BINF 8211/6211
Design and Implementation of Bioinformatics Databases
Lecture #11

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

SQL for Data manipulation

- The power of SQL is in the ability to manipulate organize and retrieve data from the database.
- SQL – “Query” language is used to ask questions of the system.

SELECT Syntax

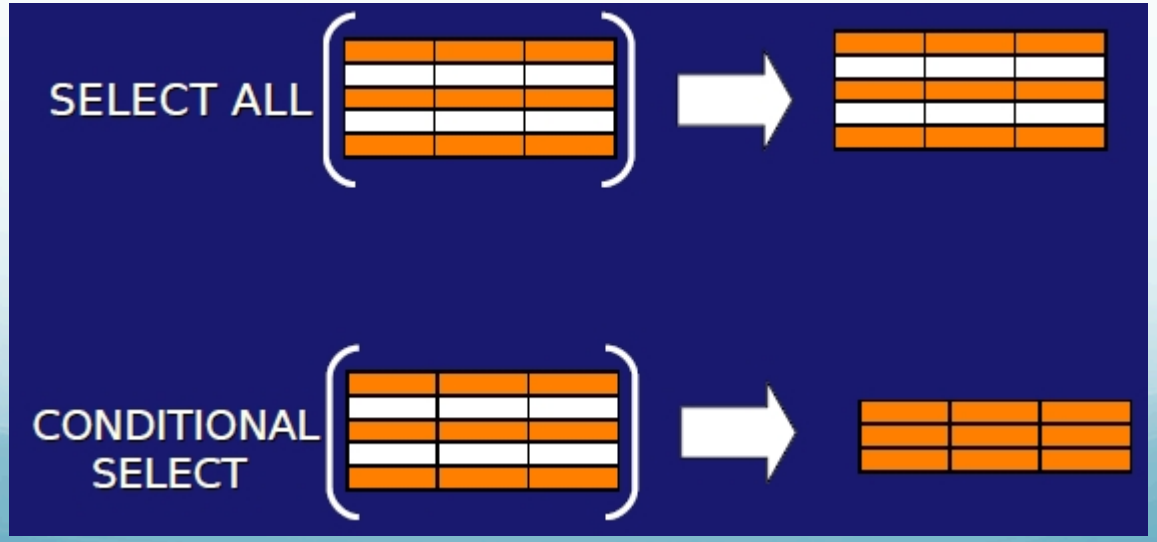
```
SELECT      <column list>
FROM        <table list>
[WHERE      <condition list>]
[ORDER BY   <column list> [ASC : DESC]]
```

- Syntax for SELECT < > FROM < > WHERE < >.
- Most common tool
  - Used to retrieve views of the data

SELECT WHERE

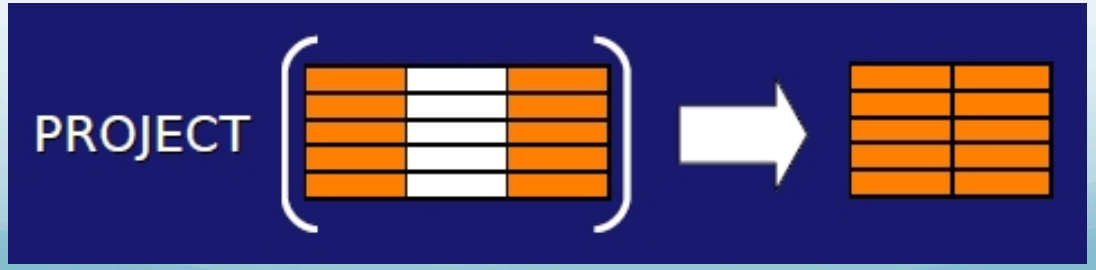
- SELECT in SQL *combines* the SELECT (RESTRICT) and PROJECT functions and can be used to join tables.
- The PROJECT function returns all of the rows in a table,
  - but RESTRICTS the new table to having only those attributes that are TRUE for the condition stated.
- The function that finds all of the rows in a table that are TRUE for the stated condition.
- The function RESTRICTS the rows that are returned given attribute values that must match your criteria – use the WHERE clause to express this.

```
SELECT <   > WHERE <   >
```



PROJECT review

- PROJECT RESTRICTS what is compared to find a TRUE condition based on columns
  - Π_x (E) where x is the condition (predicate) that must be evaluated.



Examples using the class database.

- Logical query 1: select all of the rows and columns from the *operon* table.
  - * is used for ALL, so SELECT * means SELECT ALL attributes
  - FROM gives the table name

- Logical query 2: Select only the <named = operon_id, operon_name> attributes from the *operon* table for all of the rows (the PROJECT function)

```
SELECT  *
FROM    operon
```

operon_id (PK)	operon_name	regulon_id (FK)
1	moaABCDE	2
2	deoCABD	1
3	flgAMN	1
4	yfcAML	2

```
SELECT  operon_id,
        operon_name
FROM    operon
```

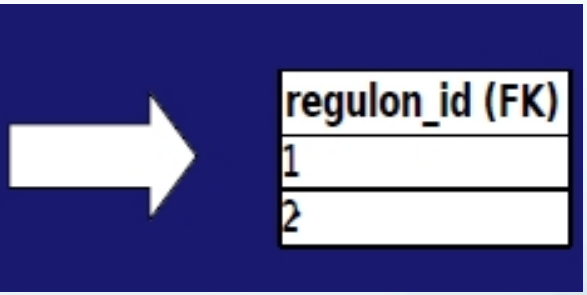
operon_id (PK)	operon_name
1	moaABCDE
2	deoCABD
3	flgAMN
4	yfcAML

The DISTINCT statement yields only the set of values that are different from each other in the specified columns.

- Logical query: list the regulons that have *at least one* operon in the *operon* table.

```
SELECT   DISTINCT regulon_id
FROM     operon
```

operon_id (PK)	operon_name	regulon_id (FK)
1	moaABCDE	2
2	deoCABD	1
3	flgAMN	1
4	yfcAML	2



regulon_id (FK)
1
2

What is the outcome if there is a NULL value in the regulon_id column?

Using the * (wildcard) in the DISTINCT specification.

operon_id (PK)	operon_name	regulon_id (FK)
1	moaABCDE	2
2	deoCABD	1
3	flgAMN	1
4	yfcAML	2

```
SELECT    DISTINCT *
FROM      operon
```

operon_id (PK)	operon_name	regulon_id (FK)
1	moaABCDE	2
2	deoCABD	1
3	flgAMN	1
4	yfcAML	2

```
SELECT    DISTINCT operon_name
FROM      operon
```

operon_name
moaABCDE
deoCABD
flgAMN
yfcAML

ORDER BY

- Allows ordering of the data
  - You can **order** on multiple attributes –
    - cascading order sequence from Left to Right.
  - Text is ordered alphabetically
  - numerical is ordered from negative to positive and increasing from zero.
- The ORDER BY statement works on *rows*, as shown.
- Your options are ASCENDING or DESCENDING.

```
SELECT      <column list>
FROM        <table list>
[WHERE      <condition list>]
[ORDER BY   <column list> [ASC : DESC]]
```

ORDER BY Example 1

- Example using the wild card output
- Logical query 1: organize the operon table to sort the operon name alphabetically.

operon_id (PK)	operon_name	regulon_id (FK)
1	moaABCDE	2
2	deoCABD	1
3	flgAMN	1
4	yfcAML	2

operon_id (PK)	operon_name	regulon_id (FK)
2	deoCABD	1
3	flgAMN	1
1	moaABCDE	2
4	yfcAML	2

```
SELECT *
FROM      operon
ORDER BY operon_name ASC
```

operon_id (PK)	operon_name	regulon_id (FK)
4	yfcAML	2
1	moaABCDE	2
3	flgAMN	1
2	deoCABD	1

```
SELECT *
FROM      operon
ORDER BY operon_name DESC
```

ORDER BY

- Multiple Attributes
  - Left to right order

- Example: re-organize the *gene* table, first by descending operon_id values, then by alphabetical gene_symbol values.

```
SELECT     operon_id, gene_symbol
FROM       gene
ORDER BY   operon_id DESC, gene_symbol ASC
```

gene_symbol (PK)	genome_start_loc	genome_end_loc	genome_strand	gene seq	operon_id (FK)
moaA	816267	817256	+	ATGGCTTCAC	1
moaB	817278	817790	+	ATGAGTCAGG	1
moaC	817793	818278	+	ATGTCGCAAC	1
moaD	818271	818516	+	ATGATTAAAG	1
moaE	818518	818970	+	ATGGCAGAAA	1
flgN	1128637	1129053	-	ATGACACGTC	3
flgM	1129058	1129351	-	ATGAGTATTG	3
flgA	1129427	1130086	-	ATGCTGATAA	3
deoC	4615346	4616125	+	ATGACTGATC	2
deoA	4616252	4617574	+	TTGTTTCTCG	2
deoB	4617626	4618849	+	ATGAAACGTG	2
deoD	4618906	4619625	+	ATGGCTACCC	2

operon_id (FK)	gene_symbol (PK)
3	flgA
3	flgM
3	flgN
2	deoA
2	deoB
2	deoC
2	deoD
1	moaA
1	moaB
1	moaC
1	moaD
1	moaE

LIKE

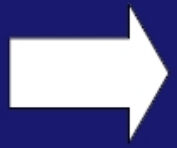
- Similar to regex
- special operator is used in combination with wildcards to find *patterns in character* attribute values.
- Standard SQL wildcard characters
  - % (percent sign): all preceding (%T) or following (T%) characters are allowed
  - _ (underscore): any single character may be substituted for the underscore position.
- Examples
  - '%e' → Irene, race, L123e
  - 'Jo%' → Johnson and Jones
  - '%n%' → Johnson, Jones and Jernigan
  - '_o_es' includes Jones, Cones, Cokes, totes and roles (etc.)

LIKE Example 1

Which of the gene names include the string 'flg' as part of the name?

```
SELECT    gene_symbol, gene_seq
FROM      gene
WHERE     gene_symbol LIKE 'flg%'
ORDER BY  gene_symbol ASC
```

gene_symbol (PK)	genome_start_loc	genome_end_loc	genome_strand	gene_seq	operon_id (FK)
moaA	816267	817256	+	ATGGCTTCAC	1
moaB	817278	817790	+	ATGAGTCAGG	1
moaC	817793	818278	+	ATGTCGCAAC	1
moaD	818271	818516	+	ATGATTAAAG	1
moaE	818518	818970	+	ATGGCAGAAA	1
flgN	1128637	1129053	-	ATGACACGTC	3
flgM	1129058	1129351	-	ATGAGTATTG	3
flgA	1129427	1130086	-	ATGCTGATAA	3
deoC	4615346	4616125	+	ATGACTGATC	2
deoA	4616252	4617574	+	TTGTTTCTCG	2
deoB	4617626	4618849	+	ATGAAACGTG	2
deoD	4618906	4619625	+	ATGGCTACCC	2



gene_symbol (PK)	gene_seq
flgA	ATGCTGATAA
flgM	ATGAGTATTG
flgN	ATGACACGTC

LIKE Example 2

Which of the gene sequences include the motif ‘GNCT’?

```
SELECT     gene_symbol, gene_seq
FROM       gene
WHERE      gene_seq LIKE '%G_CT%'
ORDER BY   gene_symbol ASC
```

gene_symbol (PK)	genome_start_loc	genome_end_loc	genome_strand	gene_seq	operon_id (FK)
moaA	816267	817256	+	ATGGCTTCAC	1
moaB	817278	817790	+	ATGAGTCAGG	1
moaC	817793	818278	+	ATGTCGCAAC	1
moaD	818271	818516	+	ATGATTAAAG	1
moaE	818518	818970	+	ATGGCAGAAA	1
flgN	1128637	1129053	-	ATGACACGTC	3
flgM	1129058	1129351	-	ATGAGTATTG	3
flgA	1129427	1130086	-	ATGCTGATAA	3
deoC	4615346	4616125	+	ATGACTGATC	2
deoA	4616252	4617574	+	TTGTTTCTCG	2
deoB	4617626	4618849	+	ATGAAACGTG	2
deoD	4618906	4619625	+	ATGGCTACCC	2

gene_symbol (PK)	gene_seq
deoC	ATGACTGATC
deoD	ATGGCTACCC
moaA	ATGGCTTCAC

IN

- IN: Uses a set of entity values to restrict the return data set

All genes whose names include ‘moaB’, ‘flgM’ and ‘deoC’.

```
SELECT     gene_symbol, operon_id
FROM       gene
WHERE      gene_symbol IN ('moaB', 'flgM', 'deoC')
ORDER BY   gene_symbol ASC
```

gene_symbol (PK)	genome_start_loc	genome_end_loc	genome_strand	gene_seq	operon_id (FK)
moaA	816267	817256	+	ATGGCTTCAC	1
moaB	817278	817790	+	ATGAGTCAGG	1
moaC	817793	818278	+	ATGTCGCAAC	1
moaD	818271	818516	+	ATGATTAAAG	1
moaE	818518	818970	+	ATGGCAGAAA	1
flgN	1128637	1129053	-	ATGACACGTC	3
flgM	1129058	1129351	-	ATGAGTATTG	3
flgA	1129427	1130086	-	ATGCTGATAA	3
deoC	4615346	4616125	+	ATGACTGATC	2
deoA	4616252	4617574	+	TTGTTTCTCG	2
deoB	4617626	4618849	+	ATGAAACGTG	2
deoD	4618906	4619625	+	ATGGCTACCC	2

gene_symbol (PK)	operon_id (FK)
deoC	2
flgM	3
moaB	1

SQL Aggregate Functions - I

- Avg(X)
  - returns the average of all non-NULL (X) in a group
  - String and BLOB values are interpreted as zero (different from NULL – why? - denominator).
  - Always returns a floating point number
    - To account for fractional averages.
- Count (X)
  - returns the number of times that X is NOT NULL in a group of values in a column.
  - COUNT(*) returns the number of rows in a query set.

Note in the following: any aggregate function that takes a single argument can be preceded by DISTINCT, which *filters out duplicates* before passing to the function

So COUNT (DISTINCT X) will return the number of distinct values in column X instead of the total number of non-null values in column X.

Aggregate Functions - Examples

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
```

```
SELECT  AVG (gene_length)
FROM GENEOPERONLENGTH;

AVG gene_length
              682.83333
```

Nested Query

- The inner query is executed first.
- The outer query is executed last.
  - The outer query is always the first SQL command –
  - here the SELECT statement.
- Example:
  - The inner query finds the average gene length
    - stored in memory.
    - available for comparing gene lengths.

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 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
```