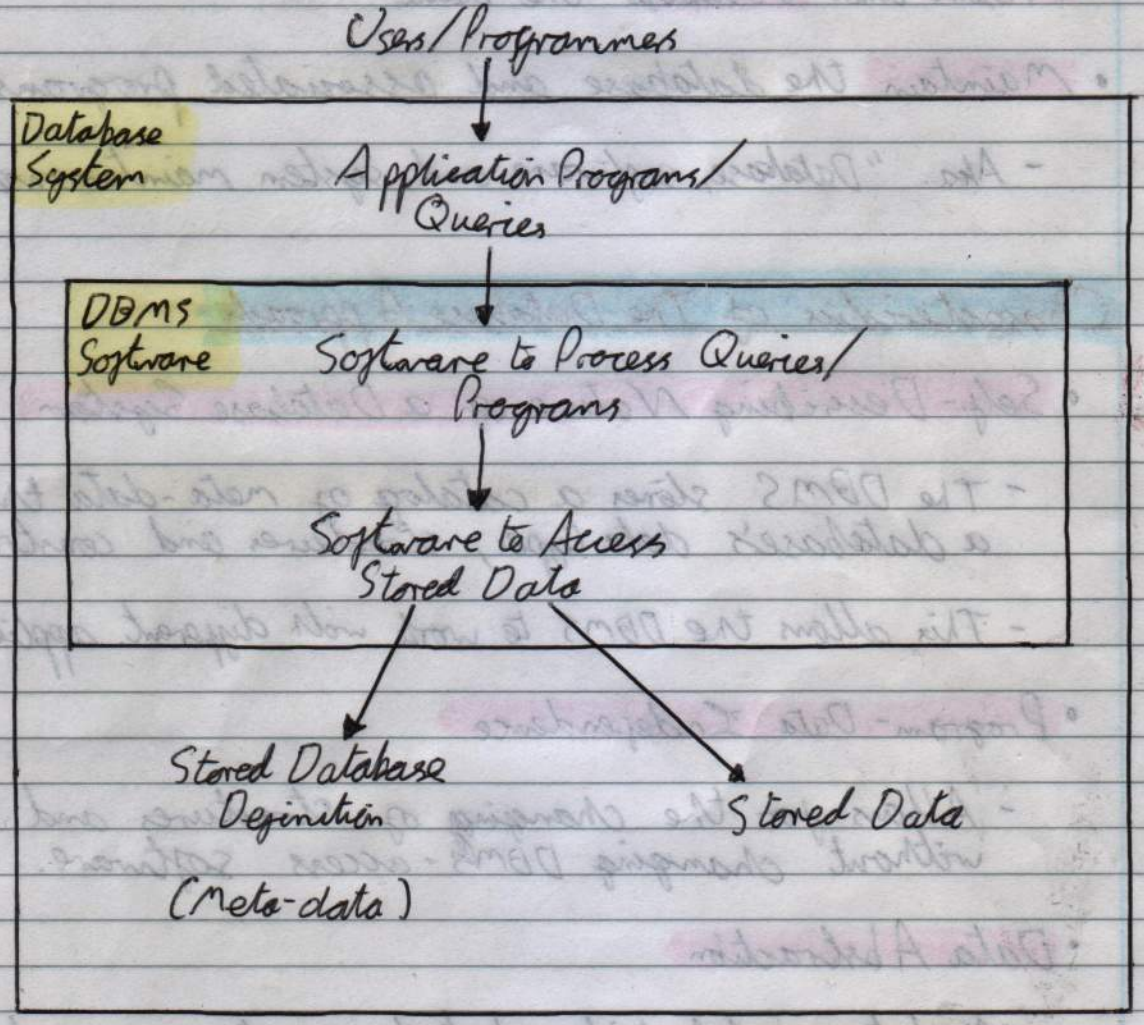
DATABASE SYSTEMS

Basic Definitions

- **Database**: a collection of related data
- **Data**: known facts that can be recorded and have meaning
- **Mini-world**: some part of the real world about which data is stored
- **Database Management System**: software to facilitate the creation / maintenance of a database
- **Database System**: the DBMS with the data itself, and sometimes applications

- **Simplified Database System**:



DBMS Functionality

- **Define** a database in terms of **data types, structures** and **constraints**.

- **Construct** or **load** the **initial database contents** on a secondary storage medium.

- Manipulate the database:
  - Retrieve data
  - Modify data
  - Access the data via applications
- Allow processing and sharing by a concurrent set of users and applications
- Protect and secure the data
- Carry out "active" processing on the data
- Present and visualise the data
- Maintain the database and associated programs
  - Aka. "Database, software and system maintenance".

Characteristics of The Database Approach

- Self-Describing Nature of a Database System
  - The DBMS stores a catalog of meta-data that describes a database's data types, structures and constraints.
  - This allows the DBMS to work with different applications
- Program-Data Independence
  - Allows for the changing of structures and organisation without changing DBMS access software.
- Data Abstraction
  - A data model hides details of storage and provides a conceptual view.
  - Programs refer only to the data model
- Support for Multiple Views of Data
  - Each user can see a view with only the data relevant to them.

- Sharing Data / Multi-User Transaction Processing
  - Concurrent users can manipulate the data
  - Concurrency control is used to ensure that each transaction is executed or aborted.
  - A recovery subsystem ensures that the effect of each transaction is permanent.
  - OLTP (Online Transaction Processing) ensures that hundreds of concurrent transactions can execute per second.

Advantages of the Database Approach.

- Controlling redundancy in storage, development and maintenance
- Providing multiple interfaces.
- Restricting access.
- Providing structures (eg. indexes) for efficient processing
- Providing backup/recovery services
- Representing complex relationships
- Enforcing integrity constraints
- Deductive and active rules to draw inferences and actions

- Additional Implications

- Potential for enforcing standards
  - Data item names, display formats, meta-data, etc.
- Reduced application development time
  - Each application can use the DB that is already developed.
- Flexibility to change data structures
  - Structure is free to evolve as new requirements are defined.

- Availability of Current Information
  - Each client can have access to a real-time dataset
- Economies of Scale
  - Consolidating data and applications reduces overlap of resources.

When Not to Use a DBMS

- Main Inhibitors:
  - High initial investment
  - Overhead for providing generality, security, concurrency control, recovery, integrity, etc...
- When a DBMS may be Unnecessary:
  - If the database and application are simple, well-defined and unlikely to change.
  - If real-time requirements may be broken by the DBMS overhead.
  - If concurrent access is not required.
- When a DBMS May Not Suffice:
  - If modelling limitations mean the complexity of the data cannot be handled.
  - If users need special operations not supported by the DBMS.

Data Models

- Data Model: a set of concepts to describe...
  - the structure of a database
  - the operations for manipulating these structures
  - the constraints that a database should obey
- Data Model Structure and Constraints
  - Constructs define the DB structure
  - Constructs include elements and their data types, groups

of elements (records, entities, tables, etc) and the relationships between these groups.
- Constraints define restrictions on valid data that must be followed at all times.

• Data Model Operations

- Specify CRUD actions by referring to data model constructs.
- Two types:
  - Basic operations (generic CRUD)
  - User-defined operations (implementation-specific)

- Data Model Categories

• Conceptual Models (high-level, semantic)

- Provide concepts close to how users perceive data
- aka. entity-based or object-based models

• Physical Models (low-level, internal)

- Provide details of how data is stored
- Usually specified through DBMS design and admin manuals

• Implementation Models (representational)

- Provide a middle ground between the two above
- Used by many commercial DBMS implementations.

- Types of Data Model

- Relational Model
- Hierarchial Model
- Object-Oriented Model
- Network Model
- Object-Relational Model

• Network Model

- Pros:
  - Can model complex relationships
  - Can handle most situations for record types and relation types
  - Navigational language - programmers can optimise code.

- Cons:
  - Navigational/procedural nature of processing
  - Database contains a complex array of pointers
    - No scope for query optimisation.

Database Schema

- **Schema**: description of the DB, including structure, data types and constraints.
- **Schema diagram**: illustrative display of a schema
- **Schema construct**: a component of a schema or an object within it

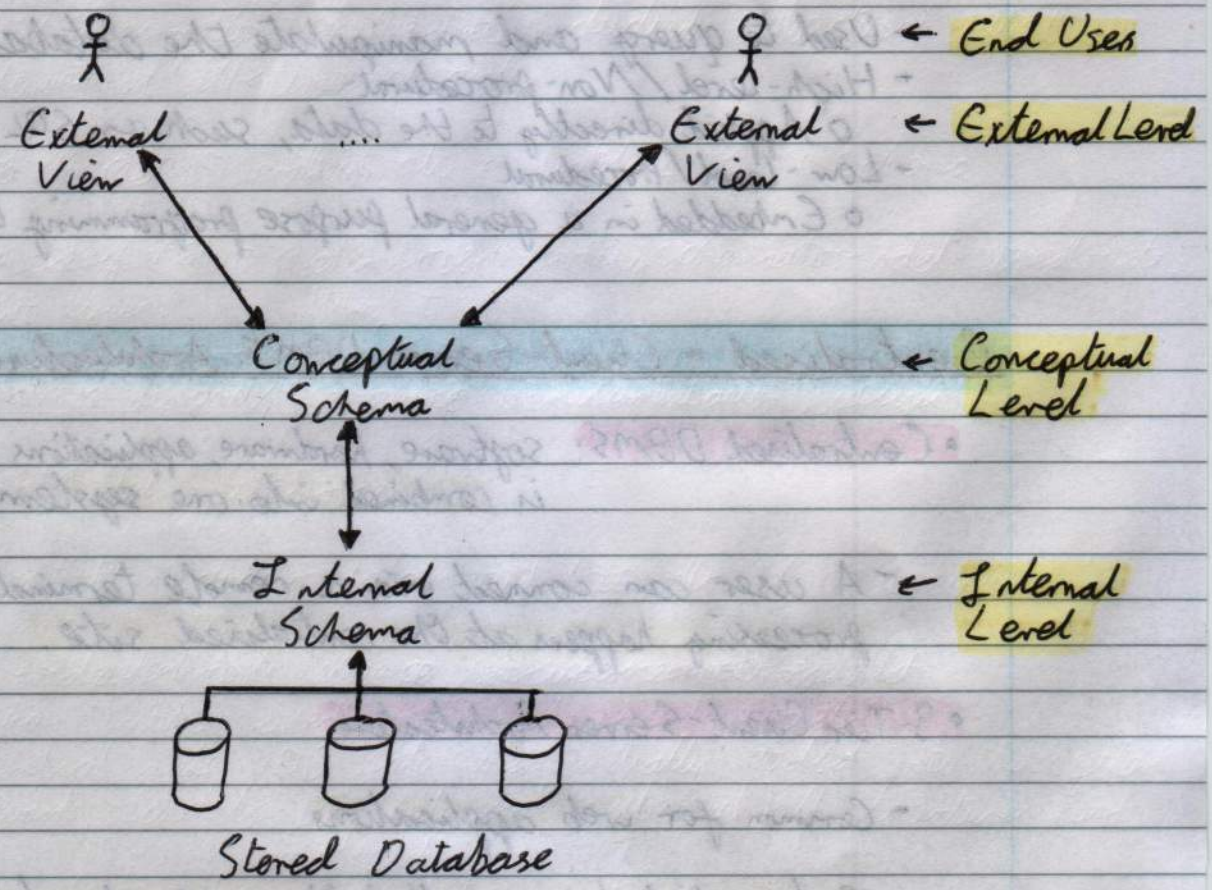
- Schema vs. State/Instance

- **DB State**
  - The DB, including data, as it exists at a particular moment in time
  - Also called an instance, occurence or snapshot
  - Initial State: DB state when initially loaded onto the system
  - Valid State: a DB state that satisfies all structures and constraints
- **The difference**
  - Schema changes infrequently
  - State changes every time the data is modified
  - Schema aka intension
  - State aka extension

Three-Schema Architecture

- Supports program-data independence and multiple views of data
- Defines schemas at three levels:
  - Internal schema for physical storage details (plus indexes, etc.)
    - Normally uses a physical data model
  - Conceptual schema for structures/constraints of the database
    - Uses a conceptual or implementation data model
  - External schema for user views
    - Uses the same ~~schema~~ model as the conceptual schema.

- The TSA:



- Mappings are needed between levels to transform requests and data.
  - Programs use an external schema, so their requests are mapped by the DBMS to the internal schema.
  - Data taken from the DBMS layer is reformatted to suit the external view

DBMS Languages

- Data Definition Language (DDL)
  - Specifies conceptual and internal schemas
  - Used by the DBA and database designers
  - Sometimes used for external schemas
  - Some DBMSs split the work:
    - Storage Definition Language (SDL)
    - View Definition Language (VDL)

- Data Manipulation Language (DML)
  - Used to query and manipulate the database
  - High-level / Non-procedural
    - Applied directly to the data, such as SQL
  - Low-level / Procedural
    - Embedded in a general purpose programming language.

Centralised + Client-Server DBMS Architectures

- Centralised DBMS: software, hardware, applications and UI processing is combined into one system
  - A user can connect via a remote terminal, but all processing happens at the centralised site.
- 3-Tier Client-Server Architecture
  - Common for web applications
  - Intermediate layer called the application/web server
    - Stores connectivity software and business logic to access data from the database server
    - Acts as a middle-man for sending requests and data between the client and server.
  - Can enhance security
    - Clients have no direct access to database

Classification of DBMSs

- Based on the data model used:
  - Traditional: relational, network, hierarchical
  - Emerging: object-oriented, object-relational
- Other classifications:
  - Single-user vs. multi-user
  - Centralised vs. distributed
- Variations of distributed DBMSs (DDBMSs):
  - Homogeneous - same DBMS at all sites
  - Heterogeneous - several autonomous DBMSs at multiple sites, connected via a network
  - Federated/Multidatabase - DBMSs loosely coupled, with some autonomy.

Database Design Process Overview

- Two parts of this:
  - Database design
  - Application design
- Database design involves developing the conceptual schema for database applications.
- Application design focusses on the programs and interfaces
  - Normally considered to be part of s/w engineering.
- Much of the process is based on data requirements and functional requirements.

Entity-Relationship Diagrams

- ERDs can be used to show database design using the ER model.

- ER Model Concepts

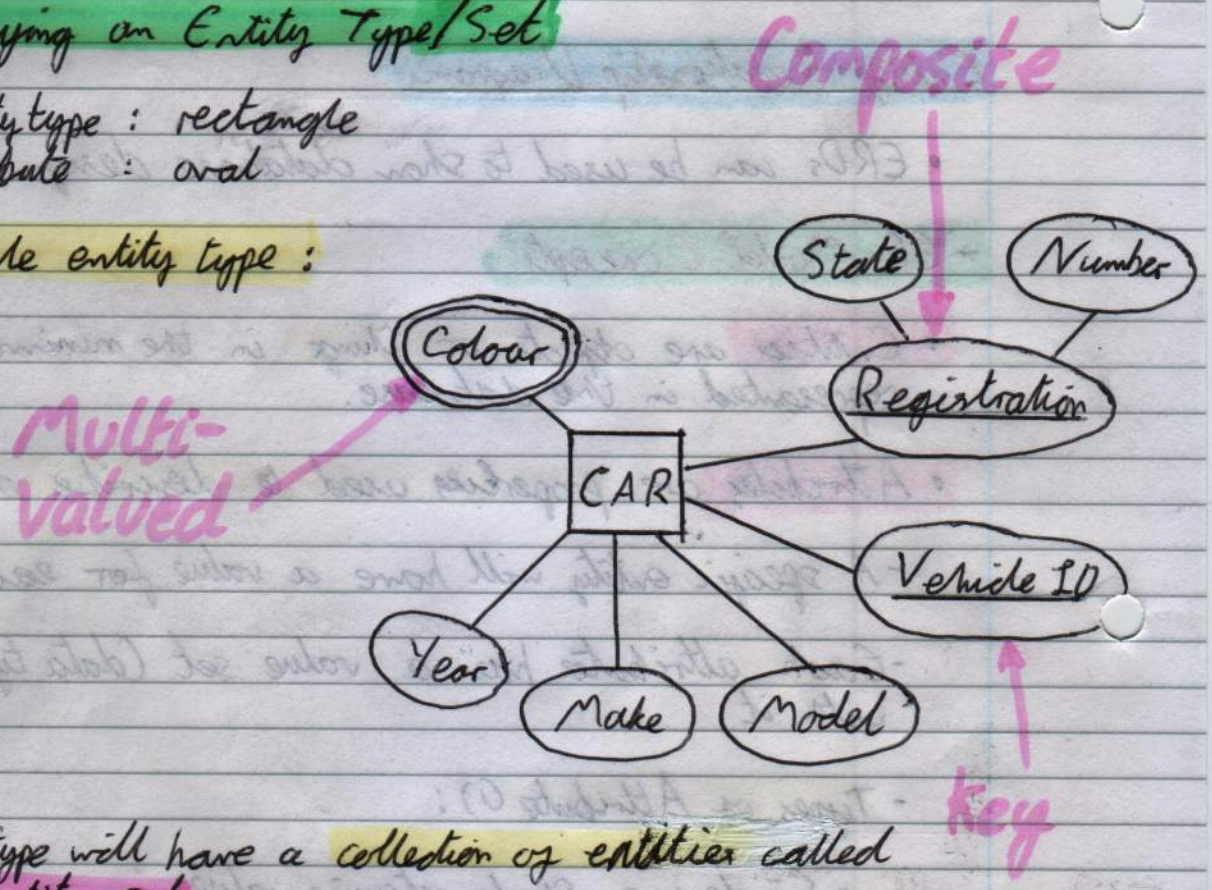
- Entities are objects or things in the miniworld that are represented in the database.
- Attributes are properties used to describe an entity.
  - A specific entity will have a value for each attribute.
  - Each attribute has a value set (data type) associated with it
  - Types of Attribute (1):
    - Simple: a single atomic value
    - Composite: made of several parts, such as an address
      some parts may also be composite
  - Types of Attribute (2):
    - Single-valued
    - Multi-valued
- Composite and multivalued attributes can be nested to any level, although this is rare.

- Entity Types and key Attributes

- Entities with the same basic attributes are grouped into an entity type.
- Each entity in an ~~ett~~ entity type must have at least one attribute (or collection of attributes) that is unique to that entity - called a key attribute.
- A key attribute may be composite.
- An entity may have multiple keys.
- Each key is <u>underlined</u> in an ERD.

- Displaying an Entity Type/Set

- Entity type : rectangle
- Attribute : oval
- Example entity type :



- Each type will have a collection of entities called the entity set.
- Example entity set:

CAR

Year, Make, Model, Registration (State, Number), ...

Car_1 : 2001, Ford, Fiesta, (NY, 123) ...

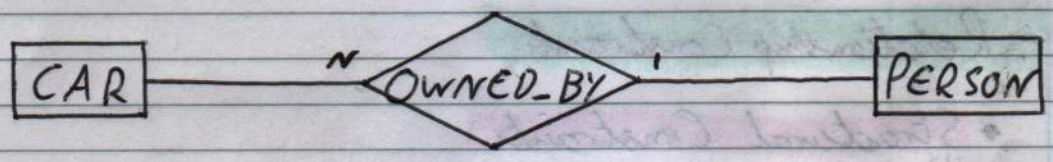
Car_2 : 2014, Fiat, Punto, (CA, 789) ...

- Relationships and Relationship Types

- A relationship relates two or more distinct entities with a specific meaning.
- Relationships are grouped into relationship types.
- The degree of a relationship type is the number of participating entities.
- Relationships are:
  - One-to-one (1:1) or
  - Many-to-one (N:1) or
  - Many-to-many (M:N)
- Relationship Type vs. Relationship Set.
  - The type identifies the schema of the relationship by specifying the relationship name, participating entities and constraints.
  - The set is the current set of relationship instances in the database (i.e. the relationship state).

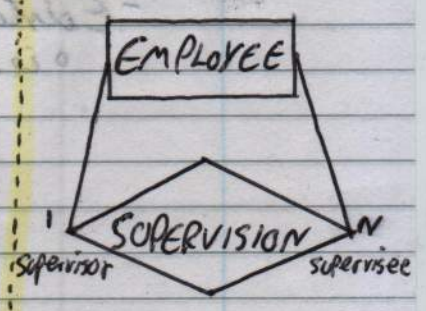
- Displaying Relationships

- A diamond box shows a relationship type.
- Example relationship type:



- Recursive Relationships

- A relationship with the same entity type in distinct roles.
- Eg. the supervision relationship has two roles:
  - Supervisor
  - Supervisee
- In an ERD, the role names must be displayed.



- Weak Entity Types

- An entity that does not have a key attribute.
- A weak entity must participate in an identifying relationship type with an identifying entity or owner.
- Weak entities are identified by the combo of:
  - A partial key of the weak entity type
  - The specific entity they are related to in the identifying entity type.
- Example:
  - A DEPENDENT (of an employee) is a weak entity.
  - It can be identified by its name (the partial key) and the EMPLOYEE to which it is related (the identifying entity).

- Relationship Attributes

- A relationship type can have attributes, which have a value for each instance of that relationship type
- Most relationship attributes are used in M:N relationships.
  - In a N:1 relationship, they can be transferred to the entity type on the N side.

- Relationship Constraints

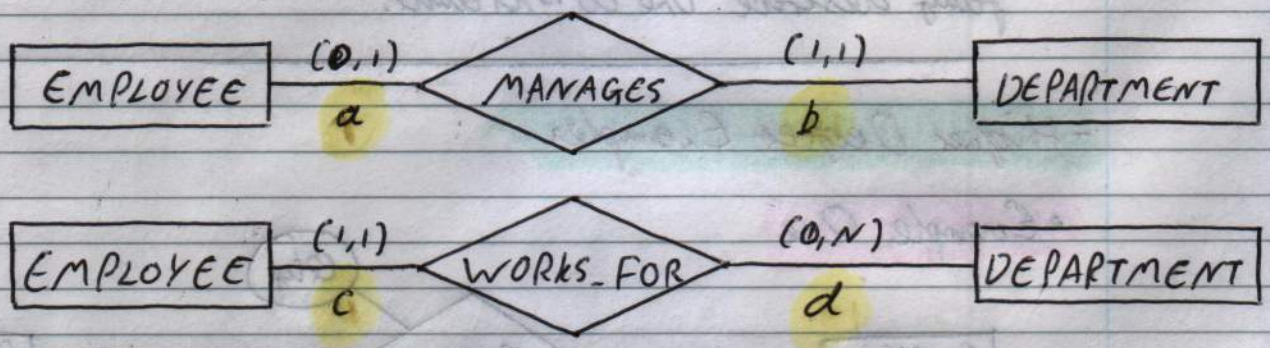
- Structural Constraints
  - Cardinality Ratio: describes the max number of relationship instances that an entity can participate in (1:1, M:N, etc)
  - Existance Dependency: specifies minimum participation
    - ie. the existence of an entity depends on it being related to another entity.

- Notation for Constraints

- **Cardinality** : 1:1, 1:N, N:1, N:M
  - Shown by placing numbers on the relationship edges.

- **Participation** (on each participating entity type
  - Total is shown by a double line (existence dependency)
  - Partial is shown by a single line.

- Alternative **(min, max)** notation:



  - Specifies how many times each entity can be part of a relationship, ie. how many instances they can participate in.

  - Eg: a) an employee can manage at most one department
    b) each department has exactly one manager
    c) an employee works for exactly one department
    d) each department can have any number of employees.

- Higher Degree Relationships

- Constraints are harder to specify for n-ary relationships when $n > 2$

Example One
- Three binary relationships can represent different information compared to a single ternary relationship.
  - If needed, both binary and n-ary relationships can be shown if they convey different meanings.

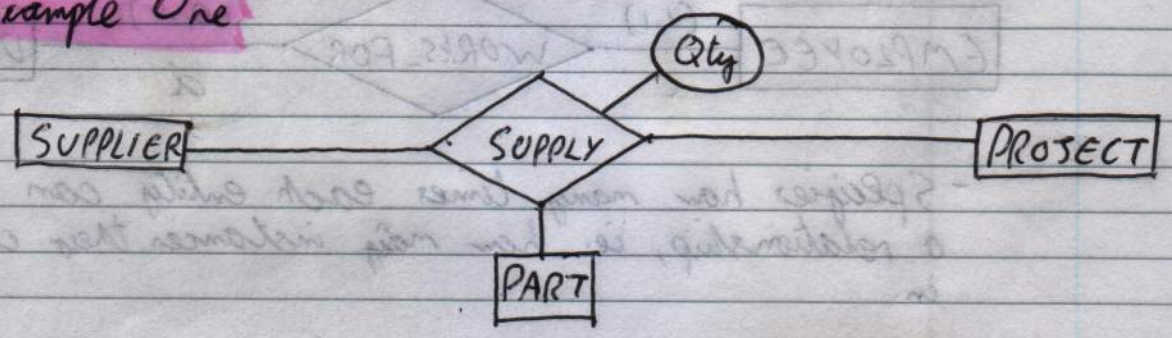
Example Two
- In some cases, a ternary relationship can be shown as a weak entity, if the model allows the weak entity to have multiple identifying relationships.

- If a given binary relationship can be derived from a higher degree relationship at all times, it is redundant.

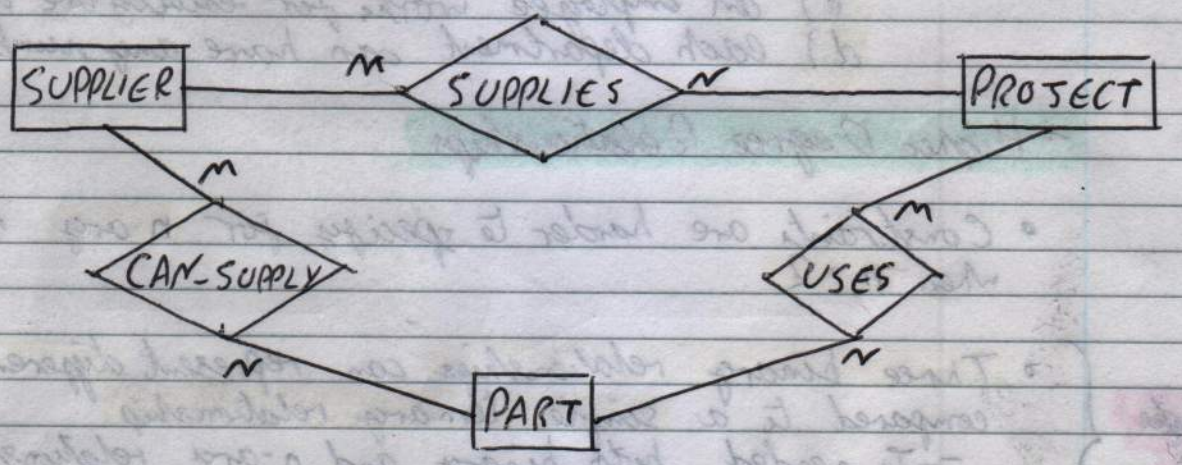
- For $n>2$ n-ary relationships, (min, max) notation does not fully describe the constraints.
  - Adding 1, M or N shows additional constraints
    - A 1 indicates that an entity can participate in at most 1 relationship instance that has the particular combination of other participating entities.
- In general, (min, max) and 1/M/N are needed to fully describe the constraints.

- Higher Degree Examples

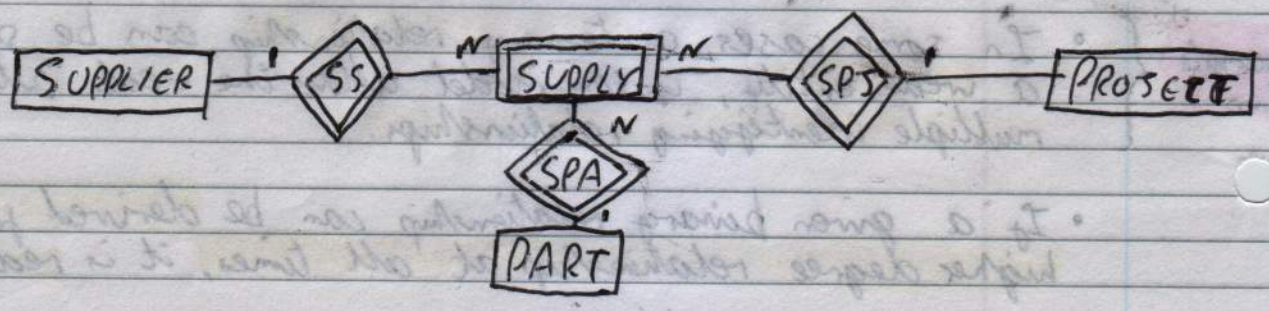
• Example One



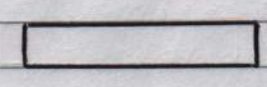
vs...



• Example Two



ERD Summary



Entity

Weak Entity

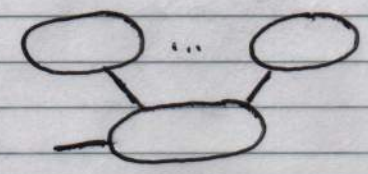
Relationship

Identifying Relationship

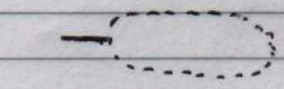
Attribute

key Attribute

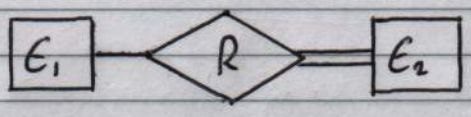
Multi-Valued Attribute



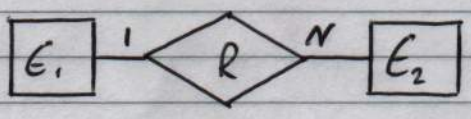
Composite Attribute



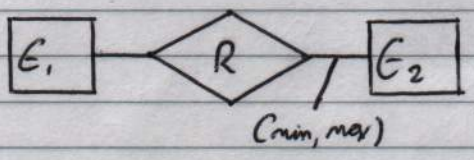
Derived Attribute



Total Participation of E_2 in R



Cardinality Ratio of 1:N for $E_1 : E_2$ in R



Structural Constraint (min, max) on Participation of E_2 in R.

Alternative Notation - UML

- UML represents classes (similar to entities) as 3-section boxes:
  - The class name
  - The class attributes
  - The class methods (not part of the basic ER Model)
- Associations (relationships) are shown as lines connecting classes.
- Used in database design and OO software design.

Relational Model of Data

- We review the formal model, but the standard model in SQL has several differences.
- A relation is a mathematical concept based on the ideas of sets.

Informal Definitions

- A relation looks like a table of values
- A relation usually contains a set of rows
- The data items in each row represent facts about a real-world entity or relationship.
  - Formally, rows = tuples
- Each column has a header describing the data in that column.
  - Formally, column header = attribute name / attribute
- Each row has a data item (or set of items) that uniquely identifies it - called the key.
  - Row IDs may be created and assigned, called surrogate keys or artificial keys.

Formal Definitions

- Schema
  - Denoted by $R(A_1, A_2 \dots A_n)$ where R is the relation name and $A_1 \dots A_n$ are attributes.

- **Domain**
  - The set of valid values for each attribute, possibly with a set format.
- **Tuple**
  - An ordered set of values, represented as $\langle v_1, v_2, \ldots, v_n \rangle$
  - A relation is a set of tuples (rows)
- **State**
  - Represented as $r(R)$ where R is the relation name.
  - A subset of the Cartesian Product of the domains of its attributes.

Relation Characteristics

- Tuples in a table are not considered ordered.
- Values in a tuple are considered ordered.
- Attributes in a relation schema are considered ordered.
- Values in tuples are considered atomic (indivisible).
- A null value can be used for tuple values that are unknown or not applicable.
- Component values of a tuple t are represented as $t.A_i$ or $t[A_i]$.
- A sub-tuple of t is represented as $t[A_i, A_j \ldots A_n]$
- A database is a set of relations.
  - $D = \{R_1, R_2 \ldots R_n\}$

Formal Definitions - Example

PRODUCT ← Relation Name

Attribute

Name	Price	QTY
Super Wow	20	0
Much Lol	30	12
Many Amaze	50	8

} Relation State

Product (Name, Price, QTY) ← Relation Schema

dom (Name) = varchar (25)
dom (Price) = $\mathbb{R}$, >0
dom (QTY) = $\mathbb{I}$, >0

} Domains

$r(\text{Product}) \subset \text{dom}(\text{Name}) \times \text{dom}(\text{Price}) \times \text{dom}(\text{QTY})$

↑ State (formal)

Cartesian Product

r (Product) = { <"Super Wow", 20, 0>,
<"Much Lol", 30, 12>,
<"Many Amaze", 50, 8> }

<...> = A tuple

For this tuple...

t[Name] = "Super Wow"

or

t.Name = "Super Wow"

Informal	Formal
Table	Relation
Column	Attribute
Row	Tuple
Possible Column Values	Domain
Table Definition	Relation Schema
Populated Table	Relation State

Relational Integrity Constraints

- Conditions that must hold on all valid relation states.

- Three main types:
  - key constraints
  - Entity integrity constraints
  - Referential integrity constraints

- Other types:
  - Semantic Integrity Constraints.
  - Domain Constraints

• key Constraints

- Next Page.

• Entity Integrity Constraints.

- The primary key attributes of any relation R cannot have null values in any tuple of r(R).

- Other attributes in a relation may also be constrained to disallow null values.

• Referential Integrity Constraints.

- Specifies a relationship between tuples in two relations:
  - The referencing relation
  - The referenced relation

- A set of attributes in R_1 is a foreign key of R_1 that references R_2 if:
  - The attribute(s) of Fk in R_1 and the attribute(s) of Pk in R_2 have the same domain(s).
  - The value of Fk in a tuple from $r(R_1)$ is the same as the value of Pk in a tuple from $r(R_2)$, or null.

- A ref. int. constraint is shown on an DB schema as a directed arc from R_1.Fk to R_2.Pk.

• Semantic Integrity Constraints

- Based on application semantics, eg. "max hours per week for an employee is 56"
- Cannot be expressed by the model, requires constraint specification language
  - Eg. SQL-99 allows "ASSERT" operations.

Update Operations on Relations

- Insert, Delete or Modify a tuple
- These should not violate integrity constraints:
  - Several operations may have to be grouped together.
  - Updates may propagate to cause other updates automatically.
- In case of violation, several actions may be taken:
  - Cancel the operation (RESTRICT or REJECT option)
  - Perform the operation but inform the user
  - Trigger additional updates to correct the damage (CASCADE or SET NULL option)
  - Execute a user-specified routine to correct the errors.
- Possible Violations - INSERT
  - Domain Constraint - one or more attribute values may be outside the specified domain.
  - Key Constraint - key collision
  - Referential Integrity - An Fk value may reference a Pk that does not exist
  - Entity Integrity - may specify null for a non-null attribute, such as the Pk.
- Possible Violations - DELETE
  - Referential Integrity - may delete a tuple referenced by other tuples in the DB.
    - Possible solutions:
      - RESTRICT - cancel deletion
      - CASCADE - delete referencing tuples
      - SET NULL - set Fk in referencing tuples to null.

- Possible Violations for UPDATE/MODIFY on <u>next page</u>.

Key Constraints

- A superkey is any collection of 1 or more attributes, for which there are no two tuples in a valid relation state that have the same values for those attributes.

- A minimal superkey (aka. key) is a key formed of the minimum set of attributes needed to uniquely identify a tuple.

- In a relation with multiple minimal superkeys, each is called a candidate key and one is chosen to be the primary key, which is underlined.

<u>Example</u>:

USER (<u>User_ID</u>, Nat_Ins_Num, Name, Email, Phone)

Minimal Superkeys:
- User_ID
- Nat_Ins_Num

Superkeys:
- {User_ID, Name}
- {Nat_Ins_Num, Phone}
- User_ID,
- {User_ID, Nat_Ins_Num}

etc...

Minimal Superkeys $\subset$ Superkeys

- The primary key is the tuple's identifier and is used to refer to it in other tuples or relations.

- Possible Violations - UPDATE/MODIFY

- Domain / NOT NULL constraint violation for any attribute.
- Updating PK
  - Similar to DELETE followed by INSERT
  - Need to specify similar options as DELETE
- Updating FK
  - May violate referential integrity.

Structured Query Language (SQL)

- Used for DDL and DML.
- Can specify authorisation, security, integrity constraints, views, transaction controls, etc.

Formal	In SQL
Relation →	Table
Attribute →	Column
Tuple →	Row

SQL: Data Definition, Constraints, Schema Changes

- SQL is used to CREATE, DROP and ALTER tables.

CREATE Schema:

- Creates a DB schema by giving it a name.
- Also specifies authorisation.

```
CREATE SCHEMA COMPANY AUTHORIZATION JSMITH;
```

CREATE Table

- Creates a base table by giving it a name and specifying attributes and their data types.
- Initial constraints are also specified.
- Can specify primary keys, secondary keys and foreign keys.

```
CREATE TABLE DEPT (
  Dept_Name        VARCHAR(15) NOT NULL,
  Dept_No          INTEGER NOT NULL,
  Mgr_SSN          CHAR(9),
  Mgr_Start_Date   DATE,
  PRIMARY KEY (Dept_No),
  UNIQUE      (Dept_Name),
  FOREIGN KEY (Mgr_SSN) REFERENCES EMPLOYEE(SSN)
);
```

Format: Column_Name DATATYPE(N) [CONSTRAINTS]

↑ Optional

- **Base tables** correspond to a file stored in the DBMS
  - Created via CREATE TABLE
- **Virtual tables** do not correspond to a file
  - Created via CREATE VIEW

Attribute Data Types

- Numeric
  - INT, FLOAT, REAL, etc...
- Character String
  - Fixed Length: CHAR(n), CHARACTER(n)
  - Variable length (up to n): VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
- Bit String
  - BIT(n), BIT VARYING(n)
- BOOLEAN
  - True or false
- DATE
  - In the form yyyy-mm-dd
- TIME
  - In the form hh:mm:ss
- TIME(i)
  - In the form hh:mm:ss.$\overbrace{xx\ldots x}^{i}$
- TIMESTAMP
  - Has DATE and TIME components
- INTERVAL
  - Specifies a relative (rather than absolute) time/date value
  - Can be TIME, DAY, MONTH or YEAR intervals
  - Can be +ve or -ve

Integrity Constraints

- NOT NULL
- DEFAULT value
- CHECK clause
  - Eg. Dept_No INT NOT NULL CHECK (Dept_Num > 0 AND Dept_Num < 21)
  - Eg. CREATE DOMAIN D_Num AS INTEGER CHECK (D_Num > 0 AND D_Num < 21);

    CREATE TABLE Dept (
    Dept_Num D_Num,
    ...
    );

    Not used often

Referential Integrity Constraints

- Ref. int. may be violated when rows/tuples are inserted, modified or deleted.
- Default behavior is to reject the violating behaviour.
- Or specify a referential triggered action as "OPTION ACTION"

Options	Actions
ON DELETE	RESTRICT
ON UPDATE	CASCADE
	SET NULL
	SET DEFAULT

RESTRICT → Dropped only if no elements in the schema

CASCADE → All tables, views and constraints are dropped

- These options/actions can be set for any foreign key.

DROP TABLE

- Deletes a base table and its definition.
- Options:
  - RESTRICT: table is dropped if not referenced in any constraints
  - CASCADE: table is dropped along with any constraints that reference it.

ALTER TABLE.

- Used to add an attribute to a table
- Eg. ALTER TABLE DEPT ADD Office_Phone VARCHAR(11);
- The new attribute will be created with NULL in each tuple, and hence the modifier NOT NULL is not allowed.

SQL: Retrieval

- Note: bags vs. sets vs. lists
  - Set: order irrelevant, no duplicates
  - List: order important, duplicates okay
  - Bag: order irrelevant, duplicates okay ← most relaxed

Basic Select

- A basic SELECT statement is called a mapping or a SELECT-FROM-WHERE block.
- Form:

```
SELECT <attributes>
FROM   <tables>
WHERE  <condition>
```

Select with Joins (Relational Database)

- ~~[illegible]~~
- Multiple tables can be queried:

```
SELECT F_Name, L_Name, Address,
FROM Employee, Department
WHERE Department.Name = "Research"
      AND Employee.Dep_No = Department.Dep_No ;
```

- Similar to SELECT-PROJECT-JOIN in relational algebra operations. (covered later)
- Where unique, attribute names don't have to be qualified by their relation name, but doing so improves readability.

Aliases

- Some queries refer to the same table twice, so aliases are used.

```
SELECT E.F_Name, E.L_Name, S.F_Name, S.L_Name,
FROM EMPLOYEE E S,
WHERE E.Supervisor_SSN = S.SSN
```

- E and S are two copies of employee: the employee and their supervisor.

- Can also be used with the AS keyword, and can be used for convenience:

```
SELECT E. ..., D. ...
FROM EMPLOYEE AS E, DEPARTMENT AS D,
WHERE ...
```

Unspecified WHERE

- A missing WHERE clause is the same as WHERE TRUE
  - It will select everything.

- If more than one relation/table is used, it will select the Cartesian Product of those tables.
  - This could result in a huge dataset

Use of *

- SELECT * means "select all of the attributes"

Use of DISTINCT

- SQL does not treat a relation as a set and duplicate tuples can be returned.

- To eliminate duplicates, the keyword DISTINCT is used:

```
SELECT DISTINCT ...
FROM ...
WHERE ...
```

Set Operations

- There is a UNION operator, and in some version of SQL there are INTERSECT and MINUS (set difference) operators.
- The resulting data is a set - duplicates are removed.
- Set operations only work on union-compatible relations:
  - Same attributes
  - In the same order

Substring Comparison

- WHERE E.F_Name LIKE "B%"
  - "where first name starts with B"
- Two reserved characters:
  % - match any number of characters
  _ - match one character

Arithmetic

- The operators + - * / can be applied to any numeric values.

ORDER BY

- Used to sort the resulting tuples/rows.
- Defaults to ascending, but the modifiers ASC and DESC can be used.

```
SELECT ...
FROM ...
WHERE ...
ORDER BY F_Name DESC;
```

Argument Order

Required { SELECT ...
FROM ...

Optional { WHERE ...
GROUP BY ... } Covered later
HAVING ... } Covered later
ORDER BY ...

SQL: Updates

Insert

- Adds one or more tuples (rows) to a table (relation).
- Format:

```
INSERT INTO Table VALUES ("val1", 28, 42, "val4");
```

A

OR

```
INSERT INTO Table (Column 1, Column 4, Column 5)
  VALUES (12, 15, 16);
```

B

OR

```
INSERT INTO Table (Column 1, Column 4, Column 5)
  VALUES (12, 15, 16), (18, 4, 5);
```

OR

```
INSERT INTO Table (Column 1, Column 4, Column 5)
  SELECT Column 6, Column 8, Column 2 FROM OtherTable
  WHERE...
```

Note A: Must be in the same order as specified for CREATE TABLE.

Note B: Only null-able or columns with a default value may be skipped.

Delete

- Includes a WHERE clause. - no WHERE clause = WHERE TRUE.
- Referential integrity should be maintained.
- Unless CASCADE is specified on a ref. int. constraint, tuples are deleted from one table at a time.
- Format:

```
DELETE FROM Table WHERE Column = "Value";
```

OR

```
DELETE FROM Table WHERE Column IN (SELECT Col
  FROM OtherTable);
```

Update

- Modifies attributes via the SET clause, constrained by the WHERE clause.
- Works in a single relation at a time.
- Ref. int. should be enforced.
- Format:

UPDATE Table SET Column = "Value" WHERE <clause> ;

SQL: Nested Queries

- A complete SELECT query can go inside the WHERE clause of another query.
  "nested" "outer"
- The comparison operator IN can also be used.
- Queries can be nested to several levels.
- An unqualified attribute refers to the relation in the innermost query.

Correlated Nested Queries

- If a condition in the WHERE clause of a nested query references an attribute of a relation declared in the outer query, they are said to be correlated.
- A query written with nested SELECT-FROM-WHERE blocks and using = or IN can always be written as a single-block query.
- Old SQL systems had a CONTAINS operator that compared two sets of values, returning TRUE if one set contains everything from the other.
  - This is no longer used.

SQL: EXISTS

- Checks whether the result of a correlated nested query is empty.
- Can be negated: WHERE NOT EXISTS (...)

Explicit Sets

- Explicitly defined sets can also be used:

... WHERE Column IN (2, 4, 9, 81);

Null values

- A WHERE-clause can check if an attribute is null with:

... WHERE Column1 IS NULL AND Column2 IS NOT NULL;

SQL: Joined Relations

- A joined relation can be specified in any FROM clause.
- Various join types:
  - "Normal" JOIN: an ON parameter specifies which attributes are used to join tuples.

    Eg: FROM (Employee JOIN Department ON Dep_No = D_No)
  - NATURAL JOIN: the same as JOIN but with no ON parameter; attributes with the same name are used to join.
  - INNER JOIN (default): a tuple is included in the result only if a matching tuple is found in the other relation.
  - OUTER JOIN: all matching tuples are returned
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

  } Covered later

SQL: Aggregate functions

- COUNT, SUM, MAX, MIN and AVG

SQL: Grouping

- Useful if we want to apply aggregate functions to sub-groups of tuples in a relation.
- Each sub-group has the same value for the grouping attribute(s).
- Eg. SELECT COUNT(*) FROM Table GROUP BY Column;
- The grouping attribute(s) must appear in the SELECT clause.

HAVING

- The HAVING keyword specifies conditions on the sub-groups, rather than single tuples.
- Eg. SELECT COUNT(*) FROM Table GROUP BY Column HAVING COUNT(*)>2;

SQL: Summary

- SELECT: lists attributes or functions to be retrieved.
- FROM: lists relations (or aliases) needed in the query.
- WHERE: specifies conditions for the selection and joining of tuples.
- GROUP BY: specifies grouping
- HAVING: filters grouping
- ORDER BY: orders results.

SQL: Views

- A virtual table derived from others.
- Limited updates, but full querying.
- Mostly used as a convenience.
- SQL:

```
CREATE VIEW Name AS
SELECT ... FROM ... WHERE...
GROUP BY ... ;
```

} Create

```
SELECT Column FROM Name WHERE ... ;
```

→ Query

```
DROP Name;
```

→ Delete

Relational Algebra

- Operations in RA specify queries.
- The algebra is "closed" - the result and input of each operator is a relation.
- A sequence of operations forms an expression.

Overview of Symbols

- Unary operators:

SELECT	σ	(sigma)
PROJECT	π	(pi)
RENAME	ρ	(rho)

- Set operations:

UNION	$\cup$
INTERSECTION	$\cap$
DIFFERENCE	$-$
CART. PRODUCT	$\times$

- Binary Relational Operators:

JOIN	(several styles exist)	$\bowtie$
DIVISION		$\div$

- Additional Operators

OUTER JOINs
OUTER UNION
MIN
MAX
AVG
SUM
COUNT

SELECT σ

- $\sigma_{\langle CONDITION \rangle}(R)$ will select all tuples from R that match the given condition.
- $\sigma(R)$ will select <u>all</u> tuples from R.
- Conditions are commutative, so...

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$$

- Conditions can be joined, so...

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c1 \text{ AND } c2}(R)$$

- Conditions can be disjoint, so...

$$\sigma_{c1 \text{ OR } c2}(R)$$

... is totally valid.

- σ will select all attributes/colums in a relation/table.

PROJECT π

- $\pi_{A_1, A_2, A_6}(R)$ will return only columns/attributes A_1, A_2 and A_6 from R.
- π will remove duplicate tuples, because the result is a set of tuples and sets do not allow duplicates.

RENAME ρ

- $\rho_S(R)$ will rename the relation R to S.
- $\rho_{(B_1, \ldots B_n)}(R)$ will rename the attributes in R to $B_1 \ldots B_n$
- $\rho_{S(B_1, \ldots B_n)}(R)$ will rename the relation R to S <u>and</u> the attributes in R to $B_1 \ldots B_n$.

Intermediate Results

$\pi_{FNAME, LNAME}(\sigma_{AGE \geq 18}(PEOPLE))$ → ADULTS ← $\sigma_{AGE \geq 18}(PEOPLE)$
RESULT ← $\pi_{FNAME, LNAME}(ADULTS)$

UNION $\cup$

- $\cup$ behaves exactly as expected.
- Duplicate tuples are removed.
- To execute $A \cup B$, A and B must be union-compatible
  - Same number of attributes
  - corresponding attributes must be type-compatible.
- The relation resulting from $A \cup B$ has the same set of attribute names as A.

INTERSECTION $\cap$ and DIFFERENCE $-$

- See above.

CART. PRODUCT $\times$ (aka. CROSS PRODUCT in SQL)

- Combines tuples from 2 relations.
- $A(a_1, \ldots a_n) \times B(b_1 \ldots b_n)$ will result in a relation Q with...
  - The attributes $\langle a_1 \ldots a_n, b_1 \ldots b_n \rangle$ in that order
  - A size of $|A| + |B|$
- The operands do <u>not</u> have to be type-compatible.
- Useful Example:

$F_EMPS \leftarrow \sigma_{SEX='F'}(EMPLOYEES)$

$F_EMP_NAMES \leftarrow \pi_{FNAME, LNAME, SSN}(F_EMPS)$

$EMP_DEPENDENTS \leftarrow F_EMP_NAMES \times DEPENDENTS$

$REAL_DEPENDENTS \leftarrow \sigma_{SSN=E_SSN}(EMP_DEPENDENTS)$

$RESULT \leftarrow \pi_{FNAME, LNAME, D_NAME}(REAL_DEPENDENTS)$

RESULT now contains the names of female employees and their dependents.

JOIN ⋈

- A sequence of X followed by σ
- Allows us to combine related tuples from various relations.
- $R \bowtie_{\text{JOIN CONDITION}} S$ joins all records from R and S where the join condition holds.
- Example:

$$\text{RESULT} \leftarrow \text{MANAGERS} \bowtie_{M.ID = E.M_ID} \text{EMPLOYEES}$$

EQUI JOIN

- The same as a regular JOIN (above), but using only equality comparisons.
- The attributes being compared do not need to have the same names.
- In each tuple of the result there will be one or more pairs of attributes with identical values.

NATURAL JOIN *

- This removes the join condition in an EQUIJOIN.
- The standard definition is to join based on all attributes with the same name in both relations
- Hint: this can be well-combined with a RENAME operation.
- The result keeps only one of the same-named attributes.

Side Note:

SELECT, PROJECT, RENAME, UNION, DIFFERENCE and CART. PRODUCT form the complete set of operations, because any relational algebra expression can be formed from them

Outer Joins

- Inner joins lose data by eliminating ~~per~~ tuples without a match.

- Outer joins allow you to keep all of the tuples from the left, right or both relations, regardless of whether or not they are matching tuples.

LEFT OUTER JOIN ⟕

- keeps every tuple from the left relation (ie. R in $R ⟕ S$)

- If no matching tuple is found on the right, the attributes are padded with null values.

RIGHT OUTER JOIN ⟖

- As above, but reversed.

FULL OUTER JOIN ⟗

- keeps all tuples from both relations
  - Matching where possible, padding with null otherwise.

OUTER UNION

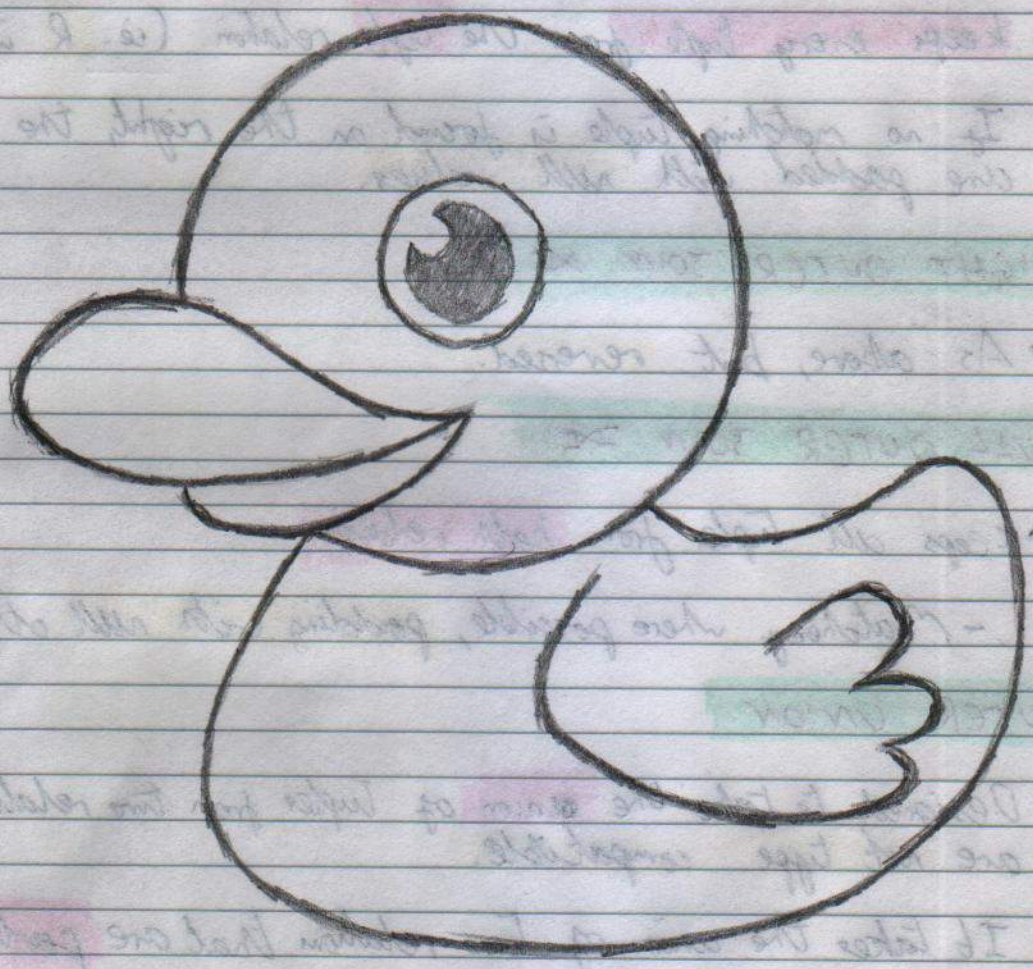
- Designed to take the union of tuples from two relations that are not type compatible.

- It takes the union of two relations that are partially type-compat.

- Type compatible attributes appear only once in the result.

- Example: assuming X, Y and Z are all of different types...

$$R(X,Y) \text{ OUTER UNION } S(X,Z) = T(X,Y,Z).$$



DIVISION ÷

- This is applied to two relations, such that

  $R(Z) \div S(X) = T(Y)$

  where X is a subset of Z and $Y = Z - X$

  $(\&: X \cup Y = Z)$

- For a tuple t to appear in the result T, the values in t must appear in R in combination with <u>every</u> tuple in S.

- Example:

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

$R(A,B) \div S(A) = T(B)$

Aggregate Functions

- Aggregate function cannot be expressed in basic relational algebra.

- We express them with the Functional Operation $\mathcal{F}$

  - $_{\{grouping\}}\mathcal{F}_{\{aggregates\}}$ (relation)

- Examples:

  - $\mathcal{F}_{sum\ value}$ (Stock)
  - $_{DEPT}\mathcal{F}_{COUNT\ StaffID}$ (Staff)
  - $\mathcal{F}_{COUNT\ x,\ AVG\ x}$ (Relation)
  - $_{DEPT}\mathcal{F}_{COUNT\ StaffID,\ AVG\ Salary}$ (Staff).

Semantics of Relation Attributes

- Semantics = how to interpret the values in a relation and how they relate to each other.

- Generally, the easier it is to explain the semantics of a relation, the better the relation schema design.

Guideline 1: Informally, each tuple in a relation should represent one entity or relationship instance.

- Attributes of different entities should not be mixed in the same relation.

- Foreign keys should be used to refer to other entities.

- Entity and relationship attributes should be kept apart as much as possible.

Guideline 2: Design a schema that does not suffer from insert/delete/update anomalies.

- If anomalies are present, note them so that applications can take them into account.

Guideline 3: Design tuples so that they will have as few null values as possible.

- Null values can cause problems with aggregate functions

- Null values waste storage space.

- Attributes that are frequently null can be placed in separate relations with the primary key.

Guideline 4: Design schemas that can be joined by equality conditions on attributes that are PKs or FKs in a way that ensures no spurious tuples are created.

- Avoid relations with matching attributes that are not PK/FK combinations.

Functional Dependencies

- These are used to specify formal measures of how good a relational DB design is.
- Keys define normal forms for relations.
- Constraints are derived from the meaning and interrelationships of data attributes.

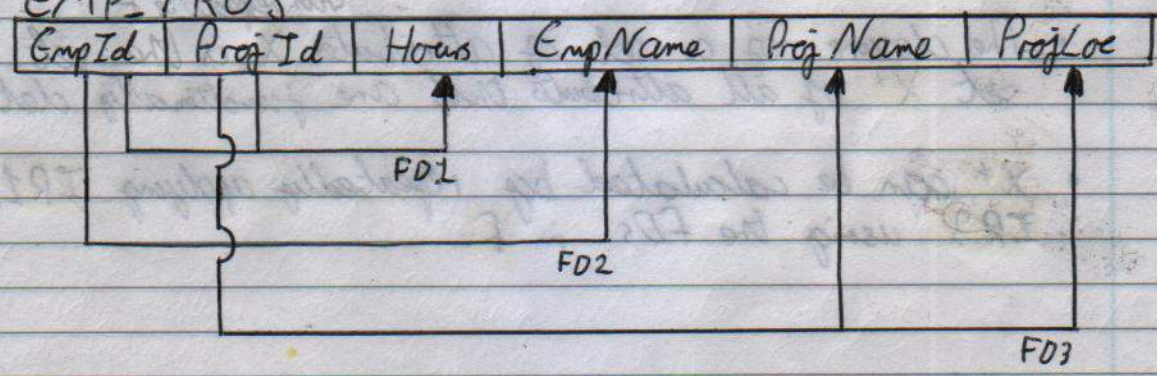
> A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y.

- Functional dependency (FD) between attributes X and Y for relation R:
  - For any two tuples t_1 and t_2 in an instance of R, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$
  - $X \rightarrow Y$ if whenever two tuples have the same values for X they must have the same values for Y.
  - $X \rightarrow Y$ in R specifies a constraint on all instances $r(R)$.
  - Written $X \rightarrow Y$; can be shown on a schema diagram with an arrow.

> If k is a key of R, then k functionally determines all attr in R

Example

EMP_PROJ



FD1: {EmpId, ProjId} → Hours
FD2: EmpId → EmpName
FD3: ProjId → {ProjName, ProjLoc}

Inference Rules for FDs

Reflexive (IR1)

- If Y is a subset of X, then $X \rightarrow Y$

Augmentation (IR2)

- If $X \rightarrow Y$, then $XZ \rightarrow XY$
- NB: $XZ \equiv X \cup Z$.

Transitive (IR3)

- If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

IR1, IR2 and IR3 form a sound and complete set of rules.

Decomposition

- If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Union

- If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Pseudotransitivity

- If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Closures

- The closure of a set F of FDs is the set F^+ formed of all the FDs that can be inferred from F.
- The closure of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X.
- X^+ can be calculated by repeatedly applying IR1, IR2 and IR3 using the FDs in F.

Normalisation

- The process of decomposing unsatisfactory relations by breaking up their attributes into smaller relations.
- This uses PKs and FDs to reduce redundancy and anomalies.
- Refer to superkey / key / minimal superkey / candidate key definitions earlier.
- Prime - an attribute that is a member of some candidate key.

1st Normal Form

- NO composite attributes
- NO multivalue attributes
- NO nested relations

} Considered to be part of the definition of "relation".

- To normalise: remove nested relation and place in a separate relation together with the primary key.

2nd Normal Form

- Every non-prime attribute must be fully functionally dependent on the PK.
  - ie. every non-PK attribute must depend on the entire PK.
- To normalise: decompose and set up a new relation with each partial key and the attributes that are dependant on it. Keep the original relation with the original PK and any attributes that are fully dependent on it.

3rd Normal Form

- In 2NF and...
- No transitive dependency between the PK and a non-prime attribute
  - $X \rightarrow Y$, $Y \rightarrow Z$, with X as PK, is okay if Y is a candidate key!

- To Normalise: decompose and set up a relation that includes the non-key attributes that functionally determine other non-key attributes.

REMINDER:

1NF - the key

2NF - the whole key

3NF - and nothing but the key

Boyce-Codd Normal Form (BCNF):

- If an FD $X \rightarrow A$ exists, then X must be a superkey.
- This is stricter than 3NF, but very similar:
  - Only if $X \rightarrow A$ holds with X not being a superkey and A being prime then we have 3NF but not BCNF.