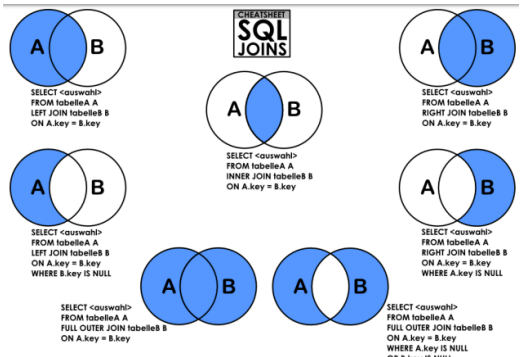
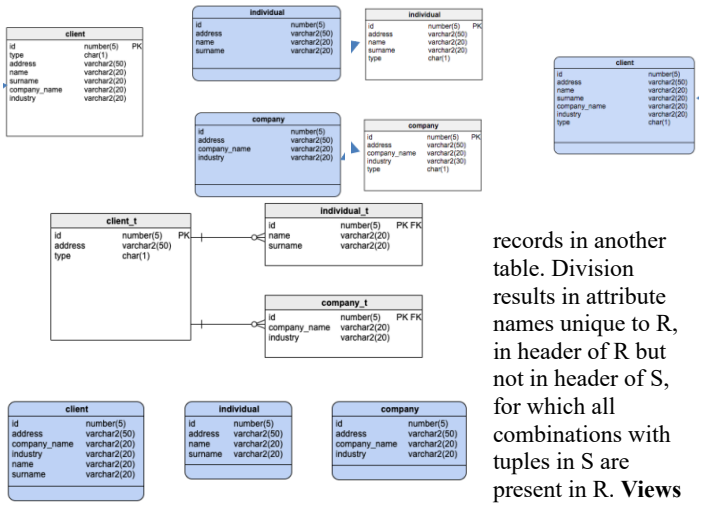
Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Entity sets, relationship sets, attributes. **ER** model advantages: conceptually simple, better visual representation, effective communication, integrated with relational model, easy conversion to data model; disadvantages: limited constraints, loss of information, limited relationship representation, no data manipulation representation, no notation industry standard. **Relational** model: relation, tuple/row, column/attribute. Domain: set of allowed values. Atomic: indivisible attribute. Null is a member of every domain. Weak entity can't be uniquely identified by attributes alone, must use FK in conjunction with other attributes to create PK. **Database** schema is logical structure, instance is snapshot of data at given instant. **Key** K is the superkey of relation R if values are sufficient to identify a unique tuple {id}, {id,name}. K is candidate key if minimal {id}, one selected to be primary key. Foreign key references relation. Composite key, neither column is unique key alone. Alternate (unique keys that are not primary), natural (formed from values inherently existing in real world), surrogate (artificial system-generated identifier), unique. **Relational algebra** select $\sigma_{id="1"}$(student), and∧ or∨ not¬, project π_{id}(student), union ∪, set difference −, cartesian product (instructor × teaches), rename ρ. **SQL** DML provides the ability to query, insert, delete, modify tuples, DDL includes commands for integrity constraints, view definition, transactions, authorization. Select (*, all, distinct, as, where, from), insert, update, delete, create table, alter table, join _ on/where, with, delete from _ where, insert into _ values (), update _ set. Cartesian product vs. theta join definition: $\sigma_{ins.ID=tea.ID}(ins \times tea) = ins \bowtie_{ins.ID=tea.ID} tea$. **Comparison** involving null (other than is/not null) is always null. SQL requires an **associative** entity if the relationship is many-to-many or has properties that are not attributes of connected entities. **Domain** types: char(n) fixed length string with length n, varchar(n) variable length, int, smallint, numeric(p,d) fixed point number with precision of p digits with d decimals, real, double precision, float(n) floating point number with precision of at least n digits, datetime, boolean. Constraints: not null, primary/foreign key, unique, check(P) predicate in. Natural join matches tuples with same values for all common attributes.



Alias tables that appear multiple times in queries. **Codd's** 12 rules: rule 3 systematic treatment of null values, can be interpreted as missing, unknown, or N/A data. RA uses = comparison. **Aggregate** functions: avg, min, max, sum, count, group by (having instead of where). Attributes in select clauses outside of aggregate functions must appear in group by list. Left, right, full outer join on _ using _. **Set** operations between two select statements: union, intersect, except. Use all to retain duplicates. **Where** clauses: some, all, exists/not, unique. From clause for nested subqueries. With clause to define temporary relations available to the query in which it occurs (with _ as _, … select …). Scalar subquery used when single return value expected. **Specialization**/generalization is-a: overlapping/disjoint object can be member of more than one/only one specialized subclass, overlapping separate arrows, disjoint merged arrows incomplete only some instances of parent class are specialized and other instances only have common attributes, common every instance of parent class has one or more unique attributes. Complex attributes: single atomic/composite (full name), single/multivalued (phone number), derived (age). Anti-join keeps all records in the original table except those that match the other table. Semi-join returns rows from one table for which there are matching



records in another table. Division results in attribute names unique to R, in header of R but not in header of S, for which all combinations with tuples in S are present in R. **Views** hide certain data from some users. One/two/three table implementation. Create view _ as _, saves expression to refer to virtual relation that is substituted into queries. V1 depends directly on V2 if V2 is used to define V1. V1 depends on V2 if either V1 depends directly on V2 or there is a patch of dependencies. V is recursive if it depends on itself. Materialized views are physically-stored copies of views for certain database systems, will go out of date if relations used in query are updated, requires maintenance. Insert into view values (), must be represented by insertion into corresponding relation, otherwise reject insert or insert null. Codd's rule 4 **data dictionary** contains metadata about schema, integrity constraints, authorization, physical file organization, compiled by DDL compiler. Select from _ where TABLE_SCHEMA=, TABLE_NAME=. REST representational state transfer architecture. **Index** on attribute of relation allows DBMS to find tuples that have specified value of attribute efficiently, create index _ on _. Declarative languages are not Turing complete. **Functions**/procedures allow business logic to be stored in db and executed from SQL statements. Language constructs compound statements may contain multiple statements and declare local variables between begin and end. While boolean, do statements, end while. Repeat statements, until boolean, end repeat. For loop. Declare var. If, elseif, else, then, end if. Create function name(args) returns _. Create procedure name(in, out). Functions can be in SQL statements, procedures must be called on their own, call _. Triggers are executed automatically by the system as a side effect of a modification to the db. Before/after insert/update/delete, referencing old/new row/table, for each row/table. Databases today provide built-in materialized view facilities, encapsulation facilities can be used instead of triggers. Triggers not used when risk of unintended execution of triggers, other risks leading to failure of critical transactions or cascading execution.

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

Authorization read, insert, update, delete, all privileges. Index allows creation/deletion of indices, resources allows creation of new relations, alteration allows addition/deletion of attributes, drop allows deletion of relations. Grant privilege list on relation/view to user list. Revoke privilege list on relation/view from user list. Roles distinguish among users, create role _, grant role to users. CTE common table expression defines temporary result set to use in select statement. Recursion, with recursive _.

List purposes for database systems compared to managing data by writing applications that process files. - Database systems allow for domain and integrity constraints to be placed on data which ensure its consistency. - Database systems use standardized query languages like SQL that are optimized for efficient and fast storage, manipulation, and retrieval of data, and can represent complex relationships between data. - Database systems are scalable and secure and can handle concurrency control between multiple users accessing data at the same time. - Database systems provide a layer of abstraction between data and application code,

allowing changes to be made to either the database schema or the application independently of the other. **Database** systems provide users with an abstraction view of the data. a. Briefly explain the concept. Database systems allow users to interact with the data without needing to fully understand how the data is stored. Users can use high-level query languages like SQL to view and update data in specific ways without needing to implement a complex program or understand the physical storage of the data. Furthermore, changes can be made to the database schema without affecting any applications that access the data. b. Why do writing applications that access files do not provide an abstraction? Applications that access files do not provide an abstraction because they require developers to write custom programs to manually parse data and understand the physical structures of the files they are accessing. This results in tight coupling between the files containing the data and the application code, meaning that changes made to either would result in significant changes needing to be made to the other. Finally, accessing files directly means that users lose out on the advantages of database systems, including a standard query language, representation of complex relationships between data, data integrity, scalability, security, and concurrency. **What** are the 3 levels of data abstraction that a DBMS provides? The 3 levels of data abstraction that a DBMS provides are the conceptual, logical, and physical levels. **Explain** the difference between database schema and database instance. What two concepts in an object-oriented language correspond to schema and instance? A database schema represents the logical structure of the data being stored, including tables, fields, constraints, and relationships. A database instance is a specific state of the database schema which contains the actual data stored in the database. In terms of object-oriented language concepts, a schema is analogous to a class, while an instance is analogous to an object of a class. **Briefly** describe the concepts of data definition language and data manipulation language. What are the two types of data manipulation language? Data definition language (DDL) includes commands for specifying integrity constraints, defining views, controlling transactions, and establishing data relationships. Data manipulation language (DML) provides the ability to perform create, read, update, and delete operations on tuples in the database. The two types of DML are procedural DML and declarative DML. Procedural DML requires a user to specify what data are needed and how to get it, and declarative DML requires a user to specify what data are needed without specifying how to get it. **Briefly** explain two-tier and three-tier database application architectures. Is a full-stack web application a two-tier architecture or a three-tier architecture? Two-tier and three-tier database application architectures describe how many parts a database application is partitioned into. In two-tier architecture, the application resides at the client machine and invokes database system functionality at the server machine. In three-tier architecture, the client machine acts as a front-end and does not contain any direct database calls. Rather, the client communicates with an application server, which in turn communicates with a database system to access data. A full-stack web application is a three-tier architecture. **What** are the four types of database users based on skill level and for what they use the database. Which type of user defines schema and defines what information users can access? The four types of database users are: - Naive users: unsophisticated users who invoke a previously written application program. - Application programmers: professionals who write application programs. - Sophisticated users: users who interact with the system without writing programs either by using a database query language or by using tools such as data analysis software. - Specialized users: users who write specialized database applications that don't fit into the traditional data-processing framework. Database administrators define the schema and define what information users can access. **Advantages** of ER Modeling: - ER models are conceptually simple and can be easily drawn if the relationships between entities and attributes are known. - ER models are a good visual diagrammatic representation of the logical structure of a database and allow for easy understanding of relationships among entities. - ER models are an effective communication tool for database designers with other technical and non-technical stakeholders. - ER models can easily be converted into relational data models using tables, as well as many other data models including hierarchical and network data models. Disadvantages of ER Modeling: - ER models have limited constraints and specifications, which can result in a loss of some information content. - ER models have a limited ability to represent complex relationships in comparison to other data models such as relational data models. - ER models have a limited ability to represent data manipulation. - ER models lack an industry standard for notation and can require specialized software tools to create.

Briefly explain the concepts of "top-down" and "bottom-up" data modeling. In top-down data modeling, lower-level entity sets that have distinctive attributes or relationships are specialized from a higher-level entity set. In bottom-up data modeling, lower-level entity sets that share the same attributes or relationships are generalized into a higher-level entity set. **What** are two reasons it is necessary to use an associative entity to implement a relationship set/relationship between entities? One reason it is necessary to use an associative entity is if the relationship being implemented is a many-to-many relationship. Another reason to use an associative entity is if there are properties on the relationship that are not attributes of the connected entities. **Briefly** explain the concepts of the degree of a relationship/relationship set, and the cardinality of a relationship. The degree of a relationship is the number of entity sets that the relationship involves. The cardinality of a relationship is the number of entities to which another entity can be associated with via the relationship. **A** theta-join is of the form $r \bowtie \theta\ s$, where r and s are relations (tables). What is an equivalent relational algebra statement that only uses π, σ,×,∪, −, ρ? Your answer may not need to use all the operators. $r \bowtie \theta\ s = \sigma\theta(r \times s)$. **Consider** a domain for an attribute/column in which the possible values are in the range 0.00 to 99.99, and there can only be two digits after the decimal point. What SQL data type would you use? NUMERIC(4, 2). **Consider** the create table statement below, which columns could be the primary key in a MySQL database and why? CREATE TABLE IF NOT EXISTS course_management.students_fixed (CU_id INT NOT NULL, uni VARCHAR(12) NULL, email TEXT NOT NULL, first_name TEXT NULL, last_name TEXT NULL, middle_name TEXT NULL); The CU_id column could be the primary key in a MySQL database because it is a unique identifier for a student, and it is marked as not able to be null. The uni column is a unique identifier for a student, but it is able to be null, so it could not be the primary key. The email, first_name, last_name, and middle_name columns are not guaranteed to be unique identifiers for a student, so they could not be the primary key. **Why** does using a natural join sometimes produce incorrect results or results that do not make any sense? A natural join automatically joins tables based on columns with the same name. If two tables have columns with the same name but with different meanings, using a natural join might produce incorrect or nonsensical results. Similarly, using a natural join might inadvertently include columns with the same name that were not intended to be used for the join, which might lead to incorrect data associations since natural joins do not allow you to specify which columns to join on. Some SQL database management systems do not implement the full outer join operation. Describe how to write an equivalent query to a full outer join using other SQL capabilities. An equivalent query to a full outer join can be achieved by performing a left join, performing a right join, then combining the results of the left and right joins with a union. **SELECT** statements and UNION statements behave differently with respect to duplicates in a result set. Briefly explain the differences. How can you make SELECT and UNION behave the same with respect to duplicates? SELECT statements return all selected rows, including duplicates by default. UNION statements combine the results from multiple SELECT statements and automatically remove duplicates from the result. To make a SELECT statement remove duplicates, the DISTINCT keyword can be added to the statement. To make a UNION statement include duplicates, the UNION ALL operation can be used. **Explain** the concept of "arity" in SQL statements. "arity" in SQL statements refers to the number of attributes/columns in a table. This is often relevant when considering set operations, where both input relations in the operation must have the same arity and have compatible attribute domains. Arity also refers to the number of operands that a SQL statement takes, which is important to consider when structuring a SQL function, operator, or statement.