ACID transactions – Atomic (all or not), Consistent(all same), Isolated(cant change one comment at same time, Durable(lose power, get back data)

- **Relational model**

A relational database consists of only relations (or tables). No other data structure is included. 1. Each row in a table has the same columns of primitive data. 2. No row has a “list” or array of elements in a column. 3. No row has an attribute that other rows don’t have. 4. Types of primitives – Integer, Character Strings (fixed or variable length), Float, Date/Time, Boolean. Does a Relational Model Limit the types of data relationships that can be stored? Not Really – until you enforce constraints like keys.

- **keys**

Defining keys gives a table meaning beyond a collection of attribute values. It answers the “one-row-per-what(key)“ question. 1. A key is an attribute (or combination of attributes) of a table where we do not allow two rows of the table to have the same value of all the attributes of the key (i.e. a key specifies a unique row) – Common Examples: Social Security Number, Employee ID. 2. A single key of a table can be made up of multiple attributes of the table – E.g. 'Last Name', 'First Name' combo might be a key. 3. A single table may have multiple keys – 'CustomerID', 'CustomerEmail' 4. A primary key keep the key cannot be NULL. 5. A table can have one primary key, but multiple keys

- **foreign keys**

A foreign key is an attribute or group of attributes in a relational database table that link data between two tables. 1. It acts as a cross-reference between tables because it references a key of another table 2. A Foreign Key says "this attribute is a key in another table" 3. A foreign key constraint on attribute "a" in Table B indicates that attribute "a" is a key in Table A, and that its value in Table B must exist in Table A. 4. A foreign key constraint on attribute "a" in Table B does not mean "a" has to be a key of Table B. 5.It may or may not also be a key (or part of a key) in the referring table […e.g. Deposit.AcctNo is a foreign key that references Account(base).Number]

- **Schema** —-[table name, column name, type, constraints (约束)]

For every column of every table, the schema specifies allowable values. [e.g. Number must be a 3-digit number; Owner must be a 30-character; string Type must be 'checking' or ‘savings'] The set of allowable values for an column is called the domain of the column. 1. Select the tables, with a name for each table. 2. Select columns for each table and give the domain for each column. 3. Specify the key(s) for each table. 4. Specify all appropriate foreign keys.

- **SELECT … FROM … WHERE …**

Calculate as another definition of SELECT AcctNo, Amount
Using as another definition of FROM ATMWithdrawal
Filter By as another definition of WHERE Amount < 50;

- **JOINS (Cross, Inner, Full Outer, Left Outer, Right Outer)**

SELECT C.Name, S.Name FROM Customer C JOIN Salesperson S ON C.SPNum = S.Number WHERE C.CRating < 6;
= SELECT C.Name, S.Name FROM Customer C, Salesperson S WHERE C.SPNum = S.Number AND C.CRating < 6;
SELECT Course.CNumber, Course.CName, Room FROM Course JOIN Offering USING (CNumber);
= SELECT C.Number, C.Name, O.Room FROM Course C JOIN Offering O ON C.CNumber = O.CNumber;
INNER JOIN (base join) can be zero, and generally can be less than M or N
SELECT C.Name, S.Name FROM Customer C INNER JOIN Salesperson S ON C.SPNum = S.Number;
= SELECT C.Name, S.Name FROM Customer C JOIN Salesperson S ON C.SPNum = S.Number;
CROSS JOIN (all. row = AXB)
SELECT * FROM Customer, Salesperson; SELECT * FROM Customer CROSS JOIN Salesperson;
NATURAL JOIN (JOIN same name column)
SELECT CName, TName FROM Course NATURAL JOIN Offering NATURAL JOIN Teacher; = SELECT CName, TName FROM Course C, Offering O, Teacher T WHERE C.CNumber = O.CNumber AND O.TNumber = T.TNumber;
FULL JOIN
has all the rows of Inner Join, Left Join and Right Join. [Merge tables on a key,keeping everything in both the LEFT and RIGHT tables,even if they have no match (missing columns are set to NULL).]The number of rows that return cannot be less than the max of (M, N).
LEFT JOIN
has all the rows of Inner Join, AND it has all rows in A that don't have a match in B. For the rows in A with no match in B, the value of B's attributes in the result set are NULL. [Merge tables on a key,but keep all records in the LEFT table,even if they don’t have a corresponding match in the RIGHT table(missing columns are set to NULL).] The number of rows that return cannot be less than M.
RIGHT JOIN
has all the rows of Inner Join, AND it has all rows in B that don't have a match in A. For the rows in B with no match in A, the value of A's attributes in the result set are NULL.[Merge tables on a key,but keep everything in the RIGHT table,even if they don’t have a corresponding match in the LEFT table(missing columns are set to NULL).] The number of rows that return cannot be less than N.

- **Aggregates … GROUP BY, HAVING**

Find the biggest salary in city and show it

```
SELECT city, MAX(salary)
FROM Agent                          HAVING 前是find biggest salary each city.
GROUP BY city
HAVING MAX(salary) >=   ALL(
SELECT MAX(salary)
FROM Agent.
GROUP BY city  )
```

- **UNION, INTERSECT, EXCEPT (with and without ALL)**

UNION [A+B] U - OR

INTERSECT [找A,B 中一样的] - AND (PI c-ownerChecking-account) ∩ (PI s-ownerSavings-account)

EXCEPT [A-B 从A减掉所有B] - AND NOT () - ()

- **Subqueries** (returning a single value, returning a table, correlated)

1. Find out which agents make the maximum salary - [SELECT B.agent_id FROM Agent B WHERE B.salary = (SELECT MAX(A.salary) FROM Agent A)]
2. Find which agents make the maximum salary for their countries. [SELECT B.agent_id, B.country FROM Agent B WHERE B.country = 'France' AND B.salary = (SELECT MAX(A.salary) FROM Agent A WHERE A.country = 'France')]
3. Subquery with "IN" – can be equivalent to a join. [SELECT S.Number, S.Name FROM Salesperson S WHERE S.Number IN (SELECT C.SPNum FROM Customer C WHERE C.Name = S.Name);] = [SELECT DISTINCT S.Number, S.Name FROM Salesperson S, Customer C WHERE S.Number = C.SPNum AND C.Name = S.Name;]
4. EXISTS/NOT EXISTS return true&false. SELECT C.Name FROM Customer C WHERE EXISTS (SELECT * FROM Salesperson S WHERE S.Number = C.SPNum AND S.Name = C.Name);

1.You can write subqueries that return multiple columns. SELECT ord_num, agent_code, ord_amount FROM orders WHERE (agent_code, ord_amount) IN (SELECT agent_code, MIN(ord_amount) FROM orders GROUP BY agent_code);
2. You can't write correlated subqueries in the FROM clause – There's no "outer loop" to perform the subquery on one-by-one 3. You can write IN, or other subquery operators for constant lists SELECT agent_name, agent_salary FROM agent WHERE agent_name IN ('Tom', 'Mary', 'Sue')

- **CREATE, update commands**

Create a Heroes table with columns for Character Name, Intelligence, Strength, Combat, and Species. a. With Character Name as the primary key b. Intelligence /Combat limited in value from 0-100 c. Species limited to "Human, Mutant, Android, or Asgardian"

```
CREATE TABLE Hero
(Character_name     character(50) PRIMARY KEY,
Intelligence integer CHECK(Intelligence >=0 AND Intelligence <= 100),
Strength      integer,
Combat        integer CHECK(Combat >=0 AND Combat <= 100),
Species       character(50) CHECK (Species = 'Human' OR Species= 'Mutant' OR Species='Android' OR Species=
'Asgardian')   );
```

Add columns for Real Name and Joined In.

```
ALTER TABLE hero
add Real_name character(50),
add Joined_in character(50);
```

Update the existing rows in the table to add Real Name and Joined In information.

```
UPDATE hero
SET real_name = 'Anthony Edward "Tony"Stark', joined_in ='9/1/1963'
WHERE character_name = 'Iron Man';
```

Insert rows for characters with Combat equal to 60.

```
INSERT INTO hero
VALUES('Black Panther',90,800,70,70,60,'Human','T'Challa','5/1/1968');
```

- **Views** - for security - can't always be updated - BUT, A view can be used for data updates if – It is defined on a single base table – Using only selection and projection – No aggregates – No DISTINCT

Create an SQL view definition for a table. [PowerInfo(power, num_heros, max_intel, max_speed, max_dur)] That lists the number of different superhero having each power and the maximum intelligence, speed, and durability of the characters who have that power.

```
CREATE VIEW PowerInfo AS
SELECT power,
 COUNT(hero) AS num_heros,
 MAX(intelligence) AS max_intel,
 MAX(speed) AS max_speed,
 MAX(durability) AS max_dur
```

FROM powers NATURAL JOIN characterpowers NATURAL JOIN hero
WHERE character_power = power
GROUP BY power

- **Relational Algebra**

NULL can mean - No value makes sense here – We don't have that information – You're not allowed to see that information. Any data type can be NULL. Use IS NULL to find NULL

Owner INTO TEMP temp3
SELECT Name AS Important-Customer, Address
Grouping .Relational Algebra SELECT SPNum, AVG(CBalance), MAX(CBalance) FROM Customer GROUP BY SPNum [|^ SPNum; AVG(CBalance), MAX(CBalance)Customer]

- **ER diagrams, Entities, Attributes, Relationships, Cardinality(1..m)**

Entities: Real-world object distinguishable from other objects. An entity is described using a set of attributes.
Entity Set[长方形] : A collection of similar entities. E.g., all employees …….. Attributes Set[椭圆形]:
Relationship: Association among 2 or more entities. E.g., Attishoo's home department is Pharmacy.
Relationship Set[菱形]: Collection of similar relationships. E.g., Home (often referred to as just relationship)
Weak entity[两长方形套][两菱形套] no 'strong' no 'weak' (based on strong)

一 ER diagram to and from Relational Schema

- Create table and choose key for each entity set; include singlevalued attributes.
- Create table for each weak entity set; include single-valued attributes. Include key of owner as a foreign key in the weak entity. Set key as foreign key of owner plus local, partial key.
- For each 1:1 relationship set, add a foreign key to one of the entity sets involved in the relationship (a foreign key to the other entity in the relationship)*.
- For each 1:N relationship set, add a foreign key to the entity set on the N-side of the relationship (to reference the entity set on the 1-side of the relationship)*
- For each M:N relationship set, create a new table. Include a foreign key for each participant entity set, in the relationship set. The key for the new table is the set of all such foreign keys.
- For each multi-valued attribute, construct a separate table. Repeat the key for the entity in this new table. It will serve as a foreign key to the original table for the entity. The new key of the table will be the key of the entity plus the attribute.
- A weak entity set and its identifying relationship set are translated into a single table. Must include key of strong entity set, as a foreign key. • When the owner entity is deleted, all owned weak entities must also be deleted.
  在 CREATE 加 FOREIGN KEY (ssn) REFERENCES Employee, ON DELETE CASCADE)

- **FDs, Normalization (why and how)**

1.Expanding upon the idea of keys of a table, look for where some attributes depend on other attributes within a relation (entity or relation) 2. We'll call these Function Dependencies (FDs) 3. If your DB design doesn't enforce these functional dependencies by design, you can end up with data anomalies (data entries that don't meet the FDs), and redundant data 4. Database normalization is a systematic approach to decomposing tables to eliminate functional dependencies and data redundancy
Non-key dependencies. Emp(ssn, name, phone, dnum, dept-name). Which drum ->dept-name. We shouldn't allow two different names for same dnum in two row.
A superkey is a set of attributes from a relation that contains a key
[X-> Y] X determines Y || Reflexivity: If Y is a set of attrs, Y subset of X, then X → Y || Augmentation: If X → Y, and Z is a set of attrs, then XZ → YZ || Transitivity: If X→ Y, Y → Z, then X → Z
If F+ = G+ then the decomposition is dependency preserving
1. B → A [B,A]
2. F→F,G [F,G]
3. C,D → A,B,E,F [C,D,E,F]
4. E,F → A,B [E,F,B]
5. G→G,F

* If like SNu,PC,SuID → SA,SC,SS,Z,DID,DNa,PD,A,Pr,U,SuN,SuR,C Inventory = [SNu,SuID,PC]

Subquery returns	Operator	Meaning
A single value (1 row/col)	=, <>, >, <, LIKE …	*(same as scalar comparison)*
Multiple Rows	= SOME (ANY)	*the expression must be true for at least one row*
Multiple Rows	= ALL	*the expression must be true for all rows*
Multiple Rows	<> SOME	*the expression must be false for at least one row*
Multiple Rows	<> ALL	*the expression must be false for all rows*
Multiple Rows	IN	*the tuple matches at least one row (aka =SOME)*
Multiple Rows	NOT IN	*the tuple does not match any row (aka <> ALL)*
Multiple Rows	EXISTS	*subquery answer is not empty*
Multiple Rows	NOT EXISTS	*subquery answer is empty*
Multiple Rows	UNIQUE	*subquery answer has no duplicates*
Multiple Rows	NOT UNIQUE	*subquery answer has duplicates*

Relational Operator	Sym	SQL	Example
Selection (yes, confusing use of selection operator vs SELECT in SQL)	σ	WHERE	$\sigma_{Balance < 3000}$ Account SELECT * FROM Account WHERE Balance < 1000
Projection	π ($\underline{\pi}$)	SELECT DISTINCT (underlined is SELECT, with duplicated)	$\pi_{Number, Owner}$Account SELECT DISTINCT Number, Owner FROM Account
Cartesian (Cross) Product	×	FROM A, B	A × B SELECT * FROM A, B
Rename	ρ	AS, and *correlation names*	$\rho_{NewTable(B1, B2, B3)}$ Table SELECT A1 as B1, A2 as B2, A3 as B3 FROM Table NewTable
Join	⋈		See later in the foils
Union	∪	UNION	
Intersection	∩	INTERSECT	
Difference	—	MINUS	

Example: Group by with Having

Intermediate result: 4 groups

~~103~~	~~Quan~~	…	
101	Mary	…	5
106	Susan	…	5
107	David	…	5
102	John	…	8
110	Ying	…	8
~~109~~	~~Mike~~	…	2

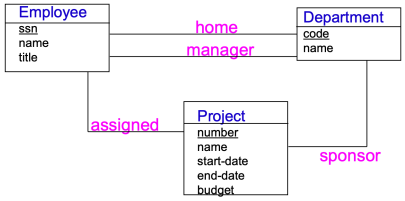
Query answer: one row per group that satisfies the HAVING clause!

SPNum	COUNT(*)
5	3
8	2

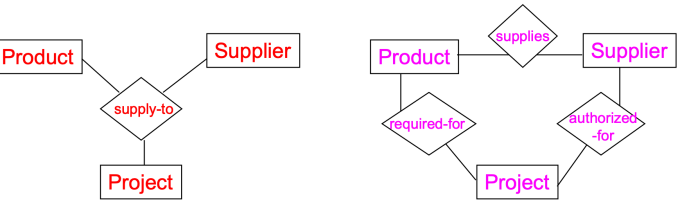
```
SELECT    SPNum, COUNT(*)
FROM      Customer
GROUP BY  SPNum
HAVING    Count(*) > 1;
```

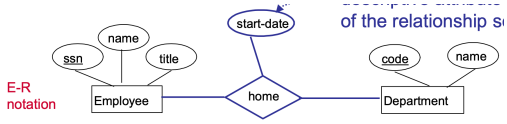
```
# creation
CREATE VIEW name AS
SELECT col1, col2
FROM table
WHERE condition
;

# updating/overwriting
CREATE OR REPLACE VIEW name AS
SELECT col1, col2
FROM table
WHERE condition
;

# deletion
DROP VIEW name;
```

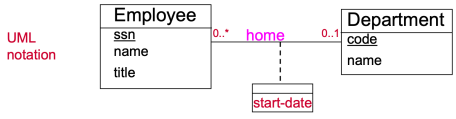
Employee (ssn, name, title, home-dept)
Project-team(ssn, number)
Department (id, name, manager)
Project (number, name, start-date, end-date, budget, sponsor)



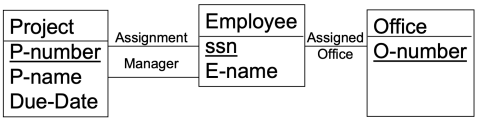
Ternary vs. Binary Relationship Sets (Ternary = 3-way, Binary = 2-way)







Entity vs. Value of an Attribute



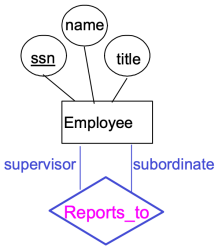
What are some reasons to model Office as an entity set?

An employee can have more than one office

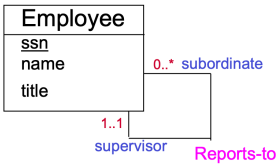
There are other attributes of Office

Office needs to participate in other relationship sets such as a relationship set connecting to telephones jacks or network drops

E-R notation



UML notation



Roles are good to add when you have a self-to-self relationship (so you can add