BINF 8211/6211
Design and Implementation of Bioinformatics Databases
Lecture #14

Dr. D. Andrew Carr
Dept. Bioinformatics and Genomics UNCC
Spring 2016

TEAM	
PK	Team_ID
FK1	Division_ID
	Team_Name

DIVISION	
PK	Div_ID
FK1	Division_Name

PLAYER	
PK	Player_ID
FK1	Division_ID
	Player_Name

The ‘fan’ trap’ above, you can’t figure out what player belongs to what team, while below you can.

DIVISION	
PK	Div_ID
FK1	Division_Name

TEAM	
PK	Team_ID
FK1	Division_ID
	Team_Name

PLAYER	
PK	Player_ID
FK1	Team_ID
	Player_Name

Attributes of a good PK.

Characteristic	Explanation
Value is unique, not null	This is the heart of a PK requirement
Non-intelligible	An embedded semantic meaning turns out to be prone to mis-typing and mis-interpretation – an alias for the descriptive term is a better choice
Permanent	A side-effect of using a semantically meaningful PK is a change in interpretation over time (updates). Then changes will cascade to PKs and lots of unnecessary fun will ensue.
Single-attribute type	A single-attribute PK is preferred although not always possible. They make FKs easier to implement, and keep linking table PKs shorter.
Numeric	Ensuring uniqueness is readily done with a counter that increments, rather than using a name that must be checked.
Confidentiality	Even with numbers, make sure that the choice does not have privacy implications (like a SSN identifier)

Business Rules
Verify rules with users and then DOCUMENT (hearsay is not admissible….)
Rules identify entities, attributes, relationships and constraints as well as intended use in derived fields and views. Make them clear, precise and simple.
Give the reason for each rule (source, demand, date), the verification test and approval code.

ENTITIES
Represent a single object
Should be in 3NF (or higher)
Should have a well-defined granularity
Have a clearly defined PK that supports the granularity needed

Attributes
Simple and single-valued (‘atomic’)
Defined default values, constraints, synonyms and aliases
Derived attributes are clearly identified and sources are included
Redundant only as needed for transaction accuracy, for history, for FKs

RELATIONSHIPS
Clearly identify all participants
Clearly define participation and cardinality rules

ER Diagram
Validate against expected processes (insert, update, delete)
Evaluate where, when and how when a history is needed
Redundant relationships are present only for clearly defined needs
Overall data redundancy is minimized

Entity Naming Conventions: limit length (DBMS sensitive)
Use short meaningful (within context) nouns
Include common abbreviations, synonyms and aliases as meta-data
Ensure model-internal uniqueness
For composite entities you may link abbreviated entity names

Attribute Naming conventions: limit length
Ensure within-entity uniqueness
Use the entity abbreviation of prefix, as appropriate (unambiguous tie)
Aim to be descriptive, within context
For PK attributes use underscore abbreviation notation (e.g.: _ID)
Be aware of and avoid DBMS-reserved words (DATE is usually one such)
Avoid special characters and spaces

Relationship Naming conventions: limit length
Use verbs that indicate the nature of the relationship (e.g. ‘codes for’)

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	1322
2	deoB	1223
1	moaA	989
2	deoC	779
2	deoD	719
3	flgA	659
1	moaB	512
1	moaC	485
1	moaE	452
3	flgN	416
3	flgM	293
1	moaD	345

```
SELECT  COUNT (*)
FROM GENEOPERONLENGTH
WHERE gene_length < 1000;

COUNT *
    10
```

```
SELECT  COUNT (DISTINCT OPERON_ID)
FROM GENEOPERONLENGTH;

COUNT DISTINCT OPERON_ID
             3
```

SQL as DML

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	1322
2	deoB	1223
1	moaA	989
2	deoC	779
2	deoD	719
3	flgA	659
1	moaB	512
1	moaC	485
1	moaE	452
3	flgN	416
3	flgM	293
1	moaD	345

```
SELECT  AVG (gene_length)
FROM GENEOPERONLENGTH;

AVG gene_length
              682.83333
```

You can nest queries – the innermost set of parentheses is carried out first – this is how you can compare the values in the column to the average.

```
SELECT operon_id, gene_id, gene_length
FROM GENEOPERONLENGTH
WHERE gene_length>(SELECT AVG (gene_length) FROM GENEOPERONLENGTH);

operon_id    gene_id      gene_length
2        deoA     1322
2        deoB     1233
1        maoA     989
2        deoC     779
2        deoD     719
```

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	1322
2	deoB	1223
1	moaA	989
2	deoC	779
2	deoD	719
3	flgA	659
1	moaB	512
1	moaC	485
1	moaE	452
3	flgN	416
3	flgM	293
1	moaD	345

```
SELECT MAX (gene_length)
FROM GENEOPERONLENGTH;

MAX (gene_length)
         1322
```

What is another statement that would yield the same answer in the first row returned?

```
SELECT MIN (gene_length)
FROM GENEOPERONLENGTH;

MIN (gene_length)
         293
```

What is another statement that would yield the same answer in the first row returned?

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	1322
2	deoB	1223
1	moaA	989
2	deoC	779
2	deoD	719
3	flgA	659
1	moaB	512
1	moaC	485
1	moaE	452
3	flgN	416
3	flgM	NULL
1	moaD	345

```
SELECT SUM (gene_length)
FROM GENEOPERONLENGTH;

SUM (gene_length)
        7901.0
```

If there had been no NULL then all values would be integers and SUM would return an integer (7901).

```
SELECT TOTAL (gene_length)
FROM GENEOPERONLENGTH;

TOTAL (gene_length)
        7901.0
```

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	NULL
2	deoB	NULL
1	moaA	NULL
2	deoC	NULL
2	deoD	NULL
3	flgA	NULL
1	moaB	NULL
1	moaC	NULL
1	moaE	NULL
3	flgN	NULL
3	flgM	NULL
1	moaD	NULL

```
SELECT SUM (gene_length)
FROM GENEOPERONLENGTH;

SUM (gene_length)
         NULL
```

```
SELECT TOTAL (gene_length)
FROM GENEOPERONLENGTH;

TOTAL (gene_length)
         0.0
```

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	NULL
2	deoB	NULL
1	moaA	NULL
2	deoC	NULL
2	deoD	NULL
3	flgA	NULL
1	moaB	NULL
1	moaC	NULL
1	moaE	NULL
3	flgN	NULL
3	flgM	NULL
1	moaD	NULL

Functions on text have a separate kind of command – often you want to either find a sub-string, or make a super-string.

Concatenate command symbol looks like: ||

You can specify a style of separator using single quotes following the elements to concatenate (‘,’)where the second part of the pair is the symbol you want.

```
SELECT (operon_id||gene_id||',') AS
Operon_Genes
FROM GENEOPERONLENGTH
WHERE operon_id < 2;

Operon_Genes
     1moaA,1moaB,1moaC,1moaE,1moaD
```

GENEOPERONLENGTH

operon_id	gene_id	gene_length
2	deoA	NULL
2	deoB	NULL
1	moaA	NULL
2	deoC	NULL
2	deoD	NULL
3	flgA	NULL
1	moaB	NULL
1	moaC	NULL
1	moaE	NULL
3	flgN	NULL
3	flgM	NULL
1	moaD	NULL

You can use the SUBSTRING command, giving the input string, the position to start checking from.

```
SELECT SUBSTR('deoA',3)
→ 'oA'
```

```
SELECT SUBSTR('deoA',3,1)
→ 'o'
```

This can be pretty handy if you have stored a long string of annotation (like in Genbank files) and you are looking for records with a particular kind of information.

LIKE can be used but might get pretty complicated.

Some design considerations have to do with the *speed* of queries.

The DBMS automatically generates unique numbers if the PK is simply made an incremented value (e.g. read the SQLite Documentation). Then uniqueness does not have to be checked with each addition.

Similarly a date for the record can be automatically generated by the system, and stored to be similarly used.

A composite PK makes FKs more cumbersome as well as other search routines, and if text labels are used may become a rather long string to search.