Experimental Finance

IEOR

Mike Lipkin, Alexander Stanton
Outline

• Pivot tables
• System design and data integrity
• A Different kind of database
Pivot Tables

- Pivots are generally not done in SQL, but very useful in certain cases

<table>
<thead>
<tr>
<th>myID</th>
<th>Date</th>
<th>strategy</th>
<th>pnl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11/25/2005</td>
<td>strategy1</td>
<td>120.23</td>
</tr>
<tr>
<td>2</td>
<td>11/26/2005</td>
<td>strategy1</td>
<td>-10</td>
</tr>
<tr>
<td>3</td>
<td>11/29/2005</td>
<td>strategy1</td>
<td>12.23</td>
</tr>
<tr>
<td>4</td>
<td>11/25/2005</td>
<td>strategy2</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>11/26/2005</td>
<td>strategy2</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>12/5/2005</td>
<td>strategy1</td>
<td>33.02</td>
</tr>
<tr>
<td>7</td>
<td>12/5/2005</td>
<td>strategy2</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>11/26/2005</td>
<td>strategy3</td>
<td>-77</td>
</tr>
<tr>
<td>9</td>
<td>11/29/2005</td>
<td>strategy4</td>
<td>32.6</td>
</tr>
<tr>
<td>10</td>
<td>11/25/2005</td>
<td>strategy3</td>
<td>121.1</td>
</tr>
<tr>
<td>11</td>
<td>11/26/2005</td>
<td>strategy4</td>
<td>14.4</td>
</tr>
<tr>
<td>12</td>
<td>12/5/2005</td>
<td>strategy3</td>
<td>87.2</td>
</tr>
<tr>
<td>13</td>
<td>12/5/2005</td>
<td>strategy4</td>
<td>287.1</td>
</tr>
<tr>
<td>14</td>
<td>11/29/2005</td>
<td>strategy2</td>
<td>-100.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>strategy1</th>
<th>strategy2</th>
<th>strategy3</th>
<th>strategy4</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/25/2005</td>
<td>120.23</td>
<td></td>
<td>12.1</td>
<td></td>
</tr>
<tr>
<td>11/26/2005</td>
<td>-10</td>
<td>55</td>
<td>-77</td>
<td>14.4</td>
</tr>
<tr>
<td>11/29/2005</td>
<td>12.23</td>
<td>-100.1</td>
<td>120.23</td>
<td>120.23</td>
</tr>
<tr>
<td>12/5/2005</td>
<td>120.23</td>
<td>120.23</td>
<td>-10</td>
<td>-10</td>
</tr>
<tr>
<td>12/6/2005</td>
<td>-10</td>
<td>-10</td>
<td>12.23</td>
<td>12.23</td>
</tr>
<tr>
<td>12/7/2005</td>
<td>-77</td>
<td>12.23</td>
<td>-100.1</td>
<td>32.6</td>
</tr>
</tbody>
</table>

Totals: 155.69  77.36  -42.54  169.46
Pivot Tables

- Pivot tables can be constructed using a SUM/CASE statement combined with a GROUP BY:

```
SELECT ticker, date,
    SUM(CASE WHEN strategy= 'strategy1' then pnl ELSE 0 END) as st1,
    SUM(CASE WHEN strategy= 'strategy2' then pnl ELSE 0 END) as st2,
    SUM(CASE WHEN strategy= 'strategy3' then pnl ELSE 0 END) as st3,
    SUM(CASE WHEN strategy= 'strategy4' then pnl ELSE 0 END) as st4
FROM ...
GROUP BY ticker, date
```

This can be much more powerful than excel when data checking is done on large datasets

- e.g. TOP 100 [..] ORDER BY max(pnl) max(avg(pnl))
System Design & Data Integrity
Data Integrity

- Might use TWA or VWAP calculation for pricing
Data Integrity

- How good are the fits now?
An Aside…

“The chart tool in Microsoft Excel permits you to fit a trend line and to display an R-squared value. If you click the Set Intercept = 0 box, the R-squared value is always incorrect in any version of Excel.”

“it reports the R-squared of the underlying linear regression of the logarithm of the data. The two numbers are close if the correlation is good, but far apart if it isn't.”
Designing a System

• How do we define our tables and database schema so to minimize these problems?

• 4 integrity methods:
  
  – **Entity Integrity**
    Ensures there are no duplicate rows in a table
  
  – **Domain Integrity**
    Restricting data values/types/format of fields
  
  – **Referential integrity**
    Ensures rows are not deleted that are required by other tables
  
  – **User-Defined Integrity**
    Other formats/checks/balances and business rules that do not fall under the categories above
Entity Integrity

• UNIQUE keyword can be used on a column:
  
  E.g. securityID is the primary key, and therefore unique in the security table

• Unique constraints can be added on multiple columns:
  
  ADD CONSTRAINT ct_SPSecurityIDDate UNIQUE (securityID, date)

• Think about the impact on AVERAGEs, SUMs etc. if uniqueness is not enforced

  UNIQUE constraints are critical, and always forgotten
Entity Integrity

• Unique Indices are often used to validate data:

  CREATE UNIQUE INDEX spSecurityIDDateIndex ON security_price (securityID, date)

• Difference between unique index and unique constraint is subtle in terms of exactly when the constraint is checked

• The table definition is where columns, foreign keys and checks are defined, and as such it is a good place to store unique constraints in terms of documentation

• Indices are separate objects that may or may not exist for a table. They are also often changed/manipulated and as such not a good place for unique constraints
Domain Integrity

- Determines what values/types/formats are allowed for various fields
  
  E.g. implied volatility is not NULL, with a default value of -99 and a “bad data” value of -99.

- It is not apparent to the end user whether the data could not be calculated using the IVY numeric solution (see appendix of IVY reference), or whether there was bad or missing data to perform the calculation.

- Most data providers (IVY, Thompson, Bloomberg, IVolatility) are not production system in-of-themselves: they provides the data but it is up to you to make it work.

- Some systems/vendors claim clean data – usually trading platforms that force you to use their scripts and algorithms… generally not money makers.

- Never expect the data to be correct - Assume the worst and investigate.
Domain Integrity - Check Constraints

• Validate data based on fixed criteria (also called rules)

• Common usage:

```
Create TABLE security_price (
    closeDate DATE,
    closePrice DECIMAL(10,2),
    volume DECIMAL(10,2) DEFAULT 0 NOT NULL,
    CONSTRAINT cc_validPriceVolume
    CHECK (closePrice IS NOT NULL AND volume IS NOT NULL)
)
```

• Ensures that there is at least some valid data in the row

• Can use any combination and formulas for checks (e.g. volume>100)

• Inspect the ivy table declarations to show their constraints

Shorthand for certain cases
Referential Integrity

- Within complex systems, referential integrity becomes highly important.
- Tables that relate to each other through a key (such as securityID) must not contain inconsistencies. Examples:
  - A security_price entry for a securityID that does not exist in the security table.
  - An option_price entry without a security_price.
  - Every price must have an associated exchange listing.
  - Every exchange entry in OPV must be listed in the exchange table.
  - Every dividend paid must have associated prices for the security.
  - Dividends can only exist in the distribution table for a security that is listed as paying dividends.

- Implementing referential checks saves a lot of data hunting headaches.
Referential Integrity - Foreign Keys

- Foreign keys implemented:
  
  ```sql
  ALTER TABLE security_price
  ADD CONSTRAINT Fk_Security_SecurityID
  FOREIGN KEY (securityID) REFERENCES Security(securityID)
  ON DELETE CASCADE ON UPDATE CASCADE
  [or ON DELETE NO ACTION ON UPDATE CASCADE]
  ```

- This statement prevents rows from being inserted without the securityID existing in the security table

- **CASCADE vs. NO ACTION**
  
  - CASCADE will cause the related table to delete rows referencing the primary key
  - NO ACTION causes the primary table not to be updated (e.g. results in a security that does not have any prices)

- Very useful, but beware of unintended performance implications (e.g. deleting a security from `security` where `security_price` and `option_price` are both traversed to deleted related records.)
Referential Integrity - Entity Relationship Diagram

1-many
1-1 With security

dbo_SECURITY
- SecurityID
- CUSIP
- Ticker
- SIC
- IndexFlag
- ExchangeFlags
- Class
- IssueType
- IndustryGroup

dbo_SECURITY_PRICE
- SecurityID
- Date
- BidLow
- AskHigh
- ClosePrice
- Volume
- TotalReturn
- AdjustmentFactor
- OpenPrice
- SharesOutstanding
- AdjustmentFactor2

dbo_OPTION_PRICE_VIEW
- SecurityID
- Date
- Root
- Suffix
- Strike
- Expiration
- CallPut
- BestBid

dbo_OPTION_INFO
- SecurityID
- DividendConvention
- ExerciseStyle
- ADSettlementFlag

dbo_EXCHANGE
- SecurityID
- Date
- SequenceNumber
- Status
- Exchange
- AddDelete
- ExchangeFlags

dbo_INDEX_DIVIDEND
- SecurityID
- Date
- Rate

dbo_DISTRIBUTION
- SecurityID
- RecordDate
- SequenceNumber
- ExDate
- Amount
- AdjustmentFactor
- DeclareDate
- PaymentDate
- LinkSecurityID
- DistributionType
- Frequency
- Time

dbo_VOLATILITY_SURFACE_VIEW
- SecurityID
- Date
- Days
- Days
- CallPut
- ImpliedVolatility
- ImpliedStrike
- ImpliedPremium
- Dispersion
Data Integrity

• Beyond built-in database checks for valid data, data integrity refers to the values of the data being accurate. E.g.
  – If an implied vol cannot be calculated, insert a value of -99
  – Don’t insert a dividend if there isn’t one
  – Don’t (or do?) insert a price on a day when pricing is not available. What value do we use if there is no data?
  – Why is the implied vol set to -99?

• Data integrity can be handled using:
  – DEFAULT in the CREATE TABLE statement
  – Data validation before data is inserted (not always easy – think real-time systems)
  – stored procedures/scripts that validate data
  – Real-time triggers
Data Integrity - Validation: scheduled or real-time

- Real-Time: immediate but time consuming

- Scheduled: Every night, a script checks the entire database for bad data and reports/corrects it

**Advantages:**
- all data is checked methodically
- can be run at “off-hours”, i.e. doesn’t impact business hours

**Disadvantages:**
- Bad data may already have been used
- Time it takes to check the data may be prohibitive
- Often the majority of rows haven’t changed and don’t need checking
Data Integrity - Triggers

• **Real Time**: Only mechanism that ensures in-line processing within the database.

• Used to enforce procedural integrity and are based on data and/or referential integrity rules that cannot be checked using constraints.

• Can perform actions based on the statement that is trying to be executed:

In the following example, the import value ‘NaN’ may come from another system that declares Nulls as NaN, unlike SQL:

```sql
INSERT INTO price(securityID, closePrice) VALUES(4233, 'NaN')
```

• This statement would fail, and if run in bulk mode may not be visible to anyone.

• By declaring a trigger, we can decide what action to take in real-time, e.g. convert to our definition of NULL, insert the problematic data into a log or “error table”, email an alert if it is critical, etc.
Data Integrity - Triggers

• Before choosing to implement a trigger, consider whether the same results can be achieved by using constraints or rules

• Use triggers if:
  – If a value in one table must be validated against a value in another table
  – If a value in another table must always change when a row gets updated (e.g. a count)
  – If complex events are required

```
CREATE TRIGGER updateReminder
ON earnings FOR UPDATE, DELETE
AS
BEGIN
  DECLARE @ticker VARCHAR(255)
  SELECT @ticker=ticker FROM INSERTED
  EXEC master..xp_sendmail 'traderJoe', 'Earnings changed for ' + @ticker
END
```
Data Integrity - Triggers

- Multiple triggers can be defined on the same table
- Notice the use of table keywords INSERTED/UPDATED/DELETED which refer to the table the trigger is acting on
- Never use the actual table name to find the inserted data since it hasn’t been committed to the database yet
- By default, triggers always fire before the INSERT/UPDATE/DELETE operation happens
- Triggers can execute before or after all other constraints and triggers have validated (useful to prevent multiple error notifications) [See keyword: “AFTER”]
Data Integrity - Triggers

- Keyword INSTEAD OF allows the operation to complete, but for rows that failed a different action is taken, e.g. writing the suspect data to a different table.

In General:

- Keep triggers simple
- Consider the insert/update/delete load on the table and what performance issues might ensue
- Triggers on views are permitted
- Beware recursive triggers which will cause serious problems…
User-Defined Integrity

- Cross check facts against other data sources and databases (or even use the telephone…)
- Key parameter checks (e.g. ensuring that calculations run by checking timestamps and changes in values)
- If contemplating a small universe of stocks with signal, make sure signals are re-established periodically to ensure the landscape has not changed
User-Defined Integrity

• Do computations have to work atomically (i.e. the entire calculation has to complete) or is information output relevant even if certain steps failed?

  E.g. a stored proc that outputs trade recommendation based on X parameters. Perhaps a certain stock’s parameters are invalid and cannot be computed, however other stocks have recommendation factors that can be computed.
  Computing a series of theoretical strikes may be invalid unless all strikes can be created within a certain interval.

• Knowing the percentage of calculations that complete

  A table that lists 3 recommended stocks is not as helpful as a table that lists 3 stocks and 200 others with a reason why they weren’t recommended (e.g. “no vol information”, “dividend missing” or “score of 22 too low”)
Constraint Summary

- **PRIMARY KEY constraint**
  
  Defines a unique row identifier based on one or more columns
  Determines physical database layout of how data is organized

- **UNIQUE constraint**
  
  Defines which column(s) must be unique

- **FOREIGN KEY constraints**
  
  Describes relationships with other entities

- **CHECK constraints**
  
  Validate data based on fixed criteria
Column-Based Databases
A Different Kind Of Database

• We’ve been looking at RDBMS, i.e. row-based databases suited for On-Line Transaction Processing (OLTP) (MS-SQL, MySQL, Postgres, Oracle, DB2 are all row-based)

• Two other types of databases are widely used: Column based (KDB, Casandra, HBase) and Document based (MongoDB, CouchDB)

• Most databases provide a translation to a mix of row-based (OLTP) and column-based services which allows for On-Line Analytical Processing (OLAP)
• RDBMS systems are aimed at relational data that need complex querying, whereas Column DBs are aimed at processing enormous amounts of mostly non-relational data.

• A true column-based database is something like KDB, which is used more and more frequently for high speed analysis, real time data and quoting engines.

• Kdb+ is multi-threaded and multi-process, can run in a distributed mode to scale across many machines, to petabyte sizes.

• It is supported under clusters, grids, clouds and other large scale distributed architectures.
Row vs. Column Databases

<table>
<thead>
<tr>
<th>EmpId</th>
<th>Lastname</th>
<th>Firstname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>Joe</td>
<td>40000</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>Mary</td>
<td>50000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>Cathy</td>
<td>44000</td>
</tr>
</tbody>
</table>

**Row-based representation:** [1,Smith,Joe,40000;2,Jones,Mary,50000;3,Johnson,Cathy,44000;]

**Column-based representation:**

[1,2,3;Smith,Jones,Johnson;Joe,Mary,Cathy;40000,50000,44000;]

- Two major columns-based efficiencies:
  - When an aggregate needs to be computed over many rows for a much smaller subset of all data
  - When new values of a column are supplied for all rows at once (e.g. algo’s computing vol surfaces)

KDB provides the stand alone database as well as tick and taq data
Benefits of KDB

- Handles in-memory databases very well
- Very fast for certain operations
- Provides a database-level superset of the language K called Q
- Single language with sufficient mathematical powers greatly simplifies coding (compared to SQL+Matlab for example)
- Array processing
- Native time series processing
- Nano second timestamps
- In addition to tables, lists and dictionaries are primitive types
- [http://www.kx.com](http://www.kx.com) (free for personal use)
Why we didn’t use KDB

- Not wide-spread
  
  Maybe 1000 programmers vs. 100,000s for SQL
  
  That may be a good thing for you!

- Most systems still run on row databases for good reason

- A challenge to learn for those used to scalar programming languages like C, Java, etc.

- You might think that SQL is not much better…
Why we didn’t use KDB

• Not wide-spread
  Maybe 1000 programmers vs. 100,000s for SQL
  That may be a good thing for you!

• Most systems still run on row databases for good reason

• A challenge to learn for those used to scalar programming
  languages like C, Java, etc.

• You might think that SQL is not much better…

• Or maybe it is:

  \[ 2!!7!4 \equiv 7 \pmod{4}, \text{ then list all integers less than the result (3), then rotate the list twice to the left to produce} \]

  “2; 0; 1”
Why we didn’t use KDB

- Intersect of two sets:

SQL: SELECT * FROM X INNER JOIN Y ON X.ID=Y.ID

Q:

intersect: {[x;y]
  x,: ()
  y,: ()
  if[(#x) < (#y)
    i: x ?/: y
    j: & i < #x
    x[?i[j]]
  ]
  i: y ?/: x
  j: & i < #y
  y[?i[j]] }


Why we didn’t use KDB

• Which dates does a symbol trade on?

SQL:  SELECT DISTINCT priceTable.symbol FROM priceTable

Q:  getDates:{[table; testSyms; startDate; endDate] symsByDate: select distinct sym by date from table[] where date within startDate endDate; firstSymList:(value symsByDate)[`sym'][0]; val:(each[@[(type firstSymList)$;;`badCast];]testSyms) except `badCast; (select date from (select date, each[@ [[seek;symList] (seek in symList)]/]val;]) sym from symsByDate) where sym=1b)[`date]}

Why you would use KDB

• List all prime numbers between 1 and R:

Q: (!R)&@&{&/x!/:2_!x}''!R

SQL:

```
SELECT * getDates:{[table; testSyms; startDate; endDate] symsByDate: select distinct sym
by date from table} where date within startDate endDate; firstSymList(value symsByDate)
[ sym][0]: val(each@[type firstSymList]$; 'badCast)\[testSyms) except badCast: (select date from (select date, each[[seek;symList]seek in
symList)]\[val][] sym from symsByDate) where sym=1b[' date])
SELECT * getDates:{[table; testSyms; startDate; endDate] symsByDate: select distinct sym
by date from table} where date within startDate endDate; firstSymList(value symsByDate)
[ sym][0]: val(each@[type firstSymList]$; 'badCast)\[testSyms) except badCast: (select date from (select date, each[[seek;symList]seek in
symList)]\[val][] sym from symsByDate) where sym=1b[' date])
```

Experimental Finance

Mike Lipkin, Alexander Stanton

Page 35
Project Tips

• Experimental finance is ultimately about (in)validating strategies, theories and ideas, and making money

• Think before you start – define your test, both failure and success tests

• Be judicious/cautious with calculations and conclusions

• Never take data at face value

• Explore the IVY Reference

• Spot check your findings manually – pen and paper if necessary

• The more ways you can test a hypothesis the better for your PNL

• Boundary conditions never fail to cause problems

• We don’t expect positive PNL (but then don’t expect a job either)

GOOD LUCK.....