Relational Query Languages:
SQL

Juliana Freire
Why SQL?

• SQL is a 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++ or Java

• Database management system figures out “best” way to execute query
  – Called “query optimization”
What is SQL?

- **Data manipulation**: ad-hoc queries and updates
  
  ```sql
  SELECT * 
  FROM Account 
  WHERE Type = "checking ";
  ```

- **Data definition**: creation of tables and views
  
  ```sql
  CREATE TABLE Account 
  (Number integer NOT NULL, 
   Owner character, 
   Balance currency, 
   Type character, 
   PRIMARY KEY (Number));
  ```

- **Control**: assertions to protect data integrity
  
  ```sql
  CHECK (Owner IS NOT NULL)
  ```
Relational Algebra vs. SQL

- Relational algebra = query only
- SQL = data manipulation + data definition + control
- SQL data manipulation is similar to, but not exactly the same as relational algebra
  - SQL is based on set and relational operations with certain modifications and enhancements
  - We will study the differences
SQL: History and Trivia

• Conceived in the mid-70s
• IBM developed SEQUEL (Structured English Query Language) as part of System R project
• Oracle beat IBM to the market…
• First standard in 1986; enhanced in 1989; significantly revised in 1992 (SQL-92 = SQL2)
• Many revisions: SQL-99 = SQL3; SQL2003, …
• Correctly pronounced “es cue ell”, not “sequel”! (Don Chamberlin)
## Database Schema for Running Example

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>Number</th>
<th>CustId</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPOSIT</td>
<td>Account</td>
<td>TransactionID</td>
<td>Date</td>
<td>Amount</td>
</tr>
<tr>
<td>CHECK</td>
<td>Account</td>
<td>Check-number</td>
<td>Date</td>
<td>Amount</td>
</tr>
<tr>
<td>ATMWITHDRAWAL</td>
<td>TransactionID</td>
<td>CustID</td>
<td>AcctNo</td>
<td>Amount</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>ID</td>
<td>Name</td>
<td>Phone</td>
<td>Address</td>
</tr>
</tbody>
</table>
### SQL in Action: Find tuples that satisfy a condition

**ATMWithdrawal table**

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

```sql
SELECT * FROM ATMWithdrawal WHERE Amount < 50;
```

**Attributes of the resulting relation**

**Relation to which the query refers**

**Condition that must be satisfied**
**ATMWithdrawal table**

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

Is the amount field of this row less than $50? **YES!**

**Query Answer table**

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
</tbody>
</table>

The WHERE clause is evaluated for each row in the table.
### ATMWithdrawal table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

Is the amount field of this record less than $50?  **NO!**

Amount < 50

Ignore this record!

### Query Answer table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
</tbody>
</table>
ATM Withdrawal table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

Is the amount field of this record less than $50? **YES!**

Query Answer table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
</tbody>
</table>
ATM Withdrawal table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

Is the amount field of this record less than $50? **YES!**

Query Answer table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
</tbody>
</table>
### ATM Withdrawal table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>102</td>
<td>$150.00</td>
<td>11/10/2000 13:15:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>100</td>
<td>$200.00</td>
<td>11/8/2000 14:14:00</td>
</tr>
</tbody>
</table>

Is the amount field of this record less than $50? **NO!**

Ignore this record!

### Query Answer table

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>WithdrawDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
</tbody>
</table>
Selection in SQL and Relational Algebra

SELECT *
FROM ATMWithdrawal
WHERE Amount < 50;

→

σ Amount < 50 ATMWithdrawal
Conditions in the WHERE Clause

• Conditions evaluate to a Boolean value: TRUE or FALSE (and also UNKNOWN…)

• Expressions built with comparison operators: =, <>, <, >, <=, and >=
  – E.g., Amount = 50; Amount <> 50

• Values to be compared can be
  – Attributes of relations in FROM clause
  – Constants
  – Arithmetic expressions, e.g., Amount < Credit - Balance

• Expressions composed with logical connectives: and, or, not
  – E.g., Amount < 50 and CustID <> 1

SELECT *
FROM ATMWithdrawal
WHERE Amount < 50;
String Operations

• Strings are enclosed within single quotes
  SELECT * FROM Customer
  WHERE name = ‘Juliana Freire’

• Pattern matching with LIKE operator
  SELECT * FROM Customer
  WHERE name LIKE ‘%Fr__re’
  – matches ‘Juliana Freire’, ‘Freire’, ‘Friere’

• Other operations:
  – String concatenation: name = ‘Juliana’ || ‘Freire’
  – String conversion: upper(name); lower(name)
Dates and Times

• Special data types for dates and times
• Date constant represented by keyword DATE followed by a quoted string
  – E.g., DATE ‘1972-03-05’
  – SELECT * FROM Students
    WHERE birth_date < DATE ‘1972-03-05’
• Time constant represented by keyword TIME followed by a quoted string
  – E.g., TIME ’11:30:02.5’ – all of you will be gone by then ;-)
Null Values

• A null value may have different meanings:
  – *Value unknown*: there is a value that belongs here, but we don’t know which, e.g., Juliana’s birthday
  – *Value inapplicable*: no value makes sense here, e.g., spouse name for a single employee
  – *Value withheld*: we are not entitled to know the value that belongs here, e.g., an unlisted phone number

• NULL is not a constant: it cannot be used explicitly as an operand in an expression
  – NULL+3 is not a legal SQL statement

• Arithmetic expressions involving NULLs return NULL
• If x is NULL, x+3 is NULL
Null Values (cont.)

- If you would like to check whether a value is or isn’t null you need to use a special expression – IS NULL, IS NOT NULL
  - SELECT name, GPA  FROM Students WHERE Students.spouse IS NULL
    List name and GPA of students who are single

  SELECT name, GPA  FROM Students WHERE Students.spouse IS NOT NULL
  List name and GPA of students who are married
Comparisons and Null Values

- Conditions evaluate to a Boolean value: TRUE or FALSE, and UNKNOWN
- Comparisons involving nulls result in UNKNOWN
  - E.g., if $x = \text{NULL}$, the condition $x > 3$ evaluates to UNKNOWN
- **Trick:** TRUE = 1; FALSE = 0; UNKNOWN = 1/2
  - $X \text{ and } Y = \min(X,Y)$
  - $X \text{ or } Y = \max(X,Y)$
  - not $X = 1 - X$
- Tuples for which the condition evaluates to UNKNOWN are not included in the result
Challenge Question

If all withdrawals have Amount greater than or equal to zero, is it the case that the query

```sql
SELECT * 
FROM   ATMWithdrawal 
WHERE  Amount >= 0; 
```

Always return a copy of the ATMWithdrawal table?

????
Another Surprising Example

• From the following Bookstore relation:

<table>
<thead>
<tr>
<th>name</th>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe’s Bar</td>
<td>HTMLP...</td>
<td>NULL</td>
</tr>
</tbody>
</table>

```
SELECT name
FROM Bookstore
WHERE price < 2.00 OR price >= 2.00;
```

UNKNOWN

UNKNOWN

UNKNOWN
Projection in SQL

SELECT AcctNo, Amount
FROM   ATMWithdrawal
WHERE  Amount < 50;

• Result will be projected onto attributes listed in SELECT clause

In Relational Algebra:
\[ \pi_{\text{AccNo}, \text{Amount}} (\sigma_{\text{Amount} < 50} \text{ATMWithdrawal}) \]
Query Answer table  (Amount < 50)

<table>
<thead>
<tr>
<th>TransactionID</th>
<th>CustId</th>
<th>AcctNo</th>
<th>Amount</th>
<th>Withdraw Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>102</td>
<td>$25.00</td>
<td>11/1/2000 9:45:00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>$40.00</td>
<td>11/1/2000 10:05:00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>100</td>
<td>$40.00</td>
<td>11/1/2000 10:07:00</td>
</tr>
</tbody>
</table>

```
SELECT AcctNo, Amount
FROM    ATMWithdrawal
WHERE  Amount < 50;
```

Consider the attributes listed in the SELECT clause.

Throw away attributes that are not listed.

Thus the final query answer is:

**Final Query Answer table**

<table>
<thead>
<tr>
<th>AcctNo</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>$25.00</td>
</tr>
<tr>
<td>101</td>
<td>$40.00</td>
</tr>
<tr>
<td>100</td>
<td>$40.00</td>
</tr>
</tbody>
</table>
More on Projection in SQL

```sql
SELECT AcctNo AS Number, Amount AS Amt
FROM ATMWithdrawal
WHERE Amount < 50;
```

- Result will be the same, but with different column headers

<table>
<thead>
<tr>
<th>Number</th>
<th>Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>$25.00</td>
</tr>
<tr>
<td>101</td>
<td>$40.00</td>
</tr>
<tr>
<td>100</td>
<td>$40.00</td>
</tr>
</tbody>
</table>

```sql
SELECT AcctNo AS Number, Amount*10 AS Amt
FROM ATMWithdrawal
WHERE Amount < 50;
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>$250.00</td>
</tr>
<tr>
<td>101</td>
<td>$400.00</td>
</tr>
<tr>
<td>100</td>
<td>$400.00</td>
</tr>
</tbody>
</table>
More on Projection in SQL

```sql
SELECT LoanNo AS AccNumber, 250 AS Amt
FROM Loan
WHERE LoanAmount > 600000;
```

<table>
<thead>
<tr>
<th>AccNumber</th>
<th>Amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>$250.00</td>
</tr>
<tr>
<td>101</td>
<td>$250.00</td>
</tr>
<tr>
<td>100</td>
<td>$250.00</td>
</tr>
</tbody>
</table>
## SQL vs Relational Algebra

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

```sql
SELECT Owner, Balance
FROM Account
```

<table>
<thead>
<tr>
<th>Owner</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Smith</td>
<td>1000.00</td>
</tr>
<tr>
<td>W. Wei</td>
<td>2000.00</td>
</tr>
<tr>
<td>J. Smith</td>
<td>1000.00</td>
</tr>
<tr>
<td>M. Jones</td>
<td>1000.00</td>
</tr>
<tr>
<td>H. Martin</td>
<td>10000.00</td>
</tr>
</tbody>
</table>
## SQL vs Relational Algebra

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>savings</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

**SELECT** Owner, Balance
**FROM** Account

Query results can be a *bag*

<table>
<thead>
<tr>
<th>Owner</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Smith</td>
<td>1000.00</td>
</tr>
<tr>
<td>W. Wei</td>
<td>2000.00</td>
</tr>
<tr>
<td>J. Smith</td>
<td>1000.00</td>
</tr>
<tr>
<td>M. Jones</td>
<td>1000.00</td>
</tr>
<tr>
<td>H. Martin</td>
<td>10000.00</td>
</tr>
</tbody>
</table>
Why Bags?

• Sets are simple and natural – but they can be inefficient to manipulate
  – Removing duplicates is expensive, possibly more than executing the query!

• There are situations where desired answer can only be obtained if bags are used
  – E.g., ((John, 27), (Mary, 20), (Ann, 20))
  – What is the average age of customers? $\text{Avg}\{27,20\}$ or $\text{Avg}\{27,20,20\}$?
How is an SQL query evaluated?

Third, the SELECT clause tells us which attributes to keep in the query answer.

```
SELECT AcctNo, Amount
FROM ATMWithdrawal
WHERE Amount < 50;
```

First, the FROM clause tells us the input tables.

Second, the WHERE clause is evaluated for all possible combinations from the input tables.
SQL query using two tables

Account(Number, CustID,Balance,Type)
Deposit(Acc_num,TID,Date,Amount)

```
SELECT Number, Balance
FROM Account, Deposit
WHERE Acc_num = Number and Amount > 10000;
```

How does this work?
Which rows, from which tables, are evaluated in the WHERE clause?
What about this one:

```
SELECT *
FROM Account, Deposit;
```
The Basic Structure of a Query

• A typical SQL query has the form:
  
  ```sql
  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P
  ```

  – $A_i$s represent attributes, $r_i$s represent relations, 
  $P$ is a predicate.

• This query is equivalent to the relational algebra expression.

  $$\Pi_{A_1, A_2, ..., A_n}(\sigma_P (r_1 \times r_2 \times ... \times r_m))$$

• The result of an SQL query is a relation
  – But not necessarily a set!
SQL query using two tables

Account(Number, CustID,Balance,Type)  
Deposit(Acc_num,TID,Date,Amount)

What are the relational algebra expressions for:

\[ \pi \text{Number, Balance} \left( \sigma \text{Acc_num = Number and Amount > 10000} (\text{Account X Deposit}) \right) \]

\[ \text{SELECT * FROM Account, Deposit;} \]

(Account X Deposit)
SQL query using two tables

Account(\texttt{Number}, \texttt{CustID}, \texttt{Balance}, \texttt{Type})
Deposit(\texttt{Number}, \texttt{TID}, \texttt{Date}, \texttt{Amount})

\begin{center}
\begin{tabular}{|l|l|}
\hline
SELECT & A.\texttt{Number}, A.\texttt{Balance} \\
FROM & Account A, Deposit D \\
WHERE & \texttt{D.Number} = \texttt{A.Number} \text{ and } \texttt{D.Amount} > 10000; \\
\hline
\end{tabular}
\end{center}

Notice that

“\textit{A}” is a correlation name for \texttt{Account}
and

“\textit{D}” is a correlation name for \texttt{Deposit}.

\textit{Correlation name} = \textit{tuple variable}

\begin{itemize}
\item You choose correlation names when you write the query.
\item Useful for disambiguating attribute names, e.g.,
\end{itemize}
Account vs. Deposit number
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

```sql
SELECT A.Owner, A.Balance
FROM Account A, Deposit D
WHERE D.Account = A.Number and A.Balance > 1000;
```

We must check every combination of one row from Account with one row from Deposit!
No! Throw it away.

WHERE D.Account = A.Number and A.Balance > 1000;
## Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

## Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

No! Throw it away.
<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

No! Throw it away.

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Account</td>
<td>Number</td>
<td>Owner</td>
<td>Balance</td>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;
<table>
<thead>
<tr>
<th>Account Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

**Deposit**

<table>
<thead>
<tr>
<th>Deposit Account T-id</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

**Yes! Place in query answer.**
WHERE D.Account = A.Number and A.Balance > 1000;
<table>
<thead>
<tr>
<th>Account</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
<th>Account</th>
<th>T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
</tbody>
</table>

No! Throw it away.
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

`WHERE D.Account = A.Number and A.Balance > 1000;`

---

No! Throw it away.
WHERE D.Account = A.Number and A.Balance > 1000;
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

No! Throw it away.
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;
Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

Deposit

<table>
<thead>
<tr>
<th>Account</th>
<th>T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

No! Throw it away. Why?
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

**WHERE**

\[ D.\text{Account} = A.\text{Number} \text{ and } A.\text{Balance} > 1000; \]

### WHERE

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
<th>Account</th>
<th>T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
</tbody>
</table>

No! Throw it away.
No! The first three fail.

WHERE D.Account = A.Number and A.Balance > 1000;
### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

WHERE D.Account = A.Number and A.Balance > 1000;

Yes! Place in query answer.
Intermediate result (after processing the FROM & WHERE clauses)

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
<th>Account T-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>1 10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td>102</td>
<td>2 10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td>105</td>
<td>4 11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

Process the SELECT

```
SELECT A.Owner, A.Balance
FROM Account A, Deposit D
WHERE D.Account = A.Number and A.Balance > 1000;
```

Final query answer:
(notice that W. Wei appears twice: Result relation is a bag)

<table>
<thead>
<tr>
<th>Owner</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>W. Wei</td>
<td>2000.00</td>
</tr>
<tr>
<td>W. Wei</td>
<td>2000.00</td>
</tr>
<tr>
<td>H. Martin</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>
Another SQL query using two tables

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Transaction-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

SELECT A.Number, A.Owner
FROM Account AS A, Deposit AS D
WHERE A.Number = D.Account and D.Amount > 300;
## SQL query using two tables (cont.)

### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

### Deposit

<table>
<thead>
<tr>
<th>Account</th>
<th>Transaction-id</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/00</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/00</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/00</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/00</td>
<td>10,000.00</td>
</tr>
</tbody>
</table>

```sql
SELECT A.Number, A.Owner
FROM Account AS A, Deposit AS D
WHERE A.Number = D.Account and D.Amount > 300;
```

### Number | Owner
---|---------|
102 | W. Wei |
104 | M. Jones |
105 | H. Martin |
### Queries and Physical Independence

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

Notice that a query is expressed against the schema.

```sql
SELECT Owner FROM Account WHERE Type = "checking";
```

But the query runs or executes against the instance (the data).

<table>
<thead>
<tr>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Smith</td>
</tr>
<tr>
<td>W. Wei</td>
</tr>
<tr>
<td>M. Jones</td>
</tr>
<tr>
<td>H. Martin</td>
</tr>
</tbody>
</table>
### Comments on Queries

<table>
<thead>
<tr>
<th>Account</th>
<th>Number</th>
<th>Owner</th>
<th>Balance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>J. Smith</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>W. Wei</td>
<td>2000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>J. Smith</td>
<td>5000.00</td>
<td>savings</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>M. Jones</td>
<td>1000.00</td>
<td>checking</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>H. Martin</td>
<td>10,000.00</td>
<td>checking</td>
<td></td>
</tr>
</tbody>
</table>

Notice that the answer to a query is always a **table**! It doesn’t always have a name (for the table). The attribute names are deduced from the input tables (or supplied by the query author). It may or may not have any rows.

- J. Smith
- W. Wei
- M. Jones
- H. Martin
Comments on Queries

Because the answer to a relational query is always a table, we can use the answer from one query as input to another query.

This means that we can create arbitrarily complex queries!

We say that relational query languages are closed when they have this property.
SQL … Extensions

Extension to the SELECT clause
- e.g., DISTINCT, SUM, COUNT, MIN, MAX, AVG and AS

Extension to the FROM clause
- e.g., correlation names and various kinds of JOINs

Extension to the WHERE clause
- e.g., AND, OR, NOT, comparators, EXISTS, IN, ANY

Several additional clauses
- e.g., ORDER BY, GROUP BY, and HAVING

And operators that expect two or more complete SQL queries as operands
- e.g., UNION and INTERSECT
Sample Database

For this discussion, we will use the following database:

Customer (Number, Name, Address, CRating, CAmount, CBalance, RegisterDate, SalespersonNum)

Foreign key: Customer.SalespersonNum references Salesperson.Number

Salesperson (Number, Name, Address, Office)
UNIONing Subqueries

```
(SELECT C.Name
FROM Customer C
WHERE C.Name LIKE "B%")
UNION
(SELECT S.Name
FROM Salesperson S
WHERE S.Name LIKE "B%");
```

Two complete queries - with UNION operator in between.

Unlike other operations, UNION eliminates duplicates!
UNION ALL

(SELECT C.Name
FROM Customer C
WHERE C.Name LIKE "B%")

UNION ALL

(SELECT S.Name
FROM Salesperson S
WHERE S.Name LIKE "B%");

UNION ALL preserves duplicates
EXCEPT (=difference)

(SELECT S.Number
FROM Salesperson)

EXCEPT

(SELECT C.SalespersonNum Number
FROM Customer C);

EXCEPT ALL retains duplicates

What is this query looking for?

Two complete queries - with EXCEPT operator in between.
EXCEPT (=difference)

(SELECT S.Number
FROM Salesperson;)

MINUS

(SELECT C.SalespersonNum Number
FROM Customer C;);

Oracle provides a MINUS operator to represent difference!
INTERSECT

(SELECT  S.Name
FROM    Salesperson)

INTERSECT

(SELECT  C.Name
FROM    Customer C);

INTERSECT ALL retains duplicates

What is this query looking for?

Two complete queries - with INTERSECT operator in between.
Eliminating Duplicates

Consider the following two queries:

```
SELECT DISTINCT Name
FROM Customer;
```

```
SELECT Name
FROM Customer;
```

The first query eliminates duplicate rows from the answer.

- Although the relational model is based on set, by default RDBMSs operate on multisets (*bags*)
- *The query writer gets to choose whether duplicates are eliminated*
Eliminating Duplicates: A Word of Caution

• In theory, placing a DISTINCT after select is harmless

• In practice, it is very expensive
  – The time it takes to sort a relation so that duplicates are eliminated can be greater than the time to execute the query itself!

  *Use DISTINCT only when you really need it*
Aggregates

• **Summarize or “aggregate” the values in a column**
• Operators: **COUNT, SUM, MIN, MAX, and AVG**
  – Apply to sets or bags of atomic values
• **SUM and AVG**: produce sum and average of a column with numerical values
• **MIN and MAX**:  
  – applied to column with numerical values, produces the smallest and largest value  
  – applied to column with character string values, produces the lexicographically first or last value
• **COUNT**: produces the number of values in a column  
  – Equivalently the number of tuples in a relation, *including duplicates*

SELECT AVG (CBalance)  
FROM Customer  
WHERE age > 35;
Aggregates and NULLs

• General rule: aggregates ignore NULL values
  – \( \text{Avg}(1,2,3,\text{NULL},4) = \text{Avg}(1,2,3,4) \)
  – \( \text{Count}(1,2,3,\text{NULL},4) = \text{Count}(1,2,3,4) \)

• But…
  – \( \text{Count}(\ast) \) returns the total number of tuples, regardless whether they contain NULLs or not
Aggregates and Duplicates

- Aggregates apply to bags
- If you want sets instead, use DISTINCT

```sql
SELECT COUNT(Name) FROM Customer;
SELECT COUNT(DISTINCT Name) FROM Customer;
```

Answer: 3  Answer: 2

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Smith</td>
</tr>
<tr>
<td>W. Wei</td>
</tr>
<tr>
<td>J. Smith</td>
</tr>
</tbody>
</table>
Note: Full-Relation Operations

• DISTINCT and aggregates act on relations as a whole, rather than on individual tuples

• More on aggregates later!
SQL … Extensions

SELECT…

FROM...

WHERE...

Extension to the SELECT clause
e.g., DISTINCT, SUM, COUNT, MIN, MAX, AVG and AS

Extension to the FROM clause
e.g., correlation names, subqueries and various kinds of JOINs
Joins

There are a number of join types that can be expressed in the FROM clause:

– inner join (the theta join)
– cross join (Cartesian product)
– natural join
– left outer join
– right outer join
– full outer join
– union join
There are a number of join types that can be expressed in the FROM clause:

- inner join (the theta join)
- cross join
- natural join
- left outer join
- right outer join
- full outer join
- union join

These are syntactic sugar ... they can be expressed in a basic SELECT..FROM..WHERE query.
Joins

There are a number of join types that can be expressed in the FROM clause:

– inner join (the regular join)
– cross join
– natural join
– left outer join
– right outer join
– full outer join
– union join

There are new operators but can be expressed in a complex SQL query involving the union operator.
How would you write the following query?

Customer (Number, Name, SalespersonNum)
Salesperson(Number, Name, Address, Office)

• List the names of all salespeople and their customers.
  
  SELECT Customer.name, Salesperson.name 
  FROM Customer, Salesperson 
  WHERE Salesperson.number=SalespersonNum

• Does this SQL query do the job?
How would you write the following query?

Customer (Number, Name, SalespersonNum)
Salesperson(Number, Name, Address, Office)

• List the names of all salespeople and their customers.
  
  SELECT Customer.name, Salesperson.name
  FROM Customer, Salesperson
  WHERE Salesperson.number = SalespersonNum

• Does this SQL query do the job?
No! salespeople without customers will *not* be listed.
Outer Join

• An extension of the join operation that avoids loss of information.
• Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
• Uses *null* values to pad *dangling tuples*
LEFT OUTER JOIN

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>

JOIN on C.SalespersonNum = S.Number gives us:
“smith” with “johnson” and “jones” with “johnson”

LEFT OUTER JOIN on C.SalespersonNum = S.Number gives us:
JOIN plus “wei” with “<null>” salesperson
- Lists all customers and their salesperson, if any
RIGHT OUTER JOIN

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>

JOIN on C.SalespersonNum = S.Number gives us:
“smith” with “johnson” and “jones” with “johnson”

RIGHT OUTER JOIN on C.SalespersonNum = S.Number gives us:
INNER JOIN plus “<null>” customer with “miller”
- Lists customers that have a salesperson, and salespersons that do not have a customer
FULL OUTER JOIN

FULL OUTER JOIN = LEFT OUTER JOIN ∪ RIGHT OUTER JOIN

FULL OUTER JOIN on C.SalespersonNum = S.Number gives us:

INNER JOIN
   plus “wei” with “<null>” salesperson
   plus “<null>” customer with “miller”

- Lists all customer-salesperson pairs, and customers that do not have a salesperson, and salespersons that do not have a customer
Subqueries in the FROM clause

• The FROM clause takes a relation, but results from SQL queries are themselves relations, so we can use them in the FROM clause, too!

```
SELECT (N.CRating+1) AS CIncrRating
FROM (SELECT * FROM Customer WHERE CRating = 0) AS N
WHERE N.CBalance = 0
```

• Can this be written without nesting?

```
SELECT (CRating+1) AS CIncrRating
FROM Customer
WHERE CRating = 0 AND CBalance = 0
```
Subqueries

• A parenthesized SELECT-FROM-WHERE statement (subquery) can be used as a value in a number of places, including FROM and WHERE clauses.

• Example: in place of a relation in the FROM clause, we can use a subquery and then query its result.
  – Must use a tuple-variable to name tuples of the result.
SQL … Extensions

- Extension to the SELECT clause:
  - e.g., SUM, COUNT, MIN, MAX, AVG and AS

- Extension to the FROM clause:
  - e.g., correlation names and various kinds of JOINs

- Extension to the WHERE clause:
  - e.g., AND, OR, NOT, comparators, EXISTS, IN, ANY
WHERE Clause: Comparison Operators

- `<`, `>`, `=`, `<>`, `>=`, `<=
  - Compare two values as expected
  - Operates on numbers as well as text values
  - Amount < 50 and CustID <> 1
  - custName = ‘Juliana’ || ‘Freire’ (string concatenation)

- LIKE
  - Compare a text value with a pattern
  - ‘%’ compares with zero or more characters
  - ‘_’ compares with exactly one character
  - custName LIKE ‘%Fr__re’ – matches ‘Juliana Freire’, ‘Freire’, ‘Friere’
Subqueries in the WHERE clause

SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating IN (SELECT MAX (C2.CRating)
FROM Customer C2);

Find all customers where their credit rating is equal to the highest credit rating that appears in the database.

To understand semantics of nested queries, think of a nested loops evaluation: for each customer tuple, check the qualification by computing the subquery
## IN <subquery>: Example

### Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

```sql
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating IN (SELECT MAX (C2.CRating)
                        FROM Customer C2);
```

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating IN {10}
```

Result: 3, wei
NOT IN <subquery>: Example

<table>
<thead>
<tr>
<th>Customer</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Name</td>
<td>Address</td>
<td>CRating</td>
<td>CAmount</td>
<td>CBalance</td>
<td>SalespersonNum</td>
</tr>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating NOT IN
(SELECT MAX (C2.CRating)
FROM Customer C2);

Result: ?

SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating NOT IN {10}
Conditions Involving Relations: IN and NOT IN

SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating IN
(SELECT MAX (C2.CRating)
FROM Customer C2;);

• <attribute-name A> IN (subquery S): tests set membership
  – A is equal to one of the values in S

• <attribute-name A> NOT IN (subquery S)
  – A is equal to no value in S
Conditions Involving Relations: EXISTS

- **EXISTS** R is true if R is not empty
- **NOT EXISTS** R is true if R is empty

```
SELECT   C.Name
FROM     Customer C
WHERE    EXISTS (SELECT *
                   FROM    Salesperson S
                   WHERE   S.Number =
                            C.SalespersonNum);
```

If the answer to the subquery is not empty - then the EXISTS predicate returns TRUE

Tests for empty relations

What does this query compute?
## EXISTS: Example

### Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

### Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

```sql
SELECT C.Name
FROM Customer C
WHERE EXISTS (SELECT *
    FROM Salesperson S
    WHERE S.Number = C.SalespersonNum);
```
Conditions involving relations: NOT EXISTS

- **NOT EXISTS** $R$ is true if $R$ is empty

```
SELECT C.Name
FROM Customer C
WHERE NOT EXISTS (SELECT *
                      FROM Salesperson S
                      WHERE S.Number = C.SalespersonNum);
```

If the answer to the subquery is empty - then the NOT EXISTS predicate returns TRUE

Tests for non-empty relations

What does this query compute?
NOT EXISTS: Example

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>

SELECT  C.Name
FROM    Customer C
WHERE   NOT EXISTS (SELECT *  
            FROM Salesperson S   
            WHERE S.Number = C.SalespersonNum;);
Set comparison:
ALL or ANY in a Subquery

• Syntax:
  – attribute-name comparator ALL (subquery)
  – attribute-name comparator ANY (subquery)

• A > ALL (subquery S):
  – True if A is greater than every value returned by S
  – (A <> ALL S) ≡ (A NOT IN S)

• A > ANY (subquery S)
  – True if A is greater than at least one value returned by S
  – (A = ANY S) ≡ (A IN S)
ALL or ANY in a Subquery: Example

This predicate must be true for all SalespersonNums returned by the subquery!

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number = ALL (SELECT C.SalespersonNum
                         FROM    Customer;
                   );
```

What does this query compute?

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>
ALL or ANY in a Subquery: Example

```
SELECT C.Name
FROM Customer C
WHERE C.Crating >= ALL (SELECT C1.Crating
                         FROM Customer C1);
```

*What does this query compute?*

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>
Correlated Subqueries

- The simplest subqueries can be evaluated once and for all and the result used in a higher-level query.
- More complicated subqueries must be evaluated once for each assignment of a value to a term in the subquery that comes from a tuple outside the subquery.

```sql
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.SalespersonNum
                      FROM Customer C
                      WHERE C.Name = S.Name);
```

Because the subquery mentions an attribute from a table in the outer query.
Subquery that is not correlated

```
SELECT C1.Number, C1.Name
FROM Customer C1
WHERE C1.CRating IN
  (SELECT MAX (C2.CRating)
   FROM Customer C2;
  );
```

The subquery only uses attributes from the the table mentioned in the subquery
Correlated Subqueries: Scoping

• An attribute in a subquery belongs to one of the tuple variables corresponding to the closest relation
  – In general, an attribute in a subquery belongs to one of the tuple variables in that subquery’s FROM clause
  – If not, look at the immediately surrounding subquery, then to the one surrounding that, and so on.
Correlated Subqueries: Semantics

- Analyze from the inside out
  - For each tuple in the outer query, evaluate the innermost subquery, and replace that with the resulting relation
  - Repeat

```sql
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN (SELECT C.SalespersonNum
                      FROM Customer C
                      WHERE C.Name = S.Name);
```
Correlated Subqueries: Semantics

As we range through the Salesperson tuples, each tuple provides a value for S.Name.

Can’t evaluate, don’t know The value for S.Name

De-correlate: another way to write this query:

These two queries are equivalent. Is one preferable to the other?
SQL ... Extensions

SELECT...
FROM...
WHERE...
ORDER BY...
GROUP BY...
HAVING ...

Extension to the SELECT clause
e.g., SUM, COUNT, MIN, MAX, AVG and AS

Extension to the FROM clause
e.g., correlation names and various kinds of JOINs

Extension to the WHERE clause
e.g., AND, OR, NOT, comparators, EXISTS, IN, ANY

Several additional clauses
e.g., ORDER BY, GROUP BY, and HAVING
ORDER BY

Sort the result on one or more attributes
Can specify ASC, DESC--default is ASC

SELECT Name, Address
FROM Customers
ORDER BY Name

SELECT *
FROM Customer C JOIN Salesperson S
  ON C.SalespersonNum
ORDER BY CRating DESC, C.Name, S.Name
# ORDER BY: Example

## Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

## Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>

```sql
SELECT Name, Address
FROM Customers
ORDER BY Name
```

**Answer:**

Jones, yyy  
Smith, xxx  
Wei, zzz
SELECT *
FROM Customer C JOIN Salesperson S
ON C.SalespersonNum
ORDER BY CRating DESC, C.Name, S.Name

Answer:
3, wei, zzz, 10 ...
2, jones, yyy, 7
1, smith, xxx, 5
### ORDER BY: Example

#### Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>Ann</td>
<td>aaa</td>
<td>7</td>
<td>3,000</td>
<td>20,000</td>
<td>102</td>
</tr>
<tr>
<td>4</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>102</td>
</tr>
</tbody>
</table>

#### Salesperson

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>johnson</td>
<td>aaa</td>
<td>23</td>
</tr>
<tr>
<td>102</td>
<td>miller</td>
<td>bbb</td>
<td>26</td>
</tr>
</tbody>
</table>

**Answer:**

4, wei, zzz, 10 …
3, ann, aaa, 7…
2, jones, yyy, 7…
1, smith, xxx, 5…

**SELECT ***
**FROM Customer C JOIN Salesperson S ON C.SalespersonNum**
**ORDER BY CRating DESC, C.Name, S.Name**
Grouping

- `GROUP BY` partitions a relation into groups of tuples that *agree* on the value of one or more columns.
- Useful when combined with aggregation – apply aggregation within each group.
- Any form of SQL query (e.g., with or without subqueries) can have the answer “grouped”.
- The query result contains *one output row for each group*.
GROUP BY

```sql
SELECT SalespersonNum, COUNT(*) as TotCust
FROM Customer
GROUP BY SalespersonNum;
```

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>102</td>
</tr>
</tbody>
</table>

Answer

<table>
<thead>
<tr>
<th>SalespersonNum</th>
<th>TotCust</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>2</td>
</tr>
<tr>
<td>102</td>
<td>1</td>
</tr>
</tbody>
</table>
Challenge Question

• What is the answer for the query:
  SELECT SalespersonNum
  FROM Customer
  GROUP BY SalespersonNum

Customer

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>102</td>
</tr>
</tbody>
</table>

???
Challenge Question

• What is the answer for the query:
SELECT SalespersonNum
FROM Customer
GROUP BY SalespersonNum

<table>
<thead>
<tr>
<th>Customer</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Name</td>
<td>Address</td>
<td>CRating</td>
<td>CAmount</td>
<td>CBalance</td>
<td>SalespersonNum</td>
</tr>
<tr>
<td>1</td>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>102</td>
</tr>
</tbody>
</table>

Answer

<table>
<thead>
<tr>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>
Another Challenge Question

• Can you write a simpler SQL stmt for this query?

SELECT SalespersonNum
FROM Customer
GROUP BY SalespersonNum

Answer

<table>
<thead>
<tr>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>
HAVING Clauses

• Select groups based on some aggregate property of the group
  – E.g., Only list a salesperson if he/she has more than 10 customers
• The **HAVING clause is a condition evaluated against each group**
  – A group participates in the query answer if it satisfies the HAVING predicate

```
SELECT SalespersonNum
FROM Customer
GROUP BY SalespersonNum
HAVING Count(*) > 10;
```
GROUP BY Clauses and NULLS

- Aggregates ignore NULLs
- On the other hand, NULL is treated as an ordinary value in a grouped attribute
- If there are NULLs in the Salesperson column, a group will be returned for the NULL value

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>CRating</th>
<th>CAmount</th>
<th>CBalance</th>
<th>SalespersonNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>smith</td>
<td>xxx</td>
<td>5</td>
<td>1,000</td>
<td>1,000</td>
<td>101</td>
</tr>
<tr>
<td>jones</td>
<td>yyy</td>
<td>7</td>
<td>5,000</td>
<td>4,000</td>
<td>101</td>
</tr>
<tr>
<td>wei</td>
<td>zzz</td>
<td>10</td>
<td>10,000</td>
<td>10,000</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

Answer

```
<table>
<thead>
<tr>
<th>SalespersonNum</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
</tr>
</tbody>
</table>
```

```
SELECT SalespersonNum, Count(*) AS T
FROM Customer
GROUP BY SalespersonNum
```
GROUP BY, HAVING: Note

- The only attributes that can appear in a "grouped" query answer are aggregate operators (that are applied to the group) or the grouping attribute(s)

```
SELECT SalespersonNum, COUNT(*)
FROM Customer
GROUP BY SalespersonNum;
```

```
SELECT SalespersonNum, C.Name, COUNT(*)
FROM Customer C
GROUP BY SalespersonNum;
```

```
SELECT SalespersonNum
FROM Customer
GROUP BY SalespersonNum
HAVING Count(*) > 10;
```

Incorrect!
THANKS!
### Readable SQL Queries

- **Offer visual clues to the structure of query**
  - Each ‘important’ keyword starts a new line
  - Capitalize keywords

- **Keep it compact**
  - If query or subquery is short, write in a single line

```
SELECT * FROM Customer
```

```
SELECT SalespersonNum 
FROM Customer 
GROUP BY SalespersonNum 
HAVING Count(*) > 10 
ORDER BY SalespersonNum
```
Order of Clauses in SQL Queries

- **SELECT** and **FROM** are required
- Can’t use **HAVING** without **GROUP BY**
- Whichever additional clauses appear must be in the order listed
The WITH Clause

- Complex queries are easier to write if you break them up into smaller components
- You can name a query component using the WITH clause
  - It creates a temporary view, which is valid *only* in the query where it is defined

```sql
WITH max_balance(value) AS
  SELECT MAX(balance)
  FROM Customer
SELECT Cname
FROM Customer C, max_balance
WHERE C.balance = max_balance.value
```
Modifying the Database
Database Modifications

• Some SQL statements do not return any results…

• They change the state of the database
  – Insert tuples into a relation
  – Delete certain tuples from a relation
  – Update values of certain components of existing tuples
Example
Deletion

\[
\text{DELETE FROM } R \\
\text{WHERE } \langle \text{condition} \rangle
\]

- Delete whole tuples, one relation at a time
- Finds and deletes all tuples \( t \) in \( R \) such that \( \text{condition}(t) \) is true

- Examples:

  - Delete all account records at the Perryridge branch
    
    \[
    \text{DELETE FROM account} \\
    \text{WHERE branch-name} = \text{‘Perryridge’}
    \]

  - Delete all accounts at every branch located in Needham city.
    
    \[
    \text{DELETE FROM account} \\
    \text{WHERE branch-name} \text{ IN (} \\
    \quad \text{SELECT branch-name} \\
    \quad \text{FROM branch} \\
    \quad \text{WHERE branch-city} = \text{‘Needham’} \text{ )}
    \]
What does the following statement do?

- `delete from account`
Delete: Example

- Delete the record of all accounts with balances below the average at the bank.

```sql
delete from account
where balance < (select avg (balance) from account)
```

★ Problem: as we delete tuples from `account`, the average balance changes

Solution used in SQL:
1. First, compute `avg` balance and find all tuples to delete
2. Next, delete all tuples found above (without recomputing `avg` or retesting the tuples)
Inserting a Tuple into a Relation

\[ \text{INSERT INTO } R(A_1, \ldots, A_n) \text{ VALUES } (v_1, \ldots, v_n) \]

- A tuple is created using value \( v_i \) for attribute \( A_i \), for \( i=1, \ldots, n \)

\[ \text{insert into account } (\text{branch-name, balance, account-number}) \]
\[ \text{values} \ (\text{‘Perryridge’, 1200, ‘A-9732’}) \]

\[ \text{INSERT INTO } R \text{ VALUES } (v_1, \ldots, v_n) \]

- A tuple is created using value \( v_i \) for all attributes \( A \) of \( R \)
  - \( \text{Order of values must be the same as the standard order of the attributes in the relation} \)

\[ \text{insert into account} \]
\[ \text{values} \ (\text{‘A-9732’, ‘Perryridge’, 1200}) ---- \text{correct order!} \]
Inserting a Tuple into a Relation (cont.)

\[
\begin{align*}
\text{insert into} & \\
\text{account (branch-name, account-number)} & \\
\text{values} & (\text{‘Perryridge’}, \text{‘A-9732’})
\end{align*}
\]

Is equivalent to

\[
\begin{align*}
\text{insert into} & \\
\text{account (branch-name, account-number,balance)} & \\
\text{values} & (\text{‘A-9732’}, \text{‘Perryridge’}, \text{NULL})
\end{align*}
\]

- If a value is omitted, it will become a NULL
Inserting the Results of a Query

• Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

```sql
insert into account
    select loan-number, branch-name, 200
    from loan
    where branch-name = 'Perryridge'

insert into depositor
    select customer-name, loan-number
    from loan, borrower
    where branch-name = 'Perryridge'
    and loan.account-number = borrower.account-number
```

Set of tuples to insert
Order of Insertion

```
insert into account
    select loan-number, branch-name, 200
from loan
where branch-name = 'Perryridge'
insert into depositor
    select customer-name, loan-number
from loan, borrower
where branch-name = 'Perryridge'
    and loan.account-number = borrower.account-number
```

- The **select from where** statement is fully evaluated before any of its results are inserted into the relation.

- What would happen with the following query?
  ```
  insert into table1 select * from table1
  ```
Updates

• Choose tuples to be updated using a query

```
update R
set attribute = expression
where <condition>
```

• Pay 5% interest on accounts whose balance is greater than average

```
update account
set balance = balance * 1.05
where balance > (select avg(balance)
                from account)
```
Update: Example

• Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

  – Write two update statements:
    
    update account
    set balance = balance * 1.06
    where balance > 10000

    update account
    set balance = balance * 1.05
    where balance ≤ 10000

  – The order is important, why? Accounts with balance > 10000 would be updated twice!

  – Can be done better using the case statement (next slide)
Case Statement for Conditional Updates

• Same query as before: Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.

```sql
update account
set balance = case
    when balance <= 10000
    then balance * 1.05
    else balance * 1.06
end
```
SQL as a Data Definition Language (DDL)
Data Definition Language (DDL)

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- *Integrity constraints*
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.
Domain Types in SQL

- **char(n)**. Fixed length character string, with user-specified length $n$.
- **varchar(n)**. Variable length character strings, with user-specified maximum length $n$.
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d)**. Fixed point number, with user-specified precision of $p$ digits, with $n$ digits to the right of decimal point.
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n)**. Floating point number, with user-specified precision of at least $n$ digits.

Null values are allowed in all the domain types. Declaring an attribute to be **not null** prohibits null values for that attribute.

- **create domain** construct in SQL-92 creates user-defined domain types
  
  ```sql
  create domain person-name char(20) not null
  ```
Date/Time Types in SQL (Cont.)

- **date.** Dates, containing a (4 digit) year, month and date
  - E.g. date ‘2001-7-27’
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. time ’09:00:30’ time ’09:00:30.75’
- **timestamp:** date plus time of day
  - E.g. timestamp ‘2001-7-27 09:00:30.75’
- **Interval:** period of time
  - E.g. Interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values
- Can extract values of individual fields from date/time/timestamp
  - E.g. extract (year from r.starttime)
- Can cast string types to date/time/timestamp
  - E.g. cast <string-valued-expression> as date
Create Table Construct

• An SQL relation is defined using the **create table** command:

  \[
  \text{create table } r \ (A_1 \ D_1, \ A_2 \ D_2, \ ..., \ A_n \ D_n, \\
  \text{ (integrity-constraint}_1), \\
  \text{ ..., } \\
  \text{ (integrity-constraint}_k))
  \]

  – \( r \) is the name of the relation
  – each \( A_i \) is an attribute name in the schema of relation \( r \)
  – \( D_i \) is the data type of values in the domain of attribute \( A_i \)

• Example:

  \[
  \text{create table branch} \\
  \ (branch-name \ \text{char(15)} \ \text{not null}, \\
  \ branch-city \ \text{char(30)}, \\
  \ assets \ \text{integer})
  \]
Integrity Constraints in Create Table

• not null
• primary key \((A_1, \ldots, A_n)\)
• check \((P)\), where \(P\) is a predicate
  – \(P\) must be satisfied by all tuples

Example: Declare \(\text{branch-name}\) as the primary key for \(\text{branch}\) and ensure that the values of \(\text{assets}\) are non-negative.

\begin{verbatim}
create table branch
  (branch-name char(15),
   branch-city char(30),
   assets integer,
   primary key (branch-name),
   check (assets >= 0))

primary key declaration on an attribute automatically ensures not null in SQL-92 onwards, needs to be explicitly stated in SQL-89
\end{verbatim}
Integrity Constraints in Create Table

- **foreign key** \((A_1, ..., A_n)\) references \(R\)

Example: Create the borrower table which captures the relationship between borrower and customer, and between borrower and loan

```sql
create table borrower (customer_name varchar(30),
                        loan_number number(8),
                        CONSTRAINT fk1
                        FOREIGN KEY (customer_name)
                        REFERENCES customer (customer_name),
                        CONSTRAINT fk2
                        FOREIGN KEY (loan_number)
                        REFERENCES loan )
```
Integrity Constraints in Create Table

• **ON DELETE CASCADE**
• Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted.

```sql
create table borrower (customer_name varchar(30),
    loan_number number(8),
    CONSTRAINT fk1
    FOREIGN KEY (customer_name)
    REFERENCES customer (customer_name),
    CONSTRAINT fk2
    FOREIGN KEY (loan_number)
    REFERENCES loan ON DELETE CASCADE)
```
Drop and Alter Table Constructs

• The **drop table** command deletes all information about the dropped relation from the database.

• The **alter table** command is used to add attributes to an existing relation.

  \[\text{alter table } r \text{ add } A \ D\]

  where \(A\) is the name of the attribute to be added to relation \(r\) and \(D\) is the domain of \(A\).

  – All tuples in the relation are assigned \(null\) as the value for the new attribute.

• Examples:

  – ALTER TABLE borrower ADD b_date DATE
  – DROP TABLE borrower
Drop and Alter Table Constructs (cont.)

• The **alter table** command can also be used to drop attributes of a relation

  \[ \text{alter table } r \text{ drop } A \]

  where \( A \) is the name of an attribute of relation \( r \)
  – E.g., ALTER TABLE borrower DROP b_date

• Dropping of attributes not supported by many databases

• The **alter table** command can also be used to drop or add constraints
  – More about this later!
Default Values

• Any place we declare an attribute we may add the keyword DEFAULT followed by NULL or a constant
• Example:
  – Gender CHAR(1) DEFAULT ‘?’
  – Birthdate DATE DEFAULT DATE ‘0000-00-00’
Views

- Relation that is not part of the logical model
  
  `create view v as <query expression>`
  
  where `<query expression>` is any legal relational algebra query expression. The view name is represented by `v`.

- Once a view is defined, its name can be used to refer to the virtual relation that the view generates.

- View definition is not the same as creating a new relation by evaluating the query expression.

- A view definition causes the saving of an expression; the expression is substituted into queries using the view.
CREATE VIEW AllCustomers AS
    (SELECT branch-name, cust-name
     FROM depositor D, account A
     WHERE D.account-no=A.number)
    UNION
    (SELECT branch-name, cust-name
     FROM borrower B, loan L
     WHERE B.loan-no=L.number)

• To find all customers of the Perryridge branch:
  SELECT cust-name
  FROM AllCustomers
  WHERE branch-name= ‘Perryridge’
Views: Renaming Attributes

CREATE VIEW AllCustomers (bname, cname) AS

(SELECT branch-name, cust-name
 FROM depositor D, account A
 WHERE D.account-no=A.number)

UNION

(SELECT branch-name, cust-name
 FROM borrower B, loan L
 WHERE B.loan-no=L.number)

• To find all customers of the Perryridge branch:
  SELECT cname
  FROM AllCustomers
  WHERE bname=‘Perryridge’
Interpreting Queries that use Views

• To find all customers of the Perryridge branch:
  SELECT cust-name
  FROM AllCustomers
  WHERE branch-name=‘Perryridge’

  ---->
  (SELECT branch-name, cust-name
   FROM depositor D, account A
   WHERE D.account-no=A.number AND branch-name=‘Perryridge’ )

  UNION
  (SELECT branch-name, cust-name
   FROM borrower B, loan L
   WHERE B.loan-no=L.number AND branch-name=‘Perryridge’ )
Materialized Views

• Create a real table

CREATE MATERIALIZED VIEW hr.employees AS
SELECT * FROM hr.employees@orc1.world;