SQL

Slides adapted from http://infolab.stanford.edu/~ullman/fcdb.html

Why SQL?

- SQL is a very-high-level language.
  - Say "what to do" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages like C/C++ or Java.
- Database management system figures out "best" way to execute query.
  - Called "query optimization"

Database Schemas in SQL

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

3

Example: Create Table

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     VARCHAR(20),
    price    REAL,
    PRIMARY KEY (bar, beer)
);

DROP TABLE Sells;
```

4

Simple SQL Queries

5

Select-From-Where Statements

SELECT desired attributes
FROM one or more tables
WHERE condition about target tuples of the tables

6

Example

- Using Beers(name, manf), what beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

7

Result of Query

name
Bud
Bud Lite
. . .

The answer is a relation with a single attribute, name, and tuples with the name of each beer Made by Anheuser-Busch, such as Bud.

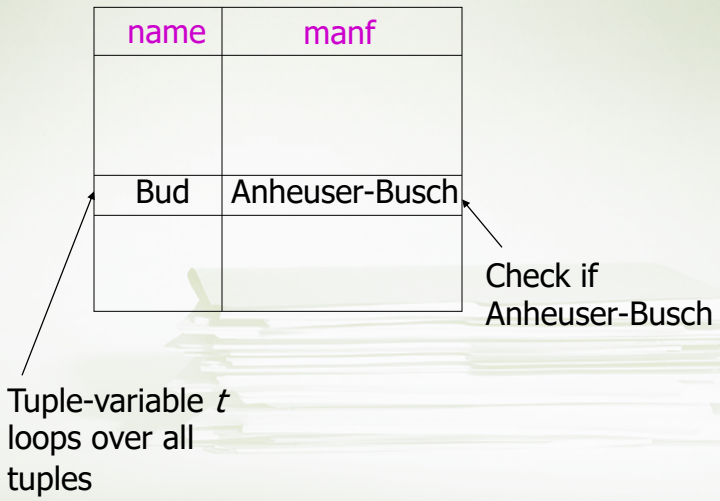
8

Meaning of Single-Relation Query

- Begin with the relation in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Apply the extended projection indicated by the SELECT clause.

9

Operational Semantics



10

Operational Semantics --- General

- Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM.
- Check if the "current" tuple satisfies the WHERE clause.
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

11

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."
- Example: Using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

12

Result of Query:

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
. . .	. . .

Now, the result has each of the attributes of Beers.

13

Renaming Attributes

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example: Using Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

14

Result of Query:

beer	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
. . .	. . .

15

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: Using Sells(bar, beer, price):

```
SELECT bar, beer,
       price*114 AS priceInYen
FROM Sells;
```

16

Result of Query

bar	beer	priceInYen
Joe's	Bud	285
Sue's	Miller	342
…	…	…

17

Example: Constants as Expressions

- Using Likes(drinker, beer):

```
SELECT drinker,
       'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```

18

Result of Query

drinker	whoLikesBud
Sally	likes Bud
Fred	likes Bud
…	…

19

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.
  - And many other operators that produce boolean-valued results.

20

Example: Complex Condition

- Using Sells(bar, beer, price), find the price Joe's Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

21

Patterns

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string with % = "any string", _ = "any character."

22

Example: LIKE

- Using Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

23

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

24

Comparing NULL's to Values

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
  - Example: *5 < null or null = null*
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).

25

Null Values and Three Valued Logic

- Three-valued logic using the truth value *unknown*:
  - AND: *(true* **and** *unknown) = unknown*
    (false **and** *unknown) = false*
    (unknown **and** *unknown) = unknown*
  - OR: (*unknown* **or** *true*) = *true*
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown) = unknown*
  - NOT: *(***not** *unknown) = unknown*

Multirelation Queries

- Interesting queries often combine data from more than one relation.
- We can address several relations in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name by "<relation>.<attribute>" .

27

Example: Joining Two Relations

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
   Frequents.drinker =
   Likes.drinker;
```

28

Formal Semantics

- Almost the same as for single-relation queries:
  1. Start with the product of all the relations in the FROM clause.
  2. Apply the selection condition from the WHERE clause.
  3. Project onto the list of attributes and expressions in the SELECT clause.

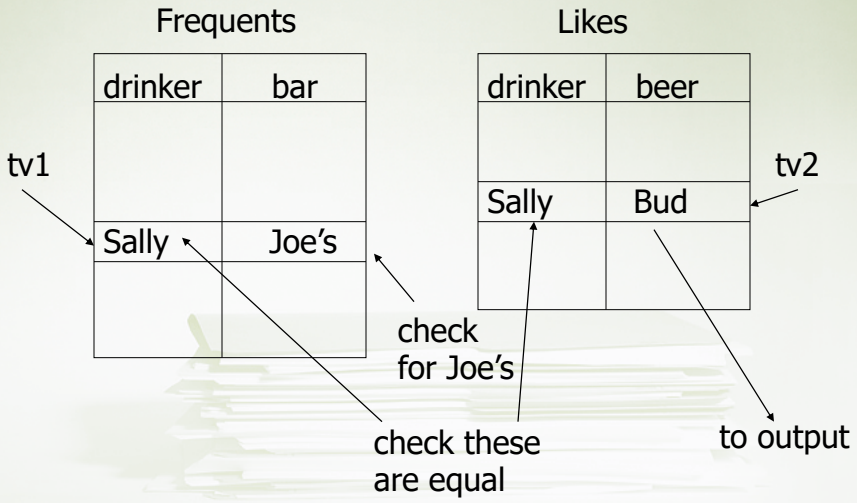
29

Operational Semantics

- Imagine one tuple-variable for each relation in the FROM clause.
  - These tuple-variables visit each combination of tuples, one from each relation.
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.

30

Example



31

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation.
- Distinguish copies by following the relation name with the name of a tuple-variable, in the FROM clause.
- It's always an option to rename relations this way.

32

Example: Self-Join

- From Beers(name, manf), find all pairs of beers by the same manufacturer.
  - Do not produce pairs like (Bud, Bud).
  - Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

33

Subqueries

34

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places.

35

Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - A run-time error occurs if there is no tuple or more than one tuple.

36

Example: Single-Tuple Subquery

- Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two steps would surely work:
  1. Find the price Joe charges for Bud.
  2. Find the bars that serve Miller at that price.

37

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE bar = 'Joe''s Bar'
                AND beer = 'Bud');
```

The price at which Joe sells Bud

38

The IN Operator

- <tuple> IN (<subquery>) is true if and only if the tuple is a member of the relation produced by the subquery.
  - Opposite: <tuple> NOT IN (<subquery>).
- IN-expressions can appear in WHERE clauses.

39

Example: IN

- Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

The set of beers Fred likes

40

Example

Are these two queries equivelent?

```
SELECT a
FROM R, S
WHERE R.b = S.b;

SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

41

This Query Pairs Tuples from R, S

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over the tuples of R and S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition; 1 gets output twice.

42

IN is a Predicate About R's Tuples

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies the condition; 1 gets output once.

43

The Exists Operator

- EXISTS (<subquery>) is true if and only if the subquery result is not empty.
- Example: From Beers(name, manf), find those beers that are only unique beer made by their manufacturer.

44

Example: EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS (
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

45

The Operator ANY

- ***x*** = ANY (<subquery>) is a boolean condition that is true iff ***x*** equals at least one tuple in the subquery result.
  - = could be any comparison operator.
- Example: ***x*** >= ANY (<subquery>) means ***x*** is not the smallest tuple produced by the subquery.
  - Note tuples must have one component only.

46

Definition of ANY Clause

- F <comp> **any (some)** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
  Where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **any** | 0 / 5 / 6) = true (read: 5 < any tuple in the relation)

(5 < **any** | 0 / 5) = false

(5 = **any** | 0 / 5) = true

(5 ≠ **any** | 0 / 5) = true (since 0 ≠ 5)

(= **any**) ≡ **in**
However, (≠ **any**) ≢ **not in**

Example: ANY

- From Sells(bar, beer, price), find the beer(s) sold at Joe's bar whose price is higher than some (at least one) beer sold at Sue's bar.

```
SELECT beer
FROM Sells
WHERE bar='Joe''s'
  and price >= ANY (SELECT price
                    FROM Sells
                    WHERE bar='Sue''s');
```

price from the outer Sells must be greater than or equal to at least one price from the inner Sells.

48

The Operator ALL

- *x* <> ALL (<subquery>) is true iff for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not in the subquery result.
- <> can be any comparison operator.
- Example: *x* >= ALL (<subquery>) means there is no tuple larger than *x* in the subquery result.

49

Definition of ALL Clause

- F <comp> **all** *r* $\Leftrightarrow$ $\forall$ *t* $\in$ *r* (F <comp> *t)*

(5 < **all** $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}$) = false

(5 < **all** $\begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}$) = true

(5 = **all** $\begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}$) = false

(5 ≠ **all** $\begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}$) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**
However, ¬(= **all**) ≠ **not in**

Example: ALL

- From Sells(bar, beer, price), find the beer(s) sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must be greater than or equal to every price from the inner Sells.

51

Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (<subquery>) UNION (<subquery>)
  - (<subquery>) INTERSECT (<subquery>)
  - (<subquery>) EXCEPT (<subquery>) (some DBMS uses MINUS)

52

Example: Intersection

- Using Likes(drinker, beer), Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:
  1. The drinker likes the beer, and
  2. The drinker frequents at least one bar that sells the beer.

53

Solution

Notice trick: subquery is really a stored table.

The drinker frequents a bar that sells the beer.

```
(SELECT * FROM Likes)
    INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar);
```

54

Bags vs Sets

- Bags allow duplicates
- When doing projection, it is faster to avoid eliminating duplicates.
  - Just work one-tuple-at-a-time.
- For set intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.

55

Controlling Duplicate Elimination

- Force the result to be a set by SELECT DISTINCT …
- Force the result to be a bag (i.e., don't eliminate duplicates) by ALL, as in UNION ALL …

56

Example: DISTINCT

- From Sells(bar, beer, price), find all the different prices charged for beers:

```
SELECT DISTINCT price
FROM Sells;
```

- Notice that without DISTINCT, each price would be listed as many times as there were bar/beer pairs at that price.

57

Example: ALL

- Using relations Frequents(drinker, bar) and Likes(drinker, beer):

```
(SELECT drinker FROM Frequents)
    UNION ALL
(SELECT drinker FROM Likes);
```

- Lists drinkers who frequent bars or like beers, where a drinkers name may appear many times in the result.

58

Aggregates

59

Aggregations

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- Also, COUNT(*) counts the number of tuples.

60

Example: Aggregation

- From Sells(bar, beer, price), find the average price of Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

61

Eliminating Duplicates in an Aggregation

- Use DISTINCT inside an aggregation.
- Example: find the number of *different* prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

62

NULL's Ignored in Aggregation

- NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
- But if there are no non-NULL values in a column, then the result of the aggregation is NULL.
  - Exception: COUNT of an empty set is 0.

63

Example: Effect of NULL's

```
SELECT count(*)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud.

```
SELECT count(price)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud at a known price.

64

Grouping

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of GROUP BY attributes, and any aggregation is applied only within each group.

65

Example: Grouping

- From Sells(bar, beer, price), find the average price for each beer:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

beer	AVG(price)
Bud	2.33
…	…

66

Example: Grouping

- From Sells(bar, beer, price) and Frequents(drinker, bar), find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute all drinker-price triples for Bud.

Then group them by drinker.

67

Restriction on SELECT Lists With Aggregation

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or
  2. An attribute in the GROUP BY list.

68

Illegal Query Example

- You might think you could find the bar that sells Bud the cheapest by:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

- But this query is illegal in SQL – only attributes in GROUP BY clause and aggregates can appear in SELECT clause.

69

Illegal Query Example (cont.)

- Correct answer:

```
SELECT bar, price
FROM Sells
WHERE beer='Bud'
  AND price = (SELECT min(price)
               FROM Sells
               WHERE beer='Bud')
```

70

Illegal Query Example (cont.)

Correct answer 2:

```
SELECT bar, price
FROM Sells
WHERE beer='Bud'
  AND price <= ALL(SELECT price
                   FROM Sells
                   WHERE beer='Bud')
```

71

HAVING Clauses

- HAVING <condition> may follow a GROUP BY clause.
- If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

72

Example: HAVING

- From Sells(bar, beer, price) and Beers(name, manf), find the average price of those beers that are served in at least three bars.

Solution

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3;
```

Beer groups with at least 3 non-NULL bars

Requirements on HAVING Conditions

- Anything goes in a subquery.
- Outside subqueries, they may refer to attributes only if they are either:
  1. A grouping attribute, or
  2. Aggregation

  (same condition as for SELECT clauses with aggregation).

75

Database Modifications

76

Database Modifications

- A *modification* command does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications:
  1. *Insert* a tuple or tuples.
  2. *Delete* a tuple or tuples.
  3. *Update* the value(s) of an existing tuple or tuples.

77

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);
- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

```
INSERT INTO Likes
  VALUES('Sally','Bud');
```

78

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- Two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

79

Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
  VALUES('Bud', 'Sally');
```

80

Adding Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- When an inserted tuple has no value for that attribute, the default will be used.

81

Example: Default Values

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
        DEFAULT '123 Sesame St.',
   phone CHAR(16)
);
```

82

Example: Default Values

```
INSERT INTO Drinkers(name)
    VALUES('Sally');
```

Resulting tuple:

name	address	phone
Sally	123 Sesame St	NULL

83

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>

  (<subquery>);

84

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally's "potential buddies," i.e., those drinkers who frequent at least one bar that Sally also frequents.

Solution

The other drinker

Pairs of Drinker tuples where the first is for Sally, the second is for someone else, and the bars are the same.

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
   d2.drinker <> 'Sally' AND
   d1.bar = d2.bar
);
```

Deletion

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>

  WHERE <condition>;

87

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
      beer = 'Bud';
```

88

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

89

Example: Delete using Subquery

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b1
WHERE EXISTS (
    SELECT b2.name
    FROM Beers b2
    WHERE b1.manf = b2.manf AND
        b1.name <> b2.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Be careful with deletions!

90

Semantics of Deletion --- (1)

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, when *b* is the tuple for Bud Lite, do we delete that tuple too?

91

Semantics of Deletion --- (2)

- Answer: we ***do*** delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied.
  2. Delete the marked tuples.

92

Updates

- To change certain attributes in certain tuples of a relation:

  UPDATE <relation>

  SET <list of attribute assignments>

  WHERE <condition on tuples>;

93

Example: Update

- Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

94

Example: Update Several Tuples

- Make $4 the maximum price for beer:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```

95

Order of Update

- These two sets of queries give different result:

```
UPDATE Sells
SET price = price*110%
WHERE price < 4.00;
UPDATE Sells
SET price = price*105%
WHERE price >= 4.00;
```

```
UPDATE Sells
SET price = price*105%
WHERE price >= 4.00;
UPDATE Sells
SET price = price*110%
WHERE price < 4.00;
```

96

Use Case

```
UPDATE Sells
SET price = CASE
                WHEN price < 4.00
                     THEN price = price*110%
                ELSE
                     price = price*105%
            END
```

97