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

```
SELECT    *
FROM      Account
WHERE     Type = "checking ";
```

- Data definition: creation of tables and views

```
CREATE TABLE Account
    (Number          integer  NOT NULL,
     Owner           character,
     Balance         currency,
     Type            character,
    PRIMARY KEY (Number));
```

- Control: assertions to protect data integrity

```
CHECK (Owner IS NOT NULL)
```

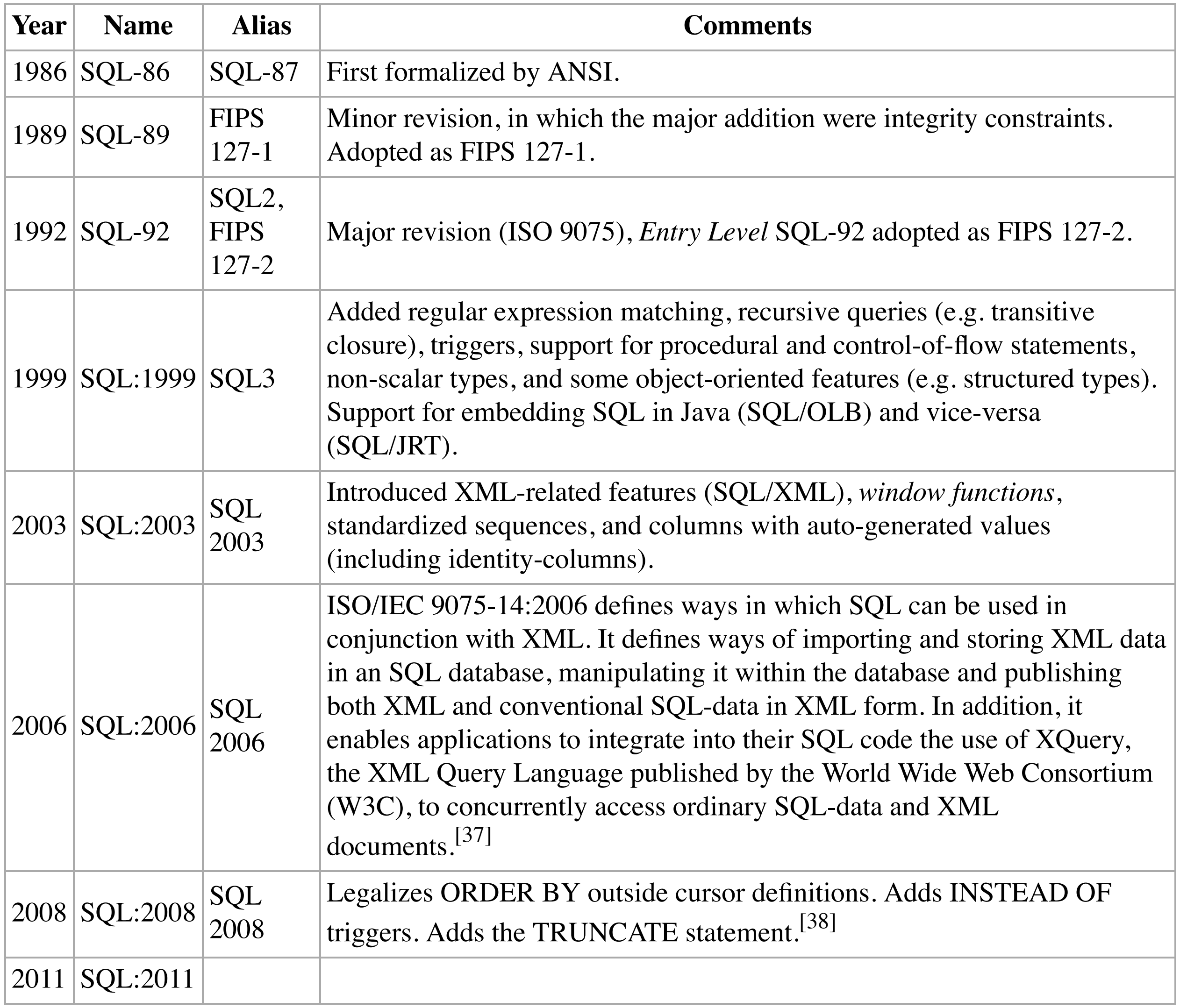
Relational Algebra vs. SQL

- 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)

SQL Standard



Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI.
1989	SQL-89	FIPS 127-1	Minor revision, in which the major addition were integrity constraints. Adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types). Support for embedding SQL in Java (SQL/OLB) and vice-versa (SQL/JRT).
2003	SQL:2003	SQL 2003	Introduced XML-related features (SQL/XML), *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[37]
2008	SQL:2008	SQL 2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers. Adds the TRUNCATE statement.[38]
2011	SQL:2011		

http://en.wikipedia.org/wiki/SQL#Standardization

Database Schema for Running Example

ACCOUNT	Number	CustId	Balance	Type

DEPOSIT	Account	TransactionID	Date	Amount

CHECK	Account	Check-number	Date	Amount

ATMWITHDRAWAL	TransactionID	CustID	AcctNo	Amount	WithdrawalDate

CUSTOMER	ID	Name	Phone	Address

SQL in Action: Find tuples that satisty a condition

ATMWithdrawal table

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

Attributes of the resulting relation

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

Relation to which the query refers

Condition that must be satisfied

ATMWithdrawal table

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

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

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

Amount < 50

Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	$25.00	11/1/2000 9:45:00

ATMWithdrawal table

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

Amount < 50

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

Ignore this record!

Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	$25.00	11/1/2000 9:45:00

ATMWithdrawal table

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

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

Amount < 50

Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	$25.00	11/1/2000 9:45:00
3	2	101	$40.00	11/1/2000 10:05:00

ATMWithdrawal table

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

Amount < 50

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

Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	$25.00	11/1/2000 9:45:00
3	2	101	$40.00	11/1/2000 10:05:00
4	2	100	$40.00	11/1/2000 10:07:00

ATMWithdrawal table

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

Amount < 50

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

Ignore this record!

Query Answer table

TransactionID	CustId	AcctNo	Amount	WithdrawDate
1	1	102	$25.00	11/1/2000 9:45:00
3	2	101	$40.00	11/1/2000 10:05:00
4	2	100	$40.00	11/1/2000 10:07:00

Selection in SQL and Relational Algebra

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

→

$\sigma_{\text{Amount} < 50}\ \text{ATMWithdrawal}$

Conditions in the WHERE Clause

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

- 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

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 = NULL, the condition x > 3 evaluates to UNKNOWN
- *Trick:* TRUE = 1; FALSE = 0; UNKNOWN=1/2
  - X **and** Y = min(X,Y)
  - X **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

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

Always return a copy of the ATMWithdrawal table?

????

Challenge Question

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

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

Always return a copy of the ATMWithdrawal table?

not if NULLs are allowed in the amount column!

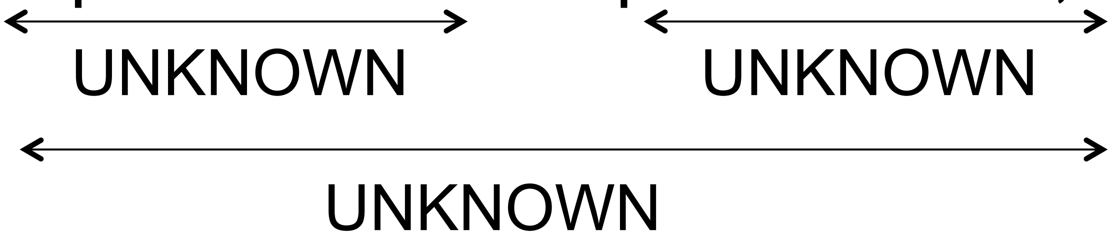
NULLs can lead to unexpected results…

Another Surprising Example

- From the following Bookstore relation:

name	book	price
Joe's Bar	HTMLP…	NULL

```
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:

$\pi_{AccNo,Amount}(\sigma_{Amount < 50}\ ATMWithdrawal)$

Query Answer table (Amount < 50)

~~TransactionID~~	~~CustId~~	AcctNo	Amount	~~WithdrawDate~~
~~1~~	~~1~~	102	$25.00	~~11/1/2000 9:45:00~~
~~3~~	~~2~~	101	$40.00	~~11/1/2000 10:05:00~~
~~4~~	~~2~~	100	$40.00	~~11/1/2000 10:07:00~~

```
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

AcctNo	Amount
102	$25.00
101	$40.00
100	$40.00

More on Projection in SQL

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

Renaming attributes

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

Number	Amt
102	$25.00
101	$40.00
100	$40.00

Generalized projection, remember?

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

Number	Amt
102	$250.00
101	$400.00
100	$400.00

More on Projection in SQL

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

AccNumber	Amt
102	$250.00
101	$250.00
100	$250.00

SQL vs Relational Algebra

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	1000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

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

	Owner	Balance
	J. Smith	**1000.00**
	W. Wei	2000.00
	J. Smith	**1000.00**
	M. Jones	1000.00
	H. Martin	10000.00

SQL vs Relational Algebra

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	1000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

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

Query results can be a *bag*

	Owner	Balance
	J. Smith	**1000.00**
	W. Wei	2000.00
	J. Smith	**1000.00**
	M. Jones	1000.00
	H. Martin	10000.00

Why Bags?

- Sets are simple and natural – but they can be inefficient to manipulate
  - Removing duplicates is expensive, possibly more than executing the query!
  - E.g., A U B, simply append the two relations—no need to eliminate duplicates
- 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? Avg({27,20}) or Avg({27,20,20})?

Relational Algebra on Bags

- A *bag* (or *multiset*) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.

Relational Operations on Bags

- Set operations: If tuple *t* appears n times in R and m times in S, it appears
  - n+m times in R U S
  - min(n,m) times in R ∩ S
  - max(0,n-m) times in R – S
- Other operations (join, projection, etc) work as expected, but duplicates are not removed from the results

Example: Bag Selection

R(

A,	B
1	2
5	6
1	2

)

$\sigma_{A+B < 5}$ (R) =

A	B
1	2
1	2

Example: Bag Projection

R(

A,	B
1	2
5	6
1	2

)

π_A (R) =

A
1
5
1

Example: Bag Product

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

Example: Bag Theta-Join

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

$R \bowtie_{R.B<S.B} S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

Bag Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
- Example: {1,2,1,1} ∩ {1,2,1,3} = {1,1,2}.

Bag Difference

- An element appears in the difference $A - B$ of bags as many times as it appears in *A*, minus the number of times it appears in *B*.
  - But never less than 0 times.
- Example: {1,2,1,1} – {1,2,3} = {1,1}.

Challenge Question

Let R, S and T be *bags*. Does the following statement hold?

(R U S) – T = (R –T) U (S –T)

????

Challenge Question

Let R, S and T be *bags*. Does the following statement hold?

$$(R\ U\ S) - T = (R - T)\ U\ (S - T)$$

Suppose $R = S = T = \{1\}$

$$(\{1\}\ U\ \{1\}) - \{1\} = \{1\}$$

<>

$$(\{1\} - \{1\})\ U\ (\{1\} - \{1\}) = \{\}$$

Beware: Bag Laws != Set Laws

- Some, but *not all* algebraic laws that hold for sets also hold for bags
- Example: the commutative law for union

  ($R \cup S = S \cup R$) *does* hold for bags

  - Since addition is commutative, adding the number of times *x* appears in *R* and *S* doesn't depend on the order of *R* and *S*.

Example: A Law That Fails

- Set union is *idempotent*, meaning that $S \cup S = S$.
- However, for bags, if x appears n times in S, then it appears $2n$ times in $S \cup S$.
- Thus $S \cup S$!= S in general.
  - e.g., {1} ∪ {1} = {1,1} != {1}.

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:

  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.

$$\prod_{A1,\ A2,\ ...,\ An}(\sigma_P\,(r_1 \;\; x\; r_2 \;\; x \;\; ... \;\; x \;\; r_m))$$

- 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:

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

$\pi_{\text{Number, Balance}} (\sigma_{\text{Acc\_num = Number and Amount > 10000}} (\text{Account X Deposit}))$

```
SELECT        *
FROM          Account, Deposit;
```

(Account X Deposit)

Account(Number, CustID,Balance,Type)

Deposit(Acc_num,TID,Date,Amount)

SQL query using two tables

Account(**Number**, CustID,Balance,Type)

Deposit(**Number**,TID,Date,Amount)

```
SELECT        A.Number, A.Balance
FROM          Account A, Deposit D
WHERE         D.Number = A.Number and D.Amount > 10000;
```

Notice that
"A" is a correlation name for Account
and
"D" is a correlation name for Deposit.

Correlation name = tuple variable

- You choose correlation names when you write the query.
- Useful for disambiguating attribute names, e.g., Account vs. Deposit number

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

```
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!

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount

notice the attributes

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Yes! Place in query answer.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Yes! Place in query answer.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

All combinations fail! →

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! Throw it away.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away. Why?

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

No! Throw it away.

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

No! The first three fail.

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

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

Number	Owner	Balance	Type	Account	T-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Deposit

Account	T-id	Date	Amount
102	1	10/22/00	500.00
102	2	10/29/00	200.00
104	3	10/29/00	1000.00
105	4	11/2/00	10,000.00

Yes! Place in query answer.

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

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

Intermediate result
(after processing the FROM & WHERE clauses)

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

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)

Owner	Balance
W. Wei	**2000.00**
W. Wei	**2000.00**
H. Martin	10,000.00

Another SQL query using two tables

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

```
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	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

```
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

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Notice that a query is expressed against the schema.

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

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

	Owner
	J. Smith
	W. Wei
	M. Jones
	H. Martin

Comments on Queries

Account	Numberq	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

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.

	Owner
	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

SELECT…

FROM…

WHERE…

ORDER BY…

GROUP BY…

HAVING …

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

(SELECT…FROM…WHERE…)
UNION
(SELECT…FROM…WHERE…)

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

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);
```

Two complete queries - with EXCEPT operator in between.

EXCEPT ALL retains duplicates

What is this query looking for?

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);
```

Two complete
queries -
with
INTERSECT
operator
in between.

INTERSECT ALL retains duplicates

What is this query looking for?