

Traditional databases contain number and text, but more recently there's different applications, such as:

- Multimedia
- Geographical Information Systems (GIS)
- Biological/genome
- Mobile
- Reactive

Database - a collection of related data

Mini-world - a part of the real world that the data stored in the database represents

Database management system (DBMS) - software package/system to facilitate the creation/maintenance of a database

Database system - the DBMS + data

A DBMS can:

- Define a database in terms of its data types, structures, and constraints
- Construct the database into another medium
- Manipulate the database
- Process concurrent users

Applications can interact with a database by generating queries and transactions.

A DBMS may provide:

- Protection to prevent unauthorized access
- Active processing to take internal actions on data
- Visualization of data

An example of a mini-world is a university. Entities within could be students, courses, instructors, and more.

There are relationships between these entities, such as:

- Sections are of specific courses
- Students take sections
- Courses have prerequisite courses
- Instructors teach sections

Meta-data - a DBMS catalog stores the description of a database

Data model - used to hide storage details and present the users with a conceptual view of the database

Different users may see different views of database which is relevant to them

Concurrency control in a DBMS guarantees that each transaction is correctly executed

OLTP - online transaction processing, allows many concurrent transactions to happen

Data model operations can vary, from retrievals and updates to user defined operations (such as update GPA for students)

Data Model Categories

- Conceptual data models - high level idea close to the way user perceive data
- Physical data models - how data is stored in the computer
- Implementation data models - concepts that fall between the conceptual and physical models, used to communicate with the DBMS
- Self-Describing data models - combining description of values with data values

Database schema - description of database

Database state - all the data in the database at a moment in time

Three-schema architecture is used to gain program-data independence and support multiple views of data.

1. Internal schema - physical data model used to describe storage and access paths
2. Conceptual schema - conceptual level to describe the structure and constraints for the whole database
3. External schema - describes different user's views

Mappings are used to map from external to internal schemas

Logical data independence - ability to change the conceptual schema without changing the external schema

Physical data independence - ability to change the internal schema without changing the conceptual schema

DBMS Architectures

Centralized DBMS - everything is combined into a single system, including software, hardware, applications, and UI processing

Basic 2-tier Client-Server Architecture - specialized servers with specialized functions, clients access each server as needed

3-Tier Client-Server Architecture -

Entity - specific things or objects in the mini-world that are represented in the database. Ex: an EMPLOYEE in a company

Attribute - properties used to describe an entity. Ex: ADDRESS of an EMPLOYEE

Types of attributes








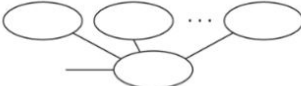



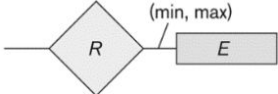
- Simple - each entity has a single atomic value. Ex: SSNs are a single value.
- Composite - attribute may be composed of several components. Ex: name is comprised of first, middle, and last name.
- Multi-valued - an entity has multiple values

Derived attributes - an attribute that can be calculated using data stored. Ex: age, derived from birthday.

Key attribute - an attribute of an entity for which each entity must have a unique value. Can be a composite attribute. Ex: CUID.

Entity set/collection - the current state of all the entities of a particular type stored in the database

Figure 3.14
Summary of the notation for ER diagrams.

| Symbol | Meaning |
|---|---|
|  | Entity |
|  | Weak Entity |
|  | Relationship |
|  | Identifying Relationship |
|  | Attribute |
|  | Key Attribute |
|  | Multivalued Attribute |
|  | Composite Attribute |
|  | Derived Attribute |
|  | Total Participation of E_2 in R |
|  | Cardinality Ratio 1: N for $E_1:E_2$ in R |
|  | Structural Constraint (min, max) on Participation of E in R |

Relationship - relates two or more distinct entities with a specific meaning. Ex: EMPLOYEE *works on* PROJECT.

Relationship type - the schema description of a relationship, relationship name and the participating entity types, gives constraints

Relationship set - current set of relationship instances represented in the database

Relationship constraints:

- Cardinality - labeled by placing numbers on the relationship edges
 - One to one (1:1)
 - One to many (1:N)
 - Many to many (M:N)
- Existence dependency - labeled by line or double line (double for mandatory)
 - Zero (optional participation)
 - One or more (mandatory participation)

Weak entity types - an entity type that does not have a key attribute, but can be identified with its relationship with an owner. Ex: a DEPENDENT on an EMPLOYEE.

A relationship type can have attributes. Ex: HOURS on a WORKS_ON relationship.

Most commonly used in many to many relationships, as in a one to many relationship, the attribute can be given to the entity with many.

Relationships that connect 3 entities are ternary relationships. With n entities, it is called n-ary. Ex of ternary: a SUPPLY relationship between SUPPLIER, PART, AND PROJECT.

Movie - title, isbn, release date, genres

Director - name, id, birthday

Actor - name, id, birthday

Studio - name, incorp date, studio number

EER - enhanced entity relations, can create subgroups of entities. Ex: EMPLOYEE is grouped into SECRETARY, ENGINEER, TECHNICIAN... Could also be a MANAGER.

This is notated with the member of notation from set notation.

An entity in the superclass can be more than one subclass. Ex: an EMPLOYEE can be both an ENGINEER and SALARIED.

Subclasses inherit all relationships and attributes of the superclass. Subclasses can have relationships and attributes that the superclass doesn't.

Total - every entity in the superclass must be a member of some subclass in the specialization/generalization. Denoted by a double line.

Partial - entity not to belong to any of the subclasses. Denoted by a single line.

Disjoint - an entity can be a member of at most one of the subclasses of the specialization. Denoted by a d.

Overlapping - the same entity may be a member of more than one subclass of the specialization. Denoted by an o.

Hierarchy - has a constraint that every subclass has only one superclass (called single inheritance); this is basically a tree structure

Lattice - a subclass can be a subclass of more than one superclass (called multiple inheritance)

Union type - superclasses can represent different entity types. Denoted by u. Ex: In a database for vehicle registration, a vehicle owner can be a PERSON, a BANK or a COMPANY.

A relation is a table of values. Contains a set of rows. A set of tuples.

The data elements in each row represent facts that correspond to a real-world entity or relationship. In the formal model, rows are called tuples. An ordered set of values.

Each column has a column header that gives the meaning of the data in that column. In the formal model, this is called the attribute name (or just attribute)

| | | | | | | | | |
|---------------|---|----------------|-------------|------------|----------------------|--------------|-----|------|
| Relation Name | | Attributes | | | | | | |
| STUDENT | | | | | | | | |
| | | Name | Ssn | Home_phone | Address | Office_phone | Age | Gpa |
| Tuples | → | Benjamin Bayer | 305-61-2435 | 373-1616 | 2918 Bluebonnet Lane | NULL | 19 | 3.21 |
| | → | Chung-cha Kim | 381-62-1245 | 375-4409 | 125 Kirby Road | NULL | 18 | 2.89 |
| | → | Dick Davidson | 422-11-2320 | NULL | 3452 Elgin Road | 749-1253 | 25 | 3.53 |
| | → | Rohan Panchal | 489-22-1100 | 376-9821 | 265 Lark Lane | 749-6492 | 28 | 3.93 |
| | → | Barbara Benson | 533-69-1238 | 839-8461 | 7384 Fontana Lane | NULL | 19 | 3.25 |

Each row has a value that uniquely identifies that row, aka a key.

The schema is the description of the relation, denoted by $R(A_1, A_2, \dots, A_N)$, where R is the name of the relation and its attributes are A_N .

Each attribute has a domain, or a valid set of values.

Relation state - a subset of the Cartesian product of the domains of its attributes

Cartesian product - set of all tuples possible given the domains of its attributes

- Formally,
 - Given $R(A_1, A_2, \dots, A_n)$
 - $r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$
- $R(A_1, A_2, \dots, A_n)$ is the **schema** of the relation
- R is the **name** of the relation
- A_1, A_2, \dots, A_n are the **attributes** of the relation
- $r(R)$: a specific **state** (or "value" or "population") of relation R – this is a *set of tuples* (rows)
 - $r(R) = \{t_1, t_2, \dots, t_n\}$ where each t_i is an n -tuple
 - $t_i = \langle v_1, v_2, \dots, v_n \rangle$ where each v_j *element-of* $\text{dom}(A_j)$

Order of tuples in the relation does not matter but the order of the data in a tuple matters.

A null value is allowed for unknown/nonapplicable values.

Component values in tuples are referred to by $t[A_i]$ or $t.A_i$.

There are 3 main types of constraints in a relational model:

1. Key constraint - split into superkey and key, a superkey is where no two tuples will have the same value, while a key is a minimal superkey, where removing any attribute of the superkey would make it no longer a key. If an entity has multiple keys, one is chosen to be the primary key. General rule in picking is the smallest of candidate keys and one that is unlikely to change.
2. Entity integrity - primary key attributes of each relation cannot have null values
3. Referential integrity - tuples in referencing relation have foreign key attributes that reference the primary key attributes of the referenced relation. The value in the foreign key of the referencing relation has to be either a value of an existing primary key of the referenced relation or null.

There are also other kinds of constraints, such as semantic integrity constraints. These are not constrained within the relational model, but instead with constraint specification languages.

Update operations include INSERT, DELETE, and MODIFY tuple.

When integrity is in danger of being violated, there are several options to take.

- Cancel the operation
- Perform the operation but inform the user of the violation
- Trigger additional updates so the violation is corrected
- Execute a user specific error correction routine

INSERT operations may violate any of the constraints.

- Domain - if one of the attribute values provided for the new tuple is not of the specified attribute domain
- Key - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- Referential integrity - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- Entity integrity - if the primary key value is null in the new tuple

DELETE operations may violate referential integrity constraints. This can be remedied by several actions such as RESTRICT (reject deletion), CASCADE (propagate new value to the foreign keys referencing that value), and SET NULL.

UPDATE operations may violate domain constraint

Algorithm to map ER diagram to relational model

1. Mapping of regular entity types
 - For each regular entity type E in the schema, create a relation R that includes all the simple attributes of E
 - For composite attributes, only add the simple attributes that will create the composite attribute
 - Choose one key attribute of E to be the primary key for R
 - If the chosen key of E is composite the simple attributes that comprise it will make up the primary key of R.
2. Mapping weak entity types
 - For each weak entity W, there is an owner entity E. Create a relation R and complete step 1 on W as attributes of R.
 - Include the primary key of the owner as the foreign key of R.
 - The primary key of R is the combination of the primary key of the owner and the partial key of W.
3. Mapping specialization or generalization
 - Add a relation for the superclass with all its attributes and its primary key.
 - Add a relation for each subclass with its subclass specific attributes.
 - Add the primary key of the superclass as the primary key of the subclass.
 - Add an arrow from the primary key of the subclass to the primary key of the superclass.

- In shared subclasses, which inherits from multiple superclasses, the subclass must have the primary key of each superclass as a foreign key. Its primary key is a combination of all its foreign keys.
- 4. Mapping of union/category types
 - Add each superclass following the normal rules for adding an entity.
 - For mapping a union type, create a new key attribute called a surrogate key.
 - Add a relation for the union type with the surrogate as the primary key. Add any other attributes of the union type as attributes of the relation.
 - In the superclass's relation, add the primary key of the union type as a foreign key and add an arc connecting the foreign key back to the primary key of the union type.
- 5. Mapping binary 1:1 relationship types
 - For each 1:1 relation R, identify the relations S and T that correspond to the entity types participating in R.
 - Choose one of the relations to include a foreign key of the primary key of the other relation. It is better to choose an entity that has total participation to have the foreign key.
- 6. Mapping binary 1:N relationship types
 - For each 1:N relation R, identify the relation S as the participating entity type on the N side.
 - Include as a foreign key in S the primary key of the relation T on the other entity in the relationship.
- 7. Mapping binary N:M relationship types
 - For each N:M relation R, create a new relation S to represent R. This is called a relationship relation.
 - Include as a foreign key in S the primary keys of both relations in R. The two foreign keys combined will form the primary key of S.
 - Include any simple attributes of R as attributes of S.
- 8. Mapping multivalued attributes
 - For each multivalued attribute A, create a new relation R.
 - R will include an attribute corresponding to A, plus the primary key attribute K as a foreign key in R.
 - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.
- 9. Mapping n-ary relationship types
 - For each n-ary relation R, create a new relationship S to represent R.
 - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
 - Include any simple attributes of the relationship as attributes of S.

A union type has multiple superclasses but only belongs to one of them. Add a relation for the union type with a key and its attributes. Then add the primary key of the union type as a foreign key to the relations for the superclasses.

SQL

Table = relation

Row = tuple

Column = attribute

Commands in SQL end with a semicolon

CREATE

To create a new table, provide the name of the table, attributes, types, and initial constraints

```
CREATE TABLE <name>
    (<attribute1>,
     <attribute2> ....
     PRIMARY KEY(<attribute_name>)
     [FOREIGN KEY(<attribute>) REFERENCES
     <table>(<attribute>)]
     [<table options>]
    );
```

Ex:

```
CREATE TABLE COMPANY.EMPLOYEE
```

Or

```
CREATE TABLE EMPLOYEE
```

Syntax for creating an attribute is [attribute_name] [datatype] [constraints]

Datatypes

Integer: INTEGER, INT, SMALLINT

Floating point: FLOAT, REAL, DOUBLE, DECIMAL(m, d)

Fixed length char: CHAR(n), CHARACTER(n)

Varying length char: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)

Fixed length bit: BIT(n)

Varying length bit: BIT VARYING(n)

Boolean: TRUE or FALSE or NULL

Date: Components include YEAR, MONTH, and DAY in YYYY-MM-DD

Timestamp: includes DATE and TIME fields

INTERVAL: Relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

Constraints on attributes

DEFAULT [value]

If NULL is not allowed for that attribute, use NOT NULL

CHECK clause. Ex: Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

Domains are similar to objects in OO programming. Ex: CREATE DOMAIN SSN_TYPE AS CHAR(9);

Tables can have constraints themselves, which are after the attribute list.

PRIMARY KEY clause specifies 1+ attributes that make up the primary key of a relation. Ex: PRIMARY KEY(Dnumber), PRIMARY KEY (Licstate, LicNumber)

UNIQUE clause specifies alternate/secondary/CANDIDATE keys in the relational model. Ex: UNIQUE(Dname)

FOREIGN KEY clause. Default rejects update on violation. Ex: FOREIGN KEY([attribute_name]) REFERENCES [table_name]([attribute_name])

Referential triggered action clause options include SET NULL, CASCADE, and SET DEFAULT.

Sometimes foreign keys may cause errors (circular references) because they refer to a table that hasn't been created yet. To fix this, use SET FOREIGN_KEY_CHECKS=0; but remember to set back to 1 when done.

DROP command is used to drop named schema elements, such as tables, domains, or constraint. Drop behavior options include CASCADE and RESTRICT. Ex: DROP SCHEMA COMPANY CASCADE;, which removes the schema and all its elements including tables, views, constraints, etc.

ALTER table command actions include adding/dropping a column, changing a column definition, and adding/dropping table constraints. Ex: ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

To drop a column, choose CASCADE or RESTRICT. Ex: ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

Default values can be added/dropped. Ex: ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn SET DEFAULT '333445555';

INSERT command will add one+ tuples to a relation.

Create a new department called "Intelligence" and make Archer the manager of it starting today
INSERT INTO EMPLOYEE VALUES('Sterling', 'M', 'Archer', '657126565', 1975-02-09, '134
Duchess Way Apt 714, New York, NY', 41000, '888665555', 1);

Make the intelligence department located in the Danger Zone
INSERT INTO DEPARTMENT VALUES('Intelligence', 7, '657126565', 2020-02-24);

Add a project called "Spying" with number 4 in the location Danger Zone that belongs to the
intelligence department
INSERT INTO PROJECT VALUES('Spying', 4, 'Danger Zone', 7);

Make Archer work on the Spying Project for 20 hours a week
INSERT INTO WORKSON VALUES('657126565', 4, 20);

Add a dependent for Archer named "Seamus" who is his son with a birthday of February 10th
2011
INSERT INTO DEPENDENT VALUES('657126565', 'Seamus', 2011-02-10, 'Son');

SELECT command is used for retrieving information from a database. SQL allows a table to
have 2+ tuples that are identical in all their attribute values.

Logical operators: =, <, <=, >, >=, <> (not equal), AND, OR, NOT

SELECT [attribute list] FROM [table list] WHERE [condition]; Ex: SELECT Fname, Lname
FROM Employee;

All Employee first and last names and their birthday
SELECT Fname, Lname, Bdate FROM Employee

All department numbers and their locations
SELECT Dnumber, Dlocation FROM Dept_Locations

All Dependent names and their birthdays
SELECT Dependent_name, Bdate FROM Dependent

All Project names, numbers and the departments they belong to
SELECT Pname, Pnumber, Dnum FROM Project

LIKE comparison operator

Pattern matching: %[string]%, _ acts as a wildcard for a single character. Ex: WHERE Address
LIKE '%Houston, TX%'; WHERE Ssn LIKE ' __1__8901';

BETWEEN comparison operator. Ex: WHERE(Salary BETWEEN 30000 AND 40000) AND Dno
= 5;

Aliases or tuple variables: declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname  
FROM EMPLOYEE AS E, EMPLOYEE AS S  
WHERE E.Super_ssn = S.Ssn;;
```

The names of all employees who work on the Computerization project, and the number of hours they work on it

```
SELECT E.Fname, E.Lname, WORKS_ON.Hours  
FROM EMPLOYEE AS E, PROJECT, WORKS_ON  
WHERE PROJECT.Pname = 'Computerization' AND WORKS_ON.Essn = E.Ssn AND  
WORKS_ON.Pno = PROJECT.Pnumber;
```

The name and Address of all employees who make \$30000 or more in salary

```
SELECT E.Fame, E.Address  
FROM EMPLOYEE AS E  
WHERE E.Salary > 30000;
```

Every Dependent name, and the name of the employee who they are dependent on

```
SELECT D.Dependent_name, E.Fname, E.Lname  
FROM DEPENDENT AS D, EMPLOYEE AS E  
WHERE E.Ssn = D.Essn;
```

The name of every employee who worked on any project for more than 20 hours a week, and the name of that project

```
SELECT E.Fname, E.Lname, P.Pname  
FROM EMPLOYEE AS E, PROJECT AS P, WORKS_ON  
WHERE WORKS_ON > 20 AND E.Ssn = WORKS_ON.Essn, WORKS_ON.Pno = P.Pnumber;
```

SQL doesn't automatically eliminate duplicate tuples. Use the keyword DISTINCT to get distinct values. Ex: SELECT DISTINCT Salary FROM EMPLOYEE;

Set operations: UNION, EXCEPT (difference), INTERSECT

Arithmetic operators also apply: +, -, *, / can be included as part of SELECT. Ex:

```
SELECT E.Fname, E.Lname 1.1 * E.Salary AS Increased_sal  
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P  
WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND P.Pname = 'ProductX';
```

ORDER BY can order results. Keywords DESC and ASC to see results in descending/ascending order of values. Ex: ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC;

Show the names of all the employees and their salaries, ordered their salary from high to low.

```
SELECT Fname, Lname, Salary
FROM EMPLOYEE
ORDER BY Salary DESC;
```

Each employee deposits 4% of their pre-tax salary into a retirement fund. Show the names of the employees and their taxable salary (salary after their retirement is taken out). Assign the name "taxable salary" to the taxable salary column

```
SELECT Fname, Lname, Salary - (Salary * 0.04) AS taxable_salary
FROM EMPLOYEE;
```

Show the names of the managers of each department and the name of the department they manage ordered by their start date, with the longest tenured manager on top

```
SELECT Fname, Lname, Dname, Mgr_start_date
FROM EMPLOYEE, DEPARTMENT
WHERE Mgr_ssn = Ssn
ORDER BY Mgr_start_date ASC;
```

DELETE removes a tuple from the relation. Ex: DELETE FROM EMPLOYEE WHERE Lname='Brown';

The safest way to delete is to use the primary key so you don't accidentally remove a tuple you didn't mean to.

UPDATE will modify attribute values of selected tuples. Ex: UPDATE PROJECT SET Plocation = 'Bellaire', Dnum = 5 WHERE Pnumber = 10;

Remove Franklin Wong's spouse as a dependent from the database (use Franklins ssn)

```
DELETE FROM DEPENDENT
WHERE Essn = '333445555' AND Relationship = 'Spouse';
```

Make Alicia Zelaya the manager of the Administration Department (use Alicia's ssn)

```
UPDATE DEPARTMENT
SET Mgr_ssn = '999887777', Mgr_start_date = '2020-03-02'
WHERE Dname = 'Administration';
```

Change the name of Product Z to "Bluth's Cornballer"

```
UPDATE PROJECT
SET Pname = 'Bluths Cornballer'
WHERE Pname = 'Product Z';
```

Remove all entries of an employee working on project number 30

```
DELETE FROM WORKS_ON
```

WHERE Pno = 30;

Nested queries go inside a WHERE clause, using IN (= ANY), ANY, or ALL. Ex: To retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN
    (SELECT DNUMBER
     FROM DEPARTMENT
     WHERE DNAME='Research');
```

Correlated queries - a condition in the nested query can reference an attribute of the outer query

What is the name of the employee with the highest salary?

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Salary >= ALL
    (SELECT Salary
     FROM EMPLOYEE
     WHERE );
```

What are the names of the employees who work on a project located in Sugarland? Do not provide duplicate names

```
SELECT DISTINCT Fname, Lname
FROM EMPLOYEE
WHERE SSN IN
    (SELECT Essn
     FROM WORKS_ON
     WHERE Pno IN
         (SELECT Pnumber
          FROM Project
          WHERE Plocation = 'Sugarland'));
```

What are the names of employees who have a son?

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Ssn IN
    (SELECT Essn
     FROM DEPENDENT
     WHERE RELATIONSHIP = 'Son');
```

What are the names of the employees who make more money than Jennifer Wallace?

```
SELECT Fname, Lname
```

```

FROM EMPLOYEE
WHERE Salary > ALL
    (SELECT Salary
     FROM EMPLOYEE
     WHERE Fname = 'Jennifer' AND Lname = 'Wallace');

```

EXISTS and NOT EXISTS checks whether the result of a correlated nested query is empty or not. They return TRUE or FALSE.

UNIQUE(q) returns TRUE if there are no duplicate tuples in the result of query q.

JOIN in the FROM clause will generate a table based on the match. Ex:

```

SELECT Fname, Lname, Address
FROM (Employee JOIN Department ON Dno = Dnumber)
WHERE Dname = 'Research';

```

There are different types of join:

- NATURAL JOIN - No join condition specified. Must have the same name.
- INNER JOIN - default type of join, tuple included only if a matching tuple exists in the other relation
- OUTER JOIN
 - LEFT - every tuple in left table must appear, padded with NULL if right tuple doesn't exist
 - RIGHT - every tuple in right table must appear, padded with NULL if left tuple doesn't exist
 - FULL - combines the results of a LEFT and RIGHT OUTER JOIN

Can nest JOIN statements. Ex:

```

SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM ((Project JOIN Department ON Dnum = Dnumber) JOIN Employee ON Mgr_ssn = Ssn)
WHERE Plocation = 'Stafford';

```

The names of all employees who work on the Computerization project, and the number of hours they work on it

```

SELECT Fname, Lname, Hours
FROM ((EMPLOYEE JOIN WORKS_ON ON Ssn = Essn) JOIN PROJECT ON Pno = Pnumber)
WHERE Pname = 'Computerization';

```

The name and location of each project, the name of the controlling department, and the name of the manager of that department

```

SELECT Pname, Plocation, Dname, Fname, Lname
FROM ((PROJECT JOIN DEPARTMENT ON Dnum = Dnumber) JOIN EMPLOYEE ON
Mgr_ssn = Ssn);

```

Every Dependent name, and the name of the employee who they are dependent on

```
SELECT Dependent_name, Fname, Lname  
FROM (DEPENDENT JOIN EMPLOYEE ON Essn = Ssn);
```

The name of every employee who worked on any project for more than 20 hours a week, and the name of that project

```
SELECT Fname, Lname, Pname  
FROM ((EMPLOYEE JOIN WORKS_ON ON Ssn = Essn) JOIN PROJECT ON Pno = Pnumber)  
WHERE Hours > 20;
```

Aggregate functions include COUNT, SUM, MAX, MIN, and AVG. Ex: SELECT SUM(Salary) AS Total_Sal, MAX(Salary), MIN(Salary), AVG(Salary) AS Average_Sal FROM EMPLOYEE;
COUNT can be used to count the number of entries. Ex: SELECT (*) FROM EMPLOYEE;
SOME and ALL work as logical or and and on booleans.
GROUP BY specifies grouping attributes

Ex: Count number of projects in each location for each department
SELECT Dname, Plocation, COUNT(*)
FROM PROJECT INNER JOIN DEPARTMENT ON Dno = Dnumber
GROUP BY Dname, Plocation;

Ex: For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT Pnumber, Pname, COUNT(*)  
FROM PROJECT, WORKS_ON  
WHERE Pnumber = Pno  
GROUP BY Pnumber, Pname  
HAVING COUNT (*) > 2;
```

How many Projects are located in Stafford?

```
SELECT COUNT(*)  
FROM PROJECT  
WHERE Plocation = 'Stafford'
```

How many hours did employees work on each project? Show the hours and Project name.
Order Project Name.

```
SELECT Pname, SUM(Hours)  
FROM PROJECT LEFT OUTER JOIN WORKS_ON ON Pnumber = Pno  
GROUP BY Pname;  
ORDER BY Pname;
```

For each employee, show their name and the number of dependents they have (Even if the number is 0)

```
SELECT Fname, Lname, COUNT(Essn)
```


FROM EMPLOYEE LEFT OUTER JOIN DEPENDENT ON Ssn = Essn
GROUP BY Fname, Lname;

Order of SQL interpretation

1. Merge tables with FROM and JOIN
2. Use WHERE to eliminate rows
3. GROUP BY
4. HAVING to eliminate groups
5. SELECT to pick the attributes
6. ORDER BY

CASE constructs are used when a value can be different based on certain conditions. Ex: CASE WHEN Dno = 5 THEN Salary + 2000 WHEN Dno = 4 THEN Salary + 1500;

CREATE ASSERTION and TRIGGER. ASSERTIONS specify additional types of constraints outside scope of built-in relational model constraints. TRIGGERS Specify automatic actions that the database system will perform when certain events and conditions occur.

A view is a single table derived from other tables called the defining tables. CREATE VIEW can create this. Ex: CREATE VIEW WORKS_ON AS SELECT Fname, Lname, Pname, Hours, FROM EMPLOYEE, PROJECT, WORKS_ON WHERE Ssn = Essn AND Pno = Pnumber;
Can delete a view using DROP VIEW.

NULL is not equal to other NULL values.

Table 7.1 Logical Connectives in Three-Valued Logic

| (a) | AND | TRUE | FALSE | UNKNOWN |
|-----|---------|---------|---------|---------|
| | | TRUE | FALSE | UNKNOWN |
| | FALSE | FALSE | FALSE | FALSE |
| | UNKNOWN | UNKNOWN | FALSE | UNKNOWN |
| (b) | OR | TRUE | FALSE | UNKNOWN |
| | | TRUE | TRUE | TRUE |
| | FALSE | TRUE | FALSE | UNKNOWN |
| | UNKNOWN | TRUE | UNKNOWN | UNKNOWN |
| (c) | NOT | | | |
| | | FALSE | | |
| | FALSE | TRUE | | |
| | UNKNOWN | UNKNOWN | | |

Embedded SQL - database commands in a general-purpose programming language using special prefixes for the preprocessor to scan for
Could also use a library of database functions, such as an API.

However, there could be differences between the models in the database and programming language. This is called impedance mismatch. Ex: relational vs object-oriented. Translation is required.

Embedded SQL (C)

EXEC SQL is the prefix used to note to the preprocessor that this is SQL. Is terminated by a semicolon or END-EXEC

Shared variables - variables used in both C and the embedded SQL statements. Will have a colon in front of it in the SQL statement.

Ex of declaring variables:

```
EXEC SQL BEGIN DECLARE SECTION;
varchar dname [16], fname [16], lname [16], address [31];
char ssn [10], bdate [11], sex [2], minit [2];
float salary, raise;
int dno, dnumber;
int SQLCODE; char SQLSTATE [6];
EXEC SQL END DECLARE SECTION;
```

Connecting to DB

```
CONNECT TO [server name] AS [connection name]
AUTHORIZATION [account name and password];
```

Change connection

```
SET CONNECTION [connection name];
```

Terminate connection

```
DISCONNECT [connection name];
```

SQLCODE and SQLSTATE are special communication variables used by the DBMS to communicate exception or error conditions.

SQLCODE

- 0 = statement executed successfully
- 100 = no more data available in query result
- < 0 = error has occurred

SQLSTATE is a string of 5 characters

- '00000' = no error/exception

Ex of getting SQL results into C variables:

EXEC SQL

```
SELECT Fname, Minit, Lname, Address, Salary
INTO :fname, :minit, :lname, :address, :salary
FROM EMPLOYEE WHERE Ssn = :ssn;
```

If (SQLCODE == 0) printf(fname, minit, lname, address, salary)

Can retrieve multiple tuples with embedded SQL by using a cursor

Cursor - points to a single tuple/row from a result of query. The cursor is declared when the SQL query is declared. The cursor is basically an iterator.

OPEN CURSOR sets cursor to a position before first row in result and becomes current row for cursor

FETCH moves cursor to next row

CLOSE CURSOR is used to indicate we're done with the info associated with that cursor

Cursor declaration syntax is DECLARE [cursor name] FOR <query> [FOR READ ONLY | FOR UPDATE | OF <attribute>]

```
//Program Segment E2:
0) prompt("Enter the Department Name: ", dname) ;
1) EXEC SQL
2)   SELECT Dnumber INTO :dnumber
3)   FROM DEPARTMENT WHERE Dname = :dname ;
4) EXEC SQL DECLARE EMP CURSOR FOR
5)   SELECT Ssn, Fname, Minit, Lname, Salary
6)   FROM EMPLOYEE WHERE Dno = :dnumber
7)   FOR UPDATE OF Salary ;
8) EXEC SQL OPEN EMP ;
9) EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
10) while (SQLCODE = = 0) {
11)   printf("Employee name is:", Fname, Minit, Lname) ;
12)   prompt("Enter the raise amount: ", raise) ;
13)   EXEC SQL
14)     UPDATE EMPLOYEE
15)     SET Salary = Salary + :raise
16)     WHERE CURRENT OF EMP ;
17)   EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
18) }
19) EXEC SQL CLOSE EMP ;
```

FOR UPDATE OF [attribute] is used when a cursor is defined for rows that have to be updated. Then in WHERE CURRENT OF, the cursor name must be specified.

SQLJ

To connect to a SQL database with Java:

```
DefaultContext cntxt = oracle.getConnection("[url name]", "[user name]", "[password]", true);
DefaultContext.setDefaultContext(cntxt);
```

SQLExceptions are a type of exception in Java for when a SQL error occurs. Using a try catch block will catch the SQLExceptions.

Iterators can retrieve multiple tuples from a result. Named iterators are associated with a query result by listing attribute names and types that appear in the result. Positional iterators list only the attribute types.

```
//Program Segment J2A:
0) dname = readEntry("Enter the Department Name: ") ;
1) try {
2)     #sql { SELECT Dnumber INTO :dnumber
3)         FROM DEPARTMENT WHERE Dname = :dname} ;
4) } catch (SQLException se) {
5)     System.out.println("Department does not exist: " + dname) ;
6)     Return ;
7) }
8) System.out.println("Employee information for Department: " + dname) ;
9) #sql iterator Emp(String ssn, String fname, String minit, String lname,
    double salary) ;
10) Emp e = null ;
11) #sql e = { SELECT ssn, fname, minit, lname, salary
12)     FROM EMPLOYEE WHERE Dno = :dnumber} ;
13) while (e.next()) {
14)     System.out.println(e.ssn + " " + e.fname + " " + e.minit + " " +
        e.lname + " " + e.salary) ;
15) } ;
16) e.close() ;
```

Embedded SQL is more common in C than it is in other languages, as C is not object-oriented so that there are no classes. Java, however, is object-oriented so we can use objects to create a library for accessing databases. This is already done and is called JDBC.

JDBC

- Connection object
- Statement object
 - PreparedStatement - uses parameter binding, which is more secure. Puts ? in place of variables. Creates a separation between the query and the data.
 - CallableStatement
- ResultSet object
- ? used as a statement parameter and is determined during runtime

Steps to use JDBC

1. Import java.sql.*
2. Load driver
 - a. JDBC works with any SQL DBMS implementation but needs to inform which driver to load.
 - i. Class.forName("oracle.jdbc.driver.OracleDriver")
 - ii. Class.forName("com.mysql.jdbc.Driver")

3. Create connection
 - a. Connection conn;
 - b. Usually private and static for security reasons. No constructor since it is dependent on the DBMS.
 - c. conn = DriverManager.getConnection(db, user, pw);
4. Run queries
 - a. Prepare statement
 - i. CallableStatement
 1. Statement stmt = conn.createStatement();
 2. String query = "SELECT fname, lname FROM EMPLOYEE WHERE Dno = 5;"
 3. ResultSet rset = stmt.executeQuery(query);
 - ii. PreparedStatement
 1. Use clearParameters() to remove any data
 2. Use setString(int pos, String data) to replace ? with data
 3. Pos is 1 indexed
 - a. String query = "SELECT fname, lname FROM EMPLOYEE WHERE Dno = ?;"
 - b. PreparedStatement stmt = conn.prepareStatement(query);
 - c. stmt.clearParameters();
 - d. stmt.setString(1, 5);
 - e. ResultSet rset = stmt.executeQuery();
 - b. Bind variables
 - c. Execute query
 - d. Parse ResultSet
 - i. A class provided by JDBC to help with impedance mismatch. Each row is a tuple, each column is an attribute.
 - ii. ResultSet.next() gets next tuple
 - iii. Points to before first row so must call next once.
 - iv. ResultSet.getString(int colIndex) gets the value at the column index, with indexing starting at 1
 - v. ResultSet.getString(String colName) gets the value at the column name
 - vi. Other gets include getInt, getDouble, etc. Cannot do Integer.parseInt(rset.getString(1)) because parseInt can't handle NULLs
 - vii. ResultSet.isNull() returns true if the last value in a get call was a NULL
5. Close connection
 - a. conn.close();

For non-queries (inserts/updates), you can call executeUpdate() instead of executeQuery()

JDBC Programming Best Practices

Use separation of concerns. Only have one file that accesses the db so that it is easier to change and enforce security protocols.

Make methods static so that you don't need to make an object of the DB class to call the methods. Return Java objects the program is using, not RSets.

Stored procedures are stored by the DBMS at the server. They can be functions or procedures. Also sometimes called Persistent Stored Modules. If a DB has several applications connecting it, all requiring the same task, you can factor out the task and put it on the DB. This reduces data transfer cost and can also be a security feature as the SQL queries would not be sent across the network and no one could learn about the DB from that data.

CREATE PROCEDURE

```
CREATE PROCEDURE <procedure name> (<parameters>)  
<local declarations>  
<procedure body> ;
```

CREATE FUNCTION

```
CREATE FUNCTION <function name> (<parameters>)  
RETURNS <return type>  
<local declarations>  
<function body> ;
```

Each parameter has a type that is one of the SQL data types. The parameters also have modes, which can be IN, OUT, or INOUT.

Conditionals look like

```
IF <condition> THEN <statement list>  
    ELSEIF <condition> THEN <statement list>  
    ...  
    ELSEIF <condition> THEN <statement list>  
    ELSE <statement list>  
END IF ;
```

Loops

```

WHILE <condition> DO
    <statement list>
END WHILE ;
REPEAT
    <statement list>
UNTIL <condition>
END REPEAT ;

```

There is also a cursor-based looping structure. The statement list in such a loop is executed once for each tuple in the query result. This has the following form:

```

FOR <loop name> AS <cursor name> CURSOR FOR <query> DO
    <statement list>
END FOR ;

```

Ex of function in SQL/PSM:

```

//Function PSM1:
0) CREATE FUNCTION Dept_size(IN deptno INTEGER)
1) RETURNS VARCHAR [7]
2) DECLARE No_of_ems INTEGER ;
3) SELECT COUNT(*) INTO No_of_ems
4) FROM EMPLOYEE WHERE Dno = deptno ;
5) IF No_of_ems > 100 THEN RETURN "HUGE"
6) ELSEIF No_of_ems > 25 THEN RETURN "LARGE"
7) ELSEIF No_of_ems > 10 THEN RETURN "MEDIUM"
8) ELSE RETURN "SMALL"
9) END IF ;

```

To get output from stored procedures, use `registerOutParameter(int pos, [datatype])`. Then use the get methods to get output value.

```
//getDBUSERByUserId is a stored procedure
String getDBUSERByUserIdSql = "{call getDBUSERByUserId(?,?,?,?)}";
callableStatement = dbConnection.prepareCall(getDBUSERByUserIdSql);
callableStatement.setInt(1, 10);
callableStatement.registerOutParameter(2, java.sql.Types.VARCHAR);
callableStatement.registerOutParameter(3, java.sql.Types.VARCHAR);
callableStatement.registerOutParameter(4, java.sql.Types.DATE);

// execute getDBUSERByUserId store procedure
callableStatement.executeUpdate();

String userName = callableStatement.getString(2);
String createdBy = callableStatement.getString(3);
Date createdDate = callableStatement.getDate(4);
```

Three-tier client-server architecture can separate information. There is the client, application/web server, and database server.

Embedded SQL vs library

In embedded SQL, query text can be checked for syntax errors at compile time. However, with complex applications that need dynamically generated queries, it's better to use function call. Function call has more flexibility and has dynamic queries. However, there is no syntax checking at compile time.

Database Security

Legal (FERPA, HIPPA), ethical (responsibility), policy (government/institutional/corporate), and multiple security level issues (different clearances).

Confidentiality

Unauthorized disclosure of data. Private data disclosed could be used for fraud or blackmail. If people don't feel safe using the system, they won't use it. Legal action could be taken if data is disclosed.

Integrity

Loss of data integrity can result in inaccuracy of reports, fraud, and erroneous decision making.

Availability

How often the database is up and able to be used. Downtime can result in financial loss and inconveniences. Ex: unavailability of medical records

Control measures

- Access control - who is allowed to access the db
- Inference control - even if identifying info is hidden, can identities be inferred with anonymous data?
- Flow control - can secure information be moved to a less secure object?
- Data encryption - how to encode data so that it is unreadable even if acquired

Database administrators are the central authority on our DB. They can create accounts, grant/revoke access privileges, and assign security levels. They're responsible for the security of the DB.

Most DBMS provide a method of logging activity.

Creating users

CREATE USER [username]@[host] IDENTIFIED BY [password]

Use localhost to only allow direct access and '%' for remote access

Can grant privileges to accounts through the kind of privilege and where it applies to. Can apply for all tables, specific tables, specific fields on tables. Can specify which hosts to allow privileges on. Can grant privileges to users to be able to use different SQL commands such as SELECT or INSERT. Can revoke privileges using REVOKE.

```
GRANT <Privilege1>, <Privilege2>, ... <PrivilegeN>
ON <Schema>.<Table>(<field>)
TO '<user>'@'<host>';
```

Can also grant user privileges with certain views. Ex: an employee has access to department info but only their department

Discretionary vs mandatory access control

Discretionary is used to grant access to specific users and is done on a user by user basis. Done through SQL. Downside includes not having control of the data after it has been accessed.

Mandatory has levels of security and divides users and data into these levels. Data also has access levels, so can continuously check if users have access to it. However, it is very rigid and not as applicable to many situations.

Role based access is a balance between these two.

Roles are created with different privileges and can be assigned to users. Users can have multiple roles. Security is tied to the type of actions the user would take.

To create roles:

```
CREATE ROLE [rolename];
```

Granting/revoking privileges is done with the same syntax as before but with role names.

Granting roles to users:

```
GRANT [rolename] TO [user]@[host];
```

SQL Injection

Giving the database SQL code to run that was not originally intended. Can log in or run any SQL code wanted, such as SELECT or DROP. Can also do this through a URL.

To guard against SQL injection attacks, validate user input by checking for unexpected characters such as ' ; and #. Use escape characters to replace characters by writing a function that places a backslash to escape any special characters. Use parameter binding to separate the query and parameters.

DBs will often only allow statistical queries. Cannot select fields directly to keep individual information protected and anonymous. However, by picking conditions to eliminate people we can narrow down the results and get a range or specific person. Can avoid this by not returning information if the number of people in the group is too low or partitioning the DB into groups with a minimum size and only allow queries to refer to the entire group.

Encryption

Convert data into encoded form for transmission. Use symmetric key encryption (user and host both have the same key to encrypt/decrypt data). Use public/asymmetric key encryption (public and private key for encrypting and decrypting).

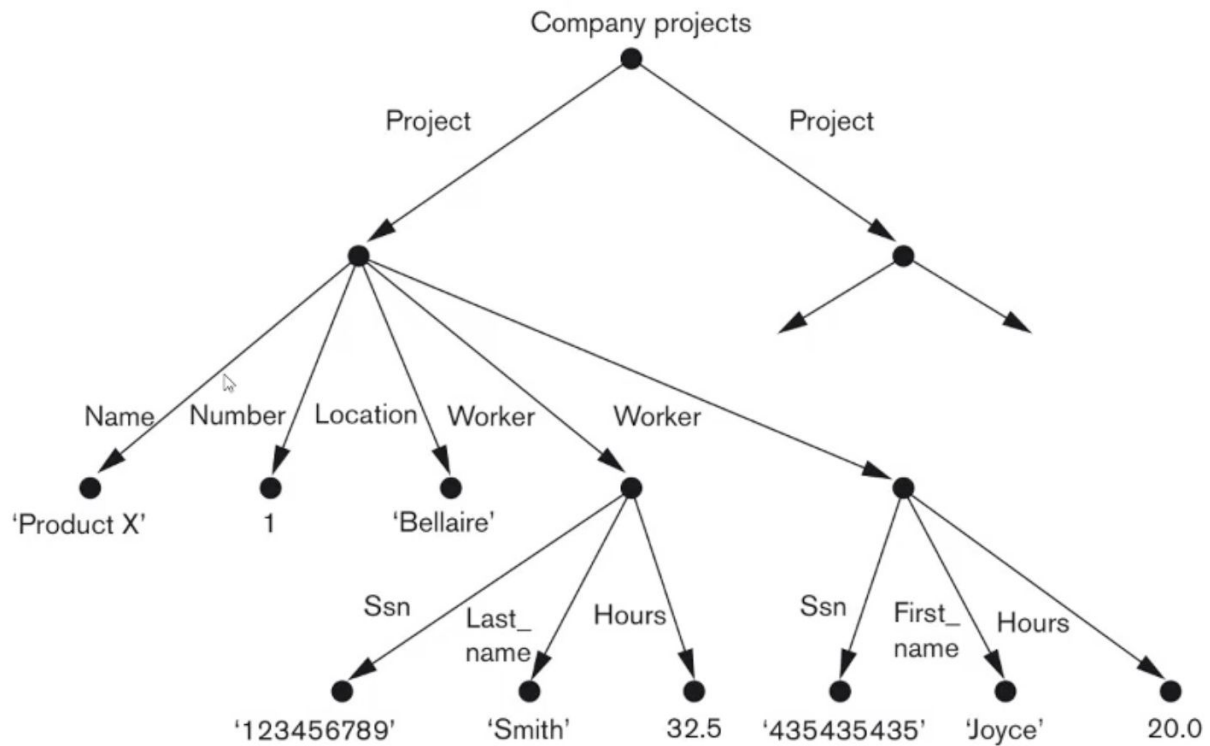
XML and JSON

Structured data - strict format. Examples include relational DB, where every record in the table follows the same format. DBMS enforces the structure of the table.

Semi-structured data- no predefined schema. Entities of the same type may have different attributes and new attributes can be introduced at any time. Is similar to a tree. The schema is mixed with the data.

Unstructured data - none/limited indication of data type. HTML have data in them but they all appear in tags that specify formatting.

Ex: of semi-structured data appearing as a tree:



XML - extensible markup language that describes the structure of data. It is self describing and can be used for both storage and retrieval. Semi structured.

XML can be easy to parse as it just looks for the tags.

A document is well-formed if it starts with an XML declaration and follows a tree model (one root, all elements have start/end tags, all elements are within start/end tags of root. A document is valid if it is well formed and follows a schema.

XML DTD - document type definition. Specifies elements and the structure of the document.

```

<!DOCTYPE Projects [
  <!ELEMENT Projects (Project+)>
  <!ELEMENT Project (Name, Number, Location, Dept_no?, Workers)>
    <!ATTLIST Project
      ProjId ID #REQUIRED>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Number (#PCDATA)>
  <!ELEMENT Location (#PCDATA)>
  <!ELEMENT Dept_no (#PCDATA)>
  <!ELEMENT Workers (Worker*)>
  <!ELEMENT Worker (Ssn, Last_name?, First_name?, Hours)>
  <!ELEMENT Ssn (#PCDATA)>
  <!ELEMENT Last_name (#PCDATA)>
  <!ELEMENT First_name (#PCDATA)>
  <!ELEMENT Hours (#PCDATA)>
]>

```

To enforce the schema, at the top of an XML document put:

<?xml version="1.0" standalone="no"?>. Standalone means that it needs to be checked against a DTD or schema.

<!DOCTYPE Projects SYSTEM "project.dtd"> says which DTD to use.

Downsides to DTD are that it has its own syntax, requires its own parsers, and has to have certain orders of elements.

XML schema uses XML syntax to specify the structure of XML documents, however this is a lot more complex.

XPath is a way of querying an XML document. Use elements to specify a path. / is a direct path, // is any path.

Ex:

/company - all of company

/company/department - all of department within company

//employee [employeeSalary gt 70000]/employeeName - names of employees with salary greater than 70k

/company/project/projectWorker [hours ge 20.0] - all project workers in the company who work more than 20 hrs a week

XQuery is a more robust querying in XML.

JSON, aka JavaScript Object Notation, is another syntax for storing/exchanging data. It is self describing semi structured data.

Syntax: `var [var name] = {"[attribute name]": [value], ...}`

To get data back out, `[var name].[attribute name]`

To set data, `[var name].[attribute name] = [new val]`

JSON vs XML

Similarities

- Self describing
- Hierarchical
- Able to be parsed
- Easily transmitted

Differences

- No end tag in JSON
- JSON is shorter
- JSON can use arrays
- JSON is easier to parse in JS

NOSQL - Not Only SQL

Developed to handle large amounts of data. They have good scalability and can add more nodes without disrupting the running service. The service must always be available so data is replicated. However, that replication may lead to consistency issues. Increased read performance, decreases write performance.

Master-slave replication model - one copy is master, all others are duplicates. Writes to master will spread changes to slaves.

Master-master replication model - every node is a master, rw on any node. Need a reconciliation model.

Sharding is the horizontal partitioning of files. This lightens the workload on each mode but makes consistency take more work. To find info quickly in millions of records, use hashing.

NOSQL requires no schema. It is self describing and semi-structured. It can use a schema if it wishes. They are less powerful than query languages (less overhead, more data). They can CRUD (create, read, update, delete). There are timestamps on data so that there is info on which is the most recent copy.

CAP theorem

CAP - consistency (is data consistent throughout replicated copies), availability (is DB available at all times), partition tolerance (can DB handle partitioning of nodes).

Document based NOSQL - stores data as collections of similar documents. Ex: MongoDB, CouchDB

Key-value NOSQL - each value has a key with it, can retrieve data very quickly

Column-based system - aka wide column. Ex: BigTable, Hbase.

Hbase - uses namespaces, tables, column families, column qualifiers

ColumnFamily:ColumnQualifier

In column based, can simplify to have repeated data in multiple cols and have it appear there in one command.

Create, put, scan, get.

Row vs columns

Row is better where you need many columns of a single row at one time. All the data for the row is given at once.

Columns are better when you need aggregate data for many rows and a small amount of columns. Also better when you need to add data to columns.

Graph based NOSQL - data represented by vertices and edges.

Vertices -> nodes -> entities.

Edges -> relationships -> relationship instances.

Nodes have label -> relationship types.

Nodes and relationships have properties -> attributes.

Properties can be grouped into map patterns.

Cypher syntax creates a node with a name, label, and list of properties. Create relationships between nodes.

Schemas are optional. Cypher is the query language, and has a fair amount of functionality.

Uses MATCH, WHERE, RETURN, DELETE, SET, and REMOVE.

Advantages and disadvantages of graphs.

Adv - good relationships, no predefined way of those relationships existing. Can add properties to those relationships. Can use graph searching algos. Reflects real life relationships well. Used in recommendation systems, social networks, and fraud detection.

Less functionality in NOSQL means it can handle more data and have more flexibility such as adding attributes on the fly, semi-structured data, and multimedia data.