti-join is the complement of semi-join: $R \triangleright S = R - R \ltimes S$

Semi-Join Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Proj $\ltimes_{Proj.pno = WorksOn.pno}$ WorksOn

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

Anti-Join Example

WorksOn Relation

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

Proj Relation

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Proj $\triangleright_{Proj.pno = WorksOn.pno}$ WorksOn

pno	pname	budget
P5	CAD/CAM	500000

Combining Operations

Relational algebra operations can be combined in one expression by nesting them:

$$\Pi_{eno,pno,dur}(\sigma_{ename='J.\ Doe'}(Emp) \bowtie \sigma_{dur>16}(WorksOn))$$

- Return the eno, pno, and duration for employee 'J. Doe' when he has worked on a project for more than 16 months.

Can also use a temporary relation variables for intermediate results.

- Use the assignment operator $\leftarrow$ for indicating that the result of an operation is assigned to a temporary relation.

$$\begin{aligned}
\text{empdoe} &\leftarrow \sigma_{ename='J.\ Doe'}(\text{Emp}) \\
\text{wodur} &\leftarrow \sigma_{dur>16}(\text{WorksOn}) \\
\text{empwo} &\leftarrow \text{empdoe} \bowtie \text{wodur} \\
\text{result} &\leftarrow \Pi_{eno,pno,dur}(\text{empwo})
\end{aligned}$$

Rename Operation

Renaming can be applied when assigning a result:

$$\text{result(EmployeeNum, ProjectNum, Duration)} \leftarrow \Pi_{eno,pno,dur}(empwo)$$

Or by using the rename operator ρ (rho):

$$\rho_{\text{result(EmployeeName, ProjectNum, Duration)}}(empwo)$$

Operator Precedence

Just like mathematical operators, the relational operators have precedence.

The precedence of operators from highest to lowest is:

- unary operators - σ, Π, ρ
- Cartesian product and joins - X, $\bowtie$, division
- intersection
- union and set difference

Parentheses can be used to changed the order of operations.

It is a good idea to ***always*** use parentheses around the argument for both unary and binary operators.

Complete Set of Relational Algebra Operators

It has been shown that the relational operators $\{\sigma, \Pi, \times, \cup, -\}$ form a complete set of operators.

- That is, any of the other operators can be derived from a combination of these 5 basic operators.

Examples:

- Intersection - $R \cap S \equiv R \cup S - ((R - S) \cup (S - R))$
- We have also seen how a join is a combination of a Cartesian product followed by a selection.

Relational Algebra Query Examples

Consider the database schema

Emp (eno, ename, title, salary)

Proj (pno, pname, budget)

WorksOn (eno, pno, resp, dur)

Queries:

- List the names of all employees.
  - $\Pi_{ename}(Emp)$
- Find the names of projects with budgets over $100,000.
  - $\Pi_{pname}(\sigma_{budget>100000}(Proj))$

Practice Questions

Relational database schema:

branch (bname, address, city, assets)
customer (cname, street, city)
deposit (accnum, cname, bname, balance)
borrow (accnum, cname, bname, amount)

1) List the names of all branches of the bank.
2) List the names of all deposit customers together with their account numbers.
3) Find all cities where at least one customer lives.
4) Find all cities with at least one branch.
5) Find all cities with at least one branch or customer.
6) Find all cities that have a branch but no customers who live in that city.

Practice Questions (2)

branch (bname, address, city, assets)
customer (cname, street, city)
deposit (accnum, cname, bname, balance)
borrow (accnum, cname, bname, amount)

1) Find the names of all branches with assets greater than $2,500,000.

2) List the name and cities of all customers who have an account with balance greater than $2,000.

3) List all the cities with at least one customer but without any bank branches.

4) Find the name of all the customers who live in a city with no bank branches.

Practice Questions (3)

branch (bname, address, city, assets)
customer (cname, street, city)
deposit (accnum, cname, bname, balance)
borrow (accnum, cname, bname, amount)

1) Find all the cities that have both customers and bank branches.

2) List the customer name and loan and deposit amounts, who have a loan larger than a deposit account at the same branch.

3) Find the name and assets of all branches which have deposit customers living in Vancouver.

4) Find all the customers who have both a deposit account and a loan at the branch with name CalgaryCentral.

5) Your own?

Other Relational Algebra Operators

There are other relational algebra operators that we will not discuss. Most notably, we often need ***aggregate operations*** that compute functions on the data.

For example, given the current operators, we cannot answer the query:

- What is the total amount of deposits at the Kelowna branch?

We will see how to answer these queries when we study SQL.

Conclusion

Relational algebra is a set of operations for answering queries on data stored in the relational model.

Operator	Symbol	Purpose
Selection	σ	Filter rows
Projection	Π	Keep only certain columns
Cartesian Product	$\times$	Combine two tables in all possible ways
Join		Combine two tables based on a condition
Union	$\cup$	Keep rows in either of two tables
Difference	-	Keep rows in first table that are not in second
Intersection	$\cap$	Keep rows that are in both tables

By combining relational operators, queries can be answered over the base relations.

Objectives

Define: relational algebra, query language

Define and perform all relational algebra operators.

List the operators which form the complete set of operators.

Given a relational schema and instance be able to translate English queries into relational algebra and show the resulting relation.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

SQL DDL: CREATE, INSERT, DELETE, UPDATE

COSC 304 – Introduction to Database Systems

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

SQL history:

- 1970 - Codd invents relational model and relational algebra
- 1974 - Donald Chamberlin (also at IBM) defined Structured English Query Language (SEQUEL)
- 1976 - SEQUEL/2 defined and renamed SQL for legal reasons.
  - Origin of pronunciation 'See-Quel' but official pronunciation is 'S-Q-L'.
- First standardized in 1986 by the American National Standards Institute (ANSI).
- 1992 - SQL2 (SQL92) revision
- 1999 - SQL3 (supports recursion, object-relational)
- Updates: SQL:2003, SQL:2006, SQL:2008, SQL:2011,SQL:2016

SQL Basic Rules

1) There is a set of *reserved words* that cannot be used as names for database fields and tables.
   - `SELECT`, `FROM`, `WHERE`, etc.

2) SQL is generally *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 (escape) a single quote character in a character constant. e.g. `'Master''s'`.

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"`

Database Identifier Question

Question: Select ***one*** valid identifier.

A) `23test`

B) `'fred'`

C) `test_!`

D) `field_`

E) `from`

Delimited Database Identifiers

Question: **True or False:** `"from"` can be used as a valid identifier according to the SQL standard.

A) True

B) False

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)

Data Type	Description
BOOLEAN	TRUE or FALSE
CHAR	Fixed length string (padded with blanks) e.g. CHAR(10)
VARCHAR	Variable length string e.g. VARCHAR(50)
BIT	Bit string e.g. BIT(4) can store '0101'
NUMERIC or DECIMAL	Exact numeric data type e.g. NUMERIC(7,2) has a precision (max. digits) of 7 and scale of 2 (# of decimals) e.g. 12345.67
INTEGER	Integer data only
SMALLINT	Smaller space than INTEGER
FLOAT or REAL	Approximate numeric data types.
DOUBLE PRECISION	Precision dependent on implementation.
DATE	Stores YEAR, MONTH, DAY
TIME	Stores HOUR, MINUTE, SECOND
DATETIME or TIMESTAMP	Stores date and time data.
INTERVAL	Time interval.
CHARACTER LARGE OBJECT	Stores a character array (e.g. for a document)
BINARY LARGE OBJECT	Stores a binary array (e.g. for a picture, movie)

Example Database - WorksOn

emp

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1975-01-05	EE	30000	E2	null
E2	M. Smith	1966-06-04	SA	50000	E5	D3
E3	A. Lee	1966-07-05	ME	40000	E7	D2
E4	J. Miller	1950-09-01	PR	20000	E6	D3
E5	B. Casey	1971-12-25	SA	50000	E8	D3
E6	L. Chu	1965-11-30	EE	30000	E7	D2
E7	R. Davis	1977-09-08	ME	40000	E8	D1
E8	J. Jones	1972-10-11	SA	50000	null	D1

workson

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

proj

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

dept

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

SQL CREATE TABLE

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

Example:

```
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)
    )
```

field must always have a value

Data Types:

CHAR(5) – always 5 chars long
VARCHAR(30) – up to 30 chars long
DECIMAL(9,2) – e.g. 1234567.99
DATE – e.g. 1998/01/18

SQL Constraints

Constraints are specified in `CREATE` and `ALTER TABLE` statements.

Types of constraints:

- 1) **Required data** - To specify that a column must always have a data value (cannot be `NULL`) specify `NOT NULL` after the column definition.
  - e.g. `eno CHAR(5) NOT NULL`
  - If a field is `UNIQUE` or a `PRIMARY KEY`, `NOT NULL` is not necessary.
- 2) **Domain constraints** - Verify that the value of a column is in a given domain using `CHECK`.
  - e.g. `title CHAR(2) CHECK (title IN (NULL,'EE','SA','PR','ME'))`
  - Forces the title to be either `NULL` or one of 4 defined values.
  - Can also be performed using user-defined types (domains).

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.

SQL Referential Integrity Example (2)

```
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
);
```

Enforcing Referential Integrity Question

Question: Select ***one*** true statement.

A) `SET NULL` can be used for the `workson.eno` foreign key.

B) `ON UPDATE CASCADE` will modify all rows in the primary key table when a value is modified in the foreign key table.

C) `SET DEFAULT` cannot be used for the `workson.eno` foreign key. (Assume a default value was specified for `eno` field).

D) If a primary key row is deleted and it is referenced by a foreign key row, `NO ACTION` will generate an error to the user.

SQL CREATE TABLE Full Syntax

Full syntax of `CREATE TABLE` statement:

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

Creating the Example Database

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

Creating the Example Database (2)

```
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
);
```

Question:

Write `CREATE TABLE` statements to build the `proj` and `dept` relations:

- `dept(dno, dname, mgreno)`
- `proj(pno, pname, budget, dno)`

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)

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

Dropping a schema:

```
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:

```
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:

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

Add field `SSN` to `Emp` relation:

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

Indicate that `SSN` is `UNIQUE` in `emp`:

```
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:

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

Example: `DROP TABLE emp;`

- Note: The database does not confirm if you really want to drop the table and delete its data. The effect of the command is immediate.
- RESTRICT will not drop object if it is used. CASCADE will drop object even if it is used.

Question: What would be the effect of the command:

```
DROP TABLE emp CASCADE;
```

Indexes

Indexes are used to speed up access to the rows of a table based on the values of certain attributes.

- An index will often significantly improve the performance of a query, however they represent an overhead as they must be updated every time the table is updated.

The general syntax for creating and dropping indexes is:

```
CREATE [UNIQUE] INDEX indexName
    ON  tableName (colName [ASC|DESC] [,...])

DROP INDEX indexName;
```

- `UNIQUE` means that each value in the index is unique.
- `ASC/DESC` specifies the sorted order of index.

Indexes Example

Creating an index on `eno` and `pno` in `WorksOn` is useful as it will speed up joins with the `Emp` and `Proj` tables respectively.

- Index is not `UNIQUE` as `eno` (`pno`) can occur many times in `WorksOn`.

```
CREATE INDEX idxEno ON workson (eno);
CREATE INDEX idxPno ON workson (pno);
```

Most DBMSs will put an index on the primary key, but if they did not, this is what it would like for `workson`:

```
CREATE UNIQUE INDEX idxPK ON workson (eno,pno);
```

`CREATE TABLE` Question

Question: How many of the following statements are **TRUE**?

1) Each field in the `CREATE TABLE` statement is separated by a comma.

2) The data type for a field is optional.

3) You can create two tables in a database with the same name (in the same schema).

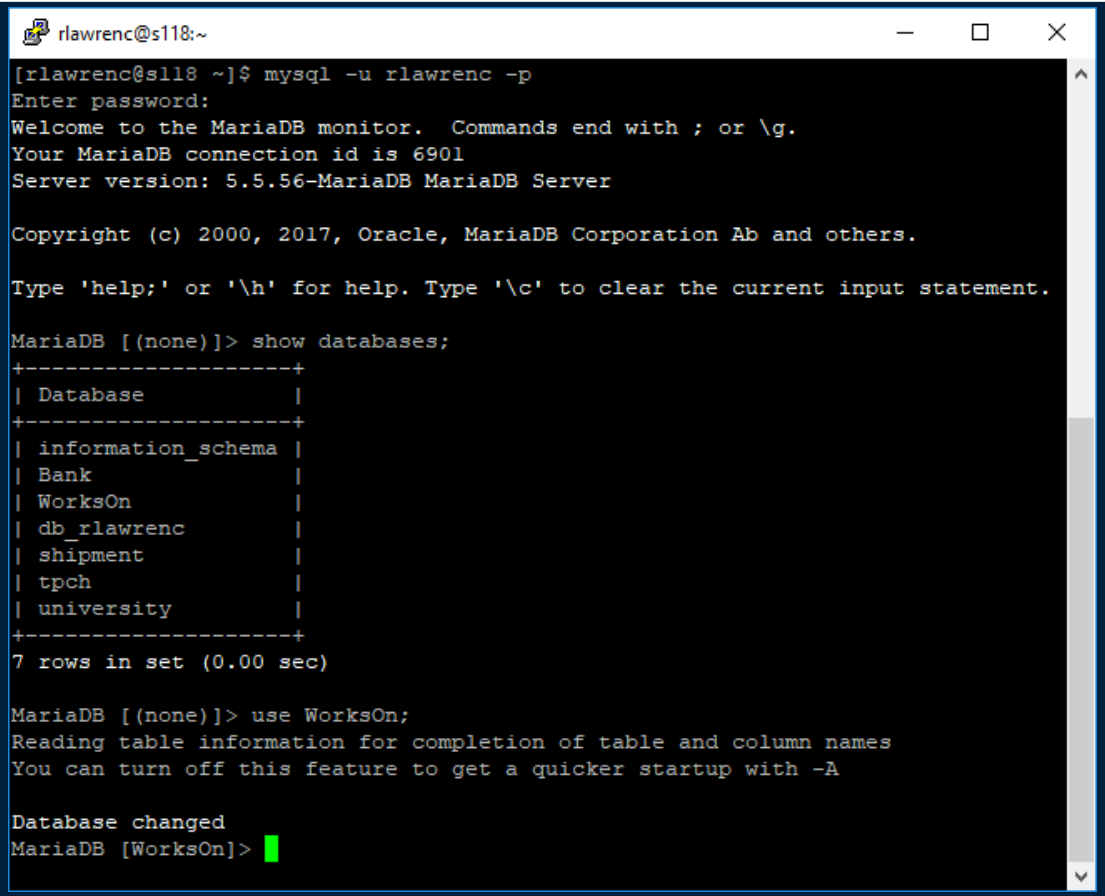
4) A table will not be dropped (with `DROP TABLE`) if it contains data.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Connecting to MySQL – Command Line

1) Use a SSH software (e.g. PuTTY) to connect to **cosc304.ok.ubc.ca**.
2) Use your UBC network user id/password (not CWL) to login.
3) Connect to MySQL with: `mysql -u <userid> -p`
   - user id (first letter first name+7 letters last name)
   - password (student number)
4) Use `workson` database or `db_userid`.
5) Run SQL commands.

Connecting to MySQL – Command Line (2)



Connecting to MySQL – GUI using SQuirreL

1) SQuirreL SQL is a GUI for SQL querying of multiple databases. Download at: **http://squirrel-sql.sourceforge.net/**

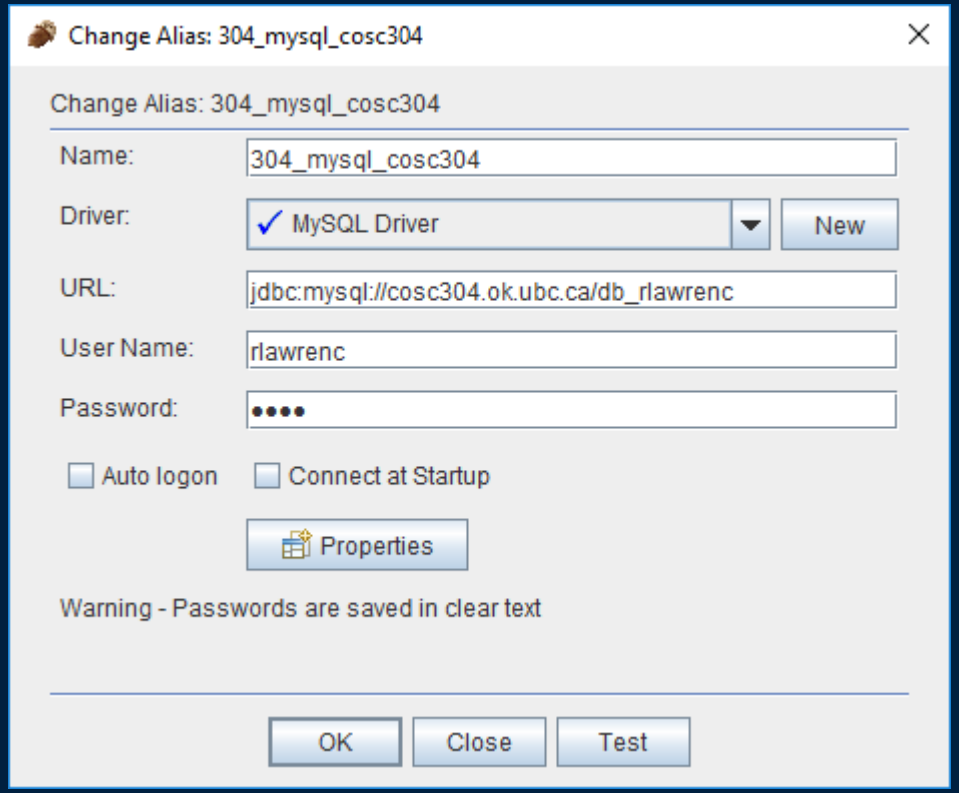
2) Install Java. Install SQuirreL.

3) Download the MySQL JDBC driver and put in lib folder of SQuirreL.

4) Create an alias to connect to SQuirreL.

5) Connect and run queries.

Connecting to MySQL – GUI using SQuirreL (2)



Try it: CREATE TABLE

Question: Create a table called `mydata` that has three fields:

- `num` – that will store a number (use `int` as data type)
- `message` – that will store a string up to 50 characters (`varchar` data type)
- `amount` – that stores a decimal number with 8 total digits and 2 decimal digits (`decimal` data type)

Connect to the sample MySQL database or use the web site **sqlfiddle.com** to try your table creation.

Adding Data using INSERT

Insert a row using the INSERT command:

```
INSERT INTO emp VALUES ('E9','S. Smith','1975-03-05',
                        'SA',60000,'E8','D1')
```

Fields: eno, ename, bdate, title, salary, supereno, dno

If you do not give values for all fields in the order they are in the table, you must list the fields you are providing data for:

```
INSERT INTO emp(eno, ename, salary)
       VALUES ('E9','S. Smith',60000)
```

Note: 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:

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

Example:

```
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:

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

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

```
INSERT INTO tmpTable
    SELECT eno, ename
    FROM emp
    WHERE title = 'EE'
```

Try it: INSERT

Question: Using the `mydata` table insert three rows:

- `(1, 'Hello', 99.45)`
- `(2, 'Goodbye', 55.99)`
- `(3, 'No Amount')`

Connect to the sample MySQL database or use the web site **sqlfiddle.com** to try.

UPDATE Statement

Updating existing rows using the `UPDATE` statement. Examples:

- 1) Increase all employee salaries by 10%.

```
UPDATE emp SET salary = salary*1.10;
```

- 2) Increase salary of employee E2 to $1 million and change his name:

```
UPDATE emp SET salary = 1000000, name='Rich Guy'
WHERE eno = 'E2';
```

Notes:

- May change (`SET`) more than one value at a time. Separate by commas.
- Use `WHERE` to filter only the rows to update.

Try it: UPDATE

Question: Using the `mydata` table and the three rows previously inserted do these updates:

- Update all `amount` fields to be `99.99`.
- Update the `num` field and set it to 10 for the record with `num = 1`.
- Update the `message` field to `'Changed'` for the record with `num = 2`.

Connect to the sample MySQL database or use the web site **sqlfiddle.com** to try.

DELETE Statement

Rows are deleted using the DELETE statement. Examples:

- 1) Fire everyone in the company.

```
DELETE FROM emp;
```

- 2) Fire everyone making over $35,000.

```
DELETE FROM emp
WHERE salary > 35000;
```

Try it: DELETE

Question: Using the `mydata` table and the three rows previously inserted do these deletes:

- Delete the row with `num = 1`.
- Delete the row(s) with `message > 'C'`.
- Delete all rows.

Connect to the sample MySQL database or use the web site **sqlfiddle.com** to try your DELETE statements.

INSERT Question

Question: How many of the following statements are **TRUE**?

1) You must always specify the fields being inserted with `INSERT` statement.

2) If you list the fields, the fields must be in the same order as the table.

3) If you do not provide a value for a number field, it will default to 1.

4) Number data items are enclosed in single quotes.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

UPDATE Question

Question: How many of the following statements are **TRUE**?

1) You may update more than one row at a time.
2) If the `UPDATE` has no `WHERE` clause, it updates all rows.
3) You may update zero or more rows using a`UPDATE` statement.
4) `UPDATE` may change more than one data value (column) in a row.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

DELETE Question

Question: How many of the following statements are **TRUE**?

1) A `DELETE` with no `WHERE` clause will delete all rows.
2) The `DELETE` keyword is case-sensitive.
3) It is possible to `DELETE` zero or more rows using a `WHERE` clause.
4) A `DELETE` statement may delete zero rows when executed.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Practice Questions

Relational database schema:

```
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%.

Conclusion

SQL is the standard query language for databases.

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

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.

Objectives

- Recognize valid and invalid identifiers.
- Explain the key types of constraints and how to enforce them: required (not null) data, domain constraints, entity integrity, referential integrity.
- Write a `CREATE TABLE` statement given a high-level description.
- List what `ALTER TABLE` can and cannot do.
- Remove a table using `DROP TABLE`.
- Create an index on fields of a table.
- Explain how an index helps improve query time.
- Write `INSERT`, `DELETE`, and `UPDATE` commands.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

SQL: `SELECT` Statement

select-project-join-order by

COSC 304 – Introduction to Database Systems

SQL Queries using SELECT

A query in SQL has the form:

```
SELECT  (list of columns or expressions)
FROM    (list of tables)
WHERE   (filter conditions)
GROUP BY  (columns)
ORDER BY  (columns)
```

Notes:

- 1) Separate the list of columns/expressions and list of tables by **commas**.
- 2) The "`*`" is used to select all columns.
- 3) Only `SELECT` required. `FROM`, `WHERE`, `GROUP BY`, `ORDER BY` are optional.

SQL and Relational Algebra

The `SELECT` statement can be mapped directly to relational algebra.

SELECT $A_1, A_2, \ldots, A_n$
FROM $R_1, R_2, \ldots, R_m$
WHERE P

is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

Example Database - WorksOn

emp Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

workson Table

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

proj Table

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

dept Table

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

SQL: Retrieving Only Some of the Columns

The ***projection operation*** creates a new table that has some of the columns of the input table. In SQL, provide the table in the `FROM` clause and the fields in the output in the `SELECT`.

Example: Return only the `eno` field from the `emp` table:

```
SELECT  eno
FROM    emp
```

`emp` Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Result

eno
E1
E2
E3
E4
E5
E6
E7
E8

SQL Projection Examples

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT eno,ename
FROM   emp
```

eno	ename
E1	J. Doe
E2	M. Smith
E3	A. Lee
E4	J. Miller
E5	B. Casey
E6	L. Chu
E7	R. Davis
E8	J. Jones

```
SELECT title
FROM   emp
```

title
EE
SA
ME
PR
SA
EE
ME
SA

Notes: 1) Duplicates are not removed during SQL projection.
2) `SELECT *` will return all columns.

Projection Question

Question: Given this table and the query:

```
SELECT eno, ename, salary
FROM   emp
```

How many columns are returned?

A) 0

B) 1

C) 2

D) 3

E) 4

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Projection Question #2

Question: Given this table and the query:

```
SELECT  salary
FROM    emp
```

How many rows are returned?

A) 0

B) 2

C) 4

D) 8

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Duplicates in SQL

One major difference between SQL and relational algebra is that relations in SQL are ***bags*** instead of sets.

- It is possible to have two or more identical rows in a relation.

Consider the query: Return all titles of employees.

```
SELECT title
FROM   emp
```

emp Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

Result

title
EE
SA
ME
PR
SA
EE
ME
SA

Duplicates in SQL - `DISTINCT` clause

To remove duplicates, use **`DISTINCT`** clause in the SQL statement:

```
SELECT DISTINCT title
FROM   emp
```

Result

title
EE
SA
ME
PR

DISTINCT Question

Question: Given this table and the query:

```
SELECT DISTINCT a, b
FROM   R
```

R Table

a	b	c
1	1	A
1	2	B
1	1	A
3	1	C
2	2	A
2	2	B

How many rows are returned?

A) 1

B) 3

C) 4

D) 6

Try it: SQL `SELECT` and Projection

Question: Using the `proj` table, write these three queries:

1) Show all rows and all columns.
2) Show all rows but only the `pno` column.
3) Show all rows but only the `pno` and `budget` columns.
4) Show unique `budget` values.

Retrieving Only Some of the Rows

The ***selection operation*** creates a new table with some of the rows of the input table. A condition specifies which rows are in the new table. The condition is similar to an `if` statement.

Example: Return the projects in department `'D2'`:

```
SELECT pno, pname, budget, dno
FROM   proj
WHERE  dno = 'D2'
```

`proj` Table

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Result

pno	pname	budget	dno
P2	DB Develop	135000	D2
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Algorithm: Scan each tuple and check if matches condition in WHERE clause.

Selection Conditions

The condition in a selection statement specifies which rows are included. It has the general form of an if statement.

The condition may consist of attributes, constants, comparison operators (<, >, =, !=, <=, >=), and logical operators (`AND`, `OR`, `NOT`).

SQL Selection Examples

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT *
FROM   emp
WHERE  title = 'EE'
```

eno	ename	title	salary
E1	J. Doe	EE	30000
E6	L. Chu	EE	30000

```
SELECT eno, ename, title, salary
FROM   emp
WHERE  salary > 35000 OR
       title = 'PR'
```

eno	ename	title	salary
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question

Question: Given this table and the query:

```
SELECT  *
FROM    emp
WHERE   title='SA'
```

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question #2

Question: Given this table and the query:

```
SELECT  *
FROM    emp
WHERE   salary > 50000 or title='PR'
```

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Try it: SQL SELECT and Filtering Rows

Question: Write these queries:

1) Return all projects with `budget` > $`250000`.

2) Show the `pno` and `pname` for projects in `dno = 'D1'`.

3) Show `pno` and `dno` for projects in `dno='D1'` or `dno='D2'`.

4) Return the employee numbers who make less than $`30000`.

5) Return list of `workson` responsibilities (`resp`) with no duplicates.

6) Return the employee (names) born after July 1, 1970 that have a `salary > 35000` and have a title of `'SA'` or `'PR'`.

Joins for Combining Tables

A ***join*** combines two tables by matching columns in each table.

`workson` Table

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

`proj` Table

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM   workson JOIN proj
   ON  workson.pno = proj.pno
```

eno	pno	resp	dur	proj.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000

Join Details and Examples

Listing multiple tables in the `FROM` clause separated by commas creates a cross product of tables. Must specify `JOIN` and `ON` or provide join condition in `WHERE` clause.

Goal: For each employee, return their name and department name.

Wrong! Cross Product

```
SELECT ename, dname
FROM   emp, dept
```

Correct! `JOIN-ON` Clause

```
SELECT ename, dname
FROM   emp JOIN dept
       ON emp.dno = dept.dno
```

Correct! Join in `WHERE`

```
SELECT ename, dname
FROM   emp, dept
WHERE  emp.dno = dept.dno
```

Correct! Order does not matter.

```
SELECT ename, dname
FROM   dept JOIN emp
       ON emp.dno = dept.dno
```

Join Query with Selection Example

You can use join, selection, and projection in the same query.

- Recall: Projection returns columns listed in `SELECT`, selection filters out rows using condition in `WHERE`, and join combines tables in `FROM` using a condition.

Example: Return the employee names who are assigned to the 'Management' department.

Projection: only name field in result

```
SELECT  ename
FROM    emp JOIN dept
          ON emp.dno = dept.dno
WHERE   dname = 'Management'
```

tables in query joined together

Selection: filter rows

Result

ename
R. Davis
J. Jones

Three Table Join Query Example

Return all projects who have an employee working on them whose title is `'EE'`:

```
SELECT pname
FROM   emp JOIN workson ON emp.eno = workson.eno
           JOIN proj ON workson.pno = proj.pno
WHERE  emp.title = 'EE'
```

Or:

```
SELECT pname
FROM   emp, proj, workson
WHERE  emp.title = 'EE' and workson.eno = emp.eno
        and workson.pno = proj.pno
```

Note: Parentheses `()` can be used to specify order of joins when using `JOIN-ON`.

SQL Query Question

Question: What query would return the name and salary of employees working on project `'P3'`:

A)
```
SELECT ename, salary
FROM emp, workson
WHERE emp.eno = workson.eno and pno = 'P3'
```

B)
```
SELECT ename, salary
FROM emp, workson, proj
WHERE emp.eno = workson.eno and pno = "P3"
```

Ordering Result Data

The query result returned is not ordered on any column by default. We can order the data using the **ORDER BY** clause:

```
SELECT    ename, salary, bdate
FROM      emp
WHERE     salary > 30000
ORDER BY  salary DESC, ename ASC;
```

- 'ASC' sorts the data in ascending order, and 'DESC' sorts it in descending order. The default is 'ASC'.
- The order of sorted attributes is significant. The first column specified is sorted on first, then the second column is used to break any ties, etc.

LIMIT and OFFSET

If you only want the first *N* rows, use a **LIMIT** clause:

```
SELECT   ename, salary FROM emp
ORDER BY salary DESC LIMIT 5
```

To start from a row besides the first, use **OFFSET**:

```
SELECT   eno, salary FROM emp
ORDER BY eno DESC
LIMIT 3  OFFSET 2
```

- `LIMIT` improves performance by reducing amount of data processed and sent by the database system.
- `OFFSET` 0 is first row, so `OFFSET` 2 would return the 3rd row.
- `LIMIT`/`OFFSET` syntax support differs between databases.

Try it: SQL SELECT with Joins and Ordering

Question: Write these queries:

1) Return all projects with `budget` < $`500000` sorted by `budget` descending.
2) List only the top 5 employees by `salary` descending. Show only their `name` and `salary`.
3) List each project `pno`, `dno`, `pname`, and `dname` ordered by `dno` ascending then `pno` ascending. Only show projects if department `name > 'D'`. Note: This query will require a join.
4) Return the list of project names for the department with name `'Consulting'`.
5) Return `workson` records (`eno`, `pno`, `resp`, `hours`) where project `budget` is > $`50000` and hours worked is < `20`.
6) **Challenge:** Return a list of all department names, the names of the projects of that department, and the name of the manager of each department.

Calculated Fields

Expressions are allowed in `SELECT` clause to perform calculations.

- When an expression is used to define an attribute, the DBMS gives the attribute a unique name such as `col1`, `col2`, etc.

Example: Return how much employee 'A. Lee' will get paid for his work on each project.

```
SELECT ename, pname, salary/52/5/8*hours
FROM   emp JOIN workson ON emp.eno=workson.eno
         JOIN proj ON workson.pno=proj.pno
WHERE  ename='A. Lee'
```

Result

ename	pname	col3
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Renaming and Aliasing

Often it is useful to rename an attribute in the final result (especially when using calculated fields). Renaming is accomplished using the keyword **AS**:

```
SELECT ename, pname, salary/52/5/8*hours AS pay
FROM   emp JOIN workson ON emp.eno = workson.eno
           JOIN proj ON proj.pno = workson.pno
WHERE  ename = 'A. Lee'
```

Result

ename	pname	pay
A. Lee	Budget	192.31
A. Lee	Maintenance	923.08

Note: AS keyword is optional.

Renaming and Aliasing Tables

Renaming is also used when two or more copies of the same table are in a query. Using ***aliases*** allows you to uniquely identify what table you are talking about.

Example: Return the employees and their managers where the managers make less than the employee.

```
SELECT E.ename, M.ename
FROM   emp as E JOIN emp as M ON E.supereno = M.eno
WHERE  E.salary > M.salary
```

Advanced Conditions - BETWEEN

Used when the condition in the WHERE clause will request tuples where one attribute value must be in a *range* of values.

Example: Return the employees who make at least $20,000 and less than or equal to $45,000.

```
SELECT ename
FROM   emp
WHERE  salary >= 20000 and salary <= 45000
```

We can use the keyword **BETWEEN** instead:

```
SELECT ename
FROM   emp
WHERE  salary BETWEEN 20000 and 45000
```

Advanced Conditions - LIKE

For strings, the **LIKE** operator is used to search for partial matches.

- Partial string matches are specified by using either "%" that replaces zero or more characters or underscore "_" that replaces a single character.

Example: Return all employee names that start with 'A'.

```
SELECT ename
FROM   emp
WHERE  ename LIKE 'A%'
```

Example: Return all employee names who have a first name that starts with 'J' and whose last name is 3 characters long.

```
SELECT ename
FROM   emp
WHERE  ename LIKE 'J. _ _ _'
```

Performance Concerns of `LIKE`

Warning: Do not use the `LIKE` operator if you do not have to.

It is often an inefficient operation as the DBMS may not be able to optimize lookup using `LIKE` as it can for equal (=) comparisons. The result is the DBMS often has to examine ALL TUPLES in the relation.

In almost all cases, adding indexes will ***not*** increase the performance of `LIKE` queries because the indexes cannot be used.

- Most indexes are implemented using B-trees that allow for fast equality searching and efficient range searches.

Advanced Conditions - IN

To specify that an attribute value should be in a given set of values, the **IN** keyword is used.

- Example: Return employees who are in one of the departments {'D1', 'D2', 'D3'}.

```
SELECT ename
FROM   emp
WHERE  dno IN ('D1','D2','D3')
```

Note that this is equivalent to using OR:

```
SELECT ename
FROM   emp
WHERE  dno = 'D1' OR dno = 'D2' OR dno = 'D3'
```

We will see more uses of `IN` and `NOT IN` with nested subqueries.

Advanced Conditions - NULL

Remember NULL indicates that an attribute does not have a value. To determine if an attribute is NULL, we use the clause **IS NULL**.

- Note that you should not test NULL values using = and <>.

Example: Return all employees who are not in a department.

```
SELECT ename
FROM   emp
WHERE  dno IS NULL
```

Example: Return all departments that have a manager.

```
SELECT dname
FROM   dept
WHERE  mgreno IS NOT NULL
```

Set Operations

Union, intersection, and difference combine results of two queries.

- `UNION`, `INTERSECT`, `EXCEPT`, `UNION ALL` (returns all rows)
- ***Union-compatible***: same # of attributes and compatible data types. Do not need to have the same name.

Example: Return the employees who are either directly supervised by 'R. Davis' or directly supervised by 'M. Smith'.

```
(SELECT E.ename
FROM   emp as E JOIN emp as M ON E.supereno = M.eno
WHERE  M.ename='R. Davis')
UNION
(SELECT E.ename
FROM   emp as E JOIN emp as M ON E.supereno = M.eno
WHERE  M.ename='M. Smith')
```

Set Operations Examples

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

workson Table

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

```
(SELECT eno FROM emp)
UNION
(SELECT eno FROM workson)
```

eno
E1
E2
E3
E4
E5
E6
E7
E8

```
(SELECT eno FROM emp)
EXCEPT
(SELECT eno FROM workson)
```

eno
E4
E8

```
(SELECT eno FROM emp)
INTERSECT
(SELECT eno FROM workson)
```

eno
E1
E2
E3
E5
E6
E7

Question: What is the meaning of this query?

```
(SELECT eno FROM workson)
EXCEPT
(SELECT eno FROM emp)
```

Set Operations Union-compatible Question

Question: Two tables have the same number of fields in the same order with the same types, but the names of some fields are different.
True or false: The two tables are union-compatible.

A) true

B) false

SELECT INTO

The result of a select statement can be stored in a temporary table using the **INTO** keyword.

```
SELECT E.ename
INTO   davisMgr
FROM   emp as E JOIN emp as M ON E.supereno = M.eno
WHERE  M.ename = 'R. Davis'
```

SQL Querying with NULL and LIKE

Question: What query would return the department names that do not have a manager or contain `'ent'`.

A)
```
SELECT dname
FROM   dept
WHERE mgreno = NULL OR dname LIKE '_ent'
```

B)
```
SELECT dname
FROM   dept
WHERE mgreno IS NULL OR dname LIKE '%ent%'
```

Try it: SQL `SELECT` Expressions, `LIKE`, `IS NULL`

Question: Write these queries:

1) Calculate the monthly salary for each employee.

2) List all employee names who do not have a supervisor.

3) List all employee names where the employee's name contains an `'S'` and workson responsibility that ends in `'ER'`.

4) Return the list of employees (names) who make less than their managers and how much less they make.

5) Return only the top 3 project budgets in descending order.

Try it: SQL SELECT Set Operations, ORDER BY

Question: Write these queries:

1) Return the list of employees sorted by salary (desc) and then title (asc).

2) Return the employees (names) who either manage a department or manage another employee.

3) Return the employees (names) who manage an employee but do not manage a department.

4) Give a list of all employees who work on a project for the 'Management' department ordered by project number (asc).

5) **Challenge:** Return the projects (names) that have their department manager working on them.

Conclusion

The **SELECT** statement is used to query data and combines the operations of selection, projection, and join.

SELECT features covered:

- **SELECT** clause to provide column list and calculate expressions
- **DISTINCT** clause to eliminate duplicates
- **FROM** clause to list tables
- **JOIN ON** syntax to join tables on a join condition
- **UNION, EXCEPT, INTERSECT** set operations
- **IS NULL** for checking if column value is null
- **ORDER BY** clause for sorting output
- **LIMIT/OFFSET** for only reducing a part of the result set

Objectives

Translate English questions into SQL queries that may require:

- `SELECT-FROM-WHERE` syntax for selection, projection, and join
- renaming and aliasing including queries with multiple copies of the same relation
- `ORDER BY`
- `LIMIT/OFFSET`
- `DISTINCT` to eliminate duplicates
- `UNION`, `EXCEPT`, `INTERSECT` set operations
- `IS NULL` or `IS NOT NULL`
- `LIKE` string pattern matching

Read SQL queries to determine their output and English meaning


UBC
THE UNIVERSITY OF BRITISH COLUMBIA

SQL: `GROUP BY`, Aggregation, Subqueries, Outer Joins

COSC 304 – Introduction to Database Systems

Aggregate Queries and Functions

Several queries cannot be answered using the simple form of the `SELECT` statement. These queries require a summary calculation to be performed. Examples:

- What is the maximum employee salary?
- What is the total number of hours worked on a project?
- How many employees are there in department 'D1'?

To answer these queries requires the use of aggregate functions. These functions operate on a single column of a table and return a single value.

Aggregate Functions

Five common aggregate functions are:

- `COUNT` - returns the # of values in a column
- `SUM` - returns the sum of the values in a column
- `AVG` - returns the average of the values in a column
- `MIN` - returns the smallest value in a column
- `MAX` - returns the largest value in a column

Notes:

- 1) `COUNT`, `MAX`, and `MIN` apply to all types of fields, whereas `SUM` and `AVG` apply to only numeric fields.
- 2) Except for `COUNT(*)` all functions ignore nulls. `COUNT(*)` returns the number of rows in the table.
- 3) Use `DISTINCT` to eliminate duplicates.

Aggregate Function Example

Return the number of employees and their average salary.

```
SELECT COUNT(eno) AS numEmp, AVG(salary) AS avgSalary
FROM   emp
```

Result

numEmp	avgSalary
8	38750

GROUP BY Clause

Aggregate functions are most useful when combined with the `GROUP BY` clause. The **`GROUP BY`** clause groups the tuples based on the values of the attributes specified.

When used in combination with aggregate functions, the result is a table where each tuple consists of unique values for the group by attributes and the result of the aggregate functions applied to the tuples of that group.

GROUP BY Example

For each employee title, return the number of employees with that title, and the minimum, maximum, and average salary.

```
SELECT   title, COUNT(eno) AS numEmp,
         MIN(salary) as minSal,
         MAX(salary) as maxSal, AVG(salary) AS avgSal
FROM     emp
GROUP BY title
```

Result

title	numEmp	minSal	maxSal	avgSal
EE	2	30000	30000	30000
SA	3	50000	50000	50000
ME	2	40000	40000	40000
PR	1	20000	20000	20000

GROUP BY Clause Rules

There are a few rules for using the `GROUP BY` clause:

- 1) A column name cannot appear in the `SELECT` part of the query unless it is part of an aggregate function or in the list of group by attributes.
  - Note that the reverse is allowed: a column can be in the `GROUP BY` without being in the `SELECT` part.

- 2) Any `WHERE` conditions are applied before the `GROUP BY` and aggregate functions are calculated.

- 3) You can group by multiple attributes. To be in the same group, all attribute values must be the same.

GROUP BY Question

Question: Given this table and the query:

```
SELECT title, SUM(salary)
FROM   emp
GROUP BY title
```

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

How many rows are returned?

A) 1

B) 2

C) 4

D) 8

GROUP BY Question #2

Question: Given this table and the query:

```
SELECT resp, pno, SUM(hours)
FROM   workson
WHERE  hours > 10
GROUP BY resp, pno
```

How many rows are returned?

A) 9 **B)** 7 **C)** 5 **D)** 1 **E)** 0

workson Table

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

HAVING Clause

The **HAVING** clause is applied ***AFTER*** the `GROUP BY` clause and aggregate functions are calculated.

It is used to filter out entire *groups* that do not match certain criteria.

The **HAVING** clause can contain any condition that references aggregate functions and the group by attributes themselves.

- However, any conditions on the `GROUP BY` attributes should be specified in the `WHERE` clause if possible due to performance reasons.

HAVING Example

Return the title and number of employees of that title where the number of employees of the title is at least 2.

```
SELECT    title, COUNT(eno) AS numEmp
FROM      emp
GROUP BY  title
HAVING    COUNT(eno) >= 2
```

Result

title	numEmp
EE	2
SA	3
ME	2

GROUP BY/HAVING Example

For employees born after December 1, 1965, return the average salary by department where the average is > 40,000.

```
SELECT    dname, AVG(salary) AS avgSal
FROM      emp JOIN dept ON emp.dno = dept.dno
WHERE     emp.bdate > DATE '1965-12-01'
GROUP BY  dname
HAVING    AVG(salary) > 40000
```

Step #1: Perform Join and Filter in WHERE clause

eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

GROUP BY/HAVING Example (2)

Step #2: GROUP BY on dname

eno	ename	bdate	title	salary	supereno	dno	dname	mgreno
E2	M. Smith	1966-06-04	SA	50000	E5	D3	Accounting	E5
E5	B. Casey	1971-12-25	SA	50000	E8	D3	Accounting	E5
E3	A. Lee	1966-07-05	ME	40000	E7	D2	Consulting	E7
E7	R. Davis	1977-09-08	ME	40000	E8	D1	Management	E8
E8	J. Jones	1972-10-11	SA	50000	null	D1	Management	E8

Step #3: Calculate aggregate functions

dname	avgSal
Accounting	50000
Consulting	40000
Management	45000

Step #4: Filter groups using HAVING clause

dname	avgSal
Accounting	50000
Management	45000

GROUP BY Examples

Return the average budget per project:

```
SELECT AVG(budget) FROM proj
```

Return the average # of hours worked on each project:

```
SELECT pno, AVG(hours) FROM workson GROUP BY pno
```

Return the departments that have projects with at least 2 'EE's working on them:

```
SELECT    DISTINCT proj.dno
FROM      proj JOIN workson ON workson.pno = proj.pno
               JOIN emp ON workson.eno=emp.eno
WHERE     emp.title = 'EE' GROUP BY  proj.dno, proj.pno
HAVING    COUNT(*) >=2
```

GROUP BY/HAVING
Multi-Attribute Example

Return the employee number, department number and hours the employee worked per department where the hours is >= 10.

```
SELECT   W.eno, D.dno, SUM(hours)
FROM     workson AS W JOIN proj AS P ON W.pno = P.pno
         JOIN dept AS D ON P.dno = D.dno
GROUP BY W.eno, D.dno
HAVING   SUM(hours) >= 10
```

Result:

eno	dno	SUM(hours)
E1	D1	12
E2	D1	24
E3	D2	48
E3	D3	10
E4	D2	18
E5	D2	24
E6	D2	48
E7	D3	36

Question:

1) How would you only return records for departments D2 and D3?

SQL Querying with GROUP BY

Question: Of the following queries, select one which is **invalid**.

A)
```
SELECT dname
FROM   dept
GROUP BY dno
```

B)
```
SELECT COUNT(*)
FROM   dept
```

C)
```
SELECT dno, COUNT(*)
FROM   dept
```

D)
```
SELECT dno, COUNT(*)
FROM   dept  WHERE mgreno > 'A'
GROUP BY dno, dname
```

Try it: SQL GROUP BY Practice Questions

Question: Write these queries:

1) Return the highest salary of any employee.

2) Return the smallest project budget.

3) Return the department number and average budget for its projects.

4) For each project, return its name and the total number of hours employees have worked on it.

5) For each employee, return the total number of hours they have worked. Only show employees with more than 30 hours.

Subqueries

SQL allows a single query to have multiple subqueries nested inside of it. This allows for more complex queries to be written.

When queries are nested, the outer statement determines the contents of the final result, while the inner `SELECT` statements are used by the outer statement (often to lookup values for `WHERE` clauses).

```
SELECT    ename, salary, bdate
FROM      emp
WHERE     salary > (SELECT AVG(salary) FROM emp)
```

A subquery can be in the `SELECT`, `FROM`, `WHERE` or `HAVING` clause.

Types of Subqueries

There are three types of subqueries:

- 1) ***scalar subqueries*** - return a single value. Often value is then used in a comparison.
  - If query is written so that it expects a subquery to return a single value, and if it returns multiple values or no values, a run-time error occurs.
- 2) ***row subquery*** - returns a single row which may have multiple columns.
- 3) ***table subquery*** - returns one or more columns and multiple rows.

Scalar Subquery Examples

Return the employees that are in the 'Accounting' department:

```
SELECT ename
FROM   emp
WHERE  dno = (SELECT dno FROM dept
              WHERE dname = 'Accounting')
```

Return all employees who work more hours than average on a single project:

```
SELECT ename
FROM   emp JOIN workson ON workson.eno = emp.eno
WHERE  workson.hours > (SELECT AVG(hours) FROM workson)
```

Table Subqueries

A table subquery returns a relation. There are several operators that can be used:

- `EXISTS` *R* - true if *R* is not empty
- *s* `IN` *R* - true if *s* is equal to one of the values of *R*
- *s* > `ALL` *R* - true if *s* is greater than **every** value in *R*
- *s* > `ANY` *R* - true if *s* is greater than **any** value in *R*

Notes:

- 1) Any of the comparison operators (<, <=, =, etc.) can be used.
- 2) The keyword `NOT` can proceed any of the operators.
  - Example: *s* `NOT IN` *R*

Table Subquery Examples

Return all departments who have a project with a budget greater than $300,000:

```
SELECT dname FROM dept WHERE dno IN
(SELECT dno FROM proj WHERE budget > 300000)
```

Return all projects that 'J. Doe' works on:

```
SELECT pname FROM proj WHERE pno IN
      (SELECT pno FROM workson WHERE eno =
         (SELECT eno FROM emp WHERE ename = 'J. Doe'))
```

EXISTS Example

The EXISTS function is used to check whether the result of a nested query is empty or not.

- EXISTS returns true if the nested query has 1 or more tuples.

Example: Return all employees who have the same name as someone else in the company.

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT * FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

ANY and ALL Example

ANY means that any value returned by the subquery can satisfy the condition.

ALL means that all values returned by the subquery must satisfy the condition.

Example: Return the employees who make more than all the employees with title 'ME' make.

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary FROM emp
                     WHERE title = 'ME')
```

Subquery Syntax Rules

1) The `ORDER BY` clause may not be used in a subquery.

2) The number of attributes in the `SELECT` clause in the subquery must match the number of attributes compared to with the comparison operator.

3) Column names in a subquery refer to the table name in the `FROM` clause of the subquery by default. You must use aliasing if you want to access a table that is present in both the inner and outer queries.

Correlated Subqueries

Most queries involving subqueries can be rewritten so that a subquery is not needed.

- This is normally beneficial because query optimizers may not do a good job at optimizing queries containing subqueries.

A nested query is ***correlated*** with the outside query if it must be re-computed for every tuple produced by the outside query. Otherwise, it is ***uncorrelated***, and the nested query can be converted to a non-nested query using joins.

A nested query is correlated with the outer query if it contains a reference to an attribute in the outer query.

Correlated Subquery Example

Return all employees who have the same name as another employee:

```
SELECT ename
FROM   emp as E
WHERE  EXISTS (SELECT eno FROM emp as E2
               WHERE E.ename = E2.ename AND
                     E.eno <> E2.eno)
```

A more efficient solution with joins:

```
SELECT E.ename
FROM   emp as E JOIN emp as E2 ON
       E.ename = E2.ename AND E.eno <> E2.eno
```

Types of Joins

A ***equijoin*** only contains the equality operator (=).

- e.g. `workson JOIN Proj ON workson.pno=proj.pno`

A ***natural join*** is equijoin of two tables with commonly named fields.

- Removes the "extra copies" of the join attributes.
- The attributes must have the same name in both relations.
- e.g. `workson NATURAL JOIN Proj`

Left outer join – contains all tuples of first table even if no match

Right outer join – contains all tuples of second table even if no match

Full outer join – contains all tuples of either table even if no match.
For a tuple that does not have a match, missing fields are `NULL`.

Specifying Outer Joins in SQL

Types: `NATURAL JOIN`, `FULL OUTER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, `INNER JOIN`, `JOIN`

- The keyword "`outer`" can be omitted for outer joins. Same with "`inner`".

Example: Return all departments (even those without projects) and their projects.

```
SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj ON dept.dno = proj.dno
```

```
SELECT dname, pname
FROM   dept LEFT OUTER JOIN proj USING (dno)
```

```
SELECT dname, pname
FROM   dept NATURAL LEFT JOIN proj
```

Equijoin Example

workson Table

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj table

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM   workson JOIN proj
   ON  workson.pno = proj.pno
```

eno	pno	resp	dur	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000

What is the meaning of this join?

Natural Join Example

`workson` Table

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

`proj` table

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM   workson NATURAL JOIN proj
```

eno	pno	resp	dur	pname	budget
E1	P1	Manager	12	Instruments	150000
E2	P1	Analyst	24	Instruments	150000
E2	P2	Analyst	6	DB Develop	135000
E3	P4	Engineer	48	Maintenance	310000
E5	P2	Manager	24	DB Develop	135000
E6	P4	Manager	48	Maintenance	310000
E7	P3	Engineer	36	CAD/CAM	250000
E7	P4	Engineer	23	Maintenance	310000

Natural join is performed by comparing *pno* in both tables.

Right Outer Join Example

workson Table

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj table

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM workson RIGHT OUTER JOIN proj P
  USING (pno)
```

eno	pno	resp	dur	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000
null	null	null	null	P5	CAD/CAM	500000

Outer Join Question

Question: Given this table and the query:

```
SELECT *
FROM workson LEFT OUTER JOIN proj P
   ON workson.pno = proj.pno
```

How many rows are returned?

A) 10

B) 9

C) 8

D) 7

workson

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P4	Engineer	36
E7	P4	Engineer	23

proj

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

Subqueries in FROM Clause

Subqueries are used in the FROM clause to produce temporary table results for use in the current query.

Example: Return the departments that have an employee that makes more than $40,000.

```
SELECT dname
FROM   Dept D, (SELECT ename, dno FROM Emp
                WHERE salary > 40000) E
WHERE D.dno = E.dno
```

- Note: The alias for the derived table is required.

SQL Querying with Subqueries

Question: What query below is equivalent to:

```
SELECT ename
FROM   emp as E
WHERE  salary > ALL (SELECT salary
                     FROM emp WHERE title = 'EE')
```

A)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT MAX(salary) FROM emp
                 WHERE title = 'EE')
```

B)

```
SELECT ename
FROM   emp as E
WHERE  salary > (SELECT SUM(salary) FROM emp
                 WHERE title = 'EE')
```

SQL Functions

Databases have many built-in functions that can be used when writing queries. Syntax and support varies between systems.

- Date: DATEDIFF, YEAR, GETDATE
- String: CONCAT, UPPER, LEFT, SUBSTRING
- Logical: CASE, IIF, ISNULL
- Aggregate: SUM, COUNT, AVG
- Note: Case-insensitive function names.

Example:

```
SELECT eno, UPPER(ename),
  CASE title WHEN 'EE' THEN 'Engineer'
  WHEN 'SA' THEN 'Admin' ELSE 'Other' END as role,
  year(bdate) as birthYear
FROM emp
WHERE salary * 2 > 60000
```

Try-it: Subquery Practice Questions

Question: Write these queries:

1) List all departments that have at least one project.

2) List the employees who are not working on any project.

3) List the employees with title 'EE' that make more than all employees with title 'PR'.

4) Find all employees who work on some project that 'J. Doe' works on.

SQL Queries using `SELECT`

A query in SQL has the form:

`SELECT` (list of columns or expressions)

`FROM` (list of tables)

`WHERE` (filter *conditions*)

`GROUP BY` (columns)

`HAVING` (group filter *conditions*)

`ORDER BY` (columns)

`LIMIT` (count) `OFFSET` (start)

Conclusion

SQL is the standard language for querying relational databases.

The **`SELECT`** statement is used to query data and combines the relational algebra operations of selection, projection, and join into one statement

- There are often many ways to specify the same query.

Queries may involve aggregate functions, outer joins, and subqueries.

Objectives

- Write SQL queries containing aggregate functions and calculated fields.
- Write SQL queries requiring nested subqueries and the use of the appropriate operators such as comparison operators for single value subqueries, `IN`, `NOT IN`, `ANY`, `ALL` for table result subqueries, and `EXISTS` and `NOT EXISTS` for multiple result subqueries which may or may not contain results.
- Lookup documentation on SQL functions supported by a particular database and use them as required in queries.
- Explain the purpose of `OUTER` and `NATURAL` joins and use them for queries.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA