SQL Summary

- Mathematical functions, for single values and groups of values.
- Storing dates and times in the database.
- Creating views with “CREATE VIEW ... AS”.
- Creating tables from tables with “CREATE TABLE ... AS”.

This session — Indexes, Sequences, Relational Algebra and Advanced Joins

Indexes

Indexes are redundant data structures that improve the performance of a database. An index makes it faster for a database to look up a row based on a value within that row.

An index has a storage overhead. Updates on a table that has an index can take than one that hasn’t. It is best to only put indexes on columns that are often used for a search.

They are not required for correctness, they do not affect the result returned by a query.

An index is automatically added to any column declared as a Primary Key.

Database implementations have their own specific indexing schemes and in some you can specify exactly what kind of index to use.

Creating an index

Creating indexes is not part of the SQL standard but nearly all databases support a common syntax.

An index can be created on a single column:

CREATE INDEX year_index ON CD.Year (year);

This will improve performance on queries that look up 'year'. e.g.:

SELECT barcode FROM CD.Year WHERE year = 1994;
Creating a multiple-key index

An index can also be created on a multiple columns:
```
CREATE INDEX index ON Pop_albums (artist, album);
```
This will improve performance on queries that look up both artist and album. e.g.:
```
SELECT barcode FROM Pop_albums WHERE artist='U2' AND album='Rattle and Hum';
```

Sequences

Some databases support an 'Identity' data type that is automatically populated with a unique reference number (URN).

This kind of column is useful when creating surrogate keys (a key that is not derived from any data in the database and whose only purpose is to act as a primary key).

For example MS Access and MS SQL Server both have an Identity data type and a GUID (Globally Unique IDenti¯er) data type.
MySQL has an AUTO_INCREMENT data type.
Oracle does not have a data type like this, it has Sequences.

Creating a Sequence

To create a sequence that starts at 1 and increments by 1 use:
```
CREATE SEQUENCE seq1;
```
To create a sequence that starts at 10 and increments by 2 use:
```
CREATE SEQUENCE seq2 INCREMENT BY 2 START WITH 10;
```
To delete a sequence use:
```
DROP SEQUENCE seq2;
```

Using a Sequence

CREATE TABLE test ( urn NUMBER, name VARCHAR(10) );
To insert a row where one of the columns should be an URN:
```
INSERT INTO test VALUES (seq1.nextval, 'Tim');
```
Relational Algebra

SQL manipulates tables. Relational operators are closed over these tables.

The result of a relational operator on one or more tables is another table.

There are five primitive operators in relational algebra — union, difference, product, projection, selection.

NB The SQL keyword SELECT is associated with projection, not selection!

Also have derived operators (cf operators in arithmetic, such as square(x) \( \equiv x \times x \)). Examples include intersection and join.

Primitive Relational Operators

The 5 primitive relational operators:

- **union** \( A \cup B \) Combines all tuples from \( A \) and \( B \).
- **difference** \( A \setminus B \) All tuples from \( A \) with those common to \( B \) (\( A \cap B \)) removed.
- **product** \( A \times B \) Cartesian product of \( A \) and \( B \).
- **project** \( A[a,b,\ldots] \) Select only attributes \( a, b, \ldots \) from relation \( A \).
- **select** \( A : C \) Select only those tuples satisfying the specified boolean condition \( C \), where \( C \) is constructed from arithmetic comparisons involving attributes by using propositional connectives.

Derived Relational Operators

- **intersection** \( A \cap B \) Common tuples to \( A \) and \( B \).
- **join** \( A(A.a = B.b) \) JOIN \( B \) Join tables \( A \) and \( B \) together — for each row match attribute \( A.a \) with \( B.b \), discarding duplicate columns.

Another example of a derived operator is divideby — this attempts to invert product, in so far as this is possible.

Codd’s completeness criterion:

A query language is complete if it can express each of the five primitive relational operators.

Union

**Union** \( A \cup B \)

Use SQL keyword UNION. Tables must be compatible ... have the same attributes (column headings).

\[
\text{(SELECT artist FROM Pop_albums}
\text{ WHERE artist LIKE 'U%'})
\text{ UNION}
\text{(SELECT artist FROM Band_members}
\text{ WHERE mbr1 = 'Grohl'});
\]

Result is a one column table containing three entries: Foo Fighters, U2 and Underworld.
**Intersection**

Intersection $A \cap B$

Use SQL keyword `INTERSECT`. Tables must be compatible. Query selects U2 and Foo Fighters.

```
(SELECT artist FROM Pop_albums)
INTERSECT
(SELECT artist FROM Band_members);
```

**Difference**

Difference

Use SQL keyword `MINUS`.

```
(SELECT artist FROM Pop_albums)
MINUS
(SELECT artist FROM Band_members);
```

Selects everything but U2 and Foo Fighters — The Verve and Underworld.

**Product**

Product $A \times B$

Part of the `SELECT` statement — list more than one table after keyword `FROM`.

```
SELECT f1, f2 FROM Forward, Reverse;
```

<table>
<thead>
<tr>
<th>Forward</th>
<th>Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>f2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

```
F \times R

<table>
<thead>
<tr>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Project**

Project $A[a, b...]

Using keyword `SELECT` as from day 1.

```
SELECT a, b FROM A;
```

```
SELECT artist, album FROM Pop_albums;
```
Join

Join $A.a = B.b$ JOIN $B$

Using SELECT as in last seminar:

```
SELECT artist, album, year
FROM Pop_albums, CD_year
WHERE Pop_albums.barcode = CD_year.barcode;
```

```
SELECT a, b FROM A, B WHERE A.a = B.b;
```

Notice how join is a combination of the product of tables and a predicate selection.

Advanced Joins

We covered a simple (equi-)join between two tables earlier in the course.

Joins combine rows from two or more tables to create a single result. Columns are compared with a Join Condition.

Pairs of rows each containing one row from each table are combined when the join condition evaluates to TRUE.

Join Types

Joins can be classified into the following categories:

- Cartesian Products
- Inner Joins (Equijoins)
- Self Joins
- Outer Joins (Left, Right and Full)

Cartesian Products (Cross Join)

If a join is performed without a Join Condition then the cartesian product of the specified tables is returned.

Cartesian products can generate many rows of meaningless data, for this reason they are often used for creating test data. Apart from this they have few practical uses.

Cartesian products form the base of all the other types of join.
**Inner Joins (Equijoins)**

An Inner Join (or Equijoin) is a join with a condition that compares columns for equality (=). Rows are combined that have equal values in the specified columns.

The order of the tables listed in the `FROM` clause has no significance.

**Inner Join Example**

The example from before:

```sql
SELECT artist, album, year
FROM Pop_albums, CD_year
WHERE Pop_albums.barcode = CD_year.barcode;
```

<table>
<thead>
<tr>
<th>ARTIST</th>
<th>ALBUM</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>The Unforgettable Fire</td>
<td>1984</td>
</tr>
<tr>
<td>U2</td>
<td>Rattle and Hum</td>
<td>1990</td>
</tr>
<tr>
<td>U2</td>
<td>Achtung Baby</td>
<td>1991</td>
</tr>
<tr>
<td>Underworld</td>
<td>Second Toughest in the Infants</td>
<td>1996</td>
</tr>
<tr>
<td>The Verve</td>
<td>Urban Hymns</td>
<td>1997</td>
</tr>
<tr>
<td>Foo Fighters</td>
<td>The Colour and the Shape</td>
<td>1997</td>
</tr>
</tbody>
</table>

**Self Joins**

A Self Join is a join of a table to itself. Put the table in the `FROM` clause twice. Self joins are very useful. Use aliases to distinguish columns in the `WHERE` clause.

**Self Join Example**

Determine artists that have released more than one album:

```sql
SELECT DISTINCT a.artist
FROM Pop_albums a, Pop_albums b
WHERE a.artist = b.artist
AND a.album <> b.album;
```

Can you think of alternative way of expressing this in SQL?
Hierachical Data

Self-joins can be useful when working with hierachical data.
Consider the following table (EMPLOYEES):

<table>
<thead>
<tr>
<th>EMPLOYEEID</th>
<th>SUPERVISORID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>NULL</td>
<td>John Balfour</td>
</tr>
<tr>
<td>101</td>
<td>100</td>
<td>Susan Saronnen</td>
</tr>
<tr>
<td>102</td>
<td>100</td>
<td>Eric La Sold</td>
</tr>
<tr>
<td>103</td>
<td>100</td>
<td>Martin Murphy</td>
</tr>
<tr>
<td>104</td>
<td>103</td>
<td>Erica Strange</td>
</tr>
<tr>
<td>105</td>
<td>103</td>
<td>Noah Tamil</td>
</tr>
</tbody>
</table>

How do we write a query to find the name of each employee's supervisor? We can use a self-join:

```
SELECT staff.name, supervisor.name FROM EMPLOYEES staff, EMPLOYEES supervisor
WHERE staff.supervisorid = supervisor.employeeid;
```

<table>
<thead>
<tr>
<th>STAFF.NAME</th>
<th>SUPERVISOR.NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Susan Saronnen</td>
<td>John Balfour</td>
</tr>
<tr>
<td>Eric La Sold</td>
<td>John Balfour</td>
</tr>
<tr>
<td>Martin Murphy</td>
<td>John Balfour</td>
</tr>
<tr>
<td>Erica Strange</td>
<td>Martin Murphy</td>
</tr>
<tr>
<td>Noah Tamil</td>
<td>Martin Murphy</td>
</tr>
</tbody>
</table>

Outer Joins

An Inner Join excludes rows from either table that don’t have a matching row in the other table.

An Outer Join allows us to return unmatched rows.

Outer Joins come in three varieties:
- LEFT - only unmatched rows from the left table are kept
- RIGHT - only unmatched rows from the right table are kept
- FULL - unmatched rows from both tables are retained

Example data

Imagine we have two tables defined as:

<table>
<thead>
<tr>
<th>CD</th>
<th>company</th>
</tr>
</thead>
<tbody>
<tr>
<td>barcode</td>
<td>company</td>
</tr>
<tr>
<td>042282289827</td>
<td>Island</td>
</tr>
<tr>
<td>042284229920</td>
<td>Island</td>
</tr>
<tr>
<td>731451034725</td>
<td>Island</td>
</tr>
<tr>
<td>026734000524</td>
<td>Junior</td>
</tr>
<tr>
<td>724384491321</td>
<td>Virgin</td>
</tr>
<tr>
<td>724385583223</td>
<td>Capital</td>
</tr>
<tr>
<td>724383719020</td>
<td>EMI</td>
</tr>
<tr>
<td>891030505032</td>
<td>Naxos</td>
</tr>
</tbody>
</table>
### Outer Left Join Example

Select company, artist, album from CD_Company left join Pop_albums on Pop_albums.barcode = CD_Company.barcode;

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>ARTIST</th>
<th>ALBUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital</td>
<td>Foo Fighters</td>
<td>The Colour and the Shape</td>
</tr>
<tr>
<td>EMI</td>
<td>U2</td>
<td>The Unforgettable Fire</td>
</tr>
<tr>
<td>Island</td>
<td>U2</td>
<td>Rattle and Hum</td>
</tr>
<tr>
<td>Island</td>
<td>U2</td>
<td>Achtung Baby</td>
</tr>
<tr>
<td>Junior</td>
<td>Underworld</td>
<td>Achtung Baby</td>
</tr>
<tr>
<td>Naxos</td>
<td>The Verve</td>
<td>Urban Hymns</td>
</tr>
<tr>
<td>Virgin</td>
<td>The Verve</td>
<td>Urban Hymns</td>
</tr>
</tbody>
</table>

Notice not all companies match up to an album and not all albums match to a company.

### Outer Right Join Example

Select year, artist from Pop_albums right join CD_year on Pop_albums.barcode = CD_year.barcode order by year;

<table>
<thead>
<tr>
<th>YEAR</th>
<th>ARTIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>U2</td>
</tr>
<tr>
<td>1988</td>
<td>U2</td>
</tr>
<tr>
<td>1991</td>
<td>U2</td>
</tr>
<tr>
<td>1992</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>Underworld</td>
</tr>
<tr>
<td>1997</td>
<td>Foo Fighters</td>
</tr>
<tr>
<td>1997</td>
<td>The Verve</td>
</tr>
</tbody>
</table>

Produces a summary of record labels and the artists they publish. Notice that the record labels EMI and Naxos are displayed even though there are no albums with these companies in the Pop_albums table.
Outer Full Join Example

SELECT company, artist, album FROM CD_Company FULL JOIN Pop_albums ON Pop_albums.barcode = CD_Company.barcode;

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>ARTIST</th>
<th>ALBUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Island</td>
<td>U2</td>
<td>The Unforgettable Fire</td>
</tr>
<tr>
<td>Island</td>
<td>U2</td>
<td>Rattle and Hum</td>
</tr>
<tr>
<td>Island</td>
<td>U2</td>
<td>Achtung Baby</td>
</tr>
<tr>
<td>Junior</td>
<td>Underworld</td>
<td>Second Toughest in the Infants</td>
</tr>
<tr>
<td>Virgin</td>
<td>The Verve</td>
<td>Urban Hymns</td>
</tr>
<tr>
<td>Capital</td>
<td>Foo Fighters</td>
<td>The Colour and the Shape</td>
</tr>
<tr>
<td>Naxos</td>
<td>EMI</td>
<td></td>
</tr>
<tr>
<td>EMI</td>
<td>Leftfield</td>
<td>Leftism</td>
</tr>
</tbody>
</table>